The Austalian National University
Research School of Computer Science
Jinbo Huang

Semester 2, 2013
Hoare Logic II Tutorial

Formal Methods in Software Engineering: Hoare Logic II

Exercise 1 **Weakest Precondition for Conditional**

Recall the definition of wp.[1] Prove the following by a case analysis on b:

$$wp(\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, Q) \Leftrightarrow (b \wedge wp(S_1, Q)) \vee (\neg b \wedge wp(S_2, Q)).$$

Solution

Consider an arbitrary state σ. There are two cases: (i) b is true in σ; (ii) b is false in σ. In the first case, executing the `if`-statement from σ is equivalent to executing S_1 from σ; hence $wp(\texttt{if}\ldots, Q)(\sigma) = wp(S_1, Q)(\sigma)$. By the rules of propositional logic, the right-hand side of the equivalence (to be proved) simplifies to $wp(S_1, Q)$, which has the value $wp(S_1, Q)(\sigma)$ in σ. In other words, the two sides have the same value in any state, and the theorem is proved. The second case is entirely symmetric to the first.

Exercise 2 **Weakest Precondition for Sequence**

In our proof of the completeness of Hoare logic (with respect to partial correctness), the fact

$$wp(S_1; S_2, Q) \Rightarrow wp(S_1, wp(S_2, Q))$$

was used without proof. An intuitive reading of this fact in English is as follows: If some precondition guarantees the postcondition Q once $S_1; S_2$ terminates, then that same precondition guarantees that the execution of S_1 alone, if it terminates, will result in a state from which the execution of S_2, if it terminates, will guarantee Q. Using this interpretation as a hint, give a formal proof of this result based on the definition of wp by showing that $wp(S_1; S_2, Q)(\sigma) \Rightarrow wp(S_1, wp(S_2, Q))(\sigma)$ for all states σ.

Solution

As suggested, we will show that $wp(S_1; S_2, Q)(\sigma) \Rightarrow wp(S_1, wp(S_2, Q))(\sigma)$ for all states σ.

If the execution of S_1 from state σ does not terminate, then $wp(S_1, X)(\sigma) = true$ for all conditions X and the implication holds. Assume that it terminates and results in state σ'.

If the execution of S_2 from state σ' does not terminate, then $wp(S_2, X)(\sigma') = true$ for all conditions X and, in particular, $wp(S_2, Q)(\sigma') = true$. This means that $wp(S_1, wp(S_2, Q))(\sigma) = true$ (because the execution of S_1 from σ has led to a state σ' that satisfies the given postcondition $wp(S_2, Q)$) and the implication holds. Assume that it terminates and results in state σ''.

Now assume the left-hand side of the implication is true, which implies $Q(\sigma'') = true$. This in turn implies that $wp(S_2, Q)(\sigma') = true$ (because the execution of S_2 from σ' has led to a state σ'' that satisfies the given postcondition Q). Hence $wp(S_1, wp(S_2, Q))(\sigma)$, the right-hand side of the implication, is true (because the execution of S_1 from σ has led to a state σ' that satisfies the given postcondition $wp(S_2, Q)$).

Exercise 3 **Proof Construction**

Give a proof of the following Hoare triple:

$[m \geq 0]$

```
fact := 1;
i := m;
while (i>0) do
    fact := fact * i;
    i := i-1
```

$[fact = m!]$

Hint: For the while-rule, use the invariant $P \equiv (fact * i! = m! \wedge i \geq 0)$.

[1] $wp(S, Q)(\sigma) = true$ iff for all states σ', (executing S from σ results in σ') $\Rightarrow (Q(\sigma') = true)$. Also recall that a condition can be regarded as a function that maps states to $\{true, false\}$.

Solution

In addition to P, we need an expression E that stays nonnegative and decreases: i is a natural choice.

To produce a proof, we work backwards from the given postcondition and the end of the program. The last (top-level) statement of the program is a `while`; hence we will need to apply the while-rule to it, which requires two premises (see the while-rule in the appendix). The b in the while-rule refers to the control condition, which in this case is $(i > 0)$. The E is the variant, which we have decided would be i. The S refers to the body of the loop, which consists of two assignments. Plugging these into the while-rule, the two premises we need to establish are step 1 and step 7 in the proof below. Step 1 is a trivial fact, which we may take for granted.

Step 7 becomes our next goal. Again we work backwards and look at the given postcondition and the last of the two assignment statements. Applying the assignment rule, we get step 2, which simplifies to step 3. Since we have a sequence (of two assignment statements), we take the precondition from step 3, use it as the postcondition for the first statement of the sequence, and, applying the assignment rule again, arrive at step 4, which simplifies to step 5. We now apply the sequencing rule to get step 6, which gives us step 7 by precondition strengthening.

Having established both premises, we apply the while-rule to get step 8, which simplifies to step 9

We now work further backwards to incorporate the rest of the program, i.e., the two assignment statements at the top of the program. Remember that when we have a sequence, the precondition for a statement is used as the postcondition of its predecessor in the sequence. This gives us steps 10 and 11. After some simplification (to get step 12) and applying the sequencing rule, we get step 13, the desired triple for the whole program.

The complete proof is given below (note that you do not need to provide narratives like the one given above to accompany your proofs in the assignment and exams):

1. $P \wedge i > 0 \Rightarrow i \geq 0$ (Trivial Fact)
2. $[fact * (i-1)! = m! \wedge i - 1 \geq 0 \wedge i - 1 < n]$ `i := i-1` $[P \wedge i < n]$ (Assignment)
3. $[fact * (i-1)! = m! \wedge i > 0 \wedge i - 1 < n]$ `i := i-1` $[P \wedge i < n]$ (2, Equivalence)
4. $[fact * i * (i-1)! = m! \wedge i > 0 \wedge i - 1 < n]$ `fact := fact * i`
   $[fact * (i-1)! = m! \wedge i > 0 \wedge i - 1 < n]$ (Assignment)
5. $[P \wedge i > 0 \wedge i - 1 < n]$ `fact := fact * i`
   $[fact * (i-1)! = m! \wedge i > 0 \wedge i - 1 < n]$ (4, Equivalence)
6. $[P \wedge i > 0 \wedge i - 1 < n]$ `fact := fact * i; i := i-1` $[P \wedge i < n]$ (5, 3, Sequencing)
7. $[P \wedge i > 0 \wedge i = n]$ `fact := fact * i; i := i-1` $[P \wedge i < n]$ (6, Precondition Strengthening)
8. $[P]$ `while (i>0) ...` $[P \wedge i \leq 0]$ (1, 7, While)
9. $[P]$ `while (i>0) ...` $[fact = m!]$ (8, Postcondition Weakening)
10. $[fact * m! = m! \wedge m \geq 0]$ `i := m` $[P]$ (Assignment)
11. $[1 * m! = m! \wedge m \geq 0]$ `fact := 1` $[fact * m! = m! \wedge m \geq 0]$ (Assignment)
12. $[m \geq 0]$ `fact := 1` $[fact * m! = m! \wedge m \geq 0]$ (11, Equivalence)
13. $[m \geq 0]$ *Program* $[fact = m!]$ (12, 10, 9, Sequencing)

Exercise 4 **Auxiliary Variables**

In our proof of the soundness of the while-rule for total correctness, we argued that

$$[P \wedge b \wedge (E = n)]\ S\ [P] \Rightarrow [P \wedge b]\ S\ [P]$$

because n is an auxiliary variable not appearing anywhere else. Give a proof of this result noting the assumption we made that an expression always returns a number. Then explain why the result does not hold without that assumption, e.g., if division is included in the language of assertions and division by zero returns a special error message that is not a number.

Solution

Assume $[P \wedge b \wedge (E = n)]\ S\ [P]$. We shall show that the triple $[P \wedge b]\ S\ [P]$ is valid. Consider an arbitrary state σ that satisfies $P \wedge b$. Since an expression always returns a number, let the value of E in σ be e. Now let σ' be a state where the value of variable n is e and the values of all other variables are as they are in σ. In other words, states σ and σ' possibly disagree on the value of variable n but are otherwise identical.

Clearly, state σ' satisfies $P \wedge b \wedge (E = n)$; hence by the assumed Hoare triple the execution of S from σ' will terminate and guarantee postcondition P. On the other hand, the execution of S from σ' and that from σ must be identical because the two states either are identical or only differ on the value of variable n, which does not appear in S. Hence the execution of S from σ must also terminate and guarantee postcondition P. That is, $[P \wedge b]\ S\ [P]$.

Now if we don't have the assumption that an expression always returns a number, the result will cease to hold. For example, let $E \equiv 1/0$ (one divided by zero), and assume that it returns an error message that is not a number. The condition $P \wedge b \wedge (E = n)$ is equivalent to $false$ because n is a number and can never equal something that isn't a number, in any state. The triple $[false]\ S\ [P]$ is trivially valid for any S and P and clearly, it will not imply $[P \wedge b]\ S\ [P]$ (just let $P \wedge b$ be true and S be a loop that trivially does not terminate).

Exercise 5 **Variant for While-Rule**

Consider the following Hoare triple:

$[x = 3 \vee x = 1 \vee x = 2]$

```
while (x != 2) do
   if (x=3) then x := 1
   else if (x=1) then x := 2 else x := x
```

$[true]$

Is this triple valid? Is it provable using our rules (given in the appendix)? In particular, can you find an expression E that stays nonnegative and keeps decreasing for use in the while-rule? Assume that you may only use the arithmetic operations $+, -, *$ in the expression.

Solution

The triple is valid by simple inspection of the code. The following is a suitable E to use: $(x-3)(x-1)x + x$. You may verify that the value of this expression strictly decreases on each iteration of the loop as long as the precondition holds.

Appendix: Rules for Total Correctness

$$[Q(e)]\ \texttt{x := e}\ [Q(x)] \qquad \text{(Assignment)}$$

$$\frac{P_s \Rightarrow P_w \qquad [P_w]\ S\ [Q]}{[P_s]\ S\ [Q]} \qquad \text{(Precondition Strengthening)}$$

$$\frac{[P]\ S\ [Q_s] \qquad Q_s \Rightarrow Q_w}{[P]\ S\ [Q_w]} \qquad \text{(Postcondition Weakening)}$$

$$\frac{[P]\ S_1\ [Q] \qquad [Q]\ S_2\ [R]}{[P]\ S_1; S_2\ [R]} \qquad \text{(Sequencing)}$$

$$\frac{[P \wedge b]\ S_1\ [Q] \qquad [P \wedge \neg b]\ S_2\ [Q]}{[P]\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ [Q]} \qquad \text{(Conditional)}$$

$$\frac{P \wedge b \Rightarrow E \geq 0 \qquad [P \wedge b \wedge (E = n)]\ S\ [P \wedge (E < n)]}{[P]\ \textbf{while}\ b\ \textbf{do}\ S\ [P \wedge \neg b]} \qquad \text{(While)}$$

where n is an auxiliary variable not appearing elsewhere