TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA

UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2010

MID-YEAR

SWEN 421

Formal Software Engineering 1

Time Allowed: 3 Hours

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

This exam will be marked out of 180.

Paper non-English to English dictionaries are allowed.

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

Question 1. Formal Specification in Z [30 marks]

Since your lecturer has trouble remembering names, you decide to provide her with a system that she can use to look up names for all sorts of things. Below is a first specification. The specification needs to talk about things and names. The state schema *Memory* records the names of named things. The *GetName* operation returns the name of things for which a name is known.

$$[Thing, Name]$$

$$\begin{array}{|l}
\text{\_\_}\mathit{Memory}\text{\_\_\_\_\_\_\_\_\_\_\_\_} \\
named : \mathbb{P}\, Thing \\
name : Thing \pfun Name \\
\hline
named = \operatorname{dom} name \\
\hline
\end{array}$$

$$\begin{array}{|l}
\text{\_\_}\mathit{GetName}\text{\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Xi Memory \\
thing? : Thing \\
name! : Name \\
\hline
thing? \in named \\
name! = name(thing?) \\
\hline
\end{array}$$

(a) [4 marks] Write a Z schema describing the initial states of your system and prove the initialisation theorem (that an empty state exists).

(b) [4 marks] Provide a normalised version of schema *GetName*.

Note: A normalised schema has only declarations of the form *name* : *expr* in its declaration part where *expr* corresponds to a base type.

(c) [4 marks] Provide an operation *Label* that accepts a thing and a name as input and updates *name* so that the given thing gets recorded as having the given name.

(d) [3 marks] You let a fellow student implement your specification and your lecturer starts using the system. But when she uses the *GetName* operation with a thing that does not have a name in the system yet, the *GetName* operation not only returns "Fool" but also replaces the names of things for which a name is known with "Fool", making the system useless. You blame your programmer but he insists that this is a valid implementation of your specification. Is he right? Explain your answer!

(e) [5 marks] Provide a robust version for operation *GetName*.
Make use of existing schemas and the schema calculus.

(f) [10 marks] Extend the system (state space, initialisation, and operations) so that it is possible to annotate things with possibly several tags. Make sure that the same tag can be used to annotate different things. Provide an operation *AddTag* that accepts a set of things and a tag as input and adds the given tag to the annotations of the given things, leaving everything else unchanged.

Question 2. Operation Refinement in Z [30 marks]

Let A and B be operations with state S, input $x? : X$ and output $y! : Y$.

Operation A is refined by operation B, written $A \sqsubseteq B$, if B produces a valid result whenever A does, and any behaviour exhibited by B is permitted by A. This is formalised as follows:

(i) $\forall S;\ x? : X \bullet \mathrm{pre}\, A \Rightarrow \mathrm{pre}\, B$ (Applicability)

(ii) $\forall \Delta S;\ x? : X;\ y! : Y \bullet \mathrm{pre}\, A \land B \Rightarrow A$ (Correctness)

Operation refinement can be understood as having two possible effects: weakening preconditions and strengthening postconditions (reducing non-determinism). In the following we consider restricted versions of refinements that allow only one of these to occur:

B is called a *pre-refinement* of A, written $A \sqsubseteq_{pre} B$, iff

$$\forall \Delta S;\ x? : X;\ y! : Y \bullet A \Leftrightarrow (\mathrm{pre}\, A \land B)$$

B is called a *post-refinement* of A, written $A \sqsubseteq_{post} B$, iff

$$\forall S;\ x? : X \bullet \mathrm{pre}\, A \Leftrightarrow \mathrm{pre}\, B$$

and

$$\forall \Delta S;\ x? : X;\ y! : Y \bullet B \Rightarrow A$$

(a) [4 marks] Prove that $\forall \Delta S;\ x? : X;\ y! : Y \bullet \mathrm{pre}\, A \land A \Leftrightarrow A$ and
$\forall S;\ x? : X \bullet \mathrm{pre}(\mathrm{pre}\, A \land B) \Leftrightarrow (\mathrm{pre}\, A \land \mathrm{pre}\, B)$.
These might help you in your later proofs.

(b) [5 marks] Prove that $A \sqsubseteq_{post} B$ implies $A \sqsubseteq B$.

(c) [5 marks] Prove that $A \sqsubseteq_{pre} B$ implies $A \sqsubseteq B$.

(d) [4 marks] Prove that an operation refinement can be split into a post-refinement followed by a pre-refinement, that is if $A \sqsubseteq B$ then $A \sqsubseteq_{post} (\mathrm{pre}\, A \land B)$ and $(\mathrm{pre}\, A \land B) \sqsubseteq_{pre} B$.

(e) [12 marks] Consider the following operations:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Search}} \\
\hline
s? : \mathrm{seq}\,\mathbb{N} \\
n? : \mathbb{N} \\
i! : \mathbb{N} \\
\hline
1 \leq i! \leq \# s? \\
s?(i!) = n? \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{SearchB}} \\
\hline
\textit{Search} \\
\hline
\forall\, i : 1 \,..\, (\# s? - 1) \bullet s?(i) \leq s?(i+1) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{SearchA}} \\
\hline
\textit{Search} \\
\hline
\forall\, i : 1 \,..\, (i! - 1) \bullet s?(i) \neq n? \\
\hline
\end{array}$$

Prove or disprove that

1) $\textit{SearchB} \sqsubseteq \textit{Search}$

2) $\textit{SearchB} \sqsubseteq_{pre} \textit{Search}$

3) $\textit{SearchB} \sqsubseteq_{post} \textit{Search}$

4) $\textit{Search} \sqsubseteq \textit{SearchB}$

5) $\textit{SearchA} \sqsubseteq_{post} \textit{Search}$

6) $\textit{Search} \sqsubseteq_{post} \textit{SearchA}$

Note: You might use the results you were asked to prove in the previous questions even if you have not completed the proofs.

Question 3. Alloy [30 marks]

The following Alloy specification provides a static model of the state space of a file system.

```
abstract sig FSObject {}
sig File extends FSObject {}
sig Dir extends FSObject { children: set (Dir+File) }
one sig Root in Dir {}

fact {
  // All file system objects apart from Root can be reached
  // by starting from Root and following the children relation
  // (^children is the transitive closure of children)
  FSObject-Root = Root.^children

  // each file system object has exactly one parent
  all o: FSObject | one children.o
}

check { some FSObject }
check { no FSObject }
```

(a) [6 marks] When running "Execute All", the Alloy Analyzer replies:

```
2 commands were executed. The results are:
   #1: No counterexample found. check$1 may be valid.
   #2: No counterexample found. check$2 may be valid.
```

The two check commands seem to contradict each other. Why does the Alloy Analyzer state that both may be valid? Explain the problem, provide a solution, and explain how similar problems can be avoided in general.

(b) [12 marks] Provide a dynamic model of a file system that includes operations for adding a new file system object and removing an existing file system object.

(c) [8 marks] Write a check command to explore whether adding a file followed by adding a directory yields the same state as adding the directory first and then the file. Select an appropriate scope and explain your choice.

(d) [4 marks] The "exactly" keyword lets you specify that a signature has some exact number of elements. Why should you use it with great care in check commands?

Question 4. Comparing Alloy and Z [30 marks]

(a) [10 marks]

Suppose you want to use the Alloy Analyzer to check that sets are closed under union. You might write:

```
sig Element {}
sig Set { elements: set Element }

check Closed {
  all s0, s1: Set | some s2: Set |
    s2.elements = s0.elements + s1.elements
}
```

What happens if you run the check command `Closed` and why?

Translate the above Alloy to Z and prove or disprove the corresponding assertion in Z.

(b) [20 marks] Compare and contrast Z and Alloy. Your discussion should include specific examples of specifications that can be easily expressed in one notation but not in the other.

Question 5. Program Verification, Semantics, and Refinement [30 marks]

(a) [10 marks] Consider the following JML annotated Java methods.

Note: The JML diverges clause allows partial correctness specifications.

```
//@ requires false;             //@ requires true;
//@ ensures true;               //@ ensures false;
int m1(int a) { ... }           int m2(int a) { ... }

//@ requires true;              //@ requires a > 0;
//@ diverges true;              //@ ensures \result * \result == a
//@ ensures false;              int m4(int a) { ... }
int m3(int a) { ... }

//@ requires a > 0;
//@ ensures \result == 0 || \result == 1
int m5(int a) { ... }
```

If possible, provide implementations of each Java method so that the implementation satisfies the given JML specification. Give a brief explanation why the given implementation satisfies the specification or why the specification cannot be implemented.

(b) [10 marks] We say that two programs S and T are:

- *operationally equivalent*, if S and T produce the same computation (i.e. perform the same sequence of steps and pass through the same sequence of states) when executed starting in any given initial state.
- *Hoare equivalent*, if for any precondition P and postcondition R, either $\{P\}S\{R\}$ and $\{P\}T\{R\}$ *both* hold or *neither* of them holds.

(i) Briefly explain what the "states" (a program passes through) are in operational semantics.

(ii) Briefly explain what $\{P\}S\{R\}$ means.

(iii) EITHER: Show that if S and T are operationally equivalent, then they are also Hoare equivalent.
OR: Give a counter-example to show that this is not the case.

(iv) EITHER: Show that if S and T are Hoare equivalent, then they are also operationally equivalent.
OR: Give a counter-example to show that this is not the case.

(c) [10 marks] Consider the specification statement:

$$m : [true,\ (m' = x \vee m' = y) \wedge m' \leq x \wedge m' \leq y]$$

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

(ii) Refine this specification statement to code. Name the refinement rules used at each step, and state and prove all proof obligations.
The assignment and conditional introduction rules are given below.

Assignment introduction: If E is an expression with the same type as w such that $pre \Rightarrow post[E/w',\ x/x']$ then $w, x : [pre,\ post] \sqsubseteq w := E$.

Conditional introduction: If $pre \Rightarrow (G_1 \vee ... \vee G_n)$ then

$$w : [pre,\ post] \sqsubseteq \left\{ \begin{array}{l} \text{if } G_1 \rightarrow w : [G_1 \wedge pre,\ post] \\ [\,]\ G_2 \rightarrow w : [G_2 \wedge pre,\ post] \\ \vdots \\ [\,]\ G_n \rightarrow w : [G_n \wedge pre,\ post] \\ \text{fi} \end{array} \right.$$

Question 6. Other Formal Methods [30 marks]

(a) [10 marks] Explain how software models can be used in testing.

(b) [10 marks] Provide an example of an Object-Z class, and name and explain its main components. Also explain what objects are and what the difference is between the state of an object and its identity.

(c) [10 marks] Discuss the validity of the following statement: "Formal methods delay the development process."

You might want to consider the following:

- the place of formal methods in the software lifecycle
- weaknesses and strength of formal methods

****