CS 292C Computer-Aided Reasoning for Software

Lecture 11: Reasoning about Programs using Hoare Logic II

Yu Feng
Spring 2022

Summary of previous lecture

- Reasoning about (partial) correctness with Hoare Logic

Simple Imperative Programming Language

Expression E

- $Z \mid V \mid E_1 + E_2 \mid E_1 * E_2$

Conditional C

$\text{True} \mid \text{False} \mid E_1 = E_2 \mid E_1 \leq E_2$

A minimalist programming language for demonstrating key features of Hoare logic.

Statement S

- skip (Skip)
- abort. (Abort)
- $V := E$ (Assignment)
- $S_1; S_2$. (Composition)
- **if** C **then** S_1 **else** S_2 (If)
- **while** C **do** S (While)

Hoare logic rules

$$\frac{}{\vdash \{P\}\ \mathrm{Skip}\ \{P\}}$$

$$\frac{\vdash \{P\}\ S_1\ \{R\} \qquad \vdash \{R\}\ S_2\ \{Q\}}{\vdash \{P\}\ S_1; S_2\ \{Q\}}$$

$$\frac{}{\vdash \{\mathrm{true}\}\ \mathrm{abort}\ \{\mathrm{false}\}}$$

$$\frac{\vdash \{P \wedge C\}\ S_1\ \{Q\} \qquad \vdash \{P \wedge \neg C\}\ S_2\ \{Q\}}{\vdash \{P\}\ \textbf{if}\ C\ \textbf{then}\ \mathbf{S_1}\ \textbf{else}\ \mathbf{S_2}\ \{Q\}}$$

$$\frac{}{\vdash \{Q[E/x]\}\ x := E\ \{Q\}}$$

$$\frac{\vdash \{I \wedge C\}\ S\ \{I\}}{\vdash \{I\}\ \textbf{while}\ C\ \textbf{do}\ S\ \{I \wedge \neg C\}}$$

$$\frac{\vdash \{P_1\}\ S\ \{Q_1\} \quad P \Rightarrow P_1 \quad Q_1 \Rightarrow Q}{\vdash \{P\}\ S\ \{Q\}}$$

Proof rule for assignment

$$\frac{\qquad}{\vdash \{Q[E/x]\}\ x := E\ \{Q\}}$$

- To prove Q holds after assignment x := E , sufficient to show that Q with E substituted for x holds before the assignment. $\boxed{?}$

- Using this rule, which of these are provable?

  - {y=4} x:=4 {y=x} 

  - {x+1=n} x:=x+1 {x=n} 

  - {y=x} y:=2 {y=x} 

  - {z = 2} y := x {y = x} 

Precondition strengthening

- Is the Hoare triple {z = 2} y := x {y = x} valid?
- Is it provable using our assignment rule?

$$\frac{\vdash \{P_1\}\ S\ \{Q\} \quad P \Rightarrow P_1}{\vdash \{P\}\ S\ \{Q\}}$$

Precondition strengthening

$$\dfrac{\dfrac{\vdash \{y = x[x/y]\} y = x \{y = x\}}{\vdash \{true\} y := x \{y = x\}} \qquad z = 2 \Rightarrow true}{\vdash \{z = 2\} y := x \{y = x\}}$$

Postcondition weakening

$$\frac{\vdash \{P\}\ S\ \{Q_1\} \quad Q_1 \Rightarrow Q}{\vdash \{P\}\ S\ \{Q\}}$$

Postcondition weakening

- Suppose we can prove {true} S {$x = y \wedge z = 2$}.
- Which of these can be proved?
  - {true} S {x=y}
  - {true} S {z = 2}
  - {true} S {z > 0}
  - {true} S {y > 2}

Proof rule for If statement

$$\frac{\vdash \{P \wedge C\}\ S_1\ \{Q\} \qquad \vdash \{P \wedge \neg C\}\ S_2\ \{Q\}}{\vdash \{P\}\ \textbf{if}\ C\ \textbf{then}\ \mathbf{S_1}\ \textbf{else}\ \mathbf{S_2}\ \{Q\}}$$

- Prove the correctness of this Hoare triple
  - $\{true\}$ if $x > 0$ then $y := x$ else $y := -x$ $\{y \geq 0\}$

Proof rule for loop

$$\frac{\vdash \{I \wedge C\}\ S\ \{I\}}{\vdash \{I\}\ \textbf{while}\ C\ \textbf{do}\ S\ \{I \wedge \neg C\}}$$

- A loop invariant I has following properties:
  - I holds before the loop
  - I holds after each iteration of the loop
- Suppose I is a loop invariant for this loop. What is guaranteed to hold after loop terminates?
- This rule simply says "If I is a loop invariant, then $I \wedge \neg C$ must hold after loop terminates"

Proof rule for loop

$$\frac{\vdash \{I \wedge C\}\ S\ \{I\}}{\vdash \{I\}\ \textbf{while}\ C\ \textbf{do}\ S\ \{I \wedge \neg C\}}$$

Consider the statement S= while x<n do x=x+1

Let's prove validity of {x ≤ n} S {x ≥ n}

What is the appropriate loop invariant?

First, let's prove x ≤ n is loop invariant. What do we need to show?

$$\cfrac{\cfrac{\cfrac{\vdash \{x+1 \leq n\} x = x+1 \{x \leq n\} \quad x \leq n \wedge x < n \Rightarrow x+1 < n+1}{\vdash \{x \leq n \wedge x < n\} x = x+1 \{x \leq n\}}}{\vdash \{x \leq n\} S \{x \leq n \wedge \neg(x < n)\}} \quad x \leq n \wedge \neg(x < n) \Rightarrow x \geq n}{\{x \leq n\} S \{x \geq n\}}$$

Invariant vs. Inductive Invariant

- Suppose I is a loop invariant for "while C do S"
- Does it always satisfy $\{I \wedge C\}\ S\ \{I\}$?
- Consider $I = j \geq 1$ and the code:

  i:=1; j:=1; while i<n do{j:=j+i; i:=i+1}

- Strengthened invariant $j \geq 1 \wedge i \geq 1$
- Key challenge in verification is finding inductive loop invariants

Example: a simple loop

- We want to infer that
  $\vdash \{x \leq 0\}$ `while` $x \leq 5$ `do` $x := x + 1$ $\{x = 6\}$
- Use the rule for while with invariant $\mathtt{I} \equiv x \leq 6$

$$\dfrac{\dfrac{\vdash x \leq 6 \wedge x \leq 5 \Rightarrow x + 1 \leq 6 \qquad \vdash \{x + 1 \leq 6\}\ x := x + 1\ \{x \leq 6\}}{\vdash \{x \leq 6 \wedge x \leq 5\}\ x := x + 1\ \{x \leq 6\}}}{\vdash \{x \leq 6\}\ \mathtt{while}\ x \leq 5\ \mathtt{do}\ x := x + 1\ \{x \leq 6 \wedge x > 5\}}$$

Manual proof construction is tedious

$\{x \leq n\}$ // precondition

while (x < n) **do**

$\{x \leq n \wedge x < n\}$ // loop invariant

x := x + 1

$\{x = n\}$ // postcondition

Hoare Logic proofs are highly manual:

- When to apply the rule of consequence?
- What loop invariants to use?

We can automate much of the proof process with **verification condition generation**!

But loop invariants still need to be provided...

Recap

- Described what does it mean that a program is correct "informally"
- Mechanisms helping us to formally define program correctness – Hoare logic
- Syntax based calculus for proving program correctness
- The goal is to automate this process

Guard commands

- Introduce loop-free guarded commands as an intermediate representation of the verification condition

$$
\begin{aligned}
c ::= \;& \mathrm{assume\ b} \\
\mid\;& \mathrm{assert\ b} \\
\mid\;& \mathrm{havoc\ x} \\
\mid\;& c_1 \,;\, c_2 \\
\mid\;& c_1 \,\square\, c_2
\end{aligned}
$$

Semantics of guard commands

- GC(`skip`) = assume true
- GC(x := e) = assume $tmp = x$; havoc x; assume ($x = e[tmp/x]$)
- GC(c_1 ; c_2) = GC(c_1) ; GC(c_2)
- GC(`if` b `then` c_1 `else` c_2) =

  (assume b; GC(c_1)) $\wedge$(assume $\neg b$; GC(c_2))

Semantics of guard commands

- GC({`I`} `while` b `do` c) =

  assert `I`;

  havoc x_1; ...; havoc x_n;

  assume `I`;

  (assume b; GC(c); assert `I`; assume false) $\square$

  assume $\neg b$

where x_1, ..., x_n are the variables modified in c

GC example

$\{n \geq 0\}$

$p := 0;$

$x := 0;$

$\{p = x * m \wedge x \leq n\}$

while $x < n$ do

$x := x + 1;$

$p := p + m$

$\{p = n * m\}$

Computing the guarded command

$\{ n \geq 0 \}$

assume $p_0 = p$; havoc p; assume $p = 0$;

assume $x_0 = x$; havoc x; assume $x = 0$;

assert $p = x * m \wedge x \leq n$;

havoc x; havoc p; assume $p = x * m \wedge x \leq n$;

(assume $x < n$;

assume $x_1 = x$; havoc x; assume $x = x_1 + 1$;

assume $p_1 = p$; havoc p; assume $p = p_1 + m$;

assert $p = x * m \wedge x \leq n$; assume false)

$\square$ assume $x \geq n$;

$\{ p = n * m \}$

VC generation

- Idea for VC generation: propagate the post-condition backwards through the program:
  - From {A} P {B}
  - Generate formula $A \Rightarrow F(P, B)$, where $F(P, B)$ is a formula describing the starting states for program to end in B
- This backwards propagation $F(P, B)$ can be formalized in terms of weakest preconditions.

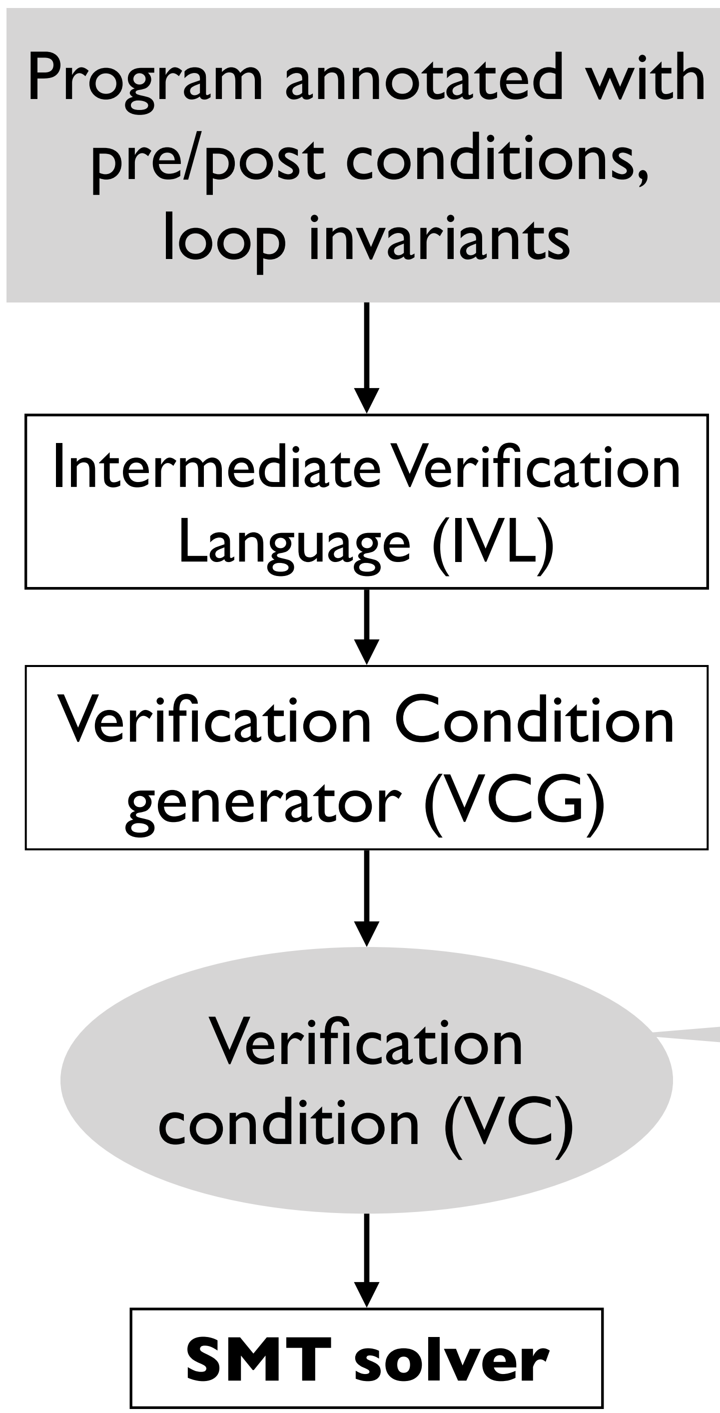
Why weakest

Verifying a Hoare triple

Theorem: {P} S {Q} is valid if the following formula is valid:

$$P \rightarrow wp(S_{IVL}, Q)$$

Automating Hoare Logic via VC generation



Forwards computation:

- Starting from the precondition, generate formulas to prove the postcondition.
- Based on computing strongest postconditions (sp).

Backwards computation:

- Starting from the postcondition, generate formulas to prove the precondition.
- Based on computing weakest preconditions (wp).

A formula φ generated automatically from the annotated program.
The program satisfies the specification if φ is valid.

VC generation with WP and SP

- **sp (S, P)**
  - The strongest predicate that holds for states produced by executing S on a state satisfying P.

Symbolic execution, covered in next lecture, computes SPs for finite programs (no unbounded loops).

- **wp (S, Q)**
  - The weakest predicate that guarantees Q will hold for states produced by executing S on a state satisfying that predicate.

Today, we'll see how to compute weakest preconditions (WP) for IMP. This lets us verify partial correctness properties.

- **{P} S {Q} is valid if**
  - $P \Rightarrow wp(S, Q)$ or $sp(S, P) \Rightarrow Q$

VC generation with WP

wp (S, Q)

- $wp(\text{skip}, Q) = Q$
- $wp(\textbf{assert}\ C, Q) = C \wedge Q$
- $wp(\textbf{assume}\ C, Q) = C \rightarrow Q$
- $wp(\textbf{havoc}\ x, Q) = Q\ [a/x]$ (a fresh in B)
- $wp(x := E, Q) = Q[E / x]$
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
- $wp(\textbf{if}\ C\ \textbf{then}\ S_1\ \textbf{else}\ S_2, Q) = (C \rightarrow wp(S_1, Q)) \wedge (\neg C \rightarrow wp(S_2, Q))$

Putting all together

- Given a Hoare triple $H \vdash \{A\}\ P\ \{B\}$
- Compute c_H = assume A; GC(P); assert B
- Compute $VC_H = WP(c_H, \text{true})$
- Check $\vdash VC_H$ using a theorem prover.

VC example

Computing the guarded command

$\{ n \geq 0 \}$

assume $p_0 = p$; havoc p; assume $p = 0$;
assume $x_0 = x$; havoc x; assume $x = 0$;
assert $p = x * m \wedge x \leq n$;
havoc x; havoc p; assume $p = x * m \wedge x \leq n$;
(assume $x < n$;
assume $x_1 = x$; havoc x; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc p; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false)
$\square$ assume $x \geq n$;

$\{ p = n * m \}$

Computing the weakest precondition

WP (assume $n \geq 0$;
assume $p_0 = p$; havoc p; assume $p = 0$;
assume $x_0 = x$; havoc x; assume $x = 0$;
assert $p = x * m \wedge x \leq n$;
havoc x; havoc p; assume $p = x * m \wedge x \leq n$;
(assume $x < n$;
assume $x_1 = x$; havoc x; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc p; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false)
$\square$ assume $x \geq n$;
assert $p = n * m$, true)

WP (assume $n \geq 0$;
assume $p_0 = p$; havoc p; assume $p = 0$;
assume $x_0 = x$; havoc x; assume $x = 0$;
assert $p = x * m \wedge x \leq n$;
havoc x; havoc p; assume $p = x * m \wedge x \leq n$,
WP (assume $x < n$;
assume $x_1 = x$; havoc x; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc p; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false, $p = n * m$)
$\wedge$ WP (assume $x \geq n$, $p = n * m$))

$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \Rightarrow pa_3 = xa_3 * m \wedge xa_3 \leq n \wedge$
$(pa_2 = xa_2 * m \wedge xa_2 \leq n \Rightarrow$
$((xa_2 < n \wedge x_1 = xa_2 \wedge xa_1 = x_1 + 1 \wedge$
$p_1 = pa_2 \wedge pa_1 = p_1 + m) \Rightarrow pa_1 = xa_1 * m \wedge xa_1 \leq n)$
$\wedge (xa_2 \geq n \Rightarrow pa_2 = n * m))$