TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

**EXAMINATIONS — 2009**

MID-YEAR

# COMP 426

# Formal Software Development

**Time Allowed:** 3 Hours

**Instructions:** Candidates should attempt **all FIVE** questions.

This exam will be marked out of 180.

Foreign language translation dictionaries are allowed.

A summary of Z mathematical notation is provided at the end of this paper.

## Question 1. Formal Specification in Z [30 marks]

Below is an initial specification for a *queue* of customers. The state schema *Queue* records the customers currently in the queue. Initially, the queue is empty. Two operations are provided: *Join* lets a new customer join the queue, and *Leave* removes the next customer waiting in the queue.

$[Customer]$

*Queue*

$content : \mathrm{iseq}\, Customer$

*Join*

$\Delta Queue$

$in? : Customer$

---

$content' = content \frown \langle in? \rangle$

*InitQueue*

$Queue'$

---

$content' = \langle\rangle$

*Leave*

$\Delta Queue$

$out! : Customer$

---

$content = \langle out! \rangle \frown content'$

**(a)** [2 marks] Explain why an *injective* sequence is used in the *Queue* schema.

**(b)** [6 marks] Calculate the preconditions for all operations.

You should show the calculation used to obtain the preconditions.

**(c)** [6 marks] Explain what it means for an operation to be *robust*, and show and explain how robust versions of the operations can be defined using the Z schema calculus.

**(d)** [13 marks] Consider a system simulating a shop with several checkouts. Each checkout has a queue associated to it, and shop customers can join and leave those queues.

$[Checkout]$

*Shop*

$queues : Checkout \to Queue$

*InitShop*

$Shop'$

---

$\mathrm{ran}\, queues' \subseteq \{InitQueue \bullet \theta Queue'\}$

Use *promotion* to define *Join* and *Leave* operations on *Shop* that let customers join and leave the queue from a given checkout. Make sure that a customer cannot be in several shop queues at once.

## Question 2. Data Refinement in Z [60 marks]

**(a)** [5 marks] Explain the relationships between data refinement, forward simulation, and backward simulation in Z.

**(b)** [5 marks] Give the formal requirements for *forward simulation* of Z data types.

**(c)** [20 marks] Prove that forward simulation is transitive.

**(d)** [18 marks] The queue given in **Question 1** is to be implemented using a fixed-sized, cyclic array of customers. Two array indexes are used to indicate where the next customer that joins the queue is stored in the array and where the next customer that is leaving the queue is stored in the array. If one of those indexes reaches the end of the array, it should "wrap-around" to the start of the array.

Write a Z specification for this more concrete queue of customers. Provide operations that are compatible with the *Join* and *Leave* operations given in **Question 1**.

**(e)** [12 marks] Explain whether the cyclic queue you specified in part **(d)** is a data refinement of the queue given in **Question 1**, and whether the queue from **Question 1** is a data refinement of your cyclic queue. In the case(s) you believe to be data refinements, give an abstraction relation, and explain briefly how the abstraction relation is used to prove data refinement.

## Question 3. Object Z [30 marks]

The following Object Z class describes robot cars that can move around within a fixed region. Each car has a position, represented by the pair $(x, y)$, and a direction, represented by $(dx, dy)$.

*Car*

$$left : \mathbb{Z} \times \mathbb{Z} \pfun \mathbb{Z} \times \mathbb{Z}$$

$$left(1,0) = (0,1) \land left(0,1) = (-1,0)$$
$$left(-1,0) = (0,-1) \land left(0,-1) = (1,0)$$

$$x, y : \mathbb{N}$$
$$dx, dy : \{-1, 0, 1\}$$

$$dx * dy = 0 \land dx + dy \neq 0$$

*INIT*

$$dx = 1$$

*Move*

$$\Delta(x, y)$$

$$x' = x + dx$$
$$y' = y + dy$$

*TurnL*

$$\Delta(dx, dy)$$

$$(dx', dy') = left(dx, dy)$$

*TurnR*

$$\Delta(dx, dy)$$

$$(dx, dy) = left(dx', dy')$$

**(a)** [2 marks] What can you say about the initial values of $x$, $y$ and $dy$?

**(b)** [3 marks] Explain the meaning of $\Delta(x, y)$ and $\Delta(dx, dy)$ in *Move* and *TurnL*, respectively. How would you express the same thing in Z?

**(c)** [5 marks] Calculate the precondition of *Move*, and explain how the interpretation of this precondition differs from preconditions in Z.

**(d)** [20 marks] Suppose you wish to describe a system in which several cars can move around within the same fixed region, with the constraint that no two cars can occupy the same location. Explain how you would specify this system in Object Z, using and/or modifying the given class as appropriate and writing any new Object Z required.

## Question 4. Object Z and CSP [30 marks]

Suppose you wish to restrict the behaviour of robot cars, as described in **Question 4**, to eliminate redundant sequences of moves; more specifically, so that:

(i) a car cannot make a sequence of more than two moves in the same direction (i.e. the sequences *TurnL*, *TurnL*, *TurnL* and *TurnR*, *TurnR*, *TurnR* are disallowed), and

(ii) a car cannot turn in one direcetion and then immediately turn in the opposite direction (i.e. the sequences *TurnL*, *TurnR* and *TurnR*, *TurnL* are disallowed).

**(a)** [6 marks] Explain how you would modify the Object Z specification of the *Car* class given in **Question 3** to impose these restrictions.

**(b)** [8 marks] Show how you can describe these restrictions in CSP, and explain how this can be combined with the Object Z specification of *Car*.

**(c)** [8 marks] Show (and explain) how the complete behaviour of a robot car can be specified in CSP, including the behaviour specified in **Question 3** and the constraints stated above.

**(d)** [8 marks] Show (and explain) how your answer to part **(c)** can be extended to describe the complete behaviour of a collection of robot cars (as described in **Question 3 (d)**, along with the constraints stated above) in CSP.

## Question 5. CSP Semantics [30 marks]

**(a)** [6 marks] Explain briefly the difference between trace refinement and failures-divergences refinement in CSP.

Note that the types of the functions *traces*, *failures* and *divergences* are:

$$\begin{aligned} &\mathit{traces} : \mathit{CSP} \to \mathbb{P}(\operatorname{seq} \mathit{Event}) \\ &\mathit{failures} : \mathit{CSP} \to \mathbb{P}(\operatorname{seq} \mathit{Event} \times \mathbb{P}\, \mathit{Event}) \\ &\mathit{divergences} : \mathit{CSP} \to \mathbb{P}(\operatorname{seq} \mathit{Event}) \end{aligned}$$

**(b)** [10 marks] Explain the difference between internal choice and external choice in CSP. Explain why these operators are distinguished by failures-divergences refinement, but not by trace refinement.

**(c)** [14 marks] The set of traces for the sequential composition of processes $P$ and $Q$ is defined as:

$$\begin{aligned} \mathit{traces}[\![P\,;\, Q]\!] = \{ & tr_P : \mathit{traces}[\![P]\!];\ tr_Q : \mathit{traces}[\![Q]\!] \ \mid \\ & (\checkmark \notin tr_P \wedge tr_Q \neq \langle\rangle) \vee tr_P \frown \langle\checkmark\rangle \in \mathit{traces}[\![P]\!] \bullet tr_P \frown tr_Q \} \end{aligned}$$

**(i)** [4 marks] Explain the meaning of the ✓ and its role in this definition.

**(ii)** [10 marks] Use this definition to prove that sequential composition is associative with respect to traces, i.e. that $\mathit{traces}[\![(P\,;\, Q)\,;\, R]\!] = \mathit{traces}[\![P\,;\, (Q\,;\, R)]\!]$.

**********