COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

Isar
Rough timeline

Intro & motivation, getting started

Foundations & Principles
- Lambda Calculus, natural deduction
- Higher Order Logic
- Term rewriting

Proof & Specification Techniques
- Isar
- Inductively defined sets, rule induction
- Datatypes, recursion, induction
- Calculational reasoning, mathematics style proofs
- Hoare logic, proofs about programs

\(^a\) a1 out; \(^b\) a1 due; \(^c\) a2 out; \(^d\) a2 due; \(^e\) session break; \(^f\) a3 out; \(^g\) a3 due
ISAR

A LANGUAGE FOR STRUCTURED PROOFS
<table>
<thead>
<tr>
<th>apply scripts</th>
<th>What about..</th>
</tr>
</thead>
<tbody>
<tr>
<td>➔ unreadable</td>
<td>➔ Elegance?</td>
</tr>
<tr>
<td>➔ hard to maintain</td>
<td>➔ Explaining deeper insights?</td>
</tr>
<tr>
<td>➔ do not scale</td>
<td>➔ Large developments?</td>
</tr>
</tbody>
</table>

No structure.  
Isar!
A typical Isar proof

proof

assume \( formula_0 \)

have \( formula_1 \) by simp

\[ \vdots \]

have \( formula_n \) by blast

show \( formula_{n+1} \) by \ldots

qed

proves \( formula_0 \implies formula_{n+1} \)

(analogous to assumes/shows in lemma statements)
Isar core syntax

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

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

statement = fix variables (∧)
   | assume proposition (⇒)
   | [from name+] (have | show) proposition proof
   | next (separates subgoals)

proposition = [name:] formula
proof and qed

proof [method] statement* qed

lemma "[A; B] ⇒ A ∧ 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** "$[A; B] \implies A \land B$"

**proof** (rule conjI)

→ **proof** (rule conjI) changes proof state to
   1. $[A; B] \implies A$
   2. $[A; B] \implies B$

→ 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.

→ [state]:
  proof block has openend 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]
  assume A: "A" [state]
next [state] . . .
Have

Can be used to make intermediate steps.

Example:

\[
\begin{align*}
\text{lemma } & (x :: \text{nat}) + 1 = 1 + x \\
\text{proof - } \\
\text{have A: } & x + 1 = \text{Suc } x \text{ by simp} \\
\text{have B: } & 1 + x = \text{Suc } x \text{ by simp} \\
\text{show } & x + 1 = 1 + x \text{ by (simp only: A B)} \\
\text{qed}
\end{align*}
\]
Backward and Forward

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: "$A \land B$"

→ 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**

\[ \text{fix } v_1 \ldots v_n \]

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

\[ \text{obtain } v_1 \ldots v_n \text{ where } \langle \text{prop} \rangle \langle \text{proof} \rangle \]

Introduces new variables together with property
DEMO
Fancy Abbreviations

this = the previous fact proved or assumed
then = from this
thus = then show
hence = then have
\textbf{with} \; A_1 \ldots A_n = \textbf{from} \; A_1 \ldots A_n \; \text{this}
\textbf{?thesis} = the last enclosing goal statement
Moreover and Ultimately

have \( X_1 : P_1 \ldots \)

have \( X_2 : P_2 \ldots \)

\vdots

have \( X_n : P_n \ldots \)

from \( X_1 \ldots X_n \) show \( \ldots \)

wastes lots of brain power

on names \( X_1 \ldots X_n \)
show \textit{formula}

\textbf{proof -}

\textbf{have} \( P_1 \lor P_2 \lor P_3 \) \textit{<proof>}

moreover \{ \textbf{assume} \( P_1 \) \ldots \textbf{have} \( ?\)thesis \textit{<proof>} \}

moreover \{ \textbf{assume} \( P_2 \) \ldots \textbf{have} \( ?\)thesis \textit{<proof>} \}

moreover \{ \textbf{assume} \( P_3 \) \ldots \textbf{have} \( ?\)thesis \textit{<proof>} \}

\textbf{ultimately show} \( ?\)thesis \textit{by blast}

\textbf{qed}

\{ \ldots \} \text{ is a proof block similar to} \textbf{proof} \ldots \textbf{qed}

\{ \textbf{assume} \( P_1 \) \ldots \textbf{have} \( P \) \textit{<proof>} \}

\text{stands for} \( P_1 \implies P \)
Mixing proof styles

from ... have ...

apply - make incoming facts assumptions
apply (...)

apply (...)
done