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** $E$
- $Z | V | E_1 + E_2 | E_1 \times E_2$

**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)
- $\text{if } C \text{ then } S_1 \text{ else } S_2$ (If)
- $\text{while } C \text{ do } S$ (While)
Recap: Inference rules for Hoare logic

⊢ \{P\} \text{skip} \{P\}

⊢ \{Q[E/x]\} x := E \{Q\}

⊢ \{P_1\} S \{Q_1\} P \Rightarrow P_1 \quad Q_1 \Rightarrow Q

⊢ \{P\} S \{Q\}

⊢ \{P\} S_1 \{R\} ⊢ \{R\} S_2 \{Q\}

⊢ \{P\} S_1; S_2 \{Q\}

⊢ \{P \land C\} S_1 \{Q\} ⊢ \{P \land \neg C\} S_2 \{Q\}

⊢ \{P\} \text{if} C \text{then} S_1 \text{else} S_2 \{Q\}

⊢ \{P\} while C \text{do} S \{P \land \neg C\}

loop invariant
Challenge: manual proof construction is tedious!

Halare Logic proofs are highly manual:

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

```plaintext
{x ≤ n}  
while (x < n) do 
  {x ≤ n ∧ x < n} // loop invariant  
  {x+1 ≤ n} // consequence  
  x := x + 1 
  {x ≤ n} // assignment  
{x ≤ n ∧ x ≥ n} // while  
{x ≥ n} // consequence
```
Challenge: manual proof construction is tedious!

{\(x \leq n\)}  // precondition
while \((x < n)\) do
{\(x \leq n\) }  // loop invariant
\(x := x + 1\)

{\(x \geq n\)}  // 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 …
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)

Verification condition (VC)

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

The program satisfies the specification if $\varphi$ is valid.

SMT solver
Automating Hoare logic with VC generation

Forwards computation:
• Starting from the precondition, generate formulas to prove the postcondition.
  • Based on computing \textit{strongest postconditions (sp)}.

Backwards computation:
• Starting from the postcondition, generate formulas to prove the precondition.
  • Based on computing \textit{weakest liberal preconditions (wp)}.
VC generation with WP and SP
VC generation with WP and SP

sp(S, P)

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

\[ \text{sp}(S, P) \]
- The strongest predicate that holds after \( S \) is executed from a state satisfying \( P \).

\[ \text{wp}(S, Q) \]
- 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

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

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

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

$wp(S, Q)$:

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

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

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

\( wp(S, Q) : \)

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

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

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

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

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

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

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

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

Loop invariant provided by an oracle (e.g., programmer).
Computing \( \text{awp}(S, Q) \)

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

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

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

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

$VC(S, Q)$:

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

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

- $\text{VC}(\text{skip}, Q) = \text{true}$
- $\text{VC}(x := E, Q) = \text{true}$
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(skip, Q) = true
• VC(x := E, Q) = true
• VC(S₁; S₂, Q) = VC(S₂, Q) \land VC(S₁, awp(S₂, Q))
• VC(if C then S₁ else S₂, Q) = VC(S₁, Q) \land VC(S₂, 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

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

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

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

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

No lecture next Wednesday!

Next Friday

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