Data Structures
&
Algorithms

Subodh Kumar

(subodh@iitd.ac.in, Bharti 422)

Dept of Computer Sc. & Engg.

Program Correctness

- Starts with a correct specification of the requirements
  - Correctness only with respect to the spec
    - If program A sends data D to program B, B will eventually get D
    - There is no deadlock or livelock
    - For all $n \geq 0$, $f(n) = k$
- Correct design
  - Algorithmically correct
- Correct implementation
  - Re-use of already correct code helps

Programming

- Understand specification
  - Formulate as formally as you can
- Devise the test plan
- Algorithmic design
  - Mind the performance
- Refine the test plan
- Implement incrementally
  - Test each time
  - Error debugging + performance debugging

Repeat as necessary

Proving Correctness

- What is correct?
  - Program? Or entire running enviroment?
  - System
- What is a proof?
  - Not: *system meets requirement*
  - But: *system cannot fail requirement*
  - Challenging

Testing is Important

Bill Gates:

"When you look at a big commercial software company like Microsoft, there's actually as much testing that goes in as development. We have as many testers as we have developers. Testers basically test all the time, and developers basically are involved in the testing process about half the time…

… We've probably changed the industry we're in. We're not in the software industry; we're in the testing industry, and writing the software is the thing that keeps us busy doing all that testing…

…The test cases are unbelievably expensive; in fact, there's more lines of code in the test harness than there is in the program itself. Often that's a ratio of about three to one."

Proof?

Dijkstra: “Program testing can be used to show the presence of bugs, but never to show their absence”

“One can never guarantee that a proof is correct, the best one can say is: ‘I have not discovered any mistakes’”

- Automated proof?
  - Impossible in the general case
- Mechanical verification is possible in some cases

col106quiz@cse.iitd.ac.in

Format: d

Select one option

- Choose the best response. Which of the following situations can cause deadlock in case of synchronized methods:

- I. The same object's or class's lock is required
- II. Multiple threads are employed
- III. The same object's or class's lock is required by two or more threads
- IV. More than one object's or class's locks are required
- V. More than one object's or class's locks are required and they are acquired in the same order by different threads

- a) All of the above
- b) II and IV
- c) III
- d) III and IV
- e) II, III, and V
- f) II

Program Correctness

To prove: If precondition is met, postcondition is met

$\boxed{\text{Precondition} \Rightarrow \text{Postcondition}}$ $\boxed{\text{Finite \#steps}}$

```
int multiply(int X, int Y)
{
    int product = 0;
    while( X > 0) {
        product += Y;
        X--;
    }
    return product;
}
```

pre: X >= 0

post: product = X*Y

For each program statement

Pre and Post Conditions

pre: x=1, y=2, z=^
    z = x+y;
post: x=1, y=2, z=3

Compose: A➜B and B➜C
⇒ A➜C

pre: x=a, y=b, t=^
    t = x; x = y; y = t;
post: x=b, y=a, t=a

Conditions and Loops

pre:
if(t) S1 else S2;
post: pre & S1-post | !pre & S2-post

Loop Invariants

Pseudo-code

- Program-like statements
  - Generally no specific syntax
- Assignments, Functions, Conditions, Loops
  - Should be clear to a COL100 graduate how to implement in her favorite language
  - Each statement should easily map to a known language or architecture

Merge-Sort Pseudocode

```
Merge-Sort(A, lo, hi): //  Merge-Sort A [lo:hi]

if (lo < hi)
    mid = ⌊(lo + hi)/2⌋
    Merge-Sort(A, lo, mid)          // Merge-Sort A from lo to mid
    Merge-Sort(A, mid+1, hi)    //  Merge-Sort A from mid+1 to hi
    Merge(A, lo, mid, hi)           // Merge A[lo:mid] and A[mid+1:hi]
```

Merge Pseudocode

```
Merge(A, lo, mid, hi):        //  Merge A with two sorted parts
                              // resp. [low:mid] and [mid+1:high]

L[0:mid-lo] = A[lo:mid]             // Copy A[low:mid] to L
R[0:hi-mid-1] = A[mid+1:hi]         // Copy A[mid+1:high]) to R
i=0                                 // Initialize i and j to 0
j=0
for k = lo to hi                    // Iterate with k in [low:high]
if L[i] < R[j]:                     // Copy min of L[i] and R[j] to A[k]
    A[k] = L[i]                     // Advance corresponding index
    i=i+1
else
    A[k] = R[j]
    j=j+1
```

Proofs of Correctness

To prove: If precondition is met, postcondition is met

Precondition ⇒ Postcondition

Finite #steps

- **Contrapositive:**
  - **! Postcondition ⇒ ! Precondition**
- **Contradiction**
  - **Asume !Postcondition**
  - **⇒ "X" ⇒ "Y" ⇒ "Z" ⇒ FALSE**
    - **Work backwards until a fallacy is implied**

Contra proof

Contra Example

$\boxed{x^2 - 6x + 5 \text{ is even} \Rightarrow x \text{ is odd}}$

x is even $\Rightarrow x^2 - 6x + 5$ is odd

$\boxed{\sqrt{2} \text{ is irrational}}$

Assume $\sqrt{2}$ is rational $\Rightarrow \frac{x}{y}$

Contradiction!

$\sqrt{2}$ is rational

Lowest form $\Rightarrow$ both cannot be even

$\Rightarrow x^2 = 2y^2$

x^2 is even $\Rightarrow x$ is $2k \Rightarrow x^2$ is $4k^2$

$4k^2 = 2y^2 \Rightarrow y^2$ is even $\Rightarrow y$ is even

Proofs of Correctness

To prove: If precondition is met, postcondition is met

Precondition ⇒ Postcondition

Finite #steps

- **Contrapositive:**
  - **! Postcondition ⇒ ! Precondition**
- **Contradiction**
  - **Asume !Postcondition**
  - **⇒ "X" ⇒ "Y" ⇒ "Z" ⇒ FALSE**
    - **Work backwards until a fallacy is implied**

Induction

Induction

- $2n = 0$, if n is a whole numbers

Base case: $n = 0$

Inductive Hypothesis: True for all $n <= k$

Show

Let n

$2n =$

Base case

Show: True for n = *some initial value*

Inductive Step [Clear Hypothesis statement]

Assume: True for all $n <= k$

Show: True for $n = k+1$

col106quiz@cse.iitd.ac.in

Format: a

```
1 interface GradeInfo_ {
2    enum LetterGrade {
3         A, Aminus, B, Bminus, C, Cminus, D, E, F, I;
4    }
5    public static int gradepoint(GradeInfo_.LetterGrade grade)
6         { return 0; } // Rest skipped here
7    public default LetterGrade grade()
8         { return I; }
9 }
10 class GradeInfo implements GradeInfo_ {
11   LetterGrade grade;
12   GradeInfo(LetterGrade grade) { this.grade = grade; }
13   public LetterGrade grade()   { return grade; }
14 }
15 class Main {
16    public static void main(String args[]) {
17     GradeInfo_ gradeinfo = new GradeInfo(LetterGrade.A);
18     System.out.println(gradeinfo.grade());
19    }
20 }
```

What's the error(s)?

a) Undefined I: line 8
b) Undefined LetterGrade.A: line 17
c) gradeinfo must be declared of type GradeInfo: line 17
d) Cannot print gradeinfo.grade: line 18
e) a) and b)
f) All of the above
g) There is no error

Loop Invariant

- Initialization
  - Invariant is true before the first iteration
- Maintenance
  - Invariant true for iteration $n \Rightarrow$

    true for iteration n+1
- Termination
  - Invariant is true for ‘all n’ at the end of the loop

Loop Invariant

- Invariant at the start of each iteration:
  - (k - lo) elements starting at A[lo] are the smallest elements of A[lo:hi], and in increasing order with A's index
- Beginning: k = lo
  - k - lo = 0
  - L and R are sorted
- Maintenance:
  - L[i] is the min of values in L, R[j] is the min in R
  - The smaller of the two is the smallest in the L $\cup$ R, for indexes >= i and >=j, respectively
  - $\Rightarrow$ A[k] gets the smallest remaining element: A[i:mid], A[j:hi]
- Termination: k = hi+1
  - hi+1-lo elements starting at A[lo:] are sorted
  - A[lo:hi] is sorted

```
L= A[left], R = A[right]
i=j=0
for k = lo to hi
   A[k] = L[i] < R[j]?
      A[k] = L[i++] :
      A[k] = R[j++]
```

$\boxed{l > m \Rightarrow A[l] >= A[m]}$