In this episode of The Verification Corner, Rustan Leino talks about Loop Invariants. He gives a brief summary of the theoretical foundations and shows how a program can sometimes be systematically constructed from its specifications.

Verification Corner

RiSE

Microsoft Research

$$\{P\}\ S\ \{Q\}$$

This slide shows a *Hoare triple*, which is used to reason about programs.

S denotes a program statement, like “x := x+y;” or “if (x < 0) x := -x;”. P and Q are conditions (also called assertions, or predicates, or boolean expressions), like “x < 10” or “x is even”.

An execution of S starts in some state and ends in some state. (We restrict our attention to program statements that are well behaved and terminate.) A Hoare triple tells you something about executions of S. The Hoare triple on this slide says that any execution of S that starts in a state satisfying P ends in some state satisfying Q.

For example,

```
  { x*x = y }  if (x < 0) x := -x;  { x*x = y }
```

says that if the statement “if (x < 0) x := -x;” is executed from a state where y equals the square of x, then it will end in a state where, again, x*x = y.

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

$$\{y=10\}\ x := y\ \{x=10\}$$

This slide shows the general form of a valid Hoare triple for an assignment statement "x := E". It says that the statement "x := E" will end in a state satisfying Q if it is started in a state satisfying "Q[E/x]", which stands for the expression Q in which any mention of variable x is replaced by the expression E.

For example, if Q is "2*x + y ≤ 11" and E is "y+1", then "Q[E/x]" is "(2*x + y ≤ 11)[y+1 / x]", which is "2*(y+1) + y ≤ 11", which can be simplified as "3*y + 2 ≤ 11", that is, "y ≤ 3". In other words, the following is a valid Hoare triple:

{ y ≤ 3 } x := y + 1 { 2*x + y ≤ 11 }

Among other things, it lets us conclude that if we execute statement "x := y + 1" from a state where x is 5 and y is 1, then it will end in a state where "2*x + y ≤ 11".

The second Hoare triple on the slide shows another example: if you want to execute "x := y" and be sure that it ends in a state satisfying "x = 10", then you must start the execution in a state where "y = 10".

$$\{x < 10\}\ x := x + 1\ \{x \leq 10\}$$

$$\{(3 \cdot x + 2 \cdot y)^2 = 81\}\ x := 3 \cdot x + 2 \cdot y\ \{x^2 = 81\}$$

Here are two more examples of valid Hoare triples for assignments.

The first says that if you want the increment-by-1 statement to end in a state where the variable is at most 10, then you must start the execution in a state where the variable is strictly less than 10. The pre-state of this Hoare triple is obtained by replacing x by “x + 1” in the condition on the post-state, and then simplifying “x+1 ≤ 10” to “x < 10”.

The second, which is more complicated, considers the statement “x := 3*x + 2*y” and the post-state condition “x*x = 81”. Again, one can figure out the most general condition on the pre-state that makes the Hoare triple valid by replacing x in the post-state condition by the right-hand side of the assignment. Without any further (thinking or) simplification, that pre-state condition is “(3*x + 2*y)*(3*x + 2*y) = 81”.

$$\{P\}\ \underline{\text{while}}\ B\ \underline{\text{do}}\ S\ \{Q\}$$

$$P \Longrightarrow J$$
$$\{J \wedge B\}\ S\ \{J\}$$
$$J \wedge \neg B \Longrightarrow Q$$

This slide considers another statement, namely a simple while loop with a loop guard B and a loop body S. If you want to be sure that every execution of the loop statement that starts in P ends in Q, then you must find a condition J that satisfies the three provisos at the bottom. The condition J is called a *loop invariant*.

The first proviso says that the loop invariant must hold before the loop statement is executed. It is expressed by a logical implication: P ==> J (pronounced “P implies J”). This proviso is analogous to the base case of a mathematical proof by induction.

The second proviso says that taking a loop iteration must maintain the loop invariant. More precisely, if the loop body, S, is started in a state where the loop invariant holds and the loop guard evaluates to true, then it ends in a state where the loop invariant holds again. This proviso is analogous to the induction case of a mathematical proof by induction.

Continued…

$$\{P\}\ \underline{\text{while}}\ B\ \underline{\text{do}}\ S\ \{Q\}$$

$$P \Longrightarrow J$$
$$\{J \wedge B\}\ S\ \{J\}$$
$$J \wedge \neg B \Longrightarrow Q$$

The third proviso says that one must pick a loop invariant that is strong enough to conclude the desired Q after the loop terminates. More precisely, it says that if the loop invariant holds and the loop guard evaluates to false, then Q holds. This proviso is analogous, in mathematics, to making sure the induction hypothesis is strong enough to conclude the theorem one is trying to prove.

For example, suppose we are trying to establishing the following Hoare triple:

{ n = 0 } while n < 99 do n := n + 2 { x = 100 }

Let's pick the loop invariant, J, to be "n is even and n ≤ 100". The first proviso is:

n = 0 ==> n is even and n ≤ 100

which holds, because 0 is even and no greater than 100. continued…

$\{P\}$ $\underline{\text{while}}$ B $\underline{\text{do}}$ S $\{Q\}$

$$P \Longrightarrow J$$
$$\{J \wedge B\}\ S\ \{J\}$$
$$J \wedge \neg B \Longrightarrow Q$$

The second proviso is:

{ n is even and n ≤ 100 and n < 99 } n := n + 2 { n is even and n ≤ 100 }

We check this Hoare triple by replacing n by "n+2" in the post-state condition, which gives us:

(n+2) is even and n+2 ≤ 100

which simplifies to:

n+2 is even and n ≤ 98

We note that the pre-state condition in the triple also simplifies to the same expression, so the proviso holds. Continued…

$$\{P\} \ \underline{\text{while}}\ B\ \underline{\text{do}}\ S\ \{Q\}$$

$$P \Longrightarrow J$$
$$\{J \land B\}\ S\ \{J\}$$
$$J \land \neg B \Longrightarrow Q$$

The third proviso is:

n is even and n ≤ 100 and not n < 99 ==> x = 100

If n is at most 100 but not less than 99, then n is either 99 or 100. Furthermore, if n is even, then n can only be 100, which is what we are trying to prove. That proves the example Hoare triple to be valid.

To fully reason about loops, one also needs to consider termination. However, in this episode of Verification Corner, we ignore that issue; come back for a different episode on termination.