

**CSE4213 Examination  
2003  
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. State 4 reasons as to why formality is important to software specification.

- (a) **Answer:** Increase in reliability
- (b) **Answer:** Assists the process of programming by contract
- (c) **Answer:** Vital for mission critical and/or safety reasons
- (d) **Answer:** Discharging proof obligations is the counterpart in software to testing in other engineering domains.
- (e) **Answer:** software operates over discrete/digital domains, and analog analysis methods do not apply.

**Answer:** Two marks for each correct answer

**Comment:** 2002 Very straightforward: few students had any difficulty with this question.

2. Write a brief comment (no more than 100 words) on the role of discharging proof obligations in B.

**Answer:** A B specification defines a **complete, correct** set of **behaviours**. Each **operation** and the **initialisation** must establish an **invariant**, a **predicate** that defines the correct behaviour. Proof obligations are **theorems** that follow from the operations and initialisation.

Discharging the proof obligations means **proving** these theorems, which gives **mathematical rigour** for the **correctness** of the specification, and any **refinement/implementation** derived from it.

A proof validates behaviour in a **complete domain**, not just at a single point.

**Comment: 2002** This question was based upon one particular slide and several comments in one lecture, but some students had obviously not taken this in! The emphasis was upon the historical evolution of formality, but other aspects could get full marks if adequately argued.

**Comment: 2003** A B specification defines a *complete, correct* set of *behaviours*. Each operation, and the initialization, must establish an *invariant*, and the proof obligations are *theorems* that the *operations* and *initialization* must obey. *Proving* these theorems gives *mathematical rigour* for the *correctness* of the specification and any *refinement* or implementation developed from it. (1 mark per italicized word).

3. 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.

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*. Each substitution is specified through an analogous statement in the *Abstract Machine Notation*.

**Comment: 2002** 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.

Not that well done (some people did not even attempt the question). Particularly missing in many answers was the idea of maintaining the invariant. **Comment: [2003**

4. The following B specification has been discussed in lectures. Read through this specification, then answer the questions following the specification (overleaf).

```

MACHINE SimpleLibrary ( BOOK , maxuser )
CONSTRAINTS maxuser  $\in \mathbb{N}_1$ 
SETS USER
PROPERTIES  $\text{card} ( \textit{USER} ) = \textit{maxuser}$ 
VARIABLES users , books_in_library , books_on_shelf , books_on_loan
INVARIANT
  users  $\subseteq \textit{USER} \wedge$ 
  books_in_library  $\subseteq \textit{BOOK} \wedge$ 
  books_on_shelf  $\subseteq \textit{books_in_library} \wedge$ 
  books_on_loan  $\in \textit{books_in_library} \leftrightarrow \textit{users} \wedge$ 
  books_on_shelf = books_in_library -  $\text{dom} ( \textit{books_on_loan} )$ 
INITIALISATION
  users ,
  books_in_library ,
  books_on_shelf ,
  books_on_loan := { } , { } , { } , { }
OPERATIONS
  AddBook ( book )  $\hat{=}$ 
    PRE book  $\in \textit{BOOK} \wedge \textit{book} \notin \textit{books_in_library}$ 
    THEN books_in_library := books_in_library  $\cup \{ \textit{book} \}$  ||
      books_on_shelf := books_on_shelf  $\cup \{ \textit{book} \}$ 
    END ;
  newuser  $\leftarrow$  NewUser  $\hat{=}$ 
    PRE users  $\neq \textit{USER}$ 
    THEN
      ANY user
      WHERE user  $\in \textit{USER} - \textit{users}$ 
      THEN users := users  $\cup \{ \textit{user} \}$  ||
        newuser := user
      END
    END ;
  Borrow ( user , book )  $\hat{=}$ 
    PRE user  $\in \textit{users} \wedge \textit{book} \in \textit{books_on_shelf}$ 
    THEN books_on_shelf := books_on_shelf - { book } ||
      books_on_loan := books_on_loan  $\cup \{ \textit{book} \mapsto \textit{user} \}$ 
    END ;
  Return ( book )  $\hat{=}$ 
    PRE book  $\in \text{dom} ( \textit{books_on_loan} )$ 
    THEN books_on_shelf := books_on_shelf  $\cup \{ \textit{book} \}$  ||
      books_on_loan := { book } books_on_loan
    END ;
  user  $\leftarrow$  Borrowed ( book )  $\hat{=}$ 
    PRE book  $\in \text{dom} ( \textit{books_on_loan} )$ 
    THEN user := books_on_loan ( book )
    END
END

```

- (a) Explain why *BOOK* is a parameter of the machine, while *USER* is local to the machine.

**Answer:** Presumably because the author wished to allow the specification to be instantiated across different externally defined sets of books, while the distinction between different users was not externally constrained, and hence could be hidden from the external interface. ♡

(Bonus mark) The return by *NewUser* of a value within the set *USER* does not violate this answer, since there is nothing an external client of this specification can do with the value except supply it as a parameter to the *Borrow* operation. ♡

- (b) Why is *books\_on\_loan* defined as a partial function, rather than a total function?

**Answer:** Because not all books in the domain (*books\_in\_library*) might be on loan. If the function were total, then all books in the library would have to be on loan.

- (c) Give a formal statement of the proof obligation for the initialisation.

**Answer:** The proof obligation for initialisation is  $[G]I$  ♡, where  $G$  is the initialisation substitution, and  $I$  is the invariant. Substituting, we get:

$$\begin{aligned} & [users, books\_in\_library, books\_on\_shelf, books\_on\_loan := \{\}, \{\}, \{\}, \{\}] \heartsuit \\ & (users \subseteq USER \wedge books\_in\_library \subseteq BOOK \wedge \\ & books\_on\_shelf \subseteq books\_in\_library \wedge \\ & books\_on\_loan \in books\_in\_library \leftrightarrow users \wedge \\ & books\_on\_shelf = books\_in\_library - \text{dom}(books\_on\_loan)) \heartsuit \end{aligned}$$

When the substitutions are made (not required), this becomes trivially true.

- (d) The second substitution in the operation *Borrow* could have been written in function update form as  $books\_on\_loan(book) := \{book \mapsto user\}$ . Give the variable update form of this ( $books\_on\_loan := \dots$ ), and explain how this is different to the use of the union operator. Why is the union valid in this case?

**Answer:**  $books\_on\_loan := books\_on\_loan \triangleleft \{book \mapsto user\}$  ♡

The function override is different because it specifically removes any previous maplet with the same domain value, whereas the union operator will add a new maplet with potentially the same domain value (and hence *books\_on\_loan* would no longer be a function. ♡

It is valid here because we know from the precondition and invariant that *book* cannot be in the domain of *books\_on\_loan*. ♡

**Comment: 2002** This question started to separate the levels of ability. The point many students omitted was the need to move the specification in the direction of developing an **algorithm** to compute (rather than define) the required value.

**Comment: 2003** ditto. a) I accept any answer that made an external/internal distinction. b) almost universally correct. c) many stated that  $I \wedge P[G]I$ , which is not the case as  $I$  is not initially established. But I overlooked this. d) The last mark was hard to get.

5. Explain how the mathematical model of a function as used in B can be used to model both structured data values (as in the C data structure `struct`) and object based systems (as in Object Oriented languages). (Hint: Recall the list machine discussed in tutorials and the file machine discussed in lectures.)

**Answer:** Functions are mappings from **domain** sets to **range** sets. Each mapping can be thought of as an ordered pair drawn from the set  $DOMAIN \times RANGE$ . The distinction between structs and objects then comes down to the nature of the domain set.

To model structured values, the domain represents a set of structured objects, such as the elements of a list. A separate function is then used to map from this set of objects to the corresponding values of the fields of the structure, so:

$$\begin{aligned} value &\in POINTERS \mapsto TYPE \\ next &\in POINTERS \mapsto POINTERS \end{aligned}$$

To model object orientation, the domain represents the set of objects of a given class. Each class attribute is then modelled by a function of the same name, which maps objects to the values of their corresponding attribute, so:

$$\begin{aligned} attribute_1 &\in CLASS \rightarrow TYPE_1 \\ attribute_2 &\in CLASS \rightarrow TYPE_2 \\ &\dots \end{aligned}$$

**Comment:** Really a simple concept taken from the lectures on the simple file system, and given the emphasis I had given in tutorials to the role of functions in B, I was surprised at how hard many students found this question. Many were confused by the reference to OO, and thought it a question about reuse of specifications and/or inheritance from specs to refinements.

There were 2 marks for each of the examples, and another 2 for each part for describing any relevant concept.

**Comment:** bf 2002 Really bookwork, but many students did not know the difference between deferred and parameter sets. There were 4 marks attached to reproducing the above B fragment: really a giveaway! The other 4 marks were given to those students who identified (and explained) that parameter sets were **externally** defined, while deferred sets were **internal** to the machine being specified.

**Comment:** bf 2003 **Very** poorly done. Many confused the inheritance ideas of SEES and EXTENDS clauses with this question, which very clearly asked how functions could be used. **Nobody** made an adequate distinction between structuring and OO (NOT the same concepts!), and nobody got full marks for this question (the best was 6/8)

6. A television station is planning to use B to specify its programming (in the non-computing sense). A day's program consists of a sequence of shows (such as *Neighbours*, *Big Brother* and *News*), each of some fixed duration (such as, but not limited to, 30/45/60 minutes).

- (a) In the specification, there are two variables: *duration* which defines the duration in minutes of any given item which is an element of the set *SHOWS*, and *program* which defines the station's program for the day. Give typing predicates for the variables (fill in the gap after the  $\in$  symbols):

$$duration \in SHOWS \leftrightarrow \mathbb{N}$$

$$program \in \text{seq } SHOWS$$

- (b) A refinement of this machine defines a new variable *whatson* with type  $whatson \in 0..1440 \rightarrow SHOWS$ , which maps an arbitrary time of day (there are 1440 minutes in a day) to the current item being transmitted by the station. Give the refinement relation between this variable and the base variables.

**Answer:**

$$\forall mins, item. ((mins, item) \in whatson) \tag{1}$$

$$\exists i, j. (i \in \text{dom } program \wedge) \tag{2}$$

$$j \in \mathbb{N} \wedge j \leq duration(i) \wedge \tag{3}$$

$$item = program(i) \tag{4}$$

$$\left( \sum k. (k < i \mid duration(k)) \right) + j = mins \tag{5}$$

where

(??) for all maplet pairs in the *whatson* function

(??) there exists an *i* and *j* where *i* indexes an item of the program

(??) and *j* is less than the duration of that program

(??) so that the indexed program equals the given item

(??) and the sum of the durations of all previous items plus the time into the current item is equal to *mins*

- (c) A television program published in a newspaper gives the program for each of a number of channels, specifically the channels 2,7,9,10,28. Give a generalisation of the *whatson* function that captures this fact.

**Answer:**

$$channel \in \{2, 7, 9, 10, 28\} \rightarrow (0..1440 \rightarrow SHOWS)$$

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

(a)  $[x := x - 1]x < y - 1$

**Answer:**  $x < y$

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

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

(c)  $[p \in booksholdings := holdings \cup \{p\}] \forall z.(z \in booksz \in holdings)$

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

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

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

**Comment: 2002** Parts a and b caused little difficulty. I did expect people to simplify  $x + 1 < y + 1$ , but did not penalise those who failed to remove the existential quantifier in b. For some reason, many students left off the LHS of the implication in c. While it is true for natural numbers that c is a tautology, you cannot simplify this further without additional constraints.

Either of the expressions  $new \notin users \wedge p \notin users - \{p\}$  or  $new \notin users$  collected two marks (the emphasis was on the substitutions, not the simplifications).

8. Consider the following data model specification for the simple list machine discussed in tutorials. In the space provided, complete the definition of the *cons* operation, which adds an element containing the value *new* to the front of the list.

**MACHINE** *SimpleList* ( *TYPE* )

**SETS**

*POINTERS*

**CONSTANTS**

*nil*

**PROPERTIES**

$nil \in POINTERS$

**VARIABLES**

*next* , *val* , *list* , *head* , *curpos*

**INVARIANT**

$list \subseteq POINTERS \wedge$

$next \in POINTERS \not\in POINTERS \wedge$

$val \in POINTERS \leftrightarrow TYPE \wedge$

$head \in list \wedge curpos \in list$

**INITIALISATION**

$list$  ,  $next$  ,  $val$  ,  $head$  ,  $curpos := \{nil\}$  ,  $\{\}$  ,  $\{\}$  ,  $nil$  ,  $nil$

**OPERATIONS**

$cons ( new ) \hat{=}$

**Answer:**

**PRE**  $new \in TYPE$

**THEN**

**ANY**  $pp$  **WHERE**  $pp \in POINTERS \wedge pp \notin list$

**THEN**

$head := pp$

$next(pp) := head$

$val(pp) := new$

$list := list \cup \{pp\}$

**END**

**END**

**END**

**Comment:**

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

```

MACHINE SimpleLibrary ( BOOK , maxuser )
  ...
  SETS USER
  ...
  OPERATIONS Borrow ( user , book )  $\hat{=}$ 
    PRE  $user \in users \wedge book \in books\_on\_shelf$ 
    THEN  $books\_on\_shelf := books\_on\_shelf - \{ book \} \parallel$ 
       $books\_on\_loan := books\_on\_loan \cup \{ book \mapsto user \}$ 
    END ;

  ...
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  $response \leftarrow BorrowR(user, book)$  that has a trivial precondition. If the book *book* is available for loan, it should record the borrowing by registered borrower *user* and return a response of *OK*. Otherwise it should return an appropriate response. (You may continue your answer overleaf)

(Question 9 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 book ∈ BOOK ♡ ∧ user ∈ USER ♡
      THEN IF user ∉ users♡
        THEN response := NotRegisteredUser
      ELSE IF book ∉ books_on_shelf♡
        THEN response := BookNotForLoan
      ELSE
        Borrow ( user , book ) ♡♡ || ♡
        response := OK ♡
      END
    END
  END ;
  ...

```

**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*. **2002** Generally OK, but there were few full marks for this Q.

10. (a) What are the five rules for establishing loop correctness?

**Answer:** Assume the following notation in all the following:

[H; WHILE P DO G VARIANT E INVARIANT Q END] R

i.

**Answer: The I Rule:** The initialisation substitution H establishes the invariant Q:

[H] Q

ii.

**Answer: The F Rule:** When the loop ends, the predicate R is true

not P & Q => R

iii.

**Answer: The T1 Rule:** If the invariant is true, the variant is a natural number

Q => E : NAT

iv.

**Answer: The T2 Rule:** Each iteration of the loop reduces the variant:

P & Q => [y := E] [G] E < y

v.

**Answer: The P Rule:** Each iteration of the loop preserves the invariant:

P & Q => [G] Q

One mark for each rule

(b) Verify **any three** of the loop correctness rules for the following loop.

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

i.

**Answer:**

[H]Q = [xx:=0]xx : NAT & xx <= 10  
= 0 : NAT & 0 <= 10

Obviously true

ii.

**Answer:**

not P & Q => R  
= not xx < 10 & xx : NAT & xx <= 10 => xx = 10

true

iii.

**Answer:**

Q => E : NAT = xx : NAT & xx <= 10 => 10-xx : NAT

Since xx ranges from 0 upto 10, then 10-xx ranges from 10 down to 0, all of which are natural numbers, QED

iv. **Answer:**

$$\begin{aligned}
 P \ \& \ Q \Rightarrow [y := E] [G] E < y \\
 &= xx < 10 \ \& \ xx : \text{NAT} \ \& \ xx \leq 10 \Rightarrow \\
 &\quad [y:=10-xx] [xx := xx + 1] 10-xx < y \\
 &= xx < 10 \ \& \ xx : \text{NAT} \ \& \ xx \leq 10 \Rightarrow \\
 &\quad [y:=10-xx] 10-(xx+1) < y \\
 &= xx < 10 \ \& \ xx : \text{NAT} \ \& \ xx \leq 10 \Rightarrow \\
 &\quad 10-(xx+1) < 10-xx \\
 &= xx < 10 \ \& \ xx : \text{NAT} \ \& \ xx \leq 10 \Rightarrow \\
 &\quad 10-1 < 10 \quad (\text{adding } xx \text{ to both sides})
 \end{aligned}$$

which is true

v. **Answer:**

$$\begin{aligned}
 P \ \& \ Q \Rightarrow [G] Q \\
 &= xx < 10 \ \& \ xx : \text{NAT} \ \& \ xx \leq 10 \Rightarrow \\
 &\quad [xx := xx + 1] xx : \text{NAT} \ \& \ xx \leq 10 \\
 &= xx < 10 \ \& \ xx : \text{NAT} \Rightarrow \\
 &\quad xx+1 : \text{NAT} \ \& \ xx+1 \leq 10
 \end{aligned}$$

clearly true