

CSE4213 Examination
2004
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 about the B Method that justify its role in software development.

(a)

(2 marks)

Answer: Increase in reliability

(b)

(2 marks)

Answer: Assists the process of programming by contract

(c)

(2 marks)

Answer: Vital for mission critical and/or safety reasons

(d)

(2 marks)

Answer: Discharging proof obligations is the counterpart in software to testing in other engineering domains.

Answer:

(e) software operates over discrete/digital domains, and analog analysis methods do not apply.

(f) Covers all aspects of the software domain from specification to code generation

Answer: Two marks for each correct answer

Comment: Generally a giveaway. Two answers only attracted half marks: “Emphasize the what and not the how” and “data hiding” are only part of the story. To get full marks for these two answers, you also had to explain why these are important in the B context and/or software engineering (since they apply to other systems as well, and are not the primary focus of B).

2. Imagine that you are a consulting software engineer. A client approaches you to build a large software system. The system is not safety critical, but there will be significant costs if the delivered system does not perform according to specification. When you explain that you will use the B Method to develop his software, he becomes agitated, and states that he cannot afford to wait too long for the system, and wants you to start "developing code immediately", and that there is an urgency to testing the system in time for delivery. What do you say to him? (Remember that he is a busy person, 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: Generally OK. Not quite as well done as Q1 (although answers quite similar). The real focus of the answers should be on addressing the concern for immediate coding, and that using B will give a better, more reliable, more correct system in the end, hence saving money!

3. (a) Explain the role of the invariant in a B specification.

(2 marks)

Answer: Either of

- defines the (legal) set of states for the corresponding abstract machine.
- constrains the variables of the specification, which defines the machine state.

or similar.

Comment: OK

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

(2 marks)

Answer: Substitutions allow/define the state transitions of the abstract machine

Comment: OK

- (c) In the Generalized Substitution Language of the B Method, the form $[G]R$ is called a *predicate transformation*. Explain why, and give a simple example.

(2 marks)

Answer: G is a substitution

R is a predicate

$[G]R$ is a predicate where the substitution G has been made in R

Example: $[xx := yy]xx \in \mathbb{N}$ becomes $yy \in \mathbb{N}$ after the substitution.

Comment: It is not enough to say that "G transforms R". You need to say something about the substitution(s).

- (d) Give the predicate transform representation of the proof obligation for the operation

WhatIsThis $\hat{=}$ **PRE P THEN G END**

(2 marks)

Answer:

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

Comment: Too many people saw the statement 5.1.1 in the concise summary and thought that was the answer ($P|G$)! The question asked for the proof obligation, not the GSL form!!

4. (a) Explain the difference between a *precondition* and a *guard*, and give an example of each. (2 marks)

Answer: A precondition states the *assumptions* under which the software will meet its specification, while a guard is an explicit evaluation that the conditions of use are met.

Comment: Generally OK. Quote (for examples): “on the feedback form it says not to write inappropriate comments (precondition), but if you threaten teachers you’ll get picked up by security (guard).” (!!)

- (b) Explain the term *weakest precondition*. Why is it important? (2 marks)

Answer: A weakest precondition is a statement of the broadest possible circumstances under which the assumptions of use hold. It is important because it allows the widest range of refinement/implementation choices.

Comment: Note that this is NOT the same as a *trivial precondition*. Quote: “Whatever you build is only as strong as your weakest precondition”.

- (c) Explain how in the traffic light specification discussed in lectures and tutorials, preconditions ensure both safety and sequencing constraints. (2 marks)

Answer: Preconditions state the assumptions under which the state may be changed. The sequencing assumptions are that the lights cycle Red,Green,Yellow,Red,..., and the safety assumptions are that when any light is showing Green or Amber, all conflicting directions show Red. So an operation to change the light to Yellow has the precondition that this light is currently Green, and that all conflicting direction lights are red.

Comment: Not generally well done.

- (d) A pedestrian light specification is to be added to the simple two way traffic light system of part (c) which INCLUDES the basic two way system. What constraints exist in the including machine (the pedestrian crossing machine) on the INVARIANT of the included machine (the two way traffic lights machine)? (2 marks)

Answer: No change of state of the included machine except through operations of the included machine. Read-only access to state variables of the included machine. Only a single (non-renaming) include.

Comment: Many students confused the including constraints with the problem domain constraints. Quote: “The including machine ... will strengthen the invariant so that pedestrians are not killed ...”

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 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?

(2 marks)

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.

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

(2 marks)

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".

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

(2 marks)

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

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

where unchanged consequents have been removed from the right hand side

Comment: OK

(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.)

(2 marks)

Answer:

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

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

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

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

(2 marks)

Answer: $y > 0$

Comment: Everyone (!) got this one right!

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

(2 marks)

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

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

(2 marks)

Answer: $dir \in \{up\} \Rightarrow \forall l.(l \in LIFT \Rightarrow mov \cup \{lft \mapsto dir\}(l) = up)$

Comment: straightforward, but elaborate, which threw a few into panic (or inaction). I gave a bonus mark to those who recognized that a further simplification is possible by using the relation image, which eliminates the universal quantifier: $dir \in \{up\} \Rightarrow (mov \cup \{lft \mapsto dir\})[LIFT] = \{up\}$ Note that if you simplify the maplet to $\{lft \mapsto up\}$ (since $dir = up$ is in the antecedent), you cannot then remove it, since you don't know whether $lft \in \text{dom}(mov)$ to start with, which it presumably wasn't, since the union operator would not make sense otherwise.

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

(2 marks)

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

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 $x > z$ is clearly false.)

8. A specification uses a data structure of a set *COLOUR* defined as

SETS

COLOUR

PROPERTIES

$COLOUR = \{ red, green, blue \}$

OPERATIONS

$add_col(cc) \hat{=} \mathbf{PRE} \ cc \in COLOUR$

$\mathbf{THEN} \ col := col \cup \{ cc \}$

END

Other operations such as *subtract_col* are similarly defined.

The specification is refined, and the data structure is refined to an array of three booleans. “Boolean” (0 or 1) is used in the C sense of boolean, where a 1 represents inclusion of the corresponding element in the set. The refinement uses the library machine *Renaming_Narr* in the following way:

CONSTANTS

red_R, green_R, blue_R, map

PROPERTIES

$red_R = 0 \wedge green_R = 1 \wedge blue_R = 2 \wedge$

$map \in \mathbb{N} \rightarrow COLOUR \wedge$

$map = \{ red_R \mapsto red, green_R \mapsto green, blue_R \mapsto blue \}$

OPERATIONS

$add_col_R(cc) \hat{=} \mathbf{PRE} \ cc \in \{ red_R, green_R, blue_R \}$

$\mathbf{THEN} \ col_Narr_STO(cc,1)$

END

Define the refinement invariant between *col* and *col_R*.

(8 marks)

Answer:

$red \in col \iff col_Narr_VAL(red_R) = 1 \wedge$

$green \in col \iff col_Narr_VAL(green_R) = 1 \wedge$

$blue \in col \iff col_Narr_VAL(blue_R) = 1$

1 mark for each of the relations

1 mark for each of the inverses (if and only if)

1 for use of VAL (or similar)

1 for any other sensible statement

Comment: Many students did not even attempt this question. 3 students obtained full marks (once you recognized what was required, it was almost trivial). I ended up giving 1 or 2 marks just for making an attempt. Some students did point out that *col_R* was missing from the refinement (as were details of *col* from the basic specification), and to be fair, the question should have stated that these were fragments of the complete specifications. However, it was fairly obvious what was meant to anyone who had listen to the refinement lectures.

9. Consider the following data model specification for a simple list machine. In the space provided, complete the definition of the *append* operation, which adds a new element containing the value *new* to the end of the list.

(8 marks)

MACHINE *SimpleList* (*TYPE*)

SETS

POINTERS

CONSTANTS

nil

PROPERTIES

$nil \in POINTERS$

VARIABLES

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

INVARIANT

$list \subseteq POINTERS \wedge$

$next \in POINTERS \rightsquigarrow POINTERS \wedge$

$val \in POINTERS \leftrightarrow TYPE \wedge$

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

INITIALISATION

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

OPERATIONS

$append (new) \hat{=}$

Answer:

PRE $new \in TYPE \wedge list \neq \{\}$ ♥

THEN

ANY pp, qq **WHERE** ♥

$pp \in list \wedge next(pp) = nil \wedge$ ♥

$qq \in POINTERS \wedge qq \notin list$ ♥

THEN

$next(pp) := qq$ ♥

$next(qq) := nil$ ♥

$val(qq) := new$ ♥

$list := list \cup \{qq\}$ ♥

END

END

One mark for each of the ♥ marked lines

END

Comment: This solution assumes that the list is not initially empty. Students supplying a solution that did not make this assumption were awarded a bonus mark. Far too many solutions were couched in terms of *curpos*. There is nothing that requires the value of this variable in the question or the answer.

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; WHILE P DO G VARIANT E INVARIANT Q 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 one of them fails, and the loop is therefore not properly formed.

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

- i. The I rule is obviously true
ii.

(1 marks)

Answer:

$$\begin{aligned} & \text{not } P \ \& \ Q \Rightarrow R \\ & = \text{not } xx \neq 10 \ \& \ xx : \text{NAT} \Rightarrow xx = 10 \end{aligned}$$

true

- iii.

(1 marks)

Answer:

$$Q \Rightarrow E : \text{NAT} = xx : \text{NAT} \Rightarrow 10-xx : \text{NAT}$$

FALSE

- iv.

(1 marks)

Answer:

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

which is true

- v.

(1 marks)

Answer:

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

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