

# COMP 4161

Data61 Advanced Course

## Advanced Topics in Software Verification

Miki Tanaka,  
Johannes Åman Pohjola,  
June Andronick,  
Christine Rizkallah

# 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) {
6:          int mid = (low + high) / 2;
7:          int midVal = a[mid];
8:
9:          if (midVal < key)
10:             low = mid + 1
11:          else if (midVal > key)
12:             high = mid - 1;
13:          else
14:             return mid; // key found
15:      }
16:      return -(low + 1); // key not found.
17:  }
```

# 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) {
6:            int mid = (low + high) / 2;
7:            int midVal = a[mid];
8:
9:            if (midVal < key)
10:                low = mid + 1
11:            else if (midVal > key)
12:                high = mid - 1;
13:            else
14:                return mid; // key found
15:        }
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](http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html)

# Organisatorials



**When**    Tue    10:00 – 12:00  
              Wed    10:00 – 12:00

**Where**   Tue:    Electrical Engineering G04 (K-G17-G04)  
              Wed:    UNSW Business School 205 (K-E12-205)

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

# About us



## The trustworthy systems verification team

- Functional correctness and security of the seL4 microkernel  
Security ↔ Isabelle/HOL model ↔ Haskell model ↔ C code ↔ Binary

# About us



## The trustworthy systems verification team

- Functional correctness and security of the seL4 microkernel  
Security ↔ Isabelle/HOL model ↔ Haskell model ↔ C code ↔ Binary
- 10 000 LOC / 500 000 lines of proof; about 25 person years of effort

# About us



## The trustworthy systems verification team

- Functional correctness and security of the seL4 microkernel  
Security ↔ Isabelle/HOL model ↔ Haskell model ↔ C code ↔ Binary
- 10 000 LOC / 500 000 lines of proof; about 25 person years of effort
- Cogent code/proof co-generation; CakeML verified compiler; etc.

# About us



## The trustworthy systems verification team

- Functional correctness and security of the seL4 microkernel  
[Security](#) ↔ [Isabelle/HOL model](#) ↔ [Haskell model](#) ↔ [C code](#) ↔ [Binary](#)
- 10 000 LOC / 500 000 lines of proof; about 25 person years of effort
- Cogent code/proof co-generation; CakeML verified compiler; etc.

Open Source

<http://sel4.systems>

<https://ts.data61.csiro.au/projects/TS/cogent.pml>

<https://cakeml.org>



# About us



## The trustworthy systems verification team

- Functional correctness and security of the seL4 microkernel  
[Security](#) ↔ [Isabelle/HOL model](#) ↔ [Haskell model](#) ↔ [C code](#) ↔ [Binary](#)
- 10 000 LOC / 500 000 lines of proof; about 25 person years of effort
- Cogent code/proof co-generation; CakeML verified compiler; etc.

Open Source

<http://sel4.systems>

<https://ts.data61.csiro.au/projects/TS/cogent.pml>

<https://cakeml.org>

**We are always embarking on exciting new projects.**

**We offer**

- summer student scholarship projects
- honours and PhD theses
- research assistant and verification engineer positions

# 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

# What you will learn



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

# 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**

# Prerequisites



**This is an advanced course.** It assumes knowledge in

- Functional programming
- First-order formal logic

# Prerequisites



**This is an advanced course.** It assumes knowledge in

- Functional programming
- First-order formal logic

The following program should make sense to you:

$$\begin{aligned}\text{map } f \ [] &= [] \\ \text{map } f \ (x:xs) &= f\ x : \text{map } f\ xs\end{aligned}$$



# Prerequisites



**This is an advanced course.** It assumes knowledge in

- Functional programming
- First-order formal logic

The following program should make sense to you:

$$\begin{aligned}\text{map } f [] &= [] \\ \text{map } f (x:xs) &= f\ x : \text{map } f\ xs\end{aligned}$$

You should be able to read and understand this formula:

$$\exists x. (P(x) \longrightarrow \forall x. P(x))$$

# Content — Using Theorem Provers



→ Intro & motivation, getting started

# Content — Using Theorem Provers



- Intro & motivation, getting started
- Foundations & Principles
  - Lambda Calculus, natural deduction
  - Higher Order Logic, Isar (part 1)
  - Term rewriting

# Content — Using Theorem Provers



- Intro & motivation, getting started
- Foundations & Principles
  - Lambda Calculus, natural deduction
  - Higher Order Logic, Isar (part 1)
  - Term rewriting
- Proof & Specification Techniques
  - Inductively defined sets, rule induction
  - Datatypes, recursion, induction, Isar (part 2)
  - Hoare logic, proofs about programs, invariants
  - C verification
  - Practice, questions, exam prep

# Content — Using Theorem Provers



Rough timeline

[today]

→ Intro & motivation, getting started

→ Foundations & Principles

- Lambda Calculus, natural deduction
- Higher Order Logic, Isar (part 1)
- Term rewriting

[1,2]

[3<sup>a</sup>]

[4]

→ Proof & Specification Techniques

- Inductively defined sets, rule induction
- Datatypes, recursion, induction, Isar (part 2)
- Hoare logic, proofs about programs, invariants
- C verification
- Practice, questions, exam prep

[5]

[6, 7<sup>b</sup>]

[8]

[9]

[10<sup>c</sup>]

---

<sup>a</sup>a1 due; <sup>b</sup>a2 due; <sup>c</sup>a3 due

# What you should do to have a chance at succeeding



# What you should do to have a chance at succeeding



→ attend lectures

# What you should do to have a chance at succeeding



- attend lectures
- try Isabelle early



# What you should do to have a chance at succeeding



- attend lectures
- try Isabelle early
- redo all the demos alone

# What you should do to have a chance at succeeding



- attend lectures
- try Isabelle early
- redo all the demos alone
- try the exercises/homework we give, when we do give some

# What you should do to have a chance at 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<sup>a</sup>

---

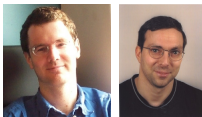
<sup>a</sup> <https://student.unsw.edu.au/plagiarism>

# Credits

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



Tobias Nipkow, Larry Paulson, Markus Wenzel



David Basin, Burkhardt Wolff

**Don't blame them, errors are ours**

# What is a proof?

to prove



# What is a proof?

**to prove**

→ from Latin probare (test, approve, prove)



(Merriam-Webster)

# What is a proof?

## to prove

- from Latin probare (test, approve, prove)
- to learn or find out by experience (archaic)



(Merriam-Webster)

# What is a proof?



(Merriam-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*



# What is a proof?



(Merriam-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:

# 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**

# 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:**

$$\frac{X \in S}{S \vdash X} \text{ (assumption)} \qquad \frac{S \cup \{X\} \vdash Y}{S \vdash X \longrightarrow Y} \text{ (impl)}$$
$$\frac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y} \text{ (conjI)} \qquad \frac{S \cup \{X, Y\} \vdash Z}{S \cup \{X \wedge Y\} \vdash Z} \text{ (conjE)}$$

# 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:**

$$\frac{X \in S}{S \vdash X} \text{ (assumption)} \quad \frac{S \cup \{X\} \vdash Y}{S \vdash X \longrightarrow Y} \text{ (impl)}$$
$$\frac{S \vdash X \quad S \vdash Y}{S \vdash X \wedge Y} \text{ (conjI)} \quad \frac{S \cup \{X, Y\} \vdash Z}{S \cup \{X \wedge Y\} \vdash Z} \text{ (conjE)}$$

### 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 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)

# 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) verification tools:

- model checking, static analysis, ...
- usually do not deliver proofs
- See COMP3153: Algorithmic Verification

# 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

→ used here for applications, learning how to prove



# What is Isabelle?

**A generic interactive proof assistant**



# 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)

# 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

# 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

# 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?**

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



No, because:



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



**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

# 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

# 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

→ OS and H/W issues reduced by using different systems

# 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

# 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 having the right prover architecture

# 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 having the right prover architecture
- inconsistent logic reduced by implementing and analysing it

# 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 having the right prover 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

- OS and H/W issues reduced by using different systems
- runtime/compiler bugs reduced by using different compilers
- faulty implementation reduced by having the right prover 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

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



**Soundness architectures**  
careful implementation

PVS

LCF approach, small proof kernel

HOL4  
Isabelle



# 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.

# 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



## **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



Formulae:  $F ::= V \mid F \longrightarrow F \mid F \wedge F \mid \text{False}$

**Syntax:**  $V ::= [A - Z]$

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

# Meta Logic – Example



Formulae:  $F ::= V \mid F \longrightarrow F \mid F \wedge F \mid \text{False}$

Syntax:  $V ::= [A - Z]$

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

logic / meta logic

$$\frac{X \in S}{S \vdash X}$$

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

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

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

# Isabelle's Meta Logic



$\wedge \Rightarrow \lambda$

$\wedge$



**Syntax:**  $\wedge x. F$  ( $F$  another meta level formula)  
**in ASCII:** `!!x. F`





**Syntax:**  $\bigwedge x. F$  ( $F$  another meta level formula)  
in ASCII: `!!x. F`

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



**Syntax:**  $A \implies B$   
in ASCII:  $A ==> B$

( $A, B$  other meta level formulae)





**Syntax:**  $A \implies B$  ( $A, B$  other meta level formulae)  
in ASCII:  $A \implies B$

**Binds to the right:**

$$A \implies B \implies C = A \implies (B \implies C)$$

**Abbreviation:**

$$\llbracket A; B \rrbracket \implies C = A \implies B \implies C$$

- ➔ 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** " $\llbracket x < 0; y < 0 \rrbracket \Longrightarrow x + y < 0$ "

# 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** " $\llbracket x < 0; y < 0 \rrbracket \Longrightarrow x + y < 0$ "

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:** 
$$\llbracket X; Y \rrbracket \Longrightarrow X \wedge Y$$

# Example: a rule with nested implication

logic:

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

# 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:

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

# 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:**

$$\llbracket X \vee Y; X \Longrightarrow Z; Y \Longrightarrow Z \rrbracket \Longrightarrow Z$$

$\lambda$



**Syntax:**  $\lambda x. F$  ( $F$  another meta level formula)  
in ASCII: `%x. F`

$\lambda$



**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

# Enough Theory!

Getting started with Isabelle



# System Architecture



**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



**Prover IDE (jEdit)** – user interface

**HOL, ZF** – object-logics

**Isabelle** – generic, interactive theorem prover

**Standard ML** – logic implemented as ADT

# System Architecture



**Prover IDE (jEdit)** – 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, or MacOS X (10.8 +)**
- **Standard ML** (PolyML implementation)
- **Java** (for jEdit)

Premade packages for Linux, Mac, and Windows + info on:  
<http://mirror.cse.unsw.edu.au/pub/isabelle/>

# Documentation



Available from <http://isabelle.in.tum.de>

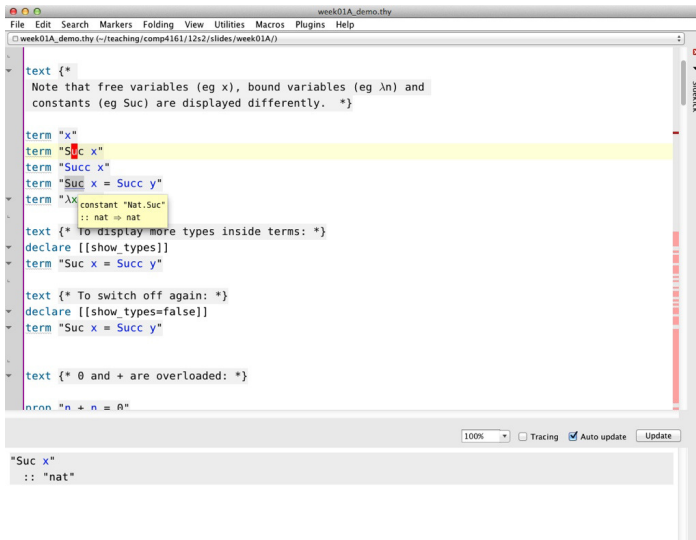
## → Learning Isabelle

- Concrete Semantics Book
- 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



```
week01A_demo.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
week01A_demo.thy (-/teaching/comp4161/12s2/slides/week01A/)

text {*
  Note that free variables (eg x), bound variables (eg λn) and
  constants (eg Suc) are displayed differently. *}

term "x"
term "Suc x"
term "Succ x"
term "Suc x = Succ y"
term "λx. constant \"Nat.Suc\"
      :: nat ⇒ nat"
text {* To display more types inside terms: *}
declare [[show_types]]
term "Suc x = Succ y"

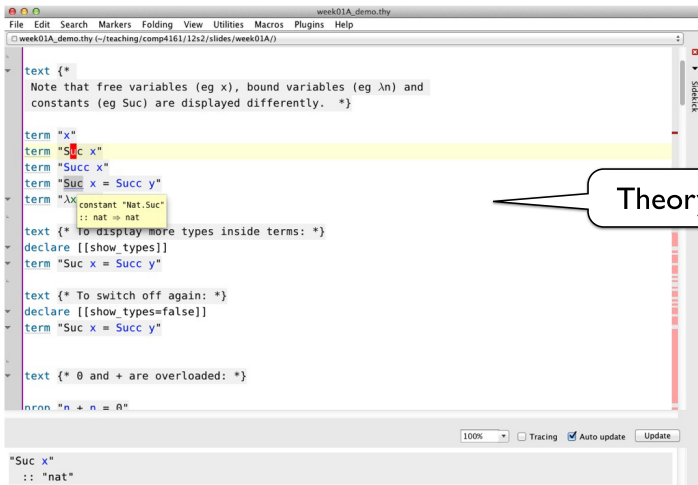
text {* To switch off again: *}
declare [[show_types=false]]
term "Suc x = Succ y"

text {* 0 and + are overloaded: *}

nonrec "n + n = 0"

"Suc x"
:: "nat"
```





```
week01A_demo.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
week01A_demo.thy (-/teaching/comp4161/12s2/slides/week01A/)

text {*
  Note that free variables (eg x), bound variables (eg λn) and
  constants (eg Suc) are displayed differently. *}

term "x"
term "Succ x"
term "Succ x"
term "Succ x = Succ y"
term "λx. constant \"Nat.Succ\"
      :: nat => nat"
text {* To display more types inside terms: *}
declare [[show_types]]
term "Suc x = Succ y"

text {* To switch off again: *}
declare [[show_types=false]]
term "Suc x = Succ y"

text {* 0 and + are overloaded: *}

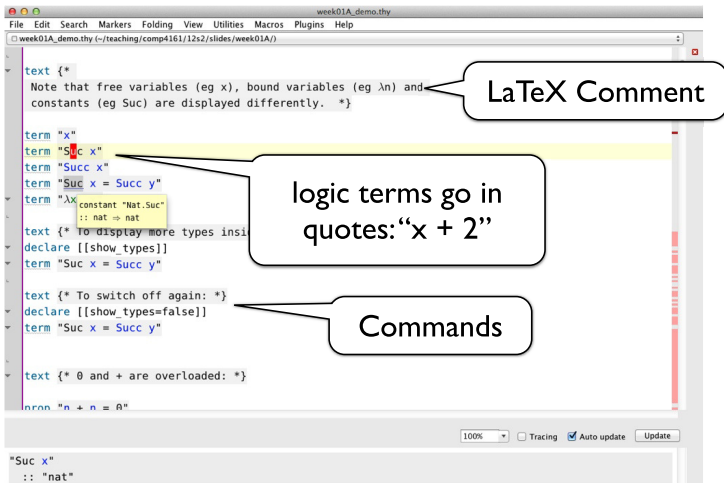
nonc "n + n = 0"

100% Tracing Auto update Update

"Succ x"
:: "nat"
```

Theory File

Isabelle Output



The screenshot shows the jEdit/PIDE editor interface with a file named `week01A_demo.thy`. The code is as follows:

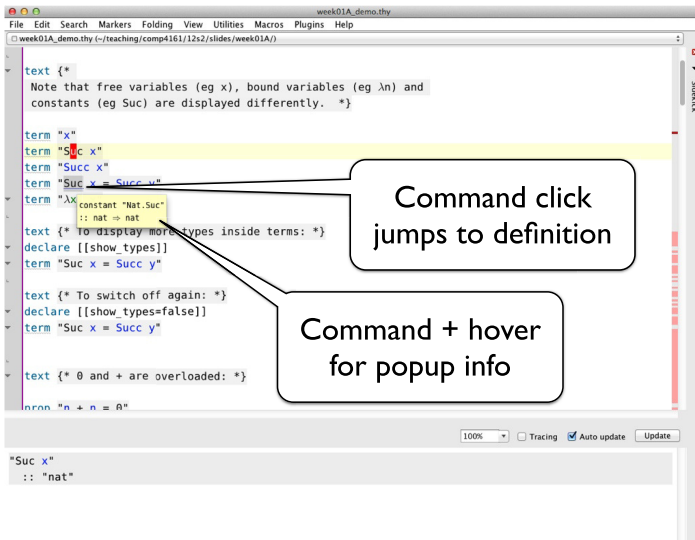
```
text {*  
  Note that free variables (eg x), bound variables (eg  $\lambda n$ ) and  
  constants (eg Suc) are displayed differently. *}  
  
term "x"  
term "Suc x"  
term "Succ x"  
term "Suc x = Succ y"  
term " $\lambda x$  constant \"Nat.Suc\"  
  :: nat  $\Rightarrow$  nat  
  
text {* To display more types inside  
declare [[show_types]]  
term "Suc x = Succ y"  
  
text {* To switch off again: *}  
declare [[show_types=false]]  
term "Suc x = Succ y"  
  
text {* 0 and + are overloaded: *}  
  
nonrec "n + n = 0"
```

Callouts from the image:

- LaTeX Comment**: Points to the comment `{* To display more types inside`.
- logic terms go in quotes: "x + 2"**: Points to the term `"Suc x"`.
- Commands**: Points to the `declare` command.

The bottom status bar shows: 100% | ☐ Tracing | ☒ Auto update | Update

# jEdit/PIDE



```
week01A_demo.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
week01A_demo.thy (-/teaching/comp4161/12s2/slides/week01A/)

text {*
  Note that free variables (eg x), bound variables (eg λn) and
  constants (eg Suc) are displayed differently. *}

term "x"
term "Suc x"
term "Succ x"
term "Suc x = Succ y"
term "λx. constant \"Nat.Suc\"
      :: nat ⇒ nat"
text {* To display more types inside terms: *}
declare [[show_types]]
term "Suc x = Succ y"

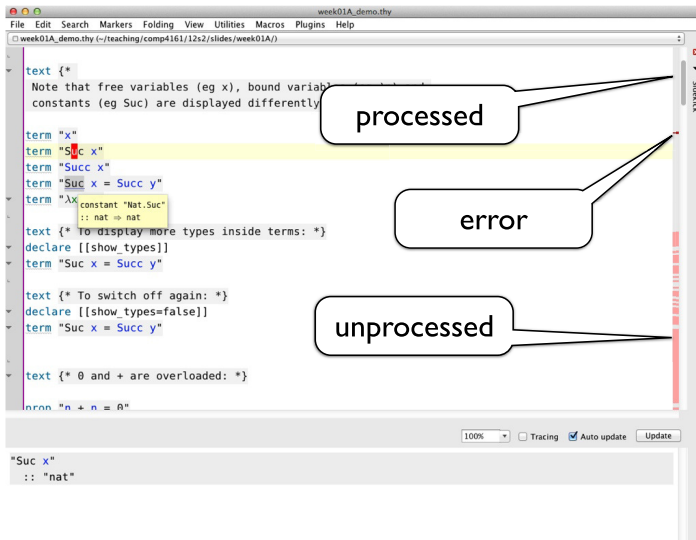
text {* To switch off again: *}
declare [[show_types=false]]
term "Suc x = Succ y"

text {* 0 and + are overloaded: *}

nonrec "n + n = 0"

-- "Suc x"
-- :: "nat"
```

# jEdit/PIDE



The screenshot shows the jEdit/PIDE editor interface with a file named `week01A_demo.thy`. The editor contains Lean 4 code with several annotations. A yellow highlight is placed over the line `term "Suc x"`, with a callout bubble labeled "processed" pointing to it. A red squiggly line is under the `Suc` in `"Suc x"`, with a callout bubble labeled "error" pointing to it. A red vertical bar is on the right side of the editor, with a callout bubble labeled "unprocessed" pointing to it. The code in the editor is as follows:

```
text {*  
  Note that free variables (eg x), bound variables (eg y), and  
  constants (eg Suc) are displayed differently  
*}  
  
term "x"  
term "Suc x"  
term "Succ x"  
term "Suc x = Succ y"  
term "λx. constant \"Nat.Suc\"  
  :: nat ⇒ nat"  
  
text {* To display more types inside terms: *}  
declare [[show_types]]  
term "Suc x = Succ y"  
  
text {* To switch off again: *}  
declare [[show_types=false]]  
term "Suc x = Succ y"  
  
text {* 0 and + are overloaded: *}  
  
nonrec "n + n = 0"
```

At the bottom of the editor, there is a status bar with a zoom level of 100%, checkboxes for Tracing and Auto update (checked), and an Update button. Below the editor, a snippet of code is visible:

```
"Suc x"  
:: "nat"
```

# Demo

# Exercises



- Download and install Isabelle from <http://mirror.cse.unsw.edu.au/pub/isabelle/>
- Step through the demo files from the lecture web page
- Write your own theory file, look at some theorems in the library, try 'find\_theorems'
- How many theorems can help you if you need to prove something containing the term  $\text{Suc}(\text{Suc } x)$ ?
- What is the name of the theorem for associativity of addition of natural numbers in the library?