6SENG005W Formal Methods: In-class test

Question 1

Given the B definitions:

$$\begin{aligned}
ANIMAL &= \{Cat, Dog, Hamster, Hyena, Wolf, Turtle\}\\
FOOD &= \{Meat, Veg, Cheese\}\\
Pet &\subseteq ANIMAL\\
Pet &= \{Cat, Dog, Hamster, Turtle\}\\
Mammal &\subseteq ANIMAL\\
Mammal &= \{Cat, Dog, Hamster, Hyena, Wolf\}\\
eats &\in ANIMAL \to FOOD\\
eats &= \{Cat \mapsto Meat, Dog \mapsto Meat, Hamster \mapsto Veg, Hyena \mapsto Meat, Wolf \mapsto Meat, Turtle \mapsto Veg\}
\end{aligned}$$

Evaluate the following expressions (2 marks each):

1. $Mammal \cap Pet$
2. $card(Pet)$
3. $eats(Hamster)$
4. $ANIMAL \setminus Mammal$
5. $ran(eats)$

Example Answer

1. $Mammal \cap Pet = \{Cat, Dog, Hamster\}$
2. $card(Pet) = 4$
3. $eats(Hamster) = Veg$
4. $ANIMAL \setminus Mammal = \{Turtle\}$
5. $ran(eats) = \{Meat, Veg\}$

Question 2

Consider this section of a B specification:

```
VARIABLES
   lower, upper
INVARIANT
   lower : NAT & upper : NAT & lower <= upper
INITIALISATION
   lower := 0 || upper := 1
```

In your own words, explain the meaning of this section and each part within it.

Example Answer

This section defines the variables of a machine and their values. (2 marks)

It first lists the names of the variables, (2 marks)

then gives the invariant which specifies their types and the additional ordering constraint. (3 points)

The initialisation sets up the variables' initial values and ensures that the invariant is initially true. (3 marks)

Question 3

Consider the relation **near** and function **half** given by

```
near : NAT <-> NAT
near = {0|->1, 1|->0, 1|->2, 2|->1, 2|->3, 3|->2, 3|->4, 4|->3}
half : NAT >+> NAT
half = {0|->0, 2|->1, 4|->2}
```

1. Is half injective? surjective? partial? What does this mean? (3 marks)
2. Evaluate near ; half. (3 marks)
3. Evaluate near /\ half. (2 marks)
4. Evaluate¬half. (2 marks)

0.1 Example Answer

1. The function is injective since no function value occurs more than once. (1 mark)
   It is not surjective since some members of NAT (e.g. 4) never occur as function values. (1 mark)
   It is partial since some members of NAT (e.g. 3) have no function value. (1 mark)
2. near ; half = {1|->0, 1|->1, 3|->1, 3|->2} (3 marks)
3. near /\ half = {2|->1} (2 marks)
4. half = {0|->0, 1|->2, 2|->4} (2 marks)

Question 4

Write a B machine Fruits which satisfies the following requirements:

- There are the following fruits that we can buy: Apples, Bananas, Lemons, Pineapples.
- Each fruit has a fixed price (choose some reasonable amounts, the exact values don't matter).
- We need to keep track of our remaining budget.
- There needs to be an operation for buying a fruit (updating the budget), provided that we have enough money left.

Example Answer

Note that the variable names, prices etc are not specified in the question and can differ.

2 marks for properties, buy operation; 1 mark for all other parts and main machine structure

```
MACHINE Fruits

   SETS
      FRUIT = { Apple, Banana, Lemon, Pineapple }

   CONSTANTS
      FruitPrice

   PROPERTIES
```

```
      FruitPrice : FRUIT -> NAT
      &
      FruitPrice = { Apple|->20, Banana|->25, Lemon|->15, Pineapple|->40 }

   VARIABLES
      budget

   INVARIANT
      budget : NAT

   INITIALISATION
      budget := 100

   OPERATIONS
      buy(fruit) =
      PRE
         fruit : Fruit & FruitPrice(fruit) <= budget
      THEN
         budget := budget - FruitPrice(fruit)
      END
END
```

Question 5

Extend the machine from question 4 to keep track of how many of each fruit we have bought. Give the updated VARIABLES, INVARIANT, INITIALISATION and OPERATIONS clauses.

Example Answer

Note that inventory could also be defined as a partial function.

2 marks for variables, invariant; 3 marks for initialisation, buy operation.

```
   VARIABLES
      budget, inventory

   INVARIANT
      budget : NAT &
      inventory : FRUIT -> NAT

   INITIALISATION
      budget := 100 || inventory := FRUIT * {0}

   OPERATIONS
      buy(fruit) =
      PRE
         fruit : Fruit & FruitPrice(fruit) <= budget
      THEN
         budget := budget - FruitPrice(fruit) || inventory = inventory <+ {fruit|->inventory(fruit)+1}
      END
```

Question 6

Consider this B machine:

```
MACHINE numbers

   VARIABLES
      left, right

   INVARIANT
```

```
      left <: NAT & right <: NAT & left <: right

   INITIALISATION
      left := {} || right := {1}

  OPERATIONS
     extend =
     BEGIN
        left := left \/ {card(right)} || right := right \/ {card(left)}
     END

END
```

Give the state after the machine:

1. after initialisation, (2 marks)
2. after executing extend, (3 marks)
3. after executing extend again. (3 marks)

In which of these states is the invariant satisfied? (2 marks)

Example Answer

The states are

1. left = {}, right = {1}
2. left = {1}, right = {0,1}
3. left = {1,2}, right = {0,1}

The invariant is satisfied in the first two states, but not the third.

Question 7

The following is an example of the structure of an abstract machine:

```
MACHINE MyMachine

SETS
   S1 = {aa,bb}, S2 = {}

CONSTANTS
   C1

PROPERTIES
   C1 <: S1 & C1 = {aa}

VARIABLES
   xx, yy

INVARIANT
   xx : S1 & yy : C1 & xx/=yy

INITIALISATION
   xx := bb || yy := aa

END
```

Explain the role that each part plays in specifying the machine.

Example Answer

2 points each for:

- The sets define underlying data types used by the other parts (1 mark)
- The constants are derived types or values defined using the sets (including pre-defined ones like NAT) (2 marks)
- The properties give the actual definitions of the constants (2 marks)
- The variables give values (possibly including sets, relations etc) which can change over time (2 marks)
- The invariant describes properties that the variable values should always satisfy (2 marks)
- The initialisation gives starting values to all variables (1 mark)

Question 8

Determine the missing assertions for the following program using pre-condition propagation.

```
    [$Assertion$ 1]
    $x := 0;$
    [$Assertion$ 2]
    $x := y;$
    [$Assertion$ 3]
    $y := y + 1$
    [$x + y = 5$]
```

Example answer

```
[$y = 2$]        ($3$ $marks$)
$x := 0;$
[$y = 2$]        ($4$ $marks$)
$x := y;$
[$x + y = 4$]    ($3$ $marks$)
$y := y + 1$
[$x + y = 5$]
```

Question 9

Determine the missing assertions for the following program using pre-condition propagation.

```
    [$Assertion$ 1]
    $IF\ x > y\ THEN$
        [$Assertion$ 2]
        $x := x - y$
        [$Assertion$ 3]
    $ELSE$    [$Assertion$ 4]
        $x := y - x$
        [$Assertion$ 5]
    $END$
    [$x \geq 0$]
```

Example answer

```
[$true$]
$IF\ x > y\ THEN$
    [$x \geq y$]
    $x := x - y$
    [$x \geq 0$]
$ELSE$    [$y \geq x$]
    $x := y - x$
    [$x \geq 0$]
$END$
[$x \geq 0$]
```

(2 marks per assertion)

Question 10

Which of the following Hoare triples are valid? Provide counterexamples for the invalid ones.

1. $[x > y]$
   $y := x$
   $[x > x]$

2. $[x < 5]$
   $y := x$
   $[y < 5]$

3. $[x = y + z]$
   $y := z$
   $[x = z + z]$

Example answer

The second triple is valid, the others are invalid. (4 marks)
A counterexample for the first triple is x=1, y=0. (3 marks)
A counterexample for the second triple is x=3, y=2, z=1. (3 marks)