

EC4219: Software Engineering

Lecture 12 — Program Verification (3) *Inductive Assertion Method (Cont'd)*

Sunbeom So
2024 Spring

Discussion: Supplementary Lectures

- Announcement: No class on 5/15 (Buddha's birthday), 5/22 (external seminar)
- We already missed a class on 5/1 (external seminar) and 5/6 (national holiday).
- We should have two supplementary lectures at least. Candidate schedules are the following.
 - ▶ 7:00 pm on 5/9 (tomorrow)
 - ▶ 10:30 am on 5/10 (this Friday)
 - ▶ 7:00 pm on 5/13 (next Monday)
 - ▶ 7:00 pm on 5/14 (next Tuesday)
 - ▶ 7:00 pm on 5/16 (next Thursday)
 - ▶ 10:30 am on 5/17 (next Friday)
 - ▶ 7:00 pm on 5/20 (Monday)
 - ▶ 7:00 pm on 5/21 (Tuesday)

This Lecture: Predicate Transformer in a Different Style

- Directly encoding the effects of `continue` and `break` is hard, because we should consider execution flows outside the given statement.

$$\begin{array}{lll} \mathbf{pre}(F, \text{assume}(c)) & \iff & c \rightarrow F \\ \mathbf{pre}(F, v := e) & \iff & F[e/v] \\ \mathbf{pre}(F, \text{continue}) & \iff & ? \\ \mathbf{pre}(F, \text{break}) & \iff & ? \end{array}$$

- To analyze programs with `continue` and `break`, VCs are generated after constructing a control-flow graph and generating a finite set of basic paths from it.
- If a program to verify does not contain them, we can verify it without control-flow graphs!

Tiny Imperative Language

To illustrate the approach, we will consider a very small imperative language.

$$\begin{aligned} lv &\rightarrow x \mid x[e] \\ e &\rightarrow n \mid lv \mid e_1 \oplus e_2 \mid \text{true} \mid \text{false} \\ &\quad \mid e_1 \prec e_2 \mid \neg e \mid e_1 \&\& e_2 \mid e_1 \parallel e_2 \\ c &\rightarrow lv := e \mid c_1; c_2 \mid \text{if } e \text{ } c_1 \text{ } c_2 \mid \text{while } [\phi] \text{ } e \text{ } c \end{aligned}$$

- \oplus and \prec are standard arithmetic ($+$, $-$, \dots) and comparison ($>$, \geq , \dots) operators, respectively.
- ϕ in the while-loop is a candidate invariant expressed in a FOL formula.

Arrays (1st attempt)

- Arrays are added in our language: $x[e_1] := e_2$
- How can we model the effect of array element assignments? Is the following definition correct?

$$\mathbf{pre}(F, x[e_1] := e_2) \iff F[e_2/x[e_1]]$$

Arrays (1st attempt)

- Arrays are added in our language: $x[e_1] := e_2$
- How can we model the effect of array element assignments? Is the following definition correct?

$$\mathbf{pre}(F, x[e_1] := e_2) \iff F[e_2/x[e_1]]$$

- Incorrect! Counterexample?

Arrays (1st attempt)

- Arrays are added in our language: $x[e_1] := e_2$
- How can we model the effect of array element assignments? Is the following definition correct?

$$\mathbf{pre}(F, x[e_1] := e_2) \iff F[e_2/x[e_1]]$$

- Incorrect! Counterexample?

$$\{i = 1\} \ x[i] := 3; x[1] := 2 \ \{x[i] = 3\}$$

Arrays (1st attempt)

- Arrays are added in our language: $x[e_1] := e_2$
- How can we model the effect of array element assignments? Is the following definition correct?

$$\mathbf{pre}(F, x[e_1] := e_2) \iff F[e_2/x[e_1]]$$

- Incorrect! Counterexample?

$$\begin{aligned} & \{i = 1\} \ x[i] := 3; x[1] := 2 \ \{x[i] = 3\} \\ \iff & \ i = 1 \rightarrow \mathbf{pre}(x[i] = 3, x[i] := 3; x[1] := 2) \\ \iff & \ i = 1 \rightarrow \mathbf{pre}(\mathbf{pre}(x[i] = 3, x[1] := 2), x[i] := 3) \\ \iff & \ i = 1 \rightarrow \mathbf{pre}(x[i] = 3, x[i] := 3) \\ \iff & \ i = 1 \rightarrow \mathbf{true} \end{aligned}$$

Although the Hoare triple is invalid, the VC is proven (unsound proof)!

Arrays (Corrected)

- The correct transformer:

$$\mathbf{pre}(F, x[e_1] := e_2) \iff F[x\langle e_1 \triangleleft e_2 \rangle / x]$$

- The idea is to replace x with a variable that is the same as x , with a potential exception at index e_1 .
- Using the corrected transformer, the VC is invalid as we wanted.

$$\begin{aligned} & \{i = 1\} \ x[i] := 3; x[1] := 2 \ \{x[i] = 3\} \\ \iff & \ i = 1 \rightarrow \mathbf{pre}(x[i] = 3, x[i] := 3; x[1] := 2) \\ \iff & \ i = 1 \rightarrow \mathbf{pre}(\mathbf{pre}(x[i] = 3, x[1] := 2), x[i] := 3) \\ \iff & \ i = 1 \rightarrow \mathbf{pre}(x\langle 1 \triangleleft 2 \rangle[i] = 3, x[i] := 3) \\ \iff & \ i = 1 \rightarrow (x\langle i \triangleleft 3 \rangle)\langle 1 \triangleleft 2 \rangle[i] = 3 \end{aligned}$$

$$\mathbf{pre}(F, \text{if } e \ c_1 \ c_2) \iff e \rightarrow \mathbf{pre}(F, c_1) \wedge \neg e \rightarrow \mathbf{pre}(F, c_2)$$

- If e holds, F must hold after executing the if-branch (c_1).
- Otherwise, F must hold after executing the else-branch (c_2).

Example: If-Statement

Consider the following statement S

$$S : x := y+1; \text{ if } x > 0 \text{ then } z := 1 \text{ else } z := -1$$

and its Hoare triple

$$\{y > -1\} S \{z = 1\}.$$

Determine the validity of the Hoare triple.

Example: If-Statement

Consider the following statement S

$S : x := y+1; \text{ if } x > 0 \text{ then } z := 1 \text{ else } z := -1$

and its Hoare triple

$$\{y > -1\} S \{z = 1\}.$$

Determine the validity of the Hoare triple.

$$\begin{aligned} & \{y > -1\} S \{z = 1\} \\ \iff & y > -1 \rightarrow \text{pre}(z = 1, S) \\ \iff & y > -1 \rightarrow \text{pre}(x > 0 \rightarrow \text{pre}(z = 1, z := 1) \\ & \quad \wedge \neg(x > 0) \rightarrow \text{pre}(z = 1, z := -1), x := y + 1) \\ \iff & y > -1 \rightarrow \text{pre}(x > 0 \rightarrow 1 = 1 \wedge \neg(x > 0) \rightarrow -1 = 1, x := y + 1) \\ \iff & y > -1 \rightarrow (y + 1 > 0 \rightarrow 1 = 1 \wedge \neg(y + 1 > 0) \rightarrow -1 = 1) \end{aligned}$$

The VC is valid.

Loops

- Unfortunately, we cannot exactly compute the weakest preconditions for loops.
- We just rely on the annotated invariant ϕ .

$$\mathbf{pre}(F, \text{while } [\phi] \text{ e } c) \iff \phi$$

Idea: since the invariant ϕ holds before entering the loop (by definition), we just compute the precondition as ϕ .

- We should impose extra conditions (why?).

Loops

- Unfortunately, we cannot exactly compute the weakest preconditions for loops.
- We just rely on the annotated invariant ϕ .

$$\text{pre}(F, \text{while } [\phi] \ e \ c) \iff \phi$$

Idea: since the invariant ϕ holds before entering the loop (by definition), we just compute the precondition as ϕ .

- We should impose extra conditions (why?).
 - ① We haven't checked that ϕ is an actual (inductive) loop invariant.
 - ② We haven't checked that $\phi \wedge \neg e$ is sufficient to establish F (i.e., we haven't check the validity of $\phi \wedge \neg e \rightarrow F$).

cf) Invariant vs. Inductive Invariant

- Suppose ϕ is an annotated loop invariant for $S : \text{while } [\phi] \text{ e } c$.
- We should check $\{\phi \wedge e\} c \{\phi\}$ is valid or not.
- Even if ϕ is true during the loop, the Hoare triple may not be valid.
- For example, consider $\phi : j \geq 1$ and the code

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

ϕ is not an inductive, provable invariant (although ϕ is true!), since the Hoare triple is invalid:

$$\{j \geq 1 \wedge i < n\} j := j + i; i := i + 1 \{j \geq 1\}$$

However, the strengthened invariant $j \geq 1 \wedge i \geq 1$ is an inductive invariant.

- In verification, we are typically interested in finding inductive invariants.

Extra Conditions for Inductive Loop Invariants

$$VC(F, \text{while } [\phi] \ e \ c) \\ \iff (\phi \wedge e \rightarrow \mathbf{pre}(\phi, c)) \wedge (\phi \wedge \neg e \rightarrow F) \wedge VC(\phi, c)$$

The other cases should be defined as well, because we may have nested loops and corresponding invariants.

$$\begin{aligned} VC(F, lv := e) &\iff true \\ VC(F, c_1; c_2) &\iff VC(F, c_2) \wedge VC(F, \mathbf{pre}(F, c_1)) \\ VC(F, \text{if } e \ c_1 \ c_2) &\iff VC(F, c_1) \wedge VC(F, c_2) \end{aligned}$$

Final: Verification of Hoare Triple

To show the validity of $\{P\} S \{Q\}$, we should check the validity of the following formula.

$$VC(Q, S) \wedge (P \rightarrow \mathbf{pre}(Q, S))$$

That is, if the formula is valid, we prove the partial correctness of the implementation.

Summary

- A different style of the predicate transformer without control-flow graphs.
 - ▶ In the previous version, the inductiveness is checked in subsequent basic paths.
 - ▶ In the new version, we explicitly impose extra conditions for it using VC .
- Encoding arrays
- Invariant vs. Inductive Invariant

Announcement: Homework 1 is out (due: 5/22)