

# Computer Science 331

## Correctness of Algorithms

Mike Jacobson

Department of Computer Science  
University of Calgary

Lectures #2-4

## Outline

- 1 Introduction
  - What is a Proof of Correctness?
  - Applications
- 2 Proof of Correctness
  - Partial Correctness
  - Termination
  - Recursive Algorithms
- 3 Final Notes
  - Additional References

## How Do We Specify a Computational Problem?

Recall: a computational problem is specified by one (or more) pairs of *preconditions* and *postconditions*.

- *Precondition*: A condition that must be satisfied when the execution of a program begins. This generally involves the algorithm's *inputs* as well as initial values of *global variables*.
- *Postcondition*: A condition that should be satisfied when the execution of a program ends. This might be
  - A set of relationships between the values of inputs (and the values of global variables when execution started) and the values of outputs (and the values of global variables on a program's termination), or
  - A description of output generated, or exception(s) raised.

## Example: Specification of a "Search" Problem

*Precondition*  $P_1$ : Inputs include

- $n$ : a positive integer
- $A$ : an integer array of length  $n$ , with entries

$$A[0], A[1], \dots, A[n-1]$$

- $key$ : An integer found in the array (ie, such that  $A[i] = key$  for at least one integer  $i$  between 0 and  $n-1$ )

*Postcondition*  $Q_1$ :

- Output is the integer  $i$  such that  $0 \leq i < n$ ,  $A[j] \neq key$  for every integer  $j$  such that  $0 \leq j < i$ , and such that  $A[i] = key$
- Inputs (and other variables) have not changed

This describes what should happen for a "successful search."

## Example: Specification of a “Search” Problem (cont.)

## Precondition

 $n$ : a positive integer $A$ : an integer array of length  $n$ 

key: An integer not found in the array (ie: such that  $A[i]$  not equal key for every integer  $i$  between 0 and  $n - 1$ )

## Postcondition

Inputs have not changed

Output NotFoundException

## Example: Specification of a “Search” Problem

A problem can be specified by multiple precondition-postcondition pairs

$$(P_1, Q_1); (P_2, Q_2); \dots; (P_k, Q_k)$$

as long as it is not possible for more than one of the preconditions

$$P_1, P_2, \dots, P_k$$

to be satisfied at the same time.

For example, if  $P_1$ ,  $Q_1$ ,  $P_2$ , and  $Q_2$  are as in the previous slides then the pair of precondition-postcondition pairs

$$(P_1, Q_1); (P_2, Q_2)$$

could specify a “search problem” in which the input is expected to be *any* positive integer  $n$ , integer array  $A$  of length  $n$ , and integer  $key$ .

## When is an Algorithm Correct?

Suppose, first, that a problem is specified by a *single* precondition-postcondition pair  $(P, Q)$ .

An algorithm (that is supposed to solve this problem) is *correct* if it satisfies the following condition: If

- inputs satisfy the given precondition  $P$  and
- the algorithm is executed

then

- the algorithm eventually halts, and the given postcondition  $Q$  is satisfied on termination.

**Note:** This does not tell us *anything* about what happens if the algorithm is executed when  $P$  is *not* satisfied.

## When is an Algorithm Correct?

Suppose, next, that  $k \geq 2$  and that a problem is specified by a sequence of  $k$  precondition-postcondition pairs

$$(P_1, Q_1); (P_2, Q_2); \dots; (P_k, Q_k)$$

where it is impossible for more than one of the preconditions to be satisfied at the same time.

An algorithm (that is supposed to solve this problem) is *correct* if the following is true for *every* integer  $i$  between 1 and  $k$ : If

- inputs satisfy the given precondition  $P_i$  and
- the algorithm is executed

then

- the algorithm eventually halts, and the given postcondition  $Q_i$  is satisfied on termination.

## When is an Algorithm Correct?

A consequence of the previous definitions: Consider a problem specified by a sequence of  $k$  precondition-postcondition pairs

$$(P_1, Q_1); (P_2, Q_2); \dots; (P_k, Q_k).$$

Then an algorithm that is supposed to solve *this* problem is *correct* if and only if it is a *correct* solution for each of the  $k$  problems that are each specified by the single precondition-postcondition pair  $P_i$  and  $Q_i$ , for  $i$  between 1 and  $k$ .

$\Rightarrow$  It is sufficient to consider problems that are specified by a single precondition and postcondition (and we will do that, from now on).

## Why are Proofs of Correctness Useful?

### Who Generates Proofs of Correctness?

- Algorithm designers (whenever the algorithm is not obvious). Other people need to see evidence that this new algorithm really *does* solve the problem!
- Note that testing *cannot* do this (in general).

### Who Uses Proofs of Correctness?

- Anyone coding, debugging, testing, or otherwise maintaining software implementing any nontrivial algorithm need to know *why* (or *how*) the algorithm does what it is supposed in order to do their jobs well.

## One Part of a Proof of Correctness: Partial Correctness

### Partial Correctness: If

- inputs satisfy the precondition  $P$ , and
- algorithm or program  $S$  is executed,

then *either*

- $S$  halts and its inputs and outputs satisfy the postcondition  $Q$

or

- $S$  does not halt, at all.

Generally written as

$$\{P\} S \{Q\}$$

**Note:** Detailed proofs rely heavily on discrete math and logic.

## How to Prove Partial Correctness of Algorithms?

Consider algorithm  $S$ :

- Divide  $S$  into sections  $S_1; S_2; \dots; S_K$ 
  - assignment statements
  - loops
  - control statements (i.e., if-then-else)
  - (other programming constructs)
- Identify intermediate assertions  $R_i$  so that
  - $\{P\} S_1 \{R_1\}$
  - $\{R_1\} S_2 \{R_2\}$
  - ...
  - $\{R_{K-1}\} S_K \{Q\}$
- After proving each of these, we can then conclude that
  - $\{P\} S_1; S_2; \dots; S_K \{Q\}$
  - equivalently,  $\{P\} S \{Q\}$

## Example: Proof of Partial Correctness

*Problem Definition:* Finding the largest entry in an integer array.

*Precondition P:* Inputs include

- $n$ : a positive integer
- $A$ : an integer array of length  $n$ , with entries  $A[0], \dots, A[n-1]$

*Postcondition Q:*

- Output is the integer  $i$  such that  $0 \leq i < n$ ,  $A[i] \geq A[j]$  for every integer  $j$  such that  $0 \leq j < n$
- Inputs (and other variables) have not changed

## Example: Pseudocode

```

int FindMax(A, n)
  i = 0
  j = 1
  while (j < n) do
    if A[j] > A[i] then
      i = j
    end if
    j = j + 1
  end while
  return i

```

## Example: Intermediate Assertion

*Intermediate Assertion I:*

- $n$ : a positive integer
- $A$ : an integer array of length  $n$
- $i = 0$  and  $j = 1$

*Divide into Sections:*

```

{P}
i = 0, j = 1
{I}
while (j < n) do
  if A[j] > A[i] then
    i = j
  end if
  j = j + 1
end while
{Q}

```

## Example: Proof of each Section

*Prove the correctness of each section of the algorithm using the intermediate assertion I:*

- 1 First Section:  $\{P\} \ i = 0; j = 0 \ \{I\}$ 
  - correctness is trivial
- 2 Second Section:  $\{I\} \ \text{while} \dots \text{end while} \ \{Q\}$ 
  - proof is needed

$\Rightarrow$  In CPSC 331, we focus on proving correctness of simple loops and recursive programs.

## Correctness of Loops

*Problem:* Show that

$$\{P\} \text{ while } G \text{ do } S \text{ end while } \{Q\}$$

*Observation:* There is generally some condition that we expect to hold at the beginning of every execution of the body of the loop. Such a condition is called a *loop invariant*.

A condition  $R$  is a *Loop Invariant* if:

- 1 **Base Property:**  $P$  implies that  $R$  is True before the first iteration of the loop
- 2 **Inductive Property:** if  $R$  is satisfied at the beginning of the  $i$ th execution of the loop body and there is an  $i + 1$ st execution, then the loop invariant holds immediately before that execution.

## Example: Loop Invariant

**Claim:** Assertion  $R$  is a loop invariant:

- $0 \leq i < j$
- $1 \leq j < n$
- $A[i] \geq A[k]$  for  $0 \leq k < j$

Prove correct by induction on the number of iterations of the loop body.

## Proof of the Loop Invariant

**Proof.**

- 1 **Base Property:** Before the first iteration of the loop:
  - $i = 0$  and  $j = 1$ ;  $j < n$  (since the loop didn't terminate);  $A[0] \geq A[0]$
- 2 **Inductive Property:** Assume that the loop body is executed  $l > 0$  times and that  $R$  is satisfied at the beginning of the  $l$ th execution. At the end of the  $l$ th execution, since  $j$  is increased by 1:
  - $1 \leq j \leq n$
  - If the if-condition was true, then  $A[j - 1] > A[i]$ ,  ~~$A[i] = A[j - 1]$~~  (new max) and because the loop invariant held at the beginning of the loop body, we have  $0 \leq i < j$  and  $A[i] \geq A[k]$  for  $0 \leq k < j$ .
  - Otherwise,  $0 \leq i < j$  and  $A[i] \geq A[j - 1]$ , so  $A[i] \geq A[k]$  for  $0 \leq k < j$ .

If there is a  $l + 1$ st execution of the loop body, then the loop test must pass before it, so  $j < n$  and  $R$  holds. □

## Correctness of Loops: Summary

*Problem:* Prove that

$$\{P\} \text{ while } G \text{ do } S \text{ end while } \{Q\}$$

*Solution:*

- 1 Identify a loop invariant  $R$  and prove:
  - **Base Property:**  $P$  implies that  $R$  is True before the first iteration of the loop
  - **Inductive Property:** if  $R$  is satisfied at the beginning of the  $i$ th execution of the loop body and there is an  $i + 1$ st execution, then the loop invariant holds immediately before that execution.

**Note:** essentially a *proof by induction* that the loop invariant holds after zero or more executions of the loop body.

- 2 Prove the **correctness of the postcondition**:
  - if the loop terminates after zero or more iterations, the Truth of  $R$  implies that  $Q$  is satisfied

## Example: Last Step

*Problem:* Prove that if the loop terminates after zero or more iterations, the Truth of

$$R : 0 \leq i < j, \quad 1 \leq j < n, \quad A[i] \geq A[k] \text{ for } 0 \leq k < j$$

implies that

$$Q : 0 \leq i < n, \quad A[i] \geq A[j] \text{ for } 0 \leq j < n$$

is satisfied.

*Solution:*

Loop terminates, this implies  $j = n$

$R$  implies  $0 \leq i < j$ , so  $0 \leq i < n$

- $R$  implies  $A[i] \geq A[k]$  for  $0 \leq k < j$ , so  $A[i] \geq A[j]$  for  $0 \leq j < n$

## Mathematical Induction

*Problem:* For all integers  $k \geq k_0$ , prove that property  $P(k)$  is True.

*Proof by Induction:*

- 1 **Base Case:** Show that  $P(k_0)$  is True
- 2 **Inductive Step:** Show that if  $P(k)$  is True for some arbitrary integer  $k \geq k_0$  (the **induction hypothesis**), then  $P(k+1)$  is True.
  - choose an arbitrary  $k \geq k_0$
  - show that  $P(k+1)$  is True if  $P(k)$  is True

## Example: Mathematical Induction

**Claim:**  $2^{2^n} - 1$  is divisible by 3 for all integers  $n \geq 1$ .

**Proof by Induction.**

Let  $P(n)$ :  $2^{2^n} - 1$  is divisible by 3

- 1 **Base Case:**
- 2 **Inductive Step:**



## Example 1: Partial Correctness of Loops

Prove the correctness of the following algorithm.

*Precondition (P):*  $n$  is a positive integer

*Postcondition (Q):*  $n$  is unchanged and  $s = \sum_{j=1}^n j$

Sum( $n$ )

$i = 1$

$s = 1$

**while**  $i < n$  **do**

$i = i + 1$

$s = s + i$

**end while**

**Claim:**  $\text{Sum}(n)$  is Partially Correct

Proof.

$P: 1 \leq i < n$   
 $1 \leq s < \text{sum from } 1 \text{ to } i \text{ of } j$

Assume loop body executed some  
 number of times,  $k$ .

$Q: i \text{ is } n$   
 $s \text{ is sum from } 1 \text{ to } n \text{ of } j$



## Another Part: Termination

*Termination:* If

- inputs satisfy the precondition  $P$ , and
  - algorithm or program  $S$  is executed,
- then
- $S$  is guaranteed to halt!

**Note:** Partial Correctness + Termination  $\Rightarrow$  Total Correctness!

Partial Correctness and Termination are often (but not always) considered separately because ...

- Different — independent — arguments are used for each
- Sometimes one condition holds, but not the other! Then the algorithm is *not* totally correct. ... but something interesting can still be established.

## Termination of Loops

*Problem:* Show that if the precondition  $P$  is satisfied and the loop

`while G do S end while`

is executed, then the loop eventually terminates.

Suppose that a *loop invariant*  $R$  for the precondition  $P$  and the above loop has already been found. You should have done this when proving the partial correctness of this loop — also useful to prove termination.

*Proof Rule:* To establish the above termination property, prove *each* of the following.

- 1 If the loop invariant  $R$  is satisfied and the loop body  $S$  is executed then the loop body terminates.
  - 2 The loop body is only executed a finite number of times.  
 (Proof technique is based on the concept of a *Loop Variant*.)
- Termination of loop body may not be trivial. ie: calls to another method, or nested loops

## Termination of Loops, Continued

*Definition:* A *loop variant* for a loop

`while G do S end while`

is a *function*  $f_L$  from program variables to the set of *integers* that satisfies the following additional properties:

- 1 The value of  $f_L$  is decreased *by at least one* every time the loop body  $S$  is executed
- 2 If the value of  $f_L$  is less than or equal to zero then the loop guard  $G$  is False (ie., the loop terminates)

**Note:** The *initial* value of  $f_L$  is an upper bound for the number of executions of the loop body before the loop terminates.

## Termination of Loops, Continued

**Problem:** Prove that if the precondition  $P$  is satisfied and the loop

```
while G do S end while
```

is executed, then the loop eventually terminates.

**Solution:**

- ① Show that if the loop invariant is satisfied and the loop body is executed then the loop body terminates
- ② Identify a loop variant  $f_L$ :
  - $f_L$  is an integer valued function
  - The value of  $f_L$  is decreased by *at least one* every time the loop body is executed
  - If the value of  $f_L$  is less than or equal to zero then the loop guard is False

## Example 1: Termination of Loops

**Claim:**  $\text{Sum}(n)$  terminates.

**Proof.**

Loop body of sum terminates. Just assignment statements.

( $i$  and  $n$  determine termination)  
Function in terms of  $n$  and  $i$ .  
Let function be  $n - i$ .

$n - 1$  is upper bound.

## Correctness of Recursive Algorithms

Suppose method A calls itself (but does not call any other methods).

In this case, it is often possible to prove the correctness of this method using *strong mathematical induction*, proceeding by induction on the “size” of the inputs.

- **Base Case:** base cases of the recursive algorithm
- **Inductive Step:** algorithm is correct for all inputs of size “up to”  $n$ , show that it is correct for inputs of size  $n + 1$

Proof proceeds by proving correctness while assuming the induction hypothesis (i.e., every recursive call returns the correct output).

## Strong Mathematical Induction

**Problem:** For all integers  $k \geq k_0$ , prove that property  $P(k)$  is True.

**Proof by Strong form of Induction:**

- ① **Base Case:** Show that  $P(k_0)$  is True
- ② **Inductive Step:** Show that if  $P(i)$  is True for **all** integers  $k_0 \leq i \leq k$  then  $P(k + 1)$  is True.
  - choose an arbitrary  $k \geq k_0$
  - show that  $P(k + 1)$  is True if  $P(k), P(k - 1), \dots, P(k_0)$  are True



## Example: Partial Correctness of Recursive Algorithms

Prove the correctness of the following algorithm.

*Precondition:*  $i$  is a positive integer

*Postcondition:* the value returned is the  $i^{\text{th}}$  Fibonacci number,  $F_i$

```
long Fib(i)
  if i == 0 then
    return 1
  end if
  if i == 1 then
    return 1
  end if
  return Fib(i-1) + Fib(i-2)
```

## Example, Continued

**Claim:**  $\text{Fib}(i)$  is partially correct.

**Proof.**

- ① **Base Case:** The algorithm is partially correct for  $i = 0$  and  $i = 1$
- ② **Inductive Step:** Assume that  $\text{Fib}(i)$  for  $i = 0, 1, \dots, k$  ( $k \geq 1$ ) returns the  $i$ -th Fibonacci number denoted by  $F_i$ . Show that  $\text{Fib}(k + 1)$  returns the  $(k + 1)$ -th Fibonacci number,  $F_{k+1}$ .



## Applications to Software Development

A proof of correctness of an algorithm includes detailed information about the expected state of inputs and variables at every step during the computation.

This information can be included in documentation as an aid to other developers. It also facilitates effective testing and debugging.

Self-study exercises can be used to learn more about this.

## What You Need to Be Able to Do!

Understand proofs of correctness

- We will supply these throughout the course.
- Proofs are an aid to describing *why* and *how* an algorithm works.

Provide your own proofs for simple iterative and recursive algorithms

- Iterative: loop invariants (partial correctness) and loop variants (termination)
- Recursive: induction

## Can This All Be Automated?

The following questions might come to mind.

- Q: Is it possible to write a program that decides whether a given program is correct, providing a proof of correctness of the given program, if it is?
- A: No! the simpler problem of determining whether a given program *halts* on a given input is “undecidable:” It has been *proved* that no computer program can solve this problem!
- Q: Can a computer program be used to *check* a proof of correctness?
- A: See our courses in “Artificial Intelligence” for information about this!

## References

Introduction to Algorithms, Section 2.1

*Recommended References:*

- Susanna S. Epp  
*Discrete Mathematics with Applications*, Third Edition  
See Section 4.5
- Michael Soltys  
*An Introduction to the Analysis of Algorithms*  
Chapter 1 contains an introduction to proofs of correctness and is freely available online!