UNIVERSITY OF WESTMINSTER

School of Computer Science and Engineering

Module: Reasoning about Programs
Module Code: 6SENG001W, 6SENG003C
Module Leader: Klaus Draeger
Date: 9th January 2019
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.

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

The B-Method's Abstract Machine Notation (AMN) is given in Appendix C.

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

UNIVERSITY OF WESTMINSTER

MARKING SCHEME
DO NOT COPY

School of Computer Science and Engineering

Module:	**Reasoning about Programs**
Module Code:	**6SENG001W, 6SENG003C**
Module Leader:	Klaus Draeger
Date:	9th January 2019
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.

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

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

Section A

Answer ALL questions from this section.
All questions in this section refer to the B Machine in Appendix A.
You may also wish to consult the B-Method notation given in Appendix C.

Question 1

The B machine $TubeSystem$ is given in Appendix A, it models a central region of the London Underground System. It defines several tube lines, some tube stations on those lines and the colour of the lines.

With reference to the $TubeSystem$ B machine evaluate the following expressions:

(a) $BakerlooStations \cap VictoriaStations$ **[1 mark]**

(b) $CentralStations - VictoriaStations$ **[1 mark]**

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

(d) $Baker\_Street \mapsto Victoria \ \in \ onLine$ **[1 mark]**

(e) $\mathrm{ran}(onLine)$ **[1 mark]**

(f) $colour(Central)$ **[1 mark]**

(g) $\bigcup\ \{\ \{\ Bond\_Street,\ Euston\_Square\ \},\ \{\},\ \{\ Warren\_Street\ \}\ \}$ **[2 marks]**

(h) $BakerlooStations \subseteq \mathrm{dom}(onLine)$ **[2 marks]**

(i) $CentralStations \cap \mathrm{dom}(onLine)$ **[2 marks]**

(j) $\mathbb{P}(BakerlooStations)$ **[3 marks]**

[TOTAL 15]

Section A

Answer ALL questions from this section.

Question 1

(a) $BakerlooStations \cap VictoriaStations \ = \ \{Oxford\_Circus\}$
[1 mark]

(b) $CentralStations - VictoriaStations \ = \ \{ \ Bond\_Street, \ Tottenham\_Court\_Road \ \}$
[1 mark]

(c) $\mathrm{card}(\ CircleStations \ ) \ = \ 3$ **[1 mark]**

(d) $Baker\_Street \mapsto Central \ \in \ onLine \ = \ FALSE$ **[1 mark]**

(e) $\mathrm{ran}(onLine) \ = \ TubeLines$ **[1 mark]**

(f) $colour(Central) \ = \ red$ **[1 mark]**

(g) $\bigcup(\ \{ \ \{ \ Bond\_Street, \ Euston\_Square \ \}, \ \{\}, \ \{ \ Warren\_Street \ \} \ \} \ )$
$= \ \{Bond\_Street, \ Euston\_Square, \ Warren\_Street \ \}$ **[2 marks]**

(h) $BakerlooStations \subseteq \mathrm{dom}(onLine) \ = \ TRUE$ **[2 marks]**

(i) $CentralStations \cap \mathrm{dom}(onLine) \ = \ CentralStations$ **[2 marks]**

(j) $\mathbb{P}(BakerlooStations)$

$$
\begin{aligned}
= \ \{ \ & \{\}, \ \{ \ Baker\_Street \ \}, \ \{ \ Regents\_Park \ \}, \ \{ \ Oxford\_Circus \ \}, \\
& \{Baker\_Street, \ Regents\_Park\}, \ \{Baker\_Street, \ Oxford\_Circus\}, \\
& \{Regents\_Park, \ Oxford\_Circus\}, \\
& \{Baker\_Street, \ Regents\_Park, \ Oxford\_Circus\} \ \}
\end{aligned}
$$

[3 marks]

[QUESTION Total 15]

Question 2

With reference to the B machine *TubeSystem* that models the London Underground System, that is given in Appendix A.

(a) From the definition of the relation $onLine$ in terms of the maplets that define it, explain why this could not have been defined as a function. **[2 marks]**

(b) The type of $colour$ is given as a total function: $TubeStation \to Colour$. Given its defined value, can it be given a more specific function type and if so what function type could it be given and why? **[3 marks]**

(c) Evaluate the following expressions:

(i) $onLine[\ \{\ Baker\_Street,\ Oxford\_Circus\ \}\ ]$ **[2 marks]**

(ii) $CircleStations \lhd onLine$ **[2 marks]**

(iii) $onLine\ \rhd\ \{\ Victoria\ \}$ **[2 marks]**

(iv) $onLine\ ⩥\ \{\ Bakerloo,\ Central,\ Victoria\ \}$ **[2 marks]**

(v) $colour\ \Lleftarrow\ \{\ Circle \mapsto red,\ Central \mapsto yellow\ \}$ **[2 marks]**

(vi) $colour^{-1}$ **[2 marks]**

(vii) $(\ CircleStations \lhd onLine\ )\ ;\ colour$ **[3 marks]**

[TOTAL 20]

Question 2

See the *TubeSystem* B machine of the London Underground System, that is given in an Appendix of the Exam paper.

(a) The relation $onLine$ in cannot be defined as a function because it contains maplets that map one element of the domain to more than one element in the range, e.g.

$$= \{ \ \ldots, \ Baker\_Street \mapsto Bakerloo, Baker\_Street \mapsto Circle, \\ Oxford\_Circus \mapsto Bakerloo, Oxford\_Circus \mapsto Central, \\ Oxford\_Circus \mapsto Victoria, \ \ldots \ \}$$

[2 marks]

[PART Total 2]

(b) Yes, the $colour$ function can be defined as a total injective function,

$$colour \in TubeStation \rightarrowtail Colour$$

Because the domain equals the target set (total) & each element in the domain is mapped to a different element in the range, i.e. it is a total injective function. **[3 marks]**

[PART Total 3]

(c) Evaluate the following expressions:

(i) $onLine[\ \{ \ Baker\_Street, \ Oxford\_Circus \ \} \ ]$
$= \{ \ Circle, Central, Bakerloo, Victoria \ \}$ **[2 marks]**

(ii) $CircleStations \lhd onLine$
$$= \{ \ Baker\_Street \mapsto Circle, \ Baker\_Street \mapsto Bakerloo, \\ Great\_Portland\_Street \mapsto Circle, \ Euston\_Square \mapsto Circle \ \}$$

[2 marks]

(iii) $onLine \ \rhd \ \{ \ Victoria \ \}$
$$= \{ \ Oxford\_Circus \mapsto Victoria, \ Warren\_Street \mapsto Victoria \ \}$$

[2 marks]

(iv) $onLine \ \ntriangleright \ \{ \ Bakerloo, \ Central, \ Victoria \ \}$
$$= \{ \ Baker\_Street \mapsto Circle, \ Great\_Portland\_Street \mapsto Circle, \\ Euston\_Square \mapsto Circle \ \}$$ **[2 marks]**

Question 3

(a) What is an abstract *B machine*? You can illustrate your answer by considering the B machines given in the Appendices. **[5 marks]**

(b) Explain the purpose of the following B machine *clauses*:

- `VARIABLES`
- `INVARIANT`
- `INITIALISATION`

You can illustrate your answer by considering the B machines given in the Appendices. **[5 marks]**

(c) With reference to the B machine *TubeSystem* that models the London Underground System, that is given in Appendix A.

You are required to add the notion of a tube passenger's *location*, in terms of the current tube station and tube line, to the *TubeSystem* B machine. Illustrate how this can be achieved using the three *clauses* from part **(b)**. **[5 marks]**

[TOTAL 15]

(v) $colour \Lleftarrow \{\ Circle \mapsto red,\ Central \mapsto yellow\ \}$
$= \{\ Bakerloo \mapsto brown, Circle \mapsto red,$
$Central \mapsto yellow, Victoria \mapsto lightblue\ \}$ **[2 marks]**

(vi) $colour^{-1}$
$= \{\ brown \mapsto Bakerloo,\ yellow \mapsto Circle,$
$red \mapsto Central,\ lightblue \mapsto Victoria\ \}$ **[2 marks]**

(vii) $(\ CircleStations \lhd onLine\ )\ ;\ colour$
$= \{\ Baker\_Street \mapsto brown),\ Baker\_Street \mapsto yellow,$
$Great\_Portland\_Street \mapsto yellow,\ Euston\_Square \mapsto yellow\ \}$

[3 marks]

[QUESTION Total 20]

Question 3

(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. **[1 mark]**

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 5]

(b) Explain the purpose of the following B Machine *clauses*:

- `VARIABLES` – declare state variable identifiers. **[1 mark]**
- `INVARIANT` – define the *state invariant* for the system, including the types of the variables & any additional constraints on them. **[2 marks]**
- `INITIALISATION` – initialise all the state variable with values that *satisfy the state invariant*. **[2 marks]**

[PART Total 5]

Question 3

(a) What is an abstract *B machine*? You can illustrate your answer by considering the B machines given in the Appendices. **[5 marks]**

(b) Explain the purpose of the following B machine *clauses*:

- `VARIABLES`
- `INVARIANT`
- `INITIALISATION`

You can illustrate your answer by considering the B machines given in the Appendices. **[5 marks]**

(c) With reference to the B machine *TubeSystem* that models the London Underground System, that is given in Appendix A.

You are required to add the notion of a tube passenger's *location*, in terms of the current tube station and tube line, to the *TubeSystem* B machine. Illustrate how this can be achieved using the three *clauses* from part **(b)**. **[5 marks]**

[TOTAL 15]

(c) To add a tube passenger's current *location*, (station and line), to the *TubeSystem* B machine, need to add the following clauses:

`VARIABLES`
$station, line$
`INVARIANT`
$station \in Stations \ \wedge \ line \in TubeLine \wedge$
$station \mapsto line \ \in \ onLine$
`INITIALISATION`
$station := Oxford\_Circus \ \ ||$
$line := Victoria$

[5 marks]

Any pair of station & line can be used to initialise the two variables, as long as they satisfy the invariant, i.e. are *consistent* with $onLine$.

[PART Total 5]

[QUESTION Total 15]

Section B

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

Question 4

Write a B machine that specifies a *luggage rack*, that is, a rack that holds a number of luggage items, e.g. cases, bags, etc.
The luggage items are added and removed from the rack in a *"last-in-first-out"* order, i.e. the first item of luggage added is the last to be removed, and the last item added is the first item to be removed.
The *luggage rack* can hold a maximum number of items of luggage.

Your `LuggageRack` B machine should include the following:

(a) Sets, constants, variables and the state invariant that is required. **[9 marks]**

(b) The following luggage rack 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)** `AddLuggage` – adds an item of luggage onto the rack; unless it is full. **[6 marks]**
- **(ii)** `RemoveLuggage` – removes an item of luggage from the rack, and returns it; unless it is empty. If it is empty then an error value should be returned. **[7 marks]**
- **(iii)** `AnyLuggageLeft` – returns *Yes* if the rack is not empty; otherwise returns *No*. **[3 marks]**

[TOTAL 25]

Section B

Answer TWO questions from this section.

Question 4

The `LuggageRack` B machine; basically its a **stack**. So would expect something similar to the following machine, but a student's version is unlikely to include all the details included in this *solution*, as long as its a stack & has the main parts will not penalise for minor errors, e.g. syntax errors, etc.

(a)
```
MACHINE LuggageRack

  SETS
    LUGGAGE  = { case1, case2, case3, case4, case5,
                 bag1,  bag2,  bag3,  bag4,  bag5,
                 null_bag } ;

    ANSWER  = { Yes, No } ;

    MESSAGE = { Luggage_Added,   ERROR_No_Space_Left,
                Luggage_Removed, ERROR_No_Luggage_Left }

  CONSTANTS
    MaxItemsOfLuggage, No_Luggage, EMPTY_LuggageRack

  PROPERTIES
    MaxItemsOfLuggage : NAT1         &  MaxItemsOfLuggage = 5   &
    No_Luggage        : LUGGAGE      &  No_Luggage = null_bag   &
    EMPTY_LuggageRack : seq(LUGGAGE) &  EMPTY_LuggageRack = []

  VARIABLES
    luggageRack

  INVARIANT
    luggageRack : seq( LUGGAGE ) &
    size( luggageRack ) <= MaxItemsOfLuggage

  INITIALISATION
    luggageRack := EMPTY_LuggageRack
```


Section B

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

Question 4

Write a B machine that specifies a *luggage rack*, that is, a rack that holds a number of luggage items, e.g. cases, bags, etc.
The luggage items are added and removed from the rack in a *"last-in-first-out"* order, i.e. the first item of luggage added is the last to be removed, and the last item added is the first item to be removed.
The *luggage rack* can hold a maximum number of items of luggage.

Your `LuggageRack` B machine should include the following:

(a) Sets, constants, variables and the state invariant that is required. **[9 marks]**

(b) The following luggage rack 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)** `AddLuggage` – adds an item of luggage onto the rack; unless it is full. **[6 marks]**
- **(ii)** `RemoveLuggage` – removes an item of luggage from the rack, and returns it; unless it is empty. If it is empty then an error value should be returned. **[7 marks]**
- **(iii)** `AnyLuggageLeft` – returns *Yes* if the rack is not empty; otherwise returns *No*. **[3 marks]**

[TOTAL 25]

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 luggage rack (stack) is the front of the sequence, but okay if the end. Might use strings for reporting rather than MESSAGE.

```
OPERATIONS

  report <-- AddLuggage( luggage ) =
      PRE
          report : MESSAGE & luggage : LUGGAGE
      THEN
          IF ( size(luggageRack) < MaxItemsOfLuggage )
          THEN
              luggageRack := luggage -> luggageRack ||
              report      := Luggage_Added
          ELSE
              report := ERROR_No_Space_Left
          END
      END ;
```

[PART Total 6]

(ii)

```
  report, topcase <-- RemoveLuggage =
      PRE
          report : MESSAGE & topcase : LUGGAGE
      THEN
          IF ( luggageRack /= EMPTY_LuggageRack )
          THEN
              luggageRack := tail(luggageRack) ||
              report      := Luggage_Removed   ||
              topcase     := first(luggageRack)
          ELSE
              report  := ERROR_No_Luggage_Left ||
              topcase := No_Luggage
          END
      END ;
```

[PART Total 7]

Question 5

Appendix B contains the `BirthdayBook` B machine.

The *Birthday Book* system is used to record people's birthdays. It does this by recording a person's *name* and *birthday*.
The system includes the following operations:

- `AddBirthday` – a person's birthday is added to the book.
- `DeleteBirthday` – a person's birthday is removed from the book.
- `FindBirthday` – find a person's birthday.
- `Reminder` – reports who has a birthday on a specific date.
- `NumKnownBirthdays` – reports number of birthdays recorded.

With reference to the `BirthdayBook` B machine (see Appendix B) answer the following questions.

(a) What is the B type of `DATE`? How would today's date be represented? **[2 marks]**

(b) What is the B type of `known`? Give an example of one of its possible values. **[1 mark]**

(c) The state variable `birthday` is defined as a *partial function* (line 25) as follows:

```
25            birthday : NAME +-> DATE
```

Discuss why you think this type of mapping was used, rather than a relation or any of the other types of functions? **[6 marks]**

(d) Given that `birthday` is a *partial function*, the three additional constraints placed on it are:

```
26            card( birthday ) <= maximum  &
27            known = dom( birthday )      &
28            NonDate /: ran( birthday )
```

Explain in **plain English** what each of these constraints mean. **[3 marks]**

[Continued Overleaf]

(iii)
```
answer <-- AnyLuggageLeft =
    PRE
        answer : ANSWER
    THEN
        IF   ( luggageRack /= EMPTY_LuggageRack )
        THEN
             answer := Yes
        ELSE
             answer := No
        END
    END
END  /* LuggageRack */
```
[PART Total 3]

[PART Total 16]

[QUESTION Total 25]

Question 5

See Exam paper Appendix for the `BirthdayBook` B machine.

(a) The B type of `DATE` is an element of the Cartesian product of days & months, i.e. an ordered pair (or maplet). **[1 mark]**

Today's date is represented: $9 \mapsto Jan$ (exam date). **[1 mark]**

[PART Total 2]

(b) The B type of `known` is a set of names. Example value is any subset of `NAME`, e.g. `known = { Jim, Sue, Mon, Zoe }`. **[1 mark]**

[PART Total 1]

(c) `birthday` is a *partial function* because the birthday book will not always hold everyone's birthday & everyone has just one birthday. **[1 mark]**

It is not:

- a relation because no one has more than one birthday, (excluding the Queen). **[1 mark]**
- a total function since the birthday book does not always hold everyone's birthday. **[1 mark]**

(e) Explain in **plain English only** the meaning of the *preconditions* for the following operations:

- **(i)** `AddBirthday` **[4 marks]**
- **(ii)** `Reminder` **[2 marks]**
- **(iii)** `NumKnownBirthdays` **[2 marks]**

(f) If the *pre-condition* of the `AddBirthday` operation is true, how does it update the state? **[2 marks]**

(g) If the `Reminder` operation is executed in the ProB tool with the parameter `10 |-> Jul`, i.e. "`Reminder( 10 |-> Jul )`", and the value of the output variables are as follows:

```
report = Birthdays_On_Date
cards  = { Jim, Sue }
```

Give values for the two state variable `known` and `birthday` that would be consistent with this. **[3 marks]**

[TOTAL 25]

- an injective function since several people can have the same birthday. **[1 mark]**
- a surjective function since not every possible date will be recorded as someone's birthday. **[1 mark]**
- a bijections, since its neither an injective, surjective or total function. **[1 mark]**

[PART Total 6]

(d) `birthday`'s constraints:

- `card( birthday ) <= maximum`
  the birthday book has a maximum limit on how many birthdays it can record. **[1 mark]**
- `known = dom( birthday )`
  the known people are those that have a birthday recorded. **[1 mark]**
- `NonDate /: ran( birthday )`
  no one can have a birthday recorded as being on the 31st February. **[1 mark]**

[PART Total 3]

(e) *Pre-conditions* for the following operations:

(i) `AddBirthday`
The name must be a proper name. **[1 mark]** The date must be valid date & it cannot be 31st February. **[2 marks]** There must be room in the birthday book to record at least one more birthday. **[1 mark]**
[SUBPART Total 4]

(ii) `Reminder`
The date must be valid date & it cannot be 31st February. **[1 mark]** Only a collection (set) of real names can be output. **[1 mark]**
[SUBPART Total 2]

(iii) `NumKnownBirthdays`
Since there is no explicit pre-condition for this operation, it is implicitly just "TRUE". **[2 marks]**
[PART Total 2]

(e) Explain in **plain English only** the meaning of the *preconditions* for the following operations:

(i) `AddBirthday` **[4 marks]**

(ii) `Reminder` **[2 marks]**

(iii) `NumKnownBirthdays` **[2 marks]**

(f) If the *pre-condition* of the `AddBirthday` operation is true, how does it update the state? **[2 marks]**

(g) If the `Reminder` operation is executed in the ProB tool with the parameter `10 |-> Jul`, i.e. "`Reminder( 10 |-> Jul )`", and the value of the output variables are as follows:

```
report = Birthdays_On_Date
cards  = { Jim, Sue }
```

Give values for the two state variable `known` and `birthday` that would be consistent with this. **[3 marks]**

[TOTAL 25]

[PART Total 8]

(f) `AddBirthday` updates the state as follows:

- Provided the person's `name` is not already in the birthday book it updates the state by adding the `name` to the known people & adds the `name` and `date` pair to the birthday book. **[1 mark]**
- If the person's `name` is already in the birthday book it does not add them again, i.e. it does not update the state. **[1 mark]**

[PART Total 2]

(g) The state to produce the ProB output: `report = Birthdays_On_Date & cards = { Jim, Sue }` for "`Reminder( 10 |-> Jul )`" is any pair of `known` & `birthday` that satisfy:

$$\{\ Jim,\ Sue\ \} \ \subseteq\ known$$
$$\{\ Jim \mapsto (10 \mapsto Jul),\ Sue \mapsto (10 \mapsto Jul)\ \}\ =\ birthday \rhd \{\ 10 \mapsto Jul\ \}$$

[3 marks]

[PART Total 3]

[QUESTION Total 25]

Question 6

Marking Scheme for Hoare Logic & Program Verification.

(a) The Hoare triple

$$[x = 0]\ y := z\ [z = x + y]$$

means that executing the instruction $y := z$ (i.e. assigning the value of z to y), starting from a state in which x is 0, leads to a state in which z equals $x + y$. **[2 marks]**

[SUBPART Total 2]

(b) **(i)** $[x < y]\ y := 0\ [x < 0]$ is invalid. **[1 mark]** Counterexample: Starting in a state with $x = 1, y = 2$ leads to a state with $x = 1, y = 0$. **[1 mark]**

[SUBPART Total 2]

Question 6

(a) Explain in your own words the meaning of the Hoare triple

$$[x = 0]\ y := z\ [z = x + y]$$

[2 marks]

(b) Which of the following Hoare triples are valid? Give a counterexample for each invalid triple.

- **(i)** $[x < y]\ y := 0\ [x < 0]$ **[2 marks]**
- **(ii)** $[x < y]\ y := y + 1\ [x < y + 1]$ **[2 marks]**
- **(iii)** $[x < y]\ y := y - 1\ [x < y - 1]$ **[2 marks]**
- **(iv)** $[true]\ x := 0\ [true]$ **[2 marks]**

(c) Find the missing assertions using pre-condition propagation.

```
[assertion 1]
  y:=y-z;
[assertion 2]
  z:=x+z;
[assertion 3]
  z:=y+z
[x<z]
```

[6 marks]

(d) Find suitable intermediate assertions for the following Hoare triple; this involves finding an invariant for the loop.

```
[y=10]
x:=0;
[invariant]
WHILE y>0 DO
[assertion 1]
  x:=x+1;
[assertion 2]
  y:=y-1
[assertion 3]
END
[x=10]
```

[9 marks]
[TOTAL 25]

(ii) $[x < y]\ y := y{+}1\ [x < y{+}1]$ is valid: the pre-condition is $x < y{+}2$, which follows from $x < y$. **[2 marks]**
[SUBPART Total 2]

(iii) $[x < y]\ y := y - 1\ [x < y - 1]$ is invalid. **[1 mark]** Counterexample: Starting in a state with $x = 1, y = 2$ leads to a state with $x = 1, y = 0$. **[1 mark]**
[SUBPART Total 2]

(iv) $[true]\ x := 0\ [true]$ is valid, since any post-state satisfies $true$. **[2 marks]**
[SUBPART Total 2]

[PART Total 10]

(c) The intermediate assertions are

1. $0 < z$ **[2 marks]**
2. $0 < y + z$ **[2 marks]**
3. $x < y + z$ **[2 marks]**

[PART Total 6]

(d) $[y = 10]$
$x := 0;$
$[x + y = 10\ \&\ y \geq 0]$ **[3 marks]**
$WHILE\ y > 0\ DO$
$[x + y = 10\ \&\ y > 0]$ **[2 marks]**
$x := x + 1;$
$[x + y = 11\ \&\ y > 0]$ **[2 marks]**
$y := y - 1$
$[x + y = 10\ \&\ y \geq 0]$ **[2 marks]**
END
$[x = 10]$

[PART Total 9]

[QUESTION Total 25]

Appendix A. London Tube System B Machine

The B machine *TubeSystem* models a central region of the London Underground System.

MACHINE $TubeSystem$
SETS
$TubeLine \ = \ \{ \ Bakerloo, \ Circle, \ Central, \ Victoria \ \}$;
$Colour \ = \ \{ \ black, brown, darkblue, green, lightblue, orange, purple, red, silver, yellow \ \}$;
$Station \ = \ \{Baker\_Street, \ Regents\_Park, \ Oxford\_Circus, \ Bond\_Street, Great\_Portland\_Street, \ Euston\_Square, \ Warren\_Street, Tottenham\_Court\_Road \ \}$

CONSTANTS
$BakerlooStations, CircleStations, CentralStations, VictoriaStations, onLine, colour$

PROPERTIES
$BakerlooStations \in \mathbb{P}(Station) \ \wedge$
$BakerlooStations = \{Baker\_Street, Regents\_Park, Oxford\_Circus\} \ \wedge$

$CircleStations \in \mathbb{P}(Station) \ \wedge$
$CircleStations = \{Baker\_Street, Great\_Portland\_Street, Euston\_Square\} \ \wedge$

$CentralStations \in \mathbb{P}(Station) \ \wedge$
$CentralStations = \{Bond\_Street, Oxford\_Circus, Tottenham\_Court\_Road\} \ \wedge$

$VictoriaStations \in \mathbb{P}(Station) \ \wedge$
$VictoriaStations = \{Warren\_Street, Oxford\_Circus\} \ \wedge$

$onLine \in Station \leftrightarrow TubeLine \ \wedge$
$onLine = \{Baker\_Street \mapsto Bakerloo, Regents\_Park \mapsto Bakerloo, Oxford\_Circus \mapsto Bakerloo, Baker\_Street \mapsto Circle, Great\_Portland\_Street \mapsto Circle, Euston\_Square \mapsto Circle, Bond\_Street \mapsto Central, Oxford\_Circus \mapsto Central, Tottenham\_Court\_Road \mapsto Central, Warren\_Street \mapsto Victoria, Oxford\_Circus \mapsto Victoria\} \ \wedge$

$colour \in TubeLine \rightarrow Colour \ \wedge$
$colour = \{Bakerloo \mapsto brown, Central \mapsto red, Circle \mapsto yellow, Victoria \mapsto lightblue\}$
END /* $TubeSystem$ */

Appendix B. Birthday Book B Machine

The following is a B Machine – `BirthdayBook` that specifies a simple system of recording people's birthdays.

```
MACHINE BirthdayBook( maximum )

CONSTRAINTS
  maximum : NAT1

SETS
  NAME   = { Tim, Tom, Ian, Jim, Sue, Liz, Mon, Zoe } ;
  MONTH  = { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec } ;
  REPORT = { Success, Already_Known, Unknown,
             Birthdays_On_Date, No_Birthdays_On_Date }

CONSTANTS
  DATE, DAY, NonDate

PROPERTIES
  DAY  = 1..31        &
  DATE = DAY * MONTH  &
  NonDate : DATE      &  NonDate = 31 |-> Feb

VARIABLES
  known, birthday

INVARIANT
  known : POW( NAME )          &
  birthday : NAME +-> DATE     &
  card( birthday ) <= maximum  &
  known = dom( birthday )      &
  NonDate /: ran( birthday )

INITIALISATION
  known    := {}   ||
  birthday := {}

```

[Continued on next page.]

```

OPERATIONS

  report <-- AddBirthday( name, date ) =
      PRE
          name : NAME  &  date : DATE  &  date /= NonDate  &
          card(birthday) < maximum
      THEN
          IF   ( name /: known )
          THEN
               known    := known \/ { name }              ||
               birthday := birthday \/ { name |-> date }  ||
               report   := Success
          ELSE
               report   := Already_Known
          END
      END
           ;

  report <-- DeleteBirthday( name ) =
      PRE
          name : NAME
      THEN
           IF   ( name : known )
           THEN
                known    := known - { name }        ||
                birthday :=  { name } <<| birthday  ||
                report   := Success
           ELSE
                report := Unknown
           END
      END
      ;

```

[Continued on next page.]


```

    report, date <-- FindBirthday( name ) =
        PRE
            name : NAME
        THEN
            IF   ( name : known )
            THEN
                 date   := birthday( name )  ||
                 report := Success
            ELSE
                 date   := NonDate     ||
                 report := Unknown
            END
        END
        ;

    report, cards <-- Reminder( date ) =
        PRE
             date : DATE  &  date /= NonDate  &  cards <: NAME
        THEN
             IF   ( date : ran( birthday ) )
             THEN
                  cards  := { name | name : known & birthday(name) = date } ||
                  report := Birthdays_On_Date
             ELSE
                  cards  := {}  ||
                  report := No_Birthdays_On_Date
             END
        END
        ;

    numBDs <-- NumKnownBirthdays =
        BEGIN
              numBDs := card( birthday )
        END

  END /* BirthdayBook */
```

Appendix C. 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*.

C.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
$x \,..\, y$	`x .. y`	range of numbers from x to y inclusive

C.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

C.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
$x \mathinner{..} y$	`x .. y`	Range of integers from x to y inclusive
$\mathbb{P}(A)$	`POW(A)`	Power set of A
$\mathrm{card}(A)$	`card(A)`	Cardinality, number of elements in set A

C.4 AMN: Set Operators & Relations

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

C.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

C.6 AMN: Ordered Pairs & Relations

B Symbol	ASCII	Description
$X \times Y$	`X * Y`	Cartesian product of X and Y
$x \mapsto y$	`x |-> y`	Ordered pair, maplet
$\mathrm{prj}_1(S,T)(x \mapsto y)$	`prj1(S,T)(x |-> y)`	Ordered pair projection function
$\mathrm{prj}_2(S,T)(x \mapsto y)$	`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

C.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

C.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
$X \mathrel{\rightarrowtail\mkern-18mu+} Y$	`X >+> Y`	Partial injection from X to Y
$X \rightarrowtail Y$	`X >-> Y`	Total injection from X to Y
$X \mathrel{\twoheadrightarrow\mkern-18mu+} Y$	`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

C.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
$\mathrm{iseq}(X)$	`iseq( X )`	Set of injective Sequences over set X
$\mathrm{size}(s)$	`size( s )`	Size (length) of Sequence s

C.10 AMN: Sequences Operators

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

C.11 AMN: Miscellaneous Symbols & Operators

B Symbol	ASCII	Description
$var := E$	`var := E`	Assignment
$S1 \mid\mid S2$	`S1 || S2`	Parallel execution of $S1$ and $S2$

C.12 AMN: Operation Statements

C.12.1 Assignment Statements

```
xx := xxval

xx, yy, zz := xxval, yyval, zzval

xx := xxval  ||  yy := yyval
```

C.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
```

C.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
```