Slide 1

Administrivia

- Quiz 6 next Tuesday. Homework on chapter 3 coming soon (really).

Slide 2

Proving Program Correctness — Overview

- 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" . . .

Slide 3

Proving Program Correctness, Continued

- 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.

Slide 4

Program Specifications

- 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 $y^2 = x$. (Or, more precisely, $y^2 \approx x$, and we would have to define what we want $\approx$ (approximately equal) to mean.)

Program Specifications, Continued

Slide 5

- 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: $(\forall i)(((0 < i < n) \;\rightarrow\; A[i-1] \leq A[i])$
- Okay? (No.)

Program Specifications and Correctness

Slide 6

- 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.

Slide 7

Specifications — Formal View

- 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.

Slide 8

How to Prove that Program Meets its Specification?

- 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 . . .

Slide 9

Sequential Composition

- "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.

Slide 10

Assignment

- 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.

Slide 11

Strengthening Preconditions, Weakening Postconditions

- Two more rules:

  If we have $\{ \ Q \ \} \quad P \quad \{ \ R \ \}$

  then for "stronger" precondition Q_1 (i.e., $Q_1 \ \rightarrow \ Q$)

  we can derive $\{ \ Q_1 \ \} \quad P \quad \{ \ R \ \}$

  and for "weaker" postcondition R_1 $(i.e., R \ \rightarrow \ R_1)$

  we can derive $\{ \ Q \ \} \quad P \quad \{ \ R_1 \ \}$

- 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.

Slide 12

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**

  $\quad P_1$

  else

  $\quad P_2$

  end if

  and we have $\{ \ (Q \ \wedge \ B) \ \} \quad P_1 \quad \{ \ R \ \}$ and $\{ \ (Q \ \wedge \ B') \ \} \quad P_2 \quad \{ \ R \ \}$
  then we can derive $\{ \ Q \ \} \quad S \quad \{ \ R \ \}$

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

Slide 13

Example

- Try an example — silly program (call it S) to compute minimum of two elements:

  if $x < y$ **then**
  $z := x$
  else
  $z := y$
  end if

- Show using rules for assignment and conditionals that

$$\{\ \mathit{true}\ \}\ \ S\ \ \{\ z = \min(x, y)\ \}$$

  and we probably should make the postcondition more precise/detailed, thus:

$$(z \le x)\ \wedge\ (z \le y)\ \wedge\ ((z = x)\ \vee\ (z = y))$$

Slide 14

Examples of Less Formal Use

- Rule for sequential composition leads to “programming with assertions” — at “interesting” points in the program, use to document/check what you know to be true at that point. Example: Program that first sorts an array, then repeatedly performs binary search. Could use assertion to document that array is sorted.
- Rule for conditionals can also be used informally: Code for “if” branch only has to work if condition is true; code for “else” branch only has to work if condition is false. Example: Function to compute root(s) of quadratic equation.

Slide 15

Sidebar: A Puzzle

- Suppose you have a jar containing white marbles and black marbles, plus an unlimited supply of extra black marbles, and you do the following:
  1. Select two marbles.
  2. If they're the same color, discard them both and put a black marble in the jar. If they're different colors, discard the black one and put the white one back in the jar.
  3. If there are at least two marbles in the jar, repeat.
- Does this end? If it does, what if anything can you say about the marble(s) in the jar when it ends?
- (Similar ideas behind "metric" for loop termination and "invariant" for loop correctness.)

Slide 16

Program Correctness and Loops

- We'll write loops in this form

  while B **do**

  $\quad P$

  end while

  After the loop terminates (assuming it does), what do we know about B? True or false?
- We also need the notion of a "loop invariant" — a predicate that, if true before we execute the loop body is true again after. More formally, Q is an invariant for the above loop if

$$\{\, Q \wedge B \,\}\ P\ \{\, Q \,\}$$

- Now we can state the rule for loops . . .

Slide 17

Program Correctness and Loops

- For program P_1 as follows

  while B **do**

  P

  end while

  and Q an invariant of the loop in P_1, we can say that

  $$\{\ Q\ \}\ \ P_1\ \ \{\ Q\ \wedge\ B'\ \}$$

- We could prove this using induction (on the number of trips through the loop).
- The idea is to choose Q such that the postcondition in the above triple ($Q\ \wedge\ B'$) is useful — i.e., helps establish something we want to be true after the loop.

Slide 18

Minute Essay

- If we want to have $\{\ R\ \}\ \ x := x * 2\ \ \{\ x < 16\ \}$, what should R be?
- Would you object if we only have 5 quizzes, not 6? I would still drop the low score.

Slide 19

Minute Essay Answer

- R should be $(x * 2) < 16$, i.e., $x < 8$