Computer Science 331

Proofs of Correctness of Algorithms

Mike Jacobson

Department of Computer Science
University of Calgary

Lectures #3–4

Outline

What is a Proof of Correctness?

Concerns both a requirements specification, including a pre-condition P and post-condition Q, and an algorithm or program S

Specifically, this is a proof that if

- inputs satisfy the pre-condition P, and
- algorithm or program S is executed,

then

- S eventually halts, and its inputs and outputs satisfy the post-condition Q

Generally expected to be a *formal* mathematical proof establishing "correctness" of pseudocode (or code) as defined above.

Why Prove Correctness?

Testing is not always sufficient:

- testing cannot prove correctness
- testing can detect errors, but is not *guaranteed* to find them
- when computer time is expensive, proving correctness can be cheaper than testing
- in safety-critical situations, proving correctness may be required

On the other hand: testing is often feasible when a proof of correctness is not, and "errors in proofs" can be missed, too!

Used to prove correctness of *algorithms*.

One Part of a Proof: Partial Correctness

If

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

then *either*

- S halts and outputs satisfy the postcondition Q

or

- S does not halt

Generally written as $\{P\}\ S\ \{Q\}$

Another Part of a Proof: Termination

If

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

then

- S is guaranteed to halt (terminate)

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* correct, but something interesting can still be established (eg. testing non-existence conjecture).

Strategy

Proving Correctness

- If partial correctness *and* termination of S has been proved then correctness of S has been proved too.

Proving Partial Correctness and Termination

There are several different *kinds of programs:*

- simple statements, including assignment statements
- sequences of subprograms
- conditional statements
- loops

There are different proof strategies for each.

Assignment Statement Initializes a Variable

Claim:

$$\{x = 1\}\ \ y := x + 1\ \ \{x = 1 \text{ and } y = 2\}$$

General Rule:

- $\{x = 1\}$ is the precondition
- $\{x = 1 \text{ and } y = 2\}$ is the postcondition
- simple argument for proof — statement sets y to $x + 1$ and leaves x unchanged

Termination: obvious (remember, we are *not* proving correctness of the runtime environment)

Assignment Changes a Variable's Value

Claim:

$$\{x_{\text{old}} = 1\} \;\; x := x + 1 \;\; \{x_{\text{new}} = 2\}$$

General Rule:

- programs are not static — variables change value as the program executes
- need subscripts to distinguish between value of x before and after the statement

Termination: obvious

Program is a Sequence of Subprograms

Claim:

$$\{x = 1\} \;\; y := x + 1;\; z := y + 1 \;\; \{z = 3\}$$

Proof.

Insert the *assertion* $\{x = 1 \wedge y = 2\}$ between the two statements

- Prove $\{x = 1\} \;\; y := x + 1 \;\; \{x = 1 \wedge y = 2\}$ (follows from a previous claim)
- Prove $\{x = 1 \wedge y = 2\} \;\; z := y + 1 \;\; \{x = 1 \wedge y = 2 \wedge z = 3\}$ (also follows from a previous claim)

Correctness of the claim follows. □

Termination: Obvious (two simple statements

Strategy for Proving Correctness of Sequences

Assume program S is a sequence of statements or blocks $S_1, \ldots, S_n$

- to prove $\{P\}\; S\; \{Q\}$, create assertions (logical statements involving the program's variables) $A_1, \ldots, A_{n-1}$ that should hold between each pair of consecutive statements
- prove $\{P\}\; S_1\; \{A_1\}, \{A_1\}\; S_2\; \{A_2\}, \ldots, \{A_{n-1}\}\; S_n\; \{Q\}$
  - this proves partial correctness
- prove that each of $S_1, \ldots, S_n$ terminates individually
  - this proves termination

If partial correctness and termination are proved, then S is correct.

Program is a Conditional Statement

Claim:

{ x is a nonnegative integer }

if even(x) **then**
  $y := x + 2$
else
  $y := x + 1$
end if

{ y is an even integer and either $y = x + 1$ or $y = x + 2$ }

. . . assuming even() decides whether its input is even

What Do We Know?

If the program's test is satisfied:

- x is even
- $y = x + 2$ and y is even (because x is even)

If the program's test is *not* satisfied:

- x is odd
- $y = x + 1$ and y is even (because x is odd)

Main idea: prove each case separately

Proving Correctness of Conditional Statements

To prove

$$\{P\} \quad \textbf{if } T \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if} \quad \{Q\}$$

Show that

- $\{\ P \wedge T\ \}\ S_1\ \{Q\ \}$
- $\{\ P \wedge \sim T\ \}\ S_2\ \{Q\ \}$

Termination:

- holds if T, S_1, and S_2 all terminate

Program is a Loop

Claim:

$\{$ n is a nonnegative integer $\}$

$i := 0$; $s := 0$;
while $i < n$ **do**
  $\{$ $i = j$, $s =$ sum of the first j positive integers, and $0 \leq i \leq n$ $\}$
  $i := i + 1$; $s := s + i$
end while

$\{$ s is the sum of the first n positive integers $\}$

Add special assertion $I(j)$ (should be true after j iterations):

$$i = j, s = \text{sum of the first } j \text{ positive integers, and } 0 \leq i \leq n$$

Loop Invariants

$I(j)$ is an example of a **loop invariant:**

> *an assertion $I(j)$ that is true immediately after the loop body has been executed j times, if the loop body is actually executed j times, for all $j \geq 0$*

Necessary Properties:

- The pre-condition implies that $I(0)$ holds
- $I(j)$ implies the post-condition whenever the loop body is executed exactly j times

Useful Properties

To establish the above "Necessary Properties," show that:

1. $I(0)$ is satisfied before the first execution of the loop!
2. If $I(j)$ is satisfied after the jth execution and there is a $j+1$st execution, then $I(j+1)$ is satisfied after the $j+1$st execution, for each integer $j \geq 0$.
3. If there is a jth execution but not a $j+1$st execution then $I(j)$ implies the postcondition (again, for each integer $j \geq 0$).

Exercise:

- Show that all three properties hold for the example. Note that $I(0)$ should hold *just before* the first execution of the loop body.
- What *proof technique* (from MATH 271) could be used with these properties to prove the claim?

Proving Termination of a Loop

Loop Variant: An *integer-valued* function f of the program's variables such that:

- the value of f is *decreased by at least one* every time the loop body is executed,
- loop terminates (immediately!) if f's value is zero or negative after any execution of the loop body.

Loop Variant for the Example Program: $f(n, i) = n - i$

- number of executions of loop body is at most $f(n, 0) = n - 0 = n$ (f evaluated with the loop index set to 0)

Note: "Loop variants" have different names in different references (when they are mentioned, at all).

Other Kinds of Programs

Useful "proof rules" like the above can also be provided for

- other kinds of tests and loops
- calls to nonrecursive methods
- a recursive use of a method

In each case the "proof rule(s)" corresponds to what the statement is supposed to do.

Exercise: Try to think of appropriate proof rules for each of the above kinds of programs.

References

Discrete Mathematics textbooks sometimes include "proofs of correctness" of algorithms as an application of *mathematical induction:*

Recommended References:

- Susanna S. Epp
  Discrete Mathematics with Applications, Third Edition
  See Section 4.5
- Kenneth H. Rosen
  Discrete Mathematics and Its Applications, Sixth Edition
  See Section 4.5

Note: Epp's text *does not* define a loop invariant in the same way as these notes do, and partial correctness and termination are not considered separately there.

For Further Reading

Textbook, Section 2.7

Each of the following — rather demanding — references is available on reserve in the library:

- Edsger W. Dijkstra
  A Discipline of Programming
- David Gries
  The Science of Programming

These may be challenging, especially for students who have not already completed PHIL 279 (or taken another course in mathematical logic)!