COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Gerwin Klein
Formal Methods

\{P\} \ldots \{Q\}
Intro & motivation, getting started with Isabelle

Foundations & Principles
- Lambda Calculus
- Higher Order Logic, natural deduction
- Term rewriting

Proof & Specification Techniques
- Inductively defined sets, rule induction
- Datatypes, recursion, induction
- More recursion, Calculational reasoning
- Hoare logic, proofs about programs
- Locales, Presentation
LAST TIME

- Calculations: also/finally
- [trans]-rules
- Code generation
Command **find_theorems** (C-c C-f) finds combinations of:

- pattern: "_ + _ + _"
- Lhs of simp rules: **simp**: "_ * (_ + _)"
- intro/elim/dest on current goal
- lemma name: **name**: assoc
- exclusions thereof: **-name**: "HOL."

**Example:**

```
find_theorems dest -"hd" name: "List."
```

finds all theorems in the current context that

- match the goal as dest rule,
- do not contain the constant "hd"
- are in the List theory (name starts with "List.")
Can define local constant in Isar proof context:

proof


    define "f ≡ big term"
    have "g = f x" . . .

like definition, not automatically unfolded (f_def)
different to let ?f = "big term"

Also available in lemma statement:

lemma . . .:
    fixes . . .
    assumes . . .
    defines . . .
    shows . . .
A CRASH COURSE IN SEMANTICS
Commands:

```
datatype com = SKIP
| Assign loc aexp (_ := _)
| Semi com com (_; _)
| Cond bexp com com (IF _ THEN _ ELSE _)
| While bexp com (WHILE _ DO _ OD)
```

```
types loc = string
types state = loc ⇒ nat

types aexp = state ⇒ nat
types bexp = state ⇒ bool
```
Usual syntax:

\[ B := 1; \]
\[ \text{WHILE } A \neq 0 \text{ DO} \]
\[ B := B \ast A; \]
\[ A := A - 1 \]
\[ \text{OD} \]

Expressions are functions from state to bool or nat:

\[ B := (\lambda \sigma. 1); \]
\[ \text{WHILE } (\lambda \sigma. \sigma A \neq 0) \text{ DO} \]
\[ B := (\lambda \sigma. \sigma B \ast \sigma A); \]
\[ A := (\lambda \sigma. \sigma A - 1) \]
\[ \text{OD} \]
So far we have defined:

- **Syntax** of commands and expressions
- **State** of programs (function from variables to values)

**Now we need:** the meaning (semantics) of programs

**How to define execution of a program?**

- A wide field of its own
- Some choices:
  - Operational (inductive relations, big step, small step)
  - Denotational (programs as functions on states, state transformers)
  - Axiomatic (pre-/post conditions, Hoare logic)
\[ \langle \text{SKIP}, \sigma \rangle \rightarrow \sigma \]

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

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

\[ b \sigma = \text{True} \quad \langle c_1, \sigma \rangle \rightarrow \sigma' \]
\[ \langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle \rightarrow \sigma' \]

\[ b \sigma = \text{False} \quad \langle c_2, \sigma \rangle \rightarrow \sigma' \]
\[ \langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle \rightarrow \sigma' \]
\[
\begin{align*}
\text{b } \sigma &= \text{False} \\
\langle \text{WHILE b DO c OD, } \sigma \rangle &\rightarrow \sigma
\end{align*}
\]

\[
\begin{align*}
\text{b } \sigma &= \text{True} \\
\langle c, \sigma \rangle &\rightarrow \sigma' \\
\langle \text{WHILE b DO c OD, } \sigma' \rangle &\rightarrow \sigma''
\end{align*}
\]

\[
\langle \text{WHILE b DO c OD, } \sigma \rangle &\rightarrow \sigma''
\]
Demo: The Definitions in Isabelle
Now we know:

→ What programs are: Syntax
→ On what they work: State
→ How they work: Semantics

So we can prove properties about programs

Example:
Show that example program from slide 8 implements the factorial.

\[
\text{lemma } \langle \text{factorial}, \sigma \rangle \longrightarrow \sigma' \implies \sigma'B = \text{fac } (\sigma A)
\]

(where \( \text{fac } 0 = 0 \), \( \text{fac } (\text{Suc } n) = (\text{Suc } n) \ast \text{fac } n \))
DEMO: EXAMPLE PROOF
Induction needed for each loop

Is there something easier?
Idea: describe meaning of program by pre/post conditions

Examples:
\{\text{True}\} \quad x := 2 \quad \{x = 2\}
\{y = 2\} \quad x := 21 \times y \quad \{x = 42\}

\{x = n\} \quad \text{IF } y < 0 \text{ THEN } x := x + y \text{ ELSE } x := x - y \quad \{x = n - |y|\}

\{A = n\} \quad \text{factorial} \quad \{B = \text{fac } n\}

Proofs: have rules that directly work on such triples
Meaning of a Hoare-Triple

\{P\} \ c \ \{Q\}

What are the assertions \(P\) and \(Q\)?

→ Here: again functions from state to bool (shallow embedding of assertions)
→ Other choice: syntax and semantics for assertions (deep embedding)

What does \{P\} \ c \ \{Q\} mean?

Partial Correctness:
\[\models \{P\} \ c \ \{Q\} \ \equiv \ (\forall \sigma \ \sigma'. P\ \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \implies Q\ \sigma')\]

Total Correctness:
\[\models \{P\} \ c \ \{Q\} \ \equiv \ (\forall \sigma. P\ \sigma \implies \exists \sigma'. \langle c, \sigma \rangle \rightarrow \sigma' \land Q\ \sigma')\]

This lecture: partial correctness only (easier)
**Hoare Rules**

\[
\begin{align*}
\{P\} & \text{ SKIP } \{P\} & \{P[x \leftrightarrow e]\} & x := e & \{P\} \\
\{P\} & c_1 \{R\} & \{R\} & c_2 \{Q\} & \{P\} & c_1; c_2 & \{Q\} \\
\{P \land b\} & c_1 \{Q\} & \{P \land \neg b\} & c_2 \{Q\} & \{P\} & \text{ IF } b \text{ THEN } c_1 \text{ ELSE } c_2 & \{Q\} \\
\{P \land b\} & c \{P\} & P \land \neg b \implies Q & \{P\} & \text{ WHILE } b \text{ DO } c \text{ OD} & \{Q\} \\
P \implies P' & \{P'\} & c \{Q'\} & Q' \implies Q & \{P\} & c \{Q\} 
\end{align*}
\]
\[ \vdash \{P\} \text{ SKIP} \{P\} \quad \vdash \{ \lambda \sigma. P (\sigma(x := e \sigma)) \} \quad x := e \quad \{P\} \]

\[ \vdash \{P\} \ c_1 \ \{R\} \quad \vdash \{R\} \ c_2 \ \{Q\} \]

\[ \vdash \{P\} \ c_1 \ c_2 \ \{Q\} \]

\[ \vdash \{ \lambda \sigma. P \ \sigma \land b \ \sigma \} \ c_1 \ \{R\} \quad \vdash \{ \lambda \sigma. P \ \sigma \land \neg b \ \sigma \} \ c_2 \ \{Q\} \]

\[ \vdash \{P\} \quad \text{IF} \ b \ \text{THEN} \ c_1 \ \text{ELSE} \ c_2 \ \{Q\} \]

\[ \vdash \{ \lambda \sigma. P \ \sigma \land b \ \sigma \} \ c \ \{P\} \quad \land \ \sigma. P \ \sigma \land \neg b \ \sigma \implies Q \ \sigma \]

\[ \vdash \{P\} \quad \text{WHILE} \ b \ \text{DO} \ c \ \text{OD} \ \{Q\} \]

\[ \land \ \sigma. P \ \sigma \implies P' \ \sigma \quad \vdash \{P'\} \ c \ \{Q'\} \quad \land \ \sigma. Q' \ \sigma \implies Q \ \sigma \]

\[ \vdash \{P\} \ c \ \{Q\} \]
Are the Rules Correct?

Soundness: \(\vdash \{P\} \ c \ \{Q\} \implies \models \{P\} \ c \ \{Q\}\)

Proof: by rule induction on \(\vdash \{P\} \ c \ \{Q\}\)

Demo: Hoare Logic in Isabelle