

CSE4213 Examination
June 2006
Instructions to Candidates

1. Examination time 2 hours
2. Answer all questions
3. There are 10 questions
4. Each question is worth 12 marks
5. Total marks 120
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 non-determinism and its role in the B Method. For each attribute, give a corresponding example that demonstrates the attribute.

(a)

Answer: non-computational: the emphasis is upon the abstract properties of the solution, not how it is computed; OR Consistent with the Principle of Semantic Emphasis: The *what* and not the *how*.

Example: **ANY** xx **WHERE** $xx \times xx = \textit{square}$

(4 marks)

(b)

Answer: Allows implementor freedom of choice.

Example: **ANY** xx **WHERE** $xx \in \mathbb{N}$, implementer can choose e.g. $xx:=0$

(4 marks)

(c)

Answer: Avoids over-specification; OR Gives weaker specification and/or pre-conditions.

Example: $xx : \in \textit{SET}$, no need to specify which element

(4 marks)

- (d) If your name is Julian, give a complete B implementation of Gregorian Simple Dates. (For Anonymous Feedback readers only.)

(0 marks)

Answer: Two marks for each correct attribute, two marks for each correct example.

The example must demonstrate the attribute.

Comment: (2004) Generally OK, although some thought that the question asked for examples of non-determinism. Nearly everyone gave the Principle of Semantic Emphasis, obviously pandering to the predilections of the examiner. A common answer (which also scored no marks) was that non-determinism had to be removed before implementation, which although true, was not the point of the question. Another common answer (which scored half marks) was to couple non-determinism with proof obligations, either in the context of making them easier (dubious), or of extending the breadth and robustness of specifications (more valid).

2. The B Toolkit uses abstract machines in specifying software behaviour. Explain how, paying particular attention to the notion of *state* and *operations* over the state.

Answer: 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.(4 marks)

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.(4 marks)

Alteration of the state variables takes place through a set of *substitutions*, defined through a *Generalised Substitution Language*. Each substitution is specified through an analogous statement in the *Abstract Machine Notation*.(4 marks)

Comment: 2004 The **how** and the **what** were key words in many students answers, and were rewarded as were the key words **finite/unbounded**, **abstract/concrete**, and the like. Few problems in general.

(12 marks)

3. (a) Explain the role of preconditions in B specifications.

Answer: Preconditions are *assumptions* about the state of the system. They are *not* tested, and if they are not satisfied (true), then the specification is not valid (behaviour is unconstrained).

Comment: Bookwork

(3 marks)

- (b) Explain the role of proof obligations in a B specification.

Answer: Proof obligations are formal requirements about how the system can change state. Discharging the proof obligations (proving them) is equivalent to verifying that all operations wherein the preconditions are satisfied will not violate the system invariant (that is, the system behaviour is consistent with the specification).

Comment: also bookwork

(3 marks)

- (c) Abstract machine specifications in B do not allow the use of the sequential substitution operator “;” (semicolon). Explain why.

Answer: Since the sequential substitution allows one substitution to follow another, there is a state defined in between the two substitutions. Abstract machine specifications must have atomic operations, with no intermediate states.

Comment: The atomicity is the key idea.

(3 marks)

- (d) In the substitution **CHOICE G OR H END**, the Generalised Substitution Language form is $G \square H$, and the substitution $[G \square H]R$ can be rewritten as $[G]R \wedge [H]R$. Explain why.

Answer: Since either choice can satisfy the invariant, it cannot matter which one is chosen. Therefore both must satisfy the invariant independently of the other.

Comment:

(3 marks)

4. From the **SimpleBank** machine, we have the following fragments:

```
...
VARIABLES
    accounts , balance
INVARIANT
    accounts  $\subseteq$  ACCOUNT  $\wedge$ 
    balance  $\in$  accounts  $\rightarrow$   $\mathbb{N}$ 
INITIALISATION
    accounts , balance := { } , { }
...
OPERATIONS
    Withdraw ( amount , account )  $\hat{=}$ 
        PRE amount  $\in$   $\mathbb{N}$   $\wedge$  account  $\in$  accounts  $\wedge$  amount  $\leq$  balance ( account )
        THEN
            balance := balance  $\Leftarrow$  { account  $\mapsto$  balance ( account ) - amount }
        END ;
```

Part of one of the proof obligations is to establish

$$balance(account) - amount \in \mathbb{N}$$

Prove this.

(12 marks)

Answer:

- (a) $balance(account) \in \mathbb{N}$ (From the invariant $balance \in accounts \rightarrow \mathbb{N}$, and the precondition $account \in accounts$)
- (b) $amount \in \mathbb{N}$ (From the precondition)
- (c) $amount < balance(account)$ (From the precondition)
- (d) Therefore, since if $a \in \mathbb{N}$ and $b \in \mathbb{N}$ and $b < a$ then $a - b \in \mathbb{N}$, QED

3 marks for each step

5. The following machines are taken from the B demo machines, and describe a person data base. Non-relevant details have been omitted. Read through this listing, then answer the questions on the following page.

MACHINE *data_base_context*

SETS

$SEX = \{ man, woman \};$
 $STATUS = \{ living, dead \}$

END

MACHINE *person_data_base* (*maxpers*)

SEES

data_base_context , *Bool_TYPE*

VARIABLES

person , *sex* , *status* , *mother* , *husband*

INVARIANT

$person \in 0 .. maxpers \wedge$
 $sex \in 1 .. person \rightarrow SEX \wedge$
 $status \in 1 .. person \rightarrow STATUS \wedge$
 $mother \in 1 .. person \leftrightarrow \text{dom} (husband) \wedge$
 $husband \in sex^{-1} [\{ woman \}] \leftrightarrow sex^{-1} [\{ man \}]$

INITIALISATION

$person, sex, status, mother, husband := 0, \{\}, \{\}, \{\}, \{\}$

OPERATIONS

mod_mother (*xx* , *yy*) $\hat{=}$
PRE
 $xx \in 1 .. person \wedge yy \in \text{dom} (husband)$
THEN
 $mother (xx) := yy$
END ;

...

DEFINITIONS

$wife \hat{=} husband^{-1}$

END

(a) Why is *data_base_context* a separate machine?

Answer: So that the sets *SEX* and *STATUS* are defined independently, and can be used (SEEN) by other machines (that include *person_data_base*).

Comment: straightforward.

(3 marks)

(b) Explain the meaning of the last two invariant clauses (those that mention *husband*).

Answer:

i. *mother* is a partial function from person (numbers) to the domain of the husband function, i.e., it identifies the mother of the argument person, and this mother must be female. It is partial to avoid having an infinite closure (not all mothers defined in this database).

ii. *husband* is a partial injection from all women persons to all man persons. It is partial because not all women are married, and an injection since it is one-to-one (one husband per wife).

Comment: The first part was not well done, with many people failing to see that it defines the *mother* of a person. The use of the domain of the husband function is just to say that mothers must be female, not that they must be married(!) Quote: "A husband is a man that has a woman".

(3 marks)

(c) Derive the proof obligations for **mod_mother**.

Answer: From the basic proof obligation relation $P \wedge I \Rightarrow [G]I$, we get

$$\begin{aligned} &xx \in 1..person \wedge yy \in \text{dom}(\text{husband}) \wedge \\ &\quad person \in 0..maxpers \wedge \\ &\quad sex \in 1..person \rightarrow SEX \wedge \\ &\quad status \in 1..person \rightarrow STATUS \wedge \\ &mother \in 1..person \leftrightarrow \text{dom}(\text{husband}) \wedge \\ &husband \in sex^{-1}[woman] \leftrightarrow sex^{-1}[man] \Rightarrow [mother(xx) := yy] \\ &\quad mother \in 1..person \leftrightarrow \text{dom}(\text{husband}) \end{aligned}$$

where unchanged consequents have been removed from the right hand side

Comment: OK

(3 marks)

(d) Write an operation **is_sibling**(*xx,yy*) that returns TRUE if *xx* and *yy* are siblings (brothers or sisters), and FALSE otherwise. (You may assume the operation *bool* that converts a predicate to a Boolean value.)

Answer:

```
res ← is_sibling ( xx , yy ) ≐
PRE
  xx ∈ dom (mother) ∧ yy ∈ dom (mother)
THEN
  res := bool ( mother(xx) = mother(yy) )
END ;
```

Comment: Nearly everyone missed the precondition (so did I at first!)

(3 marks)

6. The following code is taken from the complete Library machine discussed in lectures, and relates to the operation of reserving a book. Many non-relevant details have been omitted. Read through this listing, then answer the questions on the following page.

INVARIANT

$$\begin{aligned}
& \text{books_on_loan} \in \text{books_in_library} \leftrightarrow \text{users} \wedge \\
& \text{dom} (\text{books_on_loan}) \cap \text{books_on_shelf} = \{ \} \\
& \text{reserved} \in \text{books_in_library} \leftrightarrow \text{iseq} (\text{users}) \wedge \\
& \text{reserved} \in \text{books_in_library} \leftrightarrow \text{seq}_1 (\text{users}) \wedge \\
& \forall \text{book} . (\text{book} \in \text{dom} (\text{reserved}) \Rightarrow \\
& \quad \text{size} (\text{reserved} (\text{book})) \leq \text{maxreserve}) \wedge \\
& \text{dom} (\text{reserved}) \subseteq \text{dom} (\text{books_on_loan}) \cup \text{dom} (\text{collect}) \wedge \\
& \text{collect} \in \text{books_in_library} \leftrightarrow \text{users} \wedge \\
& \text{dom} (\text{collect}) \cap \text{dom} (\text{books_on_loan}) = \{ \} \wedge \\
& \text{dom} (\text{collect}) \cap \text{books_on_shelf} = \{ \}
\end{aligned}$$

OPERATIONS

```

Reserve ( user , book )  $\hat{=}$ 
  PRE user  $\in$  users  $\wedge$  book  $\in$  books_in_library  $\wedge$ 
    book  $\in$  dom ( books_on_loan )  $\cup$  dom ( collect )  $\wedge$ 
    ( book  $\in$  dom ( books_on_loan )  $\Rightarrow$ 
      books_on_loan ( book )  $\neq$  user )  $\wedge$ 
    ( book  $\in$  dom ( collect )  $\Rightarrow$  collect ( book )  $\neq$  user )  $\wedge$ 
    ( book  $\in$  dom ( reserved )  $\Rightarrow$ 
      size ( reserved ( book ) )  $\neq$  maxreserve )  $\wedge$ 
    ( book  $\in$  dom ( reserved )  $\Rightarrow$ 
      user  $\notin$  ran ( reserved ( book ) ) )
  THEN IF book  $\notin$  dom ( reserved )
    THEN reserved ( book ) := [ user ]
    ELSE reserved ( book ) := reserved ( book )  $\leftarrow$  user
  END
END ;

```

(a) Why is the intersection between $\text{dom}(\text{books_on_loan})$ and books_on_shelf empty?

Answer:

Comment:

(2 marks)

(b) Why is *reserved* declared as *both* an injective sequence, and a non-empty sequence?

Answer:

Comment:

(3 marks)

(c) Explain each of the conjuncts of the **Reserve** operation precondition in prose form (short dot points will suffice).

Answer:

i. xx

ii. xx

iii. xx

iv. xx

v. xx

vi. xx

vii. xx

Comment:

(7 marks)

7. Perform the following predicate transformations and simplify the resultant predicate:

(a) $[x := \text{new}, y := \{\text{new}, \text{old}\}]p \subseteq \text{books} - \{\text{new}, x\} \cup y$

Answer: $p \subseteq \text{books} - \{\text{new}, \text{new}\} \cup \{\text{new}, \text{old}\}$, which can be simplified to
 $p \subseteq \text{books} \cup \{\text{old}\}$

(3 marks)

(b) $[p \in \text{books} \Rightarrow \text{holdings} := \text{holdings} \cup \{p\}] \forall z. (z \in \text{books} \Rightarrow z \in \text{holdings})$

Answer: $p \in \text{books} \Rightarrow \forall z. (z \in \text{books} \Rightarrow z \in \text{holdings} \cup \{p\})$

(3 marks)

(c) Show that both of the sequences $y' := x + y; x' := y - x; y' := y' - y$ and $t := x; x' := y; y' := t$ establish $x' = y \wedge y' = x$, that is, they swap the values of x and y

Answer: Express each as predicate transforms:

$$[y' := x + y; x' := y - x; y' := y' - y]x' = y \wedge y' = x$$

$$[t := x; x' := y; y' := t]x' = y \wedge y' = x$$

and expand:

$$[y' := x + y; x' := y - x]x' = y \wedge y' - y = x$$

$$[t := x; x' := y]x' = y \wedge t = x$$

then

$$[y' := x + y]y - x = y \wedge y' - y = x$$

$$[t := x]y = y \wedge t = x$$

then

$$y - x = y \wedge x + y - y = x$$

$$y = y \wedge x = x$$

whence true

(6 marks)

Comment:

8. The following machine defines a set of natural numbers. The machine has just two operations, to add and remove a natural number from the represented set.

MACHINE *NatSet* (*maxnat*)

VARIABLES

natset

INVARIANT

$natset \subseteq 0 .. maxnat$

INITIALISATION

$natset := \{\}$

OPERATIONS

add (*nn*) $\hat{=}$

PRE $nn \in 0 .. maxnat$

THEN $natset := natset \cup \{ nn \}$

END ;

remove (*nn*) $\hat{=}$

PRE $nn \in 0 .. maxnat$

THEN $natset := natset - \{ nn \}$

END

val \leftarrow **cardinality** $\hat{=}$

BEGIN $val := card (natset)$

END

END

The following machine refines the above machine, and performs a data refinement by representing the set of natural numbers in a sequence. For example, if *NatSet* stores the set $\{13, 18, 56\}$, then one possible state of the refinement *NatSetR* is $\langle 18, 56, 13 \rangle$. (Others may be given by a different order of adding the elements, for example $\langle 18, 13, 56 \rangle$ is another possible representation of the original machine state.)

In the spaces provided,

- (a) Define the refinement relation that constrains the state of the refining machine to be consistent with that of the refined (original) machine.
- (b) Define the operations *add*, *remove* and *cardinality* of the refining machine.

REFINEMENT *NatSetR*

REFINES *NatSet*

VARIABLES

setvals

INVARIANT

$setvals \in \text{iseq}(\mathbb{N}) \wedge$

Answer: $\text{ran}(setvals) = \text{natset} \heartsuit\heartsuit\heartsuit$

INITIALISATION

$setvals := []$

OPERATIONS

add (*nn*) $\hat{=}$

Answer:

```
PRE  $nn \in \mathbb{N} \heartsuit$ 
THEN IF  $nn \notin \text{ran}(setvals) \heartsuit$ 
  THEN  $setvals := setvals \leftarrow nn \heartsuit$ 
  END
END
```

;

remove (*nn*) $\hat{=}$

Answer:

```
PRE  $nn \in \mathbb{N}$ 
THEN IF  $nn \in \text{ran}(setvals) \heartsuit$ 
  THEN
    LET  $ii \text{ BE } ii = setvals^{-1}(nn) \heartsuit$ 
    IN  $setvals := setvals \uparrow (ii - 1) \heartsuit^a setvals \downarrow (ii) \heartsuit$  END
  END
END
```

;
 $val \leftarrow \text{cardinality} \hat{=}$

Answer:

BEGIN $val := \text{size} (\text{setvals})$ **END** ♡♡

END

(12 marks)

9. The following tutorial example defines the *BagMath* machine. Explain i) the definition *Bag*, and ii) the use of the 3 constants *emptyBag*, *BagCount*, *BagUnion*. You may assume that the machine *Value_TYPE* defines the set *VALUE*.

MACHINE *BagMath*

SEES *Value_TYPE*

CONSTANTS

emptyBag , *BagCount* , *BagUnion*

PROPERTIES

$$\begin{aligned} & \textit{emptyBag} \in \textit{VALUE} \rightarrow \{ 0 \} \wedge \\ & \textit{BagCount} \in \textit{Bag} (\textit{VALUE}) \rightarrow (\textit{VALUE} \rightarrow \mathbb{N}) \wedge \\ & \forall bb . (bb \in \textit{Bag} (\textit{VALUE}) \Rightarrow \textit{BagCount} (bb) = \textit{emptyBag} \triangleleft bb) \wedge \\ & \textit{BagUnion} \in \textit{Bag} (\textit{VALUE}) \times \textit{Bag} (\textit{VALUE}) \rightarrow \textit{Bag} (\textit{VALUE}) \wedge \\ & \forall (b1 , b2) . (b1 \in \textit{Bag} (\textit{VALUE}) \wedge b2 \in \textit{Bag} (\textit{VALUE}) \Rightarrow \\ & \quad \textit{BagUnion} (b1 , b2) = \{ vv , nn \mid vv \in \textit{dom} (b1) \cup \textit{dom} (b2) \wedge \\ & \quad \quad (nn \in \mathbb{N}_1 \wedge nn = \textit{BagCount} (b1) (vv) + \textit{BagCount} (b2) (vv)) \}) \end{aligned}$$

DEFINITIONS

$\textit{Bag} (X) \hat{=} X \mapsto \mathbb{N}_1$

END

Answer:

emptyBag defines an empty bag as a total function that maps every value of type *VALUE* onto zero, representing the fact that there are zero of every possible values in the bag.

BagCount is a function that takes a bag of *VALUE*s, and returns a function that defines how many instances of each value of type *VALUE* there are in the bag.

BagUnion is a function that takes two bags of identical *VALUE*s, and return a single bag containing the contents of both bags.

Bag is a model of a bag of values, where there may be more than one instance of any particular value. It models the bag as a function that returns the number of instances of any particular value of type *X*.

Comment: Not well done, particularly by those who skipped the last tutorial! Again, the emphasis was upon demonstrating an insight into the use of the notation, rather than just parroting definitions of the notation used.

(12 marks)

10. In an example discussed in lectures, the specification of a *SimpleLibrary* machine contained the operation

```

MACHINE SimpleLibrary ( BOOK , maxuser )
...
OPERATIONS Borrow ( user , book )  $\hat{=}$ 
    PRE user  $\in$  users  $\wedge$  book  $\in$  books_on_shelf
    THEN books_on_shelf := books_on_shelf - { book } ||
        books_on_loan := books_on_loan  $\cup$  { book  $\mapsto$  user }
    END

```

The following B fragments define a refinement to *SimpleLibrary*

```

MACHINE SimpleLibraryAPI ( BOOK , maxuser )
    CONSTRAINTS maxuser  $\in$   $\mathbb{N}1$ 
    INCLUDES SimpleLibrary ( BOOK , maxuser )
    SETS
        RESPONSE = { OK , BookInLibrary , NoNewUsers , NotRegisteredUser ,
            BookNotForLoan , BookNotOnLoan }
    OPERATIONS
        ...
    END

```

Define a **robust** operation $response \leftarrow BorrowR(book)$ that has trivial preconditions, and borrows a book *book* for user *user* from the library with response *OK* if it is available to be borrowed, and otherwise returns an appropriate error response. (*You may continue your answer overleaf*)

(Question 10 Answer continued)

Answer:

```
MACHINE SimpleLibraryAPI ( BOOK , maxuser )
CONSTRAINTS maxuser ∈ ℕ1
INCLUDES SimpleLibrary ( BOOK , maxuser )
SETS
    RESPONSE = { OK , BookInLibrary , NoNewUsers , NotRegisteredUser ,
                BookNotForLoan , BookNotOnLoan }
OPERATIONS
    response ← BorrowR ( user , book ) ≐
    PRE user ∈ USER ∧ book ∈ BOOK ♡♡
    THEN
    SELECT
        user ∉ users ♡ THEN response := NotRegisteredUser ♡
    WHEN
        book ∈ books_on_shelf ♡♡ THEN response := BookNotForLoan ♡
    ELSE
        Borrow ( user , book ) || ♡♡♡
        response := OK ♡
    END
END ;
END /*♡ comments*/
```

Comment: The marks for each part are shown with ♡ symbols. One mark was lost if the *Borrow* operation was not used, but reimplemented from *SimpleLibrary*. An additional mark was given for any form of reflective comment or explanation. Generally OK, but there were few full marks for this Q.

(12 marks)

END OF EXAMINATION QUESTIONS