

Name: _____

Student Id: _____

THE UNIVERSITY OF NEW SOUTH WALES

COMP2111

System Modelling and Design

Time allowed: **2 hours**

Total number of questions: **30**

This paper contains both **multiple choice/multiple correct** and **short answer** questions.

The multiple choice questions are **not necessarily** of equal value.

All short answer questions are of *equal value*. The value will be given with the question.

Multiple choice questions must be answered **in pencil** on the **Generalised Answer Sheet**.

Be careful: if you wish to delete a selection, make sure that you completely erase the old selection.

Short answers are to be written in the book provided.

Answers to multiple choice written in this book **will not be read**.

Answers to any questions written on this examination paper **will not be read**. A summary of the Event B mathematical toolkit is attached to this paper. During the examination, it may be detached from the paper if desired.

This paper **may not be retained** by the candidate.

Instructions for this paper

Please read carefully

This examination contains multiple-choice questions in which there may be more than one correct answer to any question.

Marking scheme: If there are N correct answers to all multiple choice questions in the paper then each correct selection to a question will earn you $\frac{1}{N}$ of the total marks for multiple choice questions, and each incorrect selection will result in $\frac{1}{2N}$ of the total marks for multiple choice question being deducted.

For any one question, your cumulative mark for that question will never be less than zero. So, if C and W are the number of correct and incorrect selections, respectively, in a question then the mark for that question will be $\max(C - \frac{W}{2}, 0)/N * 100\%$ of the total marks for the paper.

Examples: Assume a paper with 50 questions and five choices in each question. Assume the total number of correct choices across the whole paper is 100 and suppose the total mark for the paper is 100.

In one question you select

- 1 correctly and make no other choice: your mark will be $(1/100) * 100 = 1$ mark.
- 1 correctly and 1 incorrectly: your mark will be $(1 - 0.5)/100 * 100 = 0.5$ mark.
- 2 correct and 1 incorrect: your mark will be $(1 + 1 - 0.5)/100 * 100 = 1.5$ mark.
- 1 correct and 2 incorrect: your mark will be $(1 - 0.5 - 0.5)/100 * 100 = 0$ mark.
- 1 correct and 3 incorrect: your mark will be $\max(1 - 1, 5, 0)/100 * 100 = 0$ mark.

Note

- 1) *Even though the phrasing of a question may imply more than one answer, there could be only one correct answer.*
- 2) *Answers that are only conditionally correct are not considered correct for the purposes of this examination.*

The number of correct multiple-choice answers in this paper is **57**.

Note carefully: all of the **B** mathematics in the following questions is in marked-up form, unless stated otherwise.

- 1) Choose the best completions of the following statement.
We frequently use nondeterminism in specification because:
 - A) we want to delay design decisions until later in the refinement.
 - B) we frequently need to specify nondeterministic operations.
 - C) we need to provide alternatives for the user of the system.
 - D) computers are nondeterministic.
 - E) the requirements often allow a choice of behaviour, and the specification does not need to decide between those choices.

- 2) Which of the following are *necessarily* correct?
 - A) a member of a *sequence* must have *multiplicity*, or *frequency*.
 - B) a member of a *sequence* must have a *position*.
 - C) a member of a *bag* must have a *position*.
 - D) a member of a *bag* must have *multiplicity*, or *frequency*.
 - E) *sets* are only required to provide *membership*.

- 3) Which of the following statements about *events* are *necessarily* correct?
 - A) If the guard is satisfied then the event will be correct.
 - B) The event will proceed on the assumption that the guard is satisfied.
 - C) A guard ensures that the state invariant is maintained.
 - D) If a guard is not satisfied then the event is skipped.
 - E) Maintenance of the state invariant requires the guard to be satisfied.

- 4) Given a function f that is defined as $f \in X \mapsto Y$, which of the following are correct?
 - A) $\text{dom}(f) \in \mathbb{P}(X)$
 - B) $f \in X \leftrightarrow Y$
 - C) $\text{dom}(f) = X$
 - D) $\text{dom}(f) \subset X$
 - E) $f^{-1} \in Y \rightarrow X$

- 5) Which of the following rules are correct?
 - A) $f \in X \mapsto Y \Rightarrow f \in X \rightarrow Y$
 - B) $f \in X \mapsto Y \Rightarrow f \notin X \mapsto Y$
 - C) $f \in X \mapsto Y \Rightarrow \text{dom}(f) \subset X$
 - D) $f \in X \rightarrow Y \Rightarrow \text{ran}(f) = Y$
 - E) $f \in X \mapsto Y \wedge x \in \text{dom}(f) \Rightarrow f[\{x\}] = \{f(x)\}$

6) Given $n \in \mathbb{N}$ and f^n denotes the iteration of f , which of the following rules are correct?

- A) $f \in X \rightarrow X \Rightarrow f^n \in X \rightarrow X$
- B) $f \in X \leftrightarrow X \Rightarrow f^n \in X \leftrightarrow X$
- C) $f \in X \rightsquigarrow X \Rightarrow f^n \in X \rightsquigarrow X$
- D) $f \in X \Leftrightarrow X \Rightarrow f^n \in X \Leftrightarrow X$
- E) $f \in X \rightsquigarrow X \Rightarrow f^n \in X \rightsquigarrow X$

7) Assume *FRUIT* is a carrier set. Given the specification:

$$\begin{aligned} \text{apples} &\subseteq \text{FRUIT} \\ \text{oranges} &\subseteq \text{FRUIT} \\ \text{salad} &\in \text{FRUIT} \leftrightarrow \text{FRUIT} \\ \text{box} &\in 1.\text{boxsize} \rightarrow \text{FRUIT} \end{aligned}$$

which of the following expressions are *well-typed*?

- A) $\text{oranges} \triangleleft \text{salad}$
- B) $\text{salad} \triangleleft \text{apples}$
- C) $\text{box} ; \text{salad}$
- D) $\text{box} \triangleleft \text{salad}$
- E) $\text{salad} \triangleleft \text{box}$

8) Consider the following enumerated set specifications:

$$\begin{aligned} \text{SETS} \\ \text{MEDAL} &= \{\text{Gold}, \text{Silver}, \text{Bronze}\} \\ \text{COMPETITOR} &= \{\text{Cathy}, \text{Jenny}, \text{Susie}, \text{Inga}, \text{Lisa}\} \end{aligned}$$

We require a variable *medals* that represents the mappings from a competitor (a member of *COMPETITOR*) to a medal (member of *MEDAL*) that has been won by the competitor in some event. Assume that:

- all medals are won and
- no two competitors are awarded the same medal.

Select constraints, not necessarily the strongest, that *medals* must satisfy.

- A) $\text{medals} \in \text{COMPETITOR} \rightarrow \text{MEDAL}$
- B) $\text{medals} \in \text{COMPETITOR} \leftrightarrow \text{MEDAL}$
- C) $\text{medals} \in \text{COMPETITOR} \rightsquigarrow \text{MEDAL}$
- D) $\text{medals} \in \text{COMPETITOR} \rightsquigarrow \text{MEDAL}$
- E) $\text{medals} \in \text{COMPETITOR} \Leftrightarrow \text{MEDAL}$

9) The two assumptions made in Question 8 do not allow for two competitors tying for a place. To be more realistic we could use the assumptions:

- if two competitors tie for a place then both competitors receive the same medal and the next medal (if any) is not awarded;
- at most two people tie for a place;
- otherwise all medals are won.

Select constraints, not necessarily the strongest, that *medals* must satisfy.

- A) $medals \in COMPETITOR \leftrightarrow MEDAL$
- B) $medals \in COMPETITOR \rightarrow MEDAL$
- C) $medals \in COMPETITOR \leftrightarrow\leftrightarrow MEDAL$
- D) $medals \in COMPETITOR \rightarrow\rightarrow MEDAL$
- E) $medals \in COMPETITOR \leftrightarrow\leftrightarrow\leftrightarrow MEDAL$

10) In a sales information system for a bookshop, a variable *sales* is specified by

$$sales \in DAY \rightarrow (BOOK \rightarrow \mathbb{N}_1)$$

where *DAY* is a set of days and *BOOK* is a set of books. The expression, $sales(day)(book)$ gives the number of copies of *book* sold on *day*, if any copies were sold.

Given that $day \in \text{dom}(sales)$ which of the following gives the set of books sold on a particular *day*.

- A) $\text{union}(\text{dom}(sales(day)))$
- B) $\text{dom}(sales(day))$
- C) $\text{dom}(\{\{day\} \triangleleft sales\}(day))$
- D) $\text{dom}(\text{union}(sales[\{day\}]))$
- E) $\text{dom}(\text{union}(sales(day)))$

11) Continuing the example in Question 10 above.

Which of the following expressions specifies the set of all books contained in *sales*.

- A) $\text{dom}(\text{ran}(sales))$
- B) $\sum book \cdot book \in \text{dom}(\text{ran}(sales)) \mid \text{ran}(sales)(book)$
- C) $\sum book \cdot \text{dom}(\text{ran}(sales)) > 0 \mid \text{ran}(sales)(book)$
- D) $\text{union}(\text{dom}(\text{ran}(sales)))$
- E) $\text{dom}(\text{union}(\text{ran}(sales)))$

where $\sum x \cdot P \mid E =$ “sum $E(x)$ over all x satisfying $P(x)$ ”. This is not part of the EventB toolkit, but could be defined as a constant function.

12) Complete the following statement:

Discharging all proof obligations for a machine ensures that ...

- A) *the machine is correct with respect to the requirements.*

- B) *the events of the machine are deterministic.*
- C) *all events maintain the machine invariant.*
- D) *the initialisation establishes the machine invariant.*
- E) *the machine can be implemented.*

13) A video rental store rents videos to members.

The set of members is modelled by the variable *members*, and the set of videos owned by the store is modelled by the variable *videos*. These variables are specified by

$$members \subseteq MEMBER \wedge videos \subseteq VIDEO$$

The policy of the store is that

a video is rented to a single member, but a member can rent any number of different videos.

The relation between rented video and the member who is renting it is to be modelled by the variable *rented*. Which of the following are appropriate specifications?

- A) $rented \in members \leftrightarrow videos$
- B) $rented \in videos \leftrightarrow members$
- C) $rented \in videos \rightsquigarrow members$
- D) $rented \in videos \rightarrow members$
- E) $rented \in members \rightarrow videos$

14) For some video store, not necessarily the same as in 13, *rented* is specified as

$$rented \in videos \rightsquigarrow members$$

For renting a video, we wish to specify an event **Rent** as follows

Event *Rent* $\hat{=}$
any member
 video
when *guards*

where *member* denotes the member who is renting and *video* denotes the video that is being rented.

The action of the event is given as:

$$rented(video) := member .$$

Which of the following are required for a *minimal set* of guards?

- A) $member \notin \text{ran}(rented)$
- B) $member \in members$
- C) $video \in videos$
- D) $rented(video) \neq member$
- E) $video \notin \text{dom}(rented)$

15) Continuing the example in Question 13.

The rental store allows two members to swap their videos, with the store records modified to reflect the new rental situation.

We need an event:

Event $Swap \hat{=}$
any $m1, m2$
 $v1, v2$
when $v1 \in \text{dom } rented \wedge v2 \in \text{dom } rented$
 $m1 = rented(v1) \wedge m2 = rented(v2)$
then $do\text{-}swap$

Which of the following can be used for $do\text{-}swap$?

- A) $rented(v1) := m2 \parallel rented(v2) := m1$
- B) $rented := rented \cup \{v1 \mapsto m2, v2 \mapsto m1\}$
- C) $rented(v1), rented(v2) := m2, m1$
- D) $rented := rented \Leftarrow \{v1 \mapsto m2, v2 \mapsto m1\}$
- E) $rented := rented \Leftarrow \{v1 \mapsto m2\} \Leftarrow \{v2 \mapsto m1\}$

16) Consider modelling a supermarket stock control system.

There is a requirement that:

all products in stock must have a price

A model uses the following:

- The set $PRODUCT$ models all possible products.
- The variable $price$ models the prices of some set of products.
- The variable $stock$ models the number of items in stock for some set of products.

Additionally, the variable $products$ modelling some set of products may be used.

Which of the following adequately satisfy the above requirement: (you may assume that price is adequately modelled by \mathbb{N})

- A) $price \in PRODUCT \mapsto \mathbb{N} \wedge stock \in PRODUCT \mapsto \mathbb{N}_1$
- B) $price \in PRODUCT \mapsto \mathbb{N} \wedge stock \in PRODUCT \mapsto \mathbb{N}_1 \wedge \text{dom}(stock) \subseteq \text{dom}(price)$
- C) $price \in PRODUCT \mapsto \mathbb{N} \wedge stock \in PRODUCT \mapsto \mathbb{N}_1 \wedge \text{dom}(price) \subseteq \text{dom}(stock)$
- D) $products \subseteq PRODUCT \wedge price \in products \mapsto \mathbb{N} \wedge stock \in products \mapsto \mathbb{N}_1$
- E) $products \subseteq PRODUCT \wedge price \in products \mapsto \mathbb{N} \wedge stock \in products \mapsto \mathbb{N}_1$

- 17) A specification for a seating plan has a set of guests, $guests$, a set of tables, $tables$ and positions at each table modelled as a subrange of natural numbers. The seating is modelled by the variable $seating$ specified as follows

$$seating \in guests \rightarrow (tables \times Seats)$$

where $Seats = 1 .. maxSeat$.

Select the most precise English explanation of the above specification.

- A) *The guests are seated around tables.*
 - B) *There is a seat allocated for guests at each table.*
 - C) *Every guest has a seat allocated at each table.*
 - D) *Every guest has a seat allocated at a table.*
 - E) *Guests are allocated to tables around which there are seats.*
- 18) Continuing the example in Question 17, we need to specify that no two guests are seated at the same position on any table.

Which of the following could be used?

- A) $\forall(g1, g2).(g1 \in \text{dom}(seating) \wedge g2 \in \text{dom}(seating) \wedge g1 \neq g2 \Rightarrow seating(g1) \neq seating(g2))$
 - B) $\forall(g1, g2).(seating(g1) \neq seating(g2))$
 - C) $\forall(g1, g2).(g1 \in \text{dom}(seating) \wedge g2 \in \text{dom}(seating) \Rightarrow seating(g1) \neq seating(g2))$
 - D) $\forall(g1, g2).(\text{ran}(seating(g1)) \neq \text{ran}(seating(g2)))$
 - E) $\text{card}(seating) = \text{card}(\text{ran}(seating))$
- 19) Continuing the example in Question 17, we wish to specify an event, $NewGuest$ that adds a new guest, provided there is space left at the current tables, without adding more tables. The event has parameter $guest$, and guard $guest \in GUEST \wedge guest \notin guests$. We also require a guard that ensures that more seats are available. Which of the following could be used?
- A) $\text{card}(seating) < maxSeat * \text{card}(tables)$
 - B) $\text{card}(guests) < maxSeat * \text{card}(tables)$
 - C) $\text{card}(guests) < maxSeat * \text{card}(\text{ran}(seating))$
 - D) $\text{card}(\text{ran}(seating)) < maxSeat$
 - E) $\text{ran}(seating) \neq tables \times Seats$

20) Continuing the *NewGuest* event in Question 19, we wish to allocate a seat at a table to a guest.

We do this adding extra parameters tb and ps , representing a table and a position at that table.

Which of the following could be used for the guard?

- A) $tb \in tables \wedge ps \in Seats \wedge$
 $tb \mapsto ps \in (tables \times Seats)$
- B) $tb \in tables \wedge ps \in Seats \wedge$
 $tb \mapsto ps \notin \text{ran}(seating)$
- C) $tb \in tables \wedge ps \in Seats \wedge$
 $tb \mapsto ps \in (tables \times Seats - \text{ran}(seating))$
- D) $tb \in (tables - \text{dom}(\text{ran}(seating))) \wedge$
 $ps \in (Seats - \text{ran}(\text{ran}(seating)))$
- E) $tb \in tables \wedge tb \notin \text{dom}(\text{ran}(seating)) \wedge$
 $ps \in Seats \wedge ps \notin \text{ran}(\text{ran}(seating))$

21) Which of the following statements about *guards* are *necessarily* correct?

- A) Satisfying the guards ensures that the event will be correct.
- B) If the guard is satisfied, then the event will proceed.
- C) A guard ensures that the state invariant is maintained.
- D) Maintenance of the state invariant requires the guard to be satisfied.
- E) The event will only proceed if the guard is satisfied.

22) Let $[Abort]R = false$ and $[Magic]R = true$ for all R .

Which of the following are correct?

- A) *Abort* is feasible.
- B) *Magic* is feasible.
- C) *Abort* can be refined to anything.
- D) Anything can be refined to *Magic*.
- E) A *feasible* construct can always be refined to a *feasible* construct.

23) The specification for an airline booking system contains a variable *flights* modelled as follows:

$$flights \in DATE \mapsto FLIGHT$$

where *DATE* and *FLIGHT* are carrier sets.

In a refinement, this variable is modelled by two sequences:

$$\begin{aligned} day &\in SEQ(DATE) \\ flight &\in SEQ(FLIGHT) \end{aligned}$$

Which of the following would be a suitable part of a *refinement relation*?

- A) $flights = (day^{-1}; flight)$
- B) $flights = (flight^{-1}; day)$
- C) $flights = day \times flight$
- D) $flights = \{d, f, n \cdot d = day(n) \wedge f = flight(n) \mid d \mapsto f\}$
- E) None of the above.

24) Consider the events in figure 1

Assume $x \in 1..5$

Which of the following are correct?

- A) ii) is refined by i).
- B) i) is refined by ii).
- C) ii) is refined by iii), but the refinement is infeasible.
- D) i) is refined by iii), but the refinement is infeasible.
- E) ii) is refined by iii), and the refinement is feasible.

25) Consider the sequence refinement shown in machines *Seq_ctx*, *Seq*, *List_ctx* and *SeqR* shown in figures 4,5,6,7.

Which of the following describes a property specified by the invariant of *SeqR*.

- A) *last* is never in $\text{dom}(next)$.
- B) if the sequence has more than one item then the pointer to the first item is in $\text{dom}(next)$.
- C) *next* is a bijective function.
- D) *next* is an injective function.
- E) *last* is always in the $\text{ran}(next)$.

i) **Event** *evt1* $\hat{=}$
 when *grd1* : $x \in \{1, 2, 3\}$
 then *act1* : *result* := 0
 end
Event *evt2* $\hat{=}$
 when *grd1* : $x \in \{3, 4, 5\}$
 then *act1* : *result* := 1
 end

ii) **Event** *evt3* $\hat{=}$
 when *grd1* : $x \in \{1, 2\}$
 then *act1* : *result* := 0
 end
Event *evt4* $\hat{=}$
 when *grd1* : $x \in \{3, 4, 5\}$
 then *act1* : *result* := 1
 end

iii) **Event** *evt5* $\hat{=}$
 when *grd1* : $x \in \{1, 2\}$
 then *act1* : *result* := 0
 end
Event *evt6* $\hat{=}$
 when *grd1* : $x \in \{4, 5\}$
 then *act1* : *result* := 1
 end

Figure 1: Simple Events

Answers to the following questions are to be written in the Examination Book. Each question is worth 4 marks.

- 26) This question is concerned with the Coin context machine shown in figure 2. This machine provides some functions for handling bags of coins, here called *COINS*.

We wish to add a function *COINSVAL* which takes a bag of coins and gives the total value of the coins in the bag.

In the book provided give all details of what you would need to specify this function in the context machine.

- 27) This question is concerned with the Coin context and Vending machines shown in figures 2 and 3.

Please answer the following questions in the book provided:

- a) The event *GiveChange* should set the variable *change* to a bag of coins, as specified by *COINS*. The change should be correct, but is not required to be minimal in terms of the number of coins. Also of course the *coinbox* should contain the chosen coins.
- b) Also, describe —it does not have to be formal— any extra guards for the event that would ensure the event is feasible.

- 28) Continuing question 25.

The question is concerned with the *refinement relation* that demonstrates that the list refinement shown in figure 7 simulates the sequence shown in figure 5.

In the answer book:

- a) Write your refinement relation in the provided book.
- b) With each of your predicates give a brief statement of what property you are describing.

Both the formal and informal parts of the refinement relation will be assessed.

- 29) This question refers to the RailTicket assignment on page 17

In the examination book:

- List as many design problems in this solution as you can, giving brief explanations of why they are problems.

- 30) This question refers to the Traffic Light development shown in figures 8, 9, 10 on page 23

We plan to add another event *ToGreenPlus*, which changes the light in direction *adir* to *Green*, as in *ToGreen*, but also changes the light in all other directions that consequently do not conflict with other *Green* lights.

In the answer book:

- a) Write the specification of the new event *ToGreenPlus*.
- b) Sketch a plan for the refinement of the new event using as much as possible of the current refinement.

CONTEXT Coin_ctx

SETS

COIN A set of coin denominations

CONSTANTS

ONE one dollar

TWO two dollar

FIVE five dollar

VALUE mapping from denomination to value

COINS the set of bags of coins

COINSVALUE yields the value of a bag of coins

SUBCOINS SUBBAG(b1 \mapsto b2) = TRUE \Rightarrow b1 is a subbag of b2

AXIOMS

axm1: $COIN = \{ONE, TWO, FIVE\}$

axm2: $ONE \neq TWO$

axm3: $ONE \neq FIVE$

axm4: $TWO \neq FIVE$

axm5: $VALUE \in COIN \rightarrow \mathbb{N}$

axm6: $VALUE(ONE) = 1$

axm7: $VALUE(TWO) = 2$

axm8: $VALUE(FIVE) = 5$

axm9: $COINS = COIN \rightarrow \mathbb{N}_1$

axm10: $COINSVALUE \in COINS \rightarrow \mathbb{N}$

axm11: $\forall cs \cdot cs \in COINS \Rightarrow COINSVALUE(cs) = cs(ONE) + cs(TWO) * 2 + cs(FIVE) * 5$

axm12: $SUBCOINS \in COINS \times COINS \rightarrow BOOL$

axm13: $\forall c1, c2 \cdot c1 \in COINS \wedge c2 \in COINS$

\Rightarrow

$SUBCOINS(c1 \mapsto c2) = bool(dom(c1) \subseteq dom(c2)) \wedge$

$(\forall c \cdot c \in dom(c1) \Rightarrow c1(c) \leq c2(c))$

END

Figure 2: Coin_ctx

MACHINE Vending
SEES Coin_ctx
VARIABLES
 coinbox
 topay
 paid
 change
INVARIANTS
inv1 : coinbox \in COINS
inv2 : topay \in \mathbb{N}
inv3 : paid \in \mathbb{N}
inv4 : change \in COINS
EVENTS
Initialisation
 begin act1 : coinbox : \in COIN \leftrightarrow \mathbb{N}_1
 act2 : topay := 0
 act3 : paid := 0
 act4 : change := \emptyset
 end
Event GiveChange $\hat{=}$
 when grd1 : paid \geq topay
 end
END

Figure 3: Vending machine

CONTEXT Seq_ctx
SETS
 TOKEN
CONSTANTS
 SEQ
AXIOMS
axm1 : SEQ = $\{s \cdot s \in \mathbb{N}_1 \leftrightarrow \text{TOKEN} \wedge \text{finite}(s) \wedge \text{dom}(s) = 1 \dots \text{card}(s)|s\}$
THEOREMS
thm1 : $\forall s \cdot s \in \text{SEQ} \Rightarrow s \in \text{dom}(s) \mapsto \text{TOKEN}$
END

Figure 4: Seq_ctx

MACHINE Seq
SEES Seq_ctx
VARIABLES
seq
size
tokens
INVARIANTS
inv1: $seq \in SEQ$
inv2: $size = card(seq)$
inv3: $tokens = ran(seq)$
EVENTS
Initialisation
begin *act1:* $seq := \emptyset$
act2: $size := 0$
act3: $tokens := \emptyset$
end
END

Figure 5: Sequence machine

CONTEXT List_ctx
EXTENDS Seq_ctx
CONSTANTS
ITER
CLOSE
AXIOMS
axm1: $ITER \in (TOKEN \leftrightarrow TOKEN) \rightarrow (\mathbb{N} \rightarrow (TOKEN \leftrightarrow TOKEN))$
axm2: $\forall r \cdot r \in TOKEN \leftrightarrow TOKEN \Rightarrow ITER(r)(0) = id(TOKEN)$
axm3: $\forall r, n \cdot r \in TOKEN \leftrightarrow TOKEN \wedge n \in \mathbb{N}_1 \Rightarrow ITER(r)(n) = (ITER(r)(n-1); r)$
axm4: $CLOSE \in (TOKEN \leftrightarrow TOKEN) \rightarrow (TOKEN \leftrightarrow TOKEN)$
axm5: $\forall r \cdot r \in TOKEN \leftrightarrow TOKEN \Rightarrow id(TOKEN) \subseteq CLOSE(r)$
axm6: $\forall r \cdot r \in TOKEN \leftrightarrow TOKEN \Rightarrow (CLOSE(r); r) \subseteq CLOSE(r)$
END

Figure 6: List_ctx

```

MACHINE SeqR
REFINES Seq
SEES List_ctx
VARIABLES
    seq
    first
    last
    next
INVARIANTS
    inv1 : size ≠ 0 ⇒ first ∈ tokens
    inv2 : size ≠ 0 ⇒ last ∈ tokens
    inv3 : next ∈ tokens \ {last} ↦ tokens \ {first}
    inv4 : size ≠ 0 ⇒ first = seq(1)
EVENTS
Initialisation
    begin act1 : seq := ∅
        act4 : first :∈ TOKEN
        act5 : last :∈ TOKEN
        act6 : next := ∅
    end
END

```

Figure 7: Sequence refinement machine

CONTEXT RailTicket_ctx

SETS

STATION

CONSTANTS

maxtickets

REALSTATION

PRICE

NoWhere

TownHall

Central

AXIOMS

axm7 : $maxtickets \in \mathbb{N}_1$

axm6 : $REALSTATION = STATION \setminus \{NoWhere\}$

axm1 : $PRICE \subseteq \mathbb{N}$

axm2 : $STATION = \{NoWhere, TownHall, Central\}$

axm3 : $NoWhere \neq TownHall$

axm4 : $NoWhere \neq Central$

axm5 : $TownHall \neq Central$

END

MACHINE TicketMachine

SEES RailTicket_ctx

VARIABLES

stations
ticketprice
tickets
chosen chosen station

nochosen number of tickets required

topay amount left to pay

paid amount paid

moneybox all money paid

INVARIANTS

inv1 : $stations \subseteq REALSTATION$
inv2 : $finite(stations)$
inv3 : $ticketprice \in stations \rightarrow PRICE$
inv4 : $tickets \in stations \rightarrow 0 .. maxtickets$
inv5 : $chosen \in stations \cup \{NoWhere\}$
inv6 : $topay \in \mathbb{Z}$
inv7 : $nochosen \in \mathbb{N}$
inv8 : $paid \in \mathbb{N}$
inv9 : $moneybox \in \mathbb{N}$
inv10 : $paid \leq moneybox$
inv11 : $topay \leq ticketprice(chosen) * nochosen \wedge chosen \in stations$
inv12 : $nochosen \leq tickets(chosen) \wedge chosen \in stations$

EVENTS

Initialisation

begin act1 : $stations := \emptyset$
 act2 : $ticketprice := \emptyset$
 act3 : $tickets := \emptyset$
 act4 : $chosen := NoWhere$
 act5 : $topay := 0$
 act6 : $nochosen := 0$
 act7 : $paid := 0$
 act8 : $moneybox := 0$
end

Event *InitPrice* $\hat{=}$

Set initial price

```
any station
  price
when grd1: station  $\in$  REALSTATION
  grd3: price  $\in$  PRICE
then act1: ticketprice(station) := price
end
```

Event *ChangePrice* $\hat{=}$

Change price

```
any station
  price
when grd1: station  $\in$  REALSTATION
  grd2: price  $\in$  PRICE
  grd3: station  $\in$  stations
then act1: ticketprice(station) := price
end
```

Event *AddTickets* $\hat{=}$

Add more tickets

```
any station
  count
when grd1: station  $\in$  REALSTATION
  grd2: count  $\in$   $\mathbb{N}_1$ 
  grd3: station  $\in$  stations
  grd4: tickets(station) + count  $\leq$  maxtickets
then act1: tickets(station) := tickets(station) + count
end
```

Event *Choose* $\hat{=}$

Choose a ticket and quantity

```
any station
  number
when grd2 : station  $\in$  REALSTATION
  grd3 : number  $\in$   $\mathbb{N}_1$ 
  grd1 : station  $\in$  stations
  grd4 : chosen  $\in$  stations
  grd5 : chosen  $\neq$  NoWhere
  grd6 : nochosen  $\leq$  tickets(chosen)
then act1 : nochosen := nochosen + number
  act2 : chosen := station
end
```

Event *Pay* $\hat{=}$

Make (partial) payment

```
any amount
when grd1 : amount  $\in$   $\mathbb{N}_1$ 
then act1 : topay := topay - amount
  act2 : paid := amount
  act3 : moneybox := moneybox + amount
end
```

Event *Cancel* $\hat{=}$

Cancel the current purchase

```
when grd1 : chosen  $\in$  stations
then act1 : chosen := NoWhere
  act2 : paid := 0
  act3 : topay := 0
  act4 : nochosen := 0
  act5 : moneybox := moneybox - paid
end
```

CONTEXT TrafficLights_ctx

SETS

LIGHTS

DIRECTION

CONSTANTS

Red

Green

Amber

CONFLICT

adir

AXIOMS

axm1 : $LIGHTS = \{Red, Green, Amber\}$

axm2 : $Red \neq Green$

axm3 : $Red \neq Amber$

axm4 : $Green \neq Amber$

axm5 : $finite(DIRECTION)$

DIRECTION is a finite set of directions

axm6 : $CONFLICT \in DIRECTION \leftrightarrow DIRECTION$ CONFLICT relates conflicting directions

axm7 : $CONFLICT \cap id(DIRECTION) = \emptyset$

a direction cannot conflict with itself

axm8 : $CONFLICT^{-1} = CONFLICT$

conflicts are symmetric

axm9 : $adir \in DIRECTION$

END

Figure 8: Traffic Light Context machine

Event GiveTickets $\hat{=}$

when grd1 : $chosen \in stations$

grd3 : $tickets(chosen) - nochosen \leq maxtickets$

grd4 : $tickets(chosen) - nochosen \geq 0$

grd2 : $nochosen \leq tickets(chosen)$

then act1 : $tickets(chosen) := tickets(chosen) - nochosen$

act2 : $nochosen := 0$

act3 : $chosen := NoWhere$

act4 : $topay := 0$

act5 : $paid := 0$

end

END

MACHINE ChangeLight

SEES TrafficLights_ctx

VARIABLES

lights

INVARIANTS

inv1 : $lights \in DIRECTION \rightarrow \{Red, Green\}$

inv2 : $\forall d \cdot d \in DIRECTION \wedge lights(d) = Green \Rightarrow lights[CONFLICT[\{d\}]] \subseteq \{Red\}$
Safety

EVENTS

Initialisation

begin act1 : $lights : |lights' \in DIRECTION \rightarrow \{Red, Green\} \wedge (\forall d \cdot d \in DIRECTION \wedge$
 $lights'(d) = Green \Rightarrow lights'[CONFLICT[\{d\}]] \subseteq \{Red\})$

end

Event ToGreen $\hat{=}$

begin act1 : $lights := lights \Leftarrow \{adir \mapsto Green\} \Leftarrow (CONFLICT[\{adir\}] \times \{Red\})$

end

END

Figure 9: ChangeLight machine

Figure 10: ChangeLight Refinement

```

MACHINE ChangeLightR1
REFINES ChangeLight
SEES TrafficLights_ctx
VARIABLES

    xlights    Extended lights, Red, Green and Amber lights

    delay      delay between Amber and Red or Red and Green

INVARIANTS

    inv1 : xlights ∈ DIRECTION → LIGHTS
    inv2 : ∀d·d ∈ DIRECTION ∧ xlights[{d}] ⊆ {Green, Amber}
           ⇒ xlights[CONFLICT[{d}]] ⊆ {Red}
    inv3 : CONFLICT[{adir}] ≪ lights = CONFLICT[{adir}] ≪ xlights
           Only change lights for adir and CONFLICT[{ adir} ]

    inv4 : xlights(adir) = Green ⇒ lights = xlights
    inv5 : delay ⊆ DIRECTION

EVENTS

Initialisation

    begin with lights' : lights' = xlights'
      act1 : xlights : |xlights' ∈ DIRECTION → {Red, Green}
             ∧ (∀d·d ∈ DIRECTION
                ∧ xlights'(d) = Green ⇒ xlights'[CONFLICT[{d}]] ⊆ {Red})
      act2 : delay := ∅
    end

Event ToGreen ≐
refines ToGreen

    when grd1 : xlights[CONFLICT[{adir}]] ⊆ {Red}
           grd2 : adir ∉ delay
    then act1 : xlights := xlights ≪ {adir ↦ Green}
    end

```

Event *ToAmber* $\hat{=}$

```

any dir
when grd1 : dir  $\in$  DIRECTION
    grd2 : dir  $\in$  CONFLICT[\{adir\}]
    grd3 : xlights(dir) = Green
    grd4 : xlights(adir)  $\neq$  Green
    grd5 : dir  $\notin$  delay
then act1 : xlights(dir) := Amber
    act2 : delay := delay  $\cup$  \{dir\}
end

```

Event *ToRed* $\hat{=}$

```

any dir
when grd1 : dir  $\in$  DIRECTION
    grd2 : dir  $\in$  CONFLICT[\{adir\}]
    grd3 : xlights(dir) = Amber
    grd4 : dir  $\notin$  delay
    grd5 : xlights(adir)  $\neq$  Green
then act1 : xlights(dir) := Red
    act2 : delay := delay  $\cup$  \{adir\}
end

```

Event *Delay* $\hat{=}$

```

any dir
when grd1 : dir  $\in$  delay
then act1 : delay := delay  $\setminus$  \{dir\}
end

```

VARIANT

$$4 * \text{card}(\text{xlights}^{-1}[\{\text{Green}\}]) + 2 * \text{card}(\text{xlights}^{-1}[\{\text{Amber}\}]) + \text{card}(\text{delay})$$

END

1 Answers

1.1 Multiple Choice

- 1 A,E
- 2 B, D, E
- 3 D
- 4 A, B
- 5 A,E
- 6 A, B, C, D, E
- 7 A, C
- 8 B, C, E
- 9 A, B
- 10 B, C, D
- 11 E
- 12 C,D
- 13 B
- 14 A,B,C,E
- 15 D, E
- 16 B,D
- 17 D
- 18 A,E
- 19 A, B, E
- 20 B, C
- 21 E
- 22 A,C,D,E
- 23 A,D
- 24 B,D,E
- 25 A,B,C,D

1.2 Short Answer

- 26) Given COIN is a set of coin denominations, for example

$$COIN = \{ONE, TWO, FIVE\}$$

where each coin denomination has a value given by VALUE

$$VALUE \in COIN \rightarrow NAT1$$

(assuming that no coin has value 0)

then we can have a set of functions COINS representing a bag of COIN —that is a collection of multiple instances of the elements of COIN

$$COINS = COIN \rightarrow \mathbb{N}$$

Now we want to be able to determine the value of a bag of COIN

$$COINSVAL \in COINS \rightarrow \mathbb{N}$$

the following definition of COINSVAL is for a general finite set COIN where we don't know the elements of the set.

$$\forall cs \cdot cs \in COINS \wedge \text{ran}(cs) = 0 \Rightarrow COINSVAL(cs) = 0$$

$$\begin{aligned} \forall cs, c \cdot cs \in COINS \wedge c \in cs \wedge cs(c) \neq 0 \Rightarrow \\ COINSVAL(cs) \\ = cs(c) * VALUE(c) + COINSVAL(cs \Leftarrow \{c \mapsto 0\}) \end{aligned}$$

This is a more general form of COINSVALUE accidentally included in the sample paper.

- 27) • You would need to add a parameter *change*
- and guards
 - $change \in COINS$
 - $COINSVAL(change) = paid - topay$
 - and you need a guard that says that such a bag of coins exists in the *coinbox*. This can be done formally using *SUBCOINS*
- 28) **each element of sequence is in the list**
 $\forall i \cdot i \in 0 \dots size \Rightarrow (ITER(next)(i - 1))(first) = seq(i)$
Explanation: $(ITER(next)(i - 1))$ applies *next* $i - 1$ times, and that is applied to *first*.
all the elements in the list are exactly the elements of tokens
 $CLOSE(next)[first] = tokens$
Explanation: if you follow all the paths from *first* you get exactly all of the elements of *tokens*.
- 29) No answer.
- 30) This questions is too difficult and would never be asked.