Hoare style program verification

Abhik Roychoudhury
CS 5219
National University of Singapore

Deduction
- Proving properties of programs by hand
  - Proving that a factorial program computes the factorial function at the end
- Sounds too difficult
  - What proof rules to apply?
  - In what order to apply them? (strategies)
- Automation
  - Proof rules OK, Proof Strategies hard!

Theorem Provers
- A Proof Assistant to help you prove properties by hand
  - More powerful than model checking!
  - MC only employs search
  - Less automated of course!!
  - Directly proving properties of programs
    - No abstractions to build finite-state models
- More powerful than model checking!
- Non-mechanized.
- Consider sequential programs in this lecture
- Can extend to develop proof rules for multi-threaded programs.

Remarks
- The approach of developing proof rules for reasoning about language constructs is radically different from model checking
  - Reason about programs (not transition systems)
  - Non-mechanized.
  - Notion of distinguished control locations ingrained
  - Reason about pre- and post-conditions holding before and after execution of a block of code.
- Consider sequential programs in this lecture
- Can extend to develop proof rules for multi-threaded programs.

Hoare triple
- \{\text{Pre}\} \ P \ \{\text{Post}\}
  - If \ P \ is run from a state where \ Pre \ holds and \ P \ terminates, then \ Post \ holds in the end-state [Partial correctness]
  - If \ P \ is run from a state where \ Pre \ holds, then \ P \ terminates and \ Post \ holds in the end-state [Total correctness]
  - A Hoare triple involving \ P \ is a specification about \ P.
- Note: First attempt to systematically reason about programs by Prof. C.A.R. Hoare in 1960’s.

Trivial example
- Say \ P \ is \ while \ true \ do \ x = 0 \ endwhile
  - \ P \ is partially correct w.r.t. any specification of the form \ \{\text{Pre}\} \ P \ \{\text{Post}\}
  - \ P \ is not totally correct w.r.t. any specification of the form \ \{\text{Pre}\} \ P \ \{\text{Post}\}
  - We will develop a proof system for reasoning about partial correctness
    - First step to reasoning about total correctness
Notations

- $\vdash_{\text{par}} \{ \text{Pre} \} P \{ \text{Post} \}$
  - The Hoare triple can be shown to be partially correct in our proof system
- $\vdash_{\text{tot}} \{ \text{Pre} \} P \{ \text{Post} \}$
  - The Hoare triple is partially correct.
- $\vdash_{\text{tot}} \{ \text{Pre} \} P \{ \text{Post} \}, \vdash_{\text{tot}} \{ \text{Pre} \} P \{ \text{Post} \}$
  - Similar
  - Standard notions of soundness/completeness

Factorial program

- $\{ x \geq 0 \}$
  - /* x is input */
  - $y = 1$; $z = 0$;
  - while ($z \neq x$) do
    - $z = z + 1$;
    - $y = y \times z$;
    - endwhile
  - $\{ y = x! \}$

- $\{ x \geq 0 \}$
  - /* x is input */
  - $y = 1$;
  - while ($x \neq 0$) do
    - $y = y \times x$;
    - $x = x - 1$;
  - endwhile
  - $\{ ??? \}$

The problem

- $x$ was destructively updated in Program2
  - In the end-state, we cannot say $y = x!$
  - To state correctness conditions, not enough to use program variables
  - Need to remember the original value of $x$
  - $\{ x = x_0 \land x \geq 0 \}$ Program2 $\{ y = x_0! \}$
  - $x_0$ is a universally quantified logical variable.

Logical variables

- $\{ x = x_0 \land x \geq 0 \}$ Program2 $\{ y = x_0! \}$
  - For all $x_0$, if $x = x_0$ and $x \geq 0$ and we run Program2 such that it terminates, we will have $y = x_0!$ in the end state.
  - These variables appear only in the logical formulae of pre- and post-conditions.
  - Never appear in the program being verified.
  - We now present the proof rules of our proof system.

Proof Rules

Premises

Conclusion

Both premises and conclusion are Hoare triples.

If premises specify properties about programs $C_1, C_2, \ldots, C_n$

-- the conclusion specifies a property about a bigger program $C$ typically containing $C_1, C_2, \ldots, C_n$

Rule for Assignment

$\{ \psi [x \rightarrow E] \} x = E (\psi)$

No premises in this rule.

To prove $\psi$ after the assignment, $\psi [x \rightarrow E]$ should hold before the assignment.
Why not forwards?

- \{ \varphi \} x = E \{ \psi \}
  - How to define \psi in terms of \varphi?
  - Cannot be achieved mechanically in general
  - The backwards formulation of the rule allows deducing Hoare triple by mechanically substituting \( x \).
  - Instead define \varphi in terms of \psi

Sequential Composition

\[
\{ \varphi \} C_1 \{ \psi \} \quad \{ \psi \} C_2 \{ \psi \} \\
\{ \varphi \} C_1 : C_2 \{ \psi \}
\]

Need assertion for end-state of \( C_1 \) and begin-state of \( C_2 \).

If-statement

\[
\{ \varphi \land b \} C_1 \{ \psi \} \quad \{ \varphi \land \neg b \} C_2 \{ \psi \} \\
\{ \varphi \} \text{ if } b \text{ then } C_1 \text{ else } C_2 \{ \psi \}
\]

Involves a case-split.

Pre-condition typically does not say anything about \( b \)

Needs to augmented with truth/falsehood of \( b \).

While statement

\[
\{ \varphi \land b \} C \{ \psi \} \\
\{ \varphi \} \text{ while } b \text{ do } c \{ \psi \land \neg b \}
\]

\( \psi \) is the loop invariant.

Rule for partial correctness (number of times the loop executes/ termination is not known/ not guaranteed).

Implications

\[
\psi \Rightarrow \varphi \quad \{ \varphi \} C \{ \psi \} \quad \varphi \Rightarrow \psi' \\
\{ \psi' \} C \{ \psi \}
\]

1. Strengthening the pre-condition
2. Weakening the post-condition

Why do we need this rule?

Example 1

- \{ y < 2 \} y = y + 1 \{ y < 5 \}

**Proof:**
- \{ y < 2 \}
- \{ y + 1 < 5 \} *implication rule*
- \( y = y + 1 \)
- \{ y < 5 \} *assignment rule*
Example 2

\[ \{\text{true}\} z = x; \quad z = z + y; \quad u = z \{u = x + y\} \]

Proof:

- \{true\}
- \{x + y = x + y\}
- \{z = x\}
- \{z + y = x + y\}
- \{z = z + y\}
- \{z = x + y\}
- \{u = z\}
- \{u = x + y\}

Push up the assertions starting from the post-condition of the code fragment being verified.

Example 3

\[ \{\text{true}\} \quad \text{if} \ (x > y) \quad z = y \quad \text{else} \quad z = x \{z = \min(x,y)\} \]

Proof:

- \{x = \min(x,y)\} \quad z = x \{z = \min(x,y)\}
- \{y = \min(x,y)\} \quad z = y \{z = \min(x,y)\}

How to combine these triples using the rule for if-statements in our proof system?

- Use the rule of implications

Reasoning about loops

- To prove \( \{\psi\} \) while \( b \) do \( c \) \( \{\psi\} \)
- We must
  - Find a loop invariant \( \eta \) i.e. \( \eta \land b \) \( c \) \( \eta \)
  - this means
    - \( \{\eta\} \) while \( b \) do \( c \) \( \eta \land \neg b \)
    - Show that \( \psi \Rightarrow \eta \)
    - Show that \( \eta \land \neg b \Rightarrow \psi \)
  - Use rule of impl. to prove \( \{\psi\} \) while \( b \) do \( c \) \( \{\psi\} \)

Guessing invariants

- Synthesis involves human ingenuity
- No unique inv. for a given loop
- The formula True is an invariant for any loop
- But our inv. \( \eta \) should satisfy
  - \( \varphi \Rightarrow \eta \)
  - \( (\eta \land \neg b) \Rightarrow \psi \)
- Usually choose invariants which capture relationships between variables whose values are modified at each iteration.
Verifying invariants

\[
\begin{align*}
\{\varphi\} & \quad \text{Implication} \\
\{\eta\} & \quad \text{while } b \text{ do} \\
\{\eta \land b\} & \quad \text{Implication} \\
\{\eta_1\} & \quad \text{C} \\
\{\eta \land \neg b\} & \quad \text{Implication} \\
\{\psi\} & \\
\end{align*}
\]

Push up based on structure of C

1. \(\eta \land \neg b \Rightarrow \psi\)
2. \(\eta \land b \Rightarrow \eta_1\)
3. \(\varphi \Rightarrow \eta\)

Factorial program

\[
\begin{align*}
\{\text{true}\} \\
\text{guess the loop invariant} \\
\{y = 1; z = 0;\} \\
\{y = z!\} \\
\text{while } (z \neq x) \text{ do} \\
\quad z = z + 1; y = y \cdot z \\
\text{endwhile} \\
\{y = x!\}
\end{align*}
\]

Checking the post-loop states

\[
\begin{align*}
\{\text{true}\} \\
\text{while } (z \neq x) \text{ do} \\
\quad z = z + 1; y = y \cdot z \\
\text{endwhile} \\
\{y = x!\}
\end{align*}
\]

Verifying the invariant

\[
\begin{align*}
\{\text{true}\} \\
\text{while } (z \neq x) \text{ do} \\
\quad z = z + 1; y = y \cdot z \\
\text{endwhile} \\
\text{need to prove this Hoare triple}
\end{align*}
\]
So-far

...  
{ y = z! }  
while (z = x) do  
{ y = z! \land z = x }  
z = z + 1 ; y = y*z  
{ y = z! }  
endwhile  
{ y = x! }  

Does y = z! hold before the while loop?

Checking pre-loop states

{true}  
{ 1 = 0! }  
y = 1;  
{ y = 0! }  
z = 0;  
{ y = z! }  

Need to prove this Hoare triple

Checking pre-loop states

{true}  
Implication  
{ t = 0! }  
y = 1;  
Assignment  
{ y = 0! }  
z = 0;  
Assignment  
{ y = z! }  

Proof structure

"{true} Factorial-Program \{ y = x! \}"

1. To prove y = x! at the end of the loop, we first guess a loop invariant  
   \{ y = z! \} is our choice
2. Can the choice of loop invariant in step 1 ensure y = x! at the end of the loop?  
   Yes
3. Verify that y = z! is indeed a loop invariant  
   This is the premise of the while rule

Proof structure

4. Verify that the loop invariant holds before the loop.
   Checking pre-loop states  
   Steps 3 and 4 constitute a proof of the invariant by induction on # iterations
   Step 3: induction step
   Step 4: base case of the proof
   The loop invariant itself is the ind. Hypothesis, no strengthening involved in this proof.

Choice of Loop Invariant

The loop invariant must be strong enough to be "proved" an invariant.
- The while rule is essentially accomplishing induction on # of loop iterations.
- Often guided by the choice of the post-condition after the loop
  - Our post-condition was y = x!
  - Since z = x at loop exit and z is modified at every loop iteration, choose y = z! as invariant.
Proving total correctness

- Our proof system only shows partial correctness of triples $\{\phi\} P \langle\psi\}\$
- To prove total correctness
  - Need to prove termination
  - Only the proof rule for while statement needs to change.
  - To prove termination
    - Find a non-negative integer quantity which decreases in every iteration (call it variant)

Finding variant

- $a = x; y = 1$
- while $(a > 0)$ do
  - $y = y*a; a = a-1$
- endwhile

  - Trivial to find the variant
    - $a$ in this case

Finding variant

- $y = 1; z = 0$
- while $(z != x)$ do
  - $z = z + 1; y = y*z$
- endwhile

  - Variant is $x - z$ (lifted from loop guard here)
  - In general, finding variant cannot be automated even if the loop is guaranteed to terminate.

New Proof Rule

$$\{ \eta \land b \land (E = E_0 \geq 0) \} \quad \text{C} \quad \{ \eta \land (E_0 > E \geq 0) \}$$

$E$ is the variant.
If it is $E_0$ before the loop, it strictly decreases but remains non-negative.
Of course $E$ should be non-negative before the loop starts.

Factorial program

- $\{ x \geq 0 \}$
- $y = 1; z = 0$
- while $(z != x)$ do
  - $z = z + 1$
- $y = y*z$
- endwhile

- $\{ y = x! \}$

Use the variant $x - z$ to prove termination
Use the loop invariant $y = z!$ as before for proving partial correctness

Reasoning about the loop

- From the conclusion of the while rule (total correctness)
- How to show the premise?
Reasoning about an iteration

\{y = z! \land x - z \geq 0\}
while (x != z) do
\{y = z! \land x = z \land (x - z = E0 > 0)\}
z := z + 1; \ y := y \times z;
\{y = z! \land (E0 > x - z \geq 0)\}
endwhile;
\{y = z! \land x = z\}
\{y = x!\}

This triple is the premise of the while rule.

Implication: Check it!

Need to prove this Hoare triple

\{x \geq 0\}
y = 1; \ z = 0;
\{y = 0! \land x - 0 \geq 0\}
z = 1;
\{y = 0! \land x - 1 \geq 0\}
while (x != z) do
\{y = z! \land x = z \land (x - z = E0 > 0)\}
z := z + 1; \ y := y \times z;
\{y = z! \land (E0 > x - z \geq 0)\}
endwhile;
\{y = z!\}
\{y = x!\}

Implication

x \geq 0 must hold at the initial state of the program for the program to terminate.
This fact is used here in the overall proof.

Finally, Program Refinement

\{\varphi\} Prog \{\psi\}

- Infers properties about pre- and post-states of a program
- In the flavor of program verification
- Instead, you can treat \((\varphi, \psi)\) as a specification
- Correct by construction program synthesis from given pre- and post-conditions.
- Many choices of implementation possible!
  - Use the specifications to guide the implementation instead of checking the implementation against the specification post-mortem.

Reading Material

- Chapter 4 of
  - Logic in Computer Science
    By Michael R. A. Huth and Mark D. Ryan
    QA76.9 Log.Hu
    Check E-reserves of IVLE
    - One download only!
  - Additional reading
    - See IVLE Lesson Plan

CS5219 2007-08 by Abhik