**Slide 1**

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:
5:    while (low <= high) ... (midVal > key)12:        high = mid - 1;
13:    else14:        return mid; // key found15:    }
16:    return -(low + 1); // key not found.17: }
```

6: int mid = (low + high) / 2;

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

---

**Slide 2**

---

**Slide 3**

About us

Members of the seL4 verification team

- Functional correctness of a C microkernel
- Isabelle/HOL model ↔ Haskell model ↔ C code
- 10 000 KLOC / 300 000 lines of proof script (!)
- 25 person years / $6 million


---

**Slide 4**
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

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?

(Marriam-Webster)

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. \( p = 2s \).
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

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:

\[
\begin{align*}
S \vdash X & \quad \text{(assumption)} \\
S \vdash Y & \quad \text{(impl)} \\
S \vdash X \land Y & \quad \text{(conjI)} \\
S \vdash X \land Y \\nS \vdash Z & \quad \text{(conjE)}
\end{align*}
\]

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 we know it best ;))
If I prove it on the computer, it is correct, right?

No, because:

➀ hardware could be faulty
➁ operating system could be faulty
➂ implementation runtime system could be faulty
➃ compiler could be faulty
➄ implementation could be faulty
➅ logic could be inconsistent
➆ theorem could mean something else

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

Soundness architectures

careful implementation → PVS
LCF approach, small proof kernel → HOL4
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

Syntax:
Formulae:

\[ F ::= V \mid F \rightarrow F \mid F \land F \mid False \]

V ::= [A \ldots Z]

Derivable:

\[ S \vdash X \quad X \text{ a formula, } S \text{ a set of formulae} \]

Derivable (meta logic):

\[
\begin{align*}
X & \in S \\
S & \vdash X
\end{align*}
\]

\[
\begin{align*}
S & \cup (X) \vdash Y \\
S & \vdash X \rightarrow Y
\end{align*}
\]

\[
\begin{align*}
S & \vdash X \quad S & \vdash Y \\
S & \vdash X \land Y \\
S & \cup (X) \cup Y \vdash Z \\
S & \cup (X \land Y) \vdash Z
\end{align*}
\]

Isabelle’s Meta Logic

\[
\begin{align*}
\bigwedge & \quad \implies \\
\lambda &
\end{align*}
\]

Syntax:

\[
\bigwedge x. F
\]

(F another meta level formula)

in ASCII:

\[ \forall x. F \]

• universal quantifier on the meta level
• used to denote parameters
• example and more later
Syntactically: \( A \rightarrow B \)  
(A, B other meta level formulae)

In ASCII: \( \Rightarrow \)

- **Binds to the right:**
  
  \( A \rightarrow B \rightarrow C = A \rightarrow [B \rightarrow C] \)

- **Abbreviation:**
  
  \[ [A; B] \rightarrow C = A \rightarrow B \rightarrow C \]

- **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 \( x + y < 0 \)

**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:**

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

**Variation:**

\[ \frac{S \vdash X \land Y}{S \vdash X \land Y} \]

**Isabelle:**

\[ [X; Y] \rightarrow X \land Y \]

---

**Example: a rule with nested implication**

**Logic:**

\[ \frac{X \lor Y \vdash Z}{X \lor Y \lor \neg Z \vdash Z} \]

**Variation:**

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

**Isabelle:**

\[ [X \lor Y; X \rightarrow Z; Y \rightarrow Z] \rightarrow Z \]
Syntax: $\lambda x. F$  (F another meta level formula)

in ASCII:  $\forall x. F$

- 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**