EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101M`
Also on Moodle

Reasoning (formally) about algorithms

- I/O specs: Needed for correctness proofs, performance analysis.
  E.g. for **sorting** in non-decreasing order:
  INPUT: $A[1 \ldots n]$ - an array of integers
  OUTPUT: a *permutation* B of A such that
  $B[1] \leq B[2] \leq \ldots \leq B[n]$
- CORRECTNESS: The algorithm satisfies the output specs for EVERY valid input.
- ANALYSIS: Compute the performance of the algorithm, e.g., in terms of running time

Correctness

- How can we show that the algorithm works correctly for all possible inputs of all possible sizes?
- Exhaustive testing not feasible.
- Analytical techniques are ~~useful~~ essential here.

Assertions

- An assertion is a statement about the state of the program at a specified point in its execution
- May be implemented in code, as an error-check
- Types:
  - Preconditions: Any assumptions that must be true about the code that follows
  - Postconditions: The statement of what must be true about the preceding code
  - Exit condition: The statement of what must be true to exit a loop or a method or program
  - Loop invariants: Some property that holds in each iteration of the loop, and is useful for proving correctness of the loop

Correctness Definition: Code Segment

- $\langle pre - condition \rangle \wedge \langle code \rangle \Rightarrow \langle post - condition \rangle$

- If the input meets the preconditions, then the output must meet the post-conditions.

- If the input does not meet the preconditions, then nothing is required.

Uses

- If the assertions can be checked automatically, correctness checking can be automated
- Caveat: undecidability issues
- EECS 3311 will teach you to do this in practice

Proving correctness - A Simple Example

Problem: find the maximum element of an array of integers

FIND-MAX(A)

// INPUT: $A[1..n]$ - an array of integers
// OUTPUT: an element m of A such that $m \geq A[j]$, for all $1 \leq j \leq A.length$
$max = A[1]$
for $j = 2$ **to** $A.length$
    if $max < A[j]$
        $max = A[j]$
return max

Can you think of another algorithm?

Proof by Contradiction

Proof: Suppose the algorithm is incorrect. Then for some input A, either

1. max is not an element of A or
2. A has an element $A[j]$ such that $max < A[j]$

max is initialized to and assigned to elements of A – so (1) is impossible
After the j-th iteration of the for-loop (lines 4 - 6), $max \geq A[j]$. From lines 5,6, max only increases. Therefore, upon termination, $max \geq A[j]$, which contradicts (2).

Proof by Contradiction - Remarks

- The preceding proof reasons about the whole algorithm
- It is possible to prove correctness by induction as well: this is left as an exercise for you
- What if the algorithm was very big and had many function calls, nested loops, if-then's and other standard commands?
- For example....

Pseudocode Example (page 18) Revisited

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Proof by Contradiction - Remarks

- The preceding proof reasons about the whole algorithm
- It is possible to prove correctness by induction as well: this is left as an exercise for you
- What if the algorithm was very big and had many function calls, nested loops, if-then's and other standard commands?
- Even proving that the algorithm terminates may be non-trivial!

Need a simpler, more "modular" strategy.

Correctness Proofs for Loops

Decompose the job into checking:

- Pre-condition for the loop is true

Correctness Proofs for Loops

Decompose the job into checking:

- Pre-condition for the loop is true
- Loop Invariant holds for each iteration

Correctness Proofs for Loops

Decompose the job into checking:

- Pre-condition for the loop is true
- Loop Invariant holds for each iteration
- Termination condition is met

Correctness Proofs for Loops

Decompose the job into checking:

- Pre-condition for the loop is true
- Loop Invariant holds for each iteration
- Termination condition is met
- Upon termination the post-condition holds

Correctness Proofs for Loops

Decompose the job into checking:

- Pre-condition for the loop is true
- Loop Invariant holds for each iteration
- Termination condition is met
- Upon termination the post-condition holds
- Note the similarities with induction.

Proving correctness of FindMax with LI

FIND-MAX(A)

// INPUT: $A[1..n]$ - an array of integers
// OUTPUT: an element m of A such that $m \geq A[j]$, for all $1 \leq j \leq A.length$
$max = A[1]$
for $j = 2$ **to** $A.length$
    if $max < A[j]$
        $max = A[j]$
return max

What is the precondition of the loop?

Correctness of FindMax: Steps

Show that:

- Pre-condition for the loop: *max* contains $A[1]$
- Loop Invariant for each iteration:
  At the beginning of iteration j of the for loop, *max* contains the maximum of $A[1..j-1]$
- Termination condition: $j = A.length + 1$
- Partial correctness and Termination implies post-condition: *max* is the correct maximum

Proof of the Loop Invariant - Partial Correctness

LI: At the beginning of iteration j of the for loop, *max* contains the maximum of $A[1..j-1]$

- Initialization: *max* contains $A[1]$, so $LI(1)$ is true
- Maintenance: For $j > 2$, assume $LI(j-1)$; so before iteration $j-1$, $max =$ maximum of $A[1..j-2]$

  Case 1: $A[j-1] =$ maximum of $A[1..j-1]$. In lines 5-6, *max* is set to $A[j-1]$

  Case 2: $A[j-1]$ is not the maximum of $A[1..j-1]$, so the maximum of $A[1..j-1]$ is in $A[1..j-2]$. By our assumption, *max* already has this value, and *max* is unchanged in this iteration.

Proof of the Loop Invariant - Termination

- Termination: When the loop terminates, $j = A.length + 1$ (WHY?)
- Partial correctness and Termination imply the post-condition:
  LI: At the beginning of iteration j of the for loop, *max* contains the maximum of $A[1..j-1]$
  At termination: $j = A.length + 1$
  Therefore, *max* contains the maximum of $A[1..A.length]$
  Therefore, it is the correct maximum

Loop Invariants - Summary

We must show three things about loop invariants:

- Initialization – it is true prior to the first iteration
- Maintenance – if it is true before an iteration, it remains true before the next iteration
- Termination – when loop terminates the invariant gives a useful property to show the correctness of the algorithm

$$\boxed{\text{Partial Correctness} \land \text{Termination} \Rightarrow \text{Correctness}}$$

What about more complex algorithms?

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- How to formulate loop invariants (2 loops, so 2 LI needed)
- How to prove correctness?