Getting Started with Isabelle

- Install on your machine
  - Source is at http://www.cl.cam.ac.uk/research/hvg/Isabelle/
- Will (try to) put 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 on EWS (or other command line)
  /class/cs477/bin/isabelle jedit

Overview of Isabelle/HOL

- HOL = Higher-Order Logic
- HOL = Types + Lambda Calculus + Logic
- HOL has
  - datatypes
  - recursive functions
  - logical operators (∧, ∨, ¬, →, ∀, ∃, ...)
- 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):
  
  form ::= ( form)
  | term = term
  | ¬form
  | form ∨ form
  | ∀x. form
  | ∃x. form

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

Examples

- ¬A ∧ B ∨ C ≡ ((¬A) ∧ B) ∨ C
- A ∧ B = C ≡ A ∧ (B = C)
- ∀x. P x ∧ Q x ≡ ∀x. (P x ∧ Q x)
- ∀x, y. P x ∧ Q x ≡ ∀x, y. (∃y. (P x ∧ Q x))
Isabelle Syntax

- Distinct from HOL syntax
- Contains HOL syntax within it
- Layer above HOL, but effectively (a large fragment of) HOL
- Need to not confuse them

Theory = Module

Syntax:

```
theory MyTh
imports ImpTh1 ... ImpThn
begin
  declarations, definitions, theorems, proofs, ...
end
```

- `MyTh`: name of theory being built. Must live in file `MyTh.thy`.
- `ImpThi`: name of imported theories. Importing is transitive.

Meta-logic: Basic Constructs

Implication: \( \Rightarrow \) (\( \Rightarrow \Rightarrow \) )
For separating premises and conclusion of theorems / rules

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

Universal Quantifier: \( \forall \) (\( \forall \) )
Usually inserted and removed by Isabelle automatically

Do not use inside HOL formulae

Rule/Goal Notation

```
[| A_1; ...; A_n |] \Rightarrow B
```
abbreviates

```
A_1 \Rightarrow ... \Rightarrow A_n \Rightarrow B
```
and means the rule (or potential rule):

\[
\frac{A_1; ...; A_n}{B}
\]

\( \therefore \) “and”

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

The Proof/Goal State

1. \( \forall x_1 ... x_m. [A_1; ...; A_n] \Rightarrow B \)

- \( x_1 ... x_m \): Local constants (fixed variables)
- \( A_1 ... A_n \): Local assumptions
- \( B \): Actual (sub)goal

Not dequire after here (now?)
Isabelle uses Natural Deduction proofs. It uses (modified) sequent encoding.

Rule notation:

\[ A_1 \ldots A_n \quad | \quad A_1 \ldots A_n \Rightarrow A \]

Introduction: How can I prove \( A \oplus B \)?

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

Operational Reading:

Introduction rule:
To prove \( A \) it suffices to prove \( A_1 \ldots A_n \).

Elimination rule:
If we know \( A_1 \) and we want to prove \( A \) it suffices to prove \( A_2 \ldots A_n \).

Natural Deduction for Propositional Logic:

\[ \begin{align*}
A &\quad \text{conjI} \\
A \land B &\quad \text{conjE} \\
A \land B &\quad \text{impI} \\
A \land B &\quad \text{notI} \\
\neg A &\quad \text{notE}
\end{align*} \]

More Rules:

\[ \begin{align*}
\text{iffI} &\quad A \rightarrow B, B \rightarrow A \\
\text{iffD1} &\quad A = B \\
\text{iffD2} &\quad A = B
\end{align*} \]
“Classical” Rules

\[ A \Rightarrow \text{False} \]
\[ \text{ccontr} \]
\[ A \Rightarrow A \text{ classical} \]

- \text{ccontr} and \text{classical} are not derivable from the Natural Deduction rules.
- They make the logic “classical”, i.e. “non-constructive or "non-intuitionistic".

Proof by Assumption

\[ \begin{align*}
A_1 & \ldots & A_n
\end{align*} \]

- Proof method: \text{assumption}
- Use: \text{apply assumption}
- Proves: \[ [A_1; \ldots ; A_n] \Rightarrow A \]
  by unifying \( A \) with one of the \( A_i \)

Rule Application: The Rough Idea

Applying rule \[ [A_1; \ldots ; A_n] \Rightarrow A \] to subgoal \( C \):

- Unify \( A \) and \( C \)
- Replace \( C \) with \( n \) new subgoals: \( A'_1 \ldots A'_n \)

Backwards reduction, like in Prolog

Example: rule \[ (?P; ?Q) \Rightarrow ?P \land ?Q \]

subgoal: \[ 1. A \land B \]

Result: \[ 1. A2. B \]

Rule Application: More Complete Idea

Applying rule \[ [A_1; \ldots ; A_n] \Rightarrow A \] to subgoal \( C \):

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

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: \[ [A_1; \ldots ; A_n] \Rightarrow A \]

Subgoal: \[ 1. [B_1; \ldots ; B_m] \Rightarrow C \]

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

New subgoals: \[ 1. [\sigma (B_1); \ldots ; \sigma (B_m)] \Rightarrow \sigma (A_1) \]
\[ \vdots \]
\[ n. [\sigma (B_1); \ldots ; \sigma (B_m)] \Rightarrow \sigma (A_n) \]

Proves: \[ [\sigma (B_1); \ldots ; \sigma (B_m)] \Rightarrow \sigma (C) \]

Command: \text{apply (rule <rulename>)}

Applying Elimination Rules

apply (rule <elim-rule>)

Like \text{rule} but also

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


Subgoal: 1. \[ |X; A \land B; Y| \implies Z\]

Unification: \(?P \land ?Q \equiv A \land B\) and \(?R \equiv Z\)

New subgoal: 1. \[ |X; Y| \implies |A; B| \implies Z\]

Same as: 1. \[ |X; Y; A; B| \implies Z\]