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.

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

- 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.

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

- $\oplus$ and $\prec$ are standard arithmetic $(+, -, \cdots)$ and comparison $(>, \geq, \cdots)$ 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?

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

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

Arrays (Corrected)

- The correct transformer:

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

- 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.

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

If-Statement

$$\mathbf{pre}(F, \mathtt{if}\ e\ c_1\ c_2) \iff e \to \mathbf{pre}(F, c_1) \land \neg e \to \mathbf{pre}(F, c_2)$$

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

Example: If-Statement

Consider the following statement S

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

and its Hoare triple

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

Determine the validity of the Hoare triple.

Example: If-Statement

Consider the following statement S

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

and its Hoare triple

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

Determine the validity of the Hoare triple.

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

The VC is valid.

Loops

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

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

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?).

Loops

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

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

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 $S :$ while $[\phi]\ e\ c$.
- We should check $\{\phi \wedge e\}\ S\ \{\phi\}$ 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 Hoard triple is invalid:

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

However, the strengthed 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

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

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

$$VC(F, lv := e) \iff true$$
$$VC(F, c_1; c_2) \iff VC(F, c_2) \wedge VC(F, \mathbf{pre}(F, c_1))$$
$$VC(F, \texttt{if } e \ c_1 \ c_2) \iff VC(F, c_1) \wedge VC(F, c_2)$$

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)