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 **[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

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**]

(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 $chosefrom$:

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

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

[PART Total 4]

[QUESTION Total 18]

Question 4

Function types:

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

[3 marks]

$favouriteday$ 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. **[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. **[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. **[3 marks]**
[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
```

```
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 ||
```

```
                    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**]

(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.

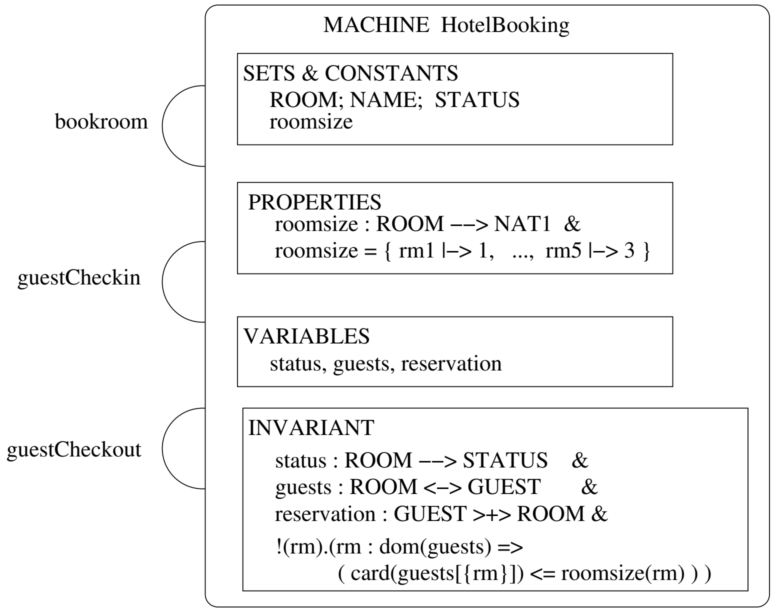
[SUBPART Total 4]

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

[SUBPART Total 1]

[PART Total 7]

(c) `HotelBooking` machine Structure Diagram.



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 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 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]