SEMESTER TWO EXAMINATIONS 2010

MODULE: CA648 Formal Programming

COURSE: M.Sc. in Software Engineering

YEAR: 1

EXAMINERS:
Dr. A. Butterfield,
Mr. D. Brady,
Dr. G. Hamilton, Ext no. 5017.

TIME ALLOWED: 3 hours

INSTRUCTIONS:
Please answer all questions.
All questions carry equal marks.

Please do not turn over this page until instructed to do so

The use of programmable or text storing calculators is expressly forbidden.
QUESTION 1  

Consider the following partial correctness specification:

\[
\begin{align*}
\{ X \geq 0 \} \\
N := 0; \\
PRODUCT := 0; \\
WHILE \ N < X \ DO \\
\quad BEGIN \\
\quad \quad PRODUCT := PRODUCT + Y; \\
\quad \quad N := N + 1 \\
\quad END \\
\{ PRODUCT = X \times Y \}
\end{align*}
\]

1(a)  [6 Marks]

Add appropriate annotations to this specification to allow it to be verified.

**Solution:**

This specification can be annotated as follows:

\[
\begin{align*}
\{ X \geq 0 \} \\
N := 0; \\
PRODUCT := 0; \ \{ X \geq 0 \land N = 0 \land PRODUCT = 0 \} \\
WHILE \ N < X \ DO \ \{ PRODUCT = N \times Y \land N \leq X \} \\
\quad BEGIN \\
\quad \quad PRODUCT := PRODUCT + Y; \\
\quad \quad N := N + 1 \\
\quad END \\
\{ PRODUCT = X \times Y \}
\end{align*}
\]

1(b)  [6 Marks]

List the verification conditions which would be generated for the annotated specification in 1(a).

**Solution:**

The following verification conditions will be generated from this specification:

1. \( X \geq 0 \Rightarrow X \geq 0 \land 0 = 0 \land 0 = 0 \)
2. \( X \geq 0 \land N = 0 \land PRODUCT = 0 \Rightarrow PRODUCT = N \times Y \land N \leq X \)
3. \( PRODUCT = N \times Y \land N \leq X \land \neg (N < X) \Rightarrow PRODUCT = X \times Y \)
4. \( PRODUCT = N \times Y \land N \leq X \land N < X \Rightarrow PRODUCT + Y = (N + 1) \times Y \land (N + 1) \leq X \)

1(c)  [8 Marks]

Verify this specification by showing that the verification conditions in 1(b) are true.

**Solution:**

These verification conditions can be proved as follows:
1. \( X \geq 0 \Rightarrow X \geq 0 \land 0 = 0 \land 0 = 0 \)
   \[ \begin{array}{c}
   \{ \text{arithmetic} \} \\
   \text{True}
   \end{array} \]

2. \( X \geq 0 \land N = 0 \land \text{PRODUCT} = 0 \Rightarrow \text{PRODUCT} = N \times Y \land N \leq X \)
   \[ \begin{array}{c}
   \{ \text{logical simplification} \} \\
   0 = 0 \times Y \land 0 \leq 0 \\
   \{ \text{arithmetic} \} \\
   \text{True}
   \end{array} \]

3. \( \text{PRODUCT} = N \times Y \land N \leq X \land \neg (N < X) \Rightarrow \text{PRODUCT} = X \times Y \)
   \[ \begin{array}{c}
   \{ \text{arithmetic} \} \\
   \text{PRODUCT} = N \times Y \land N = X \Rightarrow \text{PRODUCT} = X \times Y \\
   \{ \text{logical simplification, arithmetic} \} \\
   \text{True}
   \end{array} \]

4. \( \text{PRODUCT} = N \times Y \land N \leq X \land N < X \Rightarrow \text{PRODUCT} + Y = (N + 1) \times Y \land (N + 1) \leq X \)
   \[ \begin{array}{c}
   \{ \text{arithmetic} \} \\
   \text{PRODUCT} = N \times Y \land N < X \Rightarrow \text{PRODUCT} + Y = (N + 1) \times Y \land (N + 1) \leq X \\
   \{ \text{logical simplification, arithmetic} \} \\
   \text{True}
   \end{array} \]

**QUESTION 2**

**[TOTAL MARKS: 20]**

2(a)  \[4 \text{ Marks}\]

Explain the difference between partial and total correctness.

**Solution:**

Within a partial correctness specification \( \{ P \} C \{ Q \} \), if command \( C \) is executed in a state in which precondition \( P \) is true and \( C \) terminates, then postcondition \( Q \) will be true. Within a total correctness specification \( [ P ] C [ Q ] \), if command \( C \) is executed in a state in which precondition \( P \) is true, then \( C \) terminates and the postcondition \( Q \) will be true.

2(b)  \[4 \text{ Marks}\]

Give an example of a specification which is partially correct, but not totally correct.

**Solution:**

Any specification in which the program does not terminate will suffice. For example:

\( \{ X = 1 \} \text{ WHILE T DO SKIP } \{ Y = 1 \} \) is true, but \( [ X = 1 ] \text{ WHILE T DO SKIP } [ Y = 1 ] \) is false.

2(c)  \[12 \text{ Marks}\]

Show that the following annotated specification is true:

\[ [ X = n \land n \geq 0 ] \]
\( \text{FACT} := 1; \{ X = n \land n \geq 0 \land \text{FACT} = 1 \} \)
\( \text{WHILE } X > 0 \text{ DO } \{ \text{FACT } \times X! = n! \land X \geq 0 \} \) \[ X \]
\( \text{BEGIN} \)
\( \text{FACT} := \text{FACT } \times X; \)
\( X := X - 1 \)
\( \text{END} \)
\( \text{FACT} = n! \)
Solution:
The following verification conditions will be generated:

1. \( X = n \land n \geq 0 \Rightarrow X = n \land n \geq 0 \land 1 = 1 \)
2. \( X = n \land n \geq 0 \land \text{FACT} = 1 \Rightarrow \text{FACT} \times X! = n! \land X \geq 0 \)
3. \( \text{FACT} \times X! = n! \land X \geq 0 \land \neg(X > 0) \Rightarrow \text{FACT} = n! \)
4. \( \text{FACT} \times X! = n! \land X \geq 0 \land X > 0 \Rightarrow X \geq 0 \)
5. \( \text{FACT} \times X! = n! \land X \geq 0 \land X = v \Rightarrow (\text{FACT} \times X) \times (X - 1)! = n! \land (X - 1) \geq 0 \land (X - 1) < v \)

These verification conditions can be proved as follows:

1. \( X = n \land n \geq 0 \Rightarrow X = n \land n \geq 0 \land 1 = 1 \)
   = \{ logical simplification \} True

2. \( X = n \land n \geq 0 \land \text{FACT} = 1 \Rightarrow \text{FACT} \times X! = n! \land X \geq 0 \)
   = \{ logical simplification \}
   \( 1 \times n! = n! \)
   = \{ arithmetic \} True

3. \( \text{FACT} \times X! = n! \land X \geq 0 \land \neg(X > 0) \Rightarrow \text{FACT} = n! \)
   = \{ arithmetic \}
   \( \text{FACT} \times X! = n! \land X = 0 \Rightarrow \text{FACT} = n! \)
   = \{ logical simplification, arithmetic \}
   True

4. \( \text{FACT} \times X! = n! \land X \geq 0 \land X > 0 \Rightarrow X \geq 0 \)
   = \{ logical simplification \}
   True

5. \( \text{FACT} \times X! = n! \land X \geq 0 \land X = v \Rightarrow (\text{FACT} \times X) \times (X - 1)! = n! \land (X - 1) \geq 0 \land (X - 1) < v \)
   = \{ arithmetic \}
   \( \text{FACT} \times X! = n! \land X > 0 \land X = v \Rightarrow (\text{FACT} \times X) \times (X - 1)! = n! \land (X - 1) \geq 0 \land (X - 1) < v \)
   = \{ logical simplification, arithmetic \}
   True

QUESTION 3 [TOTAL MARKS: 20]

3(a) [4 Marks]

What are the “laws of programming” in relation to refinement? Give an example of a law and the corresponding rule or axiom in Floyd-Hoare logic from which it is derived.

Solution:
The “laws of programming” in relation to refinement define how a specification can be refined to a program. The laws are derived from Floyd-Hoare logic and may generate proof obligations which need to be discharged. An example of a law is the SKIP law: \([P, P] \supseteq \{\text{SKIP}\}\). This is derived from the SKIP axiom from Floyd-Hoare logic: \([P] \text{ SKIP } [P]\).

3(b) [4 Marks]

What is the difference between program refinement and program verification? Give one advantage of using program refinement.
Solution:
Program verification involves taking completed programs and proving that they meet their specifications. Program refinement involves performing the proof in conjunction with the development to ensure a program is correct by construction. Two advantages of program refinement are:

1. Errors are spotted earlier in the design process
2. Reasons for design decisions are available

3(c) [12 Marks]

Refine the following specification to a corresponding program:

\[[\ Y > 0, X = R + Y \times Q \land R \leq Y]\]

Solution:
\[[\ Y > 0, X = R + Y \times Q \land R \leq Y]\]
\[\geq \{ \text{Block Law } \}
\begin{align*}
&[\ Y > 0, X = R + Y \times Q \land R \leq Y] \\
\end{align*}
\[\geq \{ \text{Sequencing Law } \}
\begin{align*}
&[\ Y > 0, R = X \land Y > 0; R = X \land Y > 0 \land Q = 0]; \\
&[R = X \land Y > 0 \land Q = 0, X = R + Y \times Q \land R \leq Y] \\
\end{align*}
\[\geq \{ \text{Derived Assignment } \} \quad \vdash Y > 0 \Rightarrow X = X \land Y > 0 \\
\begin{align*}
&[R = X \land Y > 0, X = R + Y \times Q \land R \leq Y] \\
\end{align*}
\[\geq \{ \text{Sequencing Law } \}
\begin{align*}
&R := X; \\
&[R = X \land Y > 0, R = X \land Y > 0 \land Q = 0]; \\
&[R = X \land Y > 0 \land Q = 0, X = R + Y \times Q \land R \leq Y] \\
\end{align*}
\[\geq \{ \text{Derived Assignment } \} \quad \vdash R = X \land Y > 0 \Rightarrow R = X \land Y > 0 \land 0 = 0 \\
\begin{align*}
&R := X; \\
&[R = X \land Y > 0 \land Q = 0, X = R + Y \times Q \land R \leq Y] \\
\end{align*}
\[\geq \{ \text{Precondition Weakening } \} \quad \vdash R = X \land Y > 0 \land Q = 0 \Rightarrow X = R + Y \times Q \land Y > 0 \\
\begin{align*}
&R := X; \\
&Q := 0; \\
&[X = R + Y \times Q \land Y > 0, X = R + Y \times Q \land R \leq Y] \\
\end{align*}
\[\geq \{ \text{Postcondition Strengthening } \} \quad \vdash X = R + Y \times Q \land Y > 0 \land \neg(Y \leq R) \Rightarrow X = R + Y \times Q \land R \leq Y \\
\begin{align*}
&R := X; \\
&Q := 0; \\
&[X = R + Y \times Q \land Y > 0, X = R + Y \times Q \land Y > 0 \land \neg(Y \leq R)] \\
\end{align*}
\[\geq \{ \text{WHILE Law } \} \quad \vdash X = R + Y \times Q \land Y > 0 \land Y \leq R \Rightarrow R \geq 0 \\
\begin{align*}
&R := X; \\
&Q := 0; \\
&[X = R + Y \times Q \land Y > 0, X = R + Y \times Q \land Y > 0 \land \neg(Y \leq R)] \\
\end{align*}
\]
BEGIN
R := X;
Q := 0;
WHILE Y ≤ R DO
    [X = R + Y \times Q \land Y > 0 \land Y ≤ R \land R = n,
     X = R + Y \times Q \land Y > 0 \land R < n]
END ⊇ { Block Law }
BEGIN
R := X;
Q := 0;
WHILE Y ≤ R DO
    BEGIN
        [X = R + Y \times Q \land Y > 0 \land Y ≤ R \land R = n,
         X = R + Y \times Q \land Y > 0 \land R < n]
    END
END ⊇ {Sequence Law }
BEGIN
R := X;
Q := 0;
WHILE Y ≤ R DO
    BEGIN
        X := (R - Y) + Y \times Q \land Y > 0 \land (R - Y) < n;
        X := R + Y \times Q \land Y > 0 \land R < n
    END
END ⊇ { Assignment Law }
BEGIN
R := X;
Q := 0;
WHILE Y ≤ R DO
    BEGIN
        Q := Q + 1;
        R := R - Y
    END
END ⊇ { Derived Assignment }
BEGIN
R := X;
Q := 0;
WHILE Y ≤ R DO
    BEGIN
        R := R - Y
    END
END

QUESTION 4 [TOTAL MARKS: 20]
A storage manager for an operating system has a certain number maxblocks of blocks to manage on behalf of registered users of the operating system. At any one time, a block can only be allocated to one user, and a user must free a block
before it can be allocated to another user. The blocks are identified by natural numbers in the range 1..\textit{maxblocks}, and users are identified by natural numbers in the range 1..\textit{maxusers}. The following operations need to be supported:

\textbf{adduser}: an operation to add a user to the set of registered users, and to return an unallocated user identifier for this user.

\textbf{getblock}: an operation to allocate a block to a registered user. The input is the registered user identifier, and the output is a previously unallocated block, which is now allocated to the user.

\textbf{freeblock}: an operation to free a block allocated to a registered user. The inputs are the registered user identifier and the block; there is no output.

\textbf{freemem}: an operation to return the number of free blocks in memory. There is no input, and the output is the number of free blocks.

\textbf{removeuser}: an operation to remove a user from the set of registered users. This will also free all the blocks associated with the user. The input is the registered user identifier; there is no output.

4(a) [4 Marks]
Define the context for an Event-B specification of the storage manager.

\textit{Solution:}

\textbf{CONTEXT} \textsl{Storage\_ctx}

\textbf{CONSTANTS}

\textit{maxusers}

\textit{maxblocks}

\textit{USERS}

\textit{BLOCKS}

\textbf{AXIOMS}

\textbf{axm1} : \textit{maxusers} \in \mathbb{N}_1

\textbf{axm2} : \textit{maxblocks} \in \mathbb{N}_1

\textbf{axm3} : \textit{USERS} = 1..\textit{maxusers}

\textbf{axm4} : \textit{BLOCKS} = 1..\textit{maxblocks}

END

4(b) [6 Marks]
Define the variables for an Event-B specification of the storage manager. Define a suitable invariant for these variables, and show their initialisation, ensuring that this initialisation satisfies the invariant.

\textit{Solution:}
MACHINE Storage
SEES Storage_ctx
VARIABLES
allocated
registered
free

INvariants
inv1 : allocated ∈ BLOCKS ↦→ USERS
inv2 : registered ⊆ USERS
inv3 : free ∈ N

EVENTS
Initialisation
begin
act1 : allocated := ∅
act2 : registered := ∅
act3 : free := maxblocks
end

4(c) [10 Marks]
Specify the events for an Event-B specification of the storage manager, making use of the definitions in 4(a) and 4(b).

Solution:

Event adduser ≜
any
u
where
grd1 : u ∈ USERS \ registered
then
act1 : registered := registered ∪ {u}
end
Event getblock ≜
any
u
b
where
grd1 : u ∈ registered
grd2 : b ∈ BLOCKS \ dom(allocated)
then

\[ \text{act1 : allocated := allocated } \cup \{ b \mapsto u \} \]

end

Event \( freeblock \) \( \triangleq \)

any

\( b \)

\( u \)

where

\( \text{grd1 : } b \in BLOCKS \)

\( \text{grd2 : } u \in \text{registered} \)

\( \text{grd3 : } b \mapsto u \in \text{allocated} \)

then

\[ \text{act1 : allocated := } \{ b \} \triangleleft \text{allocated} \]

end

Event \( freemem \) \( \triangleq \)

begin

\[ \text{act1 : free := maxblocks } - \text{card(allocated)} \]

end

Event \( removeuser \) \( \triangleq \)

any

\( u \)

where

\( \text{grd1 : } u \in \text{registered} \)

then

\[ \text{act1 : registered := registered } \setminus \{ u \} \]

\[ \text{act2 : allocated := allocated } \triangleright \{ u \} \]

end

END

QUESTION 5 [TOTAL MARKS: 20]

5(a) [7 Marks]

Write an Event-B specification for a program computing the sum of all the numbers in an array \( a : 1..n \rightarrow \mathbb{N} \) where \( n \geq 1 \). The specification should have a constant function which computes the desired result of the program and a result variable \( result \). It should also have two abstract events \( Initialisation \) and \( Sum \) which give the appropriate precondition and postcondition respectively for \( result \).

Solution:

CONTEXT \( Sum_{ctx} \)

Module Code: CA648

Semester Two Examinations 2010
CONSTANTS
   n
   a
   sum

AXIOMS
   axm1 : n ∈ N,
   axm2 : a ∈ 1..n → N
   axm3 : sum ∈ N → N
   axm4 : (0 ↦ 0) ∈ sum
   axm5 : ∀n, s, n ↦ s ∈ sum ⇒ (n + 1 ↦ s + a(n + 1)) ∈ sum

END
MACHINE Sum
SEES Sum_ctx
VARIABLES
   result

INvariants
   inv1 : result ∈ N

EVENTS
Initialisation
   begin
      act1 : result := 0
   end
Event Sum
   begin
      act1 : result := sum(n)
   end

END

5(b) [8 Marks]

Give a refinement of the specification in 5(a) which adds variables index and sumsofar, giving the value of the current index in the array, and the sum of the numbers in the array up to this index, respectively. Your refinement should also add two further events Update and Progress. The Update event should update the value of the sumsofar variable. The convergent event Progress should be used to ensure termination by decreasing the variant. You should also refine the events Initialisation and Sum to give precise initial and final values for the result variable.

Solution:
MACHINE SumR
REFINES Sum
SEES Sum_ctx

VARIABLES
  result
  index
  sumsofar

INVARIANTS
  inv1 : index ∈ 0 .. n
  inv2 : sumsofar ∈ N

THEOREMS
  thm1 : sumsofar = sum(index)

EVENTS

Initialisation
  begin
    act1 : result := 0
    act2 : index := 0
    act3 : sumsofar := 0
  end

Event Sum ≜
  refines Sum
  when
    grd1 : index ≥ n
  then
    act1 : result := sumsofar
  end

Event Progress ≜
  when
    grd1 : index < n
  then
    act1 : index := index + 1
  end

Event Update ≜
  when
    grd1 : index < n
  then
    act1 : sumsofar := sumsofar + a(index)
  end

END
5(c) [5 Marks]

Give a program which computes the sum of all the numbers in an array and is a refinement of your answers given in 5(a) and 5(b).

Solution:
result := 0;
index := 0;
sumsofar := 0;
WHILE index < n DO
    BEGIN
        index := index + 1;
        sumsofar := sumsofar + a(index)
    END;
result := sumsofar