



COMP4161: Advanced Topics in Software Verification

HOL

Gerwin Klein, Johannes Åman Pohjola, Christine Rizkallah, Miki Tanaka
T3/2020

Last time...

- natural deduction rules for \wedge , \vee , \longrightarrow , \neg , iff...
- proof by assumption, by intro rule, elim rule
- safe and unsafe rules
- indent your proofs! (one space per subgoal)
- prefer implicit backtracking (chaining) or *rule_tac*, instead of *back*
- *prefer* and *defer*
- *oops* and *sorry*

Content

→ Foundations & Principles

- Intro, Lambda calculus, natural deduction [1,2]
- Higher Order Logic, Isar (part 1) [2,3^a]
- Term rewriting [3,4]

→ Proof & Specification Techniques

- Inductively defined sets, rule induction, datatype induction, primitive recursion [4,5]
- General recursive functions, termination proofs [7^b]
- Proof automation, Hoare logic, proofs about programs, invariants [8]
- C verification [9,10]
- Practice, questions, exam prep [10^c]

^aa1 due; ^ba2 due; ^ca3 due

Quantifiers

Scope

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

Example:

Scope

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

Example:

$$\bigwedge x y. \llbracket \forall y. P y \longrightarrow Q z y; Q x y \rrbracket \implies \exists x. Q x y$$

means

Scope

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

Example:

$$\wedge x y. \llbracket \forall y. P y \longrightarrow Q z y; Q x y \rrbracket \implies \exists x. Q x y$$

means

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

Natural deduction for quantifiers

$$\frac{}{\forall x. P x} \text{ allI}$$

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

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

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

Natural deduction for quantifiers

$$\frac{\bigwedge x. P x}{\bigvee x. P x} \text{ allI}$$

$$\frac{\bigvee x. P x}{R} \text{ allE}$$

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

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

Natural deduction for quantifiers

$$\frac{\bigwedge x. P x}{\bigvee x. P x} \text{ allI}$$

$$\frac{\bigvee x. P x \quad P ?x \implies R}{R} \text{ allE}$$

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

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

Natural deduction for quantifiers

$$\frac{\bigwedge x. P x}{\bigvee x. P x} \text{ allI}$$

$$\frac{\bigvee x. P x \quad P ?x \implies R}{R} \text{ allE}$$

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

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

Natural deduction for quantifiers

$$\frac{\bigwedge x. P x}{\forall x. P x} \text{ allI}$$

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

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

$$\frac{\exists x. P x \quad \bigwedge x. P x \implies R}{R} \text{ exE}$$

Natural deduction for quantifiers

$$\frac{\bigwedge x. P x}{\forall x. P x} \text{ allI} \qquad \frac{\forall x. P x \quad P ?x \implies R}{R} \text{ allE}$$
$$\frac{P ?x}{\exists x. P x} \text{ exI} \qquad \frac{\exists x. P x \quad \bigwedge x. P x \implies R}{R} \text{ exE}$$

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

Similar: `erule_tac`

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

Two Successful Proofs

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

Two Successful Proofs

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

apply (rule allI)

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

Two Successful Proofs

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

apply (rule allI)

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

best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x. x = x$

Two Successful Proofs

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

apply (rule allI)

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

best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x. x = x$

apply (rule refl)

Two Successful Proofs

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

apply (rule allI)

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

best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x. x = x$

apply (rule refl)

exploration

apply (rule exI)

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

Two Successful Proofs

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

apply (rule allI)

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

best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x. x = x$

apply (rule refl)

exploration

apply (rule exI)

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

apply (rule refl)

$?y \mapsto \lambda u. u$

Two Successful Proofs

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

apply (rule allI)

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

best practice

apply (rule_tac x = "x" in exI)

1. $\bigwedge x. x = x$

apply (rule refl)

simpler & clearer

exploration

apply (rule exI)

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

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 exI)

apply (rule exI)

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

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. $\bigwedge x. x = ?y$

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. $\bigwedge x. x = ?y$

apply (rule refl)

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

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. $\bigwedge x. x = ?y$

apply (rule refl)

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

Principle:

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

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

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

Demo: Quantifier Proofs

Parameter names

Parameter names are chosen by Isabelle

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

Parameter names

Parameter names are chosen by Isabelle

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

apply (rule allI)

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

Parameter names

Parameter names are chosen by Isabelle

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

apply (rule all1)

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

apply (rule_tac x = "x" in ex1)

Brittle!

Renaming parameters

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

apply (rule allI)

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

Renaming parameters

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

apply (rule all)

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

apply (rename_tac N)

1. $\bigwedge N. \exists y. N = y$

Renaming parameters

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

apply (rule_all)

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

apply (rename_tac N)

1. $\bigwedge N. \exists y. N = y$

apply (rule_tac x = "N" in ex1)

In general:

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

Forward Proof: frule and drule

apply (frule < *rule* >)

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

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

Forward Proof: frule and drule

apply (frule < *rule* >)

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

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

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

Forward Proof: frule and drule

apply (frule < rule >)

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

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

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

New subgoals: 1. $\sigma(\llbracket B_1; \dots; B_n \rrbracket) \implies A_2)$

\vdots

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

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

Forward Proof: frule and drule

apply (frule < rule >)

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

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

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

New subgoals: 1. $\sigma(\llbracket B_1; \dots; B_n \rrbracket) \implies A_2)$

\vdots

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

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

Like **frule** but also deletes B_i : **apply** (drule < rule >)

Examples for Forward Rules

$$\frac{P \wedge Q}{P} \text{ conjunct1} \quad \frac{P \wedge Q}{Q} \text{ conjunct2}$$

$$\frac{P \longrightarrow Q \quad P}{Q} \text{ mp}$$

$$\frac{\forall x. P \ x}{P \ ?_x} \text{ spec}$$

Forward Proof: 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 Proof: OF

$$r \text{ [OF } r_1 \dots r_n]$$

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

$$\text{Rule } r \quad \llbracket A_1; \dots; A_m \rrbracket \implies A$$

$$\text{Rule } r_1 \quad \llbracket B_1; \dots; B_n \rrbracket \implies B$$

Forward Proof: OF

$$r \text{ [OF } r_1 \dots r_n]$$

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

$$\text{Rule } r \quad \llbracket A_1; \dots; A_m \rrbracket \implies A$$

$$\text{Rule } r_1 \quad \llbracket B_1; \dots; B_n \rrbracket \implies B$$

$$\text{Substitution} \quad \sigma(B) \equiv \sigma(A_1)$$

Forward Proof: OF

$$r \text{ [OF } r_1 \dots r_n]$$

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

$$\text{Rule } r \quad \llbracket A_1; \dots; A_m \rrbracket \implies A$$

$$\text{Rule } r_1 \quad \llbracket B_1; \dots; B_n \rrbracket \implies B$$

$$\text{Substitution} \quad \sigma(B) \equiv \sigma(A_1)$$

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

Forward proofs: THEN

r_1 [THEN r_2] means r_2 [OF r_1]

Demo: Forward Proofs

Hilbert's Epsilon Operator



(David Hilbert, 1862-1943)

$\varepsilon x. Px$ is a value that satisfies P (if such a value exists)

Hilbert's Epsilon Operator



(David Hilbert, 1862-1943)

$\varepsilon x. P x$ is a value that satisfies P (if such a value exists)

ε also known as **description operator**.

In Isabelle the ε -operator is written $\text{SOME } x. P x$

Hilbert's Epsilon Operator



(David Hilbert, 1862-1943)

$\varepsilon x. P x$ is a value that satisfies P (if such a value exists)

ε also known as **description operator**.

In Isabelle the ε -operator is written $\text{SOME } x. P x$

$$\frac{P ?x}{P (\text{SOME } x. P x)} \text{ someI}$$

More Epsilon

ε implies Axiom of Choice:

$$\forall x. \exists y. Q x y \implies \exists f. \forall x. Q x (f x)$$

Existential and universal quantification can be defined with ε .

More Epsilon

ε implies Axiom of Choice:

$$\forall x. \exists y. Q x y \implies \exists f. \forall x. Q x (f x)$$

Existential and universal quantification can be defined with ε .

Isabelle also knows the definite description operator **THE** (aka ι):

$$\overline{(\text{THE } x. x = a)} \text{ the_eq_trivial}$$

Some Automation

More Proof Methods:

apply (intro <intro-rules>) repeatedly applies intro rules

apply (elim <elim-rules>) repeatedly applies elim rules

Some Automation

More Proof Methods:

apply (intro <intro-rules>) repeatedly applies intro rules

apply (elim <elim-rules>) repeatedly applies elim rules

apply clarify applies all safe rules
that do not split the goal

Some Automation

More Proof Methods:

apply (intro <intro-rules>) repeatedly applies intro rules

apply (elim <elim-rules>) repeatedly applies elim rules

apply clarify applies all safe rules
that do not split the goal

apply safe applies all safe rules

Some Automation

More Proof Methods:

- apply** (intro <intro-rules>) repeatedly applies intro rules
- apply** (elim <elim-rules>) repeatedly applies elim rules
- apply** clarify applies all safe rules that do not split the goal
- apply** safe applies all safe rules
- apply** blast an automatic tableaux prover (works well on predicate logic)

Some Automation

More Proof Methods:

- apply** (intro <intro-rules>) repeatedly applies intro rules
- apply** (elim <elim-rules>) repeatedly applies elim rules
- apply** clarify applies all safe rules that do not split the goal
- apply** safe applies all safe rules
- apply** blast an automatic tableaux prover (works well on predicate logic)
- apply** fast another automatic search tactic

Epsilon and Automation Demo

We have learned so far...

→ Proof rules for predicate calculus

We have learned so far...

- Proof rules for predicate calculus
- Safe and unsafe rules

We have learned so far...

- Proof rules for predicate calculus
- Safe and unsafe rules
- Forward Proof

We have learned so far...

- Proof rules for predicate calculus
- Safe and unsafe rules
- Forward Proof
- The Epsilon Operator

We have learned so far...

- Proof rules for predicate calculus
- Safe and unsafe rules
- Forward Proof
- The Epsilon Operator
- Some automation

Isar (Part 1)

A Language for Structured Proofs

Motivation

Is this true: $(A \rightarrow B) = (B \vee \neg A)$?

Motivation

Is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

YES!

```
apply (rule iffI)
  apply (cases A)
    apply (rule disjI1)
      apply (erule impE)
        apply assumption
      apply assumption
    apply (rule disjI2)
      apply assumption
    apply (rule impI)
      apply (erule disjE)
        apply assumption
      apply (erule notE)
        apply assumption
  done
```

Motivation

Is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

YES!

```
apply (rule iffI)
  apply (cases A)
    apply (rule disjI1)
    apply (erule impE)
      apply assumption
      apply assumption
    apply (rule disjI2)
      apply assumption
    apply (rule impI)
      apply (erule disjE)
        apply assumption
      apply (erule notE)
        apply assumption
  done
```

or by blast

Motivation

Is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

YES!

```
apply (rule iffI)
  apply (cases A)
    apply (rule disjI1)
      apply (erule impE)
        apply assumption
      apply assumption
    apply (rule disjI2)
      apply assumption
    apply (rule impI)
      apply (erule disjE)
        apply assumption
      apply (erule notE)
        apply assumption
  done
```

or by blast

OK it's true. But WHY?

Motivation

WHY is this true: $(A \longrightarrow B) = (B \vee \neg A)$?

Demo

lsar

apply scripts

→ unreadable

Isar

apply scripts

- unreadable
- hard to maintain

Isar

apply scripts

- unreadable
- hard to maintain
- do not scale

Isar

apply scripts

- unreadable
- hard to maintain
- do not scale

No structure.

Isar

apply scripts

- unreadable
- hard to maintain
- do not scale

What about..

- Elegance?

No structure.

Isar

apply scripts

- unreadable
- hard to maintain
- do not scale

What about..

- Elegance?
- Explaining deeper insights?

No structure.

Isar

apply scripts

- unreadable
- hard to maintain
- do not scale

What about..

- Elegance?
- Explaining deeper insights?
- Large developments?

No structure.

Isar

apply scripts

- unreadable
- hard to maintain
- do not scale

No structure.

What about..

- Elegance?
- Explaining deeper insights?
- Large developments?

Isar!

A typical Isar proof

```
proof
  assume formula0
  have formula1 by simp
  ⋮
  have formulan by blast
  show formulan+1 by ...
qed
```

A typical Isar proof

```
proof  
  assume formula0  
  have formula1 by simp  
  :  
  have formulan by blast  
  show formulan+1 by ...  
qed
```

proves $\textit{formula}_0 \implies \textit{formula}_{n+1}$

A typical Isar proof

```
proof  
  assume formula0  
  have formula1 by simp  
   $\vdots$   
  have formulan by blast  
  show formulan+1 by ...  
qed
```

proves $\textit{formula}_0 \implies \textit{formula}_{n+1}$

(analogous to **assumes/shows** in lemma statements)

Isar core syntax

proof = **proof** [method] statement* **qed**
| **by** method

Isar core syntax

proof = **proof** [method] statement* **qed**
| **by** method

method = (simp ...) | (blast ...) | (rule ...) | ...

Isar core syntax

proof = **proof** [method] statement* **qed**
| **by** method

method = (simp ...) | (blast ...) | (rule ...) | ...

statement = **fix** variables (\wedge)
| **assume** proposition (\implies)
| [**from** name⁺] (**have** | **show**) proposition proof
| **next** (separates subgoals)

Isar core syntax

proof = **proof** [method] statement* **qed**
| **by** method

method = (simp ...) | (blast ...) | (rule ...) | ...

statement = **fix** variables (\wedge)
| **assume** proposition (\implies)
| [**from** name⁺] (**have** | **show**) proposition proof
| **next** (separates subgoals)

proposition = [name:] formula

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof (rule conjI)

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

next

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

next

assume B: "B"

from B **show** "B" **by** assumption

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

next

assume B: "B"

from B **show** "B" **by** assumption

qed

proof and qed

proof [method] statement* **qed**

lemma "[A; B] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

next

assume B: "B"

from B **show** "B" **by** assumption

qed

→ **proof** (<method>) applies method to the stated goal

proof and qed

proof [method] statement* **qed**

lemma "[[A; B]] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

next

assume B: "B"

from B **show** "B" **by** assumption

qed

- **proof** (<method>) applies method to the stated goal
- **proof** applies a single rule that fits

proof and qed

proof [method] statement* **qed**

lemma "[[A; B]] \implies A \wedge B"

proof (rule conjI)

assume A: "A"

from A **show** "A" **by** assumption

next

assume B: "B"

from B **show** "B" **by** assumption

qed

- **proof** (<method>) applies method to the stated goal
- **proof** applies a single rule that fits
- **proof** - does nothing to the goal

How do I know what to Assume and Show?

Look at the proof state!

lemma "[[A; B]] \implies A \wedge B"
proof (rule conjI)

How do I know what to Assume and Show?

Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ "

proof (rule conjI)

→ **proof** (rule conjI) changes proof state to

1. $\llbracket A; B \rrbracket \implies A$
2. $\llbracket A; B \rrbracket \implies B$

How do I know what to Assume and Show?

Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ "

proof (rule conjI)

→ **proof** (rule conjI) changes proof state to

1. $\llbracket A; B \rrbracket \implies A$
2. $\llbracket A; B \rrbracket \implies B$

→ so we need 2 shows: **show** " A " and **show** " B "

How do I know what to Assume and Show?

Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ "

proof (rule conjI)

- **proof** (rule conjI) changes proof state to
 1. $\llbracket A; B \rrbracket \implies A$
 2. $\llbracket A; B \rrbracket \implies B$
- so we need 2 shows: **show** " A " and **show** " B "
- We are allowed to **assume** A ,
because A is in the assumptions of the proof state.

The Three Modes of Isar

→ `[prove]`:

goal has been stated, proof needs to follow.

The Three Modes of Isar

→ **[prove]**:

goal has been stated, proof needs to follow.

→ **[state]**:

proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ "

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ " **[prove]**

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ " **[prove]**

proof (rule conj) **[state]**

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

lemma " $\llbracket A; B \rrbracket \implies A \wedge B$ " **[prove]**

proof (rule conjI) **[state]**

assume A: "A" **[state]**

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

lemma "[A; B] \implies A \wedge B" **[prove]**

proof (rule conjI) **[state]**

 assume A: "A" **[state]**

 from A **[chain]**

The Three Modes of Isar

- **[prove]**:
goal has been stated, proof needs to follow.
- **[state]**:
proof block has opened or subgoal has been proved,
new *from* statement, goal statement or assumptions can follow.
- **[chain]**:
from statement has been made, goal statement needs to follow.

```
lemma "[A; B]  $\implies$  A  $\wedge$  B" [prove]
proof (rule conjI) [state]
  assume A: "A" [state]
  from A [chain] show "A" [prove] by assumption [state]
next [state] ...
```

Have

Can be used to make intermediate steps.

Example:

Have

Can be used to make intermediate steps.

Example:

lemma "(x :: nat) + 1 = 1 + x"

Have

Can be used to make intermediate steps.

Example:

```
lemma "(x :: nat) + 1 = 1 + x"  
proof -  
  have A: "x + 1 = Suc x" by simp  
  have B: "1 + x = Suc x" by simp  
  show "x + 1 = 1 + x" by (simp only: A B)  
qed
```

Demo

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " proof

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " **proof**

→ **proof** picks an **intro** rule automatically

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " **proof**

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "..." **proof**

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " **proof**

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "... " **proof**

- now **proof** picks an **elim** rule automatically

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " **proof**

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "... " **proof**

- now **proof** picks an **elim** rule automatically
- triggered by **from**

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " proof

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "..." **proof**

- now **proof** picks an **elim** rule automatically
- triggered by **from**
- first assumption of rule must unify with AB

Backward and Forward

Backward reasoning: ... have " $A \wedge B$ " **proof**

- **proof** picks an **intro** rule automatically
- conclusion of rule must unify with $A \wedge B$

Forward reasoning: ...

assume AB: " $A \wedge B$ "

from AB **have** "... " **proof**

- now **proof** picks an **elim** rule automatically
- triggered by **from**
- first assumption of rule must unify with AB

General case: **from** $A_1 \dots A_n$ **have** R **proof**

- first n assumptions of rule must unify with $A_1 \dots A_n$
- conclusion of rule must unify with R

Fix and Obtain

fix $v_1 \dots v_n$

Fix and Obtain

fix $v_1 \dots v_n$

Introduces new arbitrary but fixed variables
(\sim parameters, \wedge)

Fix and Obtain

fix $v_1 \dots v_n$

Introduces new arbitrary but fixed variables
(\sim parameters, \wedge)

obtain $v_1 \dots v_n$ **where** $\langle \text{prop} \rangle$ $\langle \text{proof} \rangle$

Fix and Obtain

fix $v_1 \dots v_n$

Introduces new arbitrary but fixed variables
(\sim parameters, \wedge)

obtain $v_1 \dots v_n$ **where** $\langle \text{prop} \rangle$ $\langle \text{proof} \rangle$

Introduces new variables together with property

Demo

Fancy Abbreviations

this = the previous fact proved or assumed

Fancy Abbreviations

this = the previous fact proved or assumed

then = **from** this

Fancy Abbreviations

this = the previous fact proved or assumed

then = **from** this

thus = **then show**

Fancy Abbreviations

this = the previous fact proved or assumed

then = **from** this

thus = **then show**

hence = **then have**

Fancy Abbreviations

this = the previous fact proved or assumed

then = **from** this

thus = **then show**

hence = **then have**

with $A_1 \dots A_n$ = **from** $A_1 \dots A_n$ **this**

Fancy Abbreviations

this	=	the previous fact proved or assumed
then	=	from this
thus	=	then show
hence	=	then have
with $A_1 \dots A_n$	=	from $A_1 \dots A_n$ this
?thesis	=	the last enclosing goal statement

Moreover and Ultimately

have $X_1: P_1 \dots$

have $X_2: P_2 \dots$

\vdots

have $X_n: P_n \dots$

from $X_1 \dots X_n$ **show** \dots

Moreover and Ultimately

have X_1 : P_1 ...

have X_2 : P_2 ...

⋮

have X_n : P_n ...

from $X_1 \dots X_n$ **show** ...

wastes lots of brain power
on names $X_1 \dots X_n$

Moreover and Ultimately

have X_1 : $P_1 \dots$

have X_2 : $P_2 \dots$

\vdots

have X_n : $P_n \dots$

from $X_1 \dots X_n$ show \dots

have $P_1 \dots$

moreover have $P_2 \dots$

\vdots

moreover have $P_n \dots$

ultimately show \dots

wastes lots of brain power
on names $X_1 \dots X_n$

General Case Distinctions

show *formula*
proof -

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

ultimately show ?thesis **by** blast

qed

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

ultimately show ?thesis **by** blast

qed

{ ... } is a proof block similar to **proof** ... **qed**

General Case Distinctions

show *formula*

proof -

have $P_1 \vee P_2 \vee P_3$ <proof>

moreover { **assume** P_1 ... **have** ?thesis <proof> }

moreover { **assume** P_2 ... **have** ?thesis <proof> }

moreover { **assume** P_3 ... **have** ?thesis <proof> }

ultimately show ?thesis **by** blast

qed

{ ... } is a proof block similar to **proof** ... **qed**

{ **assume** P_1 ... **have** P <proof> }

stands for $P_1 \implies P$

Mixing proof styles

```
from ...  
have ...  
  apply -      make incoming facts assumptions  
  apply (...)  
  ⋮  
  apply (...)  
done
```

We have learned so far...

→ Isar style proofs

We have learned so far...

- Isar style proofs
- proof, qed

We have learned so far...

- Isar style proofs
- proof, qed
- assumes, shows

We have learned so far...

- Isar style proofs
- proof, qed
- assumes, shows
- fix, obtain

We have learned so far...

- Isar style proofs
- proof, qed
- assumes, shows
- fix, obtain
- moreover, ultimately

We have learned so far...

- Isar style proofs
- proof, qed
- assumes, shows
- fix, obtain
- moreover, ultimately
- forward, backward

We have learned so far...

- Isar style proofs
- proof, qed
- assumes, shows
- fix, obtain
- moreover, ultimately
- forward, backward
- mixing proof styles