Slide 1

Administrivia

- Reminder: Homework 5 due today.
- Reminder: Midterm Thursday. Review sheet on the Web. Sample solutions for homeworks available in hardcopy.

Slide 2

Propositional and Predicate Logic — Things to Review

- Translating (relatively simple) English into formulas.
- Evaluating whether a formula is true given a particular assignment of values to variables (propositional logic) or a particular interpretation (predicate logic).
- Proving formulas are always true / always valid using proof rules.

Slide 3

Proof Techniques — Things to Review

- Setting up proof / “proof obligations”.
- Proving something true versus finding a counterexample.
- Proofs by induction. Recall that there are two versions, “first principle” and “second principle”. Review last problem-to-turn-in on Homework 3 for example of when the latter is useful.

Slide 4

Recursion and Recurrence Relations — Things to Review

- Recursive definitions. Note the overall idea — define larger cases in terms of smaller ones.
- Defining recurrence relations — e.g., problem about bats in homework, some of algorithm-analysis problems.
- Solving recurrence relations. Two methods discussed:

  Expand/guess/verify — can work for all, but requires proof by induction (“verify”).

  Using formulas — easier in some ways, but only works for problems that fit.

  Which one to use, how to apply. (Tricky part of the latter seems to be plugging $g(n)$ into formula.)

Slide 5

Analysis of Algorithms — Things to Review

- General idea — point is to estimate time by counting some sort of basic operations.
- Defining and solving recurrence relations for recursive algorithms. Focus on the “code” for the algorithm, as described in sample solution for Homework 5.

Slide 6

Specifications and Correctness (Review)

- If we have a program P, and a specification consisting of precondition Q and postcondition R, we write

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

  to mean that if we start in a state where Q is true and run P, we end in a state where R is true. (Also, P terminates — no “infinite” loops.)
- Example:

  $$\{\ x \geq 0\ \}\ \ y := sqrt(x);\ \ \{\ y \geq 0\ \wedge\ y^2 = x\ \}$$

Slide 7

Proofs of Program Correctness (Review)

- We looked at rules for
  - Assignment.
  - Sequential composition.
  - If/then/else.
  - Strengthening preconditions, weakening postconditions.
- We still need a rule for loops (sketched last time) . . .

Slide 8

Program Correctness and Loops

- We'll write loops in this form

  ```
  while $B$ do
      $P$
  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 9

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 10

Trivial Example

- Suppose we have

```
**while** $x > 0$ **do**
    $x := x - 1$
**end while**
```

  with x an integer variable.
- Show that after the loop $x = 0$.

Slide 11

Correctness of Loops, Continued

- The textbook isn't very explicit about this, but strictly speaking we have something else to prove — that the loop terminates!
- Can do this with a "metric" (think "measure") — integer function of program variables that decreases every time through the loop, and when it's less than or equal to zero the loop stops.
- In the silly example, we could use what? (The value of X.)

Slide 12

Program Correctness and Loops, Continued

- Things to notice about loop invariants:
  - They're not unique — could come up with many "invariants" for a given loop. (This is true about preconditions in general.)
  - The goal is to find one that's "useful" — if true at end of the loop with loop test false, helps us prove desired postcondition.
  - Sometimes helps to think in terms of "what do the variables mean?"
  - Writing down a loop invariant can help (e.g., to avoid off-by-one errors) even if you don't do a complete formal proof.
- Example — silly program to compute $z = x \times y$ by repeated addition:

```
$i := 0; z := 0;$
while $i < x$ do
        $z := z + y; i := i + 1$
end while
```

Slide 13

Program Correctness and Loops – GCD Example

- Another example — Euclid's algorithm for finding GCD (greatest common divisor, a.k.a. largest common factor) of a and b:

```
$i := a; j := b;$
while $j \neq 0$ do
    $q := i/j; r := i\%j;$
    $i := j; j := r;$
end while
```

At end, $i = gcd(a, b)$. It does?! Yes, and we can prove it, even if we don't quite understand *why*. Next slide . . .

Slide 14

Program Correctness and Loops – GCD Example, Continued

- Proposed invariant (using book's subscripting notation):

  $gcd(i_n, j_n) = gcd(a, b)$

- Prove that it's an invariant using the following lemma:

  If $a = qb + r$, then

  $gcd(a, b) = gcd(b, r)$

- Strictly speaking we also need to show that the loop stops. But this is true because j is an integer and gets smaller on every trip through the loop, and the algorithm stops when it becomes zero.

Slide 15

Proofs of Program Correctness — Recap/Evangelism

- Many examples we looked at are trivial — mostly because they're all we can do in the time we have. (Textbook's proof that Euclid's algorithm works is a notable exception.) 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". ("What do you know about the program variables at this point?" "What is this variable supposed to represent, and does the code support that?")
  - Similar ideas are very useful in reasoning about concurrent algorithms, which otherwise can be *very* tricky!

Slide 16

Minute Essay

- None — sign in.