

COMP 4161

NICTA Advanced Course

Advanced Topics in Software Verification

Toby Murray, June Andronick, Gerwin Klein

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>

Organisatorials



When Mon 14:00 – 15:30
Thu 15:00 – 16:30

Where Mon: Quadrangle G044 (E15-G044)
Thu: Mathews 309 (F23-309)

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

About us

Members of the seL4 verification team

- Functional correctness and security of a C microkernel
Security ↔ Isabelle/HOL model ↔ Haskell model ↔ C code
- 10 000 LOC / 500 000 lines of proof script (!)
- a bit under 30 person years of effort

It's all being open sourced, tomorrow!

`http://sel4.systems`

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
- background, how it works
- how to prove and specify
- how to reason about programs

Health Warning

Theorem Proving is addictive

Content — Using Theorem Provers

- Intro & motivation, getting started

- Foundations & Principles
 - Lambda Calculus, natural deduction [1,2]
 - Higher Order Logic [3^a]
 - Term rewriting [4]

- Proof & Specification Techniques
 - Inductively defined sets, rule induction [5]
 - Datatypes, recursion, induction [6, 7]
 - Hoare logic, proofs about programs, C verification [8^b,9]
 - (mid-semester break)
 - Writing Automated Proof Methods [10]
 - Isar, codegen, typeclasses, locales [11^c,12]

^a a1 due; ^b a2 due; ^c a3 due

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^a

^a www.cse.unsw.edu.au/about-us/organisational-structure/student-services/policies/

Credits

This course was originally written by



Gerwin Klein

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?

(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: 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: $A \wedge B \longrightarrow B \wedge A$ 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)

- based on rules and axioms
- can deliver proofs

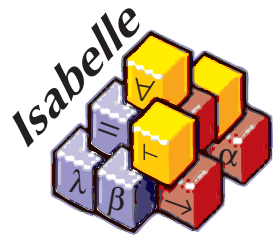
There are other (algorithmic) verification tools:

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

Why theorem proving?

- Analysing systems/programs thoroughly
- 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?

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 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

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

Meta Logic – Example

Formulae: $F ::= V \mid F \longrightarrow F \mid F \wedge F \mid False$

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



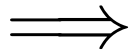
$\wedge \implies \lambda$

\bigwedge

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

in ASCII: $A ==> B$

Binds to the right:

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

Abbreviation:

$$[[A; B]] \implies C = A \implies B \implies C$$

→ 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 $x + y < 0$

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

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:

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

variation:

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

Isabelle:

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

λ

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

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 (10.7 +)**
- **Standard ML**
(PolyML fastest, SML/NJ supports more platforms)
- **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

- 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

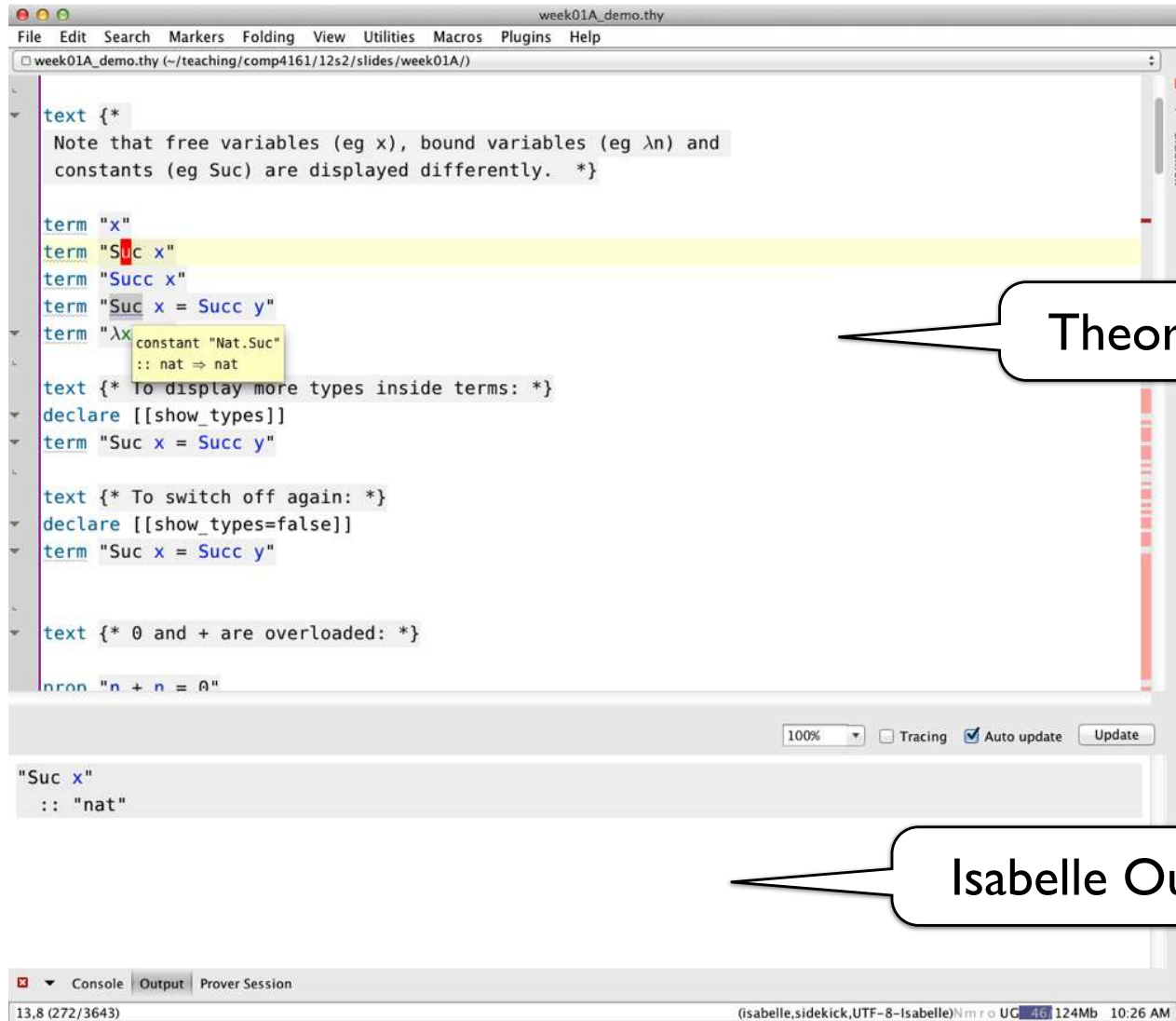
jEdit/PIDE

A screenshot of the jEdit/PIDE IDE interface. The main window displays a file named "week01A_demo.thy" with the following content:

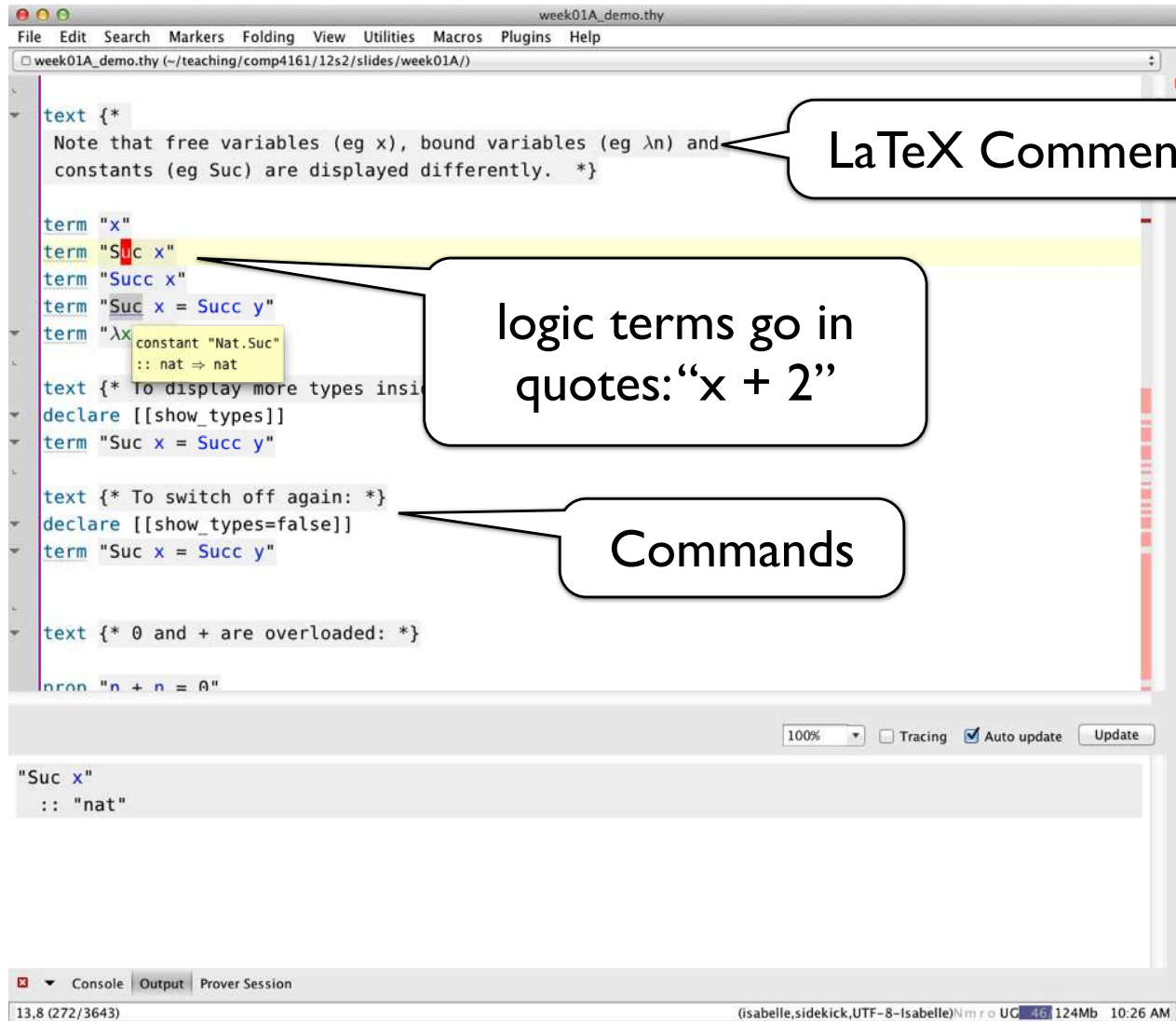
```
text {*  
  Note that free variables (eg x), bound variables (eg λn) and  
  constants (eg Suc) are displayed differently. *}  
  
term "x"  
term "Suc x"  
term "Succ x"  
term "Suc x = Succ y"  
term "λx. constant \"Nat.Suc\"  
  :: nat => nat"  
text {* To display more types inside terms: *}  
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: *}  
prop "n + n = 0"
```

A tooltip is visible over the lambda term, showing the type signature: `constant "Nat.Suc" :: nat => nat`. The interface includes a menu bar (File, Edit, Search, Markers, Folding, View, Utilities, Macros, Plugins, Help), a toolbar, a status bar at the bottom with "13.8 (272/3643)", and a console area at the bottom showing the output of the current term: `"Suc x" :: "nat"`. The status bar also shows "(isabelle.sidekick,UTF-8-Isabelle)N m r o UC 46 124Mb 10:26 AM".

jEdit/PIDE



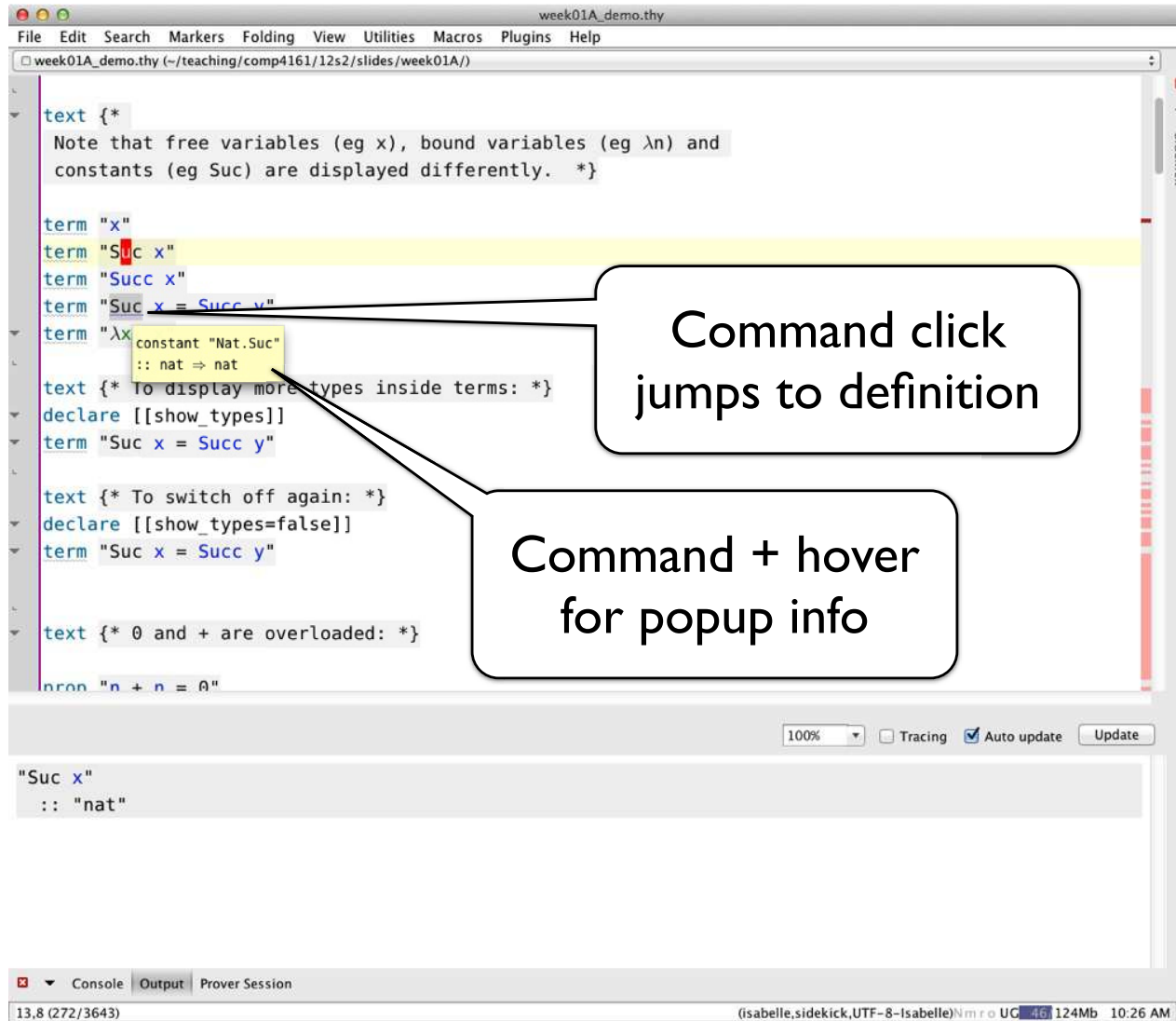
The screenshot shows the jEdit/PIDE IDE interface. The main editor window displays a theory file named 'week01A_demo.thy'. The code in the file includes comments and several Isabelle commands: `text` for comments, `term` for displaying terms, `constant` for defining constants, `declare` for setting options like `show_types`, and `prop` for stating a proposition. A yellow highlight is placed over the `term "Suc x"` line. A callout bubble labeled 'Theory File' points to this line. Below the editor, the Isabelle output window is visible, showing the output for the `"Suc x"` term: `:: "nat"`. A second callout bubble labeled 'Isabelle Output' points to this output. The bottom status bar shows the version '13.8 (272/3643)' and the session name '(isabelle.sidekick,UTF-8-Isabelle)Nmr o UC 46 124Mb 10:26 AM'.



The screenshot shows the jEdit/PIDE editor interface with a file named `week01A_demo.thy`. The editor contains several lines of code, with three callouts pointing to specific parts:

- LaTeX Comment:** A callout points to a `text` block containing a comment: `text {* Note that free variables (eg x), bound variables (eg λn) and constants (eg Succ) are displayed differently. *}`.
- logic terms go in quotes: "x + 2":** A callout points to a `term` definition: `term "Succ x"`. A tooltip for `Succ` shows: `constant "Nat.Succ" :: nat => nat`.
- Commands:** A callout points to a `term` definition: `term "Succ x = Succ y"`.

The bottom of the editor shows a preview of the rendered code: `"Succ x" :: "nat"`. The status bar at the bottom indicates version 13.8 (272/3643) and the current file is `(isabelle.sidekick,UTF-8-Isabelle)N.m.r.o.UC.46.124Mb` at 10:26 AM.



The screenshot shows the jEdit/PIDE editor interface with a file named 'week01A_demo.thy'. The code in the editor includes:

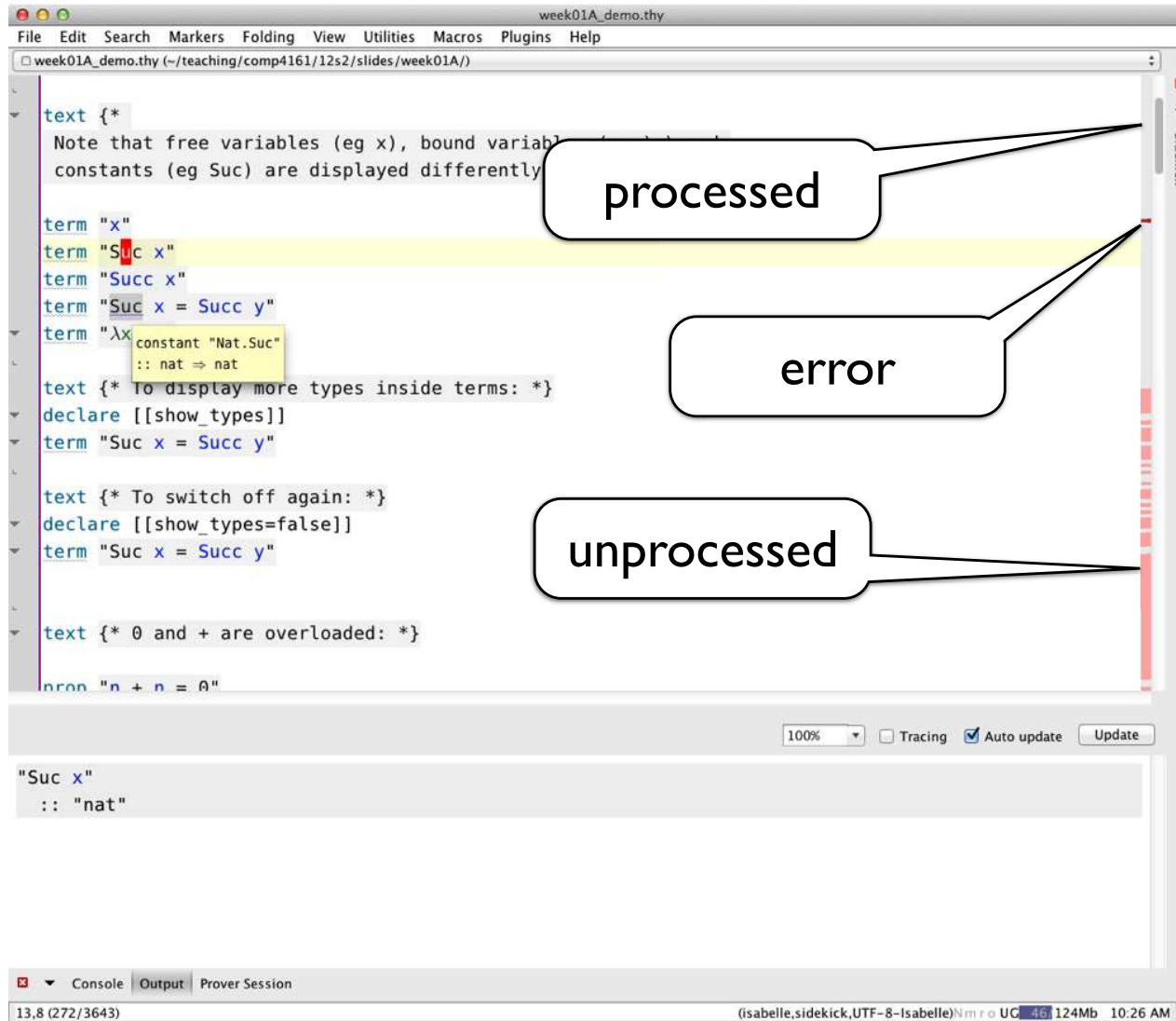
```
text {*  
Note that free variables (eg x), bound variables (eg λn) and  
constants (eg Suc) are displayed differently. *}  
  
term "x"  
term "Suc x"  
term "Succ x"  
term "Suc x = Succ y"  
term "λx  
constant "Nat.Suc"  
:: nat => nat  
text {* To display more types inside terms: *}  
declare [[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: *}  
prop "n + n = 0"
```

Two callout boxes provide instructions:

- Command click jumps to definition**: Points to the 'Suc' term in the code.
- Command + hover for popup info**: Points to the 'Nat.Suc' constant definition popup.

The bottom of the editor shows a preview of the term "Suc x" with its definition ":: nat". The status bar at the bottom indicates version 13.8 (272/3643) and system information: (isabelle.sidekick,UTF-8-Isabelle)Nmr o UC 46 124Mb 10:26 AM.

jEdit/PIDE



The screenshot shows the jEdit/PIDE editor interface with a file named 'week01A_demo.thy'. The editor contains several lines of code, including text blocks and terms. Three callout boxes are overlaid on the code:

- processed**: Points to the line `term "Suc x"`, which is highlighted in yellow.
- error**: Points to the line `term "Succ x"`, which is highlighted in yellow.
- unprocessed**: Points to the line `term "Suc x = Succ y"`, which is not highlighted.

The code in the editor includes:

```
text {*  
Note that free variables (eg x), bound variables (eg x),  
constants (eg Suc) are displayed differently  
*}  
  
term "x"  
term "Suc x"  
term "Succ x"  
term "Suc x = Succ y"  
term "\x  
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: *}  
prop "n + n = 0"
```

The bottom of the editor shows a console window with the output:

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

The status bar at the bottom indicates version 13.8 (272/3643) and the current session information: (isabelle.sidekick,UTF-8-Isabelle)Nmr o UC 46 124Mb 10:26 AM.

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 like “ $\text{Suc}(\text{Suc } x)$ ”?
- What is the name of the theorem for associativity of addition of natural numbers in the library?