COMP4161
Advanced Topics in Software Verification

HOL

Thomas Sewell, Miki Tanaka, Rob Sison

T3/2024

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 [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]]

---

[a] a1 due; [b] a2 due; [c] a3 due

QUANTIFIERS

Scope

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

Example:

$$\bigwedge x\ y.\ [\![\ \forall y.\ P\ y \longrightarrow Q\ z\ y;\ \ Q\ x\ y\ ]\!] \Longrightarrow \exists x.\ Q\ x\ y$$

means

$$\bigwedge x\ y.\ [\![\ (\forall y_1.\ P\ y_1 \longrightarrow Q\ z\ y_1);\ \ Q\ x\ y\ ]\!] \Longrightarrow (\exists x_1.\ Q\ x_1\ y)$$

Natural deduction for quantifiers

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\ \text{allI} \qquad \frac{\forall x.\ P\ x \quad P\ ?x \Longrightarrow R}{R}\ \text{allE}$$

$$\frac{P\ ?x}{\exists x.\ P\ x}\ \text{exI} \qquad \frac{\exists x.\ P\ x \quad \bigwedge x.\ P\ x \Longrightarrow R}{R}\ \text{exE}$$

- **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. $\forall x.\ \exists y.\ x = y$

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

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

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:

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

if $params(t) \subseteq x_1, \ldots, x_n$

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

apply (rule allI)

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

apply (rule_tac x = "x" in exI)

Brittle!

Renaming parameters

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

apply (rule allI)

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

apply (rename_tac N)

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

apply (rule_tac x = "N" in exI)

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

Forward Proof: frule and drule

apply (frule $<$ *rule* $>$)

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

Subgoal: 1. $[\![B_1; \ldots; B_n]\!] \Longrightarrow C$

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

New subgoals: 1. $\sigma([\![B_1; \ldots; B_n]\!] \Longrightarrow A_2)$

$\vdots$

m-1. $\sigma([\![B_1; \ldots; B_n]\!] \Longrightarrow A_m)$

m. $\sigma([\![B_1; \ldots; B_n; A]\!] \Longrightarrow C)$

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

Examples for Forward Rules

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

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

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

Forward Proof: OF

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

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

Rule r	$[\![A_1; \dots; A_m]\!] \Longrightarrow A$
Rule r_1	$[\![B_1; \dots; B_n]\!] \Longrightarrow B$
Substitution	$\sigma(B) \equiv \sigma(A_1)$
r [OF r_1]	$\sigma([\![B_1; \dots; B_n; A_2; \dots; A_m]\!] \Longrightarrow A)$

Example:

dvd_add : $[\![\ ?a\ dvd\ ?b; ?a\ dvd\ ?c\ ]\!] \Longrightarrow ?a\ dvd\ ?b + ?c$

dvd_refl : $?a\ dvd\ ?a$

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)

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

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

More Epsilon

ε implies Axiom of Choice:

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

Existential and universal quantification can be defined with ε.

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

$$\frac{}{(\text{THE}\ x.\ x = a) = a}\ \text{the\_eq\_trivial}$$

Some Automation

More Proof Methods:

apply (intro <intro-rules>)	repeatedly applies intro rules
apply (elim <elim-rules>)	repeatedly applies elim rules
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
- ➜ Safe and unsafe rules
- ➜ Forward Proof
- ➜ The Epsilon Operator
- ➜ Some automation

Isar (Part 1)

A Language for Structured Proofs

Motivation

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

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
```

or `by blast`

OK it's true. But WHY?

Motivation

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

Demo

Isar

apply scripts

- ➜ hard to read
- ➜ hard to maintain

No explicit structure.

What about..

- ➜ Elegance?
- ➜ Explaining deeper insights?

Isar!

A typical Isar proof

proof
  assume $formula_0$
  have $formula_1$ **by** simp
  ⋮
  have $formula_n$ **by** blast
  show $formula_{n+1}$ **by** . . .
qed

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

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

Isar core syntax

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

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

statement = **fix** variables ($\bigwedge$)
  | **assume** proposition ($\Longrightarrow$)
  | [**from** name$^+$] (**have** | **show**) proposition proof
  | **next** (separates subgoals)

proposition = [name:] formula

proof and qed

proof [method] statement* **qed**

lemma "$[\![A; B]\!] \Longrightarrow A \wedge B$"
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 "$\llbracket A; B \rrbracket \Longrightarrow A \wedge B$"
proof (rule conjI)

- **proof** (rule conjI) 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"
- 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.
- **[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 "$[\![A; B]\!] \Longrightarrow A \wedge B$" **[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:

lemma "$(x :: \text{nat}) + 1 = 1 + x$"
proof -
  have A: "$x + 1 = \text{Suc } x$" **by** simp
  have B: "$1 + x = \text{Suc } x$" **by** simp
  show "$x + 1 = 1 + x$" **by** (simp only: A B)
qed

Demo

Backward and Forward

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

- **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 $A_1 \ldots A_n$ **have** R **proof**

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

Fix and Obtain

fix $v_1 \dots v_n$

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

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

Introduces new variables together with property

Fancy Abbreviations

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

Demo

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 $X_1 \dots X_n$

have P_1 **...**
moreover have P_2 **...**
⋮
moreover have P_n **...**
ultimately show ...

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 $P_1 \Longrightarrow P$

Mixing proof styles

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


We have learned so far...

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

