COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein, June Andronick, Toby Murray
Content

➜ Intro & motivation, getting started

➜ Foundations & Principles
  • Lambda Calculus, natural deduction
    \[2,3,4^a\]
  • Higher Order Logic
    \[5,6^b,7\]
  • Term rewriting
    \[8,9,10^c\]

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

\[^a\] a1 out; \[^b\] a1 due; \[^c\] a2 out; \[^d\] a2 due; \[^e\] session break; \[^f\] a3 out; \[^g\] a3 due
Last Time

- More Isar
- Fix/Obtain
- Moreover/Ultimately
- Mixing Proof Styles
SPECIFICATION TECHNIQUES: SETS
Sets in Isabelle

Type ’a set: sets over type ’a

→ \{\}, \{e_1, \ldots, e_n\}, \{x. P x\}
→ e \in A, \ A \subseteq B
→ A \cup B, \ A \cap B, \ A - B, \ -A
→ \bigcup x \in A. B x, \ \bigcap x \in A. B x, \ \bigcap A, \ \bigcup A
→ \{i..j\}
→ \text{insert :: } \alpha \Rightarrow \alpha \ \text{set} \Rightarrow \alpha \ \text{set}
→ f^* A \equiv \{y. \exists x \in A. y = f x\}
→ \ldots
Proofs about Sets

Natural deduction proofs:

- equalityI: \[ A \subseteq B; \ B \subseteq A \] \implies A = B
- subsetI: (\( \forall x. x \in A \implies x \in B \)) \implies A \subseteq B
- \ldots \quad (\text{see Tutorial})
Bounded Quantifiers

\[\forall x \in A. P x \equiv \forall x. x \in A \rightarrow P x\]
\[\exists x \in A. P x \equiv \exists x. x \in A \land P x\]

\[\text{ballI: } (\wedge x. x \in A \implies P x) \implies \forall x \in A. P x\]
\[\text{bspec: } [\forall x \in A. P x; x \in A] \implies P x\]

\[\text{bexI: } [P x; x \in A] \implies \exists x \in A. P x\]
\[\text{bexE: } [\exists x \in A. P x; \wedge x. [x \in A; P x] \implies Q] \implies Q\]
DEMO: SETS
The Three Basic Ways of Introducing Types

→ **typedef**: by name only

Example: **typedef** names
Introduces new type *names* without any further assumptions

→ **types**: by abbreviation

Example: **types** \( \alpha \) rel = "\( \alpha \Rightarrow \alpha \Rightarrow \text{bool} \)"
Introduces abbreviation *rel* for existing type \( \alpha \Rightarrow \alpha \Rightarrow \text{bool} \)
Type abbreviations are immediately expanded internally

→ **typedef**: by definition as a set

Example: **typedef** new_type = "\{some set\}" <proof>
Introduces a new type as a subset of an existing type.
The proof shows that the set on the rhs in non-empty.
How typedef works

new type

existing type

Rep

Abs
How typedef works

new type

existing type

Rep

Abs
Example: Pairs

\[(\alpha, \beta) \text{ Prod}\]

1. Pick existing type: \(\alpha \Rightarrow \beta \Rightarrow \text{bool}\)
2. Identify subset:
   \[(\alpha, \beta) \text{ Prod} = \{ f. \exists a b. f = \lambda (x :: \alpha) (y :: \beta). x = a \land y = b\}\]
3. We get from Isabelle:
   - functions Abs_Prod, Rep_Prod
   - both injective
   - Abs_Prod (Rep_Prod \(x\)) = \(x\)
4. We now can:
   - define constants Pair, fst, snd in terms of Abs_Prod and Rep_Prod
   - derive all characteristic theorems
   - forget about Rep/Abs, use characteristic theorems instead
DEMO: INTRODUCING NEW TYPES
INDUCTIVE DEFINITIONS
Example

\[ \langle \text{skip}, \sigma \rangle \rightarrow \sigma \]

\[ \langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto v] \]

\[ \langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma'' \]

\[ \langle c_1; c_2, \sigma \rangle \rightarrow \sigma'' \]

\[ [b]_\sigma = \text{False} \]

\[ \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma \]

\[ [b]_\sigma = \text{True} \]

\[ \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma'' \]

\[ \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma'' \]
What does this mean?

→ \( \langle c, \sigma \rangle \longrightarrow \sigma' \) fancy syntax for a relation \( (c, \sigma, \sigma') \in E \)

→ relations are sets: \( E :: \text{(com} \times \text{state} \times \text{state}) \text{ set} \)

→ the rules define a set inductively

But which set?
Simpler Example

\[
\begin{align*}
0 & \in \mathbb{N} \\
n & \in \mathbb{N} \\
n + 1 & \in \mathbb{N}
\end{align*}
\]

\[\Rightarrow\] \( \mathbb{N} \) is the set of natural numbers \( \mathbb{N} \)

\[\Rightarrow\] But why not the set of real numbers? \( 0 \in \mathbb{R}, n \in \mathbb{R} \implies n + 1 \in \mathbb{R} \)

\[\Rightarrow\] \( \mathbb{N} \) is the \textbf{smallest} set that is \textbf{consistent} with the rules.

Why the smallest set?

\[\Rightarrow\] Objective: \textbf{no junk}. Only what must be in \( X \) shall be in \( X \).

\[\Rightarrow\] Gives rise to a nice proof principle (rule induction)

\[\Rightarrow\] Alternative (greatest set) occasionally also useful: coinduction
Rule Induction

\[\begin{align*}
0 & \in N \\
n & \in N \\
n + 1 & \in N
\end{align*}\]

induces induction principle

\[
\left[ P \ 0; \ \bigwedge n. \ P \ n \implies P \ (n + 1) \right] \implies \forall x \in X. \ P \ x
\]
DEMO: INDUCTIVE DEFINITIONS
We have learned today ...

- Sets
- Type Definitions
- Inductive Definitions