
HOL: Propositional Logic

Overview

- Natural deduction
- Rule application in Isabelle/HOL

Rule notation

$$\frac{A_1 \dots A_n}{A}$$

instead of

$$[[A_1 \dots A_n]] \implies A$$

Natural Deduction

Natural deduction

Two kinds of rules for each logical operator \oplus :

Natural deduction

Two kinds of rules for each logical operator \oplus :

Introduction: how can I prove $A \oplus B$?

Natural deduction

Two kinds of rules for each logical operator \oplus :

Introduction: how can I prove $A \oplus B$?

Elimination: what can I prove from $A \oplus B$?

Natural deduction for propositional logic

$$\frac{}{A \wedge B} \text{ conjI}$$

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{}{A \vee B} \quad \frac{}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{}{A \longrightarrow B} \text{ impI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

$$\frac{}{A = B} \text{ iffI}$$

$$\frac{}{A \implies B} \text{ iffD1} \quad \frac{}{B \implies A} \text{ iffD2}$$

$$\frac{}{\neg A} \text{ notI}$$

$$\frac{\neg A}{C} \text{ notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{ conjI}$$

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{}{A \vee B} \quad \frac{}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{}{A \longrightarrow B} \text{ impI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

$$\frac{}{A = B} \text{ iffI}$$

$$\frac{}{A \implies B} \text{ iffD1} \quad \frac{}{B \implies A} \text{ iffD2}$$

$$\frac{}{\neg A} \text{ notI}$$

$$\frac{\neg A}{C} \text{ notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{ conjI}$$

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{}{A \longrightarrow B} \text{ impI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

$$\frac{}{A = B} \text{ iffI}$$

$$\frac{}{A \implies B} \text{ iffD1} \quad \frac{}{B \implies A} \text{ iffD2}$$

$$\frac{}{\neg A} \text{ notI}$$

$$\frac{\neg A}{C} \text{ notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{ conjI}$$

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{A \implies B}{A \longrightarrow B} \text{ impI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

$$\frac{}{A = B} \text{ iffI}$$

$$\frac{}{A \implies B} \text{ iffD1} \quad \frac{}{B \implies A} \text{ iffD2}$$

$$\frac{}{\neg A} \text{ notI}$$

$$\frac{\neg A}{C} \text{ notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{ conjI}$$

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{A \implies B}{A \longrightarrow B} \text{ impI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

$$\frac{A \implies B \quad B \implies A}{A = B} \text{ iffI}$$

$$\overline{A \implies B} \text{ iffD1} \quad \overline{B \implies A} \text{ iffD2}$$

$$\frac{}{\neg A} \text{ notI}$$

$$\frac{\neg A}{C} \text{ notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{ conjI}$$

$$\frac{A \wedge B}{C} \text{ conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{ disjI1/2}$$

$$\frac{A \vee B}{C} \text{ disjE}$$

$$\frac{A \implies B}{A \longrightarrow B} \text{ impI}$$

$$\frac{A \longrightarrow B}{C} \text{ impE}$$

$$\frac{A \implies B \quad B \implies A}{A = B} \text{ iffI}$$

$$\overline{A \implies B} \text{ iffD1} \quad \overline{B \implies A} \text{ iffD2}$$

$$\frac{A \implies \text{False}}{\neg A} \text{ notI}$$

$$\frac{\neg A}{C} \text{ notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{conjI}$$

$$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{disjI1/2}$$

$$\frac{A \vee B}{C} \text{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{impI}$$

$$\frac{A \longrightarrow B}{C} \text{impE}$$

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B} \text{iffI}$$

$$\overline{A \Longrightarrow B} \text{iffD1} \quad \overline{B \Longrightarrow A} \text{iffD2}$$

$$\frac{A \Longrightarrow \text{False}}{\neg A} \text{notI}$$

$$\frac{\neg A}{C} \text{notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{conjI}$$

$$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{disjI1/2}$$

$$\frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{impI}$$

$$\frac{A \longrightarrow B}{C} \text{impE}$$

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B} \text{iffI}$$

$$\overline{A \Longrightarrow B} \text{iffD1} \quad \overline{B \Longrightarrow A} \text{iffD2}$$

$$\frac{A \Longrightarrow \text{False}}{\neg A} \text{notI}$$

$$\frac{\neg A}{C} \text{notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{conjI}$$

$$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{disjI1/2}$$

$$\frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{impI}$$

$$\frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C} \text{impE}$$

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B} \text{iffI}$$

$$\overline{A \Longrightarrow B} \text{iffD1} \quad \overline{B \Longrightarrow A} \text{iffD2}$$

$$\frac{A \Longrightarrow \text{False}}{\neg A} \text{notI}$$

$$\frac{\neg A}{C} \text{notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{conjI}$$

$$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{disjI1/2}$$

$$\frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{impI}$$

$$\frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C} \text{impE}$$

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B} \text{iffI}$$

$$\frac{A=B}{A \Longrightarrow B} \text{iffD1} \quad \frac{A=B}{B \Longrightarrow A} \text{iffD2}$$

$$\frac{A \Longrightarrow \text{False}}{\neg A} \text{notI}$$

$$\frac{\neg A}{C} \text{notE}$$

Natural deduction for propositional logic

$$\frac{A \quad B}{A \wedge B} \text{conjI}$$

$$\frac{A \wedge B \quad \llbracket A; B \rrbracket \Longrightarrow C}{C} \text{conjE}$$

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B} \text{disjI1/2}$$

$$\frac{A \vee B \quad A \Longrightarrow C \quad B \Longrightarrow C}{C} \text{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B} \text{impI}$$

$$\frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C} \text{impE}$$

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B} \text{iffI}$$

$$\frac{A = B}{A \Longrightarrow B} \text{iffD1} \quad \frac{A = B}{B \Longrightarrow A} \text{iffD2}$$

$$\frac{A \Longrightarrow \text{False}}{\neg A} \text{notI}$$

$$\frac{\neg A \quad A}{C} \text{notE}$$

Operational reading

$$\frac{A_1 \dots A_n}{A}$$

Operational reading

$$\frac{A_1 \dots A_n}{A}$$

Introduction rule:

To prove A it suffices to prove $A_1 \dots A_n$.

Operational reading

$$\frac{A_1 \dots A_n}{A}$$

Introduction rule:

To prove A it suffices to prove $A_1 \dots A_n$.

Elimination rule

If I know A_1 and want to prove A
it suffices to prove $A_2 \dots A_n$.

Equality

$$\frac{}{t = t} \text{ refl}$$

$$\frac{s = t}{t = s} \text{ sym}$$

$$\frac{r = s \quad s = t}{r = t} \text{ trans}$$

Equality

$$\frac{}{t = t} \text{ refl}$$

$$\frac{s = t}{t = s} \text{ sym}$$

$$\frac{r = s \quad s = t}{r = t} \text{ trans}$$

$$\frac{s = t \quad A(s)}{A(t)} \text{ subst}$$

Equality

$$\frac{}{t = t} \text{ refl} \quad \frac{s = t}{t = s} \text{ sym} \quad \frac{r = s \quad s = t}{r = t} \text{ trans}$$

$$\frac{s = t \quad A(s)}{A(t)} \text{ subst}$$

Rarely needed explicitly — used implicitly by *simp*

More rules

$$\frac{A \longrightarrow B \quad A}{B} \text{ mp}$$

More rules

$$\frac{A \longrightarrow B \quad A}{B} \text{ mp}$$

$$\frac{\neg A \Longrightarrow \textit{False}}{A} \text{ ccontr}$$

$$\frac{\neg A \Longrightarrow A}{A} \text{ classical}$$

More rules

$$\frac{A \longrightarrow B \quad A}{B} \text{ mp}$$

$$\frac{\neg A \Longrightarrow \text{False}}{A} \text{ ccontr}$$

$$\frac{\neg A \Longrightarrow A}{A} \text{ classical}$$

Remark:

ccontr and classical are not derivable from the ND-rules.

More rules

$$\frac{A \longrightarrow B \quad A}{B} \text{ mp}$$

$$\frac{\neg A \Longrightarrow \text{False}}{A} \text{ ccontr}$$

$$\frac{\neg A \Longrightarrow A}{A} \text{ classical}$$

Remark:

`ccontr` and `classical` are not derivable from the ND-rules.

They make the logic “classical”, i.e. “non-constructive”.

Proof by assumption

$$\frac{A_1 \quad \dots \quad A_n}{A_i} \text{ assumption}$$

Rule application: the rough idea

Applying rule $\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow A$ to subgoal C :

Rule application: the rough idea

Applying rule $\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow A$ to subgoal C :

- Unify A and C

Rule application: the rough idea

Applying rule $\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow A$ to subgoal C :

- Unify A and C
- Replace C with n new subgoals $A_1 \dots A_n$

Rule application: the rough idea

Applying rule $\llbracket A_1; \dots ; A_n \rrbracket \implies A$ to subgoal C :

- Unify A and C
- Replace C with n new subgoals $A_1 \dots A_n$

Working backwards, like in Prolog!

Rule application: the rough idea

Applying rule $\llbracket A_1; \dots ; A_n \rrbracket \implies A$ to subgoal C :

- Unify A and C
- Replace C with n new subgoals $A_1 \dots A_n$

Working backwards, like in Prolog!

Example: rule: $\llbracket ?P; ?Q \rrbracket \implies ?P \wedge ?Q$
subgoal: 1. $A \wedge B$

Rule application: the rough idea

Applying rule $\llbracket A_1; \dots ; A_n \rrbracket \implies A$ to subgoal C :

- Unify A and C
- Replace C with n new subgoals $A_1 \dots A_n$

Working backwards, like in Prolog!

Example: rule: $\llbracket ?P; ?Q \rrbracket \implies ?P \wedge ?Q$

subgoal: 1. $A \wedge B$

Result: 1. A

2. B

Rule application: the details

Rule: $\llbracket A_1; \dots ; A_n \rrbracket \implies A$
Subgoal: 1. $\llbracket B_1; \dots ; B_m \rrbracket \implies C$

Rule application: the details

Rule: $\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow A$
Subgoal: 1. $\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow C$
Substitution: $\sigma(A) \equiv \sigma(C)$

Rule application: the details

Rule: $\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow C$

Substitution: $\sigma(A) \equiv \sigma(C)$

New subgoals: 1. $\sigma(\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow A_1)$

⋮

$n.$ $\sigma(\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow A_n)$

Rule application: the details

Rule: $\llbracket A_1; \dots ; A_n \rrbracket \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow C$

Substitution: $\sigma(A) \equiv \sigma(C)$

New subgoals: 1. $\sigma(\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow A_1)$

⋮

$n.$ $\sigma(\llbracket B_1; \dots ; B_m \rrbracket \Longrightarrow A_n)$

Command:

apply(rule <rulename>)

Proof by assumption

apply assumption

proves

$$1. \ [B_1; \dots ; B_m] \implies C$$

by unifying C with one of the B_i

Proof by assumption

apply assumption

proves

$$1. \ [B_1; \dots ; B_m] \implies C$$

by unifying C with one of the B_i (backtracking!)

Demo: application of introduction rule

Applying elimination rules

`apply(erule <elim-rule>)`

Like *rule* but also

- unifies first premise of rule with an assumption
- eliminates that assumption

Applying elimination rules

`apply(erule <elim-rule>)`

Like *rule* but also

- unifies first premise of rule with an assumption
- eliminates that assumption

Example:

Rule: $\llbracket ?P \wedge ?Q; \llbracket ?P; ?Q \rrbracket \Longrightarrow ?R \rrbracket \Longrightarrow ?R$

Subgoal: 1. $\llbracket X; A \wedge B; Y \rrbracket \Longrightarrow Z$

Applying elimination rules

`apply(erule <elim-rule>)`

Like *rule* but also

- unifies first premise of rule with an assumption
- eliminates that assumption

Example:

Rule: $\llbracket ?P \wedge ?Q; \llbracket ?P; ?Q \rrbracket \Longrightarrow ?R \rrbracket \Longrightarrow ?R$

Subgoal: 1. $\llbracket X; A \wedge B; Y \rrbracket \Longrightarrow Z$

Unification: $?P \wedge ?Q \equiv A \wedge B$ and $?R \equiv Z$

Applying elimination rules

`apply(erule <elim-rule>)`

Like *rule* but also

- unifies first premise of rule with an assumption
- eliminates that assumption

Example:

Rule: $\llbracket ?P \wedge ?Q; \llbracket ?P; ?Q \rrbracket \Longrightarrow ?R \rrbracket \Longrightarrow ?R$

Subgoal: 1. $\llbracket X; A \wedge B; Y \rrbracket \Longrightarrow Z$

Unification: $?P \wedge ?Q \equiv A \wedge B$ and $?R \equiv Z$

New subgoal: 1. $\llbracket X; Y \rrbracket \Longrightarrow \llbracket A; B \rrbracket \Longrightarrow Z$

Applying elimination rules

$\text{apply}(\text{erule } \langle \text{elim-rule} \rangle)$

Like *rule* but also

- unifies first premise of rule with an assumption
- eliminates that assumption

Example:

Rule: $\llbracket ?P \wedge ?Q; \llbracket ?P; ?Q \rrbracket \Longrightarrow ?R \rrbracket \Longrightarrow ?R$

Subgoal: 1. $\llbracket X; A \wedge B; Y \rrbracket \Longrightarrow Z$

Unification: $?P \wedge ?Q \equiv A \wedge B$ and $?R \equiv Z$

New subgoal: 1. $\llbracket X; Y \rrbracket \Longrightarrow \llbracket A; B \rrbracket \Longrightarrow Z$

same as: 1. $\llbracket X; Y; A; B \rrbracket \Longrightarrow Z$

How to prove it by natural deduction

- **Intro** rules decompose formulae to the right of \implies .
apply(rule <intro-rule>)

How to prove it by natural deduction

- **Intro** rules decompose formulae to the right of \implies .
apply(rule <intro-rule>)
- **Elim** rules decompose formulae on the left of \implies .
apply(erule <elim-rule>)

Demo: examples

Safe and unsafe rules

Safe rules preserve provability

Safe and unsafe rules

Safe rules preserve provability

conjI, impI, notI, iffI, refl, ccontr,
classical, conjE, disjE

Safe and unsafe rules

Safe rules preserve provability

conjI, impI, notI, iffI, refl, ccontr,
classical, conjE, disjE

Unsafe rules can turn a provable goal into an unprovable one

Safe and unsafe rules

Safe rules preserve provability

conjI, impI, notI, iffI, refl, ccontr,
classical, conjE, disjE

Unsafe rules can turn a provable goal into an unprovable one

disjI1, disjI2, impE, iffD1, iffD2, notE

Safe and unsafe rules

Safe rules preserve provability

conjI, impI, notI, iffI, refl, ccontr,
classical, conjE, disjE

Unsafe rules can turn a provable goal into an unprovable one

disjI1, disjI2, impE, iffD1, iffD2, notE

Apply safe rules before unsafe ones

\implies **VS** \longrightarrow

- Write theorems as $\llbracket A_1; \dots; A_n \rrbracket \implies A$
not as $A_1 \wedge \dots \wedge A_n \longrightarrow A$ (to ease application)

\implies **VS** \longrightarrow

- Write theorems as $\llbracket A_1; \dots; A_n \rrbracket \implies A$
not as $A_1 \wedge \dots \wedge A_n \longrightarrow A$ (to ease application)
- *Exception* (in **apply**-style): induction variable must not occur in the premises.

\implies **VS** \longrightarrow

- Write theorems as $\llbracket A_1; \dots; A_n \rrbracket \implies A$
not as $A_1 \wedge \dots \wedge A_n \longrightarrow A$ (to ease application)
- *Exception* (in **apply**-style): induction variable must not occur in the premises.

Example: $\llbracket A; B(x) \rrbracket \implies C(x) \rightsquigarrow A \implies B(x) \longrightarrow C(x)$

\implies **VS** \longrightarrow

- Write theorems as $\llbracket A_1; \dots; A_n \rrbracket \implies A$
not as $A_1 \wedge \dots \wedge A_n \longrightarrow A$ (to ease application)
- *Exception* (in `apply`-style): induction variable must not occur in the premises.

Example: $\llbracket A; B(x) \rrbracket \implies C(x) \rightsquigarrow A \implies B(x) \longrightarrow C(x)$

Reverse transformation (after proof):

`lemma abc`*[rule_format]*: $A \implies B(x) \longrightarrow C(x)$

Demo: further techniques

HOL: Predicate Logic

Parameters

Subgoal:

1. $\bigwedge x_1 \dots x_n$. *Formula*

The x_i are called **parameters** of the subgoal.

Intuition: local constants, i.e. arbitrary but fixed values.

Parameters

Subgoal:

1. $\wedge x_1 \dots x_n$. *Formula*

The x_i are called **parameters** of the subgoal.

Intuition: local constants, i.e. arbitrary but fixed values.

Rules are automatically lifted over $\wedge x_1 \dots x_n$ and applied directly to *Formula*.

Scope

- Scope of parameters: whole subgoal
- Scope of \forall , \exists , \dots : ends with $;$ or \implies

Scope

- Scope of parameters: whole subgoal
- Scope of \forall, \exists, \dots : ends with ; or \implies

$$\wedge x y. [\forall y. P y \longrightarrow Q z y; Q x y] \implies \exists x. Q x y$$

means

$$\wedge x y. [(\forall y_1. P y_1 \longrightarrow Q z y_1); Q x y] \implies \exists x_1. Q x_1 y$$

α -Conversion

- $\forall x. P(x)$: x can appear in $P(x)$.

α -Conversion

- $\forall x. P(x)$: x can appear in $P(x)$.

Example: $\forall x. x = x$ is obtained by $P \mapsto \lambda u. u = u$

α -Conversion

- $\forall x. P(x)$: x can appear in $P(x)$.

Example: $\forall x. x = x$ is obtained by $P \mapsto \lambda u. u = u$

- $\forall x. P$: x cannot appear in P .

α -Conversion

- $\forall x. P(x)$: x can appear in $P(x)$.

Example: $\forall x. x = x$ is obtained by $P \mapsto \lambda u. u = u$

- $\forall x. P$: x cannot appear in P .

Example: $P \mapsto x = x$ yields $\forall x'. x = x$

α -Conversion

- $\forall x. P(x)$: x can appear in $P(x)$.

Example: $\forall x. x = x$ is obtained by $P \mapsto \lambda u. u = u$

- $\forall x. P$: x cannot appear in P .

Example: $P \mapsto x = x$ yields $\forall x'. x = x$

Bound variables are renamed automatically to avoid name clashes with other variables.

Natural deduction for quantifiers

$\overline{\forall x. P(x)}$ allI

$\frac{\forall x. P(x)}{R}$ allE

$\overline{\exists x. P(x)}$ exI

$\frac{\exists x. P(x)}{R}$ exE

Natural deduction for quantifiers

$$\frac{\wedge x. P(x)}{\forall x. P(x)} \text{allI}$$

$$\frac{\forall x. P(x)}{R} \text{allE}$$

$$\frac{}{\exists x. P(x)} \text{exI}$$

$$\frac{\exists x. P(x)}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\wedge x. P(x)}{\forall x. P(x)} \text{allI}$$

$$\frac{\forall x. P(x)}{R} \text{allE}$$

$$\frac{P(?x)}{\exists x. P(x)} \text{exI}$$

$$\frac{\exists x. P(x)}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\wedge x. P(x)}{\forall x. P(x)} \text{allI}$$

$$\frac{\forall x. P(x) \quad P(?x) \implies R}{R} \text{allE}$$

$$\frac{P(?x)}{\exists x. P(x)} \text{exI}$$

$$\frac{\exists x. P(x)}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\wedge x. P(x)}{\forall x. P(x)} \text{allI}$$

$$\frac{\forall x. P(x) \quad P(?x) \implies R}{R} \text{allE}$$

$$\frac{P(?x)}{\exists x. P(x)} \text{exI}$$

$$\frac{\exists x. P(x) \quad \wedge x. P(x) \implies R}{R} \text{exE}$$

Natural deduction for quantifiers

$$\frac{\wedge x. P(x)}{\forall x. P(x)} \text{allI}$$

$$\frac{\forall x. P(x) \quad P(?x) \implies R}{R} \text{allE}$$

$$\frac{P(?x)}{\exists x. P(x)} \text{exI}$$

$$\frac{\exists x. P(x) \quad \wedge x. P(x) \implies R}{R} \text{exE}$$

- allI and exE introduce new parameters ($\wedge x$).

Natural deduction for quantifiers

$$\frac{\wedge x. P(x)}{\forall x. P(x)} \text{allI} \qquad \frac{\forall x. P(x) \quad P(?x) \implies R}{R} \text{allE}$$

$$\frac{P(?x)}{\exists x. P(x)} \text{exI} \qquad \frac{\exists x. P(x) \quad \wedge x. P(x) \implies R}{R} \text{exE}$$

- allI and exE introduce new parameters ($\wedge x$).
- allE and exI introduce new unknowns ($?x$).

Instantiating rules

apply(rule_tac x = term in rule)

Like *rule*, but *?x* in *rule* is instantiated by *term* before application.

Instantiating rules

apply(rule_tac x = term in rule)

Like *rule*, but $?x$ in *rule* is instantiated by *term* before application.

Similar: *erule_tac*

Instantiating rules

`apply(rule_tac x = term in rule)`

Like *rule*, but $?x$ in *rule* is instantiated by *term* before application.

Similar: `erule_tac`

! x is in *rule*, not in the goal **!**

Two successful proofs

1. $\forall x. \exists y. x = y$

Two successful proofs

1. $\forall x. \exists y. x = y$
apply(*rule alll*)

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(*rule alll*)

1. $\wedge x. \exists y. x = y$

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exI)

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exI)

1. $\wedge x. x = x$

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exI)

1. $\wedge x. x = x$

apply(rule refl)

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exl)

1. $\wedge x. x = x$

apply(rule refl)

exploration

apply(rule exl)

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule allI)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exI)

1. $\wedge x. x = x$

apply(rule refl)

exploration

apply(rule exI)

1. $\wedge x. x = ?y x$

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exl)

1. $\wedge x. x = x$

apply(rule refl)

exploration

apply(rule exl)

1. $\wedge x. x = ?y x$

apply(rule refl)

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exl)

1. $\wedge x. x = x$

apply(rule refl)

exploration

apply(rule exl)

1. $\wedge x. x = ?y x$

apply(rule refl)

?y \mapsto

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exl)

1. $\wedge x. x = x$

apply(rule refl)

exploration

apply(rule exl)

1. $\wedge x. x = ?y x$

apply(rule refl)

$?y \mapsto \lambda u. u$

Two successful proofs

1. $\forall x. \exists y. x = y$

apply(rule alll)

1. $\wedge x. \exists y. x = y$

best practice

apply(rule_tac x = x in exl)

1. $\wedge x. x = x$

apply(rule refl)

simpler & clearer

exploration

apply(rule exl)

1. $\wedge x. x = ?y x$

apply(rule refl)

$?y \mapsto \lambda u. u$

shorter & trickier

Two unsuccessful proofs

$$1. \exists y. \forall x. x = y$$

Two unsuccessful proofs

$$1. \exists y. \forall x. x = y$$

apply(rule_tac x = ??? in exI)

Two unsuccessful proofs

1. $\exists y. \forall x. x = y$

apply(rule_tac x = ??? in ex1)

apply(rule ex1)

1. $\forall x. x = ?y$

Two unsuccessful proofs

1. $\exists y. \forall x. x = y$

apply(rule_tac x = ??? in exI)

apply(rule exI)

1. $\forall x. x = ?y$

apply(rule allI)

1. $\wedge x. x = ?y$

Two unsuccessful proofs

$$1. \exists y. \forall x. x = y$$

apply(rule_tac x = ??? in exI)

apply(rule exI)

$$1. \forall x. x = ?y$$

apply(rule allI)

$$1. \wedge x. x = ?y$$

apply(rule refl)

Two unsuccessful proofs

1. $\exists y. \forall x. x = y$

apply(rule_tac x = ??? in ex1)

apply(rule ex1)

1. $\forall x. x = ?y$

apply(rule all1)

1. $\wedge x. x = ?y$

apply(rule refl)

$?y \mapsto x$ yields $\wedge x'. x' = x$

Two unsuccessful proofs

1. $\exists y. \forall x. x = y$

apply(rule_tac x = ??? in exI)

apply(rule exI)

1. $\forall x. x = ?y$

apply(rule allI)

1. $\wedge x. x = ?y$

apply(rule refl)

$?y \mapsto x$ yields $\wedge x'. x' = x$

Principle:

?f $x_1 \dots x_n$ can only be replaced by term t

if $params(t) \subseteq \{x_1, \dots, x_n\}$

Demo: quantifier proofs

Safe and unsafe rules

Safe allI, exE

Unsafe allE, exI

Safe and unsafe rules

Safe allI, exE

Unsafe allE, exI

Create parameters first, unknowns later

Proof methods

Parameter names

Parameter names are chosen by Isabelle

Parameter names

Parameter names are chosen by Isabelle

1. $\forall x. \exists y. x = y$

`apply(rule allI)`

1. $\wedge x. \exists y. x = y$

`apply(rule_tac x = x in exI)`

Parameter names

Parameter names are chosen by Isabelle

1. $\forall x. \exists y. x = y$

`apply(rule allI)`

1. $\wedge x. \exists y. x = y$

`apply(rule_tac x = x in exI)`

Brittle!

Renaming parameters

1. $\forall x. \exists y. x = y$

apply(*rule allI*)

1. $\wedge x. \exists y. x = y$

apply(*rename_tac xxx*)

1. $\wedge xxx. \exists y. xxx = y$

apply(*rule_tac x = xxx in exI*)

Renaming parameters

1. $\forall x. \exists y. x = y$

apply(rule allI)

1. $\wedge x. \exists y. x = y$

apply(rename_tac xxx)

1. $\wedge xxx. \exists y. xxx = y$

apply(rule_tac x = xxx in exI)

In general:

(rename_tac $x_1 \dots x_n$) renames the rightmost
(inner) n parameters to $x_1 \dots x_n$

Forward proofs: frule and drule

“Forward” rule: $A_1 \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow C$

Forward proofs: frule and drule

“Forward” rule: $A_1 \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow C$

Substitution: $\sigma(B_i) \equiv \sigma(A_1)$

Forward proofs: frule and drule

“Forward” rule: $A_1 \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow C$

Substitution: $\sigma(B_i) \equiv \sigma(A_1)$

New subgoal: 1. $\sigma(\llbracket B_1; \dots ; B_n; A \rrbracket) \Longrightarrow C$

Forward proofs: frule and drule

“Forward” rule: $A_1 \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow C$

Substitution: $\sigma(B_i) \equiv \sigma(A_1)$

New subgoal: 1. $\sigma(\llbracket B_1; \dots ; B_n; A \rrbracket) \Longrightarrow C$

Command:

apply(frule rulename)

Forward proofs: frule and drule

“Forward” rule: $A_1 \Longrightarrow A$

Subgoal: 1. $\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow C$

Substitution: $\sigma(B_i) \equiv \sigma(A_1)$

New subgoal: 1. $\sigma(\llbracket B_1; \dots ; B_n; A \rrbracket \Longrightarrow C)$

Command:

apply(frule rulename)

Like *frule* but also deletes B_i :

apply(drule rulename)

frule and drule: the general case

Rule: $\llbracket A_1; \dots ; A_m \rrbracket \Longrightarrow A$

Creates additional subgoals:

$$1. \sigma(\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow A_2)$$

\vdots

$$m-1. \sigma(\llbracket B_1; \dots ; B_n \rrbracket \Longrightarrow A_m)$$

$$m. \sigma(\llbracket B_1; \dots ; B_n; A \rrbracket \Longrightarrow C)$$

Forward proofs: OF

$$r[OF r_1 \dots r_n]$$

Prove assumption 1 of theorem r with theorem r_1 ,
and assumption 2 with theorem r_2 , and ...

Forward proofs: OF

$r[OF\ r_1\ \dots\ r_n]$

Prove assumption 1 of theorem r with theorem r_1 ,
and assumption 2 with theorem r_2 , and ...

Rule r $\llbracket A_1; \dots ; A_m \rrbracket \implies A$

Rule r_1 $\llbracket B_1; \dots ; B_n \rrbracket \implies B$

Substitution $\sigma(B) \equiv \sigma(A_1)$

$r[OF\ r_1]$

Forward proofs: OF

$r[OF\ r_1\ \dots\ r_n]$

Prove assumption 1 of theorem r with theorem r_1 ,
and assumption 2 with theorem r_2 , and ...

Rule r $\llbracket A_1; \dots ; A_m \rrbracket \implies A$

Rule r_1 $\llbracket B_1; \dots ; B_n \rrbracket \implies B$

Substitution $\sigma(B) \equiv \sigma(A_1)$

$r[OF\ r_1]$ $\sigma(\llbracket B_1; \dots ; B_n; A_2; \dots ; A_m \rrbracket \implies A)$

Forward proofs: THEN

$r_1[THEN\ r_2]$ means $r_2[OF\ r_1]$

Clarifying the goal

Clarifying the goal

- **apply**(*intro ...*)
Repeated application of intro rules
Example: **apply**(*intro all*)

Clarifying the goal

- **apply**(*intro ...*)
Repeated application of intro rules
Example: **apply**(*intro allI*)
- **apply**(*elim ...*)
Repeated application of elim rules
Example: **apply**(*elim conjE*)

Clarifying the goal

- **apply**(*intro ...*)
Repeated application of intro rules
Example: **apply**(*intro allI*)
- **apply**(*elim ...*)
Repeated application of elim rules
Example: **apply**(*elim conjE*)
- **apply**(*clarify*)
Repeated application of safe rules
without splitting the goal

Clarifying the goal

- **apply**(*intro ...*)
Repeated application of intro rules
Example: **apply**(*intro allI*)
- **apply**(*elim ...*)
Repeated application of elim rules
Example: **apply**(*elim conjE*)
- **apply**(*clarify*)
Repeated application of safe rules
without splitting the goal
- **apply**(*clarsimp simp add: ...*)
Combination of *clarify* and *simp*.

Demo: proof methods