COMP 4161
NICTA Advanced Course
Advanced Topics in Software Verification
Gerwin Klein
Formal Methods

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

ORGANISATORIALS

When
Mon 13:00 – 14:30
Wed 13:00 – 14:30

Where
Mon: Webst 250
Wed: Law Th G23

http://www.cse.unsw.edu.au/~cs4161/

CONTENT — USING THEOREM PROVERS

► Intro & motivation, getting started (today)
► Foundations & Principles
  • Lambda Calculus
  • Higher Order Logic, natural deduction
  • Term rewriting
► Proof & Specification Techniques
  • Datatypes, recursion, induction
  • Inductively defined sets, rule induction
  • Calculational reasoning, mathematics style proofs
  • Hoare logic, proofs about programs

Slide 1 Slide 2 Slide 3 Slide 4
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?

to prove

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., \( p = 2k \).

Substituting this into \( 2q^2 = p^2 \) and dividing by 2 gives \( q^2 = 2k^2 \).

Hence, \( q \) is also divisible by 2. Contradiction. Qed.

NICE, BUT...

still not rigorous enough for some

- what are the rules?
- what are the axioms?
- how big can the steps be?
- what is obvious or trivial?

informal language, easy to get wrong
easy to miss something, easy to cheat

Theorem. A cat has nine tails.

Proof. No cat has eight tails. Since one cat has one more tail than no cat, it must have nine tails.
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:

- $X \in S$ (assumption)
- $S \cup \{X\} \vdash Y$ (impl)
- $S \vdash X$ $S \vdash Y$ (conjI)
- $S \cup \{X, Y\} \vdash Z$ (conjE)

Proof:

1. $\{A, B\} \vdash B$ (by assumption)
2. $\{A, B\} \vdash A$ (by assumption)
3. $\{A, B\} \vdash B \land A$ (by conjI 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

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 I 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
⇒ 1 and 2 reduced by using different systems
⇒ 3 and 4 reduced by using different compilers
⇒ inconsistent logic reduced by implementing and analysing it
⇒ wrong theorem reduced by expressive/intuitive logics

No guarantees, but assurance immensely higher than manual proof

Soundness architectures

- careful implementation
- LCF approach, small proof kernel
- explicit proofs + proof checker

PVS
HOL4
Coq
Twelf
Isabelle
HOL4

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

Syntax:
Formulae: \( F ::= V \mid F \rightarrow F \mid F \land F \mid \text{False} \)
\( V ::= [A - Z] \)

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

\[
\begin{array}{c}
\text{logic} / \text{meta logic} \\
X \in S \\
S \vdash X & S \vdash Y \\
\frac{S \vdash X}{S \vdash X \land Y} & \frac{S \vdash \{X \vdash Y\} \vdash Z} \\
\end{array}
\]
Slide 21

\[ \forall \Rightarrow \lambda \]

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

in ASCII: \( !x. \, F \)

⇒  universal quantifier on the meta level
⇒  used to denote parameters
⇒  example and more later

Slide 22

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

in ASCII: \( !x. \, F \)

⇒  universal quantifier on the meta level
⇒  used to denote parameters
⇒  example and more later

Slide 23

\[ \Rightarrow \]

EXAMPLE: A THEOREM

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

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

variation: \( x < 0; y < 0 \vdash x + y < 0 \)

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

variation: lemma "\( [x < 0, y < 0] \) \rightarrow x + y < 0"

variation: lemma assumes "\( x < 0" \) and "\( y < 0" \) shows "\( x + y < 0" \)
Example: a rule

Logic:
\[ \frac{X \land Y}{X \land Y} \]

Variation:
\[ \frac{S \vdash X \quad S \vdash Y}{S \vdash X \land Y} \]

Isabelle:
\[ \{X, Y\} \Rightarrow X \land Y \]

Example: a rule with nested implication

Logic:
\[ \begin{array}{c}
\frac{X \lor Y}{X \lor Y} \\
\frac{Z}{Z}
\end{array} \]

Variation:
\[ \frac{S \cup \{X\} \vdash Z \quad S \cup \{Y\} \vdash Z}{S \cup \{X \lor Y\} \vdash Z} \]

Isabelle:
\[ \{X \lor Y; X \Rightarrow Z; Y \Rightarrow Z\} \Rightarrow Z \]

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

Enough theory!
Getting started with Isabelle