Computer-Aided Reasoning for Software

CSE507

Reasoning about Programs II

courses.cs.washington.edu/courses/cse507/18sp/

Emina Torlak
emina@cs.washington.edu

Overview

topics

Last lecture

- Reasoning about (partial) correctness with Hoare Logic

Today

- Automating Hoare Logic with verification condition generation

Reminder

- Project proposals due today

Recap: Imperative Programming Language (IMP)

Expression E

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

Conditional C

- $\text{true} \mid \text{false} \mid E_1 = E_2 \mid E_1 \leq E_2$

Statement S

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

Recap: Inference rules for Hoare logic

$$\frac{}{\vdash \{P\}\ \textbf{skip}\ \{P\}}$$

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

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

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

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

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

loop invariant

Challenge: manual proof construction is tedious!

$\{x \leq n\}$
while $(x < n)$ **do**
  $\{x \leq n \wedge x < n\}$ // loop invariant
  $\{x+1 \leq n\}$ // consequence
  $x := x + 1$
  $\{x \leq n\}$ // assignment
$\{x \leq n \wedge x \geq n\}$ // while
$\{x \geq n\}$ // 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 ≤ n}                      // precondition
while (x < n) do
  {x ≤ n          }          // loop invariant

  x := x + 1


{x ≥ n}                      // postcondition
```

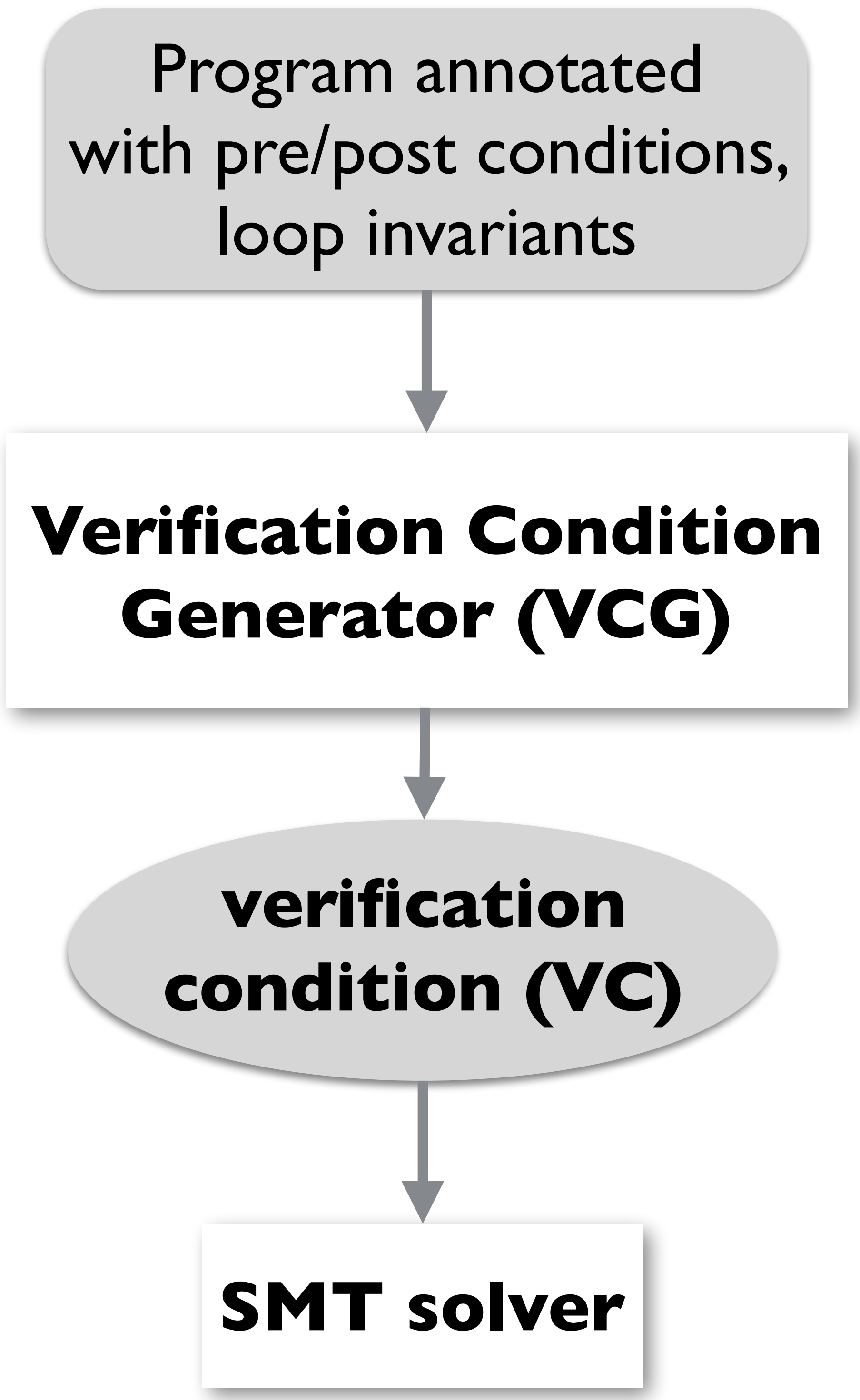
Hoare Logic proofs are highly manual:

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

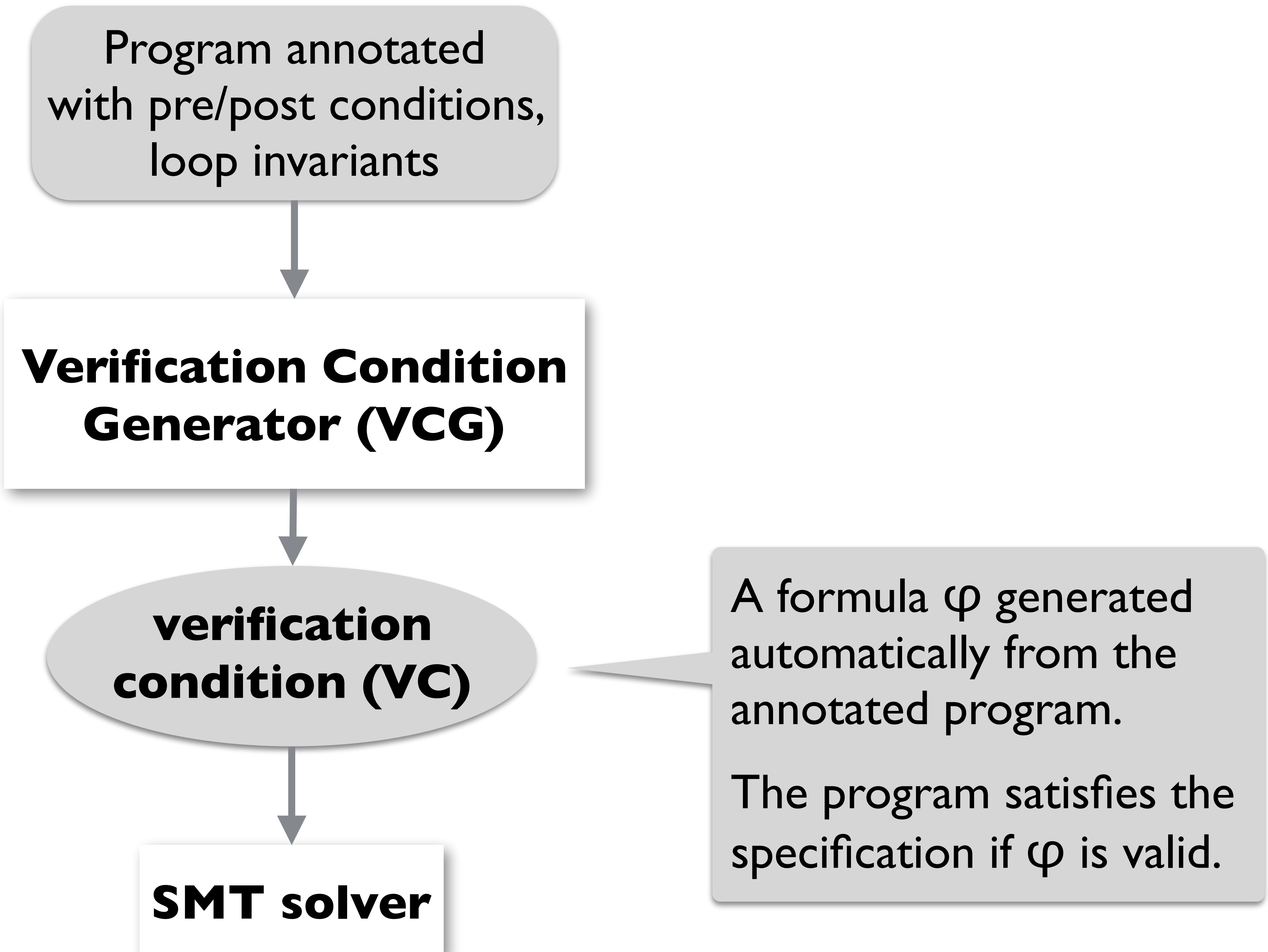
We can automate much of the proof process with verification condition generation!

- But loop invariants still need to be provided …

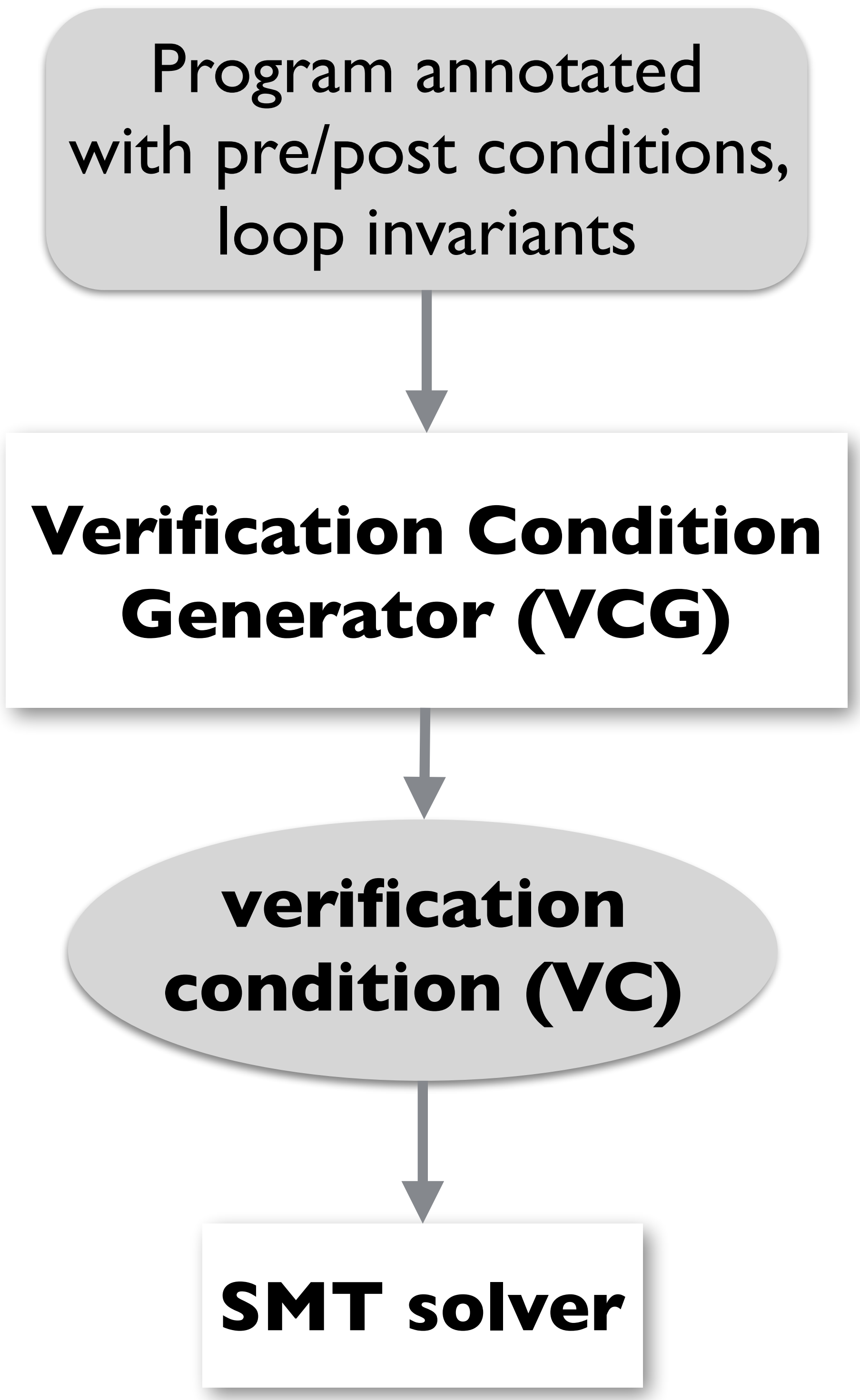
Automating Hoare logic with VC generation



Automating Hoare logic with VC generation



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

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

- $P \Rightarrow wp(S, Q)$ 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(**skip**, Q) = Q

Computing wp(S, Q)

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(x := E, Q) = Q[E / x]

Computing wp(S, Q)

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(x := E, Q) = Q[E / x]
- wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))

Computing wp(S, Q)

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(x := E, Q) = Q[E / x]
- wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))
- wp(**if** C **then** S_1 **else** S_2, Q) = (C → wp(S_1, Q)) ∧ (¬C → wp(S_2, Q))

Computing wp(S, Q)

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(x := E, Q) = Q[E / x]
- wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))
- wp(**if** C **then** S_1 **else** S_2, Q) = (C → wp(S_1, Q)) ∧ (¬C → wp(S_2, Q))
- wp(**while** C **do** S, Q) = ?

Computing wp(S, Q)

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(x := E, Q) = Q[E / x]
- wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))
- wp(**if** C **then** S_1 **else** S_2, Q) = (C → wp(S_1, Q)) ∧ (¬C → wp(S_2, Q))
- wp(**while** C **do** S, Q) = ✗

A fixpoint: in general, cannot be expressed as a syntactic construction in terms of the postcondition.

Computing wp(S, Q)

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(x := E, Q) = Q[E / x]
- wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))
- wp(**if** C **then** S_1 **else** S_2, Q) = (C → wp(S_1, Q)) ∧ (¬C → wp(S_2, Q))
- wp(**while** C **do** S, Q) = ✗

Approximate wp(S, Q) with awp(S, Q).

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 → awp(S_1, Q)) ∧ (¬C → awp(S_2, Q))
- awp(**while** C **do** {I} S, Q) = I

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 → awp(S_1, Q)) ∧ (¬C → awp(S_2, Q))
- awp(**while** C **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 → awp(S_1, Q)) ∧ (¬C → 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(**skip**, Q) = true

Computing VC(S, Q)

VC(S, Q):

- VC(**skip**, Q) = true
- VC(x := E, Q) = true

Computing VC(S, Q)

VC(S, Q):

- VC(**skip**, Q) = true
- VC(x := E, Q) = true
- $VC(S_1; S_2, Q) = VC(S_2, Q) \wedge VC(S_1, awp(S_2, Q))$

Computing VC(S, Q)

VC(S, Q):

- VC(**skip**, Q) = true
- VC(x := E, Q) = true
- VC(S_1; S_2, Q) = VC(S_2, Q) $\wedge$ VC(S_1, awp(S_2, Q))
- VC(**if** C **then** S_1 **else** S_2, Q) = VC(S_1, Q) $\wedge$ VC(S_2, Q)

Computing VC(S, Q)

VC(S, Q):

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

I is an invariant.

I is strong enough.

Verifying a Hoare triple

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

$$VC(S, Q) \wedge (P \rightarrow awp(S, Q))$$

Verifying a Hoare triple

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

$$VC(S, Q) \wedge (P \rightarrow awp(S, Q))$$

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

Next Wednesday

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