

**CSE4213 Examination
2005
Instructions to Candidates**

1. Examination time 2 hours
2. Answer all questions
3. There are 10 questions
4. Each question is worth 8 marks
5. Total marks 80
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 4 attributes concerning non-determinism and its role in the B Method.

(a)

(2 marks)

Answer: Gives weaker specification and/or preconditions

(b)

(2 marks)

Answer: Allows implementor freedom of choice

(c)

(2 marks)

Answer: Avoids over-specification

(d)

(2 marks)

Answer: non-computational: the emphasis is upon the abstract properties of the solution, not how it is computed

Answer:

(e) Consistent with the Principle of Semantic Emphasis: The *what* and not the *how*.

Answer: Two marks for each correct answer

Comment: 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. You are a recent graduate, and are now working for a large software development company. The company announces that it has been awarded a contract for a safety critical system. You remember that you learnt about how the B Method has been used to build the signalling system for the driverless trains in the Paris Metro. But nobody in the software company has heard about the B Method. What do you tell your colleagues about the B Method? (Remember that they are busy people, and will not listen to lengthy explanations. That is, keep your answer brief: dot points will suffice.)

(8 marks)

Answer: The points that should be made include

- The B Method will guarantee consistency between the delivered software and the agreed specification.
- There is a formal proof that links each specified operation to the corresponding implemented operation.
- A formal proof takes the place of discrete system testing.
- Such proofs cover the complete domain, not just discrete testing points.
- Code can be automatically generated from the (refined) specification.
- A robust, reliable, correct piece of software will save the company money in the long run.

Two marks for each correct point.

Comment:

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

(2 marks)

Answer:

Comment:

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

(2 marks)

Answer:

Comment:

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

(2 marks)

Answer:

Comment:

- (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.

(2 marks)

Answer:

Comment:

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  $\ominus$  { account  $\mapsto$  balance ( account ) - amount }  
    END ;
```

Part of one of the proof obligations is to establish

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

Prove this.

(8 marks)

Answer:

- (a) $\textit{balance}(\textit{account}) \in \mathbb{N}$ (From the invariant $\textit{balance} \in \textit{accounts} \rightarrow \mathbb{N}$, and the precondition $\textit{account} \in \textit{accounts}$)
- (b) $\textit{amount} \in \mathbb{N}$ (From the precondition)
- (c) $\textit{amount} < \textit{balance}(\textit{account})$ (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

5. A common form of proof obligation arises from function updating. If f is a function $f \in s \rightarrow t$ then a typical update substitution might be

$$f := f \triangleleft \{a \mapsto b\}$$

This gives a proof obligation of the form

$$f \triangleleft \{a \mapsto b\} \in s \rightarrow t$$

- (a) Explain the use of rewriting rules in theorem proving

(3 marks)

Answer: One mark for mentioning each of

- jokers and/or pattern matching
- Forward inferencing: apply rewrite rules to antecedent (P in $P \Rightarrow Q$)
- Backward inferencing: apply rewrite rules to consequent (Q in $P \Rightarrow Q$)

Comment: Many students recognized that it was an exercise in breaking down goals into simpler subgoals, but then did not make the connection into forward and backward inferencing.

- (b) Using the rewrite rules

$$f \in u \rightarrow t \tag{1}$$

$$u \cup \{a\} = s \tag{2}$$

$$b \in t \tag{3}$$

break the proof obligation down into three subgoals for the update of

$$balance := balance \triangleleft \{acc \mapsto 0\}$$

where the invariant states that $balance \in accounts \rightarrow \mathbb{N}$

(4 marks)

Answer: The proof obligation is

$$balance \triangleleft \{acc \mapsto 0\} \in accounts \rightarrow \mathbb{N}$$

which, when matching against the jokers in the rewrite antecedent

$$f \triangleleft \{a \mapsto b\} \in s \rightarrow t$$

gives a binding for every joker except u . Using the hypothesis $f \in u \rightarrow t$ (see below), we match this to $accounts$ and get the three new subgoals:

$$balance \in accounts \rightarrow \mathbb{N} \tag{4}$$

$$accounts \cup \{acc\} = accounts \tag{5}$$

$$0 \in \mathbb{N} \tag{6}$$

Comment: Oh dear. The dreaded missing function symbols bug struck here. The exam paper did not have any of the function arrows. Most of them were caught due to the obvious blank spaces, but the $balance \in accounts \rightarrow \mathbb{N}$ lost the $\rightarrow \mathbb{N}$ part, and the question becomes impossible if you do not know this. A few students recognized enough of the problem (it comes straight from the lecture notes) to replace this, but most simply inferred the predicate $balance \in dom(accounts) \rightarrow ran(accounts)$, which did not help much.

(c) Explain why the variable *accounts* must be treated as a special case.

(1 marks)

Answer: It is a special case, since it has no direct joker matching for *u*. Instead, we must check that there is a hypothesis (antecedent) matching $f \in u \rightarrow t$. Fortunately, there is, and *u* can be instantiated to *accounts*.

Comment: If you didn't spot the bug identified above, there was not much hope of collecting a mark for this one.

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

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

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?
(1 marks)

Answer:

Comment:

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

Answer:

Comment:

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

Answer:

- i. xx
- ii. xx
- iii. xx
- iv. xx
- v. xx
- vi. xx
- vii. xx

Comment:

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

(a) $[x := 5 - y]x - 5 > 0$

(2 marks)

Answer: $y < 0$

Comment:

(b) $[car, carsInPark := newcar, carsInPark - \{car\}]card(carsInPark \cup \{newcar\}) < capacity$
(2 marks)

Answer: $card(carsInPark - \{car\} \cup \{newcar\}) < capacity$

Comment: Don't make the mistake of performing the substitution for *carsInPark* first then substituting for $\{car\}$ in the resultant predicate (as many did)! The substitutions must be done in parallel, and *car* does not appear in the initial predicate, so there is no substitution made for it. (This was a sneaky question, I admit.)

(c) $[dir \in \{up\} \Rightarrow mov := mov \Leftarrow \{lft \mapsto dir\}] \exists l.(l \in LIFT \Rightarrow mov(l) = up)$

(2 marks)

Answer: $dir \in \{up\} \Rightarrow \exists l.(l \in LIFT \Rightarrow mov \Leftarrow \{lft \mapsto dir\}(l) = up)$ A bonus mark for recognizing that this in turn simplifies to true.

Comment: straightforward, but elaborate, which threw a few into panic (or inaction). The predicate is true (since when $l = lft$, the function application does yield *up*).

(d) $[y := x + 1; z := y - 1] x < z + y$

(2 marks)

Answer: Rewrite as $[y := x + 1]([z := y - 1]x < z + y)$ and then expand: $[y := x + 1]x < y - 1 + y$, which in turn expands to $x < 2x + 1$ which is a tautology (always true).

Comment: The rule is $[G; H]P \equiv [G]([H]P)$, that is, the second substitution is performed first. The majority of students got this right, but interestingly, most of the better students did not! (This did worry me slightly, but a good way of checking that your result is sensible is to try substituting a few actual values for x, y, z , and running the substitutions as though it were an actual program. If you try (say) 23,45,16 as initial values, you end up with 23,24,23 as the final values, when $23 < 23 + 24$ is clearly true.)

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

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 two operations *add* and *remove* of the refining machine.

REFINEMENT *NatSetR*

REFINES *NatSet*

VARIABLES

setvals

INVARIANT

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

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

INITIALISATION

$setvals := []$

OPERATIONS

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

Answer:

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

END

9. Consider the following data model specification for a simple *String* machine. The machine *SEES* the type machine *CHAR_Type*, which defines the set *CHAR*. *strs* is the set of all currently defined strings, and *values* defines the string value of each of the defined strings.

In the space provided, complete the definition of the *Substring* operation, which returns a new string given by the substring starting at *start* and finishing at *end* (inclusive, 1-origin) within the given string *str1*. (Hint: A new string is an element of the set *STRINGS* that is not in the current set of defined strings *strs*.)

(8 marks)

MACHINE *String*

SEES

CHAR_Type

SETS

STRINGS

VARIABLES

strs , *values*

INVARIANT

$strs \subseteq STRINGS \wedge$

$values \in strs \rightarrow seq(CHAR)$

INITIALISATION

$strs, values := \{\}, \{\}$

OPERATIONS

$str \leftarrow \text{Substring} (str1 , start , end) \hat{=}$

Answer:

PRE $str1 \in strs \heartsuit \wedge start \in 1..size(values(str1)\heartsuit_1) \wedge$

$end \in 1..size(values(str1)) \heartsuit_2 \wedge end \geq start \heartsuit$

THEN

ANY $nstr \heartsuit_3$ **WHERE** $nstr \in STRINGS - strs \heartsuit$

THEN

$values(nstr) := values(str1) \downarrow (start - 1) \heartsuit_4 \uparrow (end - start + 1) \heartsuit_5 \parallel$

$strs := strs \cup \{nstr\} \heartsuit \parallel$

$str \leftarrow nstr \heartsuit$

END

END

Deduct one mark for each missing construct marked \heartsuit . ($\heartsuit_{1,2}$ $size(..)$ is worth one, and $values(str1)$ is worth 1 **providing each is repeated** (deduct 0.5 marks for each missing occurrence), \heartsuit_3 the non-deterministic instantiation of the new string is worth 1, and $\heartsuit_{4,5}$ the take and drop operators are each worth 1 mark.

END

Comment:

10. (a) One of the rules for establishing loop correctness is the **The I Rule**: The initialisation substitution establishes the invariant. State the other four of the five rules for loop correctness.

Answer: Assume the following notation in all the following:

$[H; \text{WHILE } P \text{ DO } G \text{ VARIANT } E \text{ INVARIANT } Q \text{ END}] R$

i.

(1 marks)

Answer: The F Rule: When the loop ends, the predicate R is true
 $\text{not } P \ \& \ Q \Rightarrow R$

ii.

(1 marks)

Answer: The T1 Rule: If the invariant is true, the variant is a natural number
 $Q \Rightarrow E : \text{NAT}$

iii.

(1 marks)

Answer: The T2 Rule: Each iteration of the loop reduces the variant:
 $P \ \& \ Q \Rightarrow [y := E] [G] E < y$

iv.

(1 marks)

Answer: The P Rule: Each iteration of the loop preserves the invariant:
 $P \ \& \ Q \Rightarrow [G] Q$

Comment: Mostly OK. I gave a mark to those who just named the rule, but I was looking for the formal predicates. If you didn't know them, you couldn't do the next part, so it didn't matter. Many people lost a mark for adding a P to the antecedent of the T1 rule.

- (b) Evaluate these four loop correctness rules for the following loop to show that they all are true, and the loop is therefore properly formed.

```
[ xx := 10 ;
  WHILE xx > 0 DO
    xx := xx - 1 VARIANT xx INVARIANT xx : NAT
  END ] xx = 0
```

i. The I rule is obviously true

ii.

(1 marks)

Answer:

$$\begin{aligned} \text{not } P \ \& \ Q \Rightarrow R \\ &= \text{not } xx > 0 \ \& \ xx : \text{NAT} \Rightarrow xx = 0 \end{aligned}$$

true

iii.

(1 marks)

Answer:

$$Q \Rightarrow E : \text{NAT} = xx : \text{NAT} \Rightarrow xx : \text{NAT}$$

TRUE

iv.

(1 marks)

Answer:

$$\begin{aligned} P \ \& \ Q \Rightarrow [y := E] \ [G] \ E < y \\ &= xx > 0 \ \& \ xx : \text{NAT} \Rightarrow \\ &\quad [y:=xx] \ [xx := xx - 1] \ xx < y \\ &= xx > 0 \ \& \ xx : \text{NAT} \Rightarrow \\ &\quad [y:=xx] \ xx-1 < y \\ &= xx > 0 \ \& \ xx : \text{NAT} \Rightarrow \\ &\quad xx-1 < xx \end{aligned}$$

which is true

v.

(1 marks)

Answer:

$$\begin{aligned} P \ \& \ Q \Rightarrow [G] \ Q \\ &= xx > 0 \ \& \ xx : \text{NAT} \Rightarrow \\ &\quad [xx := xx - 1] \ xx : \text{NAT} \\ &= xx > 0 \ \& \ xx : \text{NAT} \Rightarrow \\ &\quad xx-1 : \text{NAT} \end{aligned}$$

clearly true

Comment: The main reason people lost marks on this part was that they could not do the substitutions properly.

END OF EXAMINATION QUESTIONS