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:

```plaintext
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 (`∧`, `∨`, `¬`, `→`, `∀`, `∃`, ...)
- 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):

  ```plaintext
  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 HOL - need to not confuse them

Theory = Module

Syntax:
thory 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: ⇒ (==>)
For separating premises and conclusion of theorems / rules

Equality: ≡ (==)
For definitions

Universal Quantifier: Λ (!!)
Usually inserted and removed by Isabelle automatically

Do not use inside HOL formulae

Rule/Goal Notation

[| A1; ... ; An |] ⇒ B
abbreviates
A1 ⇒ ... ⇒ An ⇒ B
and means the rule (or potential rule):

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

; ≈ "and"

Note: A theorem is a rule; a rule is a theorem.
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)
- \( A_1 \ldots A_m \): Local assumptions
- \( B \): Actual (sub)goal

Proof Basics

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

\[
\begin{align*}
A_1 & \Rightarrow B \\
A_2 & \Rightarrow B \\
& \vdots \\
A_n & \Rightarrow B
\end{align*}
\]

Natural Deduction

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 \( A \oplus B \)?

\[
A \Rightarrow B \\
B \Rightarrow A
\]

Operational Reading

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

**Elimination rule**: If we know \( A_i \) and we want to prove \( A \)

it suffices to prove \( A_2 \ldots A_n \)

Natural Deduction for Propositional Logic

- \( \land \): \( \text{conjI} \)
- \( \land \): \( \text{conjE} \)
- \( \lor \): \( \text{disjI} \)
- \( \lor \): \( \text{disjE} \)
- \( \Rightarrow \): \( \text{impI} \)
- \( \Rightarrow \): \( \text{impE} \)
- \( \neg \): \( \text{notI} \)
- \( \neg \): \( \text{notE} \)

Natural Deduction for Propositional Logic

- \( \rightarrow \): \( \text{iffI} \)
- \( \rightarrow \): \( \text{iffD} \)
**More Rules**

\[
\begin{align*}
A \land B & \quad \text{conjunct1} \\
A & \quad \\
B & \quad \\
A \rightarrow B & \quad \text{mp} \\
A & \quad B \\
\end{align*}
\]

Compare to elimination rules:

\[
\begin{align*}
A \land B [A; B] & \Rightarrow C \\
\text{conjE} & \\
A & \Rightarrow B \\
B & \Rightarrow C \\
\text{impE} & \\
C & \\
\end{align*}
\]

**“Classical” Rules**

\[
\begin{align*}
A & \Rightarrow \text{False} \\
\text{ccontr} & \\
A & \Rightarrow A \\
\text{classical} & \\
\end{align*}
\]

- \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**

\[
\begin{align*}
A_1 \ldots A_i \ldots A_n & \Rightarrow A_i \\
\text{Proof method: assumption} & \\
\text{Use:} & \\
\text{apply assumption} & \\
\text{Proves:} & \\
[A_1; \ldots; A_n] & \Rightarrow A \\
\end{align*}
\]

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. \( A_2 \). \( B \)

**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: 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: \[
\text{\|} ?P \land ?Q; \text{\|} ?P; ?Q \text{\|} \Rightarrow ?R \text{\|} \Rightarrow ?R
\]

Subgoal: 1. \[
\text{\|} X; A \land B; Y \text{\|} \Rightarrow \text{\|} \]

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

New subgoal: 1. \[
\text{\|} X; Y \text{\|} \Rightarrow \text{\|} A; B \text{\|} \Rightarrow Z
\]

Same as: 1. \[
\text{\|} X; Y; A; B \text{\|} \Rightarrow Z
\]