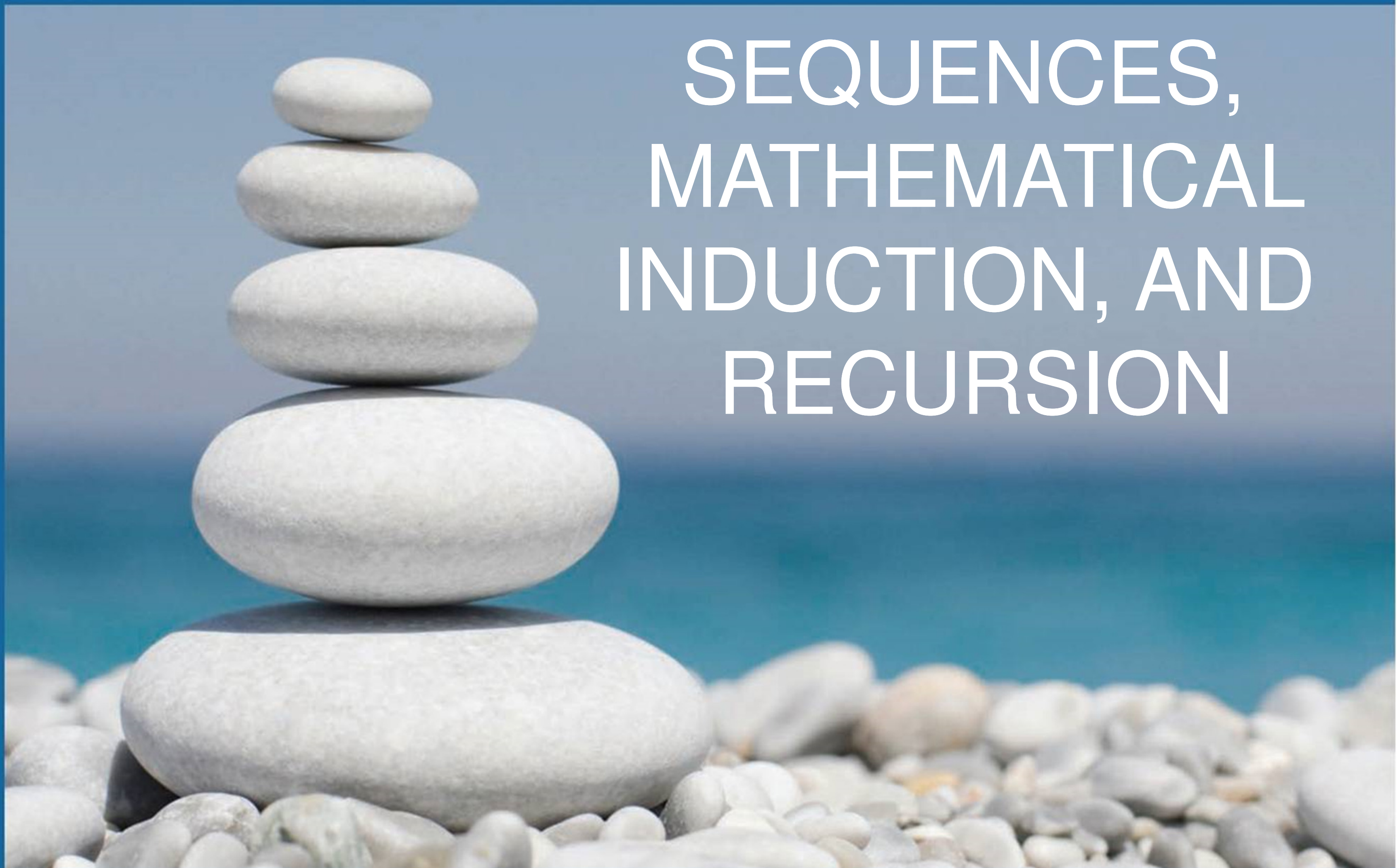
CHAPTER 5



SEQUENCES, MATHEMATICAL INDUCTION, AND RECURSION

SECTION 5.5

Application: Correctness of Algorithms

Application: Correctness of Algorithms

Each program is designed to do a specific task—calculate the mean or median of a set of numbers, compute the size of the paychecks for a company payroll, rearrange names in alphabetical order, and so forth.

We will say that a program is correct if it produces the output specified in its accompanying documentation for each set of input data of the type specified in the documentation.

Most computer programmers write their programs using a combination of logical analysis and trial and error.

Application: Correctness of Algorithms

In order to get a program to run at all, the programmer must first fix all syntax errors (such as writing **ik** instead of **if,** failing to declare a variable, or using a restricted keyword for a variable name).

When the syntax errors have been removed, however, the program may still contain logical errors that prevent it from producing correct output.

Frequently, programs are tested using sets of sample data for which the correct output is known in advance.

Application: Correctness of Algorithms

And often the sample data are deliberately chosen to test the correctness of the program under extreme circumstances.

But for most programs the number of possible sets of input data is either infinite or unmanageably large, and so no amount of program testing can give perfect confidence that the program will be correct for all possible sets of legal input data.

Assertions

Assertions

Consider an algorithm that is designed to produce a certain final state from a certain initial state. Both the initial and final states can be expressed as predicates involving the input and output variables.

Often the predicate describing the initial state is called the **pre-condition for the algorithm**, and the predicate describing the final state is called the **post-condition for the algorithm**.

Example 1 – *Algorithm Pre-Conditions and Post-Conditions*

Here are pre- and post-conditions for some typical algorithms.

a. Algorithm to compute a product of nonnegative integers

Pre-condition: The input variables m and n are nonnegative integers.

Post-condition: The output variable p equals mn.

Example 1 – *Algorithm Pre-Conditions and Post-Conditions*

cont'd

b. Algorithm to find quotient and remainder of the division of one positive integer by another

Pre-condition: The input variables a and b are positive integers.

Post-condition: The output variables q and r are integers such that $a = bq + r$ *and* $0 \leq r < b$.

Example 1 – *Algorithm Pre-Conditions and Post-Conditions*

cont'd

c. Algorithm to sort a one-dimensional array of real numbers

Pre-condition: The input variable $A[1], A[2], \ldots, A[n]$ is a one-dimensional array of real numbers.

Post-condition: The output variable $B[1], B[2], \ldots, B[n]$ is a one-dimensional array of real numbers with same elements as $A[1], A[2], \ldots, A[n]$ but with the property that $B[i] \leq B[j]$ whenever $i \leq j$.

Loop Invariants

Loop Invariants

The method of loop invariants is used to prove correctness of a loop with respect to certain pre- and post-conditions. It is based on the principle of mathematical induction.

Suppose that an algorithm contains a **while** loop and that entry to this loop is restricted by a condition G, called the **guard**.

Suppose also that assertions describing the current states of algorithm variables have been placed immediately preceding and immediately following the loop.

Loop Invariants

The assertion just preceding the loop is called the **pre-condition for the loop** and the one just following is called the **post-condition for the loop**. The annotated loop has the following appearance:

[*Pre-condition for the loop*]

while (G)

  [*Statements in the body of the loop. None contain branching statements that lead outside the loop.*]

end while

[*Post-condition for the loop*]

Loop Invariants

- **Definition**

A loop is defined as **correct with respect to its pre- and post-conditions** if, and only if, whenever the algorithm variables satisfy the pre-condition for the loop and the loop terminates after a finite number of steps, the algorithm variables satisfy the post-condition for the loop.

Establishing the correctness of a loop uses the concept of loop invariant. A **loop invariant** is a predicate with domain a set of integers, which satisfies the condition: For each iteration of the loop, if the predicate is true before the iteration, then it is true after the iteration.

Loop Invariants

Furthermore, if the predicate satisfies the following two additional conditions, the loop will be correct with respect to it pre- and post-conditions:

1. It is true before the first iteration of the loop.

2. If the loop terminates after a finite number of iterations, the truth of the loop invariant ensures the truth of the post-condition of the loop.

Loop Invariants

The following theorem, called the *loop invariant theorem*, formalizes these ideas.

Theorem 5.5.1 Loop Invariant Theorem

Let a **while** loop with guard G be given, together with pre- and post-conditions that are predicates in the algorithm variables. Also let a predicate $I(n)$, called the **loop invariant,** be given. If the following four properties are true, then the loop is correct with respect to its pre- and post-conditions.

I. Basis Property: The pre-condition for the loop implies that $I(0)$ is true before the first iteration of the loop.

II. Inductive Property: For all integers $k \geq 0$, if the guard G and the loop invariant $I(k)$ are both true before an iteration of the loop, then $I(k+1)$ is true after iteration of the loop.

III. Eventual Falsity of Guard: After a finite number of iterations of the loop, the guard G becomes false.

IV. Correctness of the Post-Condition: If N is the least number of iterations after which G is false and $I(N)$ is true, then the values of the algorithm variables will be as specified in the post-condition of the loop.

Example 2 – *Correctness of a Loop to Compute a Product*

The following loop is designed to compute the product *mx* for a nonnegative integer *m* and a real number *x*, without using a built-in multiplication operation. Prior to the loop, variables *i* and *product* have been introduced and given initial values *i* = 0 and *product* = 0.

[Pre-condition: m is a nonnegative integer,
x is a real number, $i = 0$*, and* $product = 0$*.]*

while $(i \neq m)$

1. $product := product + x$
2. $i := i + 1$

end while

[Post-condition: $product = mx$*]*

Example 2 – *Correctness of a Loop to Compute a Product* cont'd

Let the loop invariant be

$$I(n): \ i = n \quad \text{and} \quad product = nx$$

The guard condition *G* of the **while** loop is

$$G: \ i \neq m$$

Use the loop invariant theorem to prove that the **while** loop is correct with respect to the given pre- and post-conditions.

Example 2 – *Solution*

I. Basis Property: *[$I(0)$ is true before the first iteration of the loop.]*
$I(0)$ is "$i = 0$ and $product = 0 \cdot x$", which is true before the first iteration of the loop because $0 \cdot x = 0$.

II. Inductive Property: *[If $G \wedge I(k)$ is true before a loop iteration (where $k \geq 0$), then $I(k + 1)$ is true after the loop iteration.]*

Example 2 – *Solution*

cont'd

Suppose k is a nonnegative integer such that $G \wedge I(k)$ is true before an iteration of the loop. Then as execution reaches the top of the loop, $i \neq m$, $product = kx$, and $i = k$.

Since $i \neq m$, the guard is passed and statement 1 is executed. Before execution of statement 1,

$$\text{product}_{\text{old}} = kx.$$

Thus execution of statement 1 has the following effect:

$$\text{product}_{\text{new}} = \text{product}_{\text{old}} + x = kx + x = (k+1)x.$$

Example 2 – *Solution*

cont'd

Similarly, before statement 2 is executed,

$$i_{\text{old}} = k,$$

so after execution of statement 2,

$$i_{\text{new}} = i_{\text{old}} + 1 = k + 1.$$

Hence after the loop iteration, the statement $I(k + 1)$, namely, ($i = k + 1$ and $product = (k + 1)x$), is true. This is what we needed to show.

Example 2 – *Solution*

cont'd

III. Eventual Falsity of Guard: *[After a finite number of iterations of the loop, G becomes false.]*

The guard G is the condition $i \neq m$, and m is a nonnegative integer.

By I and II, it is known that

for all integers $n \geq 0$, if the loop is iterated n times, then $i = n$ and product $= nx$.

So after m iterations of the loop, $i = m$.

Thus G becomes false after m iterations of the loop.

Example 2 – *Solution*

cont'd

IV. Correctness of the Post-Condition: *[If N is the least number of iterations after which G is false and I*(N) *is true*, *then the value of the algorithm variables will be as specified in the post-condition of the loop*.*]*

According to the post-condition, the value of *product* after execution of the loop should be mx.

But if G becomes false after N iterations, $i = m$. And if $I(N)$ is true, $i = N$ and *product* $= Nx$.

Since both conditions (G false and $I(N)$ true) are satisfied, $m = i = N$ and *product* $= mx$ as required.

Correctness of the Division Algorithm

Correctness of the Division Algorithm

The division algorithm is supposed to take a nonnegative integer a and a positive integer d and compute nonnegative integers q and r such that $a = dq + r$ and $0 \leq r < d$.

Initially, the variables r and q are introduced and given the values $r = a$ and $q = 0$.

Correctness of the Division Algorithm

The crucial loop, annotated with pre- and post-conditions, is the following:

[Pre-condition: a is a nonnegative integer and d is a positive integer, $r = a$, and $q = 0$.]

while $(r \geq d)$

1. $r := r - d$
2. $q := q + 1$

end while

[Post-condition: q and r are nonnegative integers with the property that $a = qd + r$ and $0 \leq r < d$.]

Correctness of the Division Algorithm

Proof:

To prove the correctness of the loop, let the loop invariant be

$$I(n): r = a - nd \geq 0 \quad \text{and} \quad n = q.$$

The guard of the **while** loop is

$$G: r \geq d$$

Correctness of the Division Algorithm

I. Basis property: *[$I(0)$ is true before the first iteration of the loop.]*

$I(0)$ is "$r = a - 0 \cdot d \geq 0$ and $q = 0$." But by the pre-condition, $r = a$, $a \geq 0$, and $q = 0$. So since $a = a - 0 \cdot d$, then $r = a - 0 \cdot d$ and $I(0)$ is true before the first iteration of the loop.

II. Inductive Property: *[If $G \wedge I(k)$ is true before an iteration of the loop (where $k \geq 0$), then $I(k + 1)$ is true after iteration of the loop.]*

Correctness of the Division Algorithm

Suppose k is a nonnegative integer such that $G \wedge I(k)$ is true before an iteration of the loop. Since G is true, $r \geq d$ and the loop is entered. Also since $I(k)$ is true, $r = a - kd \geq 0$ *and* $k = q$. Hence, before execution of statements 1 and 2,

$$r_{\text{old}} \geq d \quad \text{and} \quad r_{\text{old}} = a - kd \quad \text{and} \quad q_{\text{old}} = k.$$

When statements 1 and 2 are executed, then,

$$r_{\text{new}} = r_{\text{old}} - d = (a - kd) - d = a - (k+1)d \qquad 5.5.2$$

and

$$q_{\text{new}} = q_{\text{old}} + 1 = k + 1 \qquad 5.5.3$$

Correctness of the Division Algorithm

In addition, since $r_{old} \geq d$ before execution of statements 1 and 2, after execution of these statements,

$$r_{new} = r_{old} - d \geq d - d \geq 0. \qquad 5.5.4$$

Putting equations (5.5.2), (5.5.3), and (5.5.4) together shows that after iteration of the loop,

$$r_{new} \geq 0 \quad \text{and} \quad r_{new} = a - (k+1)d \quad \text{and} \quad q_{new} = k + 1.$$

Hence $I(k + 1)$ is true.

Correctness of the Division Algorithm

III. Eventual Falsity of the Guard: *[After a finite number of iterations of the loop, G becomes false.]*

The guard *G is the condition* $r \geq d$. Each iteration of the loop reduces the value of *r* by *d* and yet leaves *r* nonnegative.

Thus the values of *r* form a decreasing sequence of nonnegative integers, and so (by the well-ordering principle) there must be a smallest such *r*, say r_{min}.

Correctness of the Division Algorithm

Then $r_{min} < d$. *[If* r_{min} *were greater than* d, *the loop would iterate another time, and a new value of r equal to* $r_{min} – d$ *would be obtained*. *But this new value would be smaller than* r_{min} *which would contradict the fact that* r_{min} *is the smallest remainder obtained by repeated iteration of the loop*.*]*

Hence as soon as the value $r = r_{min}$ is computed, the value of r becomes less than d, and so the guard G is false.

Correctness of the Division Algorithm

IV. Correctness of the Post-Condition: *[If N is the least number of iterations after which G is false and I(N) is true, then the values of the algorithm variables will be as specified in the post-condition of the loop.]*

Suppose that for some nonnegative integer N, G is false and $I(N)$ is true. Then $r < d$, $r = a - Nd$, $r \geq 0$, and $q = N$. Since $q = N$, by substitution,

$$r = a - qd.$$

Or, adding qd to both sides,

$$a = qd + r.$$

Correctness of the Division Algorithm

Combining the two inequalities involving *r* gives

$$0 \leq r < d.$$

But these are the values of *q* and *r* specified in the post-condition, so the proof is complete.

Correctness of the Euclidean Theorem

Correctness of the Euclidean Theorem

The Euclidean algorithm is supposed to take integers A and B with $A > B \geq 0$ and compute their greatest common divisor. Just before the crucial loop, variables a, b, and r have been introduced with $a = A$, $b = B$, and $r = B$.

Correctness of the Euclidean Theorem

The crucial loop, annotated with pre- and post-conditions, is the following:

[Pre-condition: A and B are integers
with $A > B \geq 0, a = A, b = B, r = B$.*]*

while $(b \neq 0)$

1. $r := a \bmod b$
2. $a := b$
3. $b := r$

end while

[Post-condition: $a = \gcd(A, B)$*]*

Correctness of the Euclidean Theorem

Proof:

To prove the correctness of the loop, let the invariant be

$$I(n)\colon\ \gcd(a, b) = \gcd(A, B) \quad \text{and} \quad 0 \le b < a.$$

The guard of the **while** loop is

$$G\colon\ b \neq 0.$$

Correctness of the Euclidean Theorem

I. Basis Property: *[I(0) is true before the first iteration of the loop.]*

$I(0)$ is

$$\gcd(A, B) = \gcd(a, b) \quad \text{and} \quad 0 \le b < a.$$

According to the pre-condition,

$$a = A, \quad b = B, \quad r = B, \quad \text{and} \quad 0 \le B < A.$$

Hence $\gcd(A, B) = \gcd(a, b)$. Since $0 \le B < A$, $b = B$, and $a = A$ then $0 \le b < a$. Hence $I(0)$ is true.

Correctness of the Euclidean Theorem

II. Inductive Property: *[If $G \wedge I(k)$ is true before an iteration of the loop (where $k \geq 0$), then $I(k + 1)$ is true after iteration of the loop.]*

Suppose k is a nonnegative integer such that $G \wedge I(k)$ is true before an iteration of the loop. *[We must show that $I(k + 1)$ is true after iteration of the loop.]* Since G is true, $b_{old} \neq 0$ and the loop is entered. And since $I(k)$ is true, immediately before statement 1 is executed,

$$\gcd(a_{old}, b_{old}) = \gcd(A, B) \quad \text{and} \quad 0 \leq b_{old} < a_{old}. \qquad 5.5.5$$

Correctness of the Euclidean Theorem

After execution of statement 1,

$$r_{\text{new}} = a_{\text{old}} \; mod \; b_{\text{old}}.$$

Thus, by the quotient-remainder theorem,

$$a_{\text{old}} = b_{\text{old}} \cdot q + r_{\text{new}} \quad \text{for some integer } q$$

and r_{new} has the property that

$$0 \le r_{\text{new}} < b_{\text{old}}. \qquad 5.5.6$$

By Lemma 4.8.2,

$$\gcd(a_{\text{old}}, b_{\text{old}}) = \gcd(b_{\text{old}}, r_{\text{new}}).$$

So by the equation of (5.5.5),

$$\gcd(b_{\text{old}}, r_{\text{new}}) = \gcd(A, B). \qquad 5.5.7$$

Correctness of the Euclidean Theorem

When statements 2 and 3 are executed,

$$a_{new} = b_{old} \quad \text{and} \quad b_{new} = r_{new}. \qquad 5.5.8$$

Substituting equations (5.5.8) into equation (5.5.7) yields

$$\gcd(a_{new}, b_{new}) = \gcd(A, B). \qquad 5.5.9$$

And substituting the values from the equations in (5.5.8) into inequality (5.5.6) gives

$$0 \le b_{new} < a_{new}. \qquad 5.5.10$$

Hence after the iteration of the loop, by equation (5.5.9) and inequality (5.5.10),

$$\gcd(a, b) = \gcd(A, B) \quad \text{and} \quad 0 \le b < a,$$

which is $I(k + 1)$. *[This is what we needed to show.]*

Correctness of the Euclidean Theorem

III. Eventual Falsity of the Guard: *[After a finite number of iterations of the loop, G becomes false.]*

Each value of *b* obtained by repeated iteration of the loop is nonnegative and less than the previous value of *b*.

Correctness of the Euclidean Theorem

Thus, by the well-ordering principle, there is a least value b_{min}. The fact is that $b_{min} = 0$.

[If b_{min} *is not* 0, *then the guard is true*, *and so the loop is iterated another time*. *In this iteration a value of r is calculated that is less than the previous value of b*, b_{min}. *Then the value of b is changed to r*, *which is less than* b_{min}. *This contradicts the fact that* b_{min} *is the least value of b obtained by repeated iteration of the loop*. *Hence* $b_{min} = 0$.*]*

Since $b_{min} = 0$, the guard is false immediately following the loop iteration in which b_{min} is calculated.

Correctness of the Euclidean Theorem

IV. Correctness of the Post-Condition: *[If N is the least number of iterations after which G is false and I(N) is true, then the values of the algorithm variables will be as specified in the post-condition.]*

Suppose that for some nonnegative integer N, G is false and $I(N)$ is true. *[We must show the truth of the post- condition: $a = gcd(A, B)$.]* Since G is false, $b = 0$, and since $I(N)$ is true,

$$\gcd(a, b) = \gcd(A, B). \qquad 5.5.11$$

Correctness of the Euclidean Theorem

Substituting $b = 0$ into equation (5.5.11) gives

$$\gcd(a, 0) = \gcd(A, B).$$

But by Lemma 4.8.1,

$$\gcd(a, 0) = a.$$

Hence $a = \gcd(A, B)$ *[as was to be shown].*