COMP4161 Advanced Topics in Software Verification



HOL

Thomas Sewell, Miki Tanaka, Rob Sison T3/2024



Last time...

- → natural deduction rules for \land , \lor , \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 	[4,5]
	 Datatype induction, primitive recursion 	[5,7]
	 General recursive functions, termination proofs 	[7]
	 Proof automation, Isar (part 2) 	[8 ^b]
	 Hoare logic, proofs about programs, invariants 	[8,9]
	C verification	[9,10]
	 Practice, questions, exam prep 	[10 ^c]

^aa1 due; ^ba2 due; ^ca3 due

QUANTIFIERS



- Scope of parameters: whole subgoal
- Scope of \forall, \exists, \ldots ends with ; or \Longrightarrow

Example:





- Scope of parameters: whole subgoal
- Scope of \forall, \exists, \ldots : ends with ; or \Longrightarrow

Example:

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

means

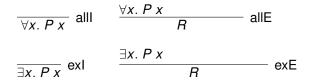




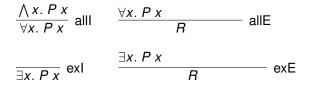
- Scope of parameters: whole subgoal
- Scope of \forall, \exists, \ldots : ends with ; or \Longrightarrow

Example:

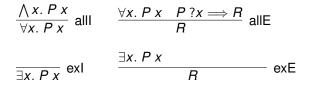




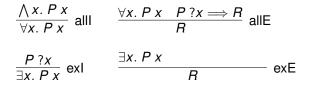




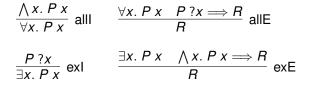














$$\frac{\bigwedge x. P x}{\forall x. P x} \text{ all} \qquad \frac{\forall x. P x \quad P?x \Longrightarrow R}{R} \text{ allE}$$
$$\frac{P?x}{\exists x. P x} \text{ exl} \qquad \frac{\exists x. P x \quad \bigwedge x. P x \Longrightarrow R}{R} \text{ exE}$$

- **allI** and **exE** introduce new parameters $(\land x)$.
- **allE** and **exl** 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



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



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

apply (rule allI)

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



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 exl)

1. $\bigwedge x. x = x$



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

apply (rule all)

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

best practice

apply (rule_tac $x = x^{*}$ in exl)

1.
$$\bigwedge x \cdot x = x$$

apply (rule refl)



1. $\forall x. \exists y. x = y$ apply (rule alll)1. $\bigwedge x. \exists y. x = y$ best practiceapply (rule_tac x = "x" in exl)1. $\bigwedge x. x = x$ 1. $\bigwedge x. x = ?y x$ apply (rule refl)



1. $\forall x. \exists y. x = y$ apply (rule alll)1. $\bigwedge x. \exists y. x = y$ best practiceapply (rule_tac x = "x" in exl)1. $\bigwedge x. x = x$ 1. $\bigwedge x. x = x$ 1. $\bigwedge x. x = ?y x$ apply (rule refl)?y $\mapsto \lambda u.u$



1. $\forall x. \exists y. x = y$ apply (rule all) 1. $\bigwedge x$. $\exists y$. x = ybest practice exploration **apply** (rule_tac $x = x^{*}$ in exl) apply (rule exl) 1. $\bigwedge x \cdot x = x$ 1. $\bigwedge x$. x = ?y xapply (rule refl) apply (rule refl) $?y \mapsto \lambda u.u$

simpler & clearer

shorter & trickier



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



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

apply (rule_tac x = ??? in exl)



1. $\exists y. \forall x. x = y$ apply (rule_tac x = ??? in exl)apply (rule exl)1. $\forall x. x = ?y$

ਿ ਜਤ

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

apply (rule_tac x = ??? in exl)

apply (rule exl) 1. $\forall x. x = ?y$ apply (rule alll) 1. $\bigwedge x. x = ?y$



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

apply (rule_tac x = ??? in exl)

apply (rule exl) 1. $\forall x. x = ?y$ apply (rule alll) 1. $\bigwedge x. x = ?y$ apply (rule refl) $?y \mapsto x$ yields $\bigwedge x'. x' = x$



1. $\exists y. \forall x. x = y$ apply (rule_tac x = ??? in exl) 1. $\forall x. x = ?y$ apply (rule alll) 1. $\land x. x = ?y$ apply (rule alll) 1. $\land x. x = ?y$ apply (rule refl) ? $y \mapsto x$ yields $\land x'. x' = x$

Principle:

?f $x_1 \dots x_n$ can only be replaced by term t if $params(t) \subseteq x_1, \dots, x_n$



Safe and Unsafe Rules



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.
$$\bigwedge \mathbf{x}$$
. $\exists \mathbf{y}$. $\mathbf{x} = \mathbf{y}$

Parameter names

Parameter names are chosen by Isabelle

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

apply (rule allI)

1.
$$\bigwedge \mathbf{x}$$
. $\exists y$. $x = y$

apply (rule_tac x = "x" in exl)

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

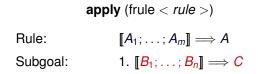
apply (rename_tac N)

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

apply (rule_tac x = "N" in exl)

In general: (rename_tac $x_1 \dots x_n$) renames the rightmost (inner) *n* parameters to $x_1 \dots x_n$







apply (frule < <i>rule</i> >)		
Rule:	$\llbracket A_1;\ldots;A_m \rrbracket \Longrightarrow A$	
Subgoal:	1. $\llbracket B_1; \ldots; B_n \rrbracket \Longrightarrow C$	
Substitution:	$\sigma(\mathbf{B}_i) \equiv \sigma(\mathbf{A}_1)$	



14 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

apply (frule < *rule* >) Rule: $\llbracket A_1; \ldots; A_m \rrbracket \Longrightarrow A$ 1. $[B_1; \ldots; B_n] \Longrightarrow C$ Subgoal: Substitution: $\sigma(B_i) \equiv \sigma(A_1)$ New subgoals: 1. $\sigma(\llbracket B_1; \ldots; B_n \rrbracket \Longrightarrow A_2)$ ŝ m-1. $\sigma(\llbracket B_1; \ldots; B_n \rrbracket \Longrightarrow A_m)$ m. $\sigma(\llbracket B_1; \ldots; B_n; A \rrbracket \Longrightarrow C)$



apply (frule < *rule* >) Rule: $\llbracket A_1; \ldots; A_m \rrbracket \Longrightarrow A$ 1. $[B_1; \ldots; B_n] \Longrightarrow C$ Subgoal: Substitution: $\sigma(B_i) \equiv \sigma(A_1)$ New subgoals: 1. $\sigma(\llbracket B_1; \ldots; B_n \rrbracket \Longrightarrow A_2)$ ÷ m-1. $\sigma(\llbracket B_1; \ldots; B_n \rrbracket \Longrightarrow A_m)$ m. $\sigma(\llbracket B_1; \ldots; B_n; A \rrbracket \Longrightarrow C)$

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



Examples for Forward Rules

$$\frac{P \land Q}{P} \text{ conjunct1} \qquad \frac{P \land Q}{Q} \text{ conjunct2}$$
$$\frac{P \longrightarrow Q}{Q} P \text{ mp}$$
$$\frac{\forall x. P x}{P?x} \text{ spec}$$



15 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

Forward Proof: OF

r [**OF** *r*₁ . . . *r_n*]

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



16 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

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

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

Rule r	$\llbracket A_1; \ldots; A_m \rrbracket \Longrightarrow A$
Rule r ₁	$\llbracket B_1;\ldots;B_n \rrbracket \Longrightarrow B$



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

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

Rule r	$[\![A_1;\ldots;A_m]\!]$	$\Longrightarrow A$
		_

Rule r_1 $\llbracket B_1; \ldots; B_n \rrbracket \Longrightarrow B$

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



$$r$$
 [**OF** $r_1 \ldots r_n$]

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

Rule r	$\llbracket A_1;\ldots;A_m \rrbracket \Longrightarrow A$
Rule r ₁	$\llbracket B_1;\ldots;B_n\rrbracket \Longrightarrow B$
Substitution	$\sigma(B) \equiv \sigma(A_1)$
r [OF r ₁]	$\sigma(\llbracket B_1;\ldots;B_n;A_2;\ldots;A_m\rrbracket \Longrightarrow A)$



Forward Proof: OF

r [**OF** *r*₁ . . . *r_n*]

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

Rule r	$\llbracket A_1;\ldots;A_m \rrbracket \Longrightarrow A$
Rule r ₁	$\llbracket B_1;\ldots;B_n rbracket \Longrightarrow B$
Substitution	$\sigma(B) \equiv \sigma(A_1)$
<i>r</i> [OF <i>r</i> ₁]	$\sigma(\llbracket B_1;\ldots;B_n;A_2;\ldots;A_m\rrbracket \Longrightarrow A)$

Example:

 $dvd_add : [?a dvd ?b;?a dvd ?c] \implies ?a dvd ?b + ?c dvd_refl : ?a dvd ?a$

 $dvd_add[OF dvd_refl]: [?a dvd?c] \implies ?a dvd?a+?c$



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)

ε x. Px is a value that satisfies P (if such a value exists)



19 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

Hilbert's Epsilon Operator



(David Hilbert, 1862-1943)

 ε x. Px is a value that satisfies P (if such a value exists)

 ε also known as **description operator**. In Isabelle the ε -operator is written SOME *x*. *P x*



Hilbert's Epsilon Operator



(David Hilbert, 1862-1943)

ε x. Px is a value that satisfies P (if such a value exists)

ε also known as **description operator**. In Isabelle the ε -operator is written SOME *x*. *P x*

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



More Epsilon

 ε implies Axiom of Choice: $\forall x. \exists y. Q x y \Longrightarrow \exists f. \forall x. Q x (f x)$

Existential and universal quantification can be defined with ε .



More Epsilon

 ε implies Axiom of Choice: $\forall x. \exists y. Q x y \Longrightarrow \exists f. \forall x. Q x (f x)$

Existential and universal quantification can be defined with ε .

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

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



More Proof Methods:

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



More Proof Methods:

apply (intro <intro-rules>)
apply (elim <elim-rules>)
apply clarify

repeatedly applies intro rules repeatedly applies elim rules applies all safe rules that do not split the goal



More Proof Methods:

```
apply (intro <intro-rules>)
apply (elim <elim-rules>)
apply clarify
```

apply safe

repeatedly applies intro rules repeatedly applies elim rules applies all safe rules that do not split the goal applies all safe rules



More Proof Methods:

```
apply (intro <intro-rules>)
apply (elim <elim-rules>)
apply clarify
apply safe
```

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



More Proof Methods:

apply (intro <intro-rules>)
apply (elim <elim-rules>)
apply clarify
apply safe
apply blast
apply fast

repeatedly applies intro rules repeatedly applies elim rules applies all safe rules that do not split the goal applies all safe rules an automatic tableaux prover (works well on predicate logic) another automatic search tactic



EPSILON AND AUTOMATION DEMO

➔ Proof rules for predicate calculus



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



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



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

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



ISAR (PART 1) A LANGUAGE FOR STRUCTURED PROOFS

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



25 | COMP4161| T Sewell, M Tanaka, R Sison CC-BY-4.0 License

Is this true: $(A \longrightarrow B) = (B \lor \neg A)$? YES!

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



```
Is this true: (A \longrightarrow B) = (B \lor \neg A)?
YES!
```

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

Of by blast



```
Is this true: (A \longrightarrow B) = (B \lor \neg A)?
YES!
```

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

Of by blast

OK it's true. But WHY?



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

Demo



27 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

apply scripts

→ hard to read



- → hard to read
- → hard to maintain



- → hard to read
- → hard to maintain

No explicit structure.

- → hard to read
- → hard to maintain

→ Elegance?

No explicit structure.



What about..

- hard to read
- → hard to maintain
- → Elegance?
- → Explaining deeper insights?

No explicit structure.





What about..

- hard to read
- → hard to maintain

- → Elegance?
- → Explaining deeper insights?

No explicit structure.

Isar!

A typical Isar proof

proof assume formula₀ have formula₁ by simp : have formula_n by blast show formula_{n+1} by ... qed

A typical Isar proof

proof assume formula₀ have formula₁ by simp : have formula_n by blast show formula_{n+1} by ... qed

proves formula₀ \implies formula_{n+1}



A typical Isar proof

proof assume formula₀ have formula₁ by simp : have formula_n by blast show formula_{n+1} by ... qed

proves $formula_0 \implies formula_{n+1}$

(analogous to assumes/shows in lemma statements)



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



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

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



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

```
method = (simp \dots) | (blast \dots) | (rule \dots) | \dots
```

```
statement = fix variables (∧)

| assume proposition (⇒)

| [from name<sup>+</sup>] (have | show) proposition proof

| next (separates subgoals)
```



```
proof = proof [method] statement* qed

| by method

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

statement = fix variables (∧)

| assume proposition (⇒)

| [from name<sup>+</sup>] (have | show) proposition proof

| next (separates subgoals)
```

proposition = [name:] formula

proof and qed

proof [method] statement* qed

 $\textbf{lemma "} \llbracket A; B \rrbracket \Longrightarrow A \land B"$



proof and qed

proof [method] statement* qed

lemma " $\llbracket A; B \rrbracket \implies A \land B$ " proof (rule conjl)



```
lemma "[[A; B]] ⇒ A ∧ B"
proof (rule conjl)
assume A: "A"
from A show "A" by assumption
```



```
lemma "[[A; B]] ⇒ A ∧ B"
proof (rule conjl)
    assume A: "A"
    from A show "A" by assumption
next
```



```
lemma "[[A; B]] ⇒ A ∧ B"
proof (rule conjl)
    assume A: "A"
    from A show "A" by assumption
next
    assume B: "B"
    from B show "B" by assumption
```



```
lemma "\llbracket A; B \rrbracket \implies A \land B"

proof (rule conjl)

assume A: "A"

from A show "A" by assumption

next

assume B: "B"

from B show "B" by assumption

qed
```



```
lemma "[\![A; B]\!] \implies A \land B"
proof (rule conjl)
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



```
lemma "[A; B] \implies A \land B"

proof (rule conjl)

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



```
lemma "[A; B] \implies A \land B"

proof (rule conjl)

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



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \land B$ " proof (rule conjl)



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \land B$ " proof (rule conjl)

→ proof (rule conjl) changes proof state to

1.
$$\llbracket A; B \rrbracket \Longrightarrow A$$

2. $\llbracket A; B \rrbracket \Longrightarrow B$



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \land B$ " proof (rule conjl)

- → proof (rule conjl) changes proof state to
 - $1.\,\llbracket A;B\rrbracket \Longrightarrow A$
 - $2.\,\llbracket A;B\rrbracket \Longrightarrow B$
- → so we need 2 shows: **show** "A" and **show** "B"



Look at the proof state!

lemma " $\llbracket A; B \rrbracket \implies A \land B$ " proof (rule conjl)

- → proof (rule conjl) changes proof state to
 - $1. \llbracket A; B \rrbracket \Longrightarrow A$
 - $\textbf{2.} \ \llbracket \textbf{A}; \textbf{B} \rrbracket \Longrightarrow \textbf{B}$
- → 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.



→ [prove]:

goal has been stated, proof needs to follow.



33 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

→ [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.

→ [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]:



→ [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 " $\llbracket A; B \rrbracket \Longrightarrow A \land B$ "



→ [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 " $\llbracket A; B \rrbracket \implies A \land B$ " [prove]



→ [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 " $[A; B] \implies A \land B$ " [prove] proof (rule conji) [state]



→ [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]:

```
lemma "[A; B] \implies A \land B" [prove]
proof (rule conjl) [state]
assume A: "A" [state]
```



→ [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]:

```
lemma "[A; B] \implies A \land B" [prove]
proof (rule conjl) [state]
   assume A: "A" [state]
   from A [chain]
```





→ [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]:

```
lemma "[A; B]] ⇒ A ∧ B" [prove]
proof (rule conjl) [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)

ged
```





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



36 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

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

→ proof picks an intro rule automatically



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

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



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

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

Forward reasoning:

assume AB: " $A \land B$ " from AB have "..." proof



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

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

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof

→ now proof picks an elim rule automatically



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

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

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof

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



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

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

Forward reasoning: ...

assume AB: " $A \land B$ " from AB have "..." proof

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



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

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

Forward reasoning: ...

assume AB: " $A \land B$ " 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 $A_1 \ldots A_n$ have R proof

- → first *n* assumptions of rule must unify with $A_1 \ldots A_n$
- → conclusion of rule must unify with R



fix *v*₁ . . . *v*_n



SW

fix *v*₁ . . . *v*_n

Introduces new arbitrary but fixed variables $(\sim \text{ parameters}, \Lambda)$



fix *v*₁ . . . *v*_n

Introduces new arbitrary but fixed variables $(\sim \text{ parameters}, \Lambda)$

obtain $v_1 \dots v_n$ where < prop > < proof >



37 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

fix *v*₁ . . . *v*_n

Introduces new arbitrary but fixed variables $(\sim \text{ parameters}, \Lambda)$

obtain $v_1 \dots v_n$ where <prop> <proof>

Introduces new variables together with property



this = the previous fact proved or assumed



this = the previous fact proved or assumed

then = from this



this	=	the previous fact proved or assumed		

then = from this thus = then show



this	=	the previous fact proved or assumed
------	---	-------------------------------------

then	=	from this
thus	=	then show
hence	=	then have

38 | COMP4161 | T Sewell, M Tanaka, R Sison CC-BY-4.0 License

this	=	the previous fact proved or assumed
thus hence	=	from this then show then have from $A_1 \dots A_n$ this



this	=	the previous fact proved or assumed
thus hence	=	from this then show then have 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
:
have X_n: P_n \dots
from X_1 \dots X_n show \dots
```



Moreover and Ultimately

```
have X_1: P_1 \dots
have X_2: P_2 \dots
:
have X_n: P_n \dots
from X_1 \dots X_n show \dots
```

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



Moreover and Ultimately

```
have X<sub>1</sub>: P<sub>1</sub>...
have X<sub>2</sub>: P<sub>2</sub>...
:
have X<sub>n</sub>: P<sub>n</sub>...
from X<sub>1</sub>...X<sub>n</sub> show ....
```

wastes lots of brain power on names $X_1 \dots X_n$ have $P_1 \dots$ moreover have $P_2 \dots$ moreover have $P_n \dots$ ultimately show ...



40 | COMP4161| T Sewell, M Tanaka, R Sison CC-BY-4.0 License

show formula proof -



```
show formula proof - have P_1 \lor P_2 \lor P_3 <proof>
```



```
\begin{array}{l} \text{show formula} \\ \text{proof} \\ \text{have } P_1 \lor P_2 \lor P_3 \ < \text{proof} \\ \text{moreover} \quad \{ \text{ assume } P_1 \ \dots \ \text{have } ? \text{thesis } < \text{proof} > \} \end{array}
```



```
show formula

proof -

have P_1 \lor P_2 \lor P_3 <proof>

moreover { assume P_1 \ldots have ?thesis <proof> }

moreover { assume P_2 \ldots have ?thesis <proof> }
```

```
show formula

proof -

have P_1 \lor P_2 \lor P_3 <proof>

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

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

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



```
\begin{array}{l} \mbox{show formula} \\ \mbox{proof} - \\ \mbox{have } P_1 \lor P_2 \lor P_3 \ < \mbox{proof} > \\ \mbox{moreover} \quad \{ \mbox{ assume } P_1 \ \dots \ \mbox{have ?thesis } < \mbox{proof} > \} \\ \mbox{moreover} \quad \{ \mbox{ assume } P_2 \ \dots \ \mbox{have ?thesis } < \mbox{proof} > \} \\ \mbox{moreover} \quad \{ \mbox{ assume } P_3 \ \dots \ \mbox{have ?thesis } < \mbox{proof} > \} \\ \mbox{ultimately show ?thesis by blast} \\ \mbox{qed} \end{array}
```



```
show formula

proof -

have P_1 \lor P_2 \lor 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
```

```
icense
```



```
\begin{array}{l} \mbox{show formula} \\ \mbox{proof} - \\ \mbox{have } P_1 \lor P_2 \lor P_3 \ < \mbox{proof} > \\ \mbox{moreover} \ \left\{ \mbox{ assume } P_1 \ \dots \ \mbox{have ?thesis } < \mbox{proof} > \right\} \\ \mbox{moreover} \ \left\{ \mbox{ assume } P_2 \ \dots \ \mbox{have ?thesis } < \mbox{proof} > \right\} \\ \mbox{moreover} \ \left\{ \mbox{ assume } P_3 \ \dots \ \mbox{have ?thesis } < \mbox{proof} > \right\} \\ \mbox{ultimately show ?thesis by blast} \\ \mbox{qed} \\ \left\{ \ \dots \right\} \ \mbox{is a proof block similar to proof} \ \dots \ \mbox{qed} \end{array}
```

```
\{ \text{ assume } P_1 \dots \text{ have } P < proof > \} \\ \text{ stands for } P_1 \Longrightarrow P
```



Mixing proof styles

```
from ...

have ...

apply - make incoming facts assumptions

apply (...)

:

apply (...)

done
```



→ Isar style proofs



- → Isar style proofs
- ➔ proof, qed



43 | COMP4161| T Sewell, M Tanaka, R Sison CC-BY-4.0 License

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



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



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



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



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

