Tutorial Letter 202/0/2020

Formal Program Verification

COS4892

Year module

School of Computing

This tutorial letter contains the solutions to Assignment 2.

Dear Student

Hope you are well. This tutorial letter contains the solutions to the second assignment for the year. Apologies for the late return.

Best wishes for your COS4892 studies.

Question 1

Prove the following theorems as stated in Barber:

- Theorem 3.0 on page 68
- Theorem 3.4 on page 73
- Theorem 3.5 on page 74
- Theorem 3.8 on page 77

Make use a diagram in your proof if appropriate. Refer to **Barber** for proof.

Question 2	4 x 3 = 12 marks

Use the proof rules of Chapters 9 and 10 in **RB** as well as selected portions in Chapter 3 of **Baber** to derive preconditions for the following post conditions and program statements. Show full workings.

2.1	{?} i := i + 1; k := k * 2; j := j – 1 {i = j}

Answer:

Consider: {P} i := i + 1{P1}; {P1}k := k * 2{P2}; {P2} j := j – 1 {i = j}

We progressively apply the complete assignment axiom and the sequential composition to progressively calculate P2, P1 and P.

P2: i = j-1 {i = j - 1} j:= j-1{i = j} because i = j it means if j := j-1 so too will i := j-1, as they are equal
P1: i = j-1 {i = j – 1} k := k * 2 {i = j – 1}
P: i = j-2 {i + 1 = j – 1} i:= i + 1 {i = j – 1}

2.2	{?} while (0 ≤ x ≤ 3) do y := y * x; x := x + 1 {x = 4 ∧ y = n!}

Answer:

We use the corollary of Section 3.3.2, p. 82 in **Baber**, except that we do not insist that the Ci's be complete preconditions. **[Q = Z_0 or Z_1 or Z_2 or . . .]**. Assume we have a statement of the form **while B do S end while** and some precondition P. Start with Z_0.

Z_0: Z_0 is the postcondition P, i.e. $x = 0$ and $y = 4$! in this case.
Therefore, Z_0 : $x = 4$ and $y = 1$!

Z_1: Z_1 is determined by finding first a precondition C_0 and then combining C_0 with the expression B.
$Z_1 = B$ and C_0

C_0 is found by substituting S with $x = 4$ and $y = 1$ in Z_0.
C_0: $\equiv y = 1 * 4$ and $x + 1 = 4$
$\equiv y = 4$ and $x = 3$

$Z_1 = B$ and C_0, i.e. is ($0 \leq x \leq 3$ and $y = 4$ and $x = 3$),
which simplifies to $y = 4$ and $x = 3$.
Z_1 : $y = 4$ and $x = 3$

Z_2. $Z_2 = B$ and C_1

C_1 is found by substituting S with $y = 4$ and $x = 3$ in Z_1.
C_1: $\equiv y = 4 * 3$ and $x + 1 = 3$
$\equiv y = 12$ and $x = 2$

$Z_2 = B$ and C_1, i.e. is ($0 \leq x \leq 3$ and $y = 12$ and $x = 2$),
which simplifies to $y = 12$ and $x = 2$.
Z_2 : $y = 12$ and $x = 2$

Z_3. $Z_3 = B$ and C_2

C_2 is found by substituting S with $y = 12$ and $x = 2$ in Z_2.
C_2: $\equiv y = 12 * 2$ and $x + 1 = 2$
$\equiv y = 24$ and $x = 1$

$Z_3 = B$ and C_2, i.e. is ($0 \leq x \leq 3$ and $y = 24$ and $x = 1$),
which simplifies to $y = 24$ and $x = 1$.
Z_3 : $y = 24$ and $x = 1$

Z_4. $Z_4 = B$ and C_3

C_3 is found by substituting S with $y = 24$ and $x = 1$ in Z_3.
C_3: $\equiv y = 24 * 1$ and $x + 1 = 1$
$\equiv y = 24$ and $x = 0$

$Z_4 = B$ and C_3, i.e. is ($0 \leq x \leq 3$ and $y = 24$ and $x = 0$),
which simplifies to $y = 24$ and $x = 0$.
Z_3 : $y = 24$ and $x = 0$

Z_5. $Z_5 = B$ and C_4

C_4 is found by substituting S with $y = 24$ and $x = 0$ in Z_4.
C_4: $\equiv y = 24 * 0$ and $x + 1 = 0$
$\equiv y = 0$ and $x = -1$

Z_5 = B and C_4, i.e. is ($0 \leq x \leq 3$ and $y = 24$ and $x = 0$), which is false, i.e. which picks out the empty subset of D, the set of all data environments. This tells us to stop; there is no point in trying to find Z_6 and so on, since they do not add any data environments to the precondition.

Therefore, the precondition for **while B do S endwhile** is, in this case, Z_0 or Z_1 or Z_2 or Z_3 or Z_4;

(x = 4) or (x = 3) or (x = 2) or (x = 1) or (x = 0)

Hence
$\{0 \leq x \leq 4\}$ while $(0 \leq x \leq 3)$ do $y := y * x; x := x + 1$ $\{x = 4 \wedge y = n!\}$

The precondition we are calculating is therefore $\{0 \leq x \leq 4\}$

2.3 **{?}**
if even(x) → x:= x - 1 **// x is an even integer**
□ odd(x) → z:= z + y x **// x is an odd integer**
fi
{x ≥ 0 ∧ z + y * x = a * b }

Answer:

Baber in Chapter 3, pages 73 - 76 gives the following formula for calculating a comprehensive precondition for an if statement:

$$P \equiv (P1 \wedge B) \vee (P2 \wedge \neg B)$$

where P1 and P2 are the individual preconditions for each branch of the guarded construct, B is the condition in the if-part of the construct and ¬B represents the condition for the else-part of the condition. Amongst other things, you must learn formula above and know it for exam purposes.

First, we calculate P1 and P2 in the usual way, using the Assignment Axiom:

So, $\{P1\}\ x := x - 1\ \{x \geq 0 \wedge z + y * x = a * b\}$

$\{P2\}\ z := z + y\ \{x \geq 0 \wedge z + y * x = a * b\}$

We may now apply the assignment axiom to determine to determine P1 and P2.

P1:

$\{?\}\ x := x - 1\ \{x \geq 0 \wedge z + y * x = a * b\}$
≡ {assignment axiom}
$\{x-1 \geq 0 \wedge z + y * (x - 1) = a * b\}$ $x := x - 1$ $\{x \geq 0 \wedge z + y * x = a * b\}$
≡ {simplification}
$\{x \geq 1 \wedge z + y * (x - 1) = a * b\}$ $x := x - 1$ $\{x \geq 0 \wedge z + y * x = a * b\}$

P1: x ≥ 1 ∧ z + y*(x-1) = a * b

P2:

$\{?\}\ z := z + y * x\ \{x \geq 0 \wedge z + y * x = a * b\}$

$\equiv$ {assignment axiom}

$\{ x \geq 0 \wedge (z + y * x) + y * x = a * b\}$ $\quad z := z + y * x\ \{x \geq 0 \wedge z + y * x = a * b\}$

$\equiv$ {addition}

$\{ x \geq 0 \wedge z + 2y*x = a * b \}$ $\quad z := z + y * x\ \{x \geq 0 \wedge z + y * x = a * b\}$

P2: $x \geq 0 \wedge z + 2y*x = a * b$

We can now combine the two preconditions P1 and P2 to obtain P.

$P \equiv (x \geq 1 \wedge z + y * (x - 1) = a * b \wedge \text{even}(x)) \vee (x \geq 0 \wedge z + 2y*x = a * b \wedge \text{odd}(x))$

2.4 {?} while $0 \geq x \geq -2$ do x:=x - 1 endwhile $\{x = -3\}$

Answer:

We use the corollary of Section 3.3.2, p. 82 in **Baber**, except that we do not insist that the Ci's be complete preconditions. **[Q = Z_0 or Z_1 or Z_2 or . . .]**. Assume we have a statement of the form **while B do S end while** and some precondition P. Start with Z_0.

Z_0: Z_0 is the postcondition P, i.e. $x = -3$ in this case.
Therefore, $Z_0 : x = -3$

Z_1: Z_1 is determined by finding first a precondition C_0 and then combining C_0 with the expression B.
$Z_1 = B$ and C_0

C_0 is found by substituting x - 1 in Z_0.
C_0: $\equiv x - 1 = -3$
$\equiv x = -2$

$Z_1 = B$ and C_0, i.e. is ($0 \geq x \geq -2$ and $x = -2$), which simplifies to $x = -2$.
$Z_1 : x = -2$

Z_2. $Z_2 = B$ and C_1

C_1 is found by substituting x - 1 in Z_1.
C_1: $\equiv x - 1 = -2$
$\equiv x = -1$

$Z_2 = B$ and C_1, i.e. is ($0 \geq x \geq -2$ and $x = -1$), which simplifies to $x = -1$.
$Z_2 : x = -1$

Z_3. $Z_3 = B$ and C_2

C_2 is found by substituting x - 1 in Z_2.
C_2: $\equiv x - 1 = -1$
$\equiv x = 0$

Z_3 = B and C_2, i.e. is ($0 \geq x \geq -2$ and $x = 0$), which simplifies to $x = 0$.
$Z_3 : x = 0$

Z_4. Z_4 = B and C_3

C_3 is found by substituting x - 1 in Z_3.
C_3: $\equiv x - 1 = 0$
$\equiv x = 1$

Z_4 = B and C_3, i.e. is ($0 \geq x \geq -2$ and $x = 1$), which is false, i.e. which picks out the empty subset of D, the set of all data environments. This tells us to stop; there is no point in trying to find Z_5 and so on, since they do not add any data environments to the precondition.

Therefore, the precondition for **while B do S endwhile** is, in this case, Z_0 or Z_1 or Z_2 or Z_3;

$(x = -3)$ or $(x = -2)$ or $(x = -1)$ or $(x = 0)$

Hence
$\{-3 \leq x \leq 0\}$ while $0 \geq x \geq -2$ do x := x - 1 endwhile $\{x = -3\}$

The precondition we are calculating is therefore

$\{-3 \leq x \leq 0\}$ while $0 \geq x \geq -2$ do x:=x - 1 endwhile $\{x = -3\}$

Question 3	2 marks

Explain how to verify a Hoare triple {P} S {Q} with precondition P, program S and post condition Q.

Answer:

A twofold process may be used to verify a Hoare triple:

(a) Calculate a precondition P' for the given postcondition Q and the program S.
(b) Show that the given precondition implies the calculated one; i.e. $P \Rightarrow P'$.

Question 4	4 x 3 = 12 marks

Verify the following Hoare triples:

4.1 $\{x = y\}$ if $(x = 0)$ then $x := y + 1$ else $z := y + 1$ $\{(x = y + 1) \vee (z = x + 1)\}$

Answer:

We may re-write the if statement into its guarded-command equivalent form:

$\{x = y\}$
if $x = 0 \rightarrow x := y + 1$
$\square\ x \neq 0 \rightarrow z := y + 1$
fi
$\{(x = y + 1) \vee (z = x + 1)\}$

Let P1 and P2 be the preconditions for each branch of the guarded construct. As mentioned earlier, the precondition P' of the if statement is given by the formula:

$P' = (P1 \wedge B) \vee (P2 \wedge \neg B)$

We apply the assignment axiom to calculate P1 and P2.

$\{P1\}\ x := y + 1\ \{(x = y + 1) \vee (z = x + 1)\}$

P1 $\equiv$
$(y + 1 = y + 1) \vee ((z = y + 1) + 1)$
$\equiv$
$\text{true} \vee z = y + 2$
$\equiv$
$z = y + 2$

$\{P2\}\ z := y + 1\ \{(x = y + 1) \vee (z = x + 1)\}$

P2 $\equiv$
$(x = y + 1) \vee (y + 1 = x + 1)$
$\equiv$
$x = y + 1 \vee (y + 1 = x + 1)$
$\equiv$
$x = y + 1 \vee x = y$
$\equiv$
$x = y + 1 \vee x = y$
$\equiv$
$x = y + 1$

Now $P' \equiv (z = y + 2 \wedge x = 0) \vee (x = y + 1 \wedge x \neq 0)$

To finalise, we now check if P implies P'.

P $\equiv$
$x = y$
$\equiv$
$(z = 2) \vee (x = x + 1 \wedge x \neq 0)$

It is clear that $(\forall x)\ x \in P \Rightarrow x \in P'$ that is, $P \Rightarrow P'$

4.2 $\{x = y\}$ if $(x = 0)$ then $x := y + 1$ else $z := y + 1$ $\{(z = 1) \rightarrow (x = 1)\}$

Answer:

We first re-write the if statement into its guarded-command equivalent form:

$\{x = y\}$
if $x = 0 \rightarrow x := y + 1$
$\square\ x \neq 0 \rightarrow z := y + 1$
fi
$\{(z = 1) \Longrightarrow (x = 1)\}$

Let P1 and P2 be the preconditions for each branch of the guarded construct. As mentioned earlier, the precondition P' of the if statement is given by the formula:

$P' = (P1 \wedge B) \vee (P2 \wedge \neg B)$

We apply the assignment axiom to calculate P1 and P2.

$\{P1\}\ x := y + 1\ \{(z = 1) \Longrightarrow (x = 1)\}$

P1 $\equiv$
$(z = 1) \Longrightarrow (y + 1 = 1)$
$\equiv$
$(z = 1) \Longrightarrow (y = 0)$

$\{P2\}\ z := y + 1\ \{(z = 1) \Longrightarrow (x = 1)\}$

P2 $\equiv$
$(y + 1 = 1) \Longrightarrow (x = 1)$
$\equiv$
$(y = 0) \Longrightarrow (x = 1)$

Now $P' \equiv ((z = 1) \Longrightarrow (y = 0) \wedge (x = 0)) \vee ((y = 0) \Longrightarrow (x = 1)) \wedge (x \neq 0))$

To finalise, we now check if P implies P'.

Remember the truth table of $\Longrightarrow$ (**RB** on page 56)

p	q	$p \Longrightarrow q$
true	true	true
true	false	false
false	true	true
false	false	true

First consider where $x = 0$, i.e. $(z = 1) \Longrightarrow (y = 0) \wedge (x = 0)$

If $(x = y)$, then $(z = 1)$ $\Longrightarrow$ $(x = 0) \wedge (x = 0)$
true $\Longrightarrow$ true
true

Therefore P => P'

Now consider where $x \neq 0$, i.e. $(y = 0) \Longrightarrow (x = 1) \wedge (x \neq 0)$

If $x = y$, then $(x = 0) \quad \Longrightarrow \quad (x = 1) \wedge (x \neq 0)$
$\text{false} \quad \Longrightarrow \quad \text{true}$
true
Therefore $P \Rightarrow P'$

It is clear that $(\forall x)\ x \in P \Rightarrow x \in P'$ that is, $P \Rightarrow P'$

4.3 $\{3 \leq |x| \leq 4\}$ if $x < 0$ then $y := -x$ else $y := x$ endif $\{2 \leq y \leq 4\}$

Answer:

We may rewrite the if statement into its guarded-command equivalent form:

$\{3 \leq |x| \leq 4\}$
if $x < 0 \rightarrow y := -x$
$\square\ x > 0 \rightarrow y := x$
fi
$\{2 \leq y \leq 4\}$

Let P1 and P2 are the preconditions for each branch of the guarded construct. As mentioned earlier, the precondition P' of the if statement is given by the formula:

$P' = (P1 \wedge B) \vee (P2 \wedge \neg B)$

We apply the assignment axiom to calculate P1 and P2.

$\{P1\}\ y := -x \quad \{2 \leq y \leq 4\}$

$P1 \equiv 2 \leq -x \leq 4$
$\equiv -4 \leq x \leq -2$

$\{P2\}\ y := x \quad \{2 \leq y \leq 4\}$

$P2 \equiv 2 \leq x \leq 4$

Now $P' \equiv (-4 \leq x \leq -2 \wedge x < 0) \vee (2 \leq x \leq 4 \wedge x \geq 0)$
$\equiv (-4 \leq x \leq -2) \vee (2 \leq x \leq 4)$.

To finalise, we now check if P implies P'.

$P \equiv 3 \leq |x| \leq 4$
$\equiv -4 \leq -x \leq -3 \vee 3 \leq x \leq 4$

It is clear that $(\forall x)\ x \in P \Rightarrow x \in P'$ that is, $P \Rightarrow P'$

Question 5	2 marks

Name the two steps necessary to prove that an arbitrary property *P* of natural numbers is true for all natural numbers *n;* i.e. what is the principle of simple mathematical induction?

Answer:

RB pages 172 and 173 (example 12.8)

An arbitrary property *P* of natural numbers is provably true for all natural numbers *n*, if it is possible to prove

(i) *P.0* is true, and
(ii) For all *n*, *P.(n+ 1)* follows from the assumption that *P.n* is true

Before doing question 6 to 8, read the following:

From the theory developed on pages 184 (bottom) - 186 of **RB** and summarised by **Baber** on p 87 *the correctness of a loop* can be proved by following five (5) steps below. Take note that **RB** refer to the body of the loop as *T*, and **Baber** as *S*.

Step 1: Determine the loop invariant (*inv*) by inspection.

Step 2: Prove that the initialisation establishes the truth of the loop invariant.

Step 3: Prove that the body of the loop preserves the truth of the loop invariant; i.e. prove that the truth of the loop invariant and the truth of the loop condition (i.e. $\neg$*done*) before execution of the loop body together imply the truth of the loop invariant after execution of body loop.

Step 4: Prove that the truth of the loop invariant and the termination condition which is the negation of the loop condition (i.e. $\neg\neg$*done* $\equiv$ *done*) together imply the correctness of the final result.

Step 5: Prove that the termination condition (i.e. *done*) will be fulfilled in a finite number of executions of the loop body, i.e. that the loop will terminate in finite time.

Question 6	10 marks

Verify {P} S {Q} where S is the following subprogram which searches an array A for the value of a variable x:

k := 1;
while (k ≤ n ∧ A(k) ≠ x) **do** k:=k+1 **end while**

The precondition P is {n ∈ Z ∧ 0 ≤ n} where Z is the set of integers and where n is intended to indicate the number of entries in the linear array A.

The post condition Q is:

n ∈ Z ∧ k ∈ Z	
and 1 ≤ k ≤ n+1	(this gives the range of k)
and $\overset{k-1}{i=1}$A(i) ≠ x	(so all entries before the k-th ≠ x)
and k ≤ n **and** A(k) = x	(i.e. either A(k) = x)
or k = n+1	(or no entry in A equals x).

In addition, the programmer has specified the following loop invariant:

n ∈ Z **and** k ∈ Z **and** 1 ≤ k ≤ n+1 (k must lie in its range)

and $i = 1^{k-1} A(i) \neq x$ (so we haven't previously found x).

Hint: Ignore the bound function *bf* when verifying Steps 1 – 4 above. Use the bound function for verifying Step 5. You may also rewrite the loop as a guarded construct first.

Answer:

To prove the correctness of the loop construct we follow steps 1 – 5 outlined on page 13 of your first tutorial letter.

Step 1: Determine the loop invariant, called *inv* in the question. This is easy since it is given:

$n \in Z$ **and** $k \in Z$ **and**
$1 \leq k \leq (n + 1)$ **and**
$(\forall i) 1 \leq i \leq k - 1 \Rightarrow A(i) \neq x$ [This says that x is not in any of A(1), A(2), ..., A(k-1)]

Step 2: Prove that the initialization, k := 1, establishes the truth of the loop invariant. We show this by substituting for k in *inv*:

$n \in Z$ **and** (2.1)
$1 \in Z$ **and** (2.2)
$1 \leq 1 \leq (n + 1)$ **and** (2.3)
$(\forall i) 1 \leq i \leq 0 \Rightarrow A(i) \neq x$ (2.4)

- It's easy to see that assertions (2.1) and (2.2) hold.
- Assertion (2.3) holds because $n \geq 0$.
- Assertion (2.4) is true since the antecedent of the implication is false ($1 \leq i \leq 0$). See truth table

p	q	$p \Longrightarrow q$
true	true	true
true	false	false
false	true	true
false	false	true

Therefore, the initialization establishes the truth of the loop invariant *inv*.

Since this question is stated in the same format as Question 2 above (i.e. a precondition P is given), we can at this point derive a new precondition, say P', and then show that our derived precondition is implied by the given precondition, P. The derived precondition is determined by calculating the value of *inv*, i.e.:

(2.1) **and** (2.2) **and** (2.3) **and** (2.4)
$\equiv n \in Z$ **and** $1 \in Z$ **and** $1 \leq 1 \leq (n + 1)$ **and** $(\forall i)(1 \leq i \leq 0 \Rightarrow A(i) \neq x)$
$\equiv n \in Z$ **and true and true and** $1 \leq (n + 1)$ **and true**
$\equiv n \in Z$ **and** $1 \leq (n + 1)$
$\equiv n \in Z$ **and** $0 \leq n$ (= P')

So, we see that our derived precondition, P' is the same as the given precondition, P.

Step 3: Prove that the body of the loop namely k := k + 1 (i.e. our *T*) preserves *inv*, i.e. show that (*inv* and ¬*done*) before an execution of *T* implies *inv* after executing *T*. We can do this in either of 2 ways:

a) Show that { *inv* and ¬*done* } k := k + 1 {*inv*}, i.e. substitute (k + 1) for k in *inv* to obtain *inv'* (say) and then show that (*inv* and ¬*done*) ⇒ *inv'*, or

b) Assume (*inv* and ¬*done*, substitute (k + 1) for k in *inv* and show that inv is preserved.

Let's follow approach (b) above. Substituting (k+1) for k in *inv* gives:

n ∈ Z **and**	(3.1)
k + 1 ∈ Z **and**	(3.2)
1 ≤ (k + 1) ≤ (n + 1) **and**	(3.3)
(∀i)1 ≤ i ≤ k ⇒ A(i) ≠ x	(3.4)

- As before assertion (3.1) holds and assertion (3.2) holds because the successor of any integer is (again) an integer.
- Now what about assertions (3.3) and (3.4)?
- Assertion (3.3)
  We have: ¬*done* ≡ (k ≤ n **and** A(k) ≠ x). Because k ≤ n it follows that k + 1 ≤ n + 1. Hence assertion (3.3) holds.
- Assertion (3.4)
  We know from *inv* before the execution of S that (∀i)(1 ≤ i ≤ k - 1 ⇒ A(i) ≠ x) and since the 2nd conjunct in ¬*done* states that A(k) ≠ x it follows that (∀i)(1 ≤ i ≤ k ⇒ A(i) ≠ x). Therefore assertion (3.4) holds.

It follows that the body of the loop namely k := k + 1 preserves *inv*.

Step 4: Prove that the truth of the loop invariant (i.e. *inv*) and the termination condition (i.e. *done*) implies the final result (i.e. the postcondition Q).

Therefore we must prove: [*inv* and *done*] ⇒ Q. (4.1)

Substituting for *inv*, done and Q in (4.1) gives us (the parts in bold after the implication are those that do not follow immediately from the antecedent and, therefore, need to be proved):

n ∈ Z **and** k ∈ Z **and** 1 ≤ k ≤ (n + 1) **and** (∀i)(1 ≤ i ≤ k - 1 ⇒ A(i) ≠ x) **and**
[k > n **or** A(k) = x]

⇒

n ∈ Z **and** k ∈ Z **and** 1 ≤ k ≤ (n + 1) **and** (∀i)(1 ≤ i ≤ k - 1 ⇒ A(i) ≠ x) **and**
(**[k ≤ n and A(k) = x] or k = n + 1**) (4.2)

The only non-trivial parts to prove in (4.2) is **[k ≤ n and A(k) = x] or k = n + 1**. The last conjunct in the antecedent of (4.2) contains the disjunct: k > n **or** A(k) = x. Suppose k > n. From a conjunct in the antecedent we also have: 1 ≤ k ≤ (n + 1). Hence from k > n **and** 1 ≤ k ≤ (n + 1) we have **k = n + 1** which proves the 2nd part of the disjunct in the last conjunct in the antecedent of (4.2).

Next we consider the part **[k ≤ n and A(k) = x]** in the consequent of (4.2). This part is proved by noting that it applies to the case when the entry x is indeed present in the array A, i.e. the opposite of k > n which boils down to [k ≤ n **and** A(k) = x] and which is the part we wanted to prove.

Hence (4.2) holds, i.e. the truth of the loop invariant and the termination condition implies the postcondition.

Step 5: Prove that the loop will terminate in finite time. This is shown by identifying a bound function, bf. A suitable bound function is: *bf* = n + 1 - k. Now we must show that (see **RB** page 185)

a) *bf* is decreased by every iteration of the loop, and
b) The number of times the loop is executed is at most the initial value of bf.

Now a) above holds, since every iteration of the loop increases k by one and, therefore, n + 1 - k decreases by 1 every time; b) also holds since the maximum number of times the loop is executed is n, which is equal to the initial value of bf, i.e. n + 1 - 1.

Another argument not involving *bf* could be:

A. The integer k starts at 1.
B. n ∈ Z is a finite number since it is the size (assumed finite) of the array A.
C. Every iteration increments k by 1.

So, if n ≥ 1 then k will reach n in a finite number of steps, i.e. the loop will terminate. Else if n < 1 (i.e. n = 0) the loop is never entered, i.e. the 'loop will terminate'.

Note that we must also make sure that all relevant variables x, i, k, n, etc. are indeed defined in the set of variables defined for our program (let's call this set D) before we can claim that each execution of the loop body is defined, since for k starting at 1 we have k ∈ *D*, the successor of any integer is again an integer and we have k + 1 ∈ *D* for each k ∈ *D*.

We must also make sure that the result of the boolean condition (k ≤ n **and** A(k) ≠ x) is defined and is a valid test. The question is, however, silent about the status of these variables. But normally one would assume that all relevant variables are appropriately defined and that all tests yield valid results, etc.

Question 7	4 marks

Given an array of integers, specify formally and develop an algorithm that will count the number of zeroes in the array.

Answer:

{0 < n}

k, c := 0, 0;

do k < n

if A(k) = 0 → c := c + 1 fi k := k + 1

od

$\{c = \langle \sum i : 0 \leq i < n \wedge A(i) = 0 : 1 \rangle\}$

Question 8	8 marks

Prove that, for all natural numbers n,

$$\langle \sum k : 0 \leq k < n : 2^k \rangle = 2^n - 1$$

Answer:

There are two steps in the proof, the basis and the induction step. To make the process clear, let us give the predicate a name say E. That is, by definition,

$$E.n = \langle \sum k : 0 \leq k < n : 2^k \rangle = 2^n - 1$$

Then the proof proceeds as follows:
Basis.

$E.0$
$=$ {definition}
$\langle \sum k : 0 \leq k < 0 : 2^k \rangle = 2^0 - 1$
$=$ {empty-range rule to simplify the summation, arithmetic for the right side of the equality}
$0 = 1 - 1$
$=$ {arithmetic and reflexivity of equality}
true .

Induction step.

$E.(n + 1)$
$=$ {definition}
$\langle \sum k : 0 \leq k < n + 1 : 2^k \rangle = 2^{n+1} - 1$
$=$ {range splitting applied to the summation}
$\langle \sum k : 0 \leq k < n : 2^k \rangle + 2^n = 2^{n+1} - 1$
$=$ {assume E.n, that is, assume that $\langle \sum k : 0 \leq k < n : 2^k \rangle = 2^n - 1$}
$(2^n - 1) + 2^n = 2^{n+1} - 1$
$=$ {property of exponentiations and simple arithmetic}

true .