

COMP 4161

NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

 $a = b = c = \dots$

Slide 1

Last time ...



- → fun, function
- → Well founded recursion

Content



Rough	timeline
-------	----------

→ Intro & motivation, getting started

[1]

→ Foundations & Principles

Lambda Calculus, natural deduction [2,3,4°]
 Higher Order Logic [5,6°,7]
 Term rewriting [8,9,10°]

→ Proof & Specification Techniques

Isar [11,12^d]
 Inductively defined sets, rule induction [13^c,15]
 Datatypes, recursion, induction [16,17^f,18,19]
 Calculational reasoning, mathematics style proofs [20]
 Hoare logic, proofs about programs [21^g,22,23]

Slide 3



CALCULATIONAL REASONING

Slide 4

1

Slide 2

2

^a a1 out; ^b a1 due; ^c a2 out; ^d a2 due; ^e session break; ^f a3 out; ^g a3 due

The Goal



$$\begin{split} x \cdot x^{-1} &= 1 \cdot (x \cdot x^{-1}) \\ &\dots &= 1 \cdot x \cdot x^{-1} \\ &\dots &= (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \\ &\dots &= (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \\ &\dots &= (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \\ &\dots &= (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \\ &\dots &= (x^{-1})^{-1} \cdot x^{-1} \\ &\dots &= 1 \end{split}$$

Can we do this in Isabelle?

- → Simplifier: too eager
- → Manual: difficult in apply style
- → Isar: with the methods we know, too verbose

Slide 5

Chains of equations



The Problem

$$\begin{array}{rcl}
a & = & b \\
\dots & = & c \\
\dots & = & d
\end{array}$$

shows a = d by transitivity of =

Each step usually nontrivial (requires own subproof)

Solution in Isar:

- → Keywords also and finally to delimit steps
- → ...: predefined schematic term variable, refers to right hand side of last expression
- → Automatic use of transitivity rules to connect steps

Slide 6

also/finally



 $\begin{array}{lll} \textbf{have} \ "t_0 = t_1" \ [\mathsf{proof}] & \mathsf{calculation} \ \mathsf{register} \\ \textbf{also} & "t_0 = t_1" \\ \textbf{have} \ "\dots = t_2" \ [\mathsf{proof}] \\ \textbf{also} & "t_0 = t_2" \\ \vdots & \vdots & \vdots \\ \textbf{also} & "t_0 = t_{n-1}" \\ \textbf{have} \ "\dots = t_n" \ [\mathsf{proof}] \end{array}$

finally

show P

— 'finally' pipes fact " $t_0 = t_n$ " into the proof

Slide 7

 $t_0 = t_n$

More about also



- \rightarrow Works for all combinations of =, \leq and <.
- → Uses all rules declared as [trans].
- → To view all combinations in Proof General:
 Isabelle/Isar → Show me → Transitivity rules

Designing [trans] Rules



have = " $l_1 \odot r_1$ " [proof] also have "... $\odot r_2$ " [proof] also

Anatomy of a [trans] rule:

- lacksquare Usual form: plain transitivity $[\![l_1\odot r_1;r_1\odot r_2]\!]\Longrightarrow l_1\odot r_2$
- ightharpoonup More general form: $[\![P\ l_1\ r_1; Q\ r_1\ r_2; A]\!] \Longrightarrow C\ l_1\ r_2$

Examples:

- ightharpoonup pure transitivity: $[\![a=b;b=c]\!] \Longrightarrow a=c$
- \rightarrow mixed: $[a \le b; b < c] \Longrightarrow a < c$
- \rightarrow substitution: $\llbracket P \ a; a = b \rrbracket \Longrightarrow P \ b$
- \rightarrow antisymmetry: $[a < b; b < a] \Longrightarrow P$

Slide 9



DEMO

Slide 10

HOL as programming language



We have

- → numbers, arithmetic
- → recursive datatypes
- → constant definitions, recursive functions
- → = a functional programming language
- → can be used to get fully verified programs

Executed using the simplifier. But:

- → slow, heavy-weight
- → does not run stand-alone (without Isabelle)

Slide 11

Generating code



Translate HOL functional programming concepts, i.e.

- → datatypes
- → function definitions
- → inductive predicates

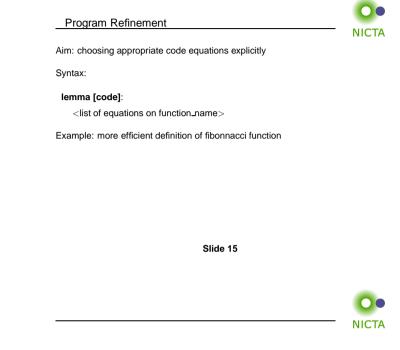
into a stand-alone code in:

- → SML
- → Ocaml
- → Haskell
- → Scala

Slide 12

Syntax	NUCTA
	NICIA
export_code ¡definition_names¿ in SML	
module_name < module_name > file " <file path="">"</file>	
export_code definition_names in Haskell	
module_name < module_name > file " <directory path="">"</directory>	
Takes a space-separated list of constants for which code shall be gen	erated.
Anything else needed for those is added implicitly Generates ML stud	ture.

Slide 13



DEMO

NICTA

DEMO

Slide 14 Slide 16

7

Inductive Predicates



Inductive specifications turned into equational ones

Example:

Syntax:

code_pred append .

Slide 17



DEMO

Slide 18

9

We have seen today ...



- → Calculations: also/finally
- → [trans]-rules
- → Code generation

Slide 19

10