VICTORIA UNIVERSITY OF WELLINGTON

Te Whare Wananga o te Upoko o te Ika a Maui

EXAMINATIONS — 2002

END OF YEAR

COMP 426

Formal Software Development

Time Allowed: 3 Hours

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

This exam will be marked out of 100.

Foreign language translation dictionaries are allowed

A summary of Z mathematical notation is provided.

Question 1. **[26 marks]**

Consider the following system description:

> A manufacturing company has a number of workers who can make the products the company sells. When an order for some quantity of these products is received, the job of making the required products is allocated to one of the workers, if any worker is not currently busy, and is otherwise placed on a queue of waiting jobs. When a worker completes his or her current job, the worker is allocated the first job on the queue, if there is one, and otherwise becomes idle (not busy).

Write a formal specification in Z, describing the above system. Your specification should describe the behaviour of the system when an order is received and when a job is completed, as well as the initial state of the system, and should include an operation to report the number of workers currently idle and the number of jobs currently in the queue. You should explain any types, schemas and other definitions used. You may ignore the details of just what constitutes an order (or a job), and assume that the set of workers is fixed.

You should first define the normal case for each operation, and then extended it to obtain a "robust" specification, explaining how the robust version is obtained. Is the robust version a refinement of the original?

Prove that no worker will be idle if the job queue is non-empty.

Question 2. **[30 marks]**

The following schema definitions are taken from Spivey's specification of the birthday book system:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{BirthdayBook}} \\
\hline
known : \mathbb{P}\, NAME \\
birthday : NAME \rightarrowtail\!\!\!\!\!+ DATE \\
\hline
known = \mathrm{dom}\, birthday \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitBirthdayBook}} \\
\hline
BirthdayBook \\
\hline
known = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AddBirthday}} \\
\hline
\Delta BirthdayBook \\
name? : NAME \\
date? : DATE \\
\hline
name? \notin known \\
birthday' = birthday \cup \\
\qquad\qquad \{name? \mapsto date?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FindBirthday}} \\
\hline
\Xi BirthdayBook \\
name? : NAME \\
date! : DATE \\
\hline
name? \in known \\
date! = birthday(name?) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Remind}} \\
\hline
\Xi BirthdayBook \\
today? : DATE \\
cards! : \mathbb{P}\, NAME \\
\hline
cards! = \\
\quad \{\, n : known \mid birthday(n) = today? \,\} \\
\hline
\end{array}
$$

In Spivey's data refinement, names and dates are stored in infinite arrays, which are modelled as total functions on the positive integers. An alternative model of this implementation is to treat names and dates as sequences, giving the concrete state:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{BirthdayBook1}} \\
\hline
names : \mathrm{iseq}\, NAME \\
dates : \mathrm{seq}\, DATE \\
\hline
\# names = \# dates \\
\hline
\end{array}
$$

(a) [4 marks] Give an abstraction relation, showing how *BirthdayBook* and *BirthdayBook*1 are related.

(b) [4 marks] Explain how an abstraction relation can be used to calculate a minimally refined data refinement of a specification.

(c) [14 marks] Use your abstraction relation from part **(a)** to calculate specifications for "concrete" versions of the birthday book operations acting on the concrete state (*BirthdayBook*1).

(d) [8 marks] Show how the specifications you derived in part **(c)** can be refined to obtain specifications that can be easily translated into code.

Question 3. **[12 marks]**

(a) [6 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)** disjunctive
- **(v)** conjunctive
- **(vi)** continuous

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

(c) [4 marks] Which of Dijkstra's healthiness conditions are **not** required in the refinement calculus? In each such case, explain why that property is not appropriate in a wide-spectrum language and give an example of a construct in the refinement calculus which does not have that property.

Question 4. **[12 marks]**

A complete, Boolean lattice, $(X, \leq)$, has:

- A unique *bottom* element, $\perp$, such that $\perp \leq x$, for all $x \in X$.
- A unique *top* element, $\top$, such that $x \leq \top$, for all $x \in X$.
- A *meet* (or greatest lower bound) operation, $\sqcap$, such that, for all $x, y, z \in X$, $x \sqcap y \leq x$, $x \sqcap y \leq y$, and $x \leq z \wedge y \leq z \Rightarrow z \leq x \sqcap y$.
- A *join* (or least upper bound) operation, $\sqcup$, such that, for all $x, y, z \in X$, $x \leq x \sqcup y$, $x \leq y \sqcup y$, and $z \leq x \wedge z \leq y \Rightarrow x \sqcup y \leq z$.

For example, in the lattice of truth values $(\{true, false\}, \Rightarrow)$, we have $\perp = false$, $\top = true$, $\sqcap = \vee$ and $\sqcup = \wedge$; and in the lattice $(U, \subseteq)$ of subsets of a set U, we have $\perp = \varnothing$, $\top = U$, $\sqcap = \cup$ and $\sqcup = \cap$.

(a) [6 marks] Consider the set, P, of programs in the language of the refinement calculus, and the refinement ordering $\sqsubseteq$.

What members of P are the bottom and top of the refinement lattice $(P, \sqsubseteq)$?

What program constructors are the meet and join operations in the lattice $(P, \sqsubseteq)$?

(b) [6 marks] Consider the set, S_Σ, of Z specifications defining operations on a state space Σ, and the operation refinement ordering $\sqsubseteq_Z$.

Does $(S_\Sigma, \sqsubseteq_Z)$ have a bottom element? If so, what is it? If not, briefly explain why.

Does $(S_\Sigma, \sqsubseteq_Z)$ have a top element? If so, what is it? If not, briefly explain why.

Does $(S_\Sigma, \sqsubseteq_Z)$ have a meet operation? If so, what is it? If not, briefly explain why.

Does $(S_\Sigma, \sqsubseteq_Z)$ have a join operation? If so, what is it? If not, briefly explain why.

Question 5. **[20 marks]**

(a) Consider the specification statement:

$$x, y\!:\!\left[\, x \neq y \;\middle/\; \{x, y\} = \{x_0, y_0\} \wedge x < y \,\right]$$

(i) [3 marks] Explain what this specification statement means, identifying each of the components of the specification statement and explaining how it contributes to the meaning of the whole statement.

(ii) [6 marks] Explain the rôle of the zero subscripts in the above specification statement, and how their use can be formalised in terms of logical constants. Write an equivalent program without zero subscripted variables.

(iii) [3 marks] Show how the specification statement above can be refined to code. State the refinement rule used at each step, and the corresponding proof obligations.

(b) Let $A[1..n]$ describe an array segment with indices ranging from 1 to n. In deriving a program to find the value in $A[1\,..\,n]$ furthest from zero, we might arrive at the following incomplete program (where $\mid x \mid$ denotes the absolute value of x):

$$\begin{array}{l} [\![\ \textbf{var}\ k \bullet \\ \quad k, x := 1, A[1]\ ; \\ \quad k, x\!:\!\left[\, n \geq 1 \wedge k = 1 \wedge x = A[1] \;\middle/\; \begin{array}{c} x \in A[1\,..\,n] \\ (\forall\, y \in A[1\,..\,n] \bullet \mid x \mid \geq \mid y \mid) \end{array} \right] \\ ]\!] \end{array} \qquad (1)$$

The next step in the derivation is to introduce a loop, using the loop introduction law:

$$w\!:\!\left[\, P \;/\; R \,\right] \sqsubseteq \textbf{do}\ [\!]_i\, G_i \rightarrow w\!:\!\left[\, I \wedge G_i \;/\; I \wedge 0 \leq t < t_0 \,\right]\ \textbf{od},$$
if $P \Rightarrow I$ and $I \wedge \neg(\bigvee_i G_i) \Rightarrow R$

(i) [2 marks] Explain the rôles I (the loop invariant) and t (the variant or bound function) in this rule.

(ii) [4 marks] Give suitable choices for I and t for refining the specification (1) above, show the result of applying the loop introduction law with these values, and prove that the resulting proof obligations hold.

(iii) [2 marks] Describe the next step you would take in completing this derivation, and explain how you chose it.

****