UNIVERSITY OF WESTMINSTER

FACULTY OF
SCIENCE & TECHNOLOGY
Department of Computer Science

Module:	**Formal Specification**
Module Code:	**ECSE610**
Module Leader:	P. Howells
Date:	18th January 2017
Start:	10:00
Time allowed:	2 Hours

Instructions for Candidates:

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.

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.

DO NOT TURN OVER THIS PAGE
UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO.

UNIVERSITY OF WESTMINSTER

FACULTY OF
SCIENCE & TECHNOLOGY
Department of Computer Science

Module:	**Formal Specification [MARKING SCHEME]**
Module Code:	**ECSE610**
Module Leader:	P. Howells
Date:	18th January 2017
Start:	10:00
Time allowed:	2 Hours

Instructions for Candidates:

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.
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.

DO NOT TURN OVER THIS PAGE
UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO.

Module: Formal Specification
Module Code: ECSE610
Date: 18th January 2017

Section A

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

Question 1

(a) Briefly explain what a B-Method *Abstract Machine (AM)* is. **[6 marks]**

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

- `EXTENDS`
- `INCLUDES`
- `PROMOTES`

[6 marks]

[TOTAL 12]

Module: Formal Specification **[MARKING SCHEME]**
Module Code: ECSE610
Date: 18th January 2017

Section A

Answer ALL questions from this section.

Question 1

(a) A B-Method *Abstract Machine (AM)*:

- A B AM is similar to the programming concepts of: modules, class definition (e.g. Java) or abstract data types. **[1 mark]**
- An B AM 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 AM are its: *name*, *local state*, represented by "encapsulated" variables that satisfies a *state invariant* & *initialisation* of the state. Its interface defined as a *collection of operations*, that can access & update the state variables. **[3 marks]**

[PART Total 6]

(b) B Abstract Machine *clauses*:

- `EXTENDS` – when an abstract machine *extends* another abstract machine it integrates the data of the included machine & makes *all of its operations part of its interface*. **[2 marks]**
- `INCLUDES` – when an abstract machine *includes* another abstract machine it integrates the data of the included machine & can use its operations, but *does not* make its *operations* part of its *interface*. **[2 marks]**
- `PROMOTES` – used to selectively add any of the operations of an included machine's operations to its interface, by *promoting them*, i.e. making them "visible". **[2 marks]**

[PART Total 6]

[QUESTION Total 12]

Question 2

Given the following B-method sets and function declarations, that can be used to model the properties on the Monopoly board game:

$$
\begin{aligned}
PROPERTY \ = \ \{ \ & Regent\_Street, \ Oxford\_Street, \ Bond\_Street, \\
& Park\_Lane, \ Mayfair, \ Kings\_Cross, \\
& Marylebone, \ Liverpool\_Street, \\
& Water\_Company, \ Electricity\_Company \ \}
\end{aligned}
$$

$$Green \ \in \ \mathbb{P}(PROPERTY)$$
$$Green \ = \ \{ \ Regent\_Street, \ Oxford\_Street, \ Bond\_Street \ \}$$

$$Dark\_Blue \ \in \ \mathbb{P}(PROPERTY)$$
$$Dark\_Blue \ = \ \{ \ Park\_Lane, \ Mayfair \ \}$$

$$Stations \ \in \ \mathbb{P}(PROPERTY)$$
$$Stations \ = \ \{ \ Kings\_Cross, \ Marylebone, \ Liverpool\_Street \ \}$$

$$price \ \in \ PROPERTY \nrightarrow \mathbb{N}$$
$$
\begin{aligned}
price \ = \ \{ \ & Regent\_Street \mapsto 300, \ Oxford\_Street \mapsto 300, \\
& Bond\_Street \mapsto 320, \ Park\_Lane \mapsto 350, \\
& Mayfair \mapsto 400, \ Kings\_Cross \mapsto 200, \\
& Marylebone \mapsto 200, \ Liverpool\_Street \mapsto 200 \ \}
\end{aligned}
$$

Evaluate the following expressions:

(a) $Green \ \cup \ Dark\_Blue$ **[1 mark]**

(b) $Stations \cap \{ \ Bond\_Street, \ Marylebone, \ Mayfair \ \}$ **[1 mark]**

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

(d) $price(Mayfair)$ **[1 mark]**

(e) $Green - \{ \ Bond\_Street, \ Kings\_Cross \ \}$ **[1 mark]**

(f) $\{Water\_Company, \ Electricity\_Company \ \} \times \{ \ 150 \ \}$ **[2 marks]**

(g) $\mathbb{P}(Stations)$ **[3 marks]**

[TOTAL 10]

Question 2

Evaluate the following expressions:

(a) $Green \cup Dark\_Blue$
$= \{ Regent\_Street, Oxford\_Street, Bond\_Street, Park\_Lane, Mayfair \}$
[**1 mark**]

(b) $Stations \cap \{ Bond\_Street, Marylebone, Mayfair \} = \{ Marylebone \}$
[**1 mark**]

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

(d) $price(Mayfair) = 400$
[**1 mark**]

(e) $Green - \{ Bond\_Street, Kings\_Cross \}$
$= \{ Regent\_Street, Oxford\_Street \}$ [**1 mark**]

(f) $\{Water\_Company, Electricity\_Company \} \times \{ 150 \}$
$= \{ Water\_Company \mapsto 150, Electricity\_Company \mapsto 150 \}$
[**2 marks**]

(g) $\mathbb{P}(Stations) =$
$\{ \{\}, \{Kings\_Cross\}, \{Marylebone\}, \{Liverpool\_Street\},$
$\{Kings\_Cross, Marylebone \}\{Kings\_Cross, Liverpool\_Street\},$
$\{Marylebone, Liverpool\_Street\}, \{ Kings\_Cross, Marylebone, Liverpool\_Street \} \}$
[**3 marks**]

[**QUESTION Total 10**]

Question 3

(a) Evaluation of expressions:

(i) $\mathrm{dom}(likes) = Person$ [**1 mark**]

(ii) $\mathrm{ran}(make) = Phone$ [**1 mark**]

(iii) $make [\{ Samsung, Apple \}]$
$= \{ S7edge, S5Neo, iPhone5, iPhone6 \}$ [**2 marks**]

Question 3

Given the following B declarations used to represent a group of friends and their mobile phone preferences:

$$
\begin{aligned}
Person &= \{\ Paul,\ Sue,\ Ian,\ John,\ Tom,\ Jim,\ Mary\ \} \\
Make &= \{\ HTC,\ Sony,\ Nokia,\ Samsung,\ Apple\ \} \\
Phone &= \{\ HTC10,\ Desire620,\ Xperia,\ Lumia950, \\
&\qquad S7edge,\ S5Neo,\ iPhone5,\ iPhone6\ \} \\
likes &\in Person \leftrightarrow Make \\
likes &= \{\ Paul \mapsto HTC,\ Sue \mapsto Nokia,\ Ian \mapsto Sony, \\
&\qquad John \mapsto Samsung,\ Tom \mapsto Apple, \\
&\qquad Jim \mapsto Nokia,\ Mary \mapsto Samsung\ \} \\
make &\in Make \leftrightarrow Phone \\
make &= \{HTC \mapsto HTC10,\ HTC \mapsto Desire620,\ Sony \mapsto Xperia, \\
&\qquad Nokia \mapsto Lumia950,\ Samsung \mapsto S7edge, \\
&\qquad Samsung \mapsto S5Neo,\ Apple \mapsto iPhone5, \\
&\qquad Apple \mapsto iPhone6\ \}
\end{aligned}
$$

(a) Evaluate the following expressions:

- **(i)** $\mathrm{dom}(likes)$ **[1 mark]**
- **(ii)** $\mathrm{ran}(make)$ **[1 mark]**
- **(iii)** $make\ [\ \{\ Samsung,\ Apple\ \}\ ]$ **[2 marks]**
- **(iv)** $\{\ Sue,\ Mary\ \} \lhd likes$ **[2 marks]**
- **(v)** $make \rhd \{\ HTC10,\ S7edge\ \}$ **[2 marks]**
- **(vi)** $likes \Leftarrow \{\ Paul \mapsto Samsung,\ Tom \mapsto Nokia\ \}$ **[2 marks]**
- **(vii)** $likes\ ;\ make$ **[4 marks]**

(b) Using the above definitions, define a new relation $chosefrom$, that relates people to the mobile phones they would chose to buy, based on their favourite make, e.g. since Sue likes $Nokia$ she would chose a $Lumia950$. You should give its type and its definition, that must be consistent with each individuals favourite make and the phones that that company makes.

[4 marks]
[TOTAL 18]

(iv) $\{ \ Sue, \ Mary \ \} \lhd likes$
$= \{ \ Sue \mapsto Nokia, \ Mary \mapsto Samsung \ \}$ **[2 marks]**

(v) $make \rhd \{ \ HTC10, \ S7edge \ \}$
$= \{ \ HTC \mapsto HTC10, \ Samsung \mapsto S7edge \ \}$ **[2 marks]**

(vi) $likes \oplus \{ \ Paul \mapsto Samsung, \ Tom \mapsto Nokia \ \}$
$= \{ \ Paul \mapsto Samsung, \ Sue \mapsto Nokia, \ Ian \mapsto Sony, \ John \mapsto Samsung, \ Tom \mapsto Nokia, \ Jim \mapsto Nokia, \ Mary \mapsto Samsung \ \}$ **[2 marks]**

(vii) $likes \, ; make = \{ \ Paul \mapsto HTC10, \ Paul \mapsto Desire620, \ Sue \mapsto Lumia950, \ Ian \mapsto Xperia, \ John \mapsto S7edge, \ John \mapsto S5Neo, \ Tom \mapsto iPhone5, \ Tom \mapsto iPhone6, \ Jim \mapsto Lumia950, \ Mary \mapsto S7edge, \ Mary \mapsto S5Neo \ \}$ **[4 marks]**

[PART Total 14]

(b) Definitions of $choosefrom$:

$$
\begin{aligned}
chosefrom &\in Person \leftrightarrow Phone \\
chosefrom &= likes \, ; make \\
&= \{ \ Paul \mapsto HTC10, \ Paul \mapsto Desire620, \ Sue \mapsto Lumia950, \\
&\qquad Ian \mapsto Xperia, \ John \mapsto S7edge, \ John \mapsto S5Neo, \\
&\qquad Tom \mapsto iPhone5, \ Tom \mapsto iPhone6, \ Jim \mapsto Lumia950, \\
&\qquad Mary \mapsto S7edge, \ Mary \mapsto S5Neo \ \}
\end{aligned}
$$

Type **[2 marks]** , Definition **[2 marks]**

[PART Total 4]

[QUESTION Total 18]

Question 4

Function types:

$$
\begin{aligned}
favouriteday &\in Person \leftrightarrow Day \\
working &\in Day \nrightarrow Person \\
birthday &\in Person \twoheadrightarrow Day
\end{aligned}
$$

[3 marks]

Question 4

Given the following B definitions:

$$\begin{aligned} Person &= \{\ Paul,\ Sue,\ Ian,\ John,\ Tom,\ Jim,\ Mary\ \} \\ Day &= \{\ Mon,\ Tue,\ Wed,\ Thu,\ Fri,\ Sat,\ Sun\ \} \end{aligned}$$

$$\begin{aligned} favouriteday &= \{\ Paul \mapsto Sat,\ Paul \mapsto Sun,\ Sue \mapsto Sun, \\ &\quad\ Ian \mapsto Wed,\ John \mapsto Fri,\ Tom \mapsto Tue\ \} \\ working &= \{Mon \mapsto Paul,\ Tue \mapsto Ian,\ Wed \mapsto Tom, \\ &\quad\ Thu \mapsto Paul,\ Fri \mapsto Sue\ \} \\ birthday &= \{\ Paul \mapsto Mon,\ Sue \mapsto Tue,\ Ian \mapsto Wed, \\ &\quad\ John \mapsto Thu,\ Tom \mapsto Fri, \\ &\quad\ Jim \mapsto Sat,\ Mary \mapsto Sun\ \} \end{aligned}$$

For each of the above relations $favouriteday$, $working$ and $birthday$ give its type definition and give a justification for your choice.
That is, for each one give an explanation of why you think it is just a *relation*, or a function, and what type of function, i.e. *total*, *partial*, *injective*, *surjective* or *bijective*. **[10 marks]**

[TOTAL 10]

$favouriteday$ is just a *relation* because one value from the domain $Paul$ maps to more than one value in the range, i.e. Sat and Sun. $[\mathbf{1\ mark}]$
$working$ is a *partial function* because no value from the domain is mapped to more than one value in the range; domain not equal to source so not total; range not equal to target so not surjective & not 1-to-1 so not injective. $[\mathbf{3\ marks}]$
$birthday$ is a *bijective function* because no value from the domain is mapped to more than one value in the range; domain is equal to source so total; range is equal to target so surjective & its 1-to-1 so injective. $[\mathbf{3\ marks}]$
$[\mathbf{QUESTION\ Total\ 10}]$

Section B

Answer TWO questions from this section.

Question 5

Stack B machine.

(a)
```
MACHINE Stack

  SETS
    ANSWER  = { Yes, No } ;
    MESSAGE = { PushSuccessful, ERRORStackFull,
                PopSuccessful,  ERRORStackEmpty }

  CONSTANTS
    MaxStackSize, ERROR_VALUE

  PROPERTIES
    MaxStackSize : NAT1    & MaxStackSize = 5 &
    ERROR_VALUE  : INTEGER & ERROR_VALUE  = -9999

  VARIABLES
    stack
```

Section B

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

Question 5

Write a B machine that specifies a *stack* of integers. The stack has a maximum size.
You stack B machine should include the following:

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

(b) The following stack operations, that deal with error handling where required and all non-enquiry operations must provide a report message that indicates whether the operation was successful or the reason why it failed.

- **(i)** *Push* – pushes an integer onto the stack; unless it is full. **[6 marks]**
- **(ii)** *Pop* – pops the integer at the top of the stack (i.e. removes it) and returns it; unless it is empty. If it is empty then an error value should be returned. **[7 marks]**
- **(iii)** *IsEmpty* – returns *Yes* if the stack is empty; otherwise returns *No*. **[3 marks]**

[TOTAL 25]

```
INVARIANT
  stack : seq( INTEGER ) & size( stack ) <= MaxStackSize

INITIALISATION
  stack := []     /* Empty stack */
```

Roughly award: SETS **[2 marks]** , CONSTANTS & PROPERTIES **[3 marks]** , VARIABLES & INVARIANT **[3 marks]** , INITIALISATION **[1 mark]** .

[PART Total 9]

(b) (i) In this version the "top" of the stack is the front of the `stack` sequence, but okay if the end. Might use strings for reporting rather than MESSAGE.

```
OPERATIONS

    report <-- Push( num ) =
        PRE
            report : MESSAGE & num : INTEGER
        THEN
            IF ( size( stack )  < MaxStackSize )
            THEN
                stack  := num -> stack       ||
                report := PushSuccessful
            ELSE
                report := ERRORStackFull
            END
        END ;
```

[PART Total 6]

(ii)

```
    report, topnum <-- Pop =
        PRE
            report : MESSAGE & topnum : INTEGER
        THEN
            IF ( stack /= [] )
            THEN
                stack  := tail( stack )   ||
                report := PopSuccessful   ||
                topnum := first( stack )
            ELSE
                report := ERRORStackEmpty ||
```

Section B

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

Question 5

Write a B machine that specifies a *stack* of integers. The stack has a maximum size.
You stack B machine should include the following:

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

(b) The following stack operations, that deal with error handling where required and all non-enquiry operations must provide a report message that indicates whether the operation was successful or the reason why it failed.

(i) *Push* – pushes an integer onto the stack; unless it is full. **[6 marks]**

(ii) *Pop* – pops the integer at the top of the stack (i.e. removes it) and returns it; unless it is empty. If it is empty then an error value should be returned. **[7 marks]**

(iii) *IsEmpty* – returns *Yes* if the stack is empty; otherwise returns *No*. **[3 marks]**

[TOTAL 25]

```
                topnum := ERROR_VALUE
            END
        END ;
```

[PART Total 7]

(iii)

```
    answer <-- IsEmpty =
        PRE
            answer : ANSWER
        THEN
            IF ( stack = [] )
            THEN
                answer := Yes
            ELSE
                answer := No
            END
        END

END  /* Stack */
```

[PART Total 3]

[PART Total 16]

[QUESTION Total 25]

Question 6

`HotelBooking` B machine.

(a) (i) `roomsize : ROOM --> NAT1`

Every room must have a maximum size, even though a room may contain less than the maximum it is not sensible to use a relation. **[2 marks]** If a *surjective* function was used, then there would have to be a room that accommodated every possible number of guests. **[2 marks]**

[SUBPART Total 4]

(ii) `guests : ROOM <-> GUEST`

The relationship between a room & guests is one-to-many, since not all rooms are singles.

[SUBPART Total 2]

Question 6

Appendix A contains the `HotelBooking` B machine, this specifies a simple hotel room booking system.
The hotel's room booking system holds the following information about its rooms and guests:

- The size of each room, i.e. maximum number of occupants, (`roomsize`).
- The status of each room, i.e. whether its occupied by guests or vacant, (`status`).
- The guests currently in each occupied room, (`guests`).
- The person who reserved a particular room, (`reservation`).

The system provides the following operations:

- `bookroom` – a person to book one of the hotel's rooms.
- `guestsCheckin` – one or more guests to check into one of the booked rooms.
- `guestsCheckout` – the guests staying in one of the booked rooms.

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

(a) With reference to the `PROPERTIES` and `INVARIANT` clauses answer the following questions using "plain English" only.

(i) `roomsize : ROOM --> NAT1`
Explain why it makes sense to use a *total function* (`-->`, $\rightarrow$) in the definition of `roomsize`, rather than a *relation*. In addition, explain why it would not make sense to use a *surjective* function. **[4 marks]**

(ii) `guests : ROOM <-> GUEST`
Explain why it makes sense to use a *relation* (`<->`, $\leftrightarrow$) to represent the guests staying in the rooms. **[2 marks]**

(iii) `reservation : GUEST >+> ROOM`
Explain what this invariant means in relation to people reserving rooms. **[3 marks]**

(iv)
```
!(rm).( rm : dom(guests) =>
          ( card( guests[ { rm } ] ) <= roomsize(rm) ) )
```
Explain what this invariant means. **[3 marks]**

[Continued Overleaf]

(iii) `reservation : GUEST >+> ROOM`

A guest can reserve only one room at a time & a room can not be reserved by more than one person at a time.

[SUBPART Total 3]

(iv)
```
!(rm).( rm : dom(guests) =>
        ( card( guests[ { rm } ] ) <= roomsize(rm) ) )
```

The number of guests in any occupied room does not exceed the maximum number of occupants for the room.

[SUBPART Total 3]

[PART Total 12]

(b) **(i)** Preconditions `bookroom`: the known person does not already have a booking & the known room has not been booked.

[SUBPART Total 2]

(ii) Preconditions `guestsCheckin`: the known room has been booked & is vacant; there is at least one guest, but not more than the room can accommodate & all of them are known.

[SUBPART Total 4]

(iii) Preconditions `guestsCheckout`: the known room must be occupied by guests.

[SUBPART Total 1]

[PART Total 7]

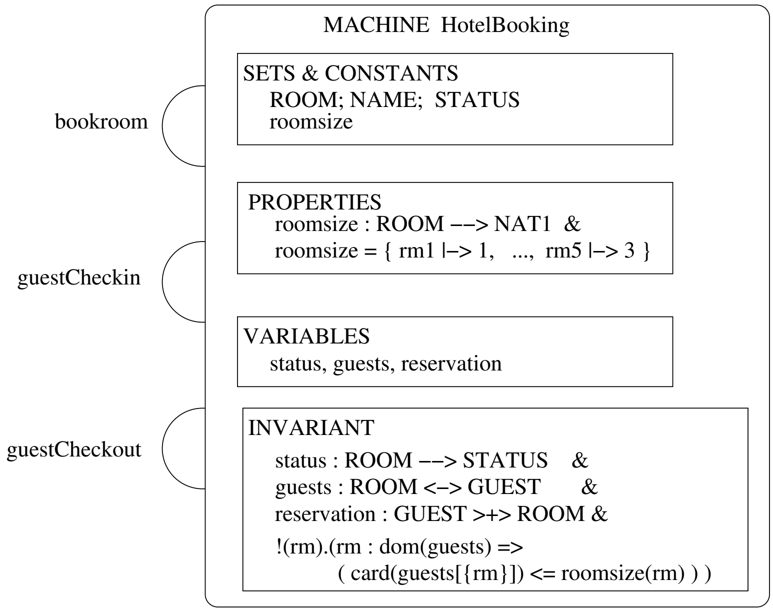
(c) `HotelBooking` machine Structure Diagram.

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

- **(i)** `bookroom` **[2 marks]**
- **(ii)** `guestsCheckin` **[4 marks]**
- **(iii)** `guestsCheckout` **[1 mark]**

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

[TOTAL 25]



Internal structure **[5 marks]** , Operations **[1 mark]** .

[PART Total 6]

[QUESTION Total 25]

Question 7

(a) **(i)** The invalid states are those that do not satisfy the B machine's invariant. The valid states are those that do satisfy the B machine's invariant. The initial states define the set of possible starting states for the B machine, i.e. its state variables; they must also be valid states. **[3 marks]**

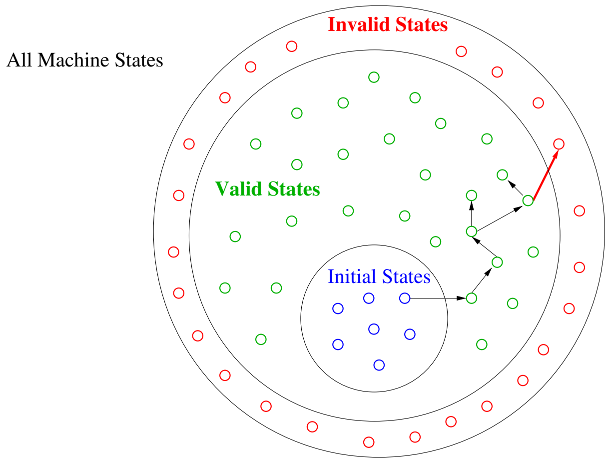
The *state invariant* is the constraints & properties (defined in a machine) that the states of the system/machine are required to satisfy during its lifetime, i.e. all of the states it passes through during its execution should satisfy them. **[1 mark]**

[SUBPART Total 4]

(ii) *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

Question 7

(a) The following diagram represents all of the possible states that a system could be in.



With reference to the above diagram:

(i) Explain the relationship between the three kinds of states and a B machine. **[4 marks]**

(ii) Explain how a B machine ensures that its operations transform its state from one valid state to another valid state. **[4 marks]**

(b) **(i)** When a B machine specification is produced, what are the particular claims that are (implicitly) made about it? **[5 marks]**

(ii) What is a "proof obligation"? **[2 marks]**

(iii) The following *proof obligations* must be proved to demonstrate that a B machine makes sense and is correct:

$$
\begin{array}{ll}
(PO1) & \exists\ Sets,\ Constants \cdot (\ Properties\ ) \\
(PO2) & Properties\ \Rightarrow\ \exists\ Vars \cdot (\ Invariant\ ) \\
(PO3) & Properties\ \wedge\ Invariant\ \wedge\ PreCondition \\
& \qquad \Rightarrow\ [\ Substitution\ ]Invariant
\end{array}
$$

Explain what property about a B machine each of these proof obligation are intended to ensure. **[10 marks]**

[TOTAL 25]

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]**

[SUBPART Total 4]

[PART Total 8]

(b) (i) B machine specification claims:

- It makes sense & is coherent.
- The deferred sets & constants can be instantiated.
- There are states that meet the invariant (otherwise it cannot be implemented).
- The initialisation establishes the invariant.
- The operations preserve the invariant.

[SUBPART Total 5]

(ii) A "proof obligation" is a predicate that captures an essential property, e.g. a claim given in part (a), about a certain aspect of a B machine. The proof obligation must be proved true to guarantee that the B machine is consistent, correct and implementable. **[2 marks]**

[SUBPART Total 2]

(iii)

$$(PO1) \quad \exists\, Sets,\ Constants \cdot (\ Properties\ )$$

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

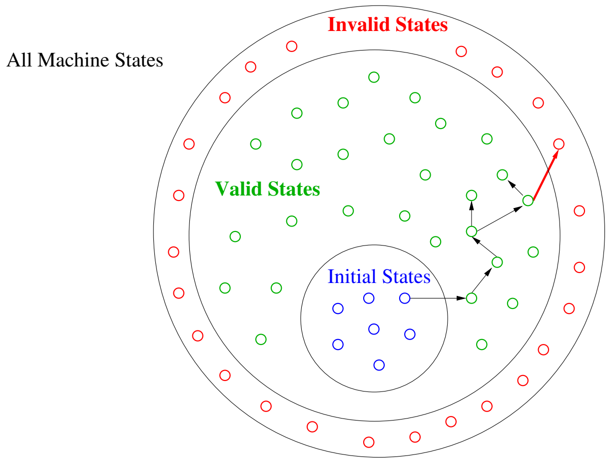
It expresses the property that for the machine to have any valid values for its sets & constants at all, it must always be possible, to find appropriate sets & constants. **[2 marks]**

$$(PO2) \quad Properties \Rightarrow \exists\, Vars \cdot (\ Invariant\ )$$

This is the *Initialisation Proof Obligation*, it is concerned with the initialisation of the machine's state variables ($Vars$). That there

Question 7

(a) The following diagram represents all of the possible states that a system could be in.



With reference to the above diagram:

(i) Explain the relationship between the three kinds of states and a B machine. **[4 marks]**

(ii) Explain how a B machine ensures that its operations transform its state from one valid state to another valid state. **[4 marks]**

(b) **(i)** When a B machine specification is produced, what are the particular claims that are (implicitly) made about it? **[5 marks]**

(ii) What is a "proof obligation"? **[2 marks]**

(iii) The following *proof obligations* must be proved to demonstrate that a B machine makes sense and is correct:

$$(PO1) \quad \exists\, Sets,\ Constants \cdot (\ Properties\ )$$
$$(PO2) \quad Properties \Rightarrow \exists\, Vars \cdot (\ Invariant\ )$$
$$(PO3) \quad Properties \wedge Invariant \wedge PreCondition \Rightarrow [\ Substitution\ ]Invariant$$

Explain what property about a B machine each of these proof obligation are intended to ensure. **[10 marks]**

[TOTAL 25]

is at least one 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). **[2 marks]**

It expresses the property that given the sets & constants the initialisation of the state variables establishes the invariant Inv, i.e. the machine's initial state satisfies the invariant. **[2 marks]**

$$(PO3) \quad Properties \wedge Invariant \wedge PreCondition \\ \Rightarrow [\ Substitution\ ]Invariant$$

This is the *Operation Proof Obligation* it is concerned with proving that the AMN specification of an operation:

```
PRE  PreCondition THEN Substitution END
```

preserves the invariant when it is invoked when the precondition is true. **[2 marks]**

If the machine is in a state in which the invariant & properties hold, & 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. **[2 marks]**
[SUBPART Total 10]

[QUESTION Total 25]

Appendix A. Hotel Booking B Machine

The following B Machine – `HotelBooking`, specifies a simple Hotel room booking system.

```
MACHINE HotelBooking

  SETS
    ROOM   = { rm1, rm2, rm3, rm4, rm5 } ;
    GUEST  = { Ian, Sue, Tom, Jim, Bill, Eddy, Rob } ;
    STATUS = { Occupied, Vacant }

  CONSTANTS
    roomsize

  PROPERTIES
    roomsize : ROOM --> NAT1  &
    roomsize = {  rm1 |-> 1, rm2 |-> 1, rm3 |-> 2,
                  rm4 |-> 2, rm5 |-> 3  }

  VARIABLES
    status,
    guests,
    reservation

  INVARIANT
    status      : ROOM --> STATUS  &
    guests      : ROOM <-> GUEST   &
    reservation : GUEST >+> ROOM
    &
    !(rm).( rm : dom(guests)  =>
             ( card( guests[ { rm } ] )  <=  roomsize(rm) )  )

  INITIALISATION
    status      := ROOM * { Vacant } ||
    guests      := {}                ||
    reservation := {}

```

[Continued on next page.]

```
OPERATIONS

  bookroom( person, rm ) =
  PRE
      ( person : GUEST ) & ( rm : ROOM ) &
      ( person /: dom(reservation) )     &
      ( rm /: ran(reservation) )
  THEN
        reservation := reservation <+ { person |-> rm }
  END ;


  guestsCheckin( rm, people ) =
  PRE
     ( rm : ROOM ) & ( people <: GUEST ) &
     ( rm : ran(reservation) )           &
     ( status(rm) = Vacant )             &
     ( people /= {} )                    &
     ( card(people) <= roomsize(rm) )
  THEN
      guests := guests <+ ( { rm } * people ) ||
      status := status <+ { rm |-> Occupied }
  END ;


  guestsCheckout( rm ) =
  PRE
     ( rm : ROOM ) & ( status(rm) = Occupied )
  THEN
      status      := status <+ { rm |-> Vacant }  ||
      guests      := { rm } <<| guests            ||
      reservation := reservation |>> { rm }
  END

END /* HotelBooking */
```

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
$\mathbb{P}_1(A)$	`POW1(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

B Symbol	ASCII	Description
$A \cup B$	`A \/ B`	Union of A and B
$A \cap B$	`A /\ B`	Intersection of A and B
$A - B$	`A - B`	Set subtraction of A and B
$\bigcup AA$	`union( AA )`	Generalised union of set of sets AA
$\bigcap AA$	`inter( AA )`	Generalised intersection of set of sets AA
$A \subseteq B$	`A <: B`	A is a subset of or equal to B
$A \not\subseteq B$	`A /<: B`	A is not a subset of or equal to B
$A \subset B$	`A <<: B`	A is a strict subset of B
$A \not\subset B$	`A /<<: B`	A is not a strict subset of B
$\{\ x \mid x \in TS \wedge C\ \}$	`{ x | x : TS & C }`	Set comprehension

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

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

B.7 AMN: Relations Operators

B Symbol	ASCII	Description
$A \lhd R$	`A <| R`	Domain restriction of R to the set A
$A \mathbin{\lhd\mkern-14mu-} R$	`A <<| R`	Domain subtraction of R by the set A
$R \rhd B$	`R |> B`	Range restriction of R to the set B
$R \mathbin{\rhd\mkern-14mu-} B$	`R |>> B`	Range anti-restriction of R by the set B
$R[B]$	`R[B]`	Relational Image of the set B of relation R
$R_1 \mathbin{\lhd\mkern-9mu+} R_2$	`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
$X \mathrel{+\mkern-12mu\to} Y$	`X +-> Y`	Partial function from X to Y
$X \to Y$	`X --> Y`	Total function from X to Y
$X \mathrel{\rightarrowtail\mkern-15mu+} Y$	`X >+> Y`	Partial injection from X to Y
$X \rightarrowtail Y$	`X >-> Y`	Total injection from X to Y
$X \mathrel{+\mkern-12mu\twoheadrightarrow} Y$	`X +->> Y`	Partial surjection from X to Y
$X \twoheadrightarrow Y$	`X -->> Y`	Total surjection from X to Y
$X \mathrel{\rightarrowtail\mkern-18mu\twoheadrightarrow} Y$	`X >->> Y`	(Total) Bijection from X to Y
$f \mathbin{\lhd\mkern-9mu+} g$	`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

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

B.11 AMN: Miscellaneous Symbols & Operators

B Symbol	ASCII	Description
$var := E$	`var := E`	Assignment
$var :\in A$	`var :: E`	Nondeterministic assignment of 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
```