FACULTY OF
SCIENCE & TECHNOLOGY

Department of Computer Science

2015 – 2016

Code:	ECSE610	**Level:** 6	**Semester:** 1
Title:	Formal Specification		
Date:	12th May 2016		
Time:	10:00 – 12:00		

INSTRUCTIONS TO CANDIDATES

Answer ALL questions in Section A and TWO questions from Section B.
Section A is worth a total of 50 marks.
Each question in section B is worth 25 marks.
The B-Method's Abstract Machine Notation (AMN) is given in Appendix B.

FACULTY OF
SCIENCE & TECHNOLOGY

Department of Computer Science

2015 – 2016

Code:	ECSE610	**Level:** 6	**Semester:** 1
Title:	Formal Specification [MARKING SCHEME]		
Date:	12th May 2016		
Time:	10:00 – 12:00		

INSTRUCTIONS TO CANDIDATES

Answer ALL questions in Section A and TWO questions from Section B.
Section A is worth a total of 50 marks.
Each question in section B is worth 25 marks.

Section A

Answer ALL questions from this section.
You may wish to consult the B-Method notation given in Appendix B.

Question 1

(a) The building block of a B-method specification is the concept of an *Abstract Machine (AM)*. Explain what a B Abstract Machine is, in particular, you should answer the following questions:

- What is it similar to?
- What is it a specification of?
- What are its main logical parts?

[6 marks]

(b) Explain the purpose of the following B Abstract Machine *clauses* and illustrate their meaning by giving an example for each clause.

- `SETS`
- `CONSTANTS`
- `PROPERTIES`
- `VARIABLES`
- `INVARIANT`
- `INITIALISATION`
- `OPERATIONS`

[14 marks]
[TOTAL 20]

Section A

Answer ALL questions from this section.

Question 1

(a)

- An Abstract Machine is similar to the programming concepts of: modules, class definition (e.g. Java) or abstract data types. **[1 mark]**
- An Abstract Machine is a specification of what a system should be like, or how it should behave (operations); but not how a system is to be built, i.e. no implementation details. **[2 marks]**
- The main logical parts of an Abstract Machine are its: *name*, *local state*, represented by "encapsulated" variables, *collection of operations*, that can access & update the state variables. **[3 marks]**

[PART Total 6]

(b) Abstract Machine *clauses*:

- `SETS` declaration of deferred and enumerated sets, i.e. user defined "abstract" sets (& types). Example. **[2 marks]**
- `CONSTANTS` declaration of constants, e.g. simple values – numbers, etc, or constant sets, relations, functions, etc. Example. **[2 marks]**
- `PROPERTIES` declaration of the types of the sets & constants; & properties they must satisfy, e.g. specific values or more general constraints. Example. **[3 marks]**
- `VARIABLES` declaration of variables. Example. **[1 mark]**
- `INVARIANT` declaration of the types of all variables & invariant properties of the variables. Example. **[3 marks]**
- `INITIALISATION` initialisation of variables with suitable values. Example. **[1 mark]**
- `OPERATIONS` declaration of the operations in the form of a header and body. Example. **[2 marks]**

[PART Total 14]

Question 2

You are given the following collection of B set and function declarations:

$$SHAPE \ = \ \{ \ Oval, \ Circle, \ Triangle, \ Rectangle, \ Square, \\ Rhombus, \ Pentagon, \ Hexagon \ \}$$

$$Quadrilaterals \ \in \ \mathbb{P}(SHAPE)$$
$$Quadrilaterals \ = \ \{ \ Rectangle, \ Square, \ Rhombus \ \}$$

$$NonPolygons \ \in \ \mathbb{P}(SHAPE)$$
$$NonPolygons \ = \ \{ \ Oval, \ Circle \ \}$$

$$edges \ \in \ SHAPE \rightarrow \mathbb{N}$$
$$edges \ = \ \{ \ Oval \mapsto 1, \ Circle \mapsto 1, \ Triangle \mapsto 3, \ Rectangle \mapsto 4, \\ Square \mapsto 4, \ Rhombus \mapsto 4, \ Pentagon \mapsto 5, \ Hexagon \mapsto 6 \ \}$$

Evaluate the following expressions:

(a) $Quadrilaterals \ \cup \ NonPolygons$ **[1 mark]**

(b) $Quadrilaterals \setminus \{ \ Rhombus \ \}$ **[1 mark]**

(c) $\mathrm{card}(edges)$ **[1 mark]**

(d) $\mathrm{dom}(edges)$ **[1 mark]**

(e) $\mathrm{ran}(edges)$ **[1 mark]**

(f) $edges(Hexagon)$ **[1 mark]**

(g) $\mathbb{P}(NonPolygons)$ **[2 marks]**

(h) $edges \rhd \{ \ 4 \ \}$ **[2 marks]**

(i) $NonPolygons \lhd edges$ **[2 marks]**

(j) $(\ Quadrilaterals \cup NonPolygons \ ) ⩤ edges$ **[3 marks]**

[TOTAL 15]

Question 2

Evaluate the following expressions:

(a) $Quadrilaterals \cup NonPolygons \ = \ \{ \ Rectangle, Square, Rhombus, Oval, Circle \ \}$
[1 mark]

(b) $Quadrilaterals \setminus \{Rhombus\} \ = \ \{ \ Rectangle, Square \ \}$
[1 mark]

(c) $\mathrm{card}(edges) = 8$
[1 mark]

(d) $\mathrm{dom}(edges) = \{ \ Rectangle, Square, Rhombus, Oval, Circle, Pentagon, Hexagon \ \}$
[1 mark]

(e) $\mathrm{ran}(edges) = \{ \ 1, 3, 4, 5, 6 \ \}$
[1 mark]

(f) $edges(Hexagon) = 6$
[1 mark]

(g) $\mathbb{P}(NonPolygons) \ = \ \{ \ \{\}, \{Oval\}, \{Circle\}, \{Oval, Circle\} \ \}$
[2 marks]

(h) $edges \rhd \{4\} = \{ \ Rectangle \mapsto 4, \ Square \mapsto 4, \ Rhombus \mapsto 4 \ \}$
[2 marks]

(i) $NonPolygons \lhd edges \ = \ \{ \ Oval \mapsto 1, \ Circle \mapsto 1 \ \}$
[2 marks]

(j) $(\ Quadrilaterals \cup NonPolygons \ ) \ntriangleleft edges$
$= \{ \ Pentagon \mapsto 5, \ Hexagon \mapsto 6 \ \}$
[3 marks]

[QUESTION Total 15]

Question 3

Given the following B declarations of the set of $Letters$ and the relations R_1, R_2 & R_3:

$$Letter \ = \ \{ \ a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z \ \}$$

$$\begin{aligned} R_1 &\in Letter \leftrightarrow \mathbb{N} \\ R_1 &= \{ \ a \mapsto 1, \ b \mapsto 1, \ c \mapsto 3, \ d \mapsto 2, \ e \mapsto 4, \\ &\qquad f \mapsto 4, \ g \mapsto 5, \ h \mapsto 6 \ \} \end{aligned}$$

$$\begin{aligned} R_2 &\in Letter \leftrightarrow \mathbb{N} \\ R_2 &= \{ \ a \mapsto 1, \ b \mapsto 1, \ b \mapsto 2, \ c \mapsto 3, \ d \mapsto 2 \ \} \end{aligned}$$

$$\begin{aligned} R_3 &\in \mathbb{N} \leftrightarrow Letter \\ R_3 &= \{ \ 1 \mapsto x, \ 2 \mapsto y, \ 4 \mapsto z \ \} \end{aligned}$$

(a) Evaluate the following expressions:

(i) $R_1 \, [\, \{ \ a, \ c, \ e \ \} \,]$ **[2 marks]**

(ii) $R_3 \mathbin{\lhd\!\!\!+} \{ \ 0 \mapsto w, \ 4 \mapsto a \ \}$ **[3 marks]**

(iii) $R_2 \ ; R_3$ **[4 marks]**

(b) For each of the relations R_1, R_2 and R_3 state whether it is just a *relation* or is also a *function*. In addition, if you decide that one of these relations is a function then state what kind of function it is, e.g. partial, total, injective, etc. **[6 marks]**

[TOTAL 15]

Question 3

(a) Evaluate the following expressions:

(i) $R_1 \; [\; \{\; a,\; c,\; e \;\} \;] \;=\; \{\; 1,\; 3,\; 4 \;\}$ **[2 marks]**

(ii) $R_3 \oplus \{\; 0 \mapsto w,\; 4 \mapsto a \;\} \;=\; \{\; 0 \mapsto w,\; 1 \mapsto x,\; 2 \mapsto y,\; 4 \mapsto a \;\}$ **[3 marks]**

(iii) $R_2 \,;\, R_3 \;=\; \{\; a \mapsto x,\; b \mapsto x,\; b \mapsto y,\; d \mapsto y \;\}$ **[4 marks]**

[PART Total 9]

(b) R_1 is a partial function, because no element in the domain maps to two elements in the range and not all numbers are in the domain. It is not an injective or surjective function. **[2 marks]**

R_2 is just a relation because it has an element in the domain mapping to more than one value in the range, e.g. b. **[1 mark]**

R_3 is a partial injective function, because no element in the domain maps to two elements in the range, not all numbers are in the domain and each element in the domain maps to a different value in the range. It is not a surjective function. **[3 marks]**

[PART Total 6]

[QUESTION Total 15]

Section B

Answer TWO questions from this section.
You may wish to consult the B-Method notation given in Appendix B.

Question 4

Write a B-Method machine that specifies a *queue* of people waiting to be "served". For example, at a bank, supermarket checkout, etc. Due to the lack of available space the queue has a maximum permitted length.
Your B machine should deal with error handling where required and should include the following:

(a) Any sets, constants and variables, and any state invariant that the *queue* requires. **[9 marks]**

(b) The queuing operations:

(i) $JoinQueue$ – a new person joins the end of the *queue*. **[6 marks]**

(ii) $GetServed$ – the next person gets "served", i.e., leaves the front of the *queue*. **[6 marks]**

(iii) $QueueStatus$ – reports via a suitable message whether the queue is *empty*, *full* or *neither full or empty*. **[4 marks]**

[TOTAL 25]

Section B

Answer TWO questions from this section.

Question 4

A Queue B machine similar to the following is expected.
Some possible acceptable alternatives:
Uses B symbols not ASCII versions, or a mixture. Enumerates `PERSON`:

```
PERSON = { Jim, Joe, ... }
```

Combines `ANSWER` & `REPORT` or uses string literals. Also likely that some less important parts are omitted, e.g. preconditions – "`report : REPORT`", use of `Nobody`. Using an ordinary sequence `seq` rather than an injective sequence `iseq`.

(a)
```
MACHINE Queue

    SETS
      PERSON ;

      ANSWER = { Queue_is_Empty, Queue_is_Full,
                 Queue_is_Neither_Empty_or_Full } ;

      REPORT = { Person_Joined_Queue,
                 ERROR_Queue_is_Full,
                 Person_Served,
                 ERROR_Queue_Empty    }

    CONSTANTS
      MaxQueueLength, EmptyQueue, Nobody

    PROPERTIES
      MaxQueueLength : NAT1  &  MaxQueueLength = 5  &
      EmptyQueue : iseq( PERSON ) &  EmptyQueue = [] &
      Nobody : PERSON

    VARIABLES
      queue
```

Section B

Answer TWO questions from this section.
You may wish to consult the B-Method notation given in Appendix B.

Question 4

Write a B-Method machine that specifies a *queue* of people waiting to be "served". For example, at a bank, supermarket checkout, etc. Due to the lack of available space the queue has a maximum permitted length.
Your B machine should deal with error handling where required and should include the following:

(a) Any sets, constants and variables, and any state invariant that the *queue* requires. **[9 marks]**

(b) The queuing operations:

- **(i)** $JoinQueue$ – a new person joins the end of the *queue*. **[6 marks]**
- **(ii)** $GetServed$ – the next person gets "served", i.e., leaves the front of the *queue*. **[6 marks]**
- **(iii)** $QueueStatus$ – reports via a suitable message whether the queue is *empty*, *full* or *neither full or empty*. **[4 marks]**

[TOTAL 25]

```
INVARIANT
  queue : iseq( PERSON ) & size( queue ) <= MaxQueueLength
  & Nobody /: ran( queue )

INITIALISATION
  queue := EmptyQueue
```

Marks for each clause: `SETS` **[3 marks]** , `CONSTANTS & PROPERTIES` **[3 marks]** , `VARIABLES INVARIANT & INITIALISATION` **[3 marks]** .

[PART Total 9]

(b) `OPERATIONS`

```
report <-- JoinQueue( person ) =
    PRE
        person : PERSON & person /: ran( queue )
        & person /= Nobody & report : REPORT
    THEN
        IF ( size( queue )  < MaxQueueLength )
        THEN
            queue  := queue <- person       ||
            report := Person_Joined_Queue
        ELSE
            report := ERROR_Queue_is_Full
        END
    END ;
```

[Subpart (b.i) 6 marks]

```
report, nextperson <-- GetServed =
    PRE
        nextperson : PERSON & report : REPORT
    THEN
        IF ( queue /= EmptyQueue )
        THEN
            nextperson := first( queue )  ||
            queue  := tail( queue )       ||
            report := Person_Served
        ELSE
            nextperson := Nobody        ||
            report := ERROR_Queue_Empty
         END
```

Section B

Answer TWO questions from this section.
You may wish to consult the B-Method notation given in Appendix B.

Question 4

Write a B-Method machine that specifies a *queue* of people waiting to be "served". For example, at a bank, supermarket checkout, etc. Due to the lack of available space the queue has a maximum permitted length.
Your B machine should deal with error handling where required and should include the following:

(a) Any sets, constants and variables, and any state invariant that the *queue* requires. **[9 marks]**

(b) The queuing operations:

- **(i)** $JoinQueue$ – a new person joins the end of the *queue*. **[6 marks]**
- **(ii)** $GetServed$ – the next person gets "served", i.e., leaves the front of the *queue*. **[6 marks]**
- **(iii)** $QueueStatus$ – reports via a suitable message whether the queue is *empty*, *full* or *neither full or empty*. **[4 marks]**

[TOTAL 25]

```
        END ;
```

[Subpart (b.ii) 6 marks]

```
    status <-- QueueStatus =
        PRE
            status : ANSWER
        THEN
            IF ( queue = EmptyQueue )
            THEN
                 status := Queue_is_Empty
            ELSIF
                ( card(queue) = MaxQueueLength )
            THEN
                 status := Queue_is_Full
             ELSE
                 status := Queue_is_Neither_Empty_or_Full
            END
        END

END  /* Queue */
```

[Subpart (b.iii) 4 marks]

[PART Total 16]

[QUESTION Total 25]

Question 5

Refer to the `Library` B machine given in the exam paper's Appendix ??.

(a) The `Library`'s invariant.

(i) `hasread : READER <-> BOOK   /* Inv-1 */`
The use of a *relation* `<->` ($\leftrightarrow$) means that a reader can have read many books & that a book may have been read by many readers. **[2 marks]**

Question 5

Appendix A contains the `Library` B machine, this specifies a simple book lending library.
The library has a catalogue of book titles (`BOOK`) and lends individual copies of each book (`COPY`) to its readers (`READER`).
The library's system holds the following information about its books and readers:

- The book title for each book copy (`copyof`).
- The books each reader has previously read (`hasread`).
- If a reader is currently reading a book then it records which copy he/she is reading (`reading`).

The system provides the following operations:

- Recording that a reader has started/finished reading a book.
- Check if a reader is currently reading a book; what book he/she is reading; has he/she read a book.

With reference to the `Library` B machine answer the following questions.

(a) The `Library`'s invariant, is given in the `INVARIANT` clause:

```
INVARIANT
  hasread : READER <-> BOOK &             /* Inv-1 */
  reading : READER >+> COPY &             /* Inv-2 */
  (reading ; copyof) /\ hasread = {}      /* Inv-3 */
```

Using "plain English" only, answer the following questions:

(i) In `Inv-1` explain what the use of a *relation* `<->` ($\leftrightarrow$) means about the relationship between readers and books. Why would it not make sense to use a function? **[4 marks]**

(ii) In `Inv-2` explain what the choice of a *partial injection* `>+>` ($\rightarrowtail\!\!\!\!\!+$) means regarding how many books a reader can read at any one time and how many readers can read the same book? What would it mean if this was changed to a relation? **[4 marks]**

(iii) Explain what `Inv-3` means. **[2 marks]**

[Continued Overleaf]

It does not make sense to use a function, as it would mean that a reader could only have read one book. If a more specific function was used, e.g. an injective function, then a book could also only ever have been read by one reader. **[2 marks]**

[SUBPART Total 4]

(ii) `reading : READER >+> COPY   /* Inv-2 */`

The use of a *partial injection* >+> ($\rightarrowtail\!\!\!\!\!+\,$) means that a reader does not have to be currently reading or can at most be reading one book & that only one reader can read a particular copy of a book at a time. **[3 marks]**

If this was changed to a relation ($\leftrightarrow$) then a reader could read several books at once & several readers could read the same copy of a book at a time. **[1 mark]** **[4 marks]**

(iii) `(reading ; copyof) /\ hasread = {} /* Inv-3 */`

It means that a reader cannot be currently reading a book he/she has previously read, i.e. a reader cannot re-read a book. **[2 marks]**

[PART Total 10]

(b) Explain "plain English" the meaning of the *preconditions* for the operations:

(i) `startReading` preconditions – the parameters are a reader & a copy of a book; the reader has not previously read it; the reader is not currently reading a book & this copy of the book is not being read by anyone.

[SUBPART Total 4]

(ii) `finishReading` preconditions – the parameters are a reader & a copy of a book; the reader is currently this copy of the book.

[SUBPART Total 3]

(iii) `currentlyReading` preconditions – the parameter is a reader & the reader is currently reading a book.

[SUBPART Total 2]

[PART Total 9]

(c) The `Library` machine's Structure Diagram.

(b) Explain in "plain English" the meaning of the *preconditions* for the operations:

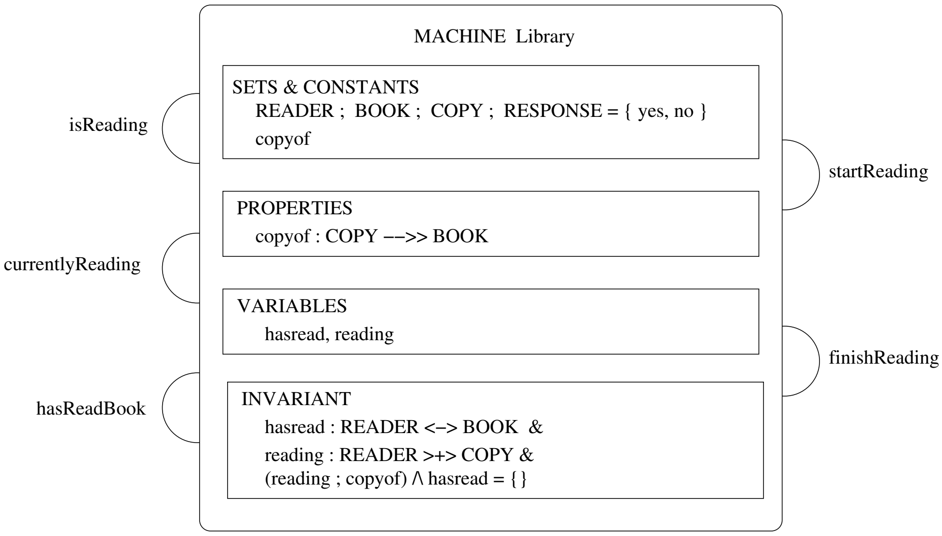
(i) `startReading` **[4 marks]**

(ii) `finishReading` **[3 marks]**

(iii) `currentlyReading` **[2 marks]**

(c) Draw the *Structure Diagram* for the `Reading` machine. **[6 marks]**

[TOTAL 25]



Internal structure **[4 marks]** , Operations **[2 marks]** .

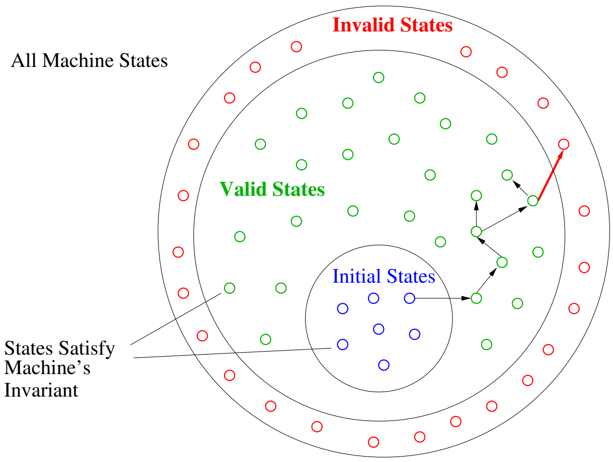
[PART Total 6]

[QUESTION Total 25]

Question 6

Answers similar to the following are expected.

(a) **(i)** Three categories of system states: *valid* states, *initial* or *start* states & *error* or *invalid* states. The diagram illustrates all 3 possible states, arrows indicate operations moving from one to another.



Question 6

(a) The B-Method is used to develop software systems. B is suitable for this task because it allows a system designer to specify important aspects of a system. Within this context of using a B machine to specify a system, answer the following questions.

(i) There are three categories of states that a system (B machine) can be in, what are they? Illustrate your answer by means of a diagram. **[4 marks]**

(ii) What is the role of the state invariant and what is its relationship to the system states? **[2 marks]**

(iii) Explain what important property the initial value(s) of a B machine's variables must satisfy. **[2 marks]**

(iv) When specifying an operation it is usually necessary to define an explicit precondition for the operation using PRE. What are preconditions and what is their purpose? **[4 marks]**

(v) If the specification of a machine's operation is required to be *"total"*, explain what this means. **[2 marks]**

(b) Given an abstract B machine specification, such as the following:

```
MACHINE Name

  SETS            Sets
  CONSTANTS       Consts
  PROPERTIES      Props
  VARIABLES       Vars
  INVARIANT       Inv
  INITIALISATION  Init

  OPERATIONS

  yy <-- op( xx ) =
          PRE  PC
          THEN Subst
          END ;
  ...
END
```

[Continued Overleaf]

[4 marks]

(ii) The *state invariant* is the constraints & properties (defined in a machine) that the states of the system/machine are required to satisfy during its lifetime, i.e. all of the states it passes through during its execution should satisfy them. The valid states are those that satisfy the *state invariant*. **[2 marks]**

(iii) The initialisation of a machine's variables defines its *initial state(s)*, i.e. defines the set of possible starting states of the system/machine. Any initial state must also be a valid state, i.e. one that satisfies the state invariant. **[2 marks]**

(iv) *Preconditions* are predicates that determine the (valid) before states of the system/machine in which the operation can successfully be completed, if they are not satisfied then the operation must not be executed. (B preconditions also include the types of input parameters & outputs.) **[2 marks]**

That is they characterise the *before* states that ensure that the new values assigned to the machine's state variables by the operation (after state) will also satisfy the state invariant, i.e. ensures a transition from one valid state to another. **[2 marks]**

(v) A *total operation* is one which has an outcome defined for all possible states. In other words there is no legal state of the system for which the operation is not defined, i.e. *"all eventualities have been catered for"*. **[2 marks]**

[PART Total 14]

(b) **(i)** Data Proof Obligation is concerned with the sets (`Sets`) & constants (`Consts`) defined in a machine & their logical & defining properties (`Props`).

These definitions "generate" the following proof obligation.

In order for the machine to have any valid values for its sets & constants at all, it must always be possible, to find appropriate sets & constants:

$$\exists \; Sets, \; Consts \cdot (\; Props \;) \qquad \text{(Data PO)}$$

[SUBPART Total 3]

(ii) Initialisation Proof Obligation is concerned with the initialisation of the machine's state variables (`Vars`). That there is at least one

Particular informal claims are usually made about its *correctness*. For example, it makes sense and is coherent; it can be instantiated; its operations perform correctly. These implicit claims are formalised in the B-Method by means of logical *"proof obligations"* on a B machine.

With reference to the above B machine describe the three main types of *proof obligations*:

(i) Data Proof Obligation **[3 marks]**

(ii) Initialisation Proof Obligation **[3 marks]**

(iii) Operation Proof Obligation **[5 marks]**

[TOTAL 25]

valid state of the machine, i.e. there is at least on set of values for the machine's state variables satisfy its invariant (`Inv`).

The first step is to prove that the initialisation `Init` establishes the invariant `Inv`, i.e. the machine's initial state satisfies the invariant.

This means given the sets & constants there must be a state that meets the machine invariant `Inv`:

$$Props \;\Rightarrow\; \exists\, Vars \cdot (\; Inv \;) \qquad \text{(Initialisation PO)}$$

[SUBPART Total 3]

(iii) Operation Proof Obligation is concerned with proving that the AMN specification of an operation:

```
PRE  PC  THEN  Subst  END
```

preserves the invariant (`Inv`) when it is invoked when the precondition (`PC`) is true.

The following has to be proved:

$$Inv \;\wedge\; Props \;\Rightarrow\; [\; Subst \;]Inv \qquad \text{(Operation PO)}$$

Note that "$[\; Subst \;]Inv$" is the "Weakest Precondition" of `Subst`. If the machine is in a state in which the invariant holds & the precondition also holds, then it should also be in a state in which execution of `Subst` is guaranteed to achieve `Inv`: after executing `Subst`, the invariant `Inv` must still be true.

[SUBPART Total 5]

[PART Total 11]

[QUESTION Total 25]

Appendix A. Library B Machine

The following is a B Machine – Library that specifies a simple book lending library.

```
MACHINE  Library

  SETS
    READER ;  BOOK ;  COPY ;  RESPONSE = { yes, no }

  CONSTANTS
    copyof

  PROPERTIES
    copyof : COPY -->> BOOK          /* Total Surjection */

  VARIABLES
    hasread, reading

  INVARIANT
    hasread : READER <-> BOOK &       /* Relation */
    reading : READER >+> COPY &       /* Partial Injection */
    (reading ; copyof) /\ hasread = {}

  INITIALISATION
    hasread := {} || reading := {}

  OPERATIONS

  startReading( rr, cc ) =
    PRE
        rr : READER  &  cc : COPY        &
        copyof(cc) /: hasread[ { rr } ]  &
        rr /: dom(reading)               &
        cc /: ran(reading)
    THEN
       reading := reading \/ { rr |-> cc }
    END ;

```

[Continued on next page.]

```
    finishReading( rr, cc ) =
      PRE
         rr : dom( reading ) &
         rr : READER  &  cc : COPY  &  reading(rr) = cc
      THEN
         hasread :=  hasread \/ { rr |-> copyof(cc) }  ||
         reading :=  { rr } <<| reading
      END ;

    resp <-- isReading( rr ) =
      PRE
         rr : READER
      THEN
        IF   ( rr : dom(reading) )
        THEN resp := yes
        ELSE resp := no
        END
      END ;

    bb <-- currentlyReading(rr) =
      PRE
         rr : READER  &  rr : dom(reading)
      THEN
         bb := copyof( reading(rr) )
    END ;

    resp <-- hasReadBook( rr, bb ) =
      PRE
         rr : READER  &  bb : BOOK
      THEN
        IF ( bb : hasread[ { rr } ] )
        THEN resp := yes
        ELSE resp := no
        END
      END

  END /* Library */
```

Appendix B. B-Method's Abstract Machine Notation (AMN)

The following tables present AMN in two versions: the "pretty printed" symbol version & the ASCII machine readable version used by the B tools: *Atelier B* and *ProB*.

B.1 AMN: Number Types & Operators

B Symbol	ASCII	Description
$\mathbb{N}$	`NAT`	Set of natural numbers from 0
$\mathbb{N}_1$	`NAT1`	Set of natural numbers from 1
$\mathbb{Z}$	`INTEGER`	Set of integers
$\mathrm{pred}(x)$	`pred(x)`	predecessor of x
$\mathrm{succ}(x)$	`succ(x)`	successor of x
$x + y$	`x + y`	x plus y
$x - y$	`x - y`	x minus y
$x * y$	`x * y`	x multiply y
$x \div y$	`x div y`	x divided by y
$x \bmod y$	`x mod y`	remainder after x divided by y
$x \widehat{\ } y$	`x ** y`	x to the power y, x^y
$\min(A)$	`min( A )`	minimum number in set A
$\max(A)$	`max( A )`	maximum number in set A
$x \mathinner{..} y$	`x .. y`	range of numbers from x to y inclusive

B.2 AMN: Number Relations

B Symbol	ASCII	Description
$x = y$	`x = y`	x equal to y
$x \neq y$	`x /= y`	x not equal to y
$x < y$	`x < y`	x less than y
$x \leq y$	`x <= y`	x less than or equal to y
$x > y$	`x > y`	x greater than y
$x \geq y$	`x >= y`	x greater than or equal to y

B.3 AMN: Set Definitions

B Symbol	ASCII	Description
$x \in A$	`x : A`	x is an element of set A
$x \notin A$	`x /: A`	x is not an element of set A
$\varnothing$, $\{ \}$	`{}`	Empty set
$\{ 1 \}$	`{ 1 }`	Singleton set (1 element)
$\{ 1, 2, 3 \}$	`{ 1, 2, 3 }`	Set of elements: 1, 2, 3
$x \mathrel{..} y$	`x .. y`	Range of integers from x to y inclusive
$\mathbb{P}(A)$	`POW(A)`	Power set of A
$\mathbb{P}_1(A)$	`POWn(A)`	Power set of Non-empty sets A
$\mathrm{card}(A)$	`card(A)`	Cardinality, number of elements in set A

B.4 AMN: Set Operators & Relations

<table>
<tr><th>B Symbol</th><th>ASCII</th><th>Description</th></tr>
<tr><td>$A \cup B$</td><td>A \/ B</td><td>Union of A and B</td></tr>
<tr><td>$A \cap B$</td><td>A /\ B</td><td>Intersection of A and B</td></tr>
<tr><td>$A \setminus B$</td><td>A \ B</td><td>Set subtraction of A and B</td></tr>
<tr><td>$\bigcup AA$</td><td>Union AA</td><td>Distributed union of AA</td></tr>
<tr><td>$\bigcap AA$</td><td>Intersection AA</td><td>Distributed intersection of AA</td></tr>
<tr><td>$A \subseteq B$</td><td>A <: B</td><td>A is a subset of or equal to B</td></tr>
<tr><td>$A \not\subseteq B$</td><td>A /<: B</td><td>A is not a subset of or equal to B</td></tr>
<tr><td>$A \subset B$</td><td>A <<: B</td><td>A is a strict subset of B</td></tr>
<tr><td>$A \not\subset B$</td><td>A /<<: B</td><td>A is not a strict subset of B</td></tr>
<tr><td>$\{ x \mid x \in TS \wedge C \}$</td><td>{ x | x : TS & C }</td><td>Set comprehension</td></tr>
</table>

B.5 AMN: Logic

B Symbol	ASCII	Description
$\neg P$	not P	Logical negation (not) of P
$P \wedge Q$	P & Q	Logical and of P, Q
$P \vee Q$	P or Q	Logical or of P, Q
$P \Rightarrow Q$	P => Q	Logical implication of P, Q
$P \Leftrightarrow Q$	P <=> Q	Logical equivalence of P, Q
$\forall\, xx \cdot (P \Rightarrow Q)$	!(xx).(P => Q)	Universal quantification of xx over $(P \Rightarrow Q)$
$\exists\, xx \cdot (P \wedge Q)$	#(xx).(P & Q)	Existential quantification of xx over $(P \wedge Q)$
$TRUE$	TRUE	Truth value $TRUE$.
$FALSE$	FALSE	Truth value $FALSE$
$BOOL$	BOOL	Set of boolean values $\{ TRUE, FALSE \}$
$bool(P)$	bool(P)	Convert predicate P into $BOOL$ value

B.6 AMN: Ordered Pairs & Relations

<table>
<tr><th>B Symbol</th><th>ASCII</th><th>Description</th></tr>
<tr><td>$X \times Y$</td><td>X * Y</td><td>Cartesian product of X and Y</td></tr>
<tr><td>(x, y)</td><td>(x, y)</td><td>Ordered pair</td></tr>
<tr><td>$x \mapsto y$</td><td>x |-> y</td><td>Ordered pair, (maplet)</td></tr>
<tr><td>$\mathrm{prj}_1(S,T)(x,y)$</td><td>prj1(S,T)(x, y)</td><td>Ordered pair projection function</td></tr>
<tr><td>$\mathrm{prj}_2(S,T)(x,y)$</td><td>prj2(S,T)(x, y)</td><td>Ordered pair projection function</td></tr>
<tr><td>$\mathbb{P}(X \times Y)$</td><td>POW(X * Y)</td><td>Set of relations between X and Y</td></tr>
<tr><td>$X \leftrightarrow Y$</td><td>X <-> Y</td><td>Set of relations between X and Y</td></tr>
<tr><td>$\mathrm{dom}(R)$</td><td>dom(R)</td><td>Domain of relation R</td></tr>
<tr><td>$\mathrm{ran}(R)$</td><td>ran(R)</td><td>Range of relation R</td></tr>
</table>

B.7 AMN: Relations Operators

B Symbol	ASCII	Description
$A \lhd R$	`A <| R`	Domain restriction of R to the set A
$A \ntriangleleft R$	`A <<| R`	Domain subtraction of R by the set A
$R \rhd B$	`R |> B`	Range restriction of R to the set B
$R \ntriangleright B$	`R |>> B`	Range anti-restriction of R by the set B
$R[B]$	`R[B]`	Relational Image of the set B of relation R
$R_1 \Leftarrow R_2$	`R1 <+ R2`	R_1 overridden by relation R_2
$R\,;Q$	`R ; Q`	Forward Relational composition
$\mathrm{id}(X)$	`id(X)`	Identity relation
R^{-1}	`R~`	Inverse relation
R^n	`iterate(R,n)`	Iterated Composition of R
R^+	`closure1(R)`	Transitive closure of R
R^*	`closure(R)`	Reflexive-transitive closure of R

B.8 AMN: Functions

B Symbol	ASCII	Description
$X \pfun Y$	`X +-> Y`	Partial function from X to Y
$X \rightarrow Y$	`X --> Y`	Total function from X to Y
$X \pinj Y$	`X >+> Y`	Partial injection from X to Y
$X \rightarrowtail Y$	`X >-> Y`	Total injection from X to Y
$X \psurj Y$	`X +->> Y`	Partial surjection from X to Y
$X \twoheadrightarrow Y$	`X -->> Y`	Total surjection from X to Y
$X \bij Y$	`X >->> Y`	(Total) Bijection from X to Y
$f \Leftarrow g$	`f <+ g`	Function f overridden by function g

B.9 AMN: Sequences

B Symbol	ASCII	Description
$[\,]$	`[]`	Empty Sequence
$[\,e1\,]$	`[ e1 ]`	Singleton Sequence
$[\,e1,\ e2\,]$	`[ e1, e2 ]`	Constructed (enumerated) Sequence
$\mathrm{seq}(X)$	`seq( X )`	Set of Sequences over set X
$\mathrm{seq}_1(X)$	`seq1( X )`	Set of non-empty Sequences over set X
$\mathrm{iseq}(X)$	`iseq( X )`	Set of injective Sequences over set X
$\mathrm{iseq}_1(X)$	`iseq1( X )`	Set of non-empty injective Sequences over set X
$\mathrm{perm}(X)$	`perm( X )`	Set of bijective Sequences (permutations) of set X
$\mathrm{size}(s)$	`size( s )`	Size (length) of Sequence s

B.10 AMN: Sequences Operators

B Symbol	ASCII	Description
$s \frown t$	`s^t`	Concatenation of Sequences s & t
$e \rightarrow s$	`e -> s`	Insert element e to front of sequence s
$s \leftarrow e$	`s <- e`	Append element e to end of sequence s
$rev(s)$	`rev( s )`	Reverse of sequence s
$first(s)$	`first( s )`	First element of sequence s
$last(s)$	`last( s )`	Last element of sequence s
$front(s)$	`front( s )`	Front of sequence s, excluding last element
$tail(s)$	`tail( s )`	Tail of sequence s, excluding first element
$conc(SS)$	`conc(SS)`	Concatenation of sequence of sequences SS
$s \uparrow n$	`s /|\ n`	Take first n elements of sequence s
$s \downarrow n$	`s |/ n`	Drop first n elements of sequence s

B.11 AMN: Miscellaneous Symbols & Operators

B Symbol	ASCII	Description
$var := E$	`var := E`	Assignment
$var :\in A$	`var :: E`	Nondeterministic assignment an element of set A to var
$S1 \| \| S2$	`S1 || S2`	Parallel execution of $S1$ and $S2$

B.12 AMN: Operation Statements

B.12.1 Assignment Statements

```
xx := xxval

xx, yy, zz := xxval, yyval, zzval

xx := xxval  ||  yy := yyval
```

B.12.2 Deterministic Statements

```
skip

BEGIN  S  END

PRE  PC  THEN  S  END

IF  B  THEN  S  END

IF  B  THEN  S1  ELSE  S2  END

IF  B1  THEN  S1  ELSIF B2  THEN  S2  ELSE  S3  END

CASE  E  OF
  EITHER  v1  THEN  S1
  OR      v2  THEN  S2
  OR      v3  THEN  S3
  ELSE
          S4
END

LET  xx  BE  xx = E  IN  S  END
```

B.12.3 Nondeterministic Statements

```
xx :: AA

ANY xx WHERE P THEN S END

CHOICE  S1  OR  S2  OR  S3  END

SELECT  B1 THEN S1
  WHEN  B2 THEN S2
  WHEN  B3 THEN S3
  ELSE
        S4
END
```

B.13 B Machine Clauses

```
MACHINE Name( Params )

  CONSTRAINTS    Cons

  EXTENDS        M1, M2, ...

  INCLUDES       M3, M4, ...
  PROMOTES       op1, op2, ...

  SEES           M5, M6, ...
  USES           M7, M8, ...

  SETS            Sets
  CONSTANTS       Consts
  PROPERTIES      Props

  VARIABLES       Vars
  INVARIANT       Inv
  INITIALISATION  Init

  OPERATIONS

    yy <-- op( xx ) =
           PRE  PC
           THEN Subst
           END ;
    ...
END
```