LAFF-On
Programming for Correctness

Margaret E. Myers

Robert A. van de Geijn

Release Date October 18, 2016

This is a work in progress

Library of Congress Cataloging-in-Publication Data not yet available
Draft Edition, Fall 2016
This "Draft Edition" allows this material to be used while we sort out through what mechanism we will publish the book.

Contents

Preface

Acknowledgments

Part 3

A Review of Logic and Proofs

3.1 Opening Remarks

3.1.1 Launch

3.1.2 Outline Week 3

3.1.3 What you should know

3.1.4 What you will learn

This week introduces the reader to the systematic derivation of algorithms for linear algebra operations. Through a very simple example we illustrate the core ideas: We describe the notation we will use to express algorithms; we show how assertions can be used to establish correctness; and we propose a goal-oriented methodology for the derivation of algorithms. We also discuss how to incorporate an analysis of the cost into the algorithm. Finally, we show how to translate algorithms to code so that the correctness of the algorithm implies the correctness of the implementation.

Upon completion of this week, you should be able to

•

Track your progress in Appendix **??**.

		not	and	or	implies	equivalent
p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Figure 3.1: Truth table for the basic logic operations.

3.2 Propositions

3.2.1 Boolean functions

Notice: much of the below was inspired by "The Science of Programming" [?]

Definition 3.1 *A Boolean variable is a variable that can take on the values true (T) or false (F).*

We will denote the set of all Boolean values $\mathbb{B} = \{T, F\}$.

Definition 3.2 *A Boolean function* $f : \mathbb{B}^k \rightarrow \mathbb{B}$ *maps a vector* $x \in \mathbb{B}^k$ *to a value in* $\mathbb{B}$.

In other words, it is a function that takes as input Boolean values and produces as output T or F.

3.2.2 Basic logic operations

We will use the symbols $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ to denote the logical "not", "and", "or", "implies", and "is equivalent" operations. In Figure 3.1 a truth table that summarizes all the basic operations is summarized. Below we discuss each individual operation. *truth table* define these operation:

Negation (not). The "not" operation maps a boolean to a boolean, $\neg : \mathbb{B} \rightarrow \mathbb{B}$. Given boolean p, $\neg p$ evaluates to F if p is originally T, and F otherwise

Conjunction (and). The "and" operation maps two boolean to a boolean, $\neg : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$. Given boolean p and q, the expression $p \wedge q$ evaluates to T if and only if both p and q are *true*.

Disjunction (or).

Implication (implies).

Equivalence (equivalent).

3.2.3 Propositions

In an algebra course you will have been introduced to algebraic expression. For example, if $x \in \mathbb{Z}$ then $2x+1$ is an algebraic expression. It consists of a variable (x), constants (2 and 1), algebraic operations (multiplication and addition), and rules by which these are composed.

Propositions are expressions that evaluate to T or F. The set of all propositions is defined by the following rules:

1. T and F are propositions.
2. Sometimes we will use an identifier to denote an expression that evaluates to T or F. An identifier is a proposition.
3. If E is a proposition, so is $\neg$E.
4. If E1 and E2 are propositions, so are
   - (E1 $\wedge$ E2),
   - (E1 $\vee$ E2),
   - (E1 $\Rightarrow$ E2), and
   - (E1 $\Leftrightarrow$ E2).

Notice that subexpressions in a proposition themselves can be algebraic expressions. For example

$$(((2x+1) > 4) \vee (y < 1) \wedge (p \vee q)),$$

where x and y are integers, and p and q are boolean variables, is a proposition. It has algebraic expressions as subexpressions, but evaluates to T or F.

Example 3.3 Examples of propositions:

$$F \quad (\neg F) \quad (p \wedge r) \quad ((p1 \wedge r1) \vee (p2 \Rightarrow r2))$$

To avoid the unnecessary extra parentheses, there is a *precedence order* in which operations are evaluated:

- First negate: $\neg p \wedge q$ is the same as $(\neg p) \wedge q$.
- Second, evaluate $\wedge$: $\neg p \wedge q \Rightarrow r$ is the same as $((\neg p) \wedge q) \Rightarrow r$. Third, evaluate $\vee$: $\neg p \wedge q \vee r \Rightarrow s$ is the same as $(((\neg b) \wedge c) \vee r) \Rightarrow s$.
- Fourth, evaluate $\Rightarrow$: $t \Leftrightarrow \neg p \wedge q \vee r \Rightarrow s$ is the same as $t \Leftrightarrow ((((\neg b) \wedge c) \vee r) \Rightarrow s)$.
- Last evaluate $\Leftrightarrow$.

We will throw in extra parentheses if we think it makes the proposition clearer! So should you!

Exercise of precedence of operations here.

3.2.4 The Basic Equivalences ("Laws of Equivalence")

Definition 3.4 *A tautology is a proposition (statement) that always evaluates to true.*

In Figure 3.2 we provide a set of tautologies that are known as the *Basic Equivalences* (*Laws of Equivalence*). These laws will be our principal tools for manipulating propositions, for example, as part of a proof. Each of these can be easily proved, for example by using a truth table:

E1	E2	(E1 $\wedge$ E2)	(E2 $\wedge$ E1)	(E1 $\wedge$ E2) $\Leftrightarrow$ (E2 $\wedge$ E1)
T	T	T	T	T
T	F	F	F	T
F	T	F	F	T
T	F	F	F	T

You should learn them and their names well.

Homework 3.2.4.1 Use the Basic Equivalences to prove the following. (Do NOT use the weakening/strengthening laws given in Figure 3.3, which we will discuss later.)

1. $(b \wedge (b \Rightarrow c)) \Rightarrow c$.
2. $p \wedge q \Rightarrow p$
3. $p \Rightarrow p \vee r$
4. $p \wedge q \Rightarrow p \vee r$
5. $((x \wedge y) \Rightarrow z) \Leftrightarrow (x \Rightarrow (y \Rightarrow z))$

☛ SEE ANSWER

Exercises 2–4 will become powerful weapons as we prove programs correct. Together we will call them the *Weakening/Strengthening Laws*. They are summarized in Figure 3.3. These form a second set of useful tautologies.

Homework 3.2.4.2 In Figure 3.3 we present three Weakening/Strengening Laws. This exercise shows that if you only decide to remember one, it should be the last one.

1. Show that (E1 $\wedge$ E2) $\Rightarrow$ E1 is a special case of (E1 $\wedge$ E2) $\Rightarrow$ (E1 $\wedge$ E3).
2. Show that E1 $\Rightarrow$ (E1 $\vee$ E3) is a special case of (E1 $\wedge$ E2) $\Rightarrow$ (E1 $\wedge$ E3).

☛ SEE ANSWER

What you will find later is that it is (E1 $\wedge$ E2) $\Rightarrow$ (E1 $\wedge$ E3) becomes our tool of choice in many proofs.

Homework 3.2.4.3 Prove the *counterpositive:* $(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$.

☛ SEE ANSWER

Let E1 , E2 , and E3 be any propositions whatsoever. Then

Commutativity:	$(E1 \wedge E2) \Leftrightarrow (E2 \wedge E1)$
	$(E1 \vee E2) \Leftrightarrow (E2 \vee E1)$
	$(E1 \Leftrightarrow E2) \Leftrightarrow (E2 \Leftrightarrow E1)$
Associativity:	$E1 \wedge (E2 \wedge E3) \Leftrightarrow (E1 \wedge E2) \wedge E3$
	$E1 \vee (E2 \vee E3) \Leftrightarrow (E1 \vee E2) \vee E3$
Distributivity:	$E1 \wedge (E2 \vee E3) \Leftrightarrow (E1 \wedge E2) \vee (E1 \wedge E3)$
	$E1 \vee (E2 \wedge E3) \Leftrightarrow (E1 \vee E2) \wedge (E1 \vee E3)$
De Morgan:	$\neg(E1 \wedge E2) \Leftrightarrow (\neg E1 \vee \neg E2)$
	$\neg(E1 \vee E2) \Leftrightarrow (\neg E1 \wedge \neg E2)$
Negation:	$\neg(\neg E1) \Leftrightarrow E1$
Excluded Middle:	$E1 \vee \neg E1 \Leftrightarrow T$
Contradiction:	$E1 \wedge \neg E1 \Leftrightarrow F$
Implication:	$(E1 \Rightarrow E2) \Leftrightarrow (\neg E1 \vee E2)$
Equality:	$(E1 \vee E2) \Leftrightarrow (E1 \Rightarrow E2) \wedge (E2 \rightarrow E1)$
$\vee$-simplification:	$E1 \vee E1 \Leftrightarrow E1$
	$E1 \vee T \Leftrightarrow T$
	$E1 \vee F \Leftrightarrow E1$
	$E1 \vee (E1 \wedge E2) \Leftrightarrow E1$
$\wedge$-simplification:	$E1 \wedge E1 \Leftrightarrow E1$
	$E1 \wedge T \Leftrightarrow E1$ $E1 \wedge F \Leftrightarrow F$ $E1 \wedge (E1 \vee E2) \Leftrightarrow E1$
Identity:	$E1 \Leftrightarrow E1$

Figure 3.2: Basic equivalences.

Weakening/ Strengthening:	$(E1 \wedge E2) \Rightarrow E1$
	$E1 \Rightarrow (E1 \vee E2)$
	$(E1 \wedge E2) \Rightarrow (E2 \vee E3)$

Figure 3.3: Weaking/strengthening laws.

3.2.5 Predicates

Homework 3.2.5.4 Let us consider a one dimensional array $b(1:n)$ (using Matlab notation), where $1 \leq n$. Let j and k be two integer variables satisfying $1 \leq j \leq k \leq n$. By $b(j:k)$ we mean the subarray of b consisting of $b(j), b(j+1), \ldots b(k)$. The segment $b(j:k)$ is empty if $j > k$.
Translate the following sentences into predicates.

1. All elements of $b(j:k)$ equal zero.
2. No value of $b(j:k)$ is zero.
3. Some values of $b(j:k)$ are zero.
4. All zeros in $b(1:n)$ are in subarray $b(j:k)$.
5. Some zeros in $b(1:n)$ are in subarray $b(j:k)$.
6. Those values in $b(1:n)$ that are not in $b(j:k)$ are in $b(j:k)$.
7. It is not the case that all zeros of $b(1:n)$ are in $b(j:k)$.
8. If $b(1:n)$ contains a zero, then so does $b(j:k)$.
9. It is not the case that all zeros of $b(1:n)$ are in $b(j:k)$.
10. Either $b(1:j)$ or $b(j:k)$ contains a zero (or both).
11. The values of $b(j:k)$ are in ascending order.
12. The segment $b(j:k)$ contains at least two zeros.
13. Every element of $b(1:j)$ is less than x and every value of $b(j+1:k)$ exceeds x.

☛ SEE ANSWER

3.2.6 Specifications

Homework 3.2.6.5 Formalize the following English specifications. Be sure to introduce necessary restrictions. Use $\min(x, y)$ and $\max(x, y)$ to denote the minimum and maximum of x and y, respectively.

1. Calculate the sum of elements $b(j : k)$.
2. Find the maximum value of $b(j : k)$.
3. Find the index of a maximum value of $b(j : k)$.
4. Store in array $c(1 : n)$ a sorted (in ascending order) permutation of $b(1 : n)$. Use the predicate $\text{perm}(b, c, n)$ to denote that $b(1 : n)$ is a permutation of $c(1 : n)$. Use the predicate $\text{ascending}(b, n)$ to indicate that the elements of $b(1 : n)$ are in ascending order.
5. Calculate the greatest power of 2 that is not greater than n.
6. Count how many zeros $b(1 : n)$ has.
7. Suppose we have an array of integers $b(1 : n)$. Each of its subsegments $b(i : j)$ has a sum. Find the largest such sum. Use the symbol $S_{i,j}$ for $\sum_{k=i}^{j} b(k)$.
8. Assume that array $b(1 : n)$ is sorted. Find the highest index of an element in b that equals x. Be sure to take care of the case where x is not in array b.

☛ SEE ANSWER

3.2.7 Weaker/stronger predicates

In our discussion, the notion of one predicate being "stronger" or "weaker" than another predicate will play an important role in reasoning about code.

Consider the code segment

$$\begin{array}{l} \{P : x = 5\} \\ y := x + 1 \\ \{R : y > 3\} \end{array}$$

This code segment evaluates to T because executing $y := x + 1$ in a state where $x = 5$ will complete (in a finite amount of time) in a state in which $y > 3$ is T.

Notice that there are many other states such that executing $y := x + 1$ leaves you in a state where $y > 3$: $\{P : x = 4\}$, $\{P : x = 3\}$ etc. Indeed, any time $x \geq 3$ it is the case that $y := x + 1$ leaves you in a state where $y > 3$. We can indicate this with

$$\begin{array}{l} \{P : x = 5\} \\ \{x \geq 3\} \\ y := x + 1 \\ \{R : y > 3\} \end{array}$$

So, it now merely suffices to determine whether $x = 5$ means $x \geq 3$, because then we know that the code segment is correct. Clearly, $(x = 5) \Rightarrow (x > 3)$ in this case, so

$\{P : x = 5\}$
$y := x + 1$
$\{R : y > 3\}$

is correct.

When two predicates p and q have the property that $p \Rightarrow q$, the predicate q is said to be *weaker* (less restrictive) than predicate p. Equivalently, p is said to be *stronger* (more restrictive) than q.

How do we most systematically show that $x \geq 3$ is weaker than $x = 5$ using what we have learned before?

$$\begin{array}{ll} & (x=5) \Rightarrow (x \geq 3) \\ \Leftrightarrow & < \text{algebra} > \\ & (x=5) \Rightarrow (x \geq 6) \vee (x=5) \vee (x=4) \\ \Leftrightarrow & < \text{Weakening/strengthening law (twice)} > \\ & T \end{array}$$

Here we employ the "weakening/strengthening laws" from Figure 3.3.

Homework 3.2.7.6 For each of the following, if applicable, determine which predicate is the weaker predicate:

1. $0 \leq x \leq 10$ and $1 \leq x < 5$.
2. $x = 5 \wedge y = 4$ and $y = 4$.
3. $x \leq 5 \vee y = 3$ and $x = 5 \wedge y = 4$.
4. T and F.
5. $(\forall i | 5 \leq i \leq 10 : b(i+1) < b(i))$ and $(\forall i | 7 \leq i \leq 10 : b(i+1) < b(i))$
6. $x \leq 1$ and $x \geq 5$.

☛ SEE ANSWER

3.3 A Simple Language

3.3.1 The weakest precondition

We are now going to define a simple language that will allow us to very precisely reason about program correctness.

Consider a Hoare triple

$$\{P\} y := x + 1 \{y > 5\}.$$

For what values of x, specified by predicate P, will executing $y := x + 1$ complete in a state where $y > 5$? Some possible answers:

- $P : x = 5$. Clearly, y will end up equaling $y = 6$ and hence $\{y > 5\}$ holds after $y := x + 1$ is executed. In other words, the predicate $\{x = 5\} y := x + 1 \{y > 5\}$ evaluates to T. It is a correct code segment.

- $P : x \geq 5$. Clearly, y will end up satisfying $\{y > 5\}$. Again, the predicate $\{x = 5\} y := x + 1 \{y > 5\}$ evaluates to T. It is a correct code segment.

The real question is

"What is the set of all states (of relevant variables) such that executions of $y := x + 1$ results in a state where $y > 5$?

Let us look at this question more generally: Consider the command S and let us assume that we would like to check whether the code segment

$\{P\}$
S
$\{R\}$

is correct. Now, what if we had a magic function, $wp(command, R)$ that gives us a predicate that describes all conditions under which S completes in a state where R is true. We then know that

$\{P\}$
S
$\{R\}$

is correct (T) only if

$\{P\}$
$\{wp(S,R)\}$
S
$\{R\}$

is correct. Now, what does this mean? If P is true, it better imply that $wp(S,R)$ is true, because no computation happens between $\{P\}$ and $\{wp(S,R)\}$. What this means is that the code segment is correct if and only if

$$P \Rightarrow wp(S,R).$$

This magic function is known as the weakest precondition so that command S completes in a state described by predicate R.

Notice

- Any P such that $\{P\}S\{R\}$ is T is a precondition so that command S completes in a state described by predicate R.
- If Q satisfies $P \Rightarrow Q$ then Q is weaker: It is *less* restrictuve than predicate Q.
- The *least* restrictive predicate Q such that $\{Q\}S\{R\}$ is T describes the set of *all* states in which statement S can be executed so that it completes (in a finite amout of time) in a state where R is T.
- The *wp* is the function that takes the command S and the postcondition R, and returns a predicate that describes the set of all states such that executing S completes (in a finite amount of time) in a state that satisfies the postcondition R.

It will take a bit of practice to fully understand and appreciate this. In the next subsections, we actually use *wp* to define a simple language.

Homework 3.3.1.7 For each of the below code segments, determine the weakest precondition (by examination):

1. $wp(\text{“}y := x - 1\text{”}, y \leq 1) =$
2. $wp(\text{“}x := x - 1\text{”}, x \leq 1) =$
3. $wp(\text{“}\alpha := \alpha + b(i)\text{”}, \alpha = \sum_{k=0}^{i} b(k)) =$

☛ SEE ANSWER

3.3.2 Properties of *wp*

We keep repeating this, but let's once again consider

$$wp(S,R)$$

and its interpretation:

> $wp(S,R)$ notes the weakest predicate so that execution of S started in a state that satisfies this predicate is guaranteed to terminate in a finite amount of time in a state that satisfies R.

Another way of saying this is

> $wp(S,R)$ notes the set of states so that if the execution of S is started in any of these states, it is guaranteed to terminate in a finite amount of time in a state such that R is *true*.

Try to internalize these interpretations.

We will now reason that for a language with reasonable semantics, the *wp* operator should obey the following properties. Afterwards, we will take them as axioms.

Law of Excluded Miracle. What if the predicate R is the state described by F (*false*)? Let's plug this into the second interpretation:

> $wp(S,F)$ notes the set of states so that if the execution of S is started in any of these states, it is guaranteed to terminate in a finite amount of time in a state such that *false* is *true*.

Now, obviously there is no state that has this property. The predicate that describes ”no states” is F (*false*). We conclude that

$$wp(S,F) = F$$

for all reasonably defined commands S. This is known as the *Law of Excluded Miracle*.

Law Distributivity of Conjunction. Next, let us consider an arbitrary command S and postconditions Q and R. Then

$$wp(S,Q) \wedge wp(S,R) = wp(S,Q \wedge R)$$

Why is this?

Remember that $wp(S,Q) \wedge wp(S,R)$ describes a set of states s. If s satisfies $wp(S,Q) \wedge wp(S,R)$ then it satisfies $wp(S,Q)$ and hence it has the property that if S is executed with state s then it will complete in a state where Q is true. Similarly, s also satisfies $wp(S,R)$ and hence it has the property that if S is executed with state s then it will complete in a state where R is true. Thus, $s$has the property that if S is executed with state s then it will complete in a state where Q and R are true. This shows that s also satisfies $wp(S,Q \wedge R)$.

Now, if s satisfies $wp(S,Q \wedge R)$ then it has the property that if S is executed with state s then it will complete in a state where $Q \wedge R$ is true. But that means it completes in a state where Q is true and hence s also satisfies $wp(S,Q)$. Similarly, we can argue that it also satisfies $wp(S,R)$. We conclude that it satisfies $wp(S,Q) \wedge wp(S,R)$.

Law of Monotonicity. The Law of Monotonicity is given by

If $Q \Rightarrow R$ then $wp(S,Q) \Rightarrow wp(S,R)$.

Here is the way we will reason that a statement S in a reasonable language has this property.

- The definition of *wp* means the following Hoare triple (annotated code segment) evaluates to *true* (is correct):

  $\{wp(S,Q)\}$
  S
  $\{Q\}$

- The fact that $Q \Rightarrow R$ means that the following annotated code segment is also correct:

  $\{wp(S,Q)\}$
  S
  $\{Q\}$
  $\{R\}$

- Hence the Hoare triple

  $\{wp(S,Q)\}$
  S
  $\{R\}$

  evaluates to T.

- But a Hoare triple only evaluates to *true* if its precondition implies the weakest precondition.

- Hence $\{wp(S,Q) \Rightarrow wp(S,R)\}$.

Law of Distributivity of Disjunction. Finally, we discuss Distributivity of Disjunction

$$(wp(S,Q) \vee wp(S,R)) \Rightarrow wp(S,Q \vee R)$$

The following exercise prepares us for the reasoning behind this axiom:

Homework 3.3.2.8 Prove that

$$((p \Rightarrow r) \wedge (q \Rightarrow r)) \Leftrightarrow ((p \vee q) \Rightarrow r)$$

☛ SEE ANSWER

Here is the way we will reason that a statement S in a reasonable language obeys Distributivity of Disjunction:

- The definition of *wp* means the following Hoare triple (annotated code segment) evaluates to *true* (is correct):

  $\{wp(S,Q)\}$
  S
  $\{Q\}$

- We know from the Weakening/Strengthening Laws that $Q \Rightarrow Q \vee R$ and hence

  $\{wp(S,Q)\}$
  S
  $\{Q\}$
  $\{Q \vee R\}$

- Hence we conclude that the Hoare triple

  $\{wp(S,Q)\}$
  S
  $\{Q \vee R\}$

  evaluates to T.

- But a Hoare triple only evaluates to *true* if its precondition implies the weakest precondition.

- Hence,
$$wp(S,Q) \Rightarrow wp(S,Q \vee R).$$

- Similarly, we can conclude that
$$wp(S,R) \Rightarrow wp(S,Q \vee R).$$

- In other words,
$$(wp(S,Q) \Rightarrow wp(S,Q \vee R)) \wedge (wp(S,R) \Rightarrow wp(S,Q \vee R))$$

- By the last homework, this is equivalent to
$$(wp(S,Q) \vee wp(S,R) \Rightarrow wp(S,Q \vee R)).$$

We take this property to now be an axiom.

3.3.3 skip

Let us consider the command **skip**, which simply doesn't do anything:

$\{P : ?\}$
skip
$\{R : x > 4\}$

From what state P will the command **skip** finish (in a finite amount of time) in a state where $x > 4$ is T? Obviously, $x > 4$ better be true before the **skip** command is executed. Reasoning through this, it becomes clear that,

$$wp(\textbf{skip}, R) = R$$

for any predicate R: The set of all states such that executing **skip** completes (in a finite amount of time) in a state where R is true is described by the predicate R.

Homework 3.3.3.9 Prove that **skip** command satisfies the Laws of Excluded Micacle, Distributivity of Conjunction, Monotonicity, and Distributed Disjunction.

☛ SEE ANSWER

3.3.4 abort

Let us consider the command **abort**, which stops the execution of the program:

$\{P\}$
abort
$\{R : x > 4\}$

From what state P will the command **abort** finish (in a finite amount of time) in a state where $x > 4$ is T? Obviously, it never reaches the point in the program where we would want $R : x > 4$ to be T. Thus, no matter what state the variables are in before **abort**, it will never reach the point in the program where the postcondition R is to be T. This means

$$wp(\textbf{abort}, R) = F$$

for any predicate R: The set of all states such that executing **abort** completes (in a finite amount of time) in a state where R is true is described by the predicate F, the most restrictive predicate of all.

Homework 3.3.4.10 Prove that **abort** command satisfies the Laws of Excluded Micacle, Distributivity of Conjunction, Monotonicity, and Distributed Disjunction.

☛ SEE ANSWER

3.3.5 Composition

Obviously, a program that consists of a single statement, S, is not a very interesting program. Consider the program that consists of two arbitrary statements

S_0
S_1
$\{R\}$

which we can also write as $S_0;S_1$. We would like to determine

$$wp(``S_0;S_1'', R)$$

Let us think about this, working backwards. The states from which executing S_1 leaves one in a state where R is true is given by $wp(S_1, R)$. Thus, over executing S_0 one must be in a state where $wp(S_1, R)$ is T. The predicate that describes the states from which executing S_0 leaves one in a state wher $wp(S_1 R)$ is T is described by $wp(S_0, wp(S_1, R))$. In other words, only the following assertions make the code segment $S_0;S_1$ complete (in a finite amount of time) in a state where R is T:

$\{wp(S_0, wp(S_1, R))\}$
S_0
$\{wp(S_1, R)\}$
S_1
$\{R\}$

Thus

$$wp(``S_0;S_1'', R) = wp(S_0; wp(S_1, R))$$

Homework 3.3.5.11 Prove that composition of statements satisfies the Laws of Excluded Micacle, Distributivity of Conjunction, Monotonicity, and Distributed Disjunction.

☛ SEE ANSWER

3.3.6 Assignment to a simple variable

Consider again the code segment

$\{P : x = 5\}$
$y := x + 1$
$\{R : y > 3\}$

While it is easy to argue that if $x = 5$ then then the assignment $y := x + 1$ leaves one in a state where $y > 3$. The question we must answer is who to describe *all* states such that executing $y := x + 1$ leaves us in a state where $y > 3$ holds. More importantly, we need to expose a systematic way of taking an arbitrary postcondition, and transforming it into the weakest precondition for an arbitrary assignment.

Going back to our example, notice that after the assignment $R : y > 3$. The expression $x + 1$ was just assigned to y. Thus, only if $(x + 1) > 3$ before the assignment will the assignment leave the program in a state where $y > 3$.

This example leads us to the definition

$$wp(``x := e'', R) = R^x_e$$

where R^x_e equals the predicate R with all *free* occurrences of variable x replaced by the expression (e). We will refer to this as the *textual substitution* of variable x by expression e. The parentheses are needed to makes sure that the order of operations is properly preserved. What we mean by a free occurrence of a variable will become clear shortly. Thus,

$$wp(``\underbrace{y := x + 1}_{y := e}'', \underbrace{y > 3}_{R}) = \underbrace{((x + 1) > 3)}_{R^y_e} = (x + 1 > 3) = (x > 2).$$

Homework 3.3.6.12 For each of the below code segments, determine the weakest precondition:

1. $wp(i := i+1, i = j)$
2. $wp(i := i+1; j := j+i, i = j)$
3. $wp(i := 2i+1; j := j+i, i = j)$
4. $wp(j := j+i; i := 2i+1, i = j)$
5. $wp(\text{“}t := i; i := j; j := t\text{”}, i = \widehat{i} \wedge j = \widehat{j})$
6. $wp(\text{“}i = 0; s := 0\text{”}, 0 \leq i < n \wedge s = (\sum j | 0 \leq j < i : b(j)))$
7. $wp(\text{“}s := s + b(i); i = i+1\text{”}, 0 < i \leq n \wedge s = (\sum j | 0 \leq j < i : b(j))) =$
   (Note: I changed the problem slightly on the evening of Oct. 13.)

☛ SEE ANSWER

Now, consider the following problem:

$$wp(\text{“}i := 0\text{”}, (\forall i | 0 \leq i < 5 : b(i) = 0))$$

A naive evaluation would yield

$$(\forall i | 0 \leq i < 5 : b(i) = 0)_0^i = (\forall 0 | 0 \leq 0 < 5 : b(0) = 0)$$

which is obviously nonsense. The problem here is that in the predicate the i is a “bound variable” of the quantification. If one replaced it with, for example, j, then

$$\begin{aligned} wp(\text{“}i := 0\text{”}, (\forall i | 0 \leq i < 5 : b(i) = 0)) &= (\forall j | 0 \leq j < 5 : b(j) = 0)_0^i \\ &= (\forall j | 0 \leq 0 < 5 : b(j) = 0) \end{aligned}$$

which does makes sense. The point: If a variable that is bound to a quantification in R appears in the assignment, then the bound variable should be replaced with something that does not appear in the assignment before performance the textual substitution.

We are now in a position to revisit how the weakest precondition can be used to prove a code segment correct. Consider an array $b(1:n)$, $0 \leq n$, a scalar variable α, and the code segment

$\{P:\ 0 \leq n \wedge \alpha = 0\}$
$S:\ k := 0$
$\{R:\ 0 \leq k \leq n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))\}$

This code segment may be part of a program that sums the entries in array $b(i)$. Is this code segment correct? To determine this, we must check that

$$P \Rightarrow wp(S, R).$$

The proof that this holds is given by

$$\begin{array}{ll} & P \Rightarrow wp(S,R) \\ \Leftrightarrow & < \text{Instantiate } P, S, \text{ and } R > \\ & (0 \leq n) \wedge (\alpha = 0)) \Rightarrow wp(\text{“}k := 0\text{”}, 0 \leq k \leq n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))) \\ \Leftrightarrow & < \text{definition of} := > \\ & ((0 \leq n) \wedge (\alpha = 0)) \Rightarrow (0 \leq k \leq n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))_0^k \\ \Leftrightarrow & < \text{definition of } R_e^k > \\ & ((0 \leq n) \wedge (\alpha = 0)) \Rightarrow (0 \leq 0 \leq n \wedge \alpha = (\sum i | 1 \leq i \leq 0 : b(i))) \\ \Leftrightarrow & < \text{Summation over empty range, poperty of} \leq > \\ & ((0 \leq n) \wedge (\alpha = 0)) \Rightarrow ((0 \ \leq n) \wedge (\alpha = 0)) \\ \Leftrightarrow & < p \Rightarrow p > \\ & T \end{array}$$

This, the code segment is correct.

Homework 3.3.6.13 Consider an array $b(1:n)$, $1 \leq n$, a scalar variable α, and the code segment

$$\begin{array}{l} \{P: \ 0 \leq k < n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))\} \\ S_0: \ k := k+1 \\ S_1: \ \alpha := \alpha + b(k) \\ \{R: \ 0 \leq k \leq n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))\} \end{array}$$

This code segment may be part of a program that sums the entries in array $b(i)$. Prove this code segment correct.

☛ SEE ANSWER

3.3.7 Assignment to an array element

Assignment to an array element is a bit more complicated. In order to define the weakest precondition of an assignment like $b(i) := x$, we need to introduce a new notation:

$$(b; i : e)$$

which equals a copy of array b, but with the ith entry set to the result of evaluating expression e.

With this new notation/function, the assignment

$$b(i) := e$$

can be thought of as the reassignment of the entire array

$$b := (b; i : e)$$

and this allows us to define

$$wp(\text{“}b(i) = e\text{”}, R) = wp(\text{“}b = (b; i : e)\text{”}, R) = R^b_{(b;i,e)}$$

much like assignment to a simple variable. Now strictly speaking it needs to be also asserted that i is in the correct range and that e is a valid expression. We will assume that implicitly.

Let us check whether this has the desired effect by considering the following code segment that starts with $b(1:n) = c(1:n)$, assigns e to $b(i)$ and then asserts that all entries in b should equal those in c, except for the ith one.

$$\{Q : 1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n : b(k) = c(k))\}$$
$$b(i) := e$$
$$\{R : 1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : b(k) = c(k)) \wedge b(i) = e\}$$

$Q \Rightarrow wp(\text{“}b(i) := e\text{”}, R)$

$\Leftrightarrow$ < equivalent definition of $:=$ >

$Q \Rightarrow wp(\text{“}b := (b; i : e)\text{”}, R)$

$\Leftrightarrow$ < definition of $:=$ >

$Q \Rightarrow R^b_{(b;i:e)}$

$\Leftrightarrow$ < instantiate R >

$Q \Rightarrow (1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : b(k) = c(k)) \wedge b(i) = e)^b_{(b;i:e)}$

$\Leftrightarrow$ < definition of R^x_e >

$Q \Rightarrow (1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : (b; i : e)(k) = c(k)) \wedge (b; i : e)(i) = e)$

$\Leftrightarrow$ < definition of $(b; i : e)$ >

$Q \Rightarrow (1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : b(k) = c(k)) \wedge T)$

$\Leftrightarrow$ < instantiate Q >

$(1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n : b(k) = c(k))) \Rightarrow (1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : b(k) = c(k)) \wedge T)$

$\Leftrightarrow$ < split range; $\wedge$-simplification >

$(1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : b(k) = c(k)) \wedge b(i) = c(i))$

$\qquad \Rightarrow (1 \leq i \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i : b(k) = c(k))$

$\Leftrightarrow$ < weakening/strengthening >

T

Homework 3.3.7.14 Consider the following code segment that swaps the contents of $b(i)$ and $b(j)$. Prove it correct.

```
$\{Q : 1 \leq i \leq j \leq n \wedge (\forall k | 1 \leq k \leq n : b(k) = c(k))\}$
$t = b(i)$
$b(i) = b(j)$
$b(j) = t$
$\{R : 1 \leq i \leq j \leq n \wedge (\forall k | 1 \leq k \leq n \wedge k \neq i \wedge k \neq j : b(k) = c(k)) \wedge b(i) = c(j) \wedge b(j) = c(i)\}$
```

☛ SEE ANSWER

3.3.8 The if statement

We are now ready to discuss the **if** statement. It takes the following form:

```
**if**
$G_0 \to S_0$
$\square\, G_1 \to S_1$
$\square \quad \vdots$
$\square\, G_{k-1} \to S_{k-1}$
**endif**
```

To describe how it works, it is best to add a precondition and a postcondition:

```
$\{P\}$
**if**
  $G_0 \to S_0$
$\square\, G_1 \to S_1$
  $\vdots$
$\square\, G_{k-1} \to S_{k-1}$
**endif**
$\{Q\}$
```

Here

- Each of $G_i \to S_0$ should be interpreted as "*if* G_i *then execute* S_i and jump to immediately after the **endif** ". Here G_i is said to be the *guard* for command S_i and $G_i \to S_i$ a *guarded statement*
- When the **if** command is reached, it must be the case that at least one of the guards evaluates to *true*. Thus, it must be the case that $P \Rightarrow (G_0 \wedge G_1 \wedge \cdots \wedge G_{k-1})$.
- If more than one guard evaluates to true, one should assume that the statement associated with one of these is executed, but which one is not prescribed.

This last bullet means that **our programs can be nondeterministic.**

Let us annotate this code segment some more, which will motivate how $wp(\textbf{if}, R)$ is defined:

```
$\{wp(\mathtt{IF},Q)\}$
if
  $G_0 \rightarrow \{wp(S_0,Q)\}S_0\{Q\}$
$[]\, G_1 \rightarrow \{wp(S_1,Q)\}S_1\{Q\}$
  $\vdots$
$[]\, G_{k-1} \rightarrow \{wp(S_{k-1},Q)\}S_{k-1}\{Q\}$
endif
$\{Q\}$
```

What does this mean? By examining this annotated code we can justify the following definition of the weakest precondition for this generic `IF` command:

$$wp(\mathtt{IF},R) = \overbrace{(G_0 \lor G_1 \lor \cdots \lor G_{k-1})}^{\text{at least one guard must evaluate to } true} \land \overbrace{(G_0 \Rightarrow wp(S_0,Q))}^{\text{if } G_i \text{ then } S_i \text{ may execute, in which case it must start in a state that guarantees completion in a state where } Q \text{ holds.}} \land (G_1 \Rightarrow wp(S_1,Q)) \;\vdots\; \land (G_{k-1} \Rightarrow wp(S_{k-1},Q)) \tag{3.1}$$

Here we say "if G_i then S_i *may* execute," because in the case where multiple guards are *true*, only one command S_i is executed. Since we don't know which one, they *all* must have the property that they leave the program in a state where Q is *true*. Implicit in this definition is that each of the guards G_i must be well-defined, since otherwise the **if** statement will abort. Equivalently, we can state (3.1) as

$$wp(\mathtt{IF},R) = \overbrace{(\exists i | 0 \le i < k : G_i)}^{\text{at least one guard must evaluate to } true} \land \overbrace{(\forall i | 0 \le i < k : G_i \Rightarrow wp(S_i,Q)).}^{\text{if } G_i \text{ then } S_i \text{ may execute, in which case it must start in a state that guarantees completion in a state where } Q \text{ holds.}}$$

(Examples taken from Gries.)

Example 3.5 Consider the following code segment.

```
if
  $x \ge 0 \;\rightarrow\; z := x$
$[]\, x \le 0 \;\rightarrow\; z := -x$
endif
```

Prove that regardless of the original value of scalar x, it sets z to the absolute value of x.

Notice that this program is not deterministic in the sense that if $x = 0$ either $z := x$ or $z := -x$ may be executed.

We must prove that

$$T \Rightarrow wp(\texttt{IF}, z = \mathrm{abs}(x))$$

Notice that $T \Rightarrow p$ is equivalent to proving p is *true*:

$wp(\texttt{IF}, z = \mathrm{abs}(x))$

$\Leftrightarrow$ < Definition of IF >

$$\underbrace{(x \geq 0 \vee x \leq 0)}_{G_0 \vee G_1} \wedge \underbrace{(x \geq 0 \Rightarrow wp(\text{“}z := x\text{”}, z = \mathrm{abs}(x))}_{G_0 \Rightarrow wp(S_0, Q)} \wedge \underbrace{(x \leq 0 \Rightarrow wp(\text{“}z := -x\text{”}, z = \mathrm{abs}(x))}_{G_1 \Rightarrow wp(S_1, Q)}$$

$\Leftrightarrow$ < definition of := >

$(x \geq 0 \vee x \leq 0) \wedge (x \geq 0 \Rightarrow x = \mathrm{abs}(x)) \wedge (x \leq 0 \Rightarrow -x = \mathrm{abs}(x))$

$\Leftrightarrow$ < algebra >

$T \wedge T \wedge T$

$\Leftrightarrow$ < $\wedge$ simplification $\times 2$ >

T

■

Let us now revisit the annotated IF command

```
{$P$}
**if**
  $G_0 \to \{P \wedge G_0\} S_0 \{Q\}$
$\square$ $G_1 \to \{P \wedge G_1\} S_1 \{Q\}$
  ⋮
$\square$ $G_{k-1} \to \{P \wedge G_{k-1}\} S_{k-1} \{Q\}$
**endif**
{$Q$}
```

Assuming all guards are well-defined, we notice that we can break up the proof of correctness into parts:

- We know that P must imply that at least one of the guards is *true*:

$$P \Rightarrow (G_0 \vee G_1 \vee \cdots \vee G_{k-1})$$

- For each of the commands S_i we must establish that

$$(P \wedge G_i) \Rightarrow wp(S_i, Q)$$

These observations can be stated as a theorem:

Theorem 3.6 (`IF` Theorem) *The annotated code segment*

```
$\{P\}$
if
$\square$ $G_0 \rightarrow S_0$
  $\vdots$
$\square$ $G_{k-1} \rightarrow S_{k-1}$
endif
$\{Q\}$
```

is correct if and only if

- $P \Rightarrow (\exists i | 0 \le i < k : G_i)$ *and*
- $(\forall i | 0 \le i < k : P \wedge G_i \Rightarrow wp(S_i, Q))$

In other words, under these conditions $P \Rightarrow wp(\mathtt{IF}, Q)$.

Homework 3.3.8.15 Prove that

1. $(p \Rightarrow (q \wedge r)) \Leftrightarrow ((p \Rightarrow q) \wedge (p \Rightarrow r))$.
2. $(p \Rightarrow (q \Rightarrow r)) \Leftrightarrow ((p \wedge q) \Rightarrow r)$
3. $(p \Rightarrow ((q_0 \vee q_1) \wedge (r_0 \Rightarrow s_0) \wedge (r_1 \Rightarrow s_1))) \Leftrightarrow ((p \Rightarrow (q_0 \vee q_1)) \wedge ((p \wedge r_0) \Rightarrow s_0) \wedge ((p \wedge r_1) \Rightarrow s_1))$
(Hint: use 1. and 2.)

☛ SEE ANSWER

Proof: (`IF` Theorem) The proof is a simple extention of the insights in the last homework, where $q = (\exists i | 0 \le i < k : G_i)$ and $r \Rightarrow s$ is $G_i \Rightarrow wp(S_i, Q)$.

With this new tool, let us revisit Example 3.5:

```
$\{T\}$
if
  $x \ge 0 \rightarrow z := x$
$\square$ $x \le 0 \rightarrow z := -x$
endif
$\{z = \mathrm{abs}(x)\}$
```

To check the correctness of this code segment, we check

- $P \Rightarrow G_0 \vee \cdots \vee G_{k-1}$:

$$\begin{array}{l} \quad P \Rightarrow (G_0 \vee G_1) \\ \Leftrightarrow \ < \text{instantiate} > \\ \quad T \Rightarrow (x \geq 0 \vee x \leq 0) \\ \Leftrightarrow \ < \text{Excluded middle} > \\ \quad T \Rightarrow T \\ \Leftrightarrow \ < \text{definition of} \Rightarrow > \\ \quad T \end{array}$$

- $P \wedge G_0 \Rightarrow wp(S_0, Q)$:

$$\begin{array}{l} \quad P \wedge G_0 \Rightarrow wp(S_0, Q) \\ \Leftrightarrow \ < \text{instantiate} > \\ \quad T \wedge (x \geq 0) \Rightarrow wp(\text{“}z := x\text{”}, z = \text{abs}(x)) \\ \Leftrightarrow \ < \wedge\text{-simplification, definition of} := > \\ \quad (x \geq 0) \Rightarrow x = \text{abs}(x) \\ \Leftrightarrow \ < \text{algebra} > \\ \quad T \end{array}$$

- $P \wedge G_1 \Rightarrow wp(S_1, Q)$:

$$\begin{array}{l} \quad P \wedge G_1 \Rightarrow wp(S_1, Q) \\ \Leftrightarrow \ < \text{instantiate} > \\ \quad T \wedge (x \leq 0) \Rightarrow wp(\text{“}z := -x\text{”}, z = \text{abs}(x)) \\ \Leftrightarrow \ < \wedge\text{-simplification, definition of} := > \\ \quad (x \leq 0) \Rightarrow -x = \text{abs}(x) \\ \Leftrightarrow \ < \text{algebra} > \\ \quad T \end{array}$$

Homework 3.3.8.16 The following code segment sets m to the maximum of x and y. Prove it correct.

```
{$T$}
if
  $x \geq y \ \rightarrow \ m := x$
▯ $x \leq y \ \rightarrow \ m := y$
endif
{$m = \max(x, y)$}
```

☛ SEE ANSWER

Homework 3.3.8.17 Prove the following code segment correct:

```
$\{(\forall j | 1 \leq j < i : m \leq b(j))\}$
**if**
  $b(i) \geq m \rightarrow$ **skip**
$\square$ $b(i) \leq m \rightarrow m := b(i)$
**endif**
$i := i+1$
$\{\{(\forall j | 1 \leq j \leq i : m \leq b(j))\}$
```

☛ SEE ANSWER

Homework 3.3.8.18 Prove the following code segment correct:

```
$\{(\forall j | 1 \leq j < i : m \geq b(j))\}$
**if**
  $b(i) \leq m \rightarrow$ **skip**
$\square$ $b(i) \geq m \rightarrow m := b(i)$
**endif**
$i := i+1$
$\{\{(\forall j | 1 \leq j < i : m \geq b(j))\}$
```

☛ SEE ANSWER

The following two exercises are a bit too advanced. I need to rework them.

Homework 3.3.8.19 The following code segment might be part of a loop that computes the maximum value m in array $b(1:n)$ as well as the index i such that $b(i) = m$. Prove it correct.

```
$\{P : (1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m)\}$
**if**
  $b(k) \geq m \rightarrow m := b(k); i := k$
$\square$ $b(k) \leq m \rightarrow$ **skip**
**endif**
$k := k+1$
$\{R : (1 \leq i < k \leq n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m)\}$
```

☛ SEE ANSWER

Homework 3.3.8.20 The following code segment might be part of a loop that computes number of zeroes in array $b(1:n)$. Prove it correct.

```
$\{P : (0 \le k < n) \wedge m = (\mathbb{N} i | 1 \le i < k : b(i) = 0)\}$
**if**
  $b(k) = 0 \rightarrow m := m+1; k := k+1$
$\square$ $b(k) \neq 0 \rightarrow k := k+1$
**endif**
$\{Q : (0 \le k \le n) \wedge m = (\mathbb{N} i | 1 \le i < k : b(i) = 0)\}$
```

Here $(\mathbb{N} i | 1 \le i < k : b(i) = 0)$ equals the number of occurrences of 0 in $b(1:k)$. You can alternatively replace it by $(\sum i | 1 \le i < k \wedge b(i) = 0 : 1)$.

☛ SEE ANSWER

3.3.9 The iterative command

In Part **??** of this course, we introduced the concept of a **while** loop:

```
**while** $G$
    $S$
**endwhile**
```

and showed how to develop loops to be correct. In this section, we first generalize this iterative command, and then return to discuss it as a special case.

Consider

```
**do**
  $G_0 \rightarrow S_0$
$\square$ $G_1 \rightarrow S_1$
      $\vdots$
$\square$ $G_{k-1} \rightarrow S_{k-1}$
**od**
```

Like for the `IF` command, each $G_i \rightarrow S_i$ is a guarded command, where S_i is a candidate for execution if the guard G_i evaluates to T. If G_i is the only guard that evaluates to *true*, then S_i is executed. If multiple guards evaluate to *true*, then exactly one of the corresponding commands is executed. The loop continues to iterate until none of the guards are `true`, at which point control proceeds to the first statement after **do**.

Let us assume that the `DO` loop can be annotated as in Figure 3.4, in preparation for a theorem regarding the correctness of a `DO` loop. Here P_{inv} denotes a *loop invariant* for the loop. To analyze the correctness of the loop $\{Q\}$`DO`$\{R\}$ we notice the following:

- P_{inv} holds before the loop commenses (at 1.),
- When the loop is entered the first time, no computation is performed before 2. is reached. Thus, P_{inv} is true there too.

```
1. $\{Q\}$
2. $\{P_{inv}\}$
   do
3. $\{P_{inv}\}$
4.   $G_0 \rightarrow \{P_{inv} \wedge G_0\}$   $S_0$   $\{P_{inv}\}$          Loop body
   $\square\, G_1 \rightarrow \{P_{inv} \wedge G_1)\}$   $S_1$   $\{P_{inv}\}$
       $\vdots$
   $\square\, G_{k-1} \rightarrow \{P_{inv} \wedge G_{k-1})\}$   $S_{k-1}$   $\{P_{inv}\}$
5. $\{P_{inv}\}$
   od
6. $\{P_{inv}\} \wedge \neg(G_0 \vee \cdots \vee G_{k-1})$
7. $\{R\}$
```

Figure 3.4: Annotated DO loop.

- The guarded commands in the loop body is such that regardless of which guard evaluates to *true* and is chosen, the execution of the corresponding command leaves the program again in a state where P_{inv} is *true*.
- As a result, P_{inv} is *true* before the executions of the loop body (at 3.) and after execution of the loop body (at 4.) for every iteration.
- **If** ever none of the loop guards evaluate to *true*, **then** the loop stops executing, leaving the program in a state where P_{inv} is still *true* and $G_0 \vee \cdots \vee G_{k-1}$ evaluates to *false*, at 6.
- Now, *if* P_{inv} implies R, then we can conclude that $\{Q\}$DO$\{R\}$ is *true*, meaning that this code segment is correct.

The above motivates the following theorem:

Theorem 3.7 (DO **Theorem, Partial Correctness)**

```
$\{Q\}$
$\{P_{inv}\}$
do
   $G_0 \rightarrow \{P_{inv} \wedge G_0\}\ S_0 \{P_{inv}\}$
$\square$ $G_1 \rightarrow \{P_{inv} \wedge G_1\}\ S_1 \{P_{inv}\}$
       $\vdots$
$\square$ $G_{k-1} \rightarrow \{P_{inv} \wedge G_{k-1}\}\ S_{k-1} \{P_{inv}\}$
od
$\{P_{inv} \wedge \neg(G_0 \vee \cdots \vee G_{k-1})\}$
$\{R\}$
```

If predicate P_{inv} satisfies

- $Q \Rightarrow P_{inv}$;
- $P_{inv} \wedge G_i \Rightarrow wp(S_i, P_{inv})$, *for* $i = 0, \ldots, k-1$; *and*
- $(P_{inv} \wedge \neg(G_0 \vee \cdots \vee G_{k-1})) \Rightarrow R$

then the code segment on the left correctly computes a state where R is true **if** *the loop terminates.*

This theorem is a bit tricky to prove and hence we skip that proof, which can be found in [**?**].

Example 3.8 *Consider the loop, which stores in s the sum of the elements of array* $b(1:10)$*:*

```
$\{T\}$
$i := 1; s := 0$
$\{P_{inv} : (1 \le i \le 11 \wedge s = (\sum j | 1 \le j < i : b(j))\}$
do
  $i \le 10 \rightarrow s = s + b(i); i := i + 1$
od
$\{R : s = (\sum j : 1 \le j \le 10 : b(j))\}$
```

and loop invariant $P_{inv} : (1 \le i \le 11 \wedge s = (\sum j | 1 \le j < i : b(j))$*. We prove partial correctness:*

- *Prove that the loop invariant is true before the loop. In other words, show that the initialization* $i := 1; s := 0$ *puts the program in a state where* P_{inv} *is true:*

$$\begin{aligned}
& T \Rightarrow wp(\text{“}i := 1; s := 0\text{”}, P_{inv}) \\
\Leftrightarrow\ & < \textit{instantiate} > \\
& T \Rightarrow wp(\text{“}i := 1; s := 0\text{”}, 1 \le i \le 11 \wedge s = (\sum j | 1 \le j < i : b(j))) \\
\Leftrightarrow\ & < \textit{definition of} :=\textit{, twice} > \\
& T \Rightarrow (1 \le 1 \le 11 \wedge 0 = (\sum j | 1 \le j < 1 : b(j))) \\
\Leftrightarrow\ & < \textit{sum over empty range; algebra} > \\
& T \Rightarrow (T \wedge T) \\
\Leftrightarrow\ & < \wedge\textit{-simplification; } T \Rightarrow T > \\
& T
\end{aligned}$$

- $P_{inv} \wedge G_i \Rightarrow wp(S_i, P_{inv})$*, for* $i = 0, \ldots, k-1$*:*

$P_{inv} \wedge G_0 \Rightarrow wp(S_0; P_{inv})$

$\Leftrightarrow < \textit{instantiate} >$

$(P_{inv} \wedge G_0) \Rightarrow wp(\text{“}s := s + b(i); i := i+1\text{”}, 1 \leq i \leq 11 \wedge s = (\sum j | 1 \leq j < i : b(j)))$

$\Leftrightarrow < \textit{definition of} := >$

$(P_{inv} \wedge G_0) \Rightarrow wp(\text{“}s := s + b(i)\text{”}, 1 \leq i+1 \leq 11 \wedge s = (\sum j | 1 \leq j < i+1 : b(j)))$

$\Leftrightarrow < \textit{definition of} := >$

$(P_{inv} \wedge G_0) \Rightarrow (1 \leq i+1 \leq 11 \wedge s + b(i) = (\sum j | 1 \leq j < i+1 : b(j)))$

$\Leftrightarrow < \textit{instantiate; algebra; split range} >$

$((1 \leq i \leq 11 \wedge s = (\sum j | 1 \leq j < i : b(j)) \wedge (i \leq 10)) \Rightarrow (0 \leq i \leq 10 \wedge s + b(i) = (\sum j | 1 \leq j < i : b(j)) + b($

$\Leftrightarrow < (1 \leq i \leq 11 \wedge i \leq 10) \Leftrightarrow (1 \leq i \leq 10); \textit{algebra} >$

$((1 \leq i \leq 10 \wedge s = (\sum j | 1 \leq j < i : b(j))) \Rightarrow (0 \leq i \leq 10 \wedge s = (\sum j | 1 \leq j < i : b(j)))$

$\Leftrightarrow < 0 \leq i \leq 10 \Leftrightarrow (0 = i \vee 1 \leq i \leq 10) >$

$((1 \leq i \leq 10 \wedge s = (\sum j | 1 \leq j < i : b(j))) \Rightarrow ((0 = i \vee 1 \leq i \leq 10) \wedge s = (\sum j | 1 \leq j < i : b(j)))$

$\Leftrightarrow < \wedge\textit{-distributivity; weakening/strengthening} >$

T

- $(P_{inv} \wedge \neg(G_0 \vee \cdots \vee G_{k-1})) \Rightarrow R$

$P_{inv} \wedge \neg G_0 \Rightarrow R$

$\Leftrightarrow < \textit{instantiate} >$

$((1 \leq i \leq 11 \wedge s = (\sum j | 1 \leq j < i : b(j)) \wedge \neg(i \leq 10))) \Rightarrow R)$

$\Leftrightarrow < \textit{algebra} >$

$(1 \leq i \leq 11 \wedge s = (\sum j | 1 \leq j < i : b(j)) \wedge i \geq 11) \Rightarrow R)$

$\Leftrightarrow < \textit{instantiate } R >$

$(1 \leq i \leq 11 \wedge s = (\sum j | 1 \leq j < i : b(j)) \wedge i \geq 11) \Rightarrow (s = (\sum j | 1 \leq j \leq 10 : b(j)))$

$\Leftrightarrow < i \leq 11 \wedge i \geq 11, \textit{ hence } i = 11, \textit{ substitute} >$

$(1 \leq i = 11 \wedge s = (\sum j | 1 \leq j < 11 : b(j))) \Rightarrow (s = (\sum j | 1 \leq j \leq 10 : b(j)))$

$\Leftrightarrow < \textit{algebra; weakening/strengthening} >$

T

Notice that in the last example we proved that **if** the loop completes, **then** we can conclude that it computes the correct value. But how do we know it completes? For this example we can informally reason

- Initially $i = 1$.

- Each time the loop body is executed, i is incremented.

- The loop only continues to execute as long as $i \leq 10$.

Hence, eventually the loop must terminate.

Let us formalize this. We define t, a function of some or all the variables encountered in the loop, to be the *bound function* for the loop if t satisfies

- $P_{inv} \wedge (G_0 \vee \cdots \vee G_{k-1}) \Rightarrow (t \geq 0)$; and

- $\{P_{inv} \wedge G_i\}\ t' := t; S_i\ \{t < t'\}$, for $i = 0, \ldots, k-1$.

The first condition says that the bound function is bounded below by zero. The second condition means that each time through the loop t decreases in value.

Bounding $t \geq 0$ is merely a choice. The lower bound could be any finite value.
Strictly speaking $t < t'$ is not enough: t must decrease every time through the iteration by at least a positive constant, since otherwise it could converge to zero without ever becoming less than zero.
The point is that the two conditions together force the number of iterations that are executed to be finite, which means the loop completes in a finite amount of time. Why? The bound function can't be bounded below and decreased infinitely often by at least a positive constant. Thus, the loop must terminate.

In Example 3.8, we can take $t = (10 - i)$. Then

- $P_{inv} \wedge (G_0 \vee \cdots \vee G_{k-1}) \Rightarrow (t \geq 0)$:

$$\begin{array}{l} \quad P_{inv} \wedge (i \leq 10) \Rightarrow ((10 - i) \geq 0) \\ \Leftrightarrow < \text{algebra; weakening/strengthening} > \\ \quad R \end{array}$$

Notice that we chose $t = 10 - i$ because then the guard $G_0 : i \leq 10$ can be rewritten as $10 - i \geq 0$.

- $\{P_{inv} \wedge G_i\} t' := t; S_i \{t < t'\}$:

$$
\begin{array}{l}
\quad \{P_{inv} \wedge G_0\} t' := t; S_0 \{t < t'\} \\
\Leftrightarrow \; < \text{definition} > \\
\quad (P_{inv} \wedge G_0) \Rightarrow wp(\text{“}t' := t; S_0\text{”}, t < t') \\
\Leftrightarrow \; < \text{Instantiate} > \\
\quad (P_{inv} \wedge G_0) \Rightarrow wp(\text{“}t' := 10 - i; s = s + b(i); i := i + 1\text{”}, 10 - i < t') \\
\Leftrightarrow \; < \text{definition of} := > \\
\quad (P_{inv} \wedge G_0) \Rightarrow wp(\text{“}t' := 10 - i; s = s + b(i)\text{”}, 10 - (i + 1) < t') \\
\Leftrightarrow \; < \text{definition of} := > \\
\quad (P_{inv} \wedge G_0) \Rightarrow wp(\text{“}t' := 10 - i\text{”}, 10 - (i + 1) < t') \\
\Leftrightarrow \; < \text{definition of} := > \\
\quad (P_{inv} \wedge G_0) \Rightarrow (10 - (i + 1) < 10 - i) \\
\Leftrightarrow \; < \text{algebra} > \\
\quad (P_{inv} \wedge G_0) \Rightarrow (-1 < 0) \\
\Leftrightarrow \; < \text{algebra} > \\
\quad (P_{inv} \wedge G_0) \Rightarrow T \\
\Leftrightarrow \; < (p \Rightarrow T) \Leftrightarrow T > \\
\quad T
\end{array}
$$

Notice that we never had to instantiate $P_{inv} \wedge G_0$. This is often the case and can save you a lot of writing.

The discussion so far leads us to the **checklist for a** `DO` **loop** in Figure 3.5.

We now return to the simpler loop

```
$S_I$
**while** $G$
    $S$
**endwhile**
```

where S_I denotes an initialization that happens before the loop commenses. Given an appropriate loop invariant P_{inv} we can now annotate the loop so that we can check its correctness:

Checklist for a DO **loop** of the form

```
$\{Q\}$
**do**
  $G_0 \rightarrow S_0$
$\Box$ $G_1 \rightarrow S_1$
      $\vdots$
$\Box$ $G_{k-1} \rightarrow S_{k-1}$
**od**
$\{R\}$
```

with loop invariant P_{inv} and bound function t:

- Show that P_{inv} holds before the loop execution begins.
- Show that $\{P_{inv} \wedge G_i\} S_i \{P_{inv}\}$ holds for $i = 0, \ldots, k-1$.
- Show that $P_{inv} \wedge \neg(G_0 \vee \cdots \vee G_{k-1}) \Rightarrow R$ holds.
- Show that $P_{inv} \wedge (G_0 \vee \cdots \vee G_{k-1}) \Rightarrow (t \geq 0)$.
- Show that $\{P_{inv} \wedge G_i\} t' := t; S_i \{t < t'\}$ holds for $i = 0, \ldots, k-1$.

We will annotate a typical loop like

```
$\{Q\}$
$\{P_{inv} :<$ loop invariant $>\}$
$\{t :<$ bound function $>\}$
**do**
  $G_0 \rightarrow S_0$
$\Box$ $G_1 \rightarrow S_1$
      $\vdots$
$\Box$ $G_{k-1} \rightarrow S_{k-1}$
**od**
$\{R\}$
```

Figure 3.5: Check list for DO command.

Theorem 3.9 (`While` **Theorem)**

```
$\{Q\}$
$S_I$
$\{P_{inv}\}$
**while** $G$
    $\{P_{inv} \wedge G\}$
    $S$
    $\{P_{inv}\}$
**endwhile**
$\{P_{inv} \wedge \neg G\}$
$\{R\}$
```

Partial correctness:

- $\{Q\}S_I\{P_{inv}\}$;
- $\{P_{inv} \wedge G\}S\{P_{inv}\}$; *and*
- $P_{inv} \wedge \neg G \Rightarrow R$

Complete correctness, in addition, for some bound function t:

- $P_{inv} \wedge G \Rightarrow t \geq 0$ *and*
- $\{P_{inv} \wedge G\}t' = t; S\{t < t'\}$

Example 3.10 *Consider the code segment that adds the elements in $a(1:n)$ to $b(1:n)$ storing the result in $c(1:n)$:*

```
$\{Q : 0 \leq n\}$
$i := 0$
$\{P_{inv} : 0 \leq i \leq n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j) + b(j))\}$
$\{t : n - i\}$
**while** $i < n$
    $i := i + 1$
    $c(i) := a(i) + b(i)$
**endwhile**
$R : (\forall j | 1 \leq j \leq n : c(j) = a(j) + b(j))$
```

Partial correctness:

- $\{Q\}S_I\{P_{inv}\}$:

$$
\begin{aligned}
& \{0 \leq n\} i := 0 \{0 \leq i \leq n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j) + b(j))\} \\
\Leftrightarrow\ & < \textit{definition of Hoare triple} > \\
& (0 \leq n) \Rightarrow wp(\text{“}i := 0\text{”}, 0 \leq i \leq n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j) + b(j))) \\
\Leftrightarrow\ & < \textit{definition of} := > \\
& (0 \leq n) \Rightarrow (0 \leq 0 \leq n \wedge (\forall j | 1 \leq j \leq 0 : c(j) = a(j) + b(j))) \\
\Leftrightarrow\ & < \textit{algebra; } \forall \textit{ over empty range} > \\
& (0 \leq n) \Rightarrow (0 \leq n \wedge T) \\
\Leftrightarrow\ & < \wedge\textit{-simplification; } p \Rightarrow p > \\
& T
\end{aligned}
$$

- $\{P_{inv} \wedge G\}S\{P_{inv}\}$:

$$P_{inv} \wedge G \Rightarrow wp(\text{“}i := i+1; c(i) := a(i)+b(i)\text{”}, 0 \leq i \leq n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)))$$
$$\Leftrightarrow < \textit{split range} >$$
$$P_{inv} \wedge G \Rightarrow wp(\text{“}i := i+1; c(i) := a(i)+b(i)\text{”},$$
$$0 \leq i \leq n \wedge (\forall j | 1 \leq j < i : c(j) = a(j)+b(j)) \wedge c(i) = a(i)+b(i))$$
$$\Leftrightarrow < \textit{definition of} := >$$
$$P_{inv} \wedge G \Rightarrow wp(\text{“}i := i+1\text{”},$$
$$0 \leq i \leq n \wedge (\forall j | 1 \leq j < i : c(j) = a(j)+b(j)) \wedge a(i)+b(i) = a(i)+b(i))$$
$$\Leftrightarrow < \textit{algebra; definition of} := >$$
$$P_{inv} \wedge G \Rightarrow$$
$$(0 \leq i+1 \leq n \wedge (\forall j | 1 \leq j < i+1 : c(j) = a(j)+b(j)) \wedge T)$$
$$\Leftrightarrow < \textit{algebra; instantiate; } \wedge\textit{-simplification} >$$
$$0 \leq i \leq n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)) \wedge i < n \Rightarrow$$
$$(-1 \leq i < n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)))$$
$$\Leftrightarrow < \textit{algebra} >$$
$$0 \leq i < n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)) \Rightarrow$$
$$((-1 = i < n \vee 0 \leq i < n) \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)))$$
$$\Leftrightarrow < \textit{weakening/strengthening} >$$
$$T$$

- $P_{inv} \wedge \neg G \Rightarrow R$:

$$0 \leq i \leq n \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)) \wedge \neg(i < n) \Rightarrow R$$
$$\Leftrightarrow < \wedge\textit{-simplification; algebra; commutitivity; associativity} >$$
$$0 \leq i \leq n \wedge (i \leq n \wedge i \geq n) \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)) \wedge \neg(i < n) \Rightarrow R$$
$$\Leftrightarrow < \textit{algebra} >$$
$$0 \leq i \leq n \wedge (i = n) \wedge (\forall j | 1 \leq j \leq i : c(j) = a(j)+b(j)) \wedge \neg(i < n) \Rightarrow R$$
$$\Leftrightarrow < \textit{substitute } i = n\textit{; instantiate } R >$$
$$0 \leq i \leq n \wedge (i = n) \wedge (\forall j | 1 \leq j \leq n : c(j) = a(j)+b(j)) \wedge \neg(i < n) \Rightarrow$$
$$(\forall j | 1 \leq j \leq n : c(j) = a(j)+b(j))$$
$$\Leftrightarrow < \textit{weakening/strengthening} >$$
$$T$$

Complete correctness, in addition, for some bound function t:

- $P_{inv} \wedge G \Rightarrow t \geq 0$

$$\begin{aligned}
& P_{inv} \wedge i < n \Rightarrow (n-i) \geq 0 \\
\Leftrightarrow & < \textit{algebra} > \\
& P_{inv} \wedge i < n \Rightarrow i \leq n \\
\Leftrightarrow & < \textit{algebra} > \\
& P_{inv} \wedge i < n \Rightarrow i < n \vee i = n \\
\Leftrightarrow & < \textit{weakening/strengthening} > \\
& T
\end{aligned}$$

Note: $t = (n-i-1)$ *would have made this proof slightly simpler.*

- $\{P_{inv} \wedge G\} t' := t; S\{t < t'\}$

$$\begin{aligned}
& P_{inv} \wedge G \Rightarrow wp(``t' := (n-i); i := i+1; c(i) := a(i)+b(i)", (n-i) < t') \\
\Leftrightarrow & < \textit{definition of} := > \\
& P_{inv} \wedge G \Rightarrow wp(``t' := (n-i); i := i+1", (n-i) < t') \\
\Leftrightarrow & < \textit{definition of} := > \\
& P_{inv} \wedge G \Rightarrow wp(``t' := (n-i)", (n-(i+1)) < t') \\
\Leftrightarrow & < \textit{definition of} := > \\
& P_{inv} \wedge G \Rightarrow ((n-(i+1)) < (n-i)) \\
\Leftrightarrow & < \textit{algebra} > \\
& P_{inv} \wedge G \Rightarrow -1 < 0 \\
\Leftrightarrow & < \textit{algebra} > \\
& P_{inv} \wedge G \Rightarrow T \\
\Leftrightarrow & < (p \Rightarrow T) \Leftrightarrow T > \\
& T
\end{aligned}$$

Homework 3.3.9.21 Consider the code segment that adds the elements in $a(1:n)$ to $b(1:n)$ storing the result in $c(1:n)$:

```
$\{0 \leq n\}$
$i := 1$
$\{P_{inv} : 1 \leq i \leq n+1 \wedge (\forall j | 1 \leq j < i : c(j) = a(j) + b(j))\}$
$\{t : n - i\}$
while $i \leq n$
    $c(i) := a(i) + b(i)$
    $i := i + 1$
endwhile
$(\forall j | 1 \leq j \leq n : c(j) = a(j) + b(j))$
```

☛ SEE ANSWER

Homework 3.3.9.22 Consider the following program that computes the quotient q and remainder r of the integer division of x by q:

```
$\{0 \leq x \wedge 0 \leq y\}$
$q := 0; r := x$
$\{P_{inv} : 0 < y \wedge 0 \leq r \wedge q * y + r = x\}$
$\{t : r\}$
while $r \geq y$
    $r := r - y$
    $q := q + 1$
endwhile
```

Give all *fully instantiated* conditions that must be proved *true* to establish partial correctness, **but do not prove them**.

☛ SEE ANSWER

Homework 3.3.9.23 Prove the partial correctness of the problem in the last homework.

☛ SEE ANSWER

Homework 3.3.9.24 Prove that the loop in the last two homeworks terminates.

☛ SEE ANSWER

3.4 Wrap Up

3.4.1 Additional Homeworks

Homework 3.4.1.25 Consider the loop

```
$\{1 \leq n\}$
$i := 1; s := 0$
$\{Q : 1 = i \leq n \wedge s = 0\}$
do
  $i \leq n \rightarrow s := s + i; i := i + 1$
od
$\{R : s = (\sum k | 1 \leq k \leq n : k)\}$
```

and loop invariant $P_{inv} : 1 \leq n \wedge 1 \leq i \leq n+1 \wedge s = (\sum k | 1 \leq k < i : k)$.

1. Prove partial correctness.
2. In addition, prove complete correctness.

☛ SEE ANSWER

Homework 3.4.1.26 Consider the loop

```
$\{T\}$
$i := 1; s := 0$
$\{Q : 0 < i \leq n \wedge s = 0\}$
do
  $i \leq n \rightarrow s := s + i; i := i + 1$
od
$\{R : s = n(n+1)/2\}$
```

and loop invariant $P_{inv} : 1 \leq i \leq n+1 \wedge s = (i-1)i/2$.

1. Prove partial correctness.
2. In addition, prove complete correctness.

☛ SEE ANSWER

- ______________

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

$\Leftrightarrow <$ $>$

Answers

Week 3: Review of Logic and Proofs

Homework 3.2.4.1 Use the Basic Equivalences to prove the following. (Do NOT use the weakening/strengthening laws given in Figure 3.3, which we will discuss later.)

1. $(b \wedge (b \Rightarrow c)) \Rightarrow c$.

   Answer: We will use an "equivalence style" of proof, in which we show that the proposition reduces to T via a sequences of equivalences:

   $$\begin{array}{ll} & (b \wedge (b \Rightarrow c)) \Rightarrow c \\ \Leftrightarrow & < \text{Implication} > \\ & (b \wedge (\neg b \vee c)) \Rightarrow c \\ \Leftrightarrow & < \text{Distribution} > \\ & ((b \wedge \neg b) \vee (b \wedge c)) \Rightarrow c \\ \Leftrightarrow & < \text{Contradiction} > \\ & (F \vee (b \wedge c)) \Rightarrow c \\ \Leftrightarrow & < \vee\text{-simplification} > \\ & (b \wedge c) \Rightarrow c \end{array}$$

   Now, at this point it would be nice to say "well, dah, of course b **and** c implies c." But we don't have that as a law. So we have to go on

   $$\begin{array}{ll} & (b \wedge c) \Rightarrow c \\ \Leftrightarrow & < \text{Implication} > \\ & \neg(b \wedge c) \vee c \\ \Leftrightarrow & < \text{De Morgan's} > \\ & (\neg b \vee \neg c) \vee c \\ \Leftrightarrow & < \text{Associativity} > \\ & \neg b \vee (\neg c \vee c) \\ \Leftrightarrow & < \text{Excluded middle} > \\ & \neg b \vee T \\ \Leftrightarrow & < \vee\text{-simplification} > \\ & T \end{array}$$

2. $p \wedge q \Rightarrow p$

   Answer: Now, we noticed that in the last proof there was a point at which we would have liked to have said "well, dah, of course b **and** c implies c." This exercise shows that particular insight in isolation. Again, we use an "equivalence style" of proof:

$p \wedge q \Rightarrow p$

$\Leftrightarrow <$ Implication $>$

$\neg(p \wedge q) \vee p$

$\Leftrightarrow <$ De Morgan's $>$

$(\neg p \vee \neg q) \vee p$

$\Leftrightarrow <$ Commutivity/Associativity $>$

$\neg q \vee (\neg p \vee p)$

$\Leftrightarrow <$ Excluded middle $>$

$\neg q \vee T$

$\Leftrightarrow <$ $\vee$-simplification $>$

T

3. $p \Rightarrow p \vee r$

Answer: Here is another one of those "well, dah" problems. Let's prove it:

$p \Rightarrow p \vee r$

$\Leftrightarrow <$ Implication $>$

$\neg p \vee (p \vee r)$

$\Leftrightarrow <$ Associativity $>$

$(\neg p \vee p) \vee r$

$\Leftrightarrow <$ Excluded middle $>$

$T \vee r$

$\Leftrightarrow <$ $\vee$-simplification $>$

T

4. $p \wedge q \Rightarrow p \vee r$ **Answer:** This one generalized the last two results.

$p \wedge q \Rightarrow p \vee r$

$\Leftrightarrow <$ Implication $>$

$\neg(p \wedge q) \vee (p \vee r)$

$\Leftrightarrow <$ De Morgan's $>$

$(\neg p \vee \neq q) \vee (p \vee r)$

$\Leftrightarrow <$ Commutivity/Associativity $>$

$\neg q \vee (\neg p \vee p) \vee r$

$\Leftrightarrow <$ Excluded middle $>$

$\neg q \vee T \vee r$

$\Leftrightarrow <$ $\vee$-simplification (twice) $>$

T

5. $((x \wedge y) \Rightarrow z) \Leftrightarrow (x \Rightarrow (y \Rightarrow z))$

   Answer: The strategy is to reduce both sides of the equivalence to the same expression

   $((x \wedge y) \Rightarrow z) \Leftrightarrow (x \Rightarrow (y \Rightarrow z))$

   $\Leftrightarrow <$ Implication, twice $>$

   $(\neg(x \wedge y) \vee z) \Leftrightarrow (\neg x \vee (y \Rightarrow z))$

   $\Leftrightarrow <$ De Morgan's, Implication $>$

   $(\neq x \vee \neg y \vee z) \Leftrightarrow (\neg x \vee \neg y \vee z)$

   $\Leftrightarrow <$ Identity $>$

   T

☛ BACK TO TEXT

Homework 3.2.4.2 In Figure 3.3 we present three Weakening/Strengening Laws. This exercise shows that if you only decide to remember one, it should be the last one.

1. Show that $(E1 \wedge E2) \Rightarrow E1$ is a special case of $(E1 \wedge E2) \Rightarrow (E1 \wedge E3)$.

   Answer: $(E1 \wedge E2) \Rightarrow (E1 \wedge E3)$ is *true* for all expressions E1, E2, and E3. If you choose $E3 = F$ then you get $(E1 \wedge E2) \Rightarrow E1$

2. Show that $E1 \Rightarrow (E1 \vee E3)$ is a special case of $(E1 \wedge E2) \Rightarrow (E1 \wedge E3)$.

   Answer: $(E1 \wedge E2) \Rightarrow (E1 \wedge E3)$ is *true* for all expressions E1, E2, and E3. If you choose $E2 = T$ then you get $E1 \Rightarrow (E1 \vee E3)$

☛ BACK TO TEXT

Homework 3.2.4.3 Prove the *counterpositive:* $(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$.

Answer: Start from the more complicated side: it is often easier to simplify.

$$\begin{array}{l} \quad \neg q \Rightarrow \neg p \\ \Leftrightarrow < \text{ implication } > \\ \quad \neg(\neg q) \vee \neg p \\ \Leftrightarrow < \text{ negation } > \\ \quad q \vee \neg p \\ \Leftrightarrow < \text{ commutativity } > \\ \quad \neg p \vee q \\ \Leftrightarrow < \text{ implication } > \\ \quad p \Rightarrow q \end{array}$$

☛ BACK TO TEXT

Homework 3.2.5.4 Let us consider a one dimensional array $b(1:n)$ (using Matlab notation), where $1 \leq n$. Let j and k be two integer variables satisfying $1 \leq j \leq k \leq n$. By $b(j:k)$ we mean the subarray of b consisting of $b(j), b(j+1), \ldots b(k)$. The segment $b(j:k)$ is empty if $j > k$.

Translate the following sentences into predicates.

1. All elements of $b(j:k)$ equal zero.

   Answer:

   $$(\forall i | j \leq i \leq k : b(i) = 0)$$

2. No value of $b(j:k)$ is zero.

   Answer:

   $$(\forall i | j \leq i \leq k : b(i) \neq 0)$$

   or

   $$\neq (\exists i | j \leq i \leq k : b(i) = 0)$$

   (You can use De Morgan's Law to show the equivalence of these statements.)

3. Some values of $b(j:k)$ are zero.

   Answer:

   $$\neq (\forall i | j \leq i \leq k : b(i) \neq 0)$$

   or

   $$(\exists i | j \leq i \leq k : b(i) = 0)$$

   (You can use De Morgan's Law to show the equivalence of these statements. Notice that here we use "at least one" to mean some. Is that really meant?)

4. All zeros in $b(1:n)$ are in subarray $b(j:k)$.

   Answer:
   $$(\forall i | 1 \leq i \leq n : b(i) = 0 \Rightarrow (j \leq i \leq k))$$
   or
   $$(\forall i | 1 \leq i < j \vee k < i \leq n : b(i) \neq 0)$$

5. Some zeros in $b(1:n)$ are in subarray $b(j:k)$.

   Answer:
   $$(\exists i | k \leq i \leq j : b(i) = 0)$$

6. Those values in $b(1:n)$ that are not in $b(j:k)$ are in $b(j:k)$.

   Answer:
   $$(\forall i | 1 \leq i \leq n : \neg(\exists m | j \leq m \leq k : b(i) = b(m)) \Rightarrow (\exists m | j \leq m \leq k : b(i) = b(m)))$$

   which, if you think about it, reduces to F *unless* $j:k$ is empty.

7. It is not the case that all zeros of $b(1:n)$ are in $b(j:k)$.
8. If $b(1:n)$ contains a zero, then so does $b(j:k)$.
9. It is not the case that all zeros of $b(1:n)$ are in $b(j:k)$.
10. Either $b(1:j)$ or $b(j:k)$ contains a zero (or both).
11. The values of $b(j:k)$ are in ascending order.
12. The segment $b(j:k)$ contains at least two zeros.
13. Every element of $b(1:j)$ is less than x and every value of $b(j+1:k)$ exceeds x.

☛ BACK TO TEXT

Homework 3.2.6.5 Formalize the following English specifications. Be sure to introduce necessary restrictions. Use $\min(x,y)$ and $\max(x,y)$ to denote the minimum and maximum of x and y, respectively.

1. Calculate the sum of elements $b(j:k)$.

   Answer: We will assume that the result of the sum is stored in variable α.
   $$\alpha = \left(\sum i | j \leq i \leq k : b(i)\right)$$
   What if $j:k$ is empty? This should return zero, which this properly captures.

2. Find the maximum value of $b(j:k)$.

   Answer: We will assume that the result of the max is stored in variable α.

$$(\exists i | j \leq i \leq k : \alpha = b(i)) \wedge (\forall i | j \leq i \leq k : b(i) \leq \alpha)$$

   What if $j:k$ is empty? Then the $\exists$ clause evaluates to F and hence the whole thing evaluates to false. This means that the program should not complete.

3. Find the index of a maximum value of $b(j:k)$.

   Answer: We will assume that the index of the value in $b(j:k)$ were the maximum is stored is computed in variable m.

$$(j \leq m \leq k) \wedge (\forall i | j \leq i \leq k : b(i) \leq b(m))$$

   What if $j:k$ is empty? Then the $(j \leq m \leq k)$ clause evaluates to F and hence the whole thing evaluates to false. This means that the program should not complete.

4. Store in array $c(1:n)$ a sorted (in ascending order) permutation of $b(1:n)$. Use the predicate $\text{perm}(b,c,n)$ to denote that $b(1:n)$ is a permutation of $c(1:n)$. Use the predicate $\text{ascending}(b,n)$ to indicate that the elements of $b(1:n)$ are in ascending order.

   Answer:

$$\text{perm}(b,c,n) \wedge (\forall i | 1 < i \leq n : c(i-1) \leq c(i))$$

   What if $1:n$ is empty? Then $\text{perm}(b,c,0)$ should evaluate to T (an empty array is a permutation of an empty array) and the $\forall$ is defined as T when the range is empty. This makes sense.

5. Calculate the greatest power of 2 that is not greater than n.

   Answer: We will assume that n is a positive integer and that the result is returned in α.

$$1 \leq n \wedge (\exists i | 0 \leq i : \alpha = 2^i) \wedge \alpha \leq n \wedge \neq (\exists i | 0 \leq i : \alpha < 2^i \leq n)$$

6. Count how many zeros $b(1:n)$ has.

   Answer: For this we introduce a new quantifier:

$$(\mathbb{N}\, i | i \in S : P(i))$$

   returns the number of elements in set S for which the predicate $P(i)$ is T.

$$\alpha = (\mathbb{N}\, i | 1 \leq i \leq n : b(i) = 0)$$

7. Suppose we have an array of integers $b(1:n)$. Each of its subsegments $b(i:j)$ has a sum. Find the largest such sum. Use the symbol $S_{i,j}$ for $\sum_{k=i}^{j} b(k)$.

   Answer:
   $$(\exists i,j|1 \leq i,j \leq n : \alpha = S_(i,j)) \wedge \neg(\exists i,j|1 \leq i,j \leq n : \alpha < S_(i,j))$$
   or
   $$(\exists i,j|1 \leq i,j \leq n : \alpha = S_(i,j)) \wedge (\forall i,j|1 \leq i,j \leq n : \alpha \geq S_(i,j))$$

8. Assume that array $b(1:n)$ is sorted. Find the highest index of an element in b that equals x. Be sure to take care of the case where x is not in array b.

   Answer: We will assume that m contains the desired index
   $$1 \leq m \leq n+1 \wedge (\forall i|m < i \leq n : b(m) < b(i))$$
   Now

☛ BACK TO TEXT

Homework 3.2.7.6 For each of the following, if applicable, determine which predicate is the weaker predicate:

1. $0 \leq x \leq 10$ and $1 \leq x < 5$.

   Answer: By examination, $0 \leq x \leq 10$ is weaker. Proof:
   $$\begin{aligned} & (1 \leq x < 5) \Rightarrow (0 \leq x \leq 10) \\ \Leftrightarrow & < \text{algebra} > \\ & (1 \leq x < 5) \Rightarrow (0 = x \vee 1 \leq x < 5 \vee 5 \leq x \leq 10) \\ \Leftrightarrow & < \text{weakening/strengening law} > \\ & T \end{aligned}$$

2. $x = 5 \wedge y = 4$ and $y = 4$.

   Answer: By examination, $y = 4$ is weaker. Proof:
   $$\begin{aligned} & (x = 5 \wedge y = 4) \Rightarrow (y = 4) \\ \Leftrightarrow & < \text{weakening/strengening law} > \\ & T \end{aligned}$$

3. $x \leq 5 \vee y = 3$ and $x = 5 \wedge y = 4$.

   Answer: By examination, $x \leq 5 \vee y = 3$ is weaker. Proof:

$$\begin{array}{ll} & (x = 5 \wedge y = 4) \Rightarrow (x \leq 5 \vee y = 3) \\ \Leftrightarrow & < \text{algebra} > \\ & (x = 5 \wedge y = 4) \Rightarrow (x < 5 \vee x = 5 \vee y = 3) \\ \Leftrightarrow & < \text{weakening/strengening law} > \\ & T \end{array}$$

4. T and F. **Answer:** By examination, T is weaker. Proof:

$$\begin{array}{ll} & F \Rightarrow T \\ \Leftrightarrow & < \wedge\text{-simplication} > \\ & F \wedge T \Rightarrow T \\ \Leftrightarrow & < \text{weakening/strengening law} > \\ & T \end{array}$$

5. $(\forall i | 5 \leq i \leq 10 : b(i+1) < b(i))$ and $(\forall i | 7 \leq i \leq 10 : b(i+1) < b(i))$ **Answer:** By examination, $(\forall i | 5 \leq i \leq 10 : b(i+1) < b(i))$ is weaker. Proof:

$$\begin{array}{ll} & (\forall i | 5 \leq i \leq 10 : b(i+1) < b(i)) \Rightarrow (\forall i | 7 \leq i \leq 10 : b(i+1) < b(i)) \\ \Leftrightarrow & < \text{split range} > \\ & (\forall i | 5 \leq i \leq 6 : b(i+1) < b(i)) \wedge (\forall i | 7 \leq i \leq 10 : b(i+1) < b(i)) \Rightarrow (\forall i | 7 \leq i \leq 10 : b(i+1) < b(i)) \\ \Leftrightarrow & < \text{weakening/strengening law} > \\ & T \end{array}$$

6. $x \leq 1$ and $x \geq 5$.

   Answer: Neither of these implies the other, so neither is weaker than the other.

☛ BACK TO TEXT

Homework 3.3.1.7 For each of the below code segments, determine the weakest precondition (by examination):

1. $wp(\text{“}y := x - 1\text{”}, y \leq 1) = (x \leq 2)$
2. $wp(\text{“}x := x - 1\text{”}, x \leq 1) = (x \leq 2)$

3. $wp(\text{"}\alpha := \alpha + b(i)\text{"}, \alpha = \sum_{k=0}^{i} b(k)) = (\alpha = \sum_{k=0}^{i} b(k))^{\alpha}_{\alpha+b(i)} = (\alpha + b(i) = \sum_{k=0}^{i} b(k)) = (\alpha = \sum_{k=0}^{i-1} b(k))$

☛ BACK TO TEXT

Homework 3.3.2.8 Prove that

$$((p \Rightarrow r) \wedge (q \Rightarrow r)) \Leftrightarrow ((p \vee q) \Rightarrow r)$$

Answer:

$$\begin{array}{ll} & (p \Rightarrow r) \wedge (q \Rightarrow r) \\ \Leftrightarrow & < \text{Implication} > \\ & (\neg p \vee r) \wedge (\neg q \vee r) \\ \Leftrightarrow & < \text{Distributivity of} > \\ & (\neg p \wedge \neg q) \vee r \\ \Leftrightarrow & < \text{De Morgan's} > \\ & \neg(p \vee q) \vee r \\ \Leftrightarrow & < \text{Implication} > \\ & p \vee q \Rightarrow r \end{array}$$

☛ BACK TO TEXT

Homework 3.3.3.9 Prove that **skip** command satisfies the Laws of Excluded Micacle, Distributivity of Conjunction, Monotonicity, and Distributed Disjunction.

Answer:

Excluded Miracle: Since $wp(\textbf{skip}, R) = R$ clearly $wp(\textbf{skip}, F) = F$.

Distributivity of Conjunction: $wp(\textbf{skip}, Q) \wedge wp(\textbf{skip}, R) = Q \wedge R = wp(\textbf{skip}, Q \wedge R)$.

Monotonicity: Assume $Q \Rightarrow R$. Since $wp(\textbf{skip}, Q) = Q$ and $wp(\textbf{skip}, R) = R$ clearly $wp(\textbf{skip}, Q) \Rightarrow wp(\textbf{skip}, R)$.

Distributivity of Disjunction: $wp(\textbf{skip}, Q) \vee wp(\textbf{skip}, R) = Q \vee R = wp(\textbf{skip}, Q \vee R)$. Hence $wp(\textbf{skip}, Q) \vee wp(\textbf{skip}, R) \Rightarrow wp(\textbf{skip}, Q \vee R)$

☛ BACK TO TEXT

Homework 3.3.4.10 Prove that **abort** command satisfies the Laws of Excluded Micacle, Distributivity of Conjunction, Monotonicity, and Distributed Disjunction.

Answer:

Excluded Miracle: Since $wp(\textbf{abort}, R) = F$ clearly $wp(\textbf{abort}, F) = F$.

Distributivity of Conjunction: $wp(\textbf{abort}, Q) \wedge wp(\textbf{abort}, R) = F \wedge F = F = wp(\textbf{abort}, Q \wedge R)$.

Monotonicity: Assume $Q \Rightarrow R$. Since $wp(\textbf{abort}, Q) = F$ clearly $wp(\textbf{abort}, Q) \Rightarrow wp(\textbf{abort}, R)$ (independent of the fact that $Q \Rightarrow R$ for this command).

Distributivity of Disjunction: $wp(\textbf{abort}, Q) \vee wp(\textbf{abort}, R) = F \vee F = F$. Hence $wp(\textbf{abort}, Q) \vee wp(\textbf{abort}, R) \Rightarrow wp(\textbf{abort}, Q \vee R)$.

☛ BACK TO TEXT

Homework 3.3.5.11 Prove that composition of statements satisfies the Laws of Excluded Micacle, Distributivity of Conjunction, Monotonicity, and Distributed Disjunction.

Answer:

Excluded Miracle: $wp(\text{``}S_0, S_1'', F) = wp(S_0, wp(S_1, F)) = wp(S_0, F) = F$.

Distributivity of Conjunction: $wp(\text{``}S_0, S_1'', Q) \wedge wp(\text{``}S_0, S_1'', R) = wp(S_0, wp(S_1, Q)) \wedge wp(S_0, wp(S_1, R)) = wp(S_0, wp(S_1, Q \wedge R)) = wp(\text{``}S_0, S_1'', Q \wedge R)$.

Monotonicity: Assume $Q \Rightarrow R$. Then $wp(\text{``}S_0, S_1'', Q) = wp(S_0, wp(S_1, Q)) \Rightarrow wp(S_0, wp(S_1, Q \Rightarrow R)) = wp(\text{``}S_0, S_1'', Q \Rightarrow R)$.

Distributivity of Disjunction: $wp(\text{``}S_0, S_1'', Q) \vee wp(\text{``}S_0, S_1'', R) = wp(S_0, wp(S_1, Q)) \vee wp(S_0, wp(S_1, R)) \Rightarrow wp(S_0, wp(S_1, Q) \vee wp(S_1, R)) \Rightarrow wp(S_0, wp(S_1, Q \vee R)) = wp(\text{``}S_0, S_1'', Q \vee R)$.

☛ BACK TO TEXT

Homework 3.3.6.12 For each of the below code segments, determine the weakest precondition:

1. $wp(i := i+1, i = j) == ((i+1) = j) = (i = j)$

2. $wp(i := i+1; j := j+i, i = j)$

   Answer:
   $$\begin{aligned} &= wp(\text{``}i := i+1\text{''}, wp(\text{``}j := j+i\text{''}, i = j)) \\ &= wp(\text{``}i := i+1\text{''}, i = j+i) \\ &= ((i+1) = (j+i)) = (1 = j) \end{aligned}$$

3. $wp(i := 2i+1; j := j+i, i = j)$

   Answer:
   $$\begin{aligned} &= wp(\text{``}i := 2i+1\text{''}, wp(\text{``}j := j+i\text{''}, i = j)) \\ &= wp(\text{``}i := 2i+1\text{''}, i = j+i) \\ &= wp(\text{``}i := 2i+1\text{''}, 0 = j) \\ &= (0 = j) \end{aligned}$$

4. $wp(j := j+i; i := 2i+1, i = j)$

Answer: $= wp(\text{“}j := j+i\text{”}, wp(\text{“}i := 2i+1\text{”}, i = j))$
$= wp(\text{“}j := j+i\text{”}, (2i+1) = (j+1))$
$= ((2i+1) = ((j+i)+1))$
$= (2i = (j+i)) = (i = j)$

5. $wp(\text{“}t := i; i := j; j := t\text{”}, i = \widehat{i} \wedge j = \widehat{j})$

Answer: $= wp(\text{“}t := i\text{”}, wp(\text{“}i := j\text{”}, wp(\text{“}j := t\text{”}, i = \widehat{i} \wedge j = \widehat{j})))$
$= wp(\text{“}t := i\text{”}, wp(\text{“}i := j\text{”}, i = \widehat{i} \wedge t = \widehat{j}))$
$= wp(\text{“}t := i\text{”}, j = \widehat{i} \wedge t = \widehat{j})$
$= (j = \widehat{i} \wedge i = \widehat{j})$

6. $wp(\text{“}i = 0; s := 0\text{”}, 0 \leq i < n \wedge s = (\sum j | 0 \leq j < i : b(j)))$

Answer: $= wp(\text{“}i = 0\text{”}, wp(\text{“}s := 0\text{”}, 0 \leq i < n \wedge s = (\sum j | 0 \leq j < i : b(j))))$
$= wp(\text{“}i = 0\text{”}, 0 \leq i < n \wedge 0 = (\sum j | 0 \leq j < i : b(j)))$
$= (0 \leq 0 < n \wedge 0 = (\sum j | 0 \leq j < 0 : b(j)))$
$= (T \wedge 0 = 0) = T$

7. $wp(\text{“}s := s + b(i); i = i+1\text{”}, 0 < i \leq n \wedge s = (\sum j | 0 \leq j < i : b(j))) =$

Answer: $= wp(\text{“}s := s+b(i)\text{”}, wp(\text{“}i = i+1\text{”}, 0 < i \leq n \wedge s = (\sum j | 0 \leq j < i : b(j))))$ (Note: I
$= wp(\text{“}s := s+b(i)\text{”}, 0 < i+1 \leq n \wedge s = (\sum j | 0 \leq j < i+1 : b(j)))$
$= (0 < i+1 \leq n \wedge s + b(i) = (\sum j | 0 \leq j < i+1 : b(j)))$
$= (0 < i < n \wedge s + b(i) = (\sum j | 0 \leq j < i : b(j))) + b(i)$
$= (0 < i < n \wedge s = (\sum j | 0 \leq j < i : b(j)))$
changed the problem slightly on the evening of Oct. 13.)

☛ BACK TO TEXT

Homework 3.3.6.13 Consider an array $b(1:n)$, $1 \leq n$, a scalar variable α, and the code segment

$\{P:\ 0 \leq k < n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))\}$
$S_0:\ k := k+1$
$S_1:\ \alpha := \alpha + b(k)$
$\{R:\ 0 \leq k \leq n \wedge \alpha = (\sum i | 1 \leq i \leq k : b(i))\}$

This code segment may be part of a program that sums the entries in array $b(i)$. Prove this code segment correct.

☛ BACK TO TEXT

Homework 3.3.7.14 Prove that

1. $(p \Rightarrow (q \wedge r)) \Leftrightarrow ((p \Rightarrow q) \wedge (p \Rightarrow r))$.

   Answer:

$$\begin{array}{ll} & (p \Rightarrow (q \wedge r)) \\ \Leftrightarrow & < \text{implication} > \\ & \neg p \vee (q \wedge r) \\ \Leftrightarrow & < \text{distributivity} > \\ & (\neg p \vee q) \wedge (\neg p \wedge r) \\ \Leftrightarrow & < \text{implication } \times 2 > \\ & (p \Rightarrow q) \wedge (p \Rightarrow r) \end{array}$$

2. $(p \Rightarrow (q \Rightarrow r)) \Leftrightarrow ((p \wedge q) \Rightarrow r)$

3. $(p \Rightarrow ((q_0 \vee q_1) \wedge (r_0 \Rightarrow s_0) \wedge (r_1 \Rightarrow s_1))) \Leftrightarrow ((p \Rightarrow (q_0 \vee q_1)) \wedge ((p \wedge r_0) \Rightarrow s_0) \wedge ((p \wedge r_1) \Rightarrow s_1))$
   (Hint: use 1. and 2.)

☛ BACK TO TEXT

Homework 3.3.7.15 The following code segment sets m to the maximum of x and y. Prove it correct.

```
{$T$}
if
   $x \geq y \rightarrow m := x$
 □ $x \leq y \rightarrow m := y$
endif
{$m = \max(x, y)$}
```

☛ BACK TO TEXT

Homework 3.3.7.16 Prove the following code segment correct:

```
{$(\forall j | 1 \leq j < i : m \leq b(j))$}
if
   $b(i) \geq m \rightarrow$ skip
 □ $b(i) \leq m \rightarrow m := b(i)$
endif
$i := i + 1$
{$\{(\forall j | 1 \leq j \leq i : m \leq b(j))\}$
```

Answer: First, let's bring it down to only having to prove the `IF` command correct:

```
$\{P : (\forall j | 1 \leq j < i : m \leq b(j))\}$
**if**
  $b(i) \geq m \;\; \rightarrow \;\;$ **skip**
$[\!]\; b(i) \leq m \;\; \rightarrow \;\; m := b(i)$
**endif**
$\{Q\}$
$i := i+1$
$\{R : (\forall j | 1 \leq j < i : m \leq b(j))\}$
```

$$\begin{aligned} Q &= wp(\text{“}i := i+1\text{”}, R) = R^{i}_{(i+1)} \\ &= (\forall j | 1 \leq j < i : m \leq b(j))^{i}_{(i+1)} = (\forall j | 1 \leq j < i+1 : m \leq b(j)) \end{aligned}$$

To check the correctness of this code segment, we check

- $P \Rightarrow G_0 \vee \cdots \vee G_{k-1}$:

  $P \Rightarrow (G_0 \vee G_1)$

  $\Leftrightarrow <$ instantiate $>$

  $P \Rightarrow (b(k) \geq m \vee b(k) \leq m)$

  $\Leftrightarrow <$ Excluded middle $>$

  $P \Rightarrow T$

  $\Leftrightarrow <$ definition of $\Rightarrow\, >$

  T

  (Notice: we did not instantiate P initially, and found out that it needed not be instantiated. This saves a lot of writing of long expressions.)

- $P \wedge G_0 \Rightarrow wp(S_0, Q)$:

$P \wedge G_0 \Rightarrow wp(S_0, Q)$

$\Leftrightarrow <$ instantiate $>$

$P \wedge G_0) \Rightarrow wp(\mathbf{skip}, (\forall j | 1 \leq j < i+1 : m \leq b(j)))$

$\Leftrightarrow <$ definition **skip** $>$

$P \wedge G_0 \Rightarrow (\forall j | 1 \leq j < i+1 : m \leq b(j))$

$\Leftrightarrow <$ instantiate $>$

$(\forall j | 1 \leq j < i : m \leq b(j)) \wedge (b(i) \geq m) \Rightarrow (\forall j | 1 \leq j < i+1 : m \leq b(j))$

$\Leftrightarrow <$ algebra; split range $>$

$(\forall j | 1 \leq j < i+1 : m \leq b(j)) \Rightarrow (\forall j | 1 \leq j < i+1 : m \leq b(j))$

$\Leftrightarrow <$ identity $>$

T

(Notice how we delayed instantiation. Also, the last two steps you may want to combine into ”split range; identity” to save writing.)

- $P \wedge G_1 \Rightarrow wp(S_1, Q)$:

$P \wedge G_1 \Rightarrow wp(S_1, Q)$

$\Leftrightarrow <$ instantiate $>$

$P \wedge G_1) \Rightarrow wp(\text{“}m := b(i)\text{”}, (\forall j | 1 \leq j < i+1 : m \leq b(j)))$

$\Leftrightarrow <$ definition $:=$ $>$

$P \wedge G_1 \Rightarrow (\forall j | 1 \leq j < i+1 : b(i) \leq b(j))$

$\Leftrightarrow <$ instantiate $>$

$(\forall j | 1 \leq j < i : m \leq b(j)) \wedge (b(i) \leq m) \Rightarrow (\forall j | 1 \leq j < i+1 : b(i) \leq b(j))$

$\Leftrightarrow <$ $\wedge$-simplification, transitivity; split range $>$

$(\forall j | 1 \leq j < i : b(i) \leq m \leq b(j)) \Rightarrow (\forall j | 1 \leq j < i : b(i) \leq b(j)) \wedge (b(i) \leq b(i))$

$\Leftrightarrow <$ identity; $\wedge$-simplification $>$

$(\forall j | 1 \leq j < i : b(i) \leq m \leq b(j)) \Rightarrow (\forall j | 1 \leq j < i : b(i) \leq b(j))$

$\Leftrightarrow <$ weakening/strengthening $>$

T

There is a ”trick” in this proof that is worth examining:

$(\forall j | 1 \leq j < i : m \leq b(j)) \wedge (b(i) \leq m)$

$\Leftrightarrow$ < definition of $\forall$ >

$((m \leq b(1)) \wedge \cdots (m \leq b(i-1)) \wedge (b(i) \leq m))$

$\Leftrightarrow$ < algebra; $\wedge$-simplification >

$((b(i) \leq m) \wedge (m \leq b(1)) \wedge \cdots (b(i) \leq m) \wedge (m \leq b(i-1)))$

$\Leftrightarrow$ < definition of $\forall$ >

$(\forall j | 1 \leq j < i : b(i) \leq m \leq b(j))$

☛ BACK TO TEXT

Homework 3.3.7.17 Prove the following code segment correct:

```
{$(\forall j|1 \leq j < i : m \geq b(j))$}
if
  $b(i) \leq m \rightarrow$ skip
▯ $b(i) \geq m \rightarrow m := b(i)$
endif
$i := i+1$
{{$(\forall j|1 \leq j < i : m \geq b(j))$}
```

☛ BACK TO TEXT

Homework 3.3.7.18 The following code segment might be part of a loop that computes the maximum value m in array $b(1:n)$ as well as the index i such that $b(i) = m$. Prove it correct.

```
{$P : (1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j|1 \leq j < k : b(j) \leq m)$}
if
  $b(k) \geq m \rightarrow m := b(k); i := k$
▯ $b(k) \leq m \rightarrow$ skip
endif
$k := k+1$
{$R : (1 \leq i < k \leq n) \wedge b(i) = m \wedge (\forall j|1 \leq j < k : b(j) \leq m)$}
```

Answer: First, let's bring it down to only having to prove the `IF` command correct:

```
{$P : (1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j|1 \leq j < k : b(j) \leq m)$}
if
  $b(k) \geq m \rightarrow m := b(k); i := k$
▯ $b(k) \leq m \rightarrow$ skip
endif
```

$Q : \{(1 \le i < k+1 \le n) \wedge b(i) = m \wedge (\forall j | 1 \le j < k+1 : b(j) \le m)\}$
$k := k+1$
$\{R : (1 \le i \le k \le n) \wedge b(i) = m \wedge (\forall j | 1 \le j < k : b(j) \le m)\}$

To check the correctness of this code segment, we check

- $P \Rightarrow G_0 \vee \cdots \vee G_{k-1}$:

$$\begin{aligned}
& P \Rightarrow (G_0 \vee G_1) \\
\Leftrightarrow & < \text{instantiate} > \\
& P \Rightarrow (b(k) \ge m \vee b(k) \le m) \\
\Leftrightarrow & < \text{Excluded middle} > \\
& P \Rightarrow T \\
\Leftrightarrow & < \text{definition of} \Rightarrow > \\
& T
\end{aligned}$$

(Notice: we did not instantiate P initially, and found out that it needed not be instantiated. This saves a lot of writing of long expressions.)

- $P \wedge G_0 \Rightarrow wp(S_0, Q)$:

$P \wedge G_0 \Rightarrow wp(S_0, Q)$

$\Leftrightarrow$ < instantiate >

$P \wedge (b(k) \geq m) \Rightarrow wp(\text{“}m := b(k)\text{”},$
$\quad wp(\text{“}i := k\text{”}, (1 \leq i < k+1 \leq n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k+1 : b(j) \leq m)))$

$\Leftrightarrow$ < definition of := >

$P \wedge (b(k) \geq m) \Rightarrow wp(\text{“}m := b(k)\text{”},$
$\quad (1 \leq k < k+1 \leq n) \wedge b(k) = m \wedge (\forall j | 1 \leq j < k+1 : b(j) \leq m)$

$\Leftrightarrow$ < definition of := >

$P \wedge (b(k) \geq m) \Rightarrow (1 \leq k < k+1 \leq n) \wedge b(k) = b(k) \wedge (\forall j | 1 \leq j < k+1 : b(j) \leq b(k))$

$\Leftrightarrow$ < algebra >

$P \wedge (b(k) \geq m) \Rightarrow (1 \leq k < n) \wedge T \wedge (\forall j | 1 \leq j \leq k : b(j) \leq b(k))$

$\Leftrightarrow$ < instantiate P; $\wedge$ simplification >

$(1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \geq m)$
$\quad \Rightarrow (1 \leq k < n) \wedge (\forall j | 1 \leq j \leq k : b(j) \leq b(k))$

$\Leftrightarrow$ < algebra; $\wedge$-simplification >

$(1 < k < n) \wedge (1 \leq i < k) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m \wedge m \leq b(k)) \wedge b(k) \leq b(k)$
$\quad \Rightarrow (1 \leq k < n) \wedge (\forall j | 1 \leq j \leq k : b(j) \leq b(k))$

$\Leftrightarrow$ < transitivity; split range >

$(1 < k < n) \wedge (1 \leq i < k) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq b(k))$
$\quad \Rightarrow (1 \leq k < n) \wedge (\forall j | 1 \leq j < k : b(j) \leq b(k)) \wedge b(k) \leq b(k)$

$\Leftrightarrow$ < algebra; $\wedge$-simplification >

$(1 < k < n) \wedge (1 \leq i < k) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq b(k))$
$\quad \Rightarrow (1 \leq k < n) \wedge (\forall j | 1 \leq j < k : b(j) \leq b(k))$

$\Leftrightarrow$ < weakening/strengthening >

T

- $P \wedge G_1 \Rightarrow wp(S_1, Q)$:

$P \wedge G_1 \Rightarrow wp(S_1, Q)$

$\Leftrightarrow <$ instantiate $>$

$P \wedge (b(k) \leq m) \Rightarrow wp(\mathbf{skip}, (1 \leq i < k+1 \leq n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k+1 : b(j) \leq m)))$

$\Leftrightarrow <$ definition of $wp(\mathbf{skip}, R)$ $>$

$P \wedge (b(k) \leq m) \Rightarrow (1 \leq i < k+1 \leq n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k+1 : b(j) \leq m)$

$\Leftrightarrow <$ algebra $>$

$P \wedge (b(k) \leq m) \Rightarrow (1 \leq i \leq k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j \leq k : b(j) \leq m)$

$\Leftrightarrow <$ instantiate P $>$

$(1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \leq m)$
$\quad \Rightarrow (1 \leq i \leq k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j \leq k : b(j) \leq m)$

$\Leftrightarrow <$ algebra; split range $>$

$(1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \leq m)$
$\quad \Rightarrow ((1 \leq i < k < n) \vee (i = k)) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \leq m)$

$\Leftrightarrow <$ $\wedge$-distributivity $>$

$(1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \leq m)$
$\quad \Rightarrow ((1 \leq i < k < n) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \leq m))$
$\qquad \vee ((i = k) \wedge b(i) = m \wedge (\forall j | 1 \leq j < k : b(j) \leq m) \wedge (b(k) \leq m))$

$\Leftrightarrow <$ weakening/strengthening $>$

T

☛ BACK TO TEXT

Homework 3.3.7.19 The following code segment might be part of a loop that computes number of zeroes in array $b(1:n)$. Prove it correct.

```
{$P : (0 \leq k < n) \wedge m = (\#i | 1 \leq i < k : b(i) = 0)$}
if
  $b(k) = 0 \rightarrow m := m+1; k := k+1$
  $\square\ b(k) \neq 0 \rightarrow k := k+1$
endif
{$Q : (0 \leq k \leq n) \wedge m = (\#i | 1 \leq i < k : b(i) = 0)$}
```

Here $(\#i | 1 \leq i < k : b(i) = 0)$ equals the number of occurrences of 0 in $b(1:k)$. You can alternatively replace it by $(\sum i | 1 \leq i < k \wedge b(i) = 0 : 1)$.

Answer: To check the correctness of this code segment, we check

- $P \Rightarrow G_0 \vee \cdots \vee G_{k-1}$:

$P \Rightarrow (G_0 \vee G_1)$

$\Leftrightarrow <$ instantiate $>$

$P \Rightarrow (b(k) = 0 \vee b(k) \neq 0)$

$\Leftrightarrow <$ Excluded middle $>$

$P \Rightarrow T$

$\Leftrightarrow <$ definition of $\Rightarrow$ $>$

T

- $P \wedge G_0 \Rightarrow wp(S_0, Q)$:

$P \wedge G_0 \Rightarrow wp(S_0, Q)$

$\Leftrightarrow <$ instantiate $>$

$P \wedge b(k) = 0$

$\Rightarrow wp(\text{“}m := m+1; k := k+1\text{”}, (0 \leq k \leq n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0))$

$\Leftrightarrow <$ definition of composition, $:=$ $>$

$P \wedge b(k) = 0 \Rightarrow (0 \leq k+1 \leq n) \wedge m+1 = (\mathbb{N} i | 1 \leq i < k+1 : b(i) = 0)$

$\Leftrightarrow <$ algebra $>$

$P \wedge b(k) = 0 \Rightarrow (0 \leq k < n) \wedge m+1 = (\mathbb{N} i | 1 \leq i < k+1 : b(i) = 0)$

$\Leftrightarrow <$ split range $>$

$P \wedge b(k) = 0$

$\Rightarrow (0 \leq k < n) \wedge m+1 = ((\mathbb{N} i | 1 \leq i < k : b(i) = 0) + (\mathbb{N} i | i = k : b(i) = 0))$

$\Leftrightarrow < (p \wedge q \Rightarrow r) \Leftrightarrow (p \wedge q \Rightarrow r \wedge q\ >$

$P \wedge b(k) = 0$

$\Rightarrow (0 \leq k < n) \wedge m+1 = ((\mathbb{N} i | 1 \leq i < k : b(i) = 0) + (\mathbb{N} i | i = k : b(i) = 0)) \wedge b(k) = 0$

$\Leftrightarrow <$ definition of $\mathbb{N}$ and $b(k) = 0 >$

$P \wedge b(k) = 0$

$\Rightarrow (0 \leq k < n) \wedge m+1 = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) + 1 \wedge b(k) = 0$

$\Leftrightarrow <$ algebra $>$

$P \wedge b(k) = 0 \Rightarrow (0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) \wedge b(k) = 0$

$\Leftrightarrow <$ instantiate $P >$

$(0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) \wedge b(k) = 0$

$\Rightarrow (0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) \wedge b(k) = 0$

$\Leftrightarrow <$ weakening/strengthening $>$

T

(Notice how long I resisted intantiating P!)

- $P \wedge G_1 \Rightarrow wp(S_1, Q)$:

$$P \wedge G_0 \Rightarrow wp(S_0, Q)$$
$\Leftrightarrow$ < instantiate >
$$P \wedge b(k) \neq 0 \Rightarrow wp(\text{“}k := k+1\text{”}, (0 \leq k \leq n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0))$$
$\Leftrightarrow$ < definition of := >
$$P \wedge b(k) \neq 0 \Rightarrow (0 \leq k+1 \leq n) \wedge m = (\mathbb{N} i | 1 \leq i < k+1 : b(i) = 0)$$
$\Leftrightarrow$ < algebra >
$$P \wedge b(k) \neq 0 \Rightarrow (0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k+1 : b(i) = 0)$$
$\Leftrightarrow$ < split range >
$$P \wedge b(k) \neq 0 \Rightarrow (0 \leq k < n) \wedge m = ((\mathbb{N} i | 1 \leq i < k : b(i) = 0) + (\mathbb{N} i | i = k : b(i) = 0))$$
$\Leftrightarrow$ < $(p \wedge q \Rightarrow r) \Leftrightarrow (p \wedge q \Rightarrow r \wedge q$ >
$$P \wedge b(k) \neq 0 \Rightarrow (0 \leq k < n) \wedge m = ((\mathbb{N} i | 1 \leq i < k : b(i) = 0) + (\mathbb{N} i | i = k : b(i) = 0)) \wedge b(k) \neq 0$$
$\Leftrightarrow$ < definition of $\mathbb{N}$ and $b(k) \neq 0$ >
$$P \wedge b(k) \neq 0 \Rightarrow (0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) \wedge b(k) \neq 0$$
$\Leftrightarrow$ < instantiate P >
$$(0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) \wedge b(k) \neq 0 \Rightarrow (0 \leq k < n) \wedge m = (\mathbb{N} i | 1 \leq i < k : b(i) = 0) \wedge b(k) \neq 0$$
$\Leftrightarrow$ < weakening/strengthening >
$$T$$

☛ BACK TO TEXT

Homework 3.3.8.20 Consider the following program that computes the quotient q and remainder r of the integer division of x by q:

```
{$0 \leq x \wedge 0 \leq y$}
$q := 0; r := x$
{$P_{inv} : 0 < y \wedge 0 \leq r \wedge q * y + r = x$}
{$t : r$}
while $r \geq y$
    $r := r - y$
    $q := q + 1$
endwhile
```

Give all *fully instantiated* conditions that must be proved *true* to establish partial correctness, **but do not prove them**.

Answer: Partial correctness:

- $\{Q\}S_I\{P_{inv}\}$:

$$(0 \leq x \wedge 0 < y) \Rightarrow wp(\text{“}q := 0; r := x\text{”}, 0 < y \wedge 0 \leq r \wedge q * y + r = x)$$

- $\{P_{inv} \wedge G\}S\{P_{inv}\}$:

$$(0 < y \wedge 0 \leq r \wedge q * y + r = x \wedge r \geq y) \Rightarrow wp(\text{“}r := r - y; q := q + 1\text{”}, 0 < y \wedge 0 \leq r \wedge q * y + r = x)$$

- $P_{inv} \wedge \neg G \Rightarrow R$:

$$(0 < y \wedge 0 \leq r \wedge q * y + r = x \wedge \neg(r \geq y)) \Rightarrow (0 \leq r < y \wedge q * y + r = x)$$

☛ BACK TO TEXT

Homework 3.3.8.21 Prove the partial correctness of the problem in the last homework.

Answer: Partial correctness:

- $\{Q\}S_I\{P_{inv}\}$:

$$\begin{aligned}
& Q \Rightarrow wp(\text{“}q := 0; r := x\text{”}, 0 < y \wedge 0 \leq r \wedge q * y + r = x) \\
\Leftrightarrow\ & < \text{definition of } := > \\
& Q \Rightarrow wp(\text{“}q := 0\text{”}, 0 < y \wedge 0 \leq x \wedge q * y + x = x) \\
\Leftrightarrow\ & < \text{definition of } := > \\
& Q \Rightarrow (0 < y \wedge 0 \leq x \wedge 0 * y + x = x) \\
\Leftrightarrow\ & < \text{instantiate } Q\text{; algebra} > \\
& (0 \leq x \wedge 0 < y) \Rightarrow (0 < y \wedge 0 \leq x \wedge T) \\
\Leftrightarrow\ & < \wedge\text{-simplification; } p \Rightarrow p > \\
& T
\end{aligned}$$

- $\{P_{inv} \wedge G\}S\{P_{inv}\}$:

$$
\begin{aligned}
& (P_{inv} \wedge G) \Rightarrow wp(\text{“}r := r-y; q := q+1\text{”}, 0 < y \wedge 0 \le r \wedge q*y+r = x) \\
\Leftrightarrow\ & < \text{definition of} := > \\
& (P_{inv} \wedge G) \Rightarrow wp(\text{“}r := r-y\text{”}, 0 < y \wedge 0 \le r \wedge (q+1)*y+r = x) \\
\Leftrightarrow\ & < \text{definition of} := > \\
& (P_{inv} \wedge G) \Rightarrow (0 < y \wedge 0 \le r \wedge (q+1)*y+(r-y) = x) \\
\Leftrightarrow\ & < \text{instantiate; algebra} > \\
& (0 < y \wedge 0 \le r \wedge q*y+r = x \wedge r \ge y) \Rightarrow (0 < y \wedge 0 \le r \wedge q*y+r = x) \\
\Leftrightarrow\ & < \text{weakening/strengthening} > \\
& T
\end{aligned}
$$

- $P_{inv} \wedge \neg G \Rightarrow R$:

$$
\begin{aligned}
& (0 < y \wedge 0 \le r \wedge q*y+r = x \wedge \neg(r \ge y)) \Rightarrow R \\
\Leftrightarrow\ & < \text{algebra; instantiate} > \\
& (0 < y \wedge 0 \le r \wedge q*y+r = x \wedge r < y)) \Rightarrow (0 \le r < y \wedge q*y+r = x) \\
\Leftrightarrow\ & < \text{algebra; instantiate} > \\
& (0 < y \wedge 0 \le r \wedge q*y+r = x \wedge r < y)) \Rightarrow (0 \le r < y \wedge q*y+r = x) \\
\Leftrightarrow\ & < \text{algebra} > \\
& (0 < y \wedge 0 \le r < y \wedge q*y+r = x)) \Rightarrow (0 \le r < y \wedge q*y+r = x) \\
\Leftrightarrow\ & < \wedge\text{-simplification} > \\
& T
\end{aligned}
$$

☛ BACK TO TEXT

Homework 3.3.8.22 Prove that the loop in the last two homeworks terminates.

Answer:

- $P_{inv} \wedge G \Rightarrow (t \ge 0)$:

$$
\begin{aligned}
& (0 < y \wedge 0 \le r \wedge r \ge y) \Rightarrow (r \ge 0) \\
\Leftrightarrow\ & < \text{algebra; weakening/strengthening} > \\
& T
\end{aligned}
$$

- $\{P_{inv} \wedge G\} t' := t; S \{t < t'\}$:

$P_{inv} \wedge G \Rightarrow wp(\text{"}t' := r; r := r - y; q := q + 1\text{"}, r < t')$

$\Leftrightarrow <$ definition of $:= >$

$P_{inv} \wedge G \Rightarrow wp(\text{"}t' := r; r := r - y\text{"}, r < t')$

$\Leftrightarrow <$ definition of $:= >$

$P_{inv} \wedge G \Rightarrow wp(\text{"}t' := r\text{"}, r - y < t')$

$\Leftrightarrow <$ definition of $:= >$

$0 < y \wedge 0 \leq r \wedge q * y + r = x \wedge G \Rightarrow r - y < r$

$\Leftrightarrow <$ algebra $>$

$0 < y \wedge 0 \leq r \wedge q * y + r = x \wedge G \Rightarrow 0 < y$

$\Leftrightarrow <$ weakening/strengthening $>$ $\quad T$

☛ BACK TO TEXT

Homework 3.3.8.23 Consider the code segment that adds the elements in $a(1:n)$ to $b(1:n)$ storing the result in $c(1:n)$:

```
$\{0 \leq n\}$
$i := 1$
$\{P_{inv} : 1 \leq i \leq n+1 \wedge (\forall j | 1 \leq j < i : c(j) = a(j) + b(j))\}$
$\{t : n - i\}$
while $i \leq n$
    $c(i) := a(i) + b(i)$
    $i := i + 1$
endwhile
$(\forall j | 1 \leq j \leq n : c(j) = a(j) + b(j))$
```

☛ BACK TO TEXT

Bibliography