Lecture Notes:
Hoare Logic

15-819O: Program Analysis Jonathan Aldrich
jonathan.aldrich@cs.cmu.edu

Revised March 2013

1 Hoare Logic

The goal of Hoare logic is to provide a formal system for reasoning about program correctness. Hoare logic is based on the idea of a specification as a contract between the implementation of a function and its clients. The specification is made up of a precondition and a postcondition. The precondition is a predicate describing the condition the function relies on for correct operation; the client must fulfill this condition. The postcondition is a predicate describing the condition the function establishes after correctly running; the client can rely on this condition being true after the call to the function.

The implementation of a function is *partially correct* with respect to its specification if, assuming the precondition is true just before the function executes, then if the function terminates, the postcondition is true. The implementation is *totally correct* if, again assuming the precondition is true before function executes, the function is guaranteed to terminate and when it does, the postcondition is true. Thus total correctness is partial correctness plus termination.

Note that if a client calls a function without fulfilling its precondition, the function can behave in any way at all and still be correct. Therefore, if it is intended that a function be robust to errors, the precondition should include the possibility of erroneous input and the postcondition should describe what should happen in case of that input (e.g. a specific exception being thrown).

Hoare logic uses Hoare Triples to reason about program correctness. A Hoare Triple is of the form $\{P\}\ S\ \{Q\}$, where P is the precondition, Q is the postcondition, and S is the statement(s) that implement the function. The (total correctness) meaning of a triple $\{P\}\ S\ \{Q\}$ is that if we start in a state where P is true and execute S, then S will terminate in a state where Q is true.

Consider the Hoare triple $\{x = 5\}\ x := x{*}2\ \{x > 0\}$. This triple is clearly correct, because if $x = 5$ and we multiply x by 2, we get $x = 10$ which clearly implies that $x > 0$. However, although correct, this Hoare triple is not a precise as we might like. Specifically, we could write a stronger postcondition, i.e. one that implies $x > 0$. For example, $x > 5 \wedge x < 20$ is stronger because it is more informative; it pins down the value of x more precisely than $x > 0$. The strongest postcondition possible is $x = 10$; this is the most useful postcondition. Formally, if $\{P\}\ S\ \{Q\}$ and for all Q such that $\{P\}\ S\ \{Q\}$, $Q \Rightarrow Q$, then Q is the strongest postcondition of S with respect to P.

We can compute the strongest postcondition for a given statement and precondition using the function $sp(S, P)$. Consider the case of a statement of the form $x := E$. If the condition P held before the statement, we now know that P still holds and that $x = E$—where P and E are now in terms of the old, pre-assigned value of x. For example, if P is $x + y = 5$, and S is $x := x + z$, then we should know that $x' + y = 5$ and $x = x' + z$, where x' is the old value of x. The program semantics doesn't keep track of the old value of x, but we can express it by introducing a fresh, existentially quantified variable x'. This gives us the following strongest postcondition for assignment:

$$sp(x := E, P) \quad = \exists x'.[x'/x]P \wedge x = [x'/x]E$$

As described in earlier lectures, the operation $[x'/x]E$ denotes the capture-avoiding substitution of x′ for x in E; we rename bound variables as we do the substitution so as to avoid conflicts.

While this scheme is workable, it is awkward to existentially quantify over a fresh variable at every statement; the formulas produced become unnecessarily complicated, and if we want to use automated theorem provers, the additional quantification tends to cause problems. Dijkstra proposed reasoning instead in terms of *weakest preconditions*, which turns out to work more clearly. If $\{P\}\ S\ \{Q\}$ and for all P such that $\{P\}\ S\ \{Q\}$, $P \Rightarrow P$, then P is the weakest precondition $wp(S, Q)$ of S with respect to Q.

We can define a function yielding the weakest precondition with respect to some postcondition for assignments, statement sequences, and if statements, as follows:

$$\begin{aligned}
wp(x := E, P) &= [E/x]P \\
wp(S;\ T, Q) &= wp(S,\ wp(T, Q)) \\
wp(\text{if } B \text{ then } S \text{ else } T, Q) &= B \Rightarrow wp(S, Q) \land \neg B \Rightarrow wp(T, Q)
\end{aligned}$$

Verifying loop statements of the form while b do S is more difficult. Because it may not be obvious how many times the loop will execute—and it may in fact be impossible even to bound the number of executions—we cannot mechanically generate a weakest precondition in the style above. Instead, we must reason about the loop inductively.

Intuitively, any loop that is trying to compute a result must operate by establishing that result one step at a time. What we need is an inductive proof that shows that each time the loop executes, we get one step closer to the final result—and that when the loop is complete (i.e. the loop condition is false) that result has been obtained. This reasoning style requires us to write down what we have computed so far after an arbitrary number of iterations; this will serve as an induction hypothesis. The literature calls this form of induction hypothesis a *loop invariant*, because it represents a condition that is always true (i.e. invariant) before and after each execution of the loop.

In order to be used in an inductive proof of a loop, a loop invariant must fulfill the following conditions:

- $P \Rightarrow I$: The invariant is initially true. This condition is necessary as a base case, to establish the induction hypothesis.

- $\{Inv \land B\}\ S\ \{Inv\}$: Each execution of the loop preserves the invariant. This isn the inductive case of the proof.

- $(Inv \land \neg B) \Rightarrow Q$: The invariant and the loop exit condition imply the postcondition. This condition is simply demonstrating that the induction hypothesis/loop invariant we have chosen is sufficiently strong to prove our postcondition S.

The procedure outlined above only verifies partial correctness, because it does not reason about how many times the loop may execute. In order to verify full correctness, we must place an upper bound on the number of remaining times the loop body will execute. This bound is typically called a *variant function*, written v, because it is variant: we must prove that it decreases each time we go through the loop. If we can also prove that whenever the bound v is equal to (or less than) zero, the loop condition will be false, then we have verified that the loop will terminate.

More formally, we must come up with an integer-valued variant function v that fulfils the following conditions:

- $Inv \wedge v \leq 0 \Rightarrow \neg B$: If we are entering the loop body (i.e. the loop condition B evaluates to true) and the invariant holds, then v must be strictly positive.
- $\{Inv \wedge B \wedge v = V\}\ S\ \{v < V\}$: The value of the variant function decreases each time the loop body executes (here V is a placeholder constant representing the "old" value of v).

2 Proofs with Hoare Logic

Consider the WHILE program used in the previous lecture notes:

```
$r := 1;$
$i := 0;$
while $i < m$ do
    $r := r * n;$
    $i := i + 1$
```

We wish to prove that this function computes the nth power of m and leaves the result in r. We can state this with the postcondition $r = n^m$.

Next, we need to determine the proper precondition. We cannot simply compute it with wp because we do not yet know what the right loop invariant is—and in fact, different loop invariants could lead to different preconditions. However, a bit of reasoning will help. We must have $m \geq 0$ because we have no provision for dividing by n, and we avoid the problematic computation of 0^0 by assuming $n > 0$. Thus our precondition will be $m \geq 0 \wedge n > 0$.

We need to choose a loop invariant. A good hueristic for choosing a loop invariant is often to modify the postcondition of the loop to make it depend on the loop index instead of some other variable. Since the loop index runs from i to m, we can guess that we should replace m with i in the postcondition $r = n^m$. This gives us a first guess that the loop invariant should include $r = n^i$.

This loop invariant is not strong enough (doesn't have enough information), however, because the loop invariant conjoined with the loop exit condition should imply the postcondition. The loop exit condition is $i \geq m$, but we need to know that $i = m$. We can get this if we add $i \leq m$ to the loop invariant. In addition, for proving the loop body correct, we will also need to add $0 \leq i$ and $n > 0$ to the loop invariant. Thus our full loop invariant will be $r = n^i \wedge 0 \leq i \leq m \wedge n > 0$.

In order to prove full correctness, we need to state a variant function for the loop that can be used to show that the loop will terminate. In this case $m - i$ is a natural choice, because it is positive at each entry to the loop and decreases with each loop iteration.

Our next task is to use weakest preconditions to generate proof obligations that will verify the correctness of the specification. We will first ensure that the invariant is initially true when the loop is reached, by propagating that invariant past the first two statements in the program:

$$\begin{array}{l} \{m \geq 0 \wedge n > 0\} \\ r := 1; \\ i := 0; \\ \{r = n^i \wedge 0 \leq i \leq m \wedge n > 0\} \end{array}$$

We propagate the loop invariant past $i := 0$ to get $r = n^0 \wedge 0 \leq 0 \leq m \wedge n > 0$. We propagate this past $r := 1$ to get $1 = n^0 \wedge 0 \leq 0 \leq m \wedge n > 0$. Thus our proof obligation is to show that:

$$\begin{array}{l} m \geq 0 \wedge n > 0 \\ \Rightarrow 1 = n^0 \wedge 0 \leq 0 \leq m \wedge n > 0 \end{array}$$

We prove this with the following logic:

$m \geq 0 \wedge n > 0$	by assumption
$1 = n^0$	because $n^0 = 1$ for all $n > 0$ and we know $n > 0$
$0 \leq 0$	by definition of $\leq$
$0 \leq m$	because $m \geq 0$ by assumption
$n > 0$	by the assumption above
$1 = n^0 \wedge 0 \leq 0 \leq m \wedge n > 0$	by conjunction of the above

We now apply weakest preconditions to the body of the loop. We will first prove the invariant is maintained, then prove the variant function decreases. To show the invariant is preserved, we have:

$$\begin{array}{l} \{r = n^i \wedge 0 \leq i \leq m \wedge n > 0 \wedge i < m\} \\ r := r * n; \\ i := i + 1; \\ \{r = n^i \wedge 0 \leq i \leq m \wedge n > 0\} \end{array}$$

We propagate the invariant past $i := i + 1$ to get $r = n^{i+1} \wedge 0 \leq i + 1 \leq m \wedge n > 0$. We propagate this past $r := r * n$ to get: $r * n = n^{i+1} \wedge 0 \leq i + 1 \leq m \wedge n > 0$. Our proof obligation is therefore:

$$\begin{aligned}&r = n^i \wedge 0 \leq i \leq m \wedge n > 0 \wedge i < m\\ &\Rightarrow r * n = n^{i+1} \wedge 0 \leq i + 1 \leq m \wedge n > 0\end{aligned}$$

We can prove this as follows:

$r = n^i \wedge 0 \leq i \leq m \wedge n > 0 \wedge i < m$	by assumption
$r * n = n^i * n$	multiplying by n
$r * n = n^{i+1}$	definition of exponentiation
$0 \leq i + 1$	because $0 \leq i$
$i + 1 < m + 1$	by adding 1 to inequality
$i + 1 \leq m$	by definition of $\leq$
$n > 0$	by assumption
$r * n = n^{i+1} \wedge 0 \leq i + 1 \leq m \wedge n > 0$	by conjunction of the above

We have a proof obligation to show that the variant function is positive when we enter the loop. The obligation is to show that the loop invariant and the entry condition imply this:

$$\begin{aligned}&r = n^i \wedge 0 \leq i \leq m \wedge n > 0 \wedge i < m\\ &\Rightarrow m - i > 0\end{aligned}$$

The proof is trivial:

$r = n^i \wedge 0 \leq i \leq m \wedge n > 0 \wedge i < m$	by assumption
$i < m$	by assumption
$m - i > 0$	subtracting i from both sides

We also need to show that the variant function decreases. We generate the proof obligation using weakest preconditions:

```
$\{r = n^i \wedge 0 \leq i \leq m \wedge n > 0 \wedge i < m \wedge m - i = V\}$
$r := r * n;$
$i := i + 1;$
$\{m - i < V\}$
```

We propagate the condition past $i := i + 1$ to get $m - (i + 1) < V$. Propagating past the next statement has no effect. Our proof obligation is therefore:

$$\begin{aligned}&r = n^i \land 0 \leq i \leq m \land n > 0 \land i < m \land m - i = V\\ &\Rightarrow m - (i+1) < V\end{aligned}$$

Again, the proof is easy:

$r = n^i \land 0 \leq i \leq m \land n > 0 \land i < m \land m - i = V$	by assumption
$m - i = V$	by assumption
$m - i - 1 < V$	by definition of $<$
$m - (i+1) < V$	by arithmetic rules

Last, we need to prove that the postcondition holds when we exit the loop. We have already hinted at why this will be so when we chose the loop invariant. However, we can state the proof obligation formally:

$$\begin{aligned}&r = n^i \land 0 \leq i \leq m \land n > 0 \land i \geq m\\ &\Rightarrow r = n^m\end{aligned}$$

We can prove it as follows:

$r = n^i \land 0 \leq i \leq m \land n > 0 \land i \geq m$	by assumption
$i = m$	because $i \leq m$ and $i \geq m$
$r = n^m$	substituting m for i in assumption