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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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/`

Copyright NICTA 2014, provided under Creative Commons Attribution License

About us

NICTA

Members of the seL4 verification team

- ➜ Functional correctness and security of a C microkernel
  Security $\leftrightarrow$ Isabelle/HOL model $\leftrightarrow$ Haskell model $\leftrightarrow$ 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

Copyright NICTA 2014, provided under Creative Commons Attribution License

What you will learn

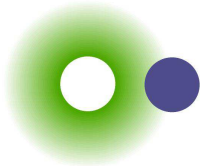
- how to use a theorem prover
- background, how it works
- how to prove and specify
- how to reason about programs

Health Warning

Theorem Proving is addictive

Copyright NICTA 2014, provided under Creative Commons Attribution License

Content — Using Theorem Provers



Rough timeline

- ➜ Intro & motivation, getting started [today]
- ➜ 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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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/`

Copyright NICTA 2014, provided under Creative Commons Attribution License

Credits

This course was originally written by

Gerwin Klein

Copyright NICTA 2014, provided under Creative Commons Attribution License

Credits

NICTA

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

What is a proof?

to prove (Merriam-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)

Copyright NICTA 2014, provided under Creative Commons Attribution License

What is a mathematical proof?

NICTA

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.

Copyright NICTA 2014, provided under Creative Commons Attribution License

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.

Copyright NICTA 2014, provided under Creative Commons Attribution License

What is a formal proof?

NICTA

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)

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)

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

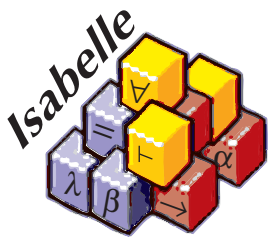
Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Main theorem proving system for this course



Isabelle

➜ used here for applications, learning how to prove

Copyright NICTA 2014, provided under Creative Commons Attribution License

What is Isabelle?

NICTA

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Why Isabelle?

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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 immensly higher than manual proof

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Meta Logic – Example

Syntax:

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

$V ::= [A - Z]$

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

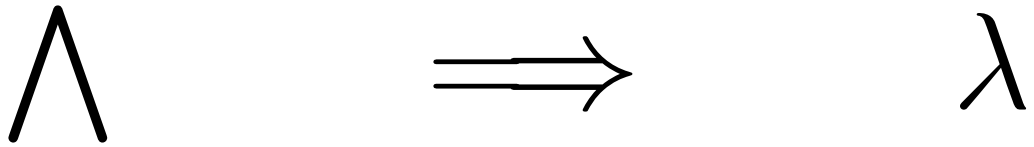
logic / meta logic

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

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Isabelle's Meta Logic



Copyright NICTA 2014, provided under Creative Commons Attribution License

$\bigwedge$

NICTA

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

$\Longrightarrow$

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Example: a theorem

NICTA

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

variation: **lemma**

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Example: a rule with nested implication

logic:

$$\dfrac{\begin{array}{ccc} & \overset{X}{\vdots} & \overset{Y}{\vdots} \\ X \vee Y & Z & Z \end{array}}{Z}$$

variation:

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

Isabelle:

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

Enough Theory!

Getting started with Isabelle

Copyright NICTA 2014, provided under Creative Commons Attribution License

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!

Copyright NICTA 2014, provided under Creative Commons Attribution License

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/`

Copyright NICTA 2014, provided under Creative Commons Attribution License

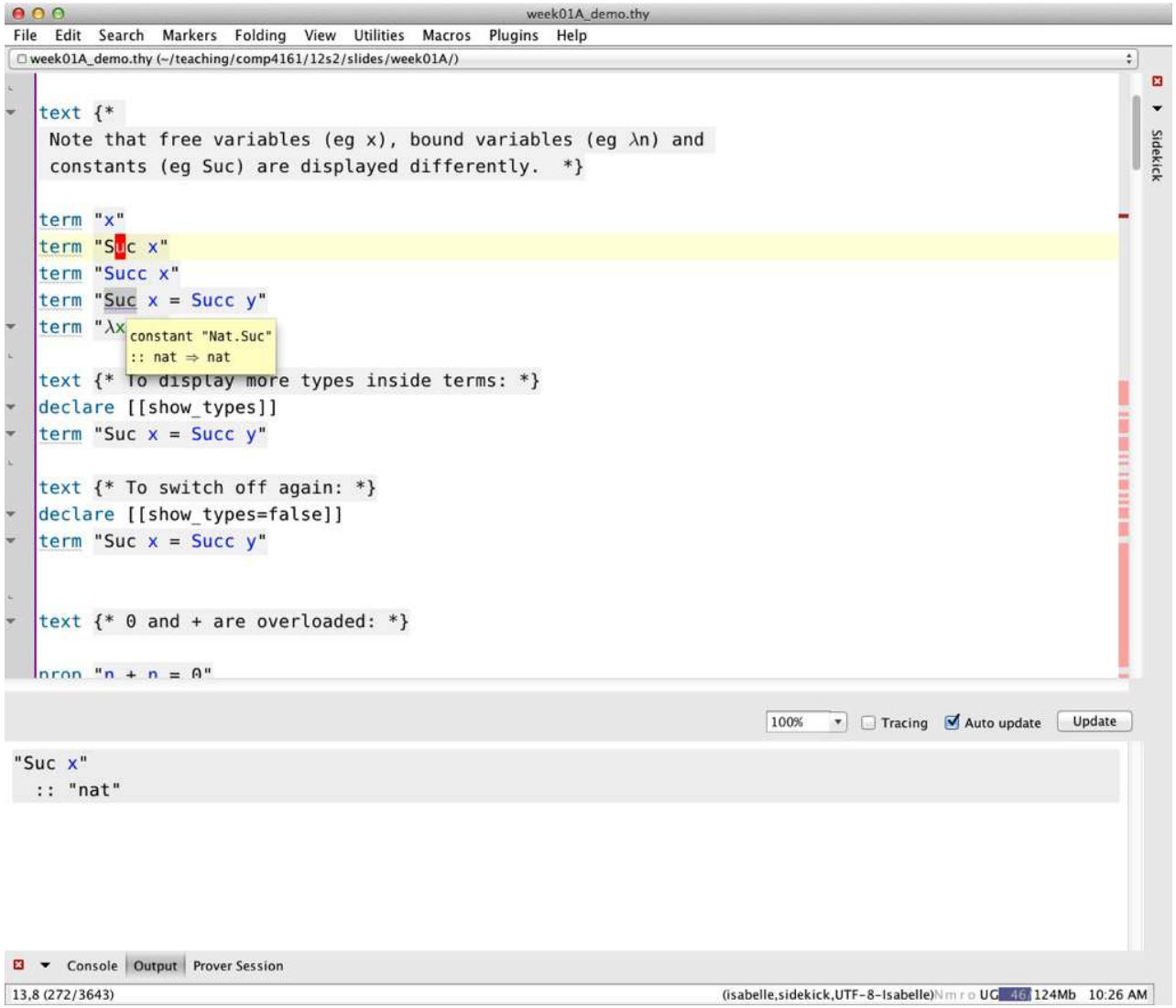
Documentation

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

Copyright NICTA 2014, provided under Creative Commons Attribution License

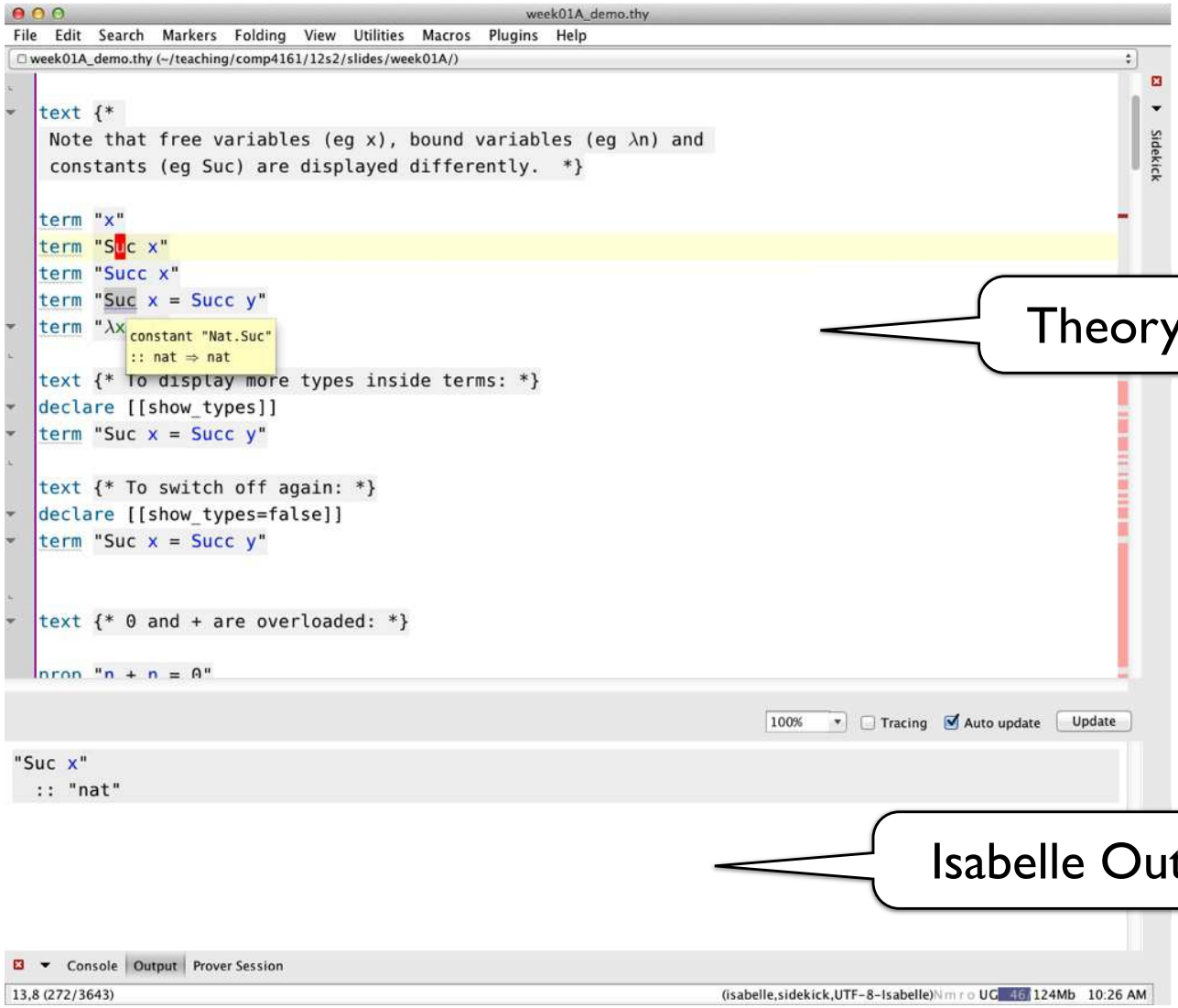
jEdit/PIDE



Copyright NICTA 2014, provided under Creative Commons Attribution License

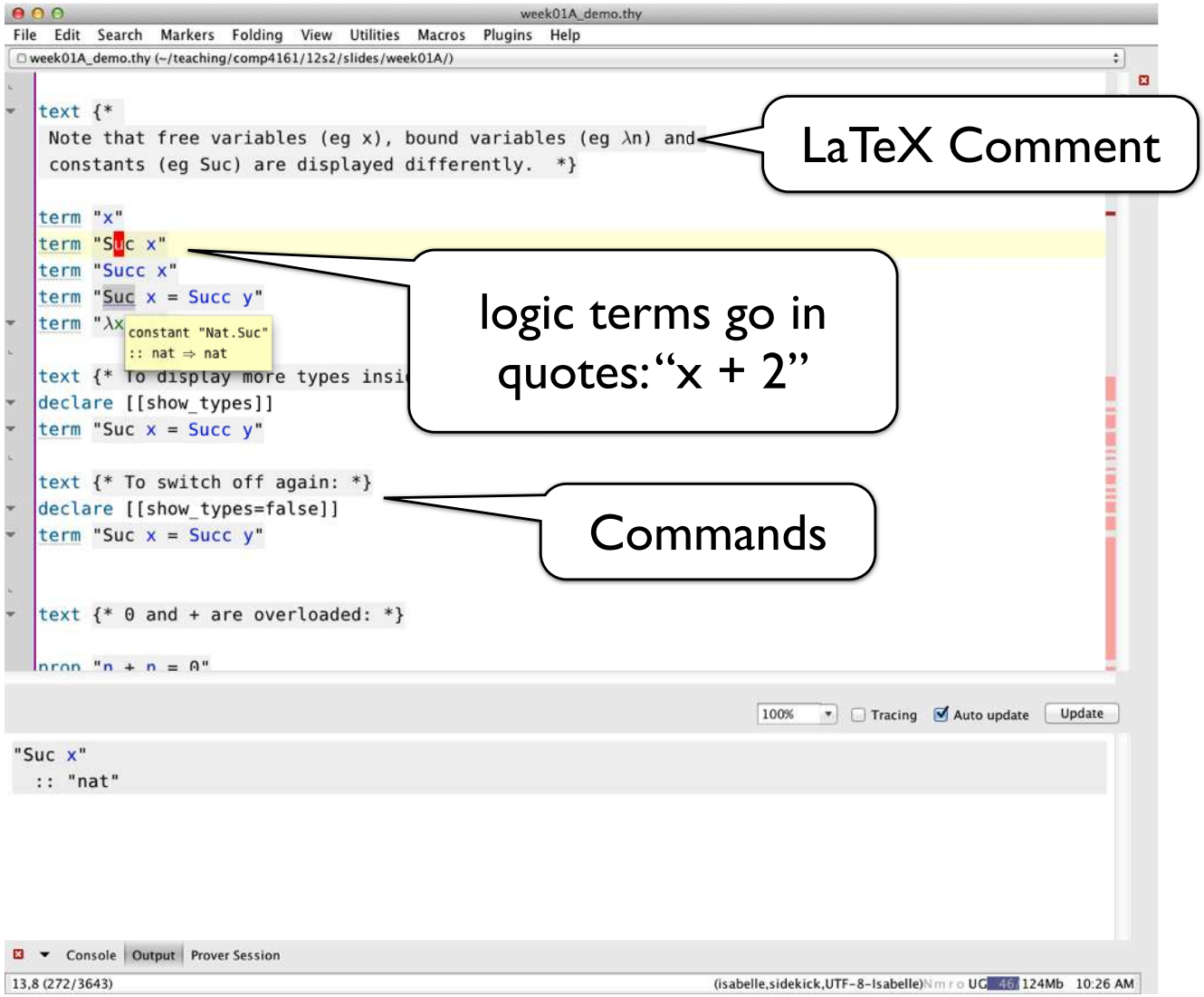
jEdit/PIDE

NICTA



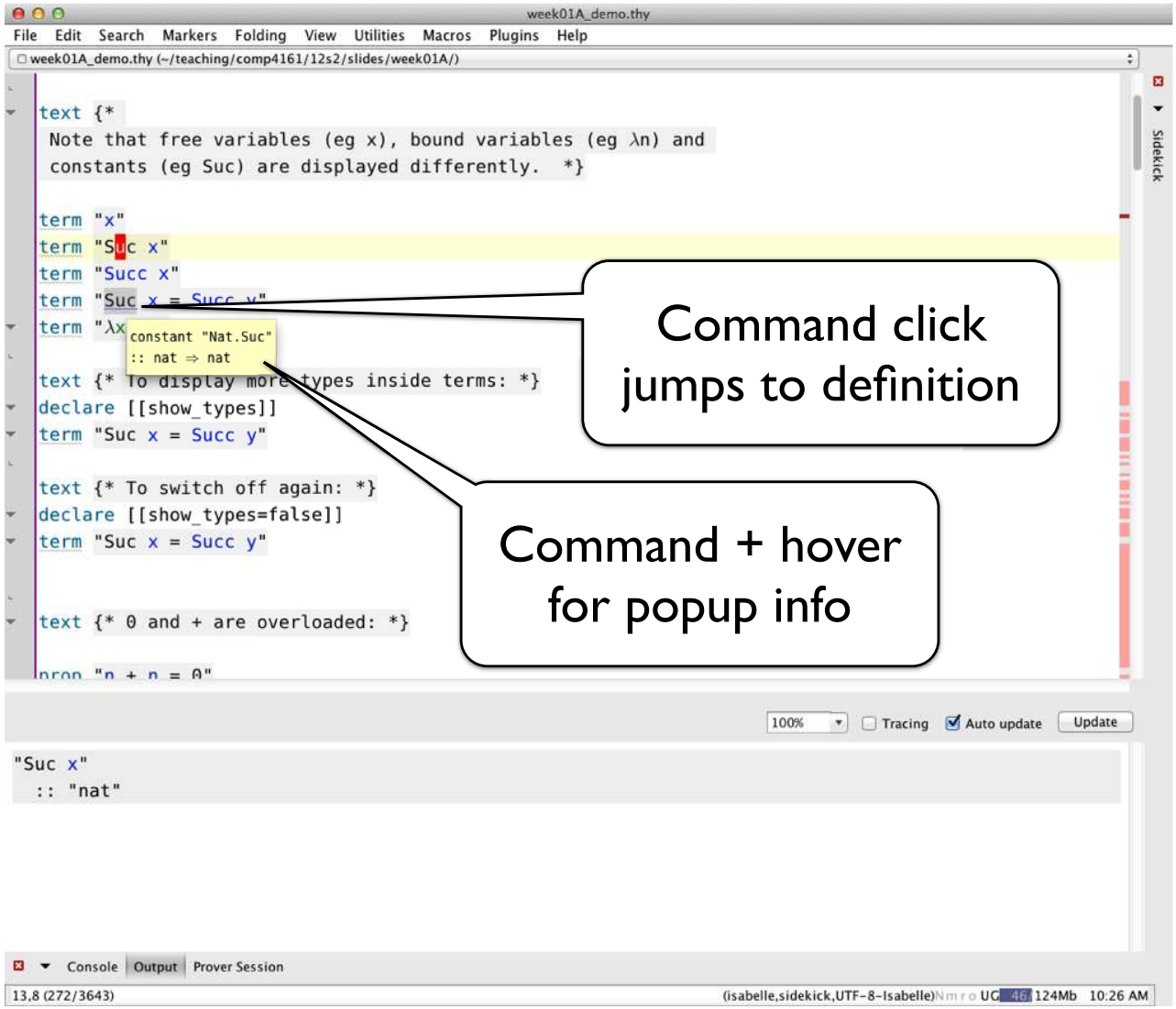
Copyright NICTA 2014, provided under Creative Commons Attribution License

jEdit/PIDE



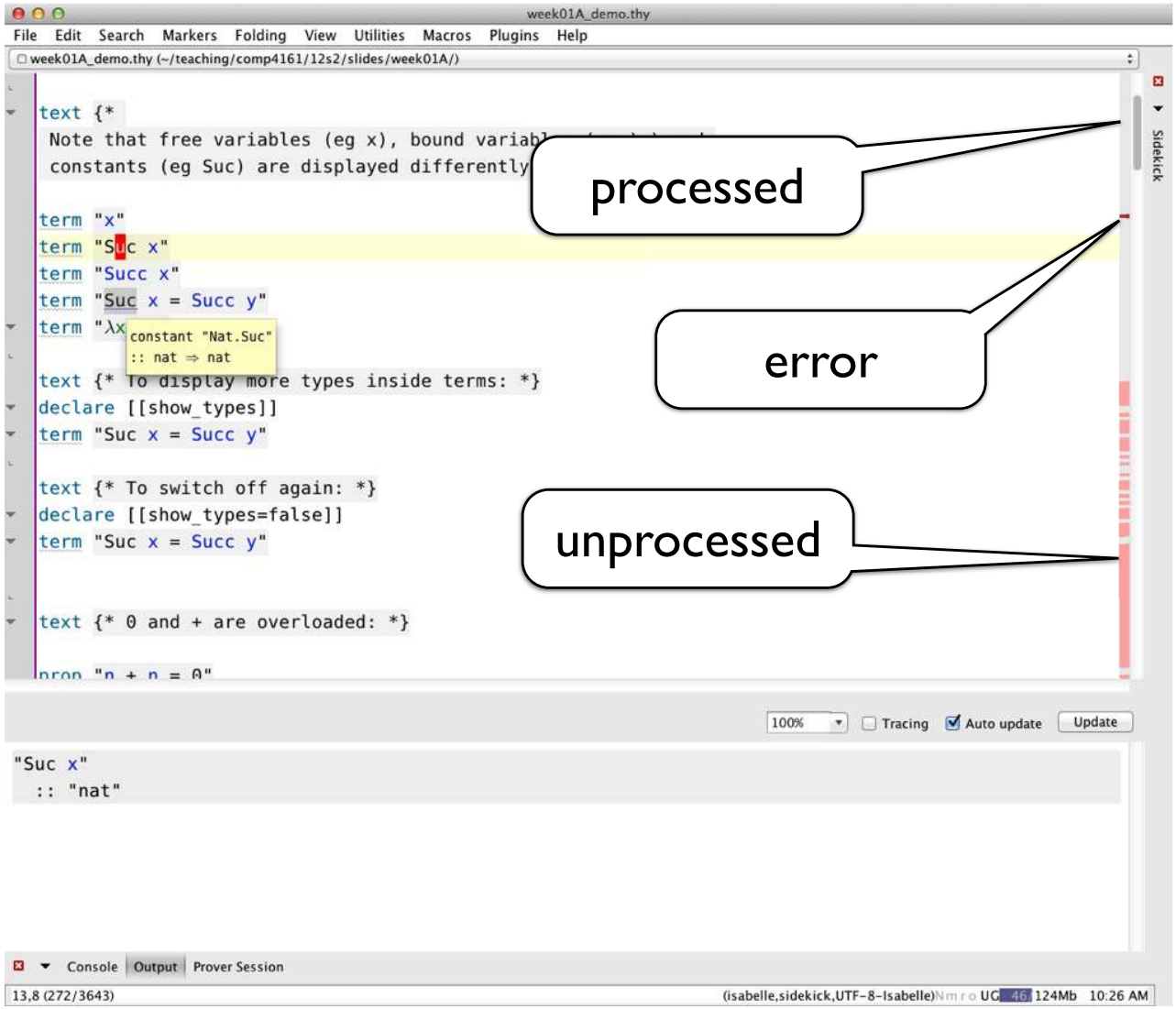
Copyright NICTA 2014, provided under Creative Commons Attribution License

jEdit/PIDE



Copyright NICTA 2014, provided under Creative Commons Attribution License

jEdit/PIDE



Copyright NICTA 2014, provided under Creative Commons Attribution License

DEMO

Copyright NICTA 2014, provided under Creative Commons Attribution License

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 "Suc(Suc x))"?
- What is the name of the theorem for associativity of addition of natural numbers in the library?

Copyright NICTA 2014, provided under Creative Commons Attribution License