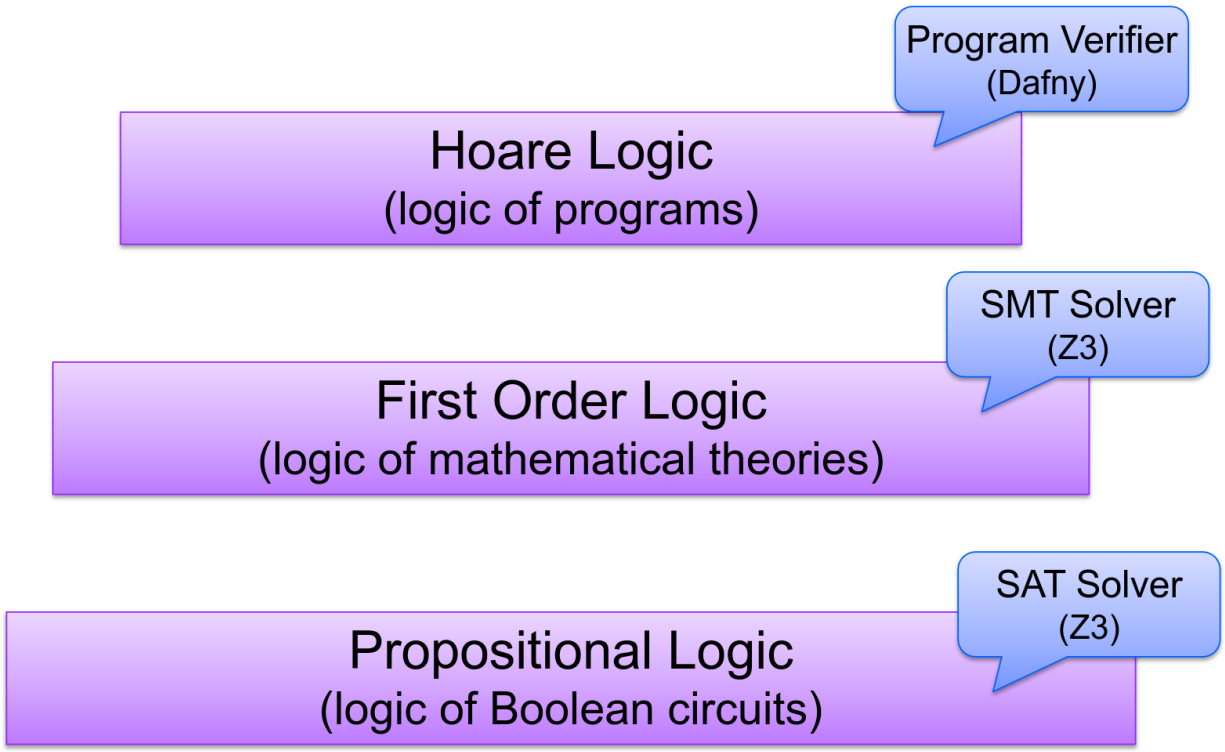
Software Testing, Quality Assurance and Maintenance Winter 2025

Lecture 12 — February 26, 2025

Patrick Lam *version 2*

We've seen this picture on the slides.



This lecture is about Hoare Logic. Again, you've seen this in SE 212, but it's worth a review as we prepare to move onto Dafny. Some of the concepts are subtle!

Axiomatic Semantics. You used axiomatic semantics in SE 212. It consists of:

- a language for stating assertions about programs; and
- rules for establishing the truth of assertions.

You have used axiomatic semantics to make and prove assertions. Some examples:

- this program terminates;
- if this program terminates, the variables `x` and `y` have the same value throughout the execution of the program;
- array accesses are within array bounds.

One can use first-order logic for assertions. There are also special-purpose specification languages, which I'll name-check but not discuss: Z (the OG specification language), Larch, JML. And there

are other logics as well: temporal logic, especially useful for specifying properties of concurrent systems; linear logic, which has been used under-the-hood for Rust uniqueness; and separation logic, for reasoning about the structure of the heap.

Hoare Triples. We write assertions about WHILE programs using Hoare triples:

$$\{A\}\ c\ \{B\}$$

which means that (1) if A holds in state q, and (2) if the semantics says $q \to q'$, then (3) B holds in q'.

One can thus write the valid assertion

$$\{y \leq x\} z := x; z := z + 1 \{y < z\}$$

which means that if $y \leq x$ and you run those two statements, you know that $y < z$ at the end of them.

Partial versus total correctness. More specifically, $\{A\}\ c\ \{B\}$ is a *partial* correctness assertion, and does not imply termination of c.

If A holds in state q, and if there exists q' such that $q \to q'$, then B holds in state q'. So, if there is no q' (i.e. c does not terminate, or gets stuck), then there is, vacuously, no obligation to show B.

There is the notion of *total* correctness as well, though much less often used. $[A]\ c\ [B]$ is the notation for *total* correctness.

If A holds in state q, **then** there exists q' such that $q \to q'$, and B holds in state q'.

More formal Hoare logic

We have assertions like A and B. We'll formalize the language we use for them, define when an assertion holds in a state, and define rules for deriving valid Hoare triples.

Our assertion language. We use *first-order predicate logic* and use WHILE expressions as atoms.

$$\begin{array}{rcl} A & ::== & \mathsf{true} \mid \mathsf{false} \mid e_1 = e_2 \mid e_1 \geq e_2 \\ & \mid & A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \Rightarrow A_2 \mid \forall x.A \mid \exists x.A \end{array}$$

There are logical variables (e.g. $\forall x$) and program variables. We don't distinguish them. For our purpose, all WHILE variables range over integers.

It is always valid to use a WHILE Boolean expression as an assertion.

Semantics of assertions. We write

$$q \models A$$

to mean that assertion A holds in state q. (It is well-defined when q is defined on all variables occuring free in A).

We define $\models$ inductively on the structure of assertions. (Start with the atomic assertions and arithmetic expressions from WHILE, and then build up the compound assertions like $\wedge$ etc.)

$q \models \mathsf{true}$	always
$q \models e_1 = e_2$	iff $\langle e_1, q\rangle \Downarrow = \langle e_2, q\rangle \Downarrow$
$q \models e_1 \geq e_2$	iff $\langle e_1, q\rangle \Downarrow \geq \langle e_2, q\rangle \Downarrow$
$q \models A_1 \wedge A_2$	iff $q \models A_1$ and $q \models A_2$
$q \models A_1 \vee A_2$	iff $q \models A_1$ or $q \models A_2$
$q \models A_1 \Rightarrow A_2$	iff $q \models A_1$ implies $q \models A_2$
$q \models \forall x.\ A$	iff $\forall n \in \mathbb{Z}.\ q[x := n] \models A$
$q \models \exists x.\ A$	iff $\exists n \in \mathbb{Z}.\ q[x := n] \models A$

Semantics of Hoare Triples. In symbols, then, partial correctness means:

$$\models \{A\}\ c\ \{B\} \text{ iff}$$
$$\forall q, q' \in Q.\ q \models A \wedge q \xrightarrow{c} q' \Rightarrow q' \models B$$

Total correctness means:

$$\models [A]\ c\ [B] \text{ iff}$$
$$\forall q \in Q.\ q \models A \Rightarrow \exists q' \in Q.\ q \xrightarrow{c} q' \wedge q' \models B$$

Quantifier selection matters!

Examples of Hoare Triples. Let's fill in the blanks for these Hoare triples. I'll put the blank versions first.

$\{\ \mathsf{true}\ \}$	$x := 5$	$\{\quad\}$
$\{\quad\}$	$x := x + 3$	$\{x = y + 3\}$
$\{\quad\}$	$x := x * 2 + 3$	$\{x > 1\}$
$\{x = a\}$	if $x < 0$ then $x := -x$	$\{\quad\}$
$\{\ \mathsf{false}\ \}$	$x := 3$	$\{\quad\}$
$\{x > 0\}$	while $x! = 0$ do $x := x - 1$	$\{\quad\}$
$\{x > 0\}$	while $x! = 0$ do $x := x - 1$	$\{\quad\}$

You can find the solutions at the end of this document.

Inference rules

We do now have a definition for $\{A\}\ c\ \{B\}$. But the definition relies heavily on having $q \xrightarrow{c} q'$, which is defined by the operational semantics. This is inconvenient: we pretty much have to run

the program to verify an assertion. And it's even harder when there are loops. And, if you want to verify $\forall x.\ A$, you have to run the program for all possible x, which is hard.

This is why you used inference rules and the axiomatic semantics in SE 212. We can symbolically verify assertions without running the program.

Inference rules for First-Order Logic.

The notation $\vdash A$ means that A can be inferred from the axioms. $B \vdash A$ means that A can be inferred from B.

We are going to use natural deduction style rules. Substitution is always confusing, so as a reminder, $A[a/x]$ means A with variable x replaced by term a ("substitute a for x").

$$\frac{A \quad B}{A \wedge B} \qquad \frac{A}{A \vee B} \qquad \frac{B}{A \vee B} \qquad \frac{A \to B \qquad A}{B}$$

$$\frac{A[e/x]}{\exists x.A} \qquad \frac{\forall x.\ A}{A[e/x]} \qquad \frac{A[a/x]}{\forall x.\ A}\ a \text{ fresh}$$

$$\frac{A \vdash B}{A \Rightarrow B} \qquad \frac{\vdash \exists x.\ A \quad A[a/x] \vdash B}{\vdash B}\ a \text{ fresh}$$

Inference rules for Hoare triples

We write $\vdash \{A\}\ c\ \{B\}$ when we can derive the triple using inference rules.

There is one inference rule for each command in WHILE.

Rule of Consequence. It is also always allowed to strengthen the pre-condition or weaken the post-condition.

$$\frac{\vdash A' \Rightarrow A \quad \{A\}\ c\ \{B\} \quad \vdash B \Rightarrow B'}{\{A'\}\ c\ \{B'\}}\ \textsc{Conseq}$$

For example, let's look at this Hoare triple.

$$\{x = 2\}\ \texttt{x := 5}\ \{x = 5\}$$

Can we weaken the precondition? That is, are there other pre-states where executing `x := 5` will still result in postcondition $x = 5$? Yep. If we start with precondition $x = 19$ and run `x := 5` we still get postcondition $x = 5$. Indeed, in *any* pre-state, running `x := 5` results in $x = 5$ being true.

So, the weakest possible precondition here is `true`, and we generally want to put the weakest precondition that we can—the one that restricts allowable states least.

$$\{\texttt{true}\}\ \texttt{x := 5}\ \{x = 5\}$$

What about strengthening the postcondition? Let's look now at this Hoare triple.

$$\{\texttt{true}\}\ \texttt{x := 5}\ \{\texttt{true}\}$$

This isn't saying anything useful. It is saying that you can execute `x := 5` in any state, and you know nothing ("true") after running it. No matter what happens, this Hoare triple is going to be valid.

Consider instead

$$\{\texttt{true}\}\ \texttt{x := 5}\ \{x > 0\}$$

This is still valid, and it's slightly more useful—a stronger postcondition, which restricts possible states more. The strongest (most restrictive) postcondition, which tells you the most about what this statement is doing, is what we had before,

$$\{\texttt{true}\}\ \texttt{x := 5}\ \{x = 5\}$$

The Rules. Here they are.

$$\{\text{A}\}\ \texttt{skip}\ \{\text{A}\} \qquad \frac{\{A\}\ s_1\ \{B\}\quad \{B\}\ s_2\ \{C\}}{\{A\}\ s_1\ \texttt{;}\ s_2\ \{C\}}$$

$$\{\text{A[e/x]}\}\ \texttt{x := e}\ \{\text{A}\} \qquad \frac{\{A \wedge b\}\ s_1\ \{B\}\quad \{A \wedge \neg b\}\ s_2\ \{B\}}{\{A\}\ \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \{B\}}$$

$$\frac{\{I \wedge b\}\ s\ \{I\}}{\{I\}\ \texttt{while}\ b\ \texttt{do}\ s\ \{I \wedge \neg b\}}$$

About that Assignment Rule. It is, yes, a bit surprising. Why is it backwards-looking?

$$\{A[e/x]\}\ \texttt{x := e}\ \{A\}$$

In the semantics, the rules always went from a state q to its successor q' (even if sometimes we worked backwards in writing derivations). One way to think about this assignment rule is that you have the postcondition A. The rule tells us what is the weakest poscondition that we can write such that, after executing assignment `x := e`, we have postcondition A.

So, what about

$$\{\qquad\qquad\}\ \texttt{x := y}\ \{x = 1\}$$

Here, we are given postcondition $\texttt{x} = 1$. The assignment statement copies the value from `y` into `x`. Thus, the weakest precondition yielding the given postcondition is

$$\{y = 1\}\ \texttt{x := y}\ \{x = 1\}$$

We can interpret this Hoare triple as: if `y` starts with value 1 and we copy that value into `x`, then we know `x` finishes with value 1. This is the weakest precondition because any other value of `y` is not going to yield $x = 1$ at the end. The triple, in general, transfers the property of being 1 from `y` to `x`.

Now, let $y + z$ be equal to 1 in the precondition

$$\{y + z = 1\}\ \texttt{x := y+z}\ \{?\}$$

Then, the strongest postcondition has x being equal to 1 (though you are allowed to say less).

$$\{y + z = 1\}\ \texttt{x := y+z}\ \{x = 1\}$$

Let's generalize a bit more.

$$\{a = 1\}\ \texttt{x := a}\ \{x = 1\}$$

We are still transferring the 1 from expression a (it doesn't have to be a variable, it's logic) to x.

And finally, we can get the full assignment rule

$$\{A[e/x]\}\ \texttt{x := e}\ \{A\}$$

We get to state A after assignment `x := e` if we had started, in the precondition, with a state like A except with term e in the precondition instead of the x we have in the postcondition. (It is actually allowed to replace any number of xs in the postcondition, including 0, to form the precondition.)

Here is an example.

$$\{x \leq 5[x + 1/x]\}\ \texttt{x := x + 1}\ \{x \leq 5\}$$

Doing the substitution, we have

$$\{x + 1 \leq 5\}\ \texttt{x := x + 1}\ \{x \leq 5\}$$

which is the weakest possible precondition that gives you $x \leq 5$ as postcondition.

Example: Conditional We deduce a postcondition that is true no matter which branch of the conditional we take.

$$\frac{D_1 :: \{\mathsf{true} \wedge y \leq 0\}\ \texttt{x := 1}\ \{x > 0\} \qquad D_2 :: \{\mathsf{true} \wedge y > 0\}\ \texttt{x := y}\ \{x > 0\}}{\{\mathsf{true}\}\ \mathsf{if}\ y \leq 0\ \mathsf{then}\ x := 1\ \mathsf{else}\ x := y\ \{x > 0\}}$$

We can deduce D_1 by the rules of consequence and assignment. In particular, we strengthen the precondition for the rule of assignment.

$$\frac{\mathsf{true} \wedge y \leq 0 \Rightarrow 1 > 0 \qquad \{1 > 0\}\ \texttt{x := 1}\ \{x > 0\}}{\{\mathsf{true} \wedge y \leq 0\}\ \texttt{x := 1}\ \{x > 0\}}$$

Similarly for D_2:

$$\frac{\mathsf{true} \wedge y > 0 \Rightarrow y > 0 \qquad \{y > 0\}\ \texttt{x := y}\ \{x > 0\}}{\{\mathsf{true} \wedge y > 0\}\ \texttt{x := y}\ \{x > 0\}}$$

Exercise: Alternative Assignment Rule. Is this rule correct?

$$\vdash \{\mathsf{true}\}\ \texttt{x := e}\ \{x = e\}$$

It's appealing, but no. If we used that rule, then we could deduce the following incorrect Hoare triple.

$$\vdash \{\mathsf{true}\}\ \texttt{x := x + 1}\ \{x = x + 1\}$$

which boils down to postcondition $0 = 1$, or false, which isn't useful.

Actually correct alternative rules. We can actually use this axiom for assignment:

$$\vdash \{A\}\ \texttt{x := e}\ \{\exists x_0.\texttt{x = e}[x_0/x] \wedge A[x_0/x]\}$$

In some sense, it is creating a copy of e with the original x substituted by fresh x_0, setting x to that copy, and also substituting the same fresh x_0 into A in place of x.

For while loops, we could use:

$$\frac{\vdash I \wedge b \Rightarrow C \qquad \{C\}\ c\ \{I\} \qquad \vdash I \wedge \neg b \Rightarrow B}{\vdash \{I\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{C\}}$$

These rules are derivable from the previous rules plus the rule of consequence.

Example: a simple loop. We want to infer the Hoare triple

$$\{x \leq 0\}\ \texttt{while}\ x \leq 5\ \texttt{do}\ x := x + 1\ \{x = 6\}$$

which says that, starting with x not positive, incrementing it while $x \leq 5$ results in final state $x = 6$.

We choose loop invariant $I : x \leq 6$ and use the while and assignment rules, along with the rule of consequence.

$$\frac{\dfrac{x \leq 6 \wedge x \leq 5 \Rightarrow x + 1 \leq 6 \qquad \{x + 1 \leq 6\}\ \texttt{x := x + 1}\ \{x \leq 6\}}{\{x \leq 6 \wedge x \leq 5\}\ \texttt{x := x + 1}\ \{x \leq 6\}}}{\{x \leq 6\}\ \texttt{while x <= 5 do x := x + 1}\ \{x \leq 6 \wedge x > 5\}}$$

We finally apply the rule of consequence.

$$\frac{x \leq 0 \Rightarrow x \leq 6 \qquad x \leq 6 \wedge x > 5 \Rightarrow x = 6 \qquad \{x \leq 6\}\ \texttt{while ...}\ \{x \leq 6 \wedge x > 5\}}{\{x \leq 0\}\ \texttt{while x <= 5 do x := x + 1}\ \{x = 6\}}$$

Inductive Loop Invariants. Here is another concept that you've seen before in SE 212, but which is well worth repeating, as you'll need this to use Dafny.

We want to be able to reason about loops without having to unroll the loop an unbounded number of times. The key is to find a *loop invariant* which is true no matter how many times the loop executes. Then we show that the loop invariant holds upon entry to the loop and is preserved by the loop body. We also need to show that it is strong enough to show what we need after the loop. Also, inside the loop, we know that the loop condition holds (that's why we're inside the loop), and upon exit from the loop, the negation of the loop condition holds (which allows us to exit).

$$\frac{\text{Pre} \Rightarrow \text{Inv} \quad \{\text{Inv} \wedge b\}\ s\ \{\text{Inv}\} \quad \{\text{Inv} \wedge \neg b\}\ s\ \{\text{Post}\}}{\{\text{Pre}\}\ \texttt{while}\ b\ \texttt{do}\ s\ \{\text{Post}\}}$$

Specifically, Inv is an *inductive loop invariant* if the following three conditions holdD:

- (Initiation): Inv holds *initially* whenever the loop is reached; i.e. the pre-condition of the loop, Pre, implies Inv.
- (Consecution): Inv is *preserved*: if Inv$\wedge b$ holds before executing the loop body, then executing all the statements in the loop body will end in a state that also satisfies Inv.
- (Safety): Inv $\wedge \neg b$ implies the desired postcondition Post.

Simple Loop using Inductive Invariants. Consider again the loop:

$$\{x \leq 0\}\ \texttt{while}\ x \leq 5\ \texttt{do}\ x := x + 1\ \{x = 6\}$$

but now we use inductive invariant $x \leq 6$. One application of the inductive invariant rule above:

$$\frac{x \leq 0 \Rightarrow x \leq 6 \quad \{x \leq 7 \wedge x \leq 5\}\ \texttt{x := x + 1}\ \{x \leq 6\} \quad x > 5 \wedge x \leq 6 \Rightarrow x = 6}{\{x \leq 0\}\ \texttt{while}\ x \leq 5\ \texttt{do}\ x := x + 1\ \{x = 6\}}$$

Example: A more interesting program. Here's a more challenging program. We are actually going to reason about multiplication.

```
1 // { n \geq 0 }
2 p := 0;
3 x := 0;
4 while x < n do
5   x := x + 1;
6   p := p + m
7 // { p = n * m }
```

This is a composition of three statements, so the only rule available to us (aside from the rule of consequence) is that for sequential composition. rule.

$$\frac{\{A\}\ s_1\ \{B\} \quad \{B\}\ s_2\ \{C\}}{\{A\}\ s_1\ ;\ s_2\ \{C\}}$$

In particular, start with $A = \{n \geq 0\}, B = \{p = n{*}m\}$, and $c_1 =$ `p := 0; x := 0`, $c_2 =$ `while ...`. Then we have:

$$\frac{\{n \geq 0\}\ \texttt{p := 0; x := 0}\ \{C\} \quad \{C\}\ \texttt{while x < n do x := x + 1; p := p + m}\ \{p = n * m\}}{\{n \geq 0\}\ \texttt{p := 0; x := 0; while x < n do x := x + 1; p := p + m}\ \{p = n * m\}}$$

for some C that we have to figure out. What could C be? We are better off figuring out what precondition we need for the while loop c_2. We have to use the rule for `while`. But that rule is

$$\frac{\{I \wedge b\}\ c\ \{I\}}{\{I\}\ \texttt{while}\ b\ \texttt{do}\ c\ \{I \wedge \neg b\}}$$

and we have $p = n * m$, which is not of the form $I \wedge \neg b$. Let $I = C$, and apply the rule of consequence, furthermore with $A = A' = I$, $B' = n * m$, and $B = I \wedge b = I \wedge x \geq n$. Recall that the rule of consequence is:

$$\frac{A' \Rightarrow A \quad \{A\}\ c\ \{B\} \quad B \Rightarrow B'}{\{A'\}\ c\ \{B'\}}$$

So we can convert $\{C\}$ `while x < n do x := x + 1; p := p + m` $\{p = n{*}m\}$ with the variable assignment above:

$$\frac{I \Rightarrow I \quad \{I\}\ \texttt{while x < n do x := x + 1; p := p + m}\ \{I \wedge x \geq n\} \quad I \wedge x \geq n \Rightarrow p = n * m}{\{I\}\ \texttt{while x < n do x := x + 1; p := p + m}\ \{p = n * m\}}$$

if we can show $I \wedge x \geq n \Rightarrow p = n * m$ (we'll do this later).

This gives us derivation tree:

$$\frac{\{n \geq 0\}\ \texttt{p := 0; x := 0}\ \{I\} \qquad \dfrac{\{I\}\ \texttt{while ...}\ \{I \wedge x \geq n\} \quad I \wedge x \geq n \Rightarrow p = n * m}{\{I\}\ \texttt{while ...}\ \{p = n * m\}}}{\{n \geq 0\}\ \texttt{p := 0; x := 0; while x < n do x := x + 1; p := p + m}\ \{p = n * m\}}$$

and we can actually apply the `while` rule:

$$\frac{\{n \geq 0\}\ \texttt{p := 0; x := 0}\ \{I\} \qquad \dfrac{\dfrac{\{I \wedge x < n\}\ \texttt{x := x + 1; p := p + m}\ \{I\}}{\{I\}\ \texttt{while ...}\ \{I \wedge x \geq n\}} \quad I \wedge x \geq n \Rightarrow \ldots}{\{I\}\ \texttt{while ...}\ \{p = n * m\}}}{\{n \geq 0\}\ \texttt{p := 0; x := 0; while x < n do x := x + 1; p := p + m}\ \{p = n * m\}}$$

Once again, we split compound statements, this time the body of the while loop:

$$\frac{\{I \wedge x < n\}\ \texttt{x := x + 1}\ \{C\} \quad \{C\}\ \ \texttt{p := p + m}\ \{I\}}{\{I \wedge x < n\}\ \texttt{x := x + 1; p := p + m}\ \{I\}}$$

and we will need to use the assignment rule $\{A[e/x]\}$ `x := e` $\{A\}$ on the assignment `p := p + m`, so that we get precondition $I[p + m/p]$, which we propagate to the first assignment also.

$$\frac{\dfrac{\{I \wedge x < n\}\ \texttt{x := x + 1}\ \{I[p + m/p]\} \quad \{I[p + m/p]\}\ \texttt{p := p + m}\ \{I\}}{\{I \wedge x < n\}\ \texttt{x := x + 1; p := p + m}\ \{I\}}}{\{I\}\ \texttt{while ...}\ \{I \wedge x \geq n\}}$$

We have precondition for `x := x + 1` as $I \wedge x < n$, but the assignment rule results in $I[x+1/x,\ p+m/p]$. The rule of consequence allows us to strengthen the precondition (well, we'll show the required $A' \Rightarrow A$ implication later):

$$\frac{I \wedge x < n \Rightarrow I[x+1/x, p+m/p]\} \quad \{I[x+1/x, p+m/p]\}\ \texttt{x := x + 1}\ \{I[p+m/p]\}}{\{I \wedge x < n\}\ \texttt{x := x + 1}\ \{I[p+m/p]\}}$$

For now, let's address the left-hand side of the proof tree, where we have the Hoare triple for the compound statement $\{n \geq 0\}$ `p := 0; x := 0` $\{I\}$.

$$\frac{\dfrac{n \geq 0 \Rightarrow I[0/p, 0/x] \quad \{I[0/p, 0/x]\}\ \texttt{p := 0}\ \{I[0/x]\}}{\{n \geq 0\}\ \texttt{p := 0}\ \{I[0/x]\}} \qquad \{I[0/x]\}\ \texttt{x := 0}\ \{I\}}{\{n \geq 0\}\ \texttt{p := 0; x := 0}\ \{I\}}$$

So we have the three open proof obligations for the rule of consequence:

$$\begin{aligned} n \geq 0 &\Rightarrow I[0/p, 0/x] \\ I \wedge x < n &\Rightarrow I[x+1/x, p+m/p] \\ I \wedge x \geq n &\Rightarrow p = n * m \end{aligned}$$

Can we find suitable I? In fact, we can use

$$I :: p = x * m \wedge x \leq n$$

which then results in the three statements:

$$\begin{aligned} n \geq 0 &\Rightarrow 0 = 0 * m \wedge 0 \leq n \\ p = p * m \wedge x \leq n \wedge x < n &\Rightarrow p + m = (x+1) * m \wedge x + 1 \leq n \\ p = x * m \wedge x \leq n \wedge x \geq n &\Rightarrow p = n * m \end{aligned}$$

which conveniently happens to work out. Basically, our example is, with the invariant inserted:

```
1 // { n \geq 0 }
2 p := 0;
3 x := 0;
4 while x < n inv p = x * m ∧ x ≤ n do
5   x := x + 1;
6   p := p + m
7 // { p = n * m }
```

The invariant is saying that p is x times m at every iteration, where x is the iteration count. The loop terminates after n iterations, leaving p as n times m, as required.

Using Hoare Rules

Hoare rules are mostly syntax directed. That is, there is one rule you can apply to each statement.

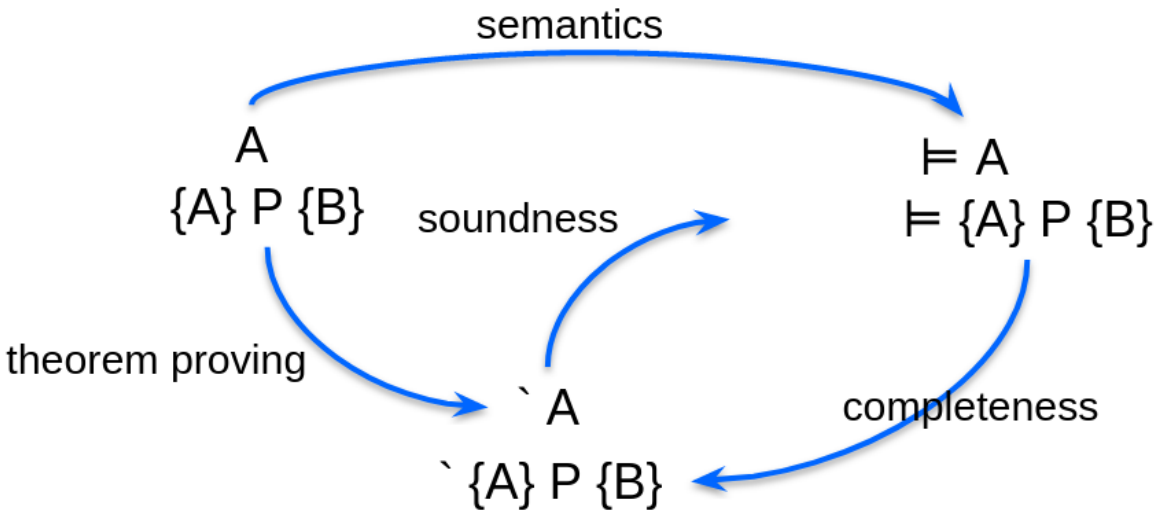
In this course, our goal is to use Dafny to automate these proofs. What are the obstacles? There has to be some sort of answer to these three questions:

- When to apply the rule of consequence?
- What invariant to use for while?
- How do you prove the implications involved in the rule of consequence?

Proving the implications, at least, is doable, using modern theorem provers. Loop invariants, though, are hard to come up with, so you'll be spending some time with that. Someone has to come up with them. For this course, it's you. In general, is it the programmer?

Hoare Logic: Summary

We have a language for asserting properties of programs, and we know when these assertions are true. We also have a symbolic method for deriving assertions. This picture summarizes the situation.



Answers: Completed Hoare triples

These are the strongest postconditions / weakest preconditions from before.

$\{$ true $\}$	$x := 5$	$\{x = 5\}$
$\{x = y\}$	$x := x + 3$	$\{x = y + 3\}$
$\{x * 2 > -1\}$	$x := x * 2 + 3$	$\{x > 1\}$
$\{x = a\}$	if $x < 0$ then $x := -x$	$\{a \leq 0 \Rightarrow x = -a \ \wedge \ a > 0 \Rightarrow x = a\}$
$\{$ false $\}$	$x := 3$	$\{$ false $\}$
$\{x > 0\}$	while $x! = 0$ do $x := x - 1$	$\{x = 0\}$
$\{x > 0\}$	while $x! = 0$ do $x := x - 1$	$\{$ false $\}$