NICTA Advanced Course

Theorem Proving
Principles, Techniques, Applications

Gerwin Klein
Formal Methods

Organisatorials

When	Mon	14:00 – 15:30
	Wed	10:30 – 12:00
	7 weeks	ends Mon, 20.9.2004
Exceptions	Mon 6.9., 13.9., 20.9. at 15:00 – 16:30	

Web page:

`http://www.cse.unsw.edu.au/~kleing/teaching/thprv-04/`

free – no credits – no assigments

What you will learn

➜ how to use a theorem prover

WHAT YOU WILL LEARN

➜ how to use a theorem prover

➜ background, how it works

What you will learn

- ➜ how to use a theorem prover
- ➜ background, how it works
- ➜ how to prove and specify

- ➜ how to use a theorem prover
- ➜ background, how it works
- ➜ how to prove and specify

Health Warning

Theorem Proving is addictive

What you will not learn

➜ semantics / model theory

WHAT YOU WILL NOT LEARN

- semantics / model theory
- soundness / completeness proofs

What you will not learn

- ➜ semantics / model theory
- ➜ soundness / completeness proofs
- ➜ decision procedures

CONTENT

➜ Intro & motivation, getting started with Isabelle (today)

CONTENT

- ➜ Intro & motivation, getting started with Isabelle (today)
- ➜ Foundations & Principles
  - Lambda Calculus
  - Higher Order Logic, natural deduction
  - Term rewriting

Content

➜ Intro & motivation, getting started with Isabelle (today)

➜ Foundations & Principles

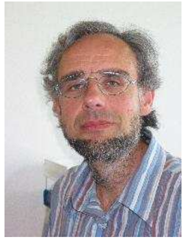
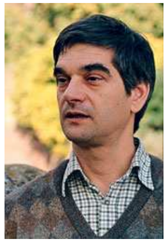
- Lambda Calculus
- Higher Order Logic, natural deduction
- Term rewriting

➜ Proof & Specification Techniques

- Datatypes, recursion, induction
- Inductively defined sets, rule induction
- Calculational reasoning, mathematics style proofs
- Hoare logic, proofs about programs

CREDITS

material (in part) shamelessly stolen from

Tobias Nipkow, Larry Paulson, Markus Wenzel

David Basin, Burkhardt Wolff

Don’t blame them, errors are mine

WHAT IS A PROOF?

to prove

WHAT IS A PROOF?

to prove (Marriam-Webster)

➜ from Latin probare (test, approve, prove)

What is a proof?

to prove (Marriam-Webster)

➜ from Latin probare (test, approve, prove)

➜ to learn or find out by experience (archaic)

What is a proof?

to prove (Marriam-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

WHAT IS A PROOF?

to prove (Marriam-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:

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.

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

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

A derivation in a formal calculus

Example: $A \wedge B \longrightarrow B \wedge A$ derivable in the following system

Rules: $\dfrac{X \in S}{S \vdash X}$ (assumption) $\qquad \dfrac{S \cup \{X\} \vdash Y}{S \vdash X \longrightarrow Y}$ (impI)

$\dfrac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y}$ (conjI) $\qquad \dfrac{S \cup \{X, Y\} \vdash Z}{S \cup \{X \wedge Y\} \vdash Z}$ (conjE)

What is a formal proof?

A derivation in a formal calculus

Example: $A \wedge B \longrightarrow B \wedge A$ derivable in the following system

Rules: $\dfrac{X \in S}{S \vdash X}$ (assumption) $\dfrac{S \cup \{X\} \vdash Y}{S \vdash X \longrightarrow Y}$ (impI)

$\dfrac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y}$ (conjI) $\dfrac{S \cup \{X, Y\} \vdash Z}{S \cup \{X \wedge Y\} \vdash Z}$ (conjE)

Proof:

1. $\{A, B\} \vdash B$ (by assumption)
2. $\{A, B\} \vdash A$ (by assumption)
3. $\{A, B\} \vdash B \wedge A$ (by conjI with 1 and 2)
4. $\{A \wedge B\} \vdash B \wedge A$ (by conjE with 3)
5. $\{\} \vdash A \wedge B \longrightarrow B \wedge A$ (by impI 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)

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

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) verifi cation tools:

- ➜ model checking, static analysis, ...
- ➜ usually do not deliver proofs

WHY THEOREM PROVING?

- ➜ Analysing systems/programs thoroughly

WHY THEOREM PROVING?

- ➜ Analysing systems/programs thoroughly
- ➜ Finding design and specification errors early

WHY THEOREM PROVING?

- ➜ Analysing systems/programs thoroughly
- ➜ Finding design and specification errors early
- ➜ High assurance (mathematical, machine checked proof)

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

WHAT IS ISABELLE?

A generic interactive proof assistant

A generic interactive proof assistant

- **generic:**
  not specialised to one particular logic
  (two large developments: HOL and ZF, will mainly use HOL)

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

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 system

➜ active development

➜ high expressiveness and automation

➜ reasonably easy to use

WHY ISABELLE?

- ➜ free
- ➜ widely used system
- ➜ active development
- ➜ high expressiveness and automation
- ➜ reasonably easy to use
- ➜ (and because I know it best ;-))

WHY ISABELLE?

- free
- widely used system
- active development
- high expressiveness and automation
- reasonably easy to use
- (and because I know it best ;-))

We will see other systems, too: HOL4, Coq, Waldmeister

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

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

No, because:

No, because:

① hardware could be faulty

IF I PROVE IT ON THE COMPUTER, IT IS CORRECT, RIGHT?

No, because:

① hardware could be faulty

② operating system could be faulty

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

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

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

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

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:

IF I PROVE IT ON THE COMPUTER, IT IS CORRECT, RIGHT?

No, but:

probability for

➜ 1 and 2 reduced by using different systems

No, but:

probability for

- ➜ 1 and 2 reduced by using different systems
- ➜ 3 and 4 reduced by using different compilers

No, but:

probability for

- ➜ 1 and 2 reduced by using different systems
- ➜ 3 and 4 reduced by using different compilers
- ➜ faulty implementation reduced by right architecture

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
- ➜ faulty implementation reduced by right architecture
- ➜ inconsistent logic reduced by implementing and analysing it

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

➜ faulty implementation reduced by right architecture

➜ inconsistent logic reduced by implementing and analysing it

➜ wrong theorem reduced by expressive/intuitive logics

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
- ➜ 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 way higher than manual proof

IF I PROVE IT ON THE COMPUTER, IT IS CORRECT, RIGHT?

Soundness architectures

careful implementation PVS

Soundness architectures

careful implementation	PVS
LCF approach, small proof kernel	HOL4
	Isabelle

Soundness architectures

careful implementation	PVS
LCF approach, small proof kernel	HOL4
	Isabelle
explicit proofs + proof checker	Coq
	Twelf
	Isabelle

META LOGIC

Meta language:

The language used to talk about another language.

Meta language:

The language used to talk about another language.

Examples:

English in a Spanish class, English in an English class

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

META LOGIC – EXAMPLE

Syntax:

Formulae: $F ::= V \quad | \quad F \longrightarrow F \quad | \quad F \wedge F \quad | \quad False$

$V ::= [A - Z]$

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

META LOGIC – EXAMPLE

Syntax:

Formulae: $F ::= V \quad | \quad F \longrightarrow F \quad | \quad F \wedge F \quad | \quad False$

$V ::= [A - Z]$

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

logic / meta logic

$$\frac{X \in S}{S \vdash X} \qquad \frac{S \cup \{X\} \vdash Y}{S \vdash X \longrightarrow Y}$$

$$\frac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y} \qquad \frac{S \cup \{X, Y\} \vdash Z}{S \cup \{X \wedge Y\} \vdash Z}$$

Isabelle's Meta Logic

$$\bigwedge \qquad\qquad \Longrightarrow \qquad\qquad \lambda$$

$\bigwedge$

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

in ASCII: `!!x. F`

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

in ASCII: `!!x. F`

- ➜ universial quantifier on the meta level
- ➜ used to denote parameters
- ➜ example and more later

Syntax: $A \Longrightarrow B$ (A, B other meta level formulae)

in ASCII: `A ==> B`

Syntax: $A \Longrightarrow B$ (A, B other meta level formulae)

in ASCII: `A ==> B`

Binds to the right:

$$A \Longrightarrow B \Longrightarrow C \quad = \quad A \Longrightarrow (B \Longrightarrow C)$$

Abbreviation:

$$[\![A; B]\!] \Longrightarrow C \quad = \quad A \Longrightarrow B \Longrightarrow C$$

- ➜ read: A and B implies C
- ➜ used to write down rules, theorems, and proof states

EXAMPLE: A THEOREM

mathematics: if $x < 0$ and $y < 0$, then $x + y < 0$

EXAMPLE: A THEOREM

mathematics: if $x < 0$ and $y < 0$, then $x + y < 0$

formal logic: $\vdash\ x < 0 \wedge y < 0 \longrightarrow x + y < 0$

variation: $x < 0; y < 0\ \vdash\ x + y < 0$

Example: a theorem

mathematics: if $x < 0$ and $y < 0$, then $x + y < 0$

formal logic: $\vdash\ x < 0 \wedge y < 0 \longrightarrow x + y < 0$

variation: $x < 0; y < 0\ \vdash\ x + y < 0$

Isabelle: **lemma** "$x < 0 \wedge y < 0 \longrightarrow x + y < 0$"

variation: **lemma** "$[\![x < 0; y < 0]\!] \Longrightarrow x + y < 0$"

EXAMPLE: A THEOREM

mathematics: if $x < 0$ and $y < 0$, then $x + y < 0$

formal logic: $\vdash\ x < 0 \land y < 0 \longrightarrow x + y < 0$

variation: $x < 0; y < 0\ \vdash\ x + y < 0$

Isabelle: **lemma** "$x < 0 \land y < 0 \longrightarrow x + y < 0$"

variation: **lemma** "$[\![x < 0; y < 0]\!] \Longrightarrow x + y < 0$"

variation: **lemma**

assumes "$x < 0$" and "$y < 0$" shows "$x + y < 0$"

Example: a rule

logic: $$\frac{X \quad Y}{X \wedge Y}$$

EXAMPLE: A RULE

logic:
$$\frac{X \quad Y}{X \wedge Y}$$

variation:
$$\frac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y}$$

EXAMPLE: A RULE

logic: $$\frac{X \quad Y}{X \wedge Y}$$

variation: $$\frac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y}$$

Isabelle: $$[\![X; Y]\!] \Longrightarrow X \wedge Y$$

EXAMPLE: A RULE WITH NESTED IMPLICATION

logic:
$$\dfrac{X \lor Y \qquad \begin{array}{c} X \\ \vdots \\ Z \end{array} \qquad \begin{array}{c} Y \\ \vdots \\ Z \end{array}}{Z}$$

EXAMPLE: A RULE WITH NESTED IMPLICATION

logic:

$$\dfrac{X \vee Y \qquad \begin{array}{c} X \\ \vdots \\ Z \end{array} \qquad \begin{array}{c} Y \\ \vdots \\ Z \end{array}}{Z}$$

variation:

$$\dfrac{S \cup \{X\} \vdash Z \qquad S \cup \{Y\} \vdash Z}{S \cup \{X \vee Y\} \vdash Z}$$

Example: a rule with nested implication

logic:

$$\frac{X \vee Y \quad \begin{array}{c} X \\ \vdots \\ Z \end{array} \quad \begin{array}{c} Y \\ \vdots \\ Z \end{array}}{Z}$$

variation:

$$\frac{S \cup \{X\} \vdash Z \quad S \cup \{Y\} \vdash Z}{S \cup \{X \vee Y\} \vdash Z}$$

Isabelle: $[\![X \vee Y; X \Longrightarrow Z; Y \Longrightarrow Z]\!] \Longrightarrow Z$

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

in ASCII: `%x. F`

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

in ASCII: `%x. F`

➜ lambda abstraction

➜ used to 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

Isabelle – generic, interactive theorem prover

SYSTEM ARCHITECTURE

Isabelle – generic, interactive theorem prover

Standard ML – logic implemented as ADT

SYSTEM ARCHITECTURE

HOL, ZF – object-logics

Isabelle – generic, interactive theorem prover

Standard ML – logic implemented as ADT

SYSTEM ARCHITECTURE

Proof General – user interface

HOL, ZF – object-logics

Isabelle – generic, interactive theorem prover

Standard ML – logic implemented as ADT

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**, **MacOS X** or **Solaris**
- ➜ **Standard ML**
  (PolyML fastest, SML/NJ supports more platforms)
- ➜ **XEmacs** or **Emacs**
  (for ProofGeneral)

If you do not have Linux, MacOS X or Solaris, try **IsaMorph**:

`http://www.brucker.ch/projects/isamorph/`

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

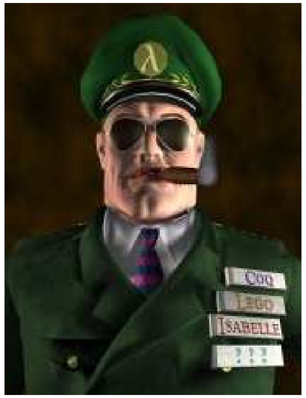
➜ Reference Manuals for Object-Logics

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)

X-Symbol Cheat Sheet

Input of funny symbols in ProofGeneral

- ➜ via menu (“X-Symbol”)
- ➜ via ASCII encoding (similar to LaTeX): `\<and>`, `\<or>`, `...`
- ➜ via abbreviation: `/\`, `\/`, `-->`, `...`
- ➜ via *rotate*: `l C-.` = λ (cycles through variations of letter)

	$\forall$	$\exists$	λ	$\neg$	$\wedge$	$\vee$	$\longrightarrow$	$\Rightarrow$
①	`\<forall>`	`\<exists>`	`\<lambda>`	`\<not>`	`/\`	`\/`	`-->`	`=>`
②	`ALL`	`EX`	`%`	`~`	`&`	`|`		

① converted to X-Symbol

② stays ASCII

DEMO

EXERCISES

- ➜ Download and install Isabelle from
  `http://isabelle.in.tum.de` or
  `http://mirror.cse.unsw.edu.au/pub/isabelle/`
- ➜ Switch on X-Symbol in ProofGeneral
- ➜ Step through the demo file from the lecture web page
- ➜ Write an own theory file, look at some theorems, try ’find theorem’