Computer-Aided Reasoning for Software

CSE507

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

Reminders

- HW2 is due tonight.

topics

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)
- **abort** (Abort)
- $V := E$ (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

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

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

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

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

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

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

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

loop invariant

Challenge: manual proof construction is tedious!

$\{x \leq n\}$
while $(x < n)$ **do**
  $\{x \leq n \land x < n\}$
  $\{x+1 \leq n\}$ // consequence
  $x := x + 1$
  $\{x \leq n\}$ // assignment
$\{x \leq n \land x \geq n\}$ // while
$\{x = n\}$ // 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
```

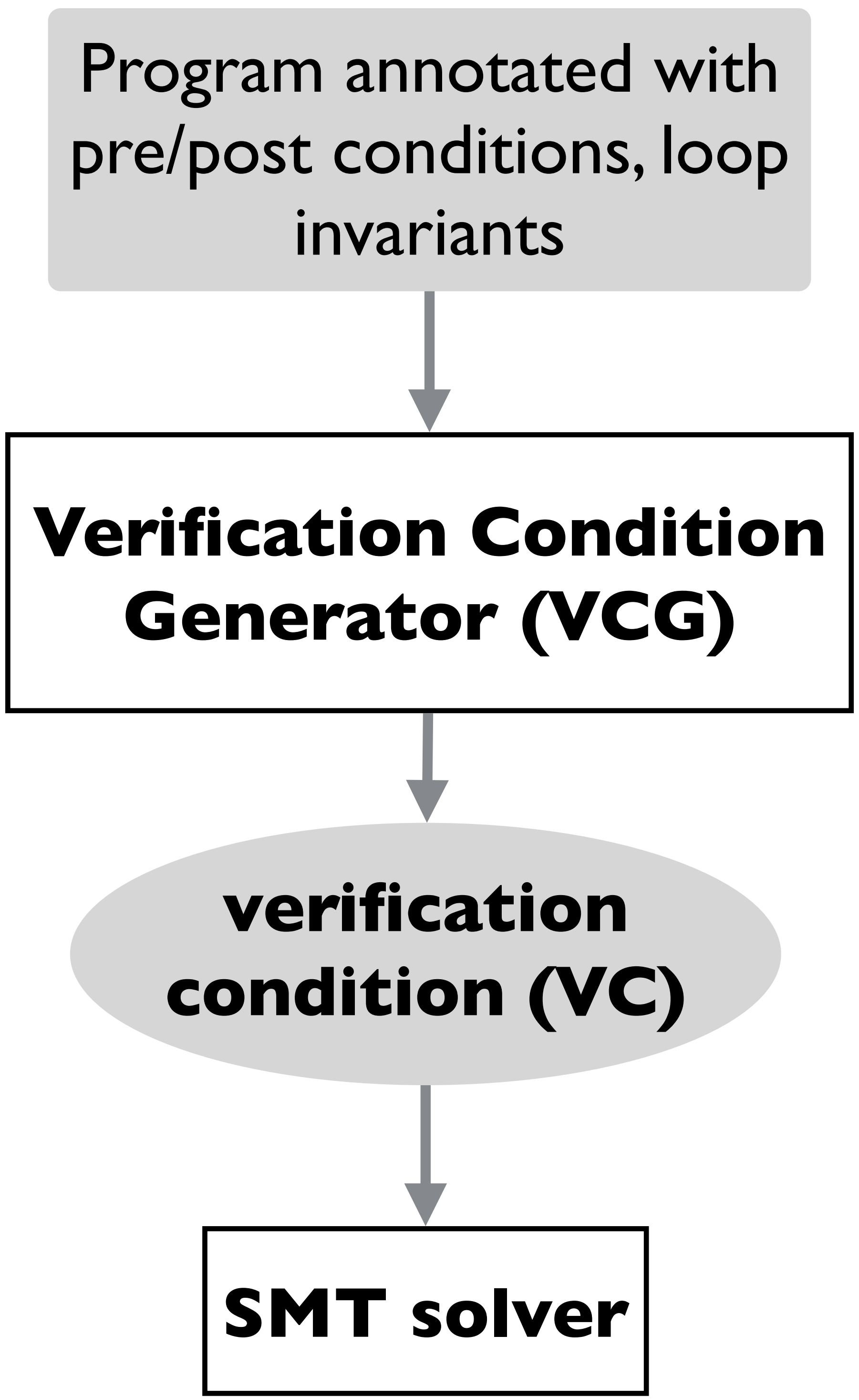
Hoare Logic proofs are highly manual:

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

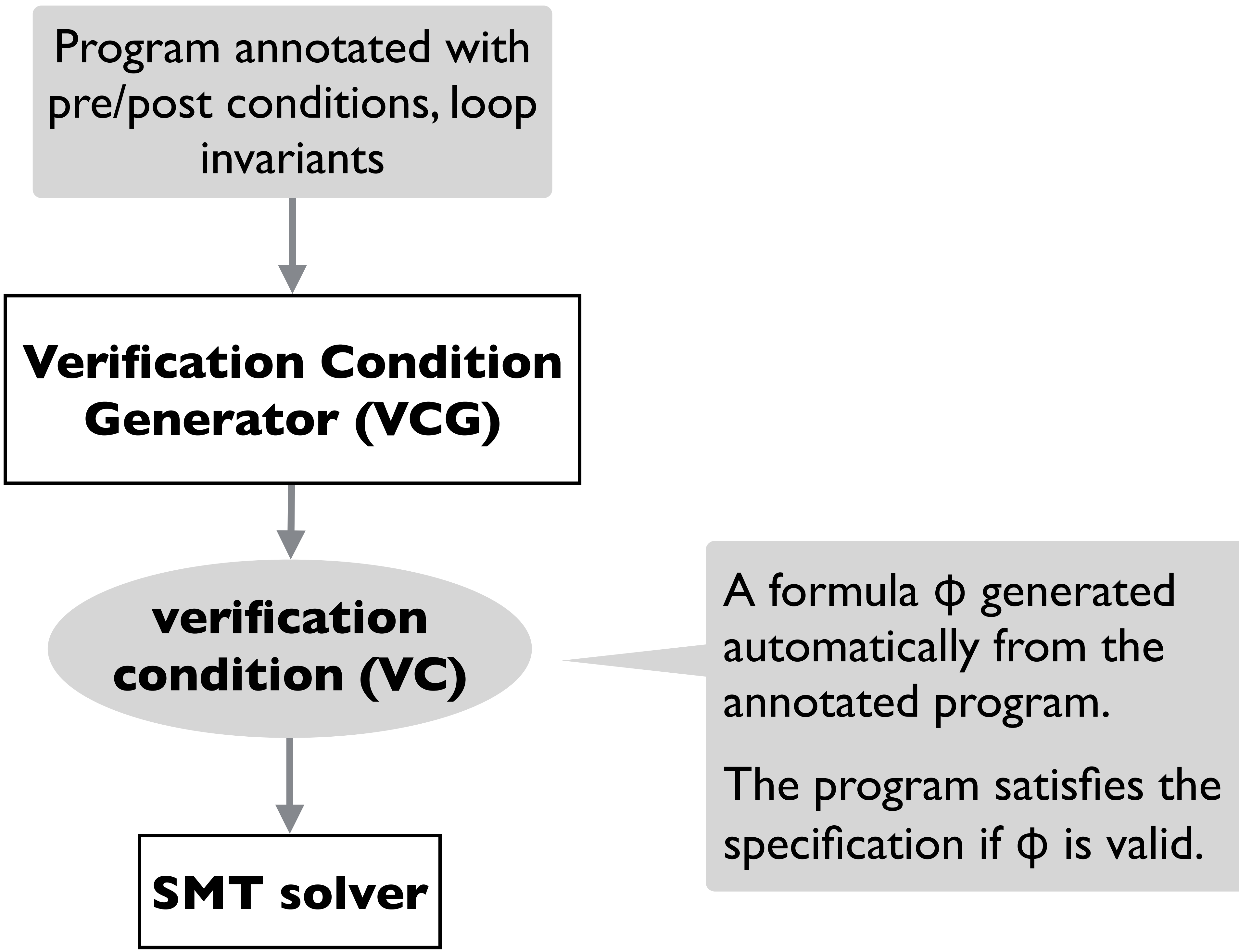
We can automate much of the proof process with verification condition generation!

- But loop invariants still need to be provided …

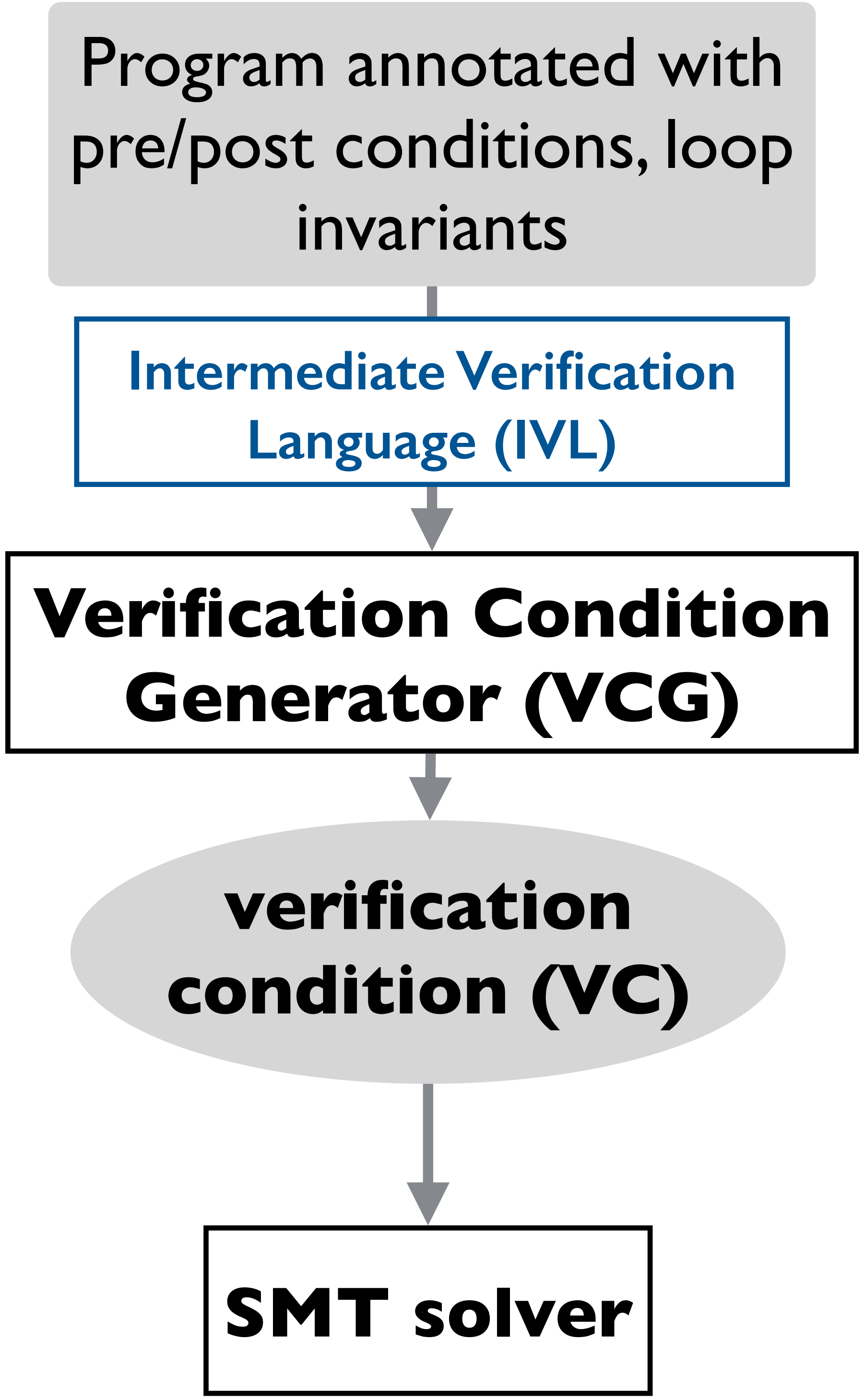
Automating Hoare logic with VC generation



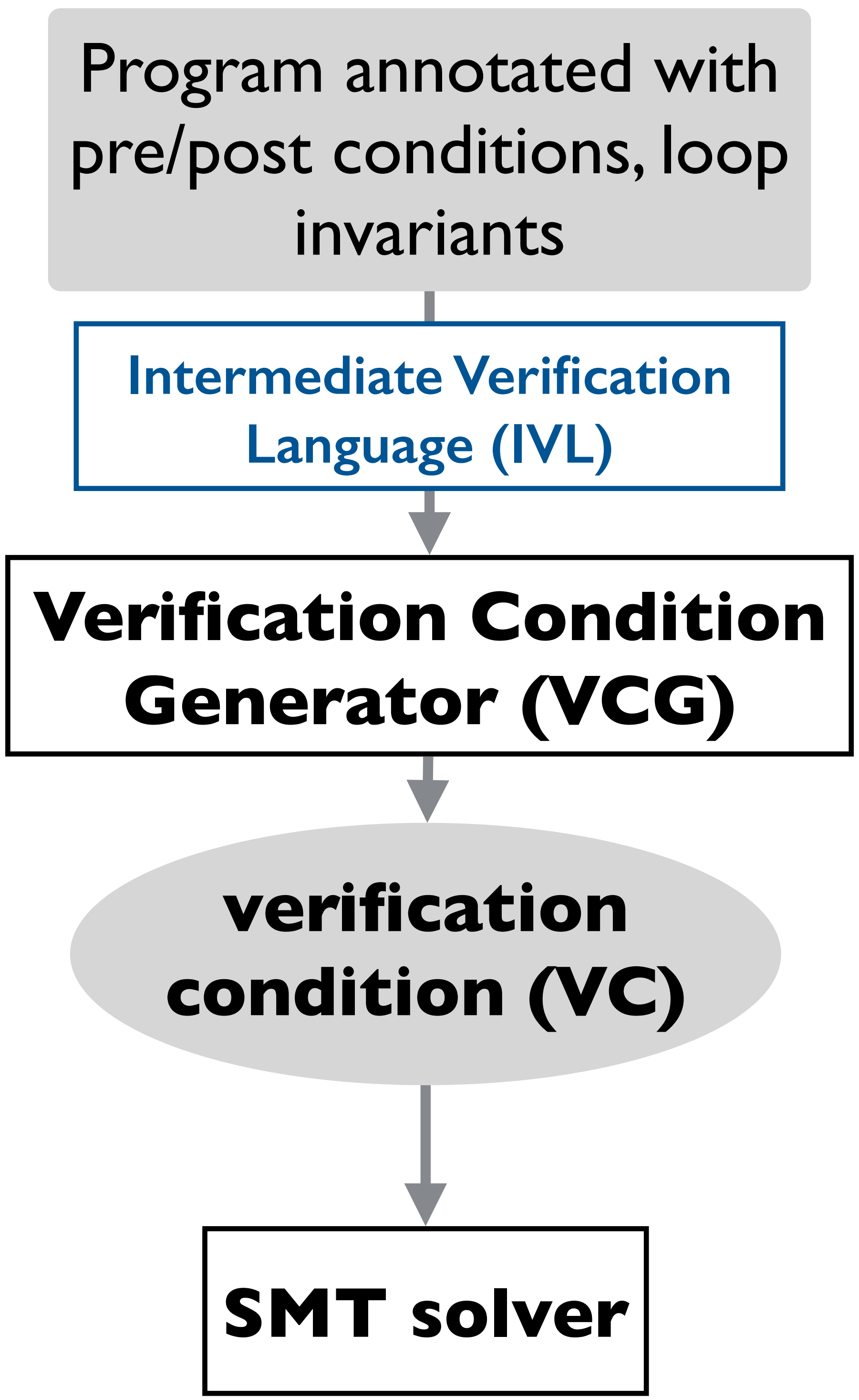
Automating Hoare logic with VC generation



Automating Hoare logic with VC generation



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 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).

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 liberal preconditions (WP) for IMP.

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)

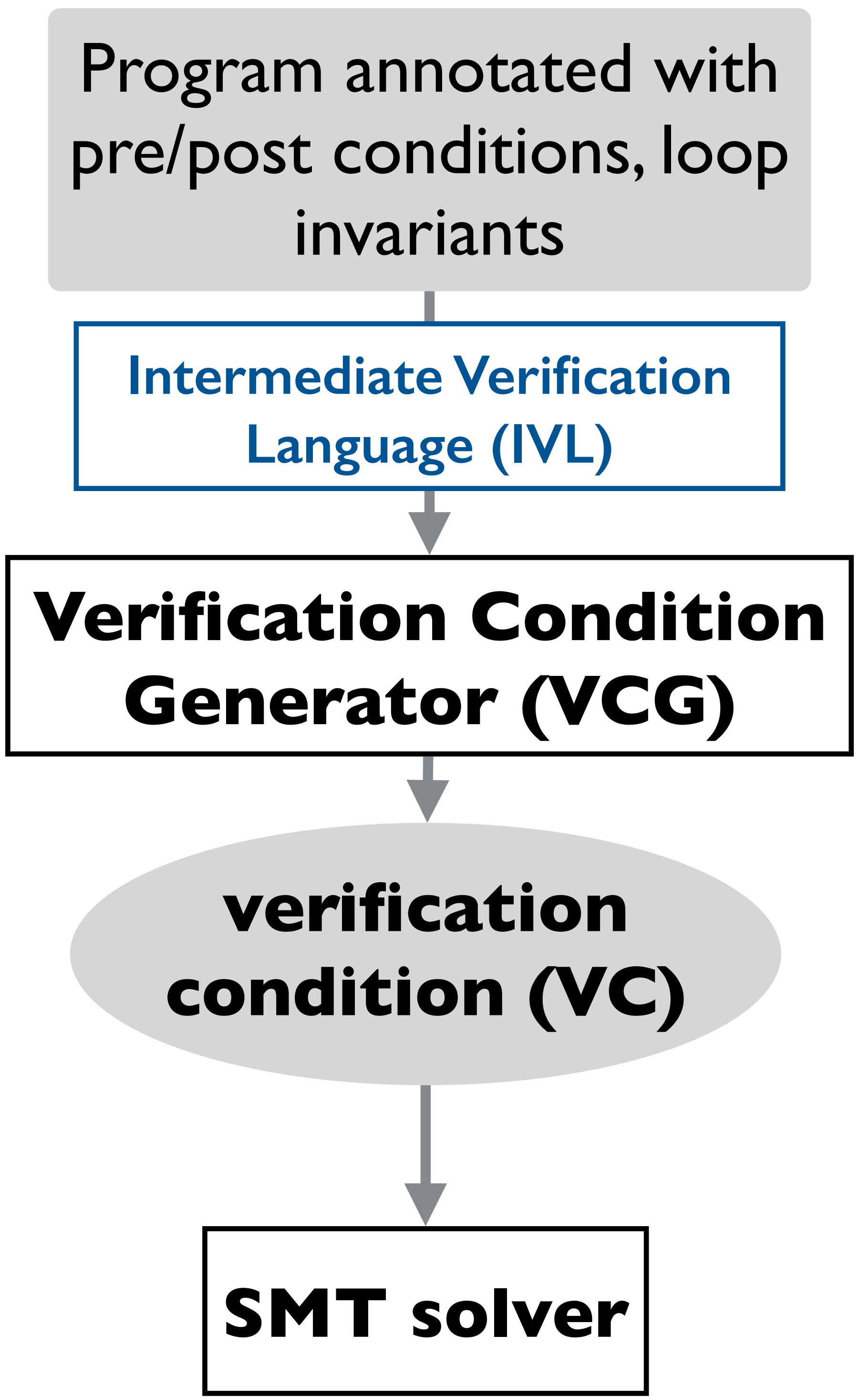
- The weakest predicate that guarantees Q will hold for states produced by executing S on a state satisfying that predicate.

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

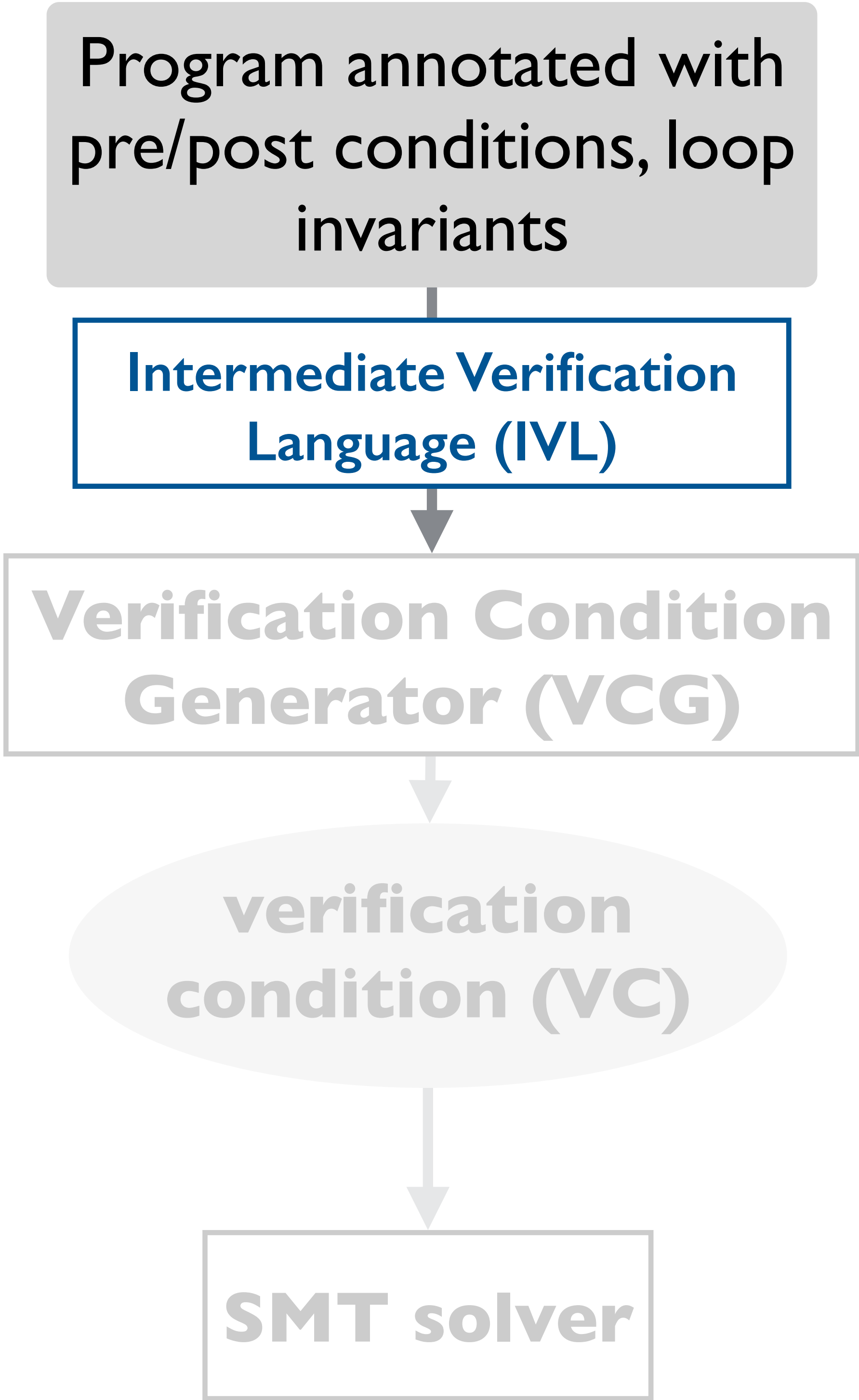
{P} S {Q} is valid if

- $P \Rightarrow wp(S, Q)$ or
- $sp(S, P) \Rightarrow Q$

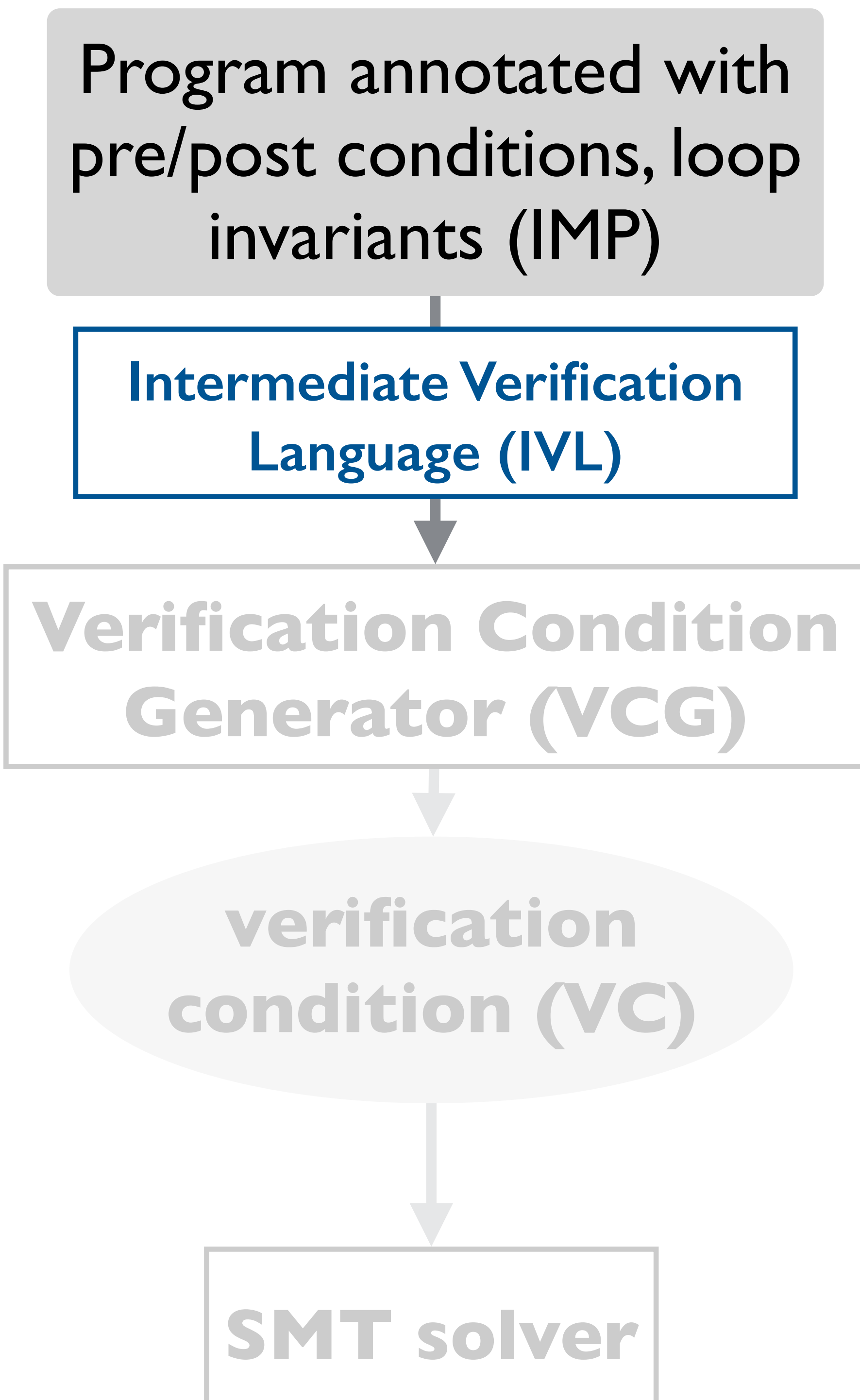
VC generation with WP



VC generation with WP



VC generation with WP: from IMP to IVL



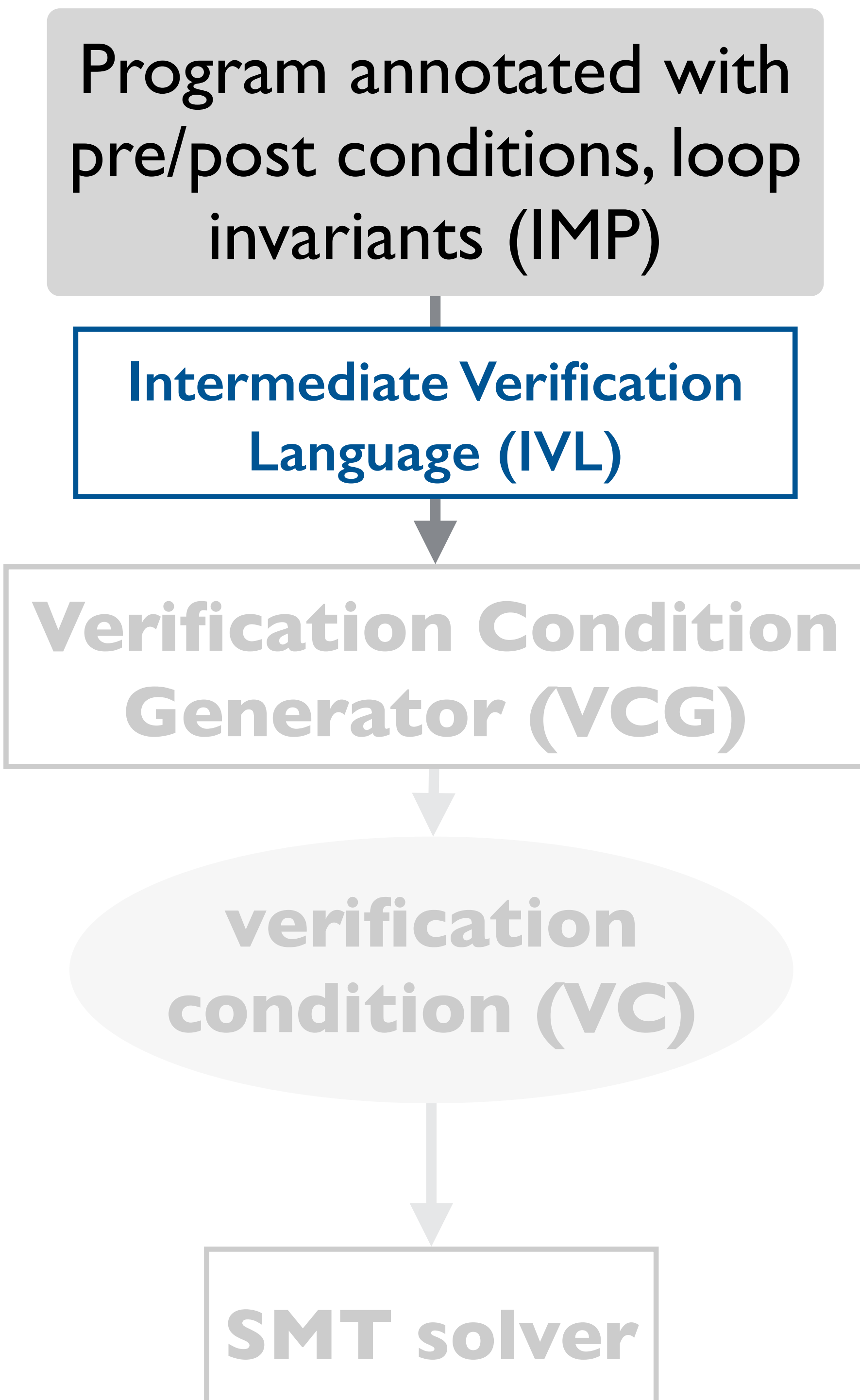
E ::= Z | V | E + E | E * E

C ::= true | false | E = E | E ≤ E

S ::= **skip** | **abort** | V := E | S; S |
if C **then** S **else** S |
while C {I} **do** S

{P} S {Q}

VC generation with WP: from IMP to IVL

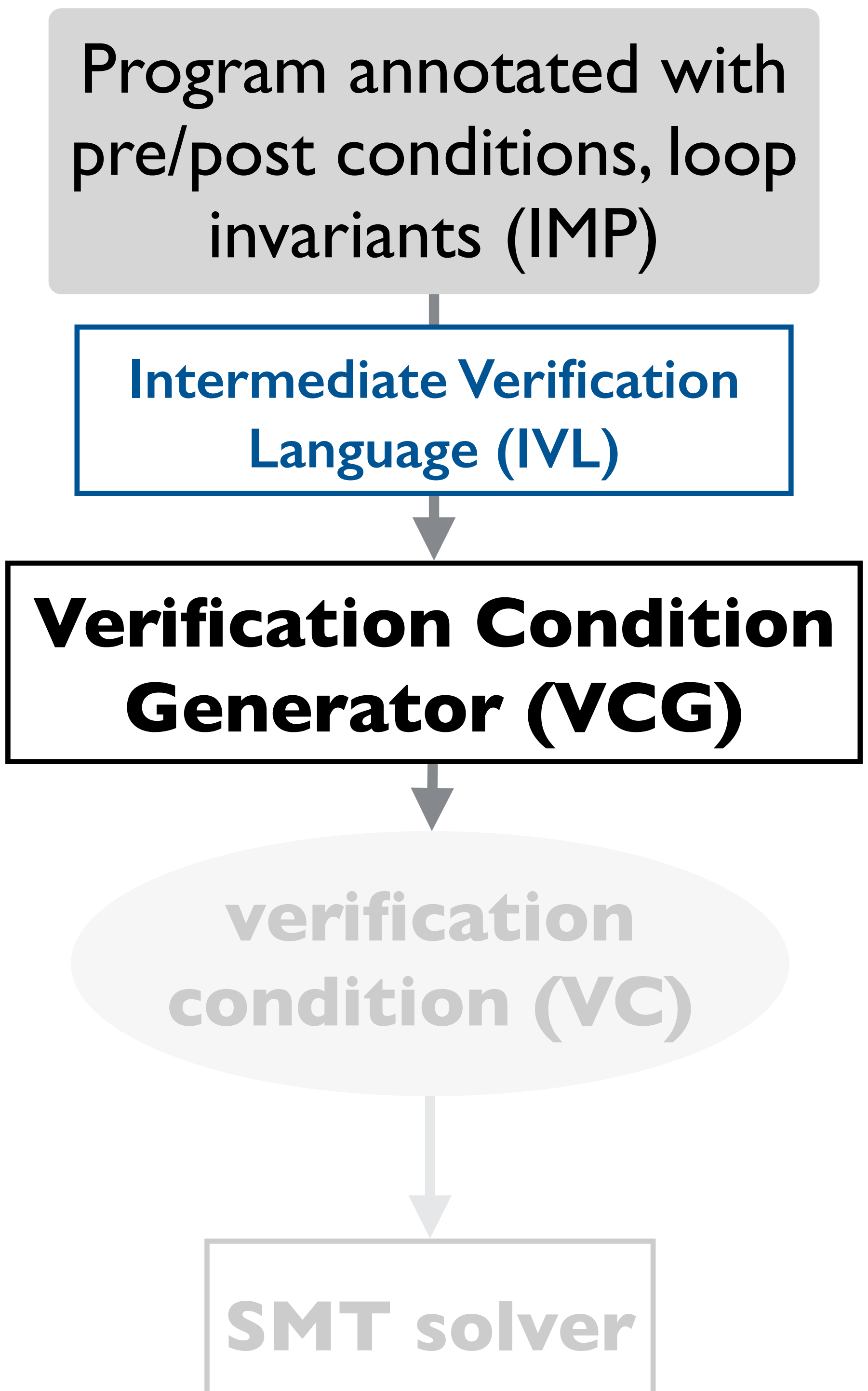


{P} S {Q}

E ::= Z | V | E + E | E * E
C ::= true | false | E = E | E ≤ E
S ::= **skip** | **abort** | V := E | S; S |
if C **then** S **else** S |
while C {I} **do** S

E ::= Z | V | E + E | E * E
C ::= true | false | E = E | E ≤ E
S ::= **skip** | **abort** | V := E | S; S |
if C **then** S **else** S |
assert C | **assume** C | **havoc** V

VC generation with WP: loop-free code



{P} S {Q}

E ::= Z | V | E + E | E * E

C ::= true | false | E = E | E ≤ E

S ::= **skip** | **abort** | V := E | S; S |
if C **then** S **else** S |
while C {I} **do** S

E ::= Z | V | E + E | E * E

C ::= true | false | E = E | E ≤ E

S ::= **skip** | **abort** | V := E | S; S |
if C **then** S **else** S |
assert C | **assume** C | **havoc** V

VC generation with WP: loop-free code

VC generation with WP: loop-free code

wp(S, Q):

VC generation with WP: loop-free code

wp(S, Q):

- wp(**skip**, Q) = Q

VC generation with WP: loop-free code

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(**abort**, Q) = true

VC generation with WP: loop-free code

wp(S, Q):

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

VC generation with WP: loop-free code

wp(S, Q):

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

VC generation with WP: loop-free code

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(**abort**, Q) = true
- 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) = $(C \rightarrow wp(S_1, Q)) \wedge (\neg C \rightarrow wp(S_2, Q))$

VC generation with WP: what about loops?

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(**abort**, Q) = true
- 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 {I} **do** S, Q) = ?

VC generation with WP: what about loops?

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(**abort**, Q) = true
- 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 $\rightarrow$ wp(S_1, Q)) $\wedge$ ($\neg$C $\rightarrow$ wp(S_2, Q))
- wp(**while** C {I} **do** S, Q) = ✗

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

VC generation with WP: what about loops?

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(**abort**, Q) = true
- 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 {I} **do** S, Q) = ✗

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

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

VC generation with WP: what about loops?

while C {I} **do** S

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

VC generation with WP: what about loops?

while C {I} **do** S

Cut the loop.

assert I;

havoc x; … *// for each loop target x*

assume I;

if C

then S; **assert** I; **assume** false;

else skip;

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

VC generation with WP: what about loops?

while C {I} **do** S

Cut the loop.

assert I;
havoc x; … *// for each loop target x*
assume I;
if C
then S; **assert** I; **assume** false;
else skip;

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

wp(S, Q):

VC generation with WP: what about loops?

while C {I} **do** S

Cut the loop.

assert I;

havoc x; … *// for each loop target x*

assume I;

if C

then S; **assert** I; **assume** false;

else skip;

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

wp(S, Q):

- wp(**assert** C, Q) = C $\wedge$ Q

VC generation with WP: what about loops?

while C {I} **do** S

Cut the loop.

assert I;

havoc x; … *// for each loop target x*

assume I;

if C

then S; **assert** I; **assume** false;

else skip;

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

wp(S, Q):

- wp(**assert** C, Q) = C $\wedge$ Q
- wp(**assume** C, Q) = C $\rightarrow$ Q

VC generation with WP: what about loops?

while C {I} **do** S

Cut the loop.

assert I;
havoc x; … *// for each loop target x*
assume I;
if C
then S; **assert** I; **assume** false;
else skip;

Use loop invariants to approximate loop behavior. Then check each invariant is correct and strong enough.

wp(S, Q):

- wp(**assert** C, Q) = $C \wedge Q$
- wp(**assume** C, Q) = $C \rightarrow Q$
- wp(**havoc** x, Q) = $\forall x . Q$

VC generation with WP: putting it all together

1. Translate IMP to IVL by cutting loops.
2. Compute WP for IVL.

wp(S, Q):

- wp(**skip**, Q) = Q
- wp(**abort**, Q) = true
- 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) =
  $(C \rightarrow wp(S_1, Q)) \wedge (\neg C \rightarrow wp(S_2, Q))$
- wp(**assert** C, Q) = $C \wedge Q$
- wp(**assume** C, Q) = $C \rightarrow Q$
- wp(**havoc** x, Q) = $\forall\, x\, .\, Q$

Verifying a Hoare triple

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

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

Verifying a Hoare triple

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

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

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 based on WPs

Next lecture

- Guest lecture by Rustan Leino!
- Verification with Dafny, Boogie, and Z3.
- On Zoom, see Canvas for the link.