

Slide 1

### Administrivia

- Reminder: Homework 3 due Monday.

Slide 2

### Proving Program Correctness

- Once you've written a program, want to have some confidence that "it works".
- What do you mean "it works"? Informally? Formally, "meets its specification" (more later).
- How do you show it works? As a grad-school colleague wrote:  
To reduce the number of errors in a program, or to increase one's confidence in a program, one can *test* the program on a given test suite. If the program is observed to behave correctly for these test cases, the program is shipped to the customer. One then hopes there will be other cases that customers try for which the program also behaves correctly.
- Is there another way to "increase your confidence" in the program? "Formal methods" ...

### Proving Program Correctness, Continued

Slide 3

- Idea of formal methods is to give a mathematical proof that a program does what it's supposed to do.
- For non-trivial programs, this is usually a lot of work, though if the program is "important" enough, might be worthwhile.
- We will do mostly trivial examples — mostly because they're all we can do in the time we have. Keep in mind, though:
  - How to make this practical, and/or how to have it done by a smart program, are subjects of ongoing research.
  - In my opinion/experience, applying these ideas informally helps you "reason about programs" — which is how careful programmers work, consciously or not.

### Program Specifications

Slide 4

- Before we can prove that a program "works", we have to define what that means — "specification".
- For many programs (the ones we'll talk about here), we care that the program produces the right output for all allowed inputs. So we can write a specification in terms of "precondition" and "postcondition". E.g., for a function `sqrt` that takes a `double x` as input and returns a `double`, we could have:
  - Precondition:  $x \geq 0$ .
  - Postcondition: For return value  $y$ ,  $y \geq 0$  and  $y^2 = x$ .
- This is trivial? Consider the following proposed specification for a sorting function with two inputs  $A$  (array of integers) and  $n$  (size of  $A$ ).
  - Precondition:  $A$  is of size  $n$ ,  $n \geq 0$ .
  - Postcondition:  $(\forall i)((0 < i < n) \rightarrow A[i - 1] \leq A[i])$

## Program Specifications and Correctness

Slide 5

- Once we have a precondition and postcondition, “the program is correct” means “if we start in a state where the precondition is true, we end in a state where the postcondition is true.”
- We’ll define rules for establishing correctness for assignment, if/then/else, sequential composition, and “while” loops. That, it turns out, is enough.

## Specifications — Formal View

Slide 6

- If we have
  - $X$  — set of input variables for program  $P$
  - $P(X)$  — set of output variables for  $P$
  - $Q(X)$  — precondition
  - $R(X, P(X))$  — postcondition

then we define “ $P$  is correct” to be

$$(\forall X)(Q(X) \rightarrow R(X, P(X)))$$

- Traditionally write this using a “Hoare triple” (C. A. R. Hoare, 1968 CACM article)

$$\{ Q \} P \{ R \}$$

with implicit quantification over all values of inputs.

## How to Prove that Program Meets its Specification?

Slide 7

- First observe that we can build up all programs from a few basics:
  - Assignment.
  - Conditional (if/then/else).
  - Sequential composition.
  - Loops (while).
- So we just (!) have to give rules for these basics, and then in principle . . .

## Sequential Composition

Slide 8

- “Sequential composition”? Fancy name for “first do this, then do that.”
- Rule is: For two programs  $P_1$  and  $P_2$   
If we have  $\{ Q \} P_1 \{ R_1 \}$   
and  $\{ R_1 \} P_2 \{ R \}$   
then we can derive  $\{ Q \} P_1; P_2 \{ R \}$
- This seems plausible, no? and we could prove it with predicate logic.

### Assignment

Slide 9

- Oddly enough, this one is tricky.
- Rule is this:  
We can derive  $\{ R_1 \} x := e \{ R_2 \}$   
where  $R_1$  is  $R_2$  with all occurrences of  $x$  replaced by  $e$ .
- This makes sense, no? If something is true about  $e$ , and then we assign  $e$  to  $x$ , then the something is true about  $x$ .

### Strengthening Preconditions, Weakening Postconditions

Slide 10

- Two more rules:  
If we have  $\{ Q \} P \{ R \}$   
then for "stronger" precondition  $Q_1$  (i.e.,  $Q_1 \rightarrow Q$ )  
we can derive  $\{ Q_1 \} P \{ R \}$   
and for "weaker" postcondition  $R_1$  (i.e.,  $R \rightarrow R_1$ )  
we can derive  $\{ Q \} P \{ R_1 \}$
- This also should make sense, and we could prove it. Also, it can be helpful in applying the rule for sequential composition when the postcondition / precondition pairs don't quite match up.

## Conditionals

- Putting off loops for now, we need one more rule, for if/then/else.
- Rule is: If we have program  $S$  of the form

**if  $B$  then**

$P_1$

**else**

$P_2$

**end if**

and we have  $\{ (Q \wedge B) \} P_1 \{ R \}$

and  $\{ (Q \wedge B') \} P_2 \{ R \}$

then we can derive  $\{ Q \} S \{ R \}$

- Again, this should make sense, and we could prove it.

Slide 11

## Minute Essay

- Did you recognize the last problem on the quiz? (It should have looked familiar from the homework.)
- If we want to have  $\{ R \} x := x * 2 \{ x < 16 \}$ , what should  $R$  be?

Slide 12