Karlsruher Institut für Technologie

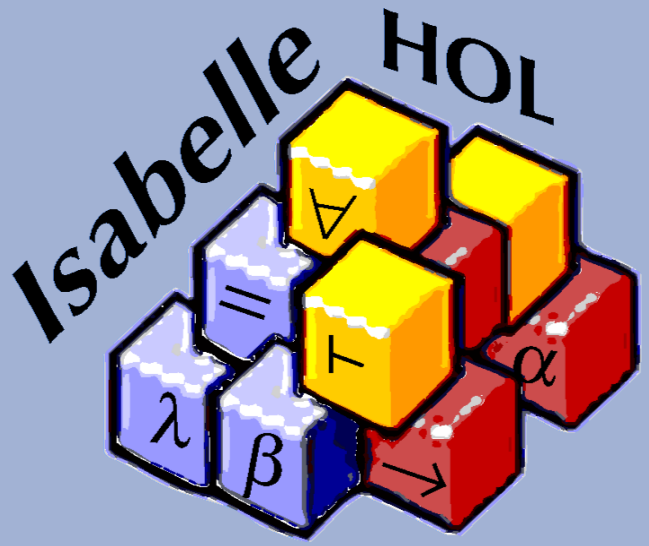
Theorembeweiserpraktikum

Anwendungen in der Sprachtechnologie

http://pp.info.uni-karlsruhe.de/lehre/SS2010/tba/

Daniel Wasserrab

IPD Snelting, Lehrstuhl Programmierparadigmen



Teil III

Quantoren in Isabelle/HOL

Quantoren in Isabelle/HOL

Karlsruher Institut für Technologie

Die üblichen zwei Quantoren der Logik:

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

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

Quantoren in Isabelle/HOL

Karlsruher Institut für Technologie

Die üblichen zwei Quantoren der Logik:

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

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

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

Quantoren in Isabelle/HOL

Die üblichen zwei Quantoren der Logik:

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

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

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

Karlsruher Institut für Technologie

Die üblichen zwei Quantoren der Logik:

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

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

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

Karlsruher Institut für Technologie

Die üblichen zwei Quantoren der Logik:

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

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

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

Karlsruher Institut für Technologie

Die üblichen zwei Quantoren der Logik:

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

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

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, das ein Wert *beliebig* ist?

Wie sagt man es Isabelle...?

Argumentation mit Quantoren erfordert Aussagen über *beliebige* Werte
Nur: wie weiss Isabelle, das ein Wert *beliebig* ist?

Lösung: Meta-Logik!
x in einer Aussage beliebig: brauchen Quantor dafür
Syntax in Isabelle: $\bigwedge x.\ [\![\ldots]\!] \Longrightarrow \ldots$
$\bigwedge$ heisst Meta-Allquantor, Variablen dahinter Parameter
Gültigkeitsbereich der Parameter: ganzes Subgoal
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, das ein Wert *beliebig* ist?

Lösung: Meta-Logik!
x in einer Aussage beliebig: brauchen Quantor dafür
Syntax in Isabelle: $\bigwedge x.\ [\![\ldots]\!] \Longrightarrow \ldots$
$\bigwedge$ heisst Meta-Allquantor, Variablen dahinter Parameter
Gültigkeitsbereich der Parameter: ganzes Subgoal
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, das ein Wert *beliebig* ist?

Lösung: Meta-Logik!
x in einer Aussage beliebig: brauchen Quantor dafür
Syntax in Isabelle: $\bigwedge x.\ [\![\ldots]\!] \Longrightarrow \ldots$
$\bigwedge$ heisst Meta-Allquantor, Variablen dahinter Parameter
Gültigkeitsbereich der Parameter: ganzes Subgoal
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 Introduktions- und Eliminationsregel:

Regeln

Jeder Quantor 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 Introduktions- und Eliminationsregel:

- `allI: (⋀x. P x) ⟹ ∀x. P x`
  Eine Aussage gilt für beliebige x (Meta-Ebene),
  also gilt sie auch für alle (HOL-Ebene)
- `allE: [[∀x. P x; P ?x ⟹ R]] ⟹ 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`

RegeIn

- *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*

RegeIn

- *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*
- *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

Karlsruher Institut für Technologie

Den Regeln *allE* und *exI* gemeinsam:
nach Anwendung der entsprechenden Methodik (also *erule* bzw. *rule*)
unspezifizierte Variablen (*?x*) in Subgoal
meist nicht gewollt, da schlecht Aussagen darüber möglich

Besser: entsprechende Variable gleich festlegen

Variablen festlegen bei Regelanwendung

Karlsruher Institut für Technologie

Den Regeln `allE` und `exI` gemeinsam:
nach Anwendung der entsprechenden Methodik (also `erule` bzw. `rule`)
unspezifizierte Variablen (`?x`) in Subgoal
meist nicht gewollt, da schlecht Aussagen darüber möglich

Besser: entsprechende Variable gleich festlegen
Methodik: `rule_tac v1 = t1` **and** `...` **and** `vk = tk` **in** `R`,
`?v1`, ..., `?vk` freie Variable in der anzuwendenden Regel `R`
(nicht im aktuellen Subgoal!)
analog: `erule_tac`

Variablen festlegen bei Regelanwendung

Karlsruher Institut für Technologie

Den Regeln `allE` und `exI` gemeinsam:
nach Anwendung der entsprechenden Methodik (also `erule` bzw. `rule`)
unspezifizierte Variablen (`?x`) in Subgoal
meist nicht gewollt, da schlecht Aussagen darüber möglich

Besser: entsprechende Variable gleich festlegen
Methodik: `rule_tac v1 = t1` **and** `...` **and** `vk = tk` **in** `R`,
`?v1`, ..., `?vk` freie Variable in der anzuwendenden Regel `R`
(nicht im aktuellen Subgoal!)
analog: `erule_tac`

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

Teil IV

Fallunterscheidung und Definition

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

Fallunterscheidung

In (klassischen) Beweisen Fallunterscheidung wichtiges Hilfsmittel

In Isabelle: einfach durch **apply** der Methode `cases P`
(P beliebiges boolesches Prädikat)
teilt aktuelles Subgoal in 2 neue auf:

- erstes mit P zusätzlich in den Annahmen
- zweites mit $\neg\ P$ in den Annahmen

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** `(cases A)` neue Subgoals:

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** `(cases A)` neue Subgoals:

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

Fallunterscheidung: Beispiel

aktuelles Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** `(cases A)` neue Subgoals:

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 Subgoal: 1. $[\![B;\ C]\!] \Longrightarrow A \wedge B \vee \neg A \wedge C$
so nicht (einfach) lösbar!

jedoch nach **apply** `(cases A)` neue Subgoals:

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*$\_def$)
kann dem Simplifier übergeben werden

definition

nand binärer Operator $\Longrightarrow$ Infixoperator bietet sich an
Syntaxdefinition in Isabelle durch Anhängen des Ausdrucks
(infixl "⋈" n) an die Deklarationszeile, wobei

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

Kann jetzt *A ⋈ B* statt *nand A B* schreiben

Teil V

Simplifikation

Simplifikation

genauer: Termersetzung, weil Ausdruck nicht notwendigerweise einfacher
Simplifikationsregeln: Gleichung
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

Simplifikation

genauer: Termersetzung, weil Ausdruck nicht notwendigerweise einfacher
Simplifikationsregeln: Gleichung
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

Beispiel: aktuelles Subgoal: $C \Longrightarrow \mathit{if}\ b\ \mathit{then}\ s1\ \mathit{else}\ s2$
`simp` wendet folgende Termersetzungsregel an:
$\mathit{if}\ ?x\ \mathit{then}\ ?m\ \mathit{else}\ ?n = ((?x \longrightarrow ?m) \wedge (\neg\ ?x \longrightarrow ?n))$
Resultat: $C \Longrightarrow (b \longrightarrow s1) \wedge (\neg\ b \longrightarrow s2)$

Simplifikation

Auch bedingte Ersetzungsregeln sind möglich, also in der Form

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

dazu: Prämissen der Regel mit aktuellen Annahmen unifizierbar

Simplifikation

Auch bedingte Ersetzungsregeln sind möglich, also in der Form

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

dazu: Prämissen der Regel mit aktuellen Annahmen unifizierbar

Simplifier modifizieren:

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

Simplifikation

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

- Zusatz *[simp]* hinter Lemmanamen
  Beispiel: **lemma** *bla [simp]: "A = True ⟹ A ∧ 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 ⟹ A ∧ 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
  Bsp: **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)`