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.

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

Evaluate the following expressions:

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

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

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

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

(e) $\text{ran}(edges) = \{ 1, 3, 4, 5, 6 \}$
[**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) $(Quadrilaterals \cup NonPolygons) \lhd\!\!\!\!- edges$
$= \{ Pentagon \mapsto 5,\ Hexagon \mapsto 6 \}$
[**3 marks**]

[**QUESTION Total 15**]

Question 3

(a) Evaluate the following expressions:

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

(ii) $R_3 \lhd \{\ 0 \mapsto w,\ 4 \mapsto a\ \}\ =\ \{\ 0 \mapsto w,\ 1 \mapsto x,\ 2 \mapsto y,\ 4 \mapsto a\ \}$ **[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.

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
```

```
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
```

```
        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**]

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* >+> ($\nrightarrow$) 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.

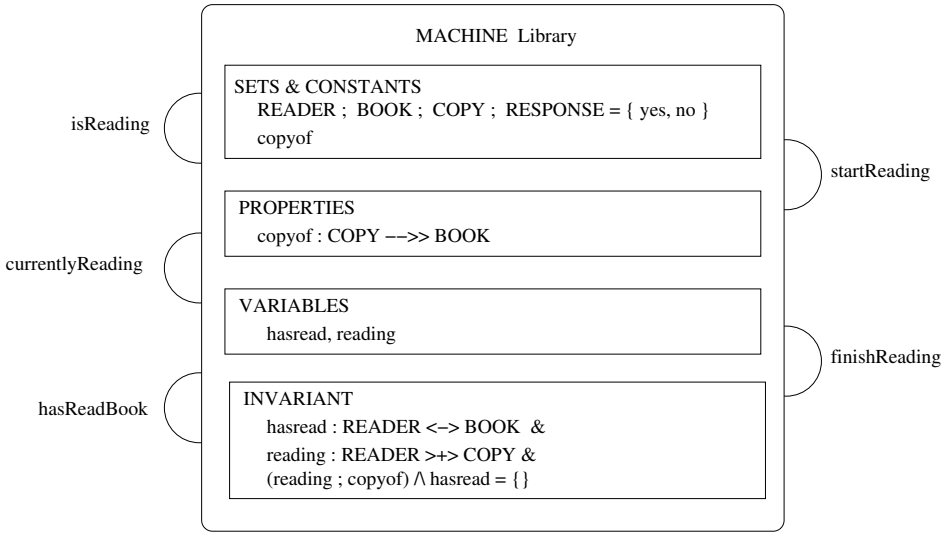
[**SUBPART Total 3**]

(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.



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

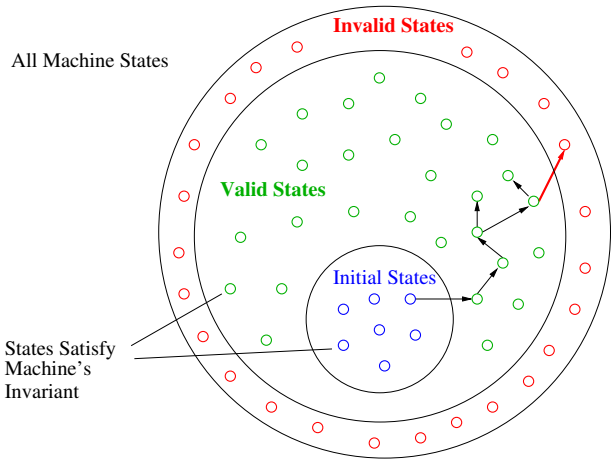
[**PART Total 6**]

[**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.



[**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

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**]