CS477 Formal Software Development Methods

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides based in part on previous lectures by Mahesh Vishwanathan, and by Gul Agha

February 6, 2014

Getting Started with Isabelle

- Choice
  - Use Isabelle on EWS
  - Install on your machine
  - Both
- On EWS
  - Assuming you are running an X client, log in to EWS:
    `ssh -Y <netid>@remlnx.ews.illinois.edu`
    - `-Y` used to forward X packets securely
  - To start Isabelle with jedit
    `/class/cs477/bin/isabelle jedit`
  - Older versions of Isabelle used emacs and ProofGeneral
  - Will assume `jedit` here

My First Theory File

File name: `my_theory.thy`
Contents:

```
theory my_theory
imports Main
begin

thm impI

lemma trivial: "A ⟶ A"
apply (rule impI)
apply assumption
done (* of lemma *)

thm trivial

end (* of theory file *)
```


Overview of Isabelle/HOL

- HOL = Higher-Order Logic
- HOL = Types + Lambda Calculus + Logic
- HOL has
  - datatypes
  - recursive functions
  - logical operators ($\wedge$, $\vee$, $\neg$, $\longrightarrow$, $\forall$, $\exists$, ...)
- Contains propositional logic, first-order logic
- HOL is very similar to a functional programming language
- Higher-order = functions are values, too!
- We'll start with propositional and first order logic

Formulae (first Approximation)

- Syntax (in decreasing priority):

$$
\begin{array}{llll}
\textit{form} & ::= & (\textit{form}) & \mid\ \textit{term} = \textit{term} \\
 & \mid & \neg\textit{form} & \mid\ \textit{form} \wedge \textit{form} \\
 & \mid & \textit{form} \vee \textit{form} & \mid\ \textit{form} \longrightarrow \textit{form} \\
 & \mid & \forall x.\ \textit{form} & \mid\ \exists x.\ \textit{form}
\end{array}
$$

and some others

- Scope of quantifiers: as far to the right as possible

Examples

- $\neg A \wedge B \vee C \equiv ((\neg A) \wedge B) \vee C$
- $A \wedge B = C \equiv A \wedge (B = C)$
- $\forall x.\ P\ x \wedge Q\ x \equiv \forall x.\ (P\ x \wedge Q\ x)$
- $\forall x. \exists y.\ P\ x\ y \wedge Q\ x \equiv \forall x. (\exists y.\ (P\ x\ y \wedge Q\ x))$

Proofs

General schema:

```
lemma name: "..."
apply (...)
:
done
```

First ... theorem statement
(...) are *proof methods*

Top-down Proofs

`sorry`

- "completes" any proof (by giving up, and accepting it)
- Suitable for top-down development of theories:
- Assume lemmas first, prove them later.

Only allowed for interactive proof!

Isabelle Syntax

- Distinct from HOL syntax
- Contains HOL syntax within it
- Also the same as HOL - need to not confuse them

Theory = Module

Syntax:

theory *MyTh*
imports $ImpTh_1$ ... $ImpTh_n$
begin
declarations, definitions, theorems, proofs, ...
end

- *MyTh*: name of theory being built. Must live in file *MyTh*.thy.
- $ImpTh_i$: name of *imported* theories. Importing is transitive.

Meta-logic: Basic Constructs

Implication: $\Longrightarrow$ (==>)
For separating premises and conclusion of theorems / rules

Equality: $\equiv$ (==)
For definitions

Universal Quantifier: $\bigwedge$ (!!)
Usually inserted and removed by Isabelle automatically

Do not use *inside* HOL formulae

Rule/Goal Notation

$$[\![A_1; \ldots ; A_n]\!] \Longrightarrow B$$

abbreviates

$$A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow B$$

and means the rule (or potential rule):

$$\frac{A_1; \ldots ; A_n}{B}$$

; $\approx$ "and"

Note: A theorem is a rule; a rule is a theorem.

The Proof/Goal State

1. $\bigwedge x_1 \dots x_m.\ [\mid A_1; \dots; A_n \mid] \Longrightarrow B$

$x_1 \dots x_m$	Local constants (fixed variables)
$A_1 \dots A_n$	Local assumptions
B	Actual (sub)goal

Proof Basics

- Isabelle uses *Natural Deduction* proofs
  - Uses (modified) *sequent* encoding
- Rule notation:

Rule	Sequent Encoding
$\dfrac{A_1 \dots A_n}{A}$	$[\![A_1, \dots, A_n]\!] \Longrightarrow A$
$\dfrac{A_1 \dots \ \begin{matrix} B \\ \vdots \\ A_i \end{matrix} \ \dots A_n}{A}$	$[\![A_1, \dots, B \Longrightarrow A_i, \dots, A_n]\!] \Longrightarrow A$

Natural Deduction

For each logical operator $\oplus$, have two kinds of rules:

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

$$\frac{?}{A \oplus B}$$

Elimination: What can I prove using $A \oplus B$?

$$\frac{\dots A \oplus B \dots}{?}$$

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 we know A_1 and we want to prove A
it suffices to prove $A_2 \dots A_n$

Natural Deduction for Propositional Logic

$$\frac{A \quad B}{A \land B}\ \texttt{conjI} \qquad \frac{A \land B \quad [\![A; B]\!] \Longrightarrow C}{C}\ \texttt{conjE}$$

$$\frac{A}{A \lor B} \quad \frac{B}{A \lor B}\ \texttt{disjI1/2} \qquad \frac{A \lor B \quad A \Longrightarrow C \quad B \Longrightarrow C}{c}\ \texttt{disjE}$$

$$\frac{A \Longrightarrow B}{A \longrightarrow B}\ \texttt{impI} \qquad \frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C}\ \texttt{impE}$$

$$\frac{A \Longrightarrow False}{\neg A}\ \texttt{notI} \qquad \frac{\neg A \quad A}{B}\ \texttt{notE}$$

Natural Deduction for Propositional Logic

$$\frac{A \Longrightarrow B \quad B \Longrightarrow A}{A = B}\ \texttt{iffI} \qquad \frac{A = B \quad A}{B}\ \texttt{iffD1}$$

$$\frac{A = B \quad B}{A}\ \texttt{iffD2}$$

More Rules

$$\frac{A \wedge B}{A}\ \textit{conjunct1} \qquad \frac{A \wedge B}{B}\ \textit{conjunct2}$$

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

Compare to elimination rules:

$$\frac{A \wedge B \quad [\![A; B]\!] \Longrightarrow C}{C}\ \textit{conjE} \qquad \frac{A \longrightarrow B \quad A \quad B \Longrightarrow C}{C}\ \textit{impE}$$

"Classical" Rules

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

- ccontr and classical are not derivable from the Natural Deduction rules.
- They make the logic *"classical"*, i.e. *"non-constructive* or *"non-intuitionistic"*.

Proof by Assumption

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

- Proof method: assumption
- Use:

  apply assumption

- Proves:

  $[\![A_1; \dots; A_n]\!] \Longrightarrow A$

  by unifying A with one of the A_i

Rule Application: The Rough Idea

Applying rule $[\![A_1; \dots; A_n]\!] \Longrightarrow A$ to subgoal C:

- Unify A and C
- Replace C with n new subgoals: $A'_1 \ \dots\ A'_n$

Backwards reduction, like in Prolog

Example: rule: $[\![?P; ?Q]\!] \Longrightarrow ?P \wedge ?Q$

subgoal: 1. $A \wedge B$

Result: 1. A2. B

Rule Application: More Complete Idea

Applying rule $[\![A_1; \dots; A_n]\!] \Longrightarrow A$ to subgoal C:

- Unify A and C with (meta)-substitution σ
- Specialize goal to $\sigma(C)$
- Replace C with n new subgoals: $\sigma(A_1)\ \dots\ \sigma(A_n)$

Note: schematic variables in C treated as existential variables

Does there exist value for $?X$ in C that makes C true?

(Still not the whole story)

rule Application

Rule: $[\![A_1; \dots; A_n]\!] \Longrightarrow A$

Subgoal: 1. $[\![B_1; \dots; B_m]\!] \Longrightarrow C$

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

New subgoals: 1. $[\![\sigma(B_1); \dots; \sigma(B_m)]\!] \Longrightarrow \sigma(A_1)$

⋮

n. $[\![\sigma(B_1); \dots; \sigma(B_m)]\!] \Longrightarrow \sigma(A_n)$

Proves: $[\![\sigma(B_1); \dots; \sigma(B_m)]\!] \Longrightarrow \sigma(C)$

Command: apply (rule <*rulename*>)

Applying Elimination Rules

`apply (erule <elim-rule>)`

Like `rule` but also

- Unifies first premise of rule with an assumption
- Eliminates that assumption instead of conclusion

Example

Rule: $[\![?P \wedge ?Q; [\![?P; ?Q]\!] \Longrightarrow ?R]\!] \Longrightarrow ?R$

Subgoal: 1. $[\![X; A \wedge B; Y]\!] \Longrightarrow Z$

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

New subgoal: 1. $[\![X; Y]\!] \Longrightarrow [\![A; B]\!] \Longrightarrow Z$

Same as: 1. $[\![X; Y; A; B]\!] \Longrightarrow Z$