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

- Code generation
- Syntax of a simple imperative language
- Operational semantics
- Program proof on operational semantics
Proofs about Programs

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 last lecture implements the factorial.

\[
\text{lemma } \langle \text{factorial}, \sigma \rangle \rightarrow \sigma' \iff \sigma' B = \text{fac } (\sigma A) \\
\text{(where } \text{fac } 0 = 0, \text{ fac } (\text{Suc } n) = (\text{Suc } n) \ast \text{fac } n)\]
Induction needed for each loop

Is there something easier?
**Floyd/Hoare**

**Idea:** describe meaning of program by pre/post conditions

**Examples:**

\{
  \text{True}
\} \quad x := 2 \quad \{ x = 2 \}

\{
  y = 2
\} \quad x := 21 \ast 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:
\[\models \{P\} \ c \ \{Q\} \equiv (\forall \sigma \ \sigma'. \ P \ \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \rightarrow Q \ \sigma')\]

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

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

\[
\begin{align*}
\{P\} & \text{ SKIP } \{P\} & \{P[x \leftarrow e]\} & x := e & \{P\} \\
\{P\} & c_1 \{R\} & {R} & c_2 \{Q\} & \{P\} & c_1; c_2 & \{Q\} \\
\{P \land b\} & c_1 \{Q\} & \{P \land \neg b\} & c_2 \{Q\} & \{P\} & \text{ IF } b \text{ THEN } c_1 \text{ ELSE } c_2 & \{Q\} \\
\{P \land b\} & c \{P\} & P \land \neg b & \Rightarrow Q & \{P\} & \text{ WHILE } b \text{ DO } c \text{ OD } & \{Q\} \\
P & \Rightarrow P' & \{P'\} & c \{Q'\} & Q' & \Rightarrow Q & \{P\} & c & \{Q\}
\end{align*}
\]
Hoare Rules

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

**Automation?**

**Problem:** While – need creativity to find right (invariant) $P$

**Solution:**
- annotate program with invariants
- then, Hoare rules can be applied automatically

**Example:**

\[
\begin{align*}
\{M = 0 \land N = 0\} \\
\text{WHILE } M \neq a \text{ INV } \{N = M \times b\} \text{ DO } &N := N + b; \ M := M + 1 \text{ OD} \\
\{N = a \times b\}
\end{align*}
\]
Weakest Preconditions

\[ \text{pre } c \ Q = \text{weakest } P \text{ such that } \{P\} \ c \ \{Q\} \]

With annotated invariants, easy to get:

\[
\begin{align*}
\text{pre } \text{SKIP } Q &= Q \\
\text{pre } (x := a) \ Q &= \lambda \sigma. \ Q(\sigma(x := a \sigma)) \\
\text{pre } (c_1; c_2) \ Q &= \text{pre } c_1 (\text{pre } c_2 \ Q) \\
\text{pre } (\text{IF } b \ \text{THEN } c_1 \ \text{ELSE } c_2) \ Q &= \lambda \sigma. \ (b \rightarrow \text{pre } c_1 \ Q \ \sigma) \land \\
&\quad (\neg b \rightarrow \text{pre } c_2 \ Q \ \sigma) \\
\text{pre } (\text{WHILE } b \ \text{INV } I \ \text{DO } c \ \text{OD}) \ Q &= I
\end{align*}
\]
Verification Conditions

\{\text{pre } c \ Q\} \ c \ \{Q\} \text{ only true under certain conditions}

These are called verification conditions $vc \ c \ Q$:

\begin{align*}
vc \ \text{SKIP } Q & = \text{ True} \\
vc \ (x := a) \ Q & = \text{ True} \\
vc \ (c_1; c_2) \ Q & = vc \ c_2 \ Q \land (vc \ c_1 (\text{pre } c_2 \ Q)) \\
vc \ (\text{IF } b \ \text{THEN } c_1 \ \text{ELSE } c_2) \ Q & = vc \ c_1 \ Q \land vc \ c_2 \ Q \\
vc \ (\text{WHILE } b \ \text{INV } I \ \text{DO } c \ \text{OD}) \ Q & = (\forall \sigma. I \sigma \land b \sigma \to \text{pre } c \ I \sigma) \land \\
& \quad \quad (\forall \sigma. I \sigma \land \neg b \sigma \to Q \sigma) \land \\
& \quad \quad vc \ c \ I
\end{align*}

$$vc \ c \ Q \land (\text{pre } c \ Q \Rightarrow P) \Rightarrow \{P\} \ c \ \{Q\}$$
Syntax Tricks

→ $x := \lambda \sigma. 1$ instead of $x := 1$ sucks

→ $\{\lambda \sigma. \sigma x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

Choices:

→ declare program variables with each Hoare triple
   - nice, usual syntax
   - works well if you state full program and only use vcg

→ separate program variables from Hoare triple (use extensible records), indicate usage as function syntactically
   - more syntactic overhead
   - program pieces compose nicely
Records in Isabelle

Records are a tuples with named components

Example:

```
record A =
  a :: nat
  b :: int
```

→ Selectors: \( a :: A \Rightarrow \text{nat}, \ b :: A \Rightarrow \text{int}, \ a \ r = \text{Suc} \ 0 \)
→ Constructors: \( (| \ a = \text{Suc} \ 0, \ b = -1 |) \)
→ Update: \( r(| a := \text{Suc} \ 0 |) \)

Records are extensible:

```
record B = A +
  c :: nat list
```

\( (| a = \text{Suc} \ 0, \ b = -1, \ c = [0, 0] |) \)
Arrays

Depending on language, model arrays as functions:

- Array access = function application:
  \[ a[i] = a \_i \]

- Array update = function update:
  \[ a[i] := v = a := a(i:= v) \]

Use lists to express length:

- Array access = nth:
  \[ a[i] = a \_! i \]

- Array update = list update:
  \[ a[i] := v = a := a[i:= v] \]

- Array length = list length:
  \[ a.length = length a \]
Choice 1

datatype ref = Ref int | Null

types heap = int ⇒ val

datatype val = Int int | Bool bool | Struct x int int bool | ... 

⇒ hp :: heap, p :: ref
⇒ Pointer access: *p = the_Int (hp (the_addr p))
⇒ Pointer update: *p := v = hp := hp ((the_addr p) := v)

⇒ a bit klunky
⇒ gets even worse with structs
⇒ lots of value extraction (the_Int) in spec and program
Choice 2 (Burstall ’72, Bornat ’00)

struct with next pointer and element

\[
\text{datatype} \quad \text{ref} \quad = \quad \text{Ref int} \mid \text{Null} \\
\text{types} \quad \text{next_hp} \quad = \quad \text{int} \Rightarrow \text{ref} \\
\text{types} \quad \text{elem_hp} \quad = \quad \text{int} \Rightarrow \text{int}
\]

\[
\rightarrow \quad \text{next :: next_hp, elem :: elem_hp, p :: ref} \\
\rightarrow \quad \text{Pointer access: } p \rightarrow \text{next} \quad = \quad \text{next (the_addr p)} \\
\rightarrow \quad \text{Pointer update: } p \rightarrow \text{next} := v \quad = \quad \text{next} := \text{next ((the_addr p) := v)}
\]

\[
\rightarrow \quad \text{a separate heap for each struct field} \\
\rightarrow \quad \text{buys you } p \rightarrow \text{next} \neq p \rightarrow \text{elem automatically (aliasing)} \\
\rightarrow \quad \text{still assumes type safe language}
\]
DEMO
WE HAVE SEEN TODAY …

- Hoare logic rules
- Soundness of Hoare logic
- Verification conditions
- Example program proofs
- Arrays, pointers