

CSE4213 Examination
June 2007
Instructions to Candidates

1. Examination time 2 hours
2. Answer all questions
3. There are 5 questions
4. Each question is worth 12 marks
5. Total marks 60
6. Calculators are not allowed
7. Use left hand page for rough working; this page will NOT be marked unless explicitly requested.
8. The *Concise Summary of the B mathematical toolkit* is supplied.

1. Give 3 attributes concerning predicates and their role in the B Method. For each attribute, give a corresponding example that demonstrates the attribute.

(a)

Answer: Predicates are statements which are either true or false

Example: $x > 5, \forall z.(z \in HOLDEN) \Rightarrow (z \in CARS)$

(2 marks)

(b)

Answer: Predicates can be used to make statements about sets (the basis for a B specification)

Example: **ANY** xx **WHERE** $xx \in HOLDENS$ defines which xx are to be used

(2 marks)

(c)

Answer: Predicates define the assumptions under which an operation is valid

Example: **PRE** $xx \in BOOKS$ **THEN** ...

(2 marks)

Answer: Predicates define the **INVARIANT** of a machine (the state and its basic properties). Example: **INVARIANT** $xx \in \mathbb{N} \wedge xx \leq 5 \dots$

Answer: Only 3 statements required, others are possible

One mark for each correct attribute, one mark for each correct example. The example must demonstrate the attribute.

Comment:

- (d) The B Toolkit uses abstract machines in specifying software behaviour. Explain the role of *state* in the use of abstract machines.

Answer: Example: An abstract machine encapsulates a *state*, defined by a set of *variables* with *values* that represent the state value of the machine. These values may have a range of *types*, defined by typed sets.

The abstract machine also has a range of *operations* which may either interrogate or modify the state variables. An essential aspect of any operation is that it must maintain the *invariant* defined by the machine. The *initialisation* of the state variables must also establish the invariant.

Alteration of the state variables takes place through a set of *substitutions*, defined through a *Generalised Substitution Language*, and performed *atomically*. Each substitution is specified through an analogous statement in the *Abstract Machine Notation*.

Alternatively: Any answer which mentions the following, each with some context:

variables
values
type
invariant
operations
substitutions
AMN/GSL
atomic

1 marks each, up to a maximum of 6

Comment: Basically a rephrasing of question 2 from 2006. It is not enough to mention the above attributes (only 1 mark per attribute), but the context must also be briefly explained (1 mark), as per the story above.

(6 marks)

2. (a) Explain the role of the *invariant* in B specifications.

Answer: The invariant defines the nature of the variables in the machine, which in turn describe the *state*. The invariant must be established by the initialisation, and re-established by every operation.

Comment: Bookwork

(3 marks)

- (b) Explain how proof obligations for operations in a B specification are derived.

Answer: Since every operation must re-establish the invariant, if the state substitution performed by the operation is G , then

$$Cst \wedge Ctx \wedge I \wedge P \Rightarrow [G]I$$

where Cst are the constraints, Ctx are the properties, I is the Invariant, and P is the Precondition. The substitution $[G]I$ must be established (proved).

Comment: accept just $I \wedge P \Rightarrow [G]I$

(3 marks)

- (c) Explain why global sets in a B development (involving more than one machine) should not be passed as parameters.

Answer: There is no guarantee that the same instantiation will be used. Such sets should be defined in so-called **Type** machines.

Comment: *instantiation* is the key word.

(3 marks)

- (d) Explain why guarded substitutions must be used with care.

Answer: The simple guarded substitution $P \Rightarrow G$ applied to some predicate R can be written

$$[P \Rightarrow G]R \equiv P \Rightarrow [G]R$$

When P is false, the substitution is true, regardless of G or R . This implies that any state is achievable! (The so-called miracle program.)

Comment: Not well done. The key point is that the transformed predicate is always true when P is false, thus satisfying **any** desired state.

(3 marks)

3. Define a machine called *ArrayType* to represent an array. The machine should have two parameters, a natural number to fix the upper limit of the indexes (the lower limit is 1) and a set for the type of values to be stored in the array. There should be operations to
 - read the value at a given index;
 - store a value at a given index;
 - exchange the values at two indexes.

The initialization and the operations to read values from the array and to exchange values in the array should warn the user of the array not to look at values at an index that has not previously had a value stored.

Answer:

It is assumed that the external set *SEES* contains appropriate result indicators.

MACHINE *Array* (*maxindex* , *TYPE*)

SEES

ERRORS

VARIABLES

arr

INVARIANT

$arr \in 1 .. maxindex \mapsto TYPE$

INITIALISATION

$arr := \{\}$

OPERATIONS

$res, tt \leftarrow \mathbf{read} (ix) \hat{=}$

IF $ix \in \mathbf{dom} (arr)$

THEN

$tt := arr (ix) \parallel res := OK$

ELSE

$res := ElementNotInitialized$

END ;

$\mathbf{set} (ix , tt) \hat{=}$

PRE

$ix \in 1 .. maxindex \wedge tt \in TYPE$

THEN

$arr := arr \triangleleft \{ ix \mapsto tt \}$

END ;

$res \leftarrow \mathbf{swap} (ix , jx) \hat{=}$

IF $ix \in \mathbf{dom} (arr) \wedge jx \in \mathbf{dom} (arr)$

THEN

$arr := arr \triangleleft \{ ix \mapsto arr (jx) , jx \mapsto arr (ix) \} \parallel res := OK$

ELSE

$res := ElementNotInitialized$

END

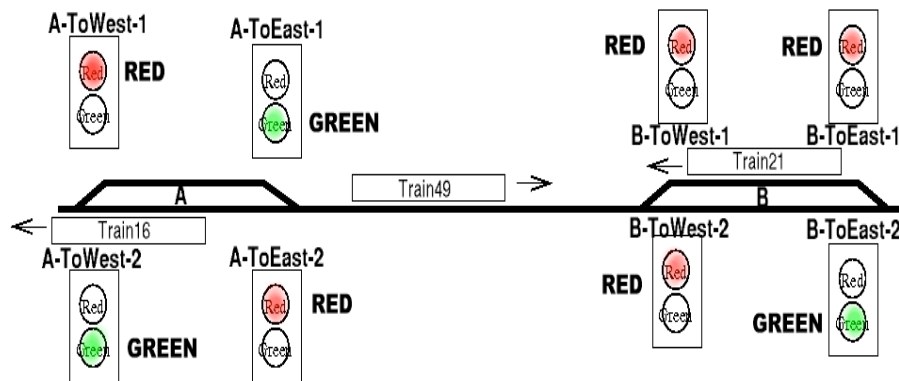
END

(12 marks)

4. The *TrafficLights* problem discussed in lectures showed how both *safety* and *sequencing* were enforced by the appropriate use of invariant and precondition predicates.

In the following railway scenario, you are to develop similar safety and sequencing rules for the operation of trains on a single piece of track. The constraints are that only one train can be travelling on the section of track at a time, and other trains are not permitted to enter at either end of the track until the occupying train has safely cleared the section. At each end of the section is a passing loop, which allows 2 trains to pass, either in the same direction (overtaking), or the opposite direction.

The following diagram gives an example:



The exit from each track in the loops are controlled by a two-aspect light, red (upper) and green (lower) only, facing towards the center of the loop. The points at each end are assumed to be automatically controlled. The current aspect of each signal is shown alongside the signal.

- Train16 is departing loop A to the west on track 2 (the lower in the diagram), and signal A-ToWest-2 is green.
- Train49 has just crossed with Train16 at Loop A, and has departed from track 1. Signal A-ToEast-1 is still showing green, but is about to change to red. In anticipation of Train49 arriving at Loop B, the signal B-ToEast-2 has changed to green, indicating a clear run through the loop.
- Train21 has been held at Loop B, awaiting the arrival of Train49. Signal B-ToWest-1 is therefore showing red.

Assume the following sets: $ASPECT = \{Red, Green\}$, $DIR = \{East, West\}$, $TRACK = \{loopAtrack1, loopAtrack2, sectionAB, loopBtrack1, loopBtrack2\}$.

The machine is to have the following operations: $ChangeToGreen(signal)$, $ChangeToRed(signal)$, $MoveTrain(trackX, trackY)$, where the last operation marks a train as moving from $trackX$ to $trackY$.

Answer the following questions on the next page.

Answer: Oh dear. I should have given a hint on how to approach this problem.

The task is **dramatically** simplified if you recognize the symmetry of the problem, and that you only have to deal with the *sectionAB*, and the entry to that section from the loops on either side. That is, the A-To-East and B-To-West signals are all that is relevant to *sectionAB*. and that the other signals relate to the (unspecified) sections on the far left and far right of the diagram.

I should also have spelt out the sets

$SIGNALAB = \{A-To-East-1, A-To-East-2, B-To-West-1, B-To-West-2\}$

(using the symmetry model), and

$TRAIN = \{\dots, Train16, \dots, Train21, \dots, Train49, \dots\}$

which simplify the answer.

(a) Define a suitable state for the machine

Answer:

$signals \in SIGNALAB \rightarrow ASPECT$

$occupied \in TRACK \leftrightarrow TRAIN$

Each signal must show an aspect (total); tracks may or may not be (partial) occupied by a particular train.

(4 marks)

(b) Define a suitable invariant for the machine

Answer:

$occupied (sectionAB) \Rightarrow$

$signal (B-To-West-1) = Red \wedge$

$signal (B-To-West-2) = Red \wedge$

$signal (A-To-East-1) = Red \wedge$

$signal (A-To-East-2) = Red$

If the section is occupied, then every signal giving access to that section must be *Red*.

(4 marks)

(c) Define suitable preconditions for the operations.

Answer:

Change_To_Green (*signal*) $\hat{=}$

PRE $signal \in SIGNALAB \Rightarrow sectionAB \notin dom (occupied)$

THEN ... **END** ;

Change_To_Red (*signal*) $\hat{=}$

PRE *TRUE*

THEN ... **END** ;

MoveTrain (*trackX* , *trackY*) $\hat{=}$

PRE $trackX \in dom(occupied) \wedge trackY \notin dom(occupied)$

THEN ... **END** ;

You can only change to green if the controlled section is unoccupied; you can always change a signal to red; and to move a train, there must be a train in the starting track, and no train in the destination track.

(4 marks)

5. The following code is taken from one of the B demonstration packages for computing the factorial of a number. It is a robust implementation of a specification that computes the factorial if it is representable within the integer range of the computer system, and 0 otherwise.

Explain the key features of the implementation, paying particular attention to the variables *done*, *count*, and *in_range*. You should make reference to the 5 rules for establishing loop correctness, namely the **Initialisation**, **Final**, **T1 variant natural**, **T2 variant decreasing** and **Preserve invariant** rules.

Useful hints:

STO_NVAR Store a natural number

SCL Make argument into a scalar

EQL args are equal, TRUE or FALSE

_PRE_MUL_NVAR Precondition that result of multiplication is still in range of scalars

_MUL_NVAR Multiply scalar by the argument

DEC decrement a scalar

math_fac the mathematical definition of factorial (unbounded)

IMPLEMENTATION

This implementation is a robust implementation of factorial

factorial_1

REFINES

factorial

SEES

Bool_TYPE , *Scalar_TYPE* , *Scalar_TYPE_Ops*

IMPORTS

An integer machine is used

factorial_Nvar (*my_maxint*)

PROPERTIES

my_maxint = *MaxScalar*

OPERATIONS

This implementation returns factorial (if possible).

```
ret ← comp_fac ( inp ) ≐
BEGIN
  VAR in_range , done , count IN
    factorial_STO_NVAR ( 1 ) ;
    count ← SCL ( inp ) ;
    done ← EQL ( count , 0 ) ;
    IF done = FALSE THEN
      in_range ← factorial_PRE_MUL_NVAR ( count ) ;
      WHILE in_range = TRUE ∧ done = FALSE DO
        factorial_MUL_NVAR ( count ) ;
        count ← DEC ( count ) ;
        done ← EQL ( count , 0 ) ;
        in_range ← factorial_PRE_MUL_NVAR ( count )
      INVARIANT
        count ∈ ℕ ∧
        count ∈ 0 .. my_maxint ∧
        factorial_Nvar ∈ 0 .. my_maxint ∧
        ( done = TRUE ⇒ count = 0 ) ∧
        ( done = TRUE ⇒ in_range = TRUE ) ∧
        ( done = FALSE ⇒ count ≠ 0 ) ∧
        ( in_range = TRUE ⇒ math_fac ( inp ) = math_fac ( count ) × factorial_Nvar ) ∧
        ( in_range = TRUE ⇒ factorial_Nvar × count ≤ my_maxint )
      VARIANT
        count
      END
    END ;
    IF done = TRUE THEN
      ret ← factorial_VAL_NVAR
    ELSE
      ret := 0
    END
  END
END
END
END
```

Write your answers on the next page

Answer:

- (a) It is a **refinement**. This means that there is another machine *factorial* that defines the basic specification, and this refinement is concerned with completing the data refinement (finite variable values) and algorithm refinement (iteration, not recursion). 1 mark for the key word **refinement**.
- (b) 1 mark for identifying the finite nature of the data
- (c) 1 mark for identifying the iteration refinement
- (d) 1 mark for identifying that *in_range* flags that the computation is still within the finite range of scalars
- (e) 1 mark for identifying that *count* is the loop counter, approaching 0
- (f) 1 mark for identifying that *done* is TRUE if the loop completes with a correct computation
- (g) 6 marks for explaining each of the loop correctness requirements and how they are met.

(12 marks)