COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray

\[ a = b = c = \ldots \]
Last time ...

- fun, function
- Well founded recursion
Content

→ Intro & motivation, getting started

→ 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]
  - Inductively defined sets, rule induction [13,15]
  - Datatypes, recursion, induction [16,17,18,19]
  - Calculational reasoning, mathematics style proofs [20]
  - Hoare logic, proofs about programs [21,22,23]

\(a\) a1 out; \(^b\) a1 due; \(^c\) a2 out; \(^d\) a2 due; \(^e\) session break; \(^f\) a3 out; \(^g\) a3 due
CALCULATIONAL REASONING
The Goal

\[ x \cdot x^{-1} = 1 \cdot (x \cdot x^{-1}) \]
\[ \ldots = 1 \cdot x \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot x^{-1} \cdot x \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot (x^{-1} \cdot x) \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot 1 \cdot x^{-1} \]
\[ \ldots = (x^{-1})^{-1} \cdot (1 \cdot x^{-1}) \]
\[ \ldots = (x^{-1})^{-1} \cdot x^{-1} \]
\[ \ldots = 1 \]

Can we do this in Isabelle?

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

The Problem

\[ a = b \]
\[ \ldots = c \]
\[ \ldots = d \]

shows \( a = d \) by transitivity of =

Each step usually nontrivial (requires own subproof)

Solution in Isar:

- Keywords **also** and **finally** to delimit steps
- \( \ldots \): predefined schematic term variable, refers to right hand side of last expression
- Automatic use of transitivity rules to connect steps
have "\(t_0 = t_1\)" [proof]

also

have "\(\ldots = t_2\)" [proof]

also

\[
\vdots
\]

also

have "\(\ldots = t_n\)" [proof]

finally

\[
t_0 = t_n
\]

calculation register

"\(t_0 = t_1\)"

"\(t_0 = t_2\)"

\[
\vdots
\]

"\(t_0 = t_{n-1}\)"

— 'finally' pipes fact "\(t_0 = t_n\)" into the proof
More about also

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

have = "l₁ ⊙ r₁" [proof]
also
have "... ⊙ r₂" [proof]
also

Anatomy of a [trans] rule:

→ Usual form: plain transitivity \([l₁ ⊙ r₁; r₁ ⊙ r₂] \implies l₁ ⊙ r₂\)
→ More general form: \([P l₁; Q r₁ r₂; A] \implies C l₁ r₂\)

Examples:

→ pure transitivity: \([a = b; b = c] \implies a = c\)
→ mixed: \([a ≤ b; b < c] \implies a < c\)
→ substitution: \([P a; a = b] \implies P b\)
→ antisymmetry: \([a < b; b < a] \implies P\)
→ monotonicity: \([a = f b; b < c; \land x y. x < y \implies f x < f y] \implies a < f c\)
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)
Generating code

Translate HOL functional programming concepts, i.e.

- datatypes
- function definitions
- inductive predicates

into a stand-alone code in:

- SML
- Ocaml
- Haskell
- Scala
Syntax

**export_code** `definition_names` **in** SML

```
module_name <module_name> file "<file path>"
```

**export_code** `definition_names` **in** Haskell

```
module_name <module_name> file "<directory path>"
```

Takes a space-separated list of constants for which code shall be generated.

Anything else needed for those is added implicitly. Generates ML structure.
DEMO
Program Refinement

Aim: choosing appropriate code equations explicitly

Syntax:

**lemma [code]:**

\(<\text{list of equations on function\_name}>\)

Example: more efficient definition of fibonacci function
DEMO
Inductive Predicates

Inductive specifications turned into equational ones

Example:

\[ \text{append } [ ] \text{ ys ys} \]
\[ \text{append } \text{xs ys zs } \Rightarrow \text{append } (x \# \text{xs }) \text{ ys } (x \# \text{zs }) \]

Syntax:

\texttt{code_pred append .}
DEMO
We have seen today ...

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