Binary Search (java.util.Arrays)

1: public static int binarySearch(int[] a, int key) {
2:     int low = 0;
3:     int high = a.length - 1;
4:     while (low <= high) {
5:         int mid = (low + high) / 2;
6:         int midVal = a[mid];
7:         if (midVal < key) low = mid + 1;
8:         else if (midVal > key) high = mid - 1;
9:         else return mid; // key found
10:     } // key not found
11:     return -(low + 1); // key not found
12: }

http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html
What you will learn

➜ how to use a theorem prover
➜ background, how it works
➜ how to prove and specify
➜ how to reason about programs

Health Warning
Theorem Proving is addictive

What you should do to have a chance of succeeding

➜ attend lectures
➜ try Isabelle early
➜ redo all the demos alone
➜ try the exercises/homework we give, when we do give some
➜ DO NOT CHEAT

• Assignments and exams are take-home. This does NOT mean you can work in groups. Each submission is personal.
• For more info, see Plagiarism Policy

Content — Using Theorem Provers

Rough timeline
[1] Intro & motivation, getting started
[today]

Foundations & Principles
• Lambda Calculus, natural deduction [2,3,4]*
• Higher Order Logic [5,6,7]
• Term rewriting [8,9,10]*

Proof & Specification Techniques
• Isar [11,12]*
• Inductively defined sets, rule induction [13,14,15]
• Datatypes, recursion, induction [16,17,18,19]
• Calculational reasoning, mathematics style proofs [20]
• Hoare logic, proofs about programs [21,22,23]

* a1 out; * a1 due; * a2 out; * a2 due; * session break; * a3 out; * a3 due

Credits

some material (in using-theorem-provers part) shamelessly stolen from

Tobias Nipkow, Larry Paulson, Markus Wenzel

David Basin, Burkhard Wolff

Don’t blame them, errors are mine
What is a proof?

Merriam-Webster

- from Latin probare (test, approve, prove)
- to learn or find out by experience (archaic)
- to establish the existence, truth, or validity of (by evidence or logic)

prove a theorem, the charges were never proved in court

pops up everywhere

- politics (weapons of mass destruction)
- courts (beyond reasonable doubt)
- religion (god exists)
- science (cold fusion works)

What is a mathematical proof?

In mathematics, a proof is a demonstration that, given certain axioms, some statement of interest is necessarily true. (Wikipedia)

Example: \( \sqrt{2} \) is not rational.
Proof: assume there is \( r \in \mathbb{Q} \) such that \( r^2 = 2 \).
Hence there are mutually prime \( p \) and \( q \) with \( r = \frac{p}{q} \).
Thus \( 2q^2 = p^2 \), i.e. \( p^2 \) is divisible by 2.
2 is prime, hence it also divides \( p \), i.e. \( p = 2s \).
Substituting this into \( 2q^2 = p^2 \) and dividing by 2 gives \( q^2 = 2s^2 \). Hence, \( q \) is also divisible by 2. Contradiction. Qed.

What is a formal proof?

A derivation in a formal calculus

Example: \( A \land B \rightarrow B \land A \) derivable in the following system

Rules:
- \( S \vdash X \) (assumption)
- \( S \cup \{ X \} \vdash Y \) (impl)
- \( S \vdash X \land Y \) (conj)
- \( S \cup \{ X, Y \} \vdash Z \) (conjE)

Proof:
1. \( \{ A, B \} \vdash B \) (by assumption)
2. \( \{ A, B \} \vdash A \) (by assumption)
3. \( \{ A, B \} \vdash B \land A \) (by conj with 1 and 2)
4. \( \{ A \land B \} \vdash B \land A \) (by conjE with 3)
5. \( \{ \} \vdash A \land B \rightarrow B \land A \) (by impl with 4)
What is a theorem prover?

Implementation of a formal logic on a computer.
- fully automated (propositional logic)
- automated, but not necessarily terminating (first order logic)
- with automation, but mainly interactive (higher order logic)
- based on rules and axioms
- can deliver proofs

There are other (algorithmic) verification tools:
- model checking, static analysis, ...
- usually do not deliver proofs

Why theorem proving?

- Analysing systems/programs thoroughly
- Finding design and specification errors early
- High assurance (mathematical, machine checked proof)
- It’s not always easy
- It’s fun

Main theorem proving system for this course

λ → ∀ = Isabelle

Isabelle
- used here for applications, learning how to prove

What is Isabelle?

A generic interactive proof assistant
- generic:
  - not specialised to one particular logic
  - (two large developments: HOL and ZF, will mainly use HOL)
- interactive:
  - more than just yes/no, you can interactively guide the system
- proof assistant:
  - helps to explore, find, and maintain proofs
Why Isabelle?

- free
- widely used systems
- active development
- high expressiveness and automation
- reasonably easy to use
- (and because we know it best :))

If I prove it on the computer, it is correct, right?

No, because:

1. hardware could be faulty
2. operating system could be faulty
3. implementation runtime system could be faulty
4. compiler could be faulty
5. implementation could be faulty
6. logic could be inconsistent
7. theorem could mean something else

If I prove it on the computer, it is correct, right?

No, but:

- probability for
  - OS and H/W issues reduced by using different systems
  - runtime/compiler bugs reduced by using different compilers
  - faulty implementation reduced by right architecture
  - inconsistent logic reduced by implementing and analysing it
  - wrong theorem reduced by expressive/intuitive logics

- No guarantees, but assurance immensely higher than manual proof
If I prove it on the computer, it is correct, right?

Soundness architectures

- Careful implementation: PVS
- LCF approach, small proof kernel: HOL4, Isabelle
- Explicit proofs + proof checker: Coq, Twelf, Isabelle, HOL4

Meta Logic

Meta language:
The language used to talk about another language.

Examples:
English in a Spanish class, English in an English class

Meta logic:
The logic used to formalize another logic

Example:
Mathematics used to formalize derivations in formal logic

Formulae: $F ::= V \mid F \rightarrow F \mid F \land F \mid \text{False}$

Syntax:

Derivable: $S \vdash X$ $X$ a formula, $S$ a set of formulae

<table>
<thead>
<tr>
<th>logic</th>
<th>meta logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \in S$</td>
<td>$S \cup {X} \vdash Y$</td>
</tr>
<tr>
<td>$S \vdash X$</td>
<td>$S \vdash X \rightarrow Y$</td>
</tr>
<tr>
<td>$S \vdash X \land Y$</td>
<td>$S \cup {X, Y} \vdash Z$</td>
</tr>
<tr>
<td>$S \cup {X, Y} \vdash Z$</td>
<td>$S \vdash X \land Y$</td>
</tr>
</tbody>
</table>

Isabelle’s Meta Logic

$\land \quad \Rightarrow \quad \lambda$
Example: a theorem

mathematics: If \( x < 0 \) and \( y < 0 \), then \( x + y < 0 \)

formal logic: \( \vdash x < 0 \land y < 0 \rightarrow x + y < 0 \)

Isabelle: \( \text{lemma } "x < 0 \land y < 0 \rightarrow x + y < 0" \)

variation: \( \text{lemma } [x < 0; y < 0] \Rightarrow x + y < 0 \)

variation: \( \text{lemma } \) assumes "\( x < 0 \)" and "\( y < 0 \)" shows "\( x + y < 0 \)"

Example: a rule

logic: \[
\begin{array}{c}
X & Y \\
\hline
X \land Y
\end{array}
\]

Isabelle: \( S \vdash X \land Y \)

Variation: \( \frac{S \vdash X \quad S \vdash Y}{S \vdash X \land Y} \)
Example: a rule with nested implication

\[
\begin{array}{c}
X \\
\vdash \\
Y \\
\vdash \\
X \lor Y \\
\vdash \\
Z \\
\vdash \\
\end{array} \\
\]

\[
\begin{array}{c}
S \cup \{X\} \vdash Z \\
S \cup \{Y\} \vdash Z \\
S \cup \{X \lor Y\} \vdash Z \\
\end{array} \\
\]

Isabelle: \([X \lor Y; X \Rightarrow Z; Y \Rightarrow Z] \Rightarrow Z\)

\[\lambda\]

Syntax: \(\lambda x. F\) (\(F\) another meta level formula)

in ASCII: \(\%x. F\)

- lambda abstraction
- used for functions in object logics
- used to encode bound variables in object logics
- more about this in the next lecture

\[\text{Enough Theory!} \]
\[\text{Getting started with Isabelle}\]

System Architecture

- Proof General – user interface
- HOL, ZF – object-logics
- Isabelle – generic, interactive theorem prover
- Standard ML – logic implemented as ADT

User can access all layers!
System Requirements

- Linux, Windows, FreeBSD, MacOS X or Solaris
- Standard ML (PolyML fastest, SML/NJ supports more platforms)
- Emacs (for ProofGeneral) or Java (for jEdit)


ProofGeneral

- User interface for Isabelle
- Runs under XEmacs or Emacs
- Isabelle process in background

Interaction via
- Basic editing in XEmacs (with highlighting etc)
- Buttons (tool bar)
- Key bindings
- ProofGeneral Menu (lots of options, try them)

Documentation

Available from http://isabelle.in.tum.de
- Learning Isabelle
  - Tutorial on Isabelle/HOL (LNCS 2283)
  - Tutorial on Isar
  - Tutorial on Locales
- Reference Manuals
  - Isabelle/Isar Reference Manual
  - Isabelle Reference Manual
  - Isabelle System Manual
- Reference Manuals for Object-Logics

X-Symbol Cheat Sheet

Input of funny symbols in ProofGeneral
- via menu ("X-Symbol")
- via ASCII encoding (similar to LaTeX): \&and, \|or, ...
- via abbreviation: /. /\, \rightarrow, ...
- via rotate: 1 \rightarrow = \lambda (cycles through variations of letter)

<table>
<thead>
<tr>
<th>( \forall )</th>
<th>( \exists )</th>
<th>( \lambda )</th>
<th>( &amp; )</th>
<th>( | )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>x</td>
<td>\rightarrow</td>
<td>\Rightarrow</td>
</tr>
</tbody>
</table>

\[ 1 \xrightarrow{\text{converted to X-Symbol}} 2 \xrightarrow{\text{stays ASCII}}]
Exercises

- Download and install Isabelle from
  http://mirror.cse.unsw.edu.au/pub/isabelle/
- Switch on X-Symbol in ProofGeneral
- Step through the demo files from the lecture web page
- Write your own theory file, look at some theorems in the library, try 'find theorem'

- How many theorems can help you if you need to prove something like "Suc(Suc x)"?
- What is the name of the theorem for associativity of addition of natural numbers in the library?