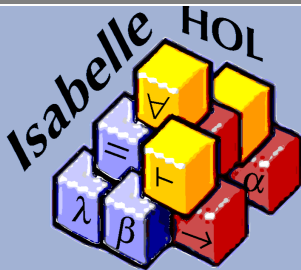


Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

LEHRSTUHL PROGRAMMIERPARADIGMEN



Teil IV

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

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

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

Die üblichen zwei Quantoren der Logik:

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

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

Gültigkeitsbereich der gebundenen Variablen:

bis zum nächsten `;` bzw. \implies

Die üblichen zwei Quantoren der Logik:

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

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

Gültigkeitsbereich der gebundenen Variablen:

bis zum nächsten `;` bzw. \implies

Beispiele

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

Die üblichen zwei Quantoren der Logik:

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

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

Gültigkeitsbereich der gebundenen Variablen:

bis zum nächsten `;` bzw. \implies

Beispiele

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

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

Die üblichen zwei Quantoren der Logik:

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

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

Gültigkeitsbereich der gebundenen Variablen:

bis zum nächsten `;` bzw. \implies

Beispiele

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

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

$\forall x. P x \implies \exists x. Q x \implies R$

Zwei verschiedene x in den Annahmen

gleichbedeutend mit $\forall y. P y \implies \exists z. Q z \implies R$

(gebundene Namen sind Schall und Rauch)

Die üblichen zwei Quantoren der Logik:

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

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

Gültigkeitsbereich der gebundenen Variablen:

bis zum nächsten `;` bzw. \implies

Beispiele

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

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

$\forall x. P x \implies \exists x. Q x \implies R$

Zwei verschiedene x in den Annahmen

gleichbedeutend mit $\forall y. P y \implies \exists z. Q z \implies R$

(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: $\wedge x. \dots \implies \dots$

\wedge heisst **Meta-Allquantor**, Variablen dahinter **Parameter**

Gültigkeitsbereich der Parameter: ganzes Teilziel

Beispiel: $\wedge x y. \forall y. P y \longrightarrow Q z y \implies Q x y \implies \exists x. Q x y$

entspricht $\wedge x y. \forall y_1. P y_1 \longrightarrow Q z y_1 \implies Q x y \implies \exists x_1. Q x_1 y$

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: $\wedge x. \dots \implies \dots$

\wedge heisst **Meta-Allquantor**, Variablen dahinter **Parameter**

Gültigkeitsbereich der Parameter: ganzes Teilziel

Beispiel: $\wedge x y. \forall y. P y \longrightarrow Q z y \implies Q x y \implies \exists x. Q x y$

entspricht $\wedge x y. \forall y_1. P y_1 \longrightarrow Q z y_1 \implies Q x y \implies \exists x_1. Q x_1 y$

Auch die **Meta-Implikation** \implies ist Teil der Meta-Logik

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: $\bigwedge x. \dots \implies \dots$

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

Gültigkeitsbereich der Parameter: ganzes Teilziel

Beispiel: $\bigwedge x y. \forall y. P y \longrightarrow Q z y \implies Q x y \implies \exists x. Q x y$

entspricht $\bigwedge x y. \forall y_1. P y_1 \longrightarrow Q z y_1 \implies Q x y \implies \exists x_1. Q x_1 y$

Auch die **Meta-Implikation** \implies ist Teil der Meta-Logik

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

Jeder Quantor hat Introduktions- und Eliminationsregel:

Jeder Quantor hat Introduktions- und Eliminationsregel:

■ $allI: (\wedge x. P x) \implies \forall x. P x$

Eine Aussage gilt für beliebige x (Meta-Ebene),
also gilt sie auch für alle (HOL-Ebene)

Jeder Quantor hat Introduktions- und Eliminationsregel:

■ $allI: (\wedge x. P x) \implies \forall x. P x$

Eine Aussage gilt für beliebige x (Meta-Ebene),
also gilt sie auch für alle (HOL-Ebene)

■ $allE: \forall x. P x \implies (P ?x \implies R) \implies R$

Eine Aussage gilt für alle x , also folgt die Konklusion, wenn ich sie
unter Verwendung der Aussage für einen (selbst wählbaren) Term
zeigen kann

Jeder Quantor hat Introduktions- und Eliminationsregel:

- $allI: (\wedge x. P x) \implies \forall x. P x$
Eine Aussage gilt für beliebige x (Meta-Ebene), also gilt sie auch für alle (HOL-Ebene)
- $allE: \forall x. P x \implies (P ?x \implies R) \implies R$
Eine Aussage gilt für alle x , also folgt die Konklusion, wenn ich sie unter Verwendung der Aussage für einen (selbst wählbaren) Term zeigen kann
- $exI: P ?x \implies \exists x. P x$
Eine Aussage gilt für einen Term $?x$, also gibt es ein x , wofür sie gilt

Jeder Quantor hat Introduktions- und Eliminationsregel:

■ $allI: (\wedge x. P x) \implies \forall x. P x$

Eine Aussage gilt für beliebige x (Meta-Ebene), also gilt sie auch für alle (HOL-Ebene)

■ $allE: \forall x. P x \implies (P ?x \implies R) \implies R$

Eine Aussage gilt für alle x , also folgt die Konklusion, wenn ich sie unter Verwendung der Aussage für einen (selbst wählbaren) Term zeigen kann

■ $exI: P ?x \implies \exists x. P x$

Eine Aussage gilt für einen Term $?x$, also gibt es ein x , wofür sie gilt

■ $exE: \exists x. P x \implies (\wedge x. P x \implies Q) \implies Q$

Eine Aussage gilt für ein x , also folgt die Konklusion, wenn ich sie unter Verwendung der Aussage für einen beliebigen, nicht weiter bestimmten Term zeigen kann

Der Befehl **fix** korrespondiert mit \wedge .

```
have "∀ x. P x"  
proof(rule allI)  
  fix x  
  show "P x" ⟨Beweis⟩  
qed
```

```
from ⟨∀ x. P x⟩  
have "Q (f x)"  
proof(rule allE)  
  fix x  
  assume "P (f x)"  
  then show "Q (f x)" ⟨Beweis⟩  
qed
```

Letzteres geht auch ohne neuen Scope:

```
from ⟨∃ x. P x⟩  
obtain x where "P x" by (rule exE)
```

```
have "∃ x. P x"  
proof(rule exI)  
  show "P (f 0)" ⟨Beweis⟩  
qed
```

```
from ⟨∃ x. P x⟩  
have "R"  
proof(rule exE)  
  fix x  
  assume "P x"  
  show "R" ⟨Beweis⟩  
qed
```

Teil V

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

$$\frac{\frac{P}{Q} \quad \frac{\neg P}{Q}}{Q}$$

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

$$\frac{\frac{P}{Q} \quad \frac{\neg P}{Q}}{Q}$$

In Isabelle: Mit der Regel `case_split`: $(P \implies Q) \implies (\neg P \implies Q) \implies Q$

Beispiel:

```
have "BB  $\vee$   $\neg$  BB"
```

```
proof(rule case_split)
```

```
  assume "BB"
```

```
  then show "BB  $\vee$   $\neg$ BB" ..
```

```
next
```

```
  assume " $\neg$  BB"
```

```
  then show "BB  $\vee$   $\neg$ BB" ..
```

```
qed
```

Statt **proof** (`rule case_split`)
geht auch **proof** (`cases "BB"`).
Vorteil: Die Annahmen sind
gleich die richtigen (sonst erfährt
Isabelle erst beim ersten **show**
was P sein sollte – das klappt
ggf. nicht zuverlässig).

Fallunterscheidung: Beispiel

Dieses Lemma wäre ohne Fallunterscheidung so **nicht** (einfach) **lösbar!**

lemma " $B \wedge C \longrightarrow (A \wedge B) \vee (\neg A \wedge C)$ "

proof

assume " $B \wedge C$ " **then have** " B "..

from $\langle B \wedge C \rangle$ **have** " C "..

show " $(A \wedge B) \vee (\neg A \wedge C)$ "

proof (*cases A*)

assume A

from $\langle A \rangle \langle B \rangle$ **have** " $A \wedge B$ "..

then show ?thesis..

next

assume " $\neg A$ "

from $\langle \neg A \rangle \langle C \rangle$ **have** " $\neg A \wedge C$ "..

then show ?thesis..

qed

qed

Teil VI

Definition

definition

Ermöglicht, einen Term zu benennen, so darüber zu abstrahieren und die Abstraktion gezielt zu öffnen

definition

Ermöglicht, einen Term zu benennen, so darüber zu abstrahieren und die Abstraktion gezielt zu öffnen

Beispiel:

```
definition solution :: "nat"  
where "solution = 42"
```

definition

Ermöglicht, einen Term zu benennen, so darüber zu abstrahieren und die Abstraktion gezielt zu öffnen

Beispiel:

```
definition solution :: "nat"  
where "solution = 42"
```

Erzeugt Regel: *solution_def*: *solution* = 42

definition

Ermöglicht, einen Term zu benennen, so darüber zu abstrahieren und die Abstraktion gezielt zu öffnen

Beispiel:

```
definition solution :: "nat"  
where "solution = 42"
```

Erzeugt Regel: *solution_def*: *solution* = 42

So können auch Funktionen definiert werden:

Beispiel:

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

definition

Ermöglicht, einen Term zu benennen, so darüber zu abstrahieren und die Abstraktion gezielt zu öffnen

Beispiel:

```
definition solution :: "nat"  
where "solution = 42"
```

Erzeugt Regel: *solution_def*: *solution* = 42

So können auch Funktionen definiert werden:

Beispiel:

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

Erzeugt Regel: *nand_def*: *nand* A B = (\neg (A \wedge B))

definition – Syntaxdefinition

`nand` ist binärer Operator

definition – Syntaxdefinition

`nand` ist binärer Operator

⇒ Infixoperator bietet sich an

definition – Syntaxdefinition

`nand` ist binärer Operator
⇒ 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
 \implies 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** " \boxtimes " 36)

where "*A* \boxtimes *B* = (\neg (*A* \wedge *B*))"

Jetzt: *A* \boxtimes *B* \boxtimes *C* gleichbedeutend mit `nand (nand A B) C`

Teil VII

Gleichungen

Das Arbeiten mit Gleichungen ist eine sehr wichtige Beweistechnik. Die Hauptregel dabei ist die Substitution:

$$\frac{s = t \quad P[s/x]}{P[t/x]}$$

Diese Regel gibt es auch in Isabelle:

subst: $s = t \implies P s \implies P t$

Das Arbeiten mit Gleichungen ist eine sehr wichtige Beweistechnik. Die Hauptregel dabei ist die Substitution:

$$\frac{s = t \quad P[s/x]}{P[t/x]}$$

Diese Regel gibt es auch in Isabelle:

```
subst: s = t  $\implies$  P s  $\implies$  P t
```

Beispiel:

```
assume "correct solution"  
with solution_def  
have "correct 42" by (rule subst)
```

Auch nützlich:

```
ssubst: s = t  $\implies$  P t  $\implies$  P s
```

```
arg_cong: x = y  $\implies$  f x = f y
```

Gleichheit ist transitiv, auch in Isabelle:

trans: $r = s \implies s = t \implies r = t$

Aber umständlich:

lemma "foo = qux"

proof(rule trans)

show "foo = bar" <Beweis 1>

next

show "bar = qux"

proof(rule trans)

show "bar = baz" <Beweis 2>

next

show "baz = qux" <Beweis 3>

qed

qed

Gleichheit ist transitiv, auch in Isabelle:

trans: $r = s \implies s = t \implies r = t$

Aber umständlich:

```
lemma "foo = qux"  
proof(rule trans)  
  show "foo = bar" <Beweis 1>  
next  
  show "bar = qux"  
  proof(rule trans)  
    show "bar = baz" <Beweis 2>  
  next  
    show "baz = qux" <Beweis 3>  
  qed  
qed
```

Besser mit **also** und **finally**:

```
lemma "foo = qux"  
proof-  
  have "foo = bar" <Beweis 1>  
  also  
  have "bar = baz" <Beweis 2>  
  also  
  have "baz = qux" <Beweis 3>  
  finally show ?thesis.  
qed
```

Aber oft **proof** (*rule trans*) zu schreiben wäre sehr umständlich.
Statt dessen: Gleichungsketten!

- Die Befehle **also** und **finally** sollten jeweils einer Aussage (**assume** oder **have**) folgen, die eine Gleichung ist.
- Das abschließende **finally** kombiniert die Aussagen per Transitivität und stellt das Ergebnis (wie **from**) bereit.
- In Ausdrücken steht ... für die rechte Seite der letzten Aussage.
- Ist ein Lemma *foo* falsch herum, kann man *foo[symmetric]* verwenden.
- Die Abkürzung *?thesis* steht für die Konklusion des aktuell zu beweisende Lemmas (vor Anwendung von Regeln!).

Typisches Muster:

proof-

`{...}`

finally show *?thesis*.

qed

Gleichungsketten (Beispiel)

lemma " $(A \wedge (A \vee B)) = A$ "

proof-

from *conj_disj_distribL*

have " $(A \wedge (A \vee B)) = ((A \wedge A) \vee (A \wedge B))$ ".

also

from *conj_absorb*

have " $((A \wedge A) \vee (A \wedge B)) = (A \vee (A \wedge B))$ " **by** (*rule arg_cong*)

also

have " $(A \vee (A \wedge B)) = A$ "

proof

<a nested proof>

qed

finally

show " $(A \wedge (A \vee B)) = A$ ".

qed

Gleichungsketten (Beispiel mit Abkürzungen)

lemma " $(A \wedge (A \vee B)) = A$ "

proof-

from *conj_disj_distribL*

have " $(A \wedge (A \vee B)) = ((A \wedge A) \vee (A \wedge B))$ ".

also

from *conj_absorb*

have " $\dots = (A \vee (A \wedge B))$ " **by** (*rule arg_cong*)

also

have " $\dots = A$ "

proof

<a nested proof>

qed

finally

show *?thesis.*

qed

Randbemerkung: moreover und ultimately

Die **also..finally**-Struktur hat einen kleinen Bruder:
moreover..ultimately. Hier werden die Aussagen nicht per Transitivität verbunden, sondern einfach gesammelt.

Randbemerkung: moreover und ultimately

Die **also..finally**-Struktur hat einen kleinen Bruder: **moreover..ultimately**. Hier werden die Aussagen nicht per Transitivität verbunden, sondern einfach gesammelt.

Damit können Verschachtelungen vermieden werden, die Beweise natürlicher aufgebaut und Wiederholungen in **from**-Befehlen verringert werden:

```
have "A ∧ B"  
proof  
  show A  
    ⟨Beweis A⟩  
next  
  show B  
    ⟨Beweis B⟩  
qed
```

```
have A  
  ⟨Beweis A⟩  
moreover  
  have B  
    ⟨Beweis B⟩  
ultimately  
  have "A ∧ B"..
```