UNIVERSITY OF WESTMINSTER

MARKING SCHEME
DO NOT COPY

SCHOOL OF
COMPUTER SCIENCE & ENGINEERING

Module Title: **Reasoning about Programs**

Module Code: **6SENG001W, 6SENG003C**

Exam Period: **January 2020**

Time Allowed: **2 Hours**

INSTRUCTIONS FOR CANDIDATES

PLEASE WRITE YOUR STUDENT ID CLEARLY AT THE TOP OF EACH PAGE.

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.

In section B, only the TWO questions with the HIGHEST MARKS will count towards the FINAL MARK for the EXAM.

THIS PAPER MUST NOT BE TAKEN OUT OF THE EXAMINATION ROOM

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

Section A

Answer ALL questions from this section.

Question 1

Scottish Islands expressions:

(a) $Outer\_Hebrides \cap \{\ Skye, South\_Uist, Mull, Benbecula\ \}$
$= \{\ South\_Uist, Benbecula\ \}$ **[1 mark]**

(b) $Inner\_Hebrides - \{\ North\_Uist, Islay, Jura\ \}$
$= \{\ Skye, Mull\ \}$ **[2 marks]**

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

(d) $Scottish\_Islands \cap \mathrm{dom}(highest\_point)$
$= Scottish\_Islands$ **[1 mark]**

(e) $\mathrm{ran}(highest\_point)$
$= \{\ 124, 347, 491, 620, 785, 799, 993, 966\ \}$ **[1 mark]**

(f) $highest\_point(Lewis\_and\_Harris)$
$= 966$ **[1 mark]**

(g) $Inner\_Hebrides \lhd highest\_point$
$= \{\ Skye \mapsto 993,\ Islay \mapsto 491,\ Mull \mapsto 966,\ Jura \mapsto 785\ \}$
[2 marks]

(h) $highest\_point \rhd 0..900$
$= \{\ Skye \mapsto 993,\ Mull \mapsto 966\ \}$ **[3 marks]**

(i) $\mathbb{P}(\ \{\ Skye, Mull, Jura\ \}\ )$

$$= \{\ \{\},\ \{\ Skye\ \}, \{\ Mull\ \}, \{\ Jura\ \},\ \{\ Skye, Mull\ \}, \{\ Skye, Jura\ \}, \{\ Mull, Jura\ \}\ \{\ Skye, Mull, Jura\ \}\ \}$$

[3 marks]

[QUESTION Total 15]

Question 2

(a) Can compose $R ; Q$. Since the *target* of R is the *same type* as the *source* of Q, i.e. $\mathbb{N}$, the two relations can be composed. **[2 marks]**

(b) The composition of R and Q:

$$R ; Q \ : \ LETTER \leftrightarrow COLOUR$$
$$R ; Q \ = \ \{ \ (a, \ red), \ (a, \ blue), \ (b, \ blue), \ (c, \ green), \ (e, \ purple) \ \}$$

[3 marks]

(c) Relations or Functions:

R is just a relation because one value in the domain a maps to two values in the range 1 and 2, e.g. $(a, \ 1), \ (a, \ 2)$.

Q is also a function, because no value in the domain maps to two or more values in the range. (It is a partial injection.) **[2 marks]**

[QUESTION Total 7]

Question 3

(a)

- 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, *state invariant* defines constraints on the state variables, that define what the valid states are. the variable's values must always satisfy the *state invariant*, *collection of operations*, that can access & update the state variables, but must always ensure that the *after state* of the operation satisfies the *state invariant*. **[5 marks]**
- An B Machine is similar to the programming concepts of: modules, class definition (e.g. Java) or abstract data types. **[1 mark]**
- The most obvious logical component of a B Machine that is not represented in a Java class is the `INVARIANT` clause that defines the *state invariant*. Java classes do not have any built in language feature that defines when an instance of a class is "valid". **[2 marks]**

[PART Total 10]

(b) See Figure 1. Three categories of system states are: *valid* states, *initial* or *start* states & *error* or *invalid* states. **[1 mark]**

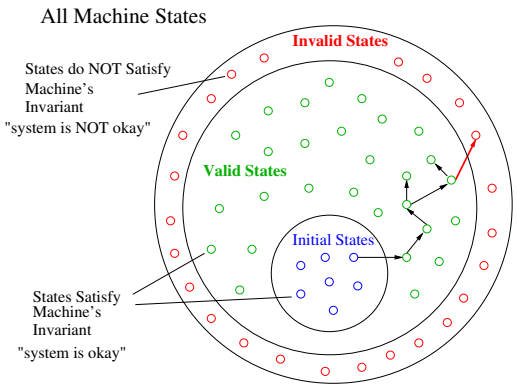

Figure 1: Possible B Machine States

[4 marks]

The *valid* states are those that satisfy the *state invariant*. The *invalid* states are those that do not satisfy the *state invariant*. The *state invariant* is the constraints & properties that the states of the machine must satisfy during its lifetime. Defined in the `INVARIANT` clause **[2 marks]**

The *initial state(s)* are the set of possible starting states of the machine. Any initial state must also be a valid state, i.e. one that satisfies the state invariant. Defined in the `INITIALISATION` clause. **[1 mark]**

[PART Total 8]

[QUESTION Total 18]

Question 4

Marking Scheme for Hoare Logic & Program Verification.

(a) The Hoare triple

$$[y > z + 1]\ x := z\ [x < y]$$

means that executing the instruction $x := z$ (i.e. assigning the value of z to x), starting from a state in which y is greater than $z + 1$, leads to a state in which x is less than y. **[2 marks]**

[PART Total 2]

(b) **(i)** $[x < y]\ x := y\ [false]$ is invalid. **[1 mark]** Counterexample: Starting in a state with $x = 1, y = 2$ leads to a state with $x = 1, y = 1$, which (like all other states) does not satisfy $false$. **[1 mark]**

[SUBPART Total 2]

(ii) $[x < y]\ x := y\ [true]$ is valid, since any post-state satisfies $true$. **[2 marks]**

[SUBPART Total 2]

(iii) $[x < y]\ y := y + 1\ [x < y + 1]$ is valid: the weakest pre-condition is $x < y + 2$, which follows from the given pre-condition. **[2 marks]**

[SUBPART Total 2]

(iv) $[x = 3]\ x := y\ [y = 3]$ is invalid. **[1 mark]** Counterexample: Starting in a state with $x = 3, y = 0$ leads to a state with $x = 0, y = 0$, which does not satisfy $y = 3$. **[1 mark]**

[SUBPART Total 2]

[PART Total 8]

[QUESTION Total 10]

Section B

Answer TWO questions from this section.

Question 5

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

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

Combines **MARKING SCHEME FOR QUESTION**
& `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 PostOfficeQueue`

```
SETS
  CUSTOMER ;

  ANSWER = { Yes, No } ;

  REPORT = { Customer_Joined_Queue,
             ERROR_Queue_is_Full,
             Customer_Served,
             ERROR_Queue_Empty    }

CONSTANTS
  MaxPOqueueLength, EmptyQueue, Nobody

PROPERTIES
  MaxPOqueueLength : NAT1  &  MaxPOqueueLength = 5  &
  EmptyQueue : iseq( CUSTOMER ) &  EmptyQueue = [] &
  Nobody : CUSTOMER

VARIABLES
  POqueue
```

```
INVARIANT
  POqueue : iseq( CUSTOMER ) & size( POqueue ) <= MaxPOqueueLength
  & Nobody /: ran( POqueue )

INITIALISATION
  POqueue := EmptyQueue
```

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

[**PART Total 9**]

(b) OPERATIONS

```
report <-- JoinPOQueue( customer ) =
    PRE
        customer : CUSTOMER & customer /: ran( POqueue )
        & customer /= Nobody & report : REPORT
    THEN
        IF ( size( POqueue )  < MaxPOqueueLength )
        THEN
            POqueue  := POqueue <- customer       ||
            report := Customer_Joined_Queue
        ELSE
            report := ERROR_Queue_is_Full
        END
    END ;
```

[Subpart (b.i) 7 marks]

```
report, nextcustomer <-- GotoCounter =
    PRE
        nextcustomer : CUSTOMER & report : REPORT
    THEN
        IF ( POqueue /= EmptyQueue )
        THEN
            nextcustomer := first( POqueue )  ||
            POqueue  := tail( POqueue )       ||
            report := Customer_Served
        ELSE
            nextcustomer := Nobody        ||
            report := ERROR_Queue_Empty
         END
```

```
        END ;
```

[Subpart (b.ii) 6 marks]

```
    answer <-- CustomersWaiting =
        PRE
            answer : ANSWER
        THEN
            IF ( POqueue = EmptyQueue )
            THEN
                 answer := No
            ELSE
                 answer := Yes
            END
        END

END  /* PostOfficeQueue */
```

[Subpart (b.iii) 3 marks]

[**PART Total 16**]

[**QUESTION Total 25**]

Question 6

See the `Club` B machine given in the exam paper's Appendix A.

(a) The `Club`'s invariants:

`queuetotal < capacity` – the length of club's waiting list is less than the maximum number of members allowed. [**2 marks**]

`members <: NAME` – the club's members is a collection/list of names. [**1 mark**]

`waiting <: NAME` – the people on the club's waiting list are a collection/list of names. [**1 mark**]

`members /\ waiting = {}` – no one can be a member & on the waiting list to join the club. [**2 marks**]

`card(members) <= capacity` – the club has a maximum membership (list of member's names). **[2 marks]**

`card(waiting) <= queuetotal` – the club has a maximum membership waiting list & given the first invariant it cannot be more than the maximum club size. **[2 marks]**

[PART Total 10]

(b) The meaning of the *preconditions* for the operations:

(i) `join` preconditions – the parameter `newmember` is the name of someone who is currently on the waiting list; the club is not full yet, i.e. hasn't reach maximum members. **[2 marks]**

(ii) `join_queue` preconditions – the parameter `newmember` is the name of someone who is currently not a club member or on the waiting list; the club's waiting list is not full yet, i.e. hasn't reach maximum limit. **[4 marks]**

(iii) `remove` preconditions – the parameter `member` is the name of someone who is currently a member of the club. **[2 marks]**

[PART Total 8]

(c) The `Club` machine's Structure Diagram – Figure 2. Internal structure

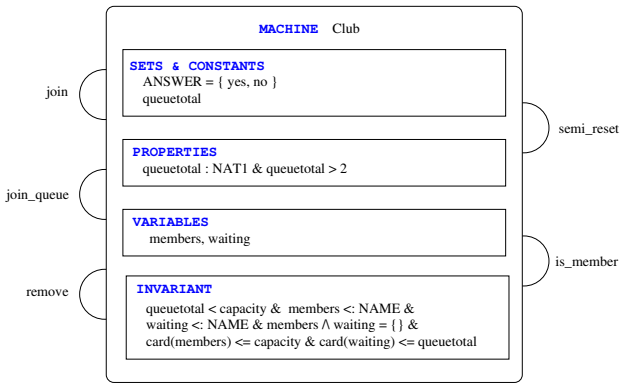

Figure 2: `Club` machine's Structure Diagram.

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

[PART Total 6]

[QUESTION Total 25]

Question 7

Marking Scheme for Hoare Logic & Program Verification.

(a) The intermediate assertions are

1. $0 = y$ **[2 marks]**
2. $0 = y + 1$ **[2 marks]**
3. $x = x + y + 1$ or $0 = y + 1$ **[2 marks]**

[PART Total 9]

(b) The invariant and intermediate assertions are:

- Invariant: $x = u * y \ \& \ y \leq v$ **[4 marks]**
- Assertion 1: $x = u * y \ \& \ y < v$ **[4 marks]**
- Assertion 2: $x = u*(y+1) \ \& \ y+1 \leq v$ (or e.g. $x = u*y+u \ \& \ y < v$) **[4 marks]**
- Assertion 3: $x = u * y \ \& \ y \leq v$

[PART Total 16]

[QUESTION Total 25]