Reasoning about Programs II

Emina Torlak
emina@cs.washington.edu
Overview

Last lecture

• Reasoning about (partial) correctness with Hoare Logic

Today

• Automating Hoare Logic with verification condition generation

Reminder

• Project proposals due today

Based on lectures by Isil Dillig, Daniel Jackson, and Viktor Kuncak
Recap: Imperative Programming Language (IMP)

**Expression** $E$
- $Z | V | E_1 + E_2 | E_1 * E_2$

**Conditional** $C$
- $\text{true} | \text{false} | E_1 = E_2 | E_1 \leq E_2$

**Statement** $S$
- $\text{skip}$ (Skip)
- $V := E$ (Assignment)
- $S_1; S_2$ (Composition)
- $\text{if C then } S_1 \text{ else } S_2$ (If)
- $\text{while C do } S$ (While)
Recap: Inference rules for Hoare logic

\[ \vdash \{P\} \text{skip} \{P\} \]

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

\[ \vdash \{P\} S \{Q\} \quad \Rightarrow \quad \{P\} S \{Q\} \]

\[ \vdash \{P\} S \{Q\} \]

\[ \vdash \{P\} S_1 \{R\} \quad \vdash \{R\} S_2 \{Q\} \]

\[ \vdash \{P\} S_1 ; S_2 \{Q\} \]

\[ \vdash \{P \land C\} S_1 \{Q\} \quad \vdash \{P \land \neg C\} S_2 \{Q\} \]

\[ \vdash \{P\} \text{if} C \text{ then } S_1 \text{ else } S_2 \{Q\} \]

\[ \vdash \{P \land C\} S \{P\} \quad \vdash \{P \land \neg C\} \]

\[ \vdash \{P\} \text{while} C \text{ do } S \{P \land \neg C\} \]

*loop invariant*
Challenge: manual proof construction is tedious!

\[
\{ x \leq n \} \\
\textbf{while} \ (x < n) \ \textbf{do} \\
\quad \{ x \leq n \land x < n \} \quad \text{// loop invariant} \\
\quad \{ x + 1 \leq n \} \quad \text{// consequence} \\
\quad x := x + 1 \\
\quad \{ x \leq n \} \quad \text{// assignment} \\
\quad \{ x \leq n \land x \geq n \} \quad \text{// while} \\
\quad \{ x \geq n \} \quad \text{// consequence}
\]

Hoare Logic proofs are highly manual:

- When to apply the rule of consequence?
- What loop invariants to use?
Challenge: manual proof construction is tedious!

\[
\{x \leq n\} \quad \text{// precondition}
while \ (x < n) \ do
\{x \leq n \quad \} \quad \text{// loop invariant}
\]

\[
x := x + 1
\]

\[
\{x \geq n\} \quad \text{// postcondition}
\]

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 …
Automating Hoare logic with VC generation

Program annotated with pre/post conditions, loop invariants

Verification Condition Generator (VCG)

verification condition (VC)

SMT solver
Automating Hoare logic with VC generation

Program annotated with pre/post conditions, loop invariants

Verification Condition Generator (VCG)

A formula $\varphi$ generated automatically from the annotated program.

The program satisfies the specification if $\varphi$ is valid.
Automating Hoare logic with 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 liberal preconditions (wp).
VC generation with WP and SP
VC generation with WP and SP

\[ \text{sp}(S, P) \]

- The strongest predicate that holds after S is executed from a state satisfying P.
VC generation with WP and SP

**sp(S, P)**

- The strongest predicate that holds after S is executed from a state satisfying P.

**wp(S, Q)**

- The weakest predicate that guarantees Q will hold after executing S from a state satisfying that predicate.
VC generation with WP and SP

sp(S, P)

- The strongest predicate that holds after S is executed from a state satisfying P.

wp(S, Q)

- The weakest predicate that guarantees Q will hold after executing S from a state satisfying that predicate.

{P} S {Q} is valid iff

- P \implies wp(S, Q) or
- sp(S, P) \implies Q
Computing $\text{wp}(S, Q)$
Computing $wp(S, Q)$

$wp(S, Q):$
Computing $wp(S, Q)$

$wp(S, Q)$:

- $wp(\text{skip}, Q) = Q$
Computing \( wp(S, Q) \)

\( wp(S, Q) : \)

- \( wp(\text{skip}, Q) = Q \)
- \( wp(x := E, Q) = Q[E / x] \)
Computing \( \text{wp}(S, Q) \)

\( \text{wp}(S, Q): \)

- \( \text{wp}(\text{skip}, Q) = Q \)
- \( \text{wp}(x := E, Q) = Q[x/E] \)
- \( \text{wp}(S_1; S_2, Q) = \text{wp}(S_1, \text{wp}(S_2, Q)) \)
Computing \( \text{wp}(S, Q) \)

\[
\text{wp}(S, Q):
\begin{align*}
\text{wp}(\text{skip}, Q) &= Q \\
\text{wp}(x := E, Q) &= Q[E / x] \\
\text{wp}(S_1; S_2, Q) &= \text{wp}(S_1, \text{wp}(S_2, Q)) \\
\text{wp(} \text{if } C \text{ then } S_1 \text{ else } S_2, Q) &= (C \rightarrow \text{wp}(S_1, Q)) \land (\neg C \rightarrow \text{wp}(S_2, Q))
\end{align*}
\]
Computing $wp(S, Q)$

$wp(S, Q)$:

- $wp(\text{skip}, Q) = Q$
- $wp(x := E, Q) = Q[E / x]$
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
- $wp(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = (C \rightarrow wp(S_1, Q)) \land (\neg C \rightarrow wp(S_2, Q))$
- $wp(\text{while } C \text{ do } S, Q) = ?$
Computing wp(S, Q)

wp(S, Q):

- \( \text{wp}(\text{skip}, Q) = Q \)
- \( \text{wp}(x := E, Q) = Q[E / x] \)
- \( \text{wp}(S_1; S_2, Q) = \text{wp}(S_1, \text{wp}(S_2, Q)) \)
- \( \text{wp}(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = (C \rightarrow \text{wp}(S_1, Q)) \land (\neg C \rightarrow \text{wp}(S_2, Q)) \)
- \( \text{wp}(\text{while } C \text{ do } S, Q) = \chi \)

A fixpoint: in general, cannot be expressed as a syntactic construction in terms of the postcondition.
Computing \( \text{wp}(S, Q) \)

\( \text{wp}(S, Q) \):  

- \( \text{wp}(\text{skip}, Q) = Q \)  
- \( \text{wp}(x := E, Q) = Q[E / x] \)  
- \( \text{wp}(S_1; S_2, Q) = \text{wp}(S_1, \text{wp}(S_2, Q)) \)  
- \( \text{wp}(\text{if} \ C \ \text{then} \ S_1 \ \text{else} \ S_2, Q) = (C \rightarrow \text{wp}(S_1, Q)) \land (\neg C \rightarrow \text{wp}(S_2, Q)) \)  
- \( \text{wp}(\text{while} \ C \ \text{do} \ S, Q) = \mathbf{X} \)

Approximate \( \text{wp}(S, Q) \) with \( \text{awp}(S, Q) \).
Computing $awp(S, Q)$

$awp(S, Q)$:

- $awp(\text{skip}, Q) = Q$
- $awp(x := E, Q) = Q[E/x]$
- $awp(S_1; S_2, Q) = awp(S_1, awp(S_2, Q))$
- $awp(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = (C \rightarrow awp(S_1, Q)) \land (\neg C \rightarrow awp(S_2, Q))$
- $awp(\text{while } C \text{ do } \{I\} S, Q) = I$
Computing \( \text{awp}(S, Q) \)

\( \text{awp}(S, Q) \):

- \( \text{awp}(\text{skip}, Q) = Q \)
- \( \text{awp}(x := E, Q) = Q[E / x] \)
- \( \text{awp}(S_1; S_2, Q) = \text{awp}(S_1, \text{awp}(S_2, Q)) \)
- \( \text{awp}(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = (C \rightarrow \text{awp}(S_1, Q)) \land (\neg C \rightarrow \text{awp}(S_2, Q)) \)
- \( \text{awp}(\text{while } C \text{ do } \{I\} S, Q) = I \)

Loop invariant provided by an oracle (e.g., programmer).
Computing $awp(S, Q)$

$awp(S, Q)$:

- $awp($skip, $Q) = Q$
- $awp(x := E, Q) = Q[E/x]$
- $awp(S_1; S_2, Q) = awp(S_1, awp(S_2, Q))$
- $awp($if $C$ then $S_1$ else $S_2$, $Q) = (C \rightarrow awp(S_1, Q)) \land (\neg C \rightarrow awp(S_2, Q))$
- $awp($while $C$ do $\{I\}$ $S$, $Q) = I$

For each statement $S$, also define $VC(S, Q)$ that encodes additional conditions that must be checked.
Computing $VC(S, Q)$
Computing $VC(S, Q)$

$VC(S, Q)$:
Computing $VC(S, Q)$

$VC(S, Q)$:

- $VC({\text{skip}}, Q) = \text{true}$
Computing $VC(S, Q)$

$VC(S, Q)$:

- $VC(\text{skip}, Q) = \text{true}$
- $VC(x := E, Q) = \text{true}$
Computing $\text{VC}(S, Q)$

$\text{VC}(S, Q)$:

- $\text{VC}(\text{skip}, Q) = \text{true}$
- $\text{VC}(x := E, Q) = \text{true}$
- $\text{VC}(S_1; S_2, Q) = \text{VC}(S_2, Q) \land \text{VC}(S_1, \text{awp}(S_2, Q))$
Computing \( VC(S, Q) \)

\( VC(S, Q): \)

- \( VC(\text{skip}, Q) = \text{true} \)
- \( VC(x := E, Q) = \text{true} \)
- \( VC(S_1; S_2, Q) = VC(S_2, Q) \land VC(S_1, \text{awp}(S_2, Q)) \)
- \( VC(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = VC(S_1, Q) \land VC(S_2, Q) \)
Computing $\text{VC}(S, Q)$

$\text{VC}(S, Q)$:

- $\text{VC}(\text{skip}, Q) = \text{true}$
- $\text{VC}(x := E, Q) = \text{true}$
- $\text{VC}(S_1; S_2, Q) = \text{VC}(S_2, Q) \land \text{VC}(S_1, \text{awp}(S_2, Q))$
- $\text{VC}(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = \text{VC}(S_1, Q) \land \text{VC}(S_2, Q)$
- $\text{VC}(\text{while } C \text{ do } \{I\} S, Q) = (I \land C \rightarrow \text{awp}(S,I)) \land \text{VC}(S,I) \land (I \land \neg C \rightarrow Q)$

$I$ is an invariant.
$I$ is strong enough.
Verifying a Hoare triple

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

$$VC(S, Q) \land (P \rightarrow awp(S, Q))$$
Verifying a Hoare triple

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

\[ VC(S, Q) \land (P \rightarrow \text{awp}(S, Q)) \]

The other direction doesn’t hold because loop invariants may not be strong enough or they may be incorrect.

Might get false alarms.
Summary

Today

• Automating Hoare Logic with VCG

Next Wednesday

• Guest lecture by Rustan Leino!
• Verification with Dafny, Boogie, and Z3.