Getting Started with Isabelle

- Possibilities:
  - Use Isabelle on EWS
  - Install on your machine
- 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
    - First time you use it, it will rebuild all its core theories (takes minutes)

System Architecture

- **Isabelle/jEdit**
  - jEdit based interface
- **Isar**
  - Isabelle proof scripting language
- **Isabelle/HOL**
  - Isabelle instance for HOL
- **Isabelle**
  - Generic theorem prover
- **Standard ML**
  - Implementation language

Demo: my_theory

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 (* end 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 (\(\land, \lor, \neg, \rightarrow, \forall, \exists\), ...)
- Contains propositional logic, first-order logic
- **HOL** is very similar to a functional programming language
- Higher-order = functions are values, too!
- Booleans are values, too! And predicates are functions.
- We’ll start with propositional and first order logic
Formulae (first Approximation)

- Syntax (in decreasing priority):

  - `form ::= (form) | term = term | ¬form | form ∧ form | form ∨ form | form → form | ∀x. form | ∃x. form`
  - and some others

- 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 y ∧ Q x ≡ ∀x. (∃y. (P x y ∧ 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 (large subset 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`
  - `ImpTh`: name of imported theories. Importing is transitive.
Meta-logic: Basic Constructs

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

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

Universal Quantifier: \( \Lambda (\ldots !\ldots) \)
Usually inserted and removed by Isabelle automatically

Do not use inside HOL formulae

Rule/Goal Notation
\[|A_1; \ldots ; A_n|\] = \( \Rightarrow \) \( B \)
abbreviates \( A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow B \)

Means the rule (or potential rule):
\[ A_1; \ldots ; A_n \]
by
Plugins → Plugin Options… → Isabelle → General →
Print Mode = brackets

The Proof/Goal State
1. \( \Lambda x_1 \ldots x_m. \; [|A_1; \ldots ; A_n|] \Rightarrow B \)

\( x_1 \ldots x_m \) Local constants (fixed variables)
\( A_1 \ldots A_n \) Local subgoals / assumptions
\( B \) Actual goal / conclusion

Operational Reading
For each logical operator \( \oplus \), have two kinds of rules:

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

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

Natural Deduction

Proof Basics

Isabelle uses Natural Deduction proofs

Rule notation:
\[
\begin{array}{c}
\text{Rule} \\
A_1 \ldots A_n \\
\ \ \ A \\
\hline
B \\
\hline
A_1 \ldots A_n \Rightarrow A \\
\hline
\end{array}
\]

Sequent Encoding
\[
[ A_1, \ldots, A_n ] \Rightarrow A
\]

\[ [ A_1; \ldots ; A_n | ] \]

Proof/Goal Notation

Operational Reading
**Natural Deduction for Propositional Logic**

- **Rules**
  - **Conjunction** (conjI): \( A \land B \)
  - **Disjunction** (disjI): \( A \lor B \)
  - **Implication** (impI): \( A \rightarrow B \)
  - **Negation** (notI): \( \neg A \)

- **Elimination Rules**
  - **Conjunction Elimination** (conjE): \( A \land B \rightarrow C \)
  - **Disjunction Elimination** (disjE): \( A \lor B \rightarrow \)

- **Proof by Assumption**
  - **Assumption** (assumption): \( A_1 \ldots A_n \)
  - **Application** (apply assumption): \( A \rightarrow B \)

- **Example**
  - **Rule**: \( A \rightarrow B \rightarrow A \)

**“Classical” Rules**

- **Negation of Implication** (iffI): \( 
  \begin{align*}
  & \neg A \\
  \rightarrow & B \\
  \rightarrow & A
  \end{align*}
\)

- **Classical Negation** (classical): \( A \)

**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: \( A'_1 \ldots A'_n \)

**Example**

- **Rule**: \( [?P, ?Q] \rightarrow \neg P \land ?Q \)

**Subgoal**: 
1. \( A \land B \)
2. \( B \)
Applying rule \([A_1; \ldots; A_n] \implies A\) 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)

Example

Rule:
\([?P \land ?Q; ?P; ?Q] \implies ?R\) \implies ?R.
Subgoal: 1. \([X; A \land B; Y] \implies Z\)
Unification:
\(?P \land ?Q \equiv A \land B\) and \(?R \equiv Z\)
\{?P \mapsto A; ?Q \mapsto B; ?R \mapsto Z\}
New subgoal: 1. \([X; Y] \implies [A; B] \implies Z\)
Same as: 1. \([X; Y; A; B] \implies Z\)

Demo: my_theory