VICTORIA UNIVERSITY OF WELLINGTON
*Te Whare Wananga o te Upoko o te Ika a Maui*

# EXAMINATIONS — 2001

END OF YEAR

COMP 426

Formal Software Development

Time Allowed: 3 Hours

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

This exam will be marked out of 100.

A summary of Z mathematical notation is provided.

**Question 1.** **[30 marks]**

In a very simple banking system, the information recorded about an individual bank account includes the account holder's name and address, the current balance and the overdraft limit. To specify such a system in Z, we first declare *Name* and *Address* as basic types (as we are not concerned with their structure).

$$[Name, Address]$$

We will assume that amounts of money are given as integers (it doesn't matter whether they are expressed in dollars, cents, hundredths of cents, etc.).

$$Amount == \mathbb{Z}$$

Negative amounts are used to represent negative balances and withdrawals. Overdrafts are expressed as negative numbers, so that an overdraft can be treated simply as a lower bound for the balance.

We can now define a schema to describe an account. The overdraft limit cannot be greater than zero, and the balance cannot be less than the overdraft limit.

```
┌─ Account ──────────────────────────
│ name : Name
│ addr : Address
│ balance : Amount
│ od_limit : Amount
├──────────
│ od_limit ≤ 0
│ od_limit ≤ balance
└────────────────────────────────────
```

When a new account is created, the name, address and initial overdraft limit must be supplied, and the initial balance is zero.

```
┌─ NewAccount ───────────────────────
│ Account
│ name? : Name
│ addr? : Address
│ od_limit? : Amount
├──────────
│ name = name?
│ addr = addr?
│ od_limit = od_limit?
│ balance = 0
└────────────────────────────────────
```

We need an operation to describe how a transaction is processed. In the simple system described here, a transaction is described by the amount (positive for a deposit, negative for a withdrawal), and the effect of a successful transaction is just to add the amount of the transaction to the account balance.

$$
\begin{array}{|l}
\hline
\textit{ProcessTransaction} \\
\Delta Account \\
amount? : Amount \\
\hline
amount? \neq 0 \\
balance' = balance + amount? \\
name' = name \\
addr' = addr \\
od\_limit' = od\_limit \\
\hline
\end{array}
$$

(The banking system will also require one or more operations to update the account holder's name, address and/or overdraft limit, but these are not relevant here.)

**(a)** [5 marks] Calculate the *precondition* for *ProcessTransaction*, and show how a robust version of this operation can be defined using the Z schema calculus.

You should explain the calculation used to obtain the precondition and how the robust specification is constructed.

**(b)** [20 marks] A more realistic version of a banking system would keep a history of all transactions that have been applied to an account, and provide an operation to produce a "statement" listing the transactions that have occurred within a specified time period. For simplicity, we will assume that a statement always covers the entire life-time of the account.

Explain how you would modify the given specification to incorporate these features. You should assume that each transaction is described by the date and the amount, and that the statement will show the date and amount of each transaction and the resulting account balance.

You should explain the purpose of your modifications, and state any additional assumptions you make. Give any new definitions that are required, and describe any modifications required to the given definitions — you do not need to write out new versions of the given definitions in full. You may assume the existence of a function $sum : \mathrm{seq}\,\mathbb{Z} \longrightarrow \mathbb{Z}$, which sums a sequence of numbers.

**(c)** [5 marks] Explain what is meant by *promotion* in Z, and outline how this technique would be used to specify a banking system with many accounts.

You do not need to give a complete specification for such a banking system; just enough to illustrate the use of promotion.

**Question 2.** **[30 marks]**

A class register system is used to record the students who are enrolled in a course and which of them have completed the homework.

$$[STUDENT]$$

$$\begin{array}{|l}\hline \textit{Register} \\ enrolled : \mathbb{P}\, STUDENT \\ completed : \mathbb{P}\, STUDENT \\ \hline completed \subseteq enrolled \\ \hline \end{array}$$

Initially, no students are enrolled.

$$\begin{array}{|l}\hline \textit{InitRegister} \\ Register \\ \hline enrolled = \varnothing \\ \hline \end{array}$$

We have an operation to enrol a new student, who must not already be enrolled and has not yet done the homework.

$$\begin{array}{|l}\hline \textit{Enrol} \\ \Delta Register \\ s? : STUDENT \\ \hline s? \notin enrolled \\ \\ enrolled' = enrolled \cup \{s?\} \\ completed' = completed \\ \hline \end{array}$$

We can record that a student has completed the homework, provided that the student is enrolled and is not already recorded as having completed the homework.

$$\begin{array}{|l}\hline \textit{Record} \\ \Delta Register \\ s? : STUDENT \\ \hline s? \in enrolled \setminus completed \\ \\ enrolled' = enrolled \\ completed' = completed \cup \{s?\} \\ \hline \end{array}$$

To implement this system, we need to choose a suitable data representation for the information recorded by the system and implement the operations in terms of this representation.

A straightforward representation for the class register system would be to store all of the names of students enrolled in the course as a sequence, with the students who have completed the homework listed first, and keep a count of the number of students who have completed the homework. Thus, the concrete state would be defined as:

$$\begin{array}{|l}\hline \textit{Register1} \\ names : \mathrm{iseq}\, STUDENT \\ done : \mathbb{N} \\ \hline done \leq \#names \\ \hline \end{array}$$

**(a)** [5 marks] Explain what Spivey means by the terms *direct refinement* and *deferred refinement*. Give a brief justification for the claim that any implementation that can be obtained by direct refinement can also be obtained by deferred refinement followed by operational refinement.

**(b)** [4 marks] Give an *abstraction relation* specifying the relationship between the abstract and concrete states for the class register system (i.e. between *Register* and *Register*1), as described above.

**(c)** [10 marks] Use your abstraction relation to *calculate* specifications for "concrete" versions of *InitRegister*, *Enrol* and *Record* operating on the concrete state (*Register*1).

**(d)** [6 marks] Show how the specifications you derived in part **(c)** can be refined to obtain specifications that can be easily translated into code.

**(e)** [5 marks] Discuss briefly *ONE* of the following statements:

**A:** There are cases of valid data refinement that cannot be verified using the techniques described in Spivey's book "The Z Notation: A Reference manual".

**B:** When developing systems using Z, it is necessary to eliminate all Z schema calculus operators before performing either data refinement or operation refinement.

**Question 3.** **[20 marks]**

**(a)** [5 marks] Give weakest precondition semantics for the following statements (as they are defined in Dijkstra's programming calculus and the refinement calculus):

- **(i)** **skip**
- **(ii)** **abort**
- **(iii)** $x := e$
- **(iv)** $S \; ; \; T$
- **(v)** **if** $G \rightarrow S \; [] \; H \rightarrow T$ **fi**

**(b)** [5 marks] Define the following properties of a statement $S$, in terms of its weakest precondition semantics:

- **(i)** monotonic (with respect to implication)
- **(ii)** feasible (or strict)
- **(iii)** terminating
- **(iv)** conjunctive
- **(v)** disjunctive

**(c)** [2 marks] Which of the above properties does Dijkstra require as *healthiness conditions*?

**(d)** [4 marks] Which of the above properties are **not** required in the refinement calculus? In each such case, give an example of a construct in the refinement calculus which does not have that property.

**(e)** [4 marks] Show that *sequential composition* is monotonic, with respect to refinement.

**Question 4.** **[20 marks]**

The following is an incomplete derivation of a program in the refinement calculus, where $count(x)$ is an abbreviation for $\#\{\, i : 1 \mathinner{..} x-1 \mid A[i] < 0 \wedge 0 < A[i+1] \,\}$, $\#s$ is the size of set $s$, and $A$ is an array with index range $1 \mathinner{..} n$.

$$c\colon\left[\, n > 0 \;\middle/\; c = count(n) \,\right] \qquad (1)$$

(1) $\sqsubseteq$ $\langle$ *introduce variable* $\rangle$

**var** $k$ $\bullet$

$$k, c\colon\left[\, n > 0 \;\middle/\; c = count(n) \,\right] \qquad (2)$$

(2) $\sqsubseteq$ $\langle$ *split specification* $\rangle$

$$k, c\colon\left[\, n > 0 \;\middle/\; 1 \le k \le n \wedge c = count(k) \,\right]; \qquad (3)$$

$$k, c\colon\left[\, 1 \le k \le n \wedge c = count(k) \;\middle/\; c = count(n) \,\right] \qquad (4)$$

(3) $\sqsubseteq$ $\langle$ *introduce assignment* $\rangle$

$k, c := 1, 0$

(4) $\sqsubseteq$ $\langle$ *introduce* **do** $\rangle$

**do** $k \neq n \rightarrow$

$$k, c\colon\left[\, 1 \le k < n \wedge c = count(k) \;\middle/\; \begin{array}{c} 1 \le k \le n \wedge c = count(k) \\ 0 \le n-k < n-k_0 \end{array} \,\right] \qquad (5)$$

**od**

(5) $\sqsubseteq$ $\langle$ *following assignment, contract frame* $\rangle$

$$c\colon\left[\, 1 \le k < n \wedge c = count(k) \;\middle/\; c = count(k+1) \,\right]; \qquad (6)$$

$k := k+1$

(6) $\sqsubseteq$ $\langle$ *introduce* **if** $\rangle$

**if** $A[k] < 0 \rightarrow$

$$c\colon\left[\, 1 \le k < n \wedge c = count(k) \wedge A[k] < 0 \;\middle/\; c = count(k+1) \,\right] \qquad (7)$$

$[\!]$ $A[k] \ge 0 \rightarrow$

$$c\colon\left[\, 1 \le k < n \wedge c = count(k) \wedge A[k] \ge 0 \;\middle/\; c = count(k+1) \,\right] \qquad (8)$$

**fi**

(8) $\sqsubseteq$ $\langle$ *introduce* **skip** $\rangle$

**skip**

**(a)** [4 marks] Write down the program resulting from the above derivation.

**(b)** [5 marks] For each step in the above derivation, give the *proof obligation(s)* that must be satisfied in order for the refinement step to be valid.

**(c)** [3 marks] What is the *loop invariant* used in the above derivation? Explain how it can be obtained from the initial specification (1).

**(d)** [2 marks] What is the *variant* used in the above derivation? Explain how it is used to ensure that the loop terminates.

**(e)** [6 marks]

Complete the derivation, by refining the specification statement labelled (7) to code.

For each step in this derivation, indicate the law used, write down the necessary proof obligations and briefly justify why they are valid.

*******************************