Victoria University of Wellington

DEGREE EXAMINATIONS — 1993

(END OF YEAR)

COMP 426

Computational Logic 2

Time Allowed: THREE hours.

Instructions: Candidates should attempt all questions.
This exam will be marked out of 100.

Question 1. **[35 marks]**

This question concerns an extension of Wing's Library Problem (as described by Diller) in which a borrower is able to place a "hold" on a book if every copy of the book is currently out (or on hold), so that a copy will be held for him/her when it is returned. You are given part of a Z specification for this version of the problem. The English text concentrates on the parts that are different from Diller's version of the specification.

The schema *ParaLibrary* contains information about books and their authors and subjects; it is unchanged.

$$
\begin{array}{|l}
\textit{ParaLibrary} \\
\hline
\textit{instanceof} : \textit{Copy} \twoheadrightarrow \textit{Book} \\
\textit{writtenby} : \textit{Book} \twoheadrightarrow \mathbb{F}\,\textit{Author} \\
\textit{about} : \textit{Book} \twoheadrightarrow \mathbb{F}\,\textit{Subject} \\
\hline
\operatorname{dom} \textit{writtenby} \subseteq \operatorname{ran} \textit{instanceof} \\
\operatorname{dom} \textit{about} \subseteq \operatorname{ran} \textit{instanceof} \\
\hline
\end{array}
$$

The schema *LibraryDB* contains information about staff and borrowers, and borrowing records. There are two new state components: a set *onhold*, which contains the copies that are currently "on hold" (i.e. waiting to be issued to someone who has placed a hold on them), and a partial function *heldfor*, which indicates the borrower for whom each copy in *onhold* is being held.

$$
\begin{array}{|l}
\hline
\textit{LibraryDB} \\
\textit{borrower}, \textit{staff} : \mathbb{F}\, \textit{Person} \\
\textit{available}, \textit{checkedout}, \textit{onhold} : \mathbb{F}\, \textit{Copy} \\
\textit{previouslyborrowedby}, \textit{borrowedby}, \textit{heldfor} : \textit{Copy} \nrightarrow \textit{Person} \\
\hline
\textit{borrower} \cap \textit{staff} = \varnothing \\
\mathsf{disjoint}\ \langle \textit{available}, \textit{checkedout}, \textit{onhold} \rangle \\
\operatorname{dom} \textit{borrowedby} = \textit{checkedout} \\
\operatorname{ran} \textit{borrowedby} \subseteq \textit{borrower} \\
\operatorname{dom} \textit{previouslyborrowedby} \subseteq \textit{available} \cup \textit{checkedout} \cup \textit{onhold} \\
\operatorname{ran} \textit{previouslyborrowedby} \subseteq \textit{borrower} \\
\operatorname{dom} \textit{heldfor} = \textit{onhold} \\
\operatorname{ran} \textit{heldfor} \subseteq \textit{borrower} \\
(\forall\, p : \textit{borrower} \bullet \#(\textit{borrowedby}^{\sim}(\!|\{p\}|\!)) \leq \textit{MaxCopiesAllowed}) \\
\hline
\end{array}
$$

The additional/modified constraints here are:

- The sets *available*, *checkedout* and *onhold* are disjoint (i.e. a copy cannot be both available and checked out, available and on hold, or checked out and on hold).
- The domain of *previouslyborrowedby* has been extended to include *onhold*.
- The domain of *heldfor* is *onhold* and its range is a subset of *borrower* (i.e. books can only be held for registered borrowers).

Information about holds that have been placed on books is contained in the schema *HoldsInfo*. This information is modelled using a total function, *holds*, which records all current holds on each book. Thus, $holds(b)$ is the sequence of all borrowers with holds on book b, in the order the holds were placed.

$$
\begin{array}{|l}
\hline
\textit{HoldsInfo} \\
\textit{ParaLibrary} \\
\textit{LibraryDB} \\
\textit{holds} : \textit{Book} \longrightarrow \operatorname{seq} \textit{Person} \\
\hline
(\forall\, b : \operatorname{dom} \textit{holds} \bullet \operatorname{ran}(\textit{holds}(b)) \subseteq \textit{borrower}) \\
(\forall\, b : \operatorname{dom} \textit{holds} \bullet \textit{instanceof}^{\sim}(\!|\{b\}|\!) \subseteq \textit{checkedout} \cup \textit{onhold}) \\
(\forall\, b : \operatorname{dom} \textit{holds} \bullet \\
\qquad (\forall\, c : \textit{checkedout} \mid \textit{instanceof}(c) = b \bullet \textit{borrowedby}(c) \notin \operatorname{ran}(\textit{holds}(b)))) \\
(\forall\, b : \operatorname{dom} \textit{holds} \bullet \\
\qquad (\forall\, i, j : \operatorname{dom}(\textit{holds}(b)) \bullet \textit{holds}(b)(i) = \textit{holds}(b)(j) \Rightarrow i = j)) \\
\hline
\end{array}
$$

The following constraints apply to holds:

- Only an authorised borrowed can place a hold on a book. (Note that a sequence is a finite function whose domain is $1..n$ for some natural number n; thus for a sequence s, $\operatorname{ran} s$ is the set of elements in s.)

- A hold can only be placed on a book if all copies of that book are checked out or on hold.
- A user cannot place a hold on a book they have out.
- A user cannot place two holds on the same book.

The schema *LibraryState* gives the complete state of the library system:

$$
\begin{array}{|l}
\hline
\textit{LibraryState} \\
\textit{ParaLibrary} \\
\textit{LibraryDB} \\
\textit{HoldsInfo} \\
\hline
\operatorname{dom} \textit{instanceof} = \textit{available} \cup \textit{checkedout} \cup \textit{onhold} \\
\hline
\end{array}
$$

This has been modified to allow copies in the domain of *instanceof* to be on hold as well as being available or checked out.

In answering the following questions, you should explain the conditions you write and any additional assumptions you make.

(a) [7 marks] There is a new operation to place a hold on a book. The schema *PlaceHold* handles the case where this operation is successful.

$$
\begin{array}{|l}
\hline
\textit{PlaceHold} \\
\Delta \textit{LibraryState} \\
\Xi \textit{ParaLibrary} \\
\Xi \textit{LibraryDB} \\
n? : \textit{Person} \\
b? : \textit{Book} \\
\hline
n? \in \textit{borrower} \\
b? \in \operatorname{ran} \textit{instanceof} \\
(\forall c : \textit{Copy} \mid \textit{instanceof}(c) = b? \bullet c \in \textit{checkedout} \cup \textit{onhold}) \\
(\forall c : \textit{Copy} \mid \textit{instanceof}(c) = b? \bullet \textit{borrowedby}(c) \neq n?) \\
n? \notin \operatorname{ran}(\textit{holds}(b?)) \\
\textit{holds}' = \textit{holds} \oplus \{b? \mapsto \textit{holds}(b?) \frown \langle n? \rangle\} \\
\hline
\end{array}
$$

The first five conditions say that:

- The person requesting the hold must be a registered borrower.
- The book must be owned by the library.
- All copies of the book must be checked out or on hold.

- The requestor must not already have a copy of the book out.
- The requestor must not already have a hold on the book.

The last condition says that the new value of *holds* is obtained by adding $n?$ to the list of holds for $b?$.

(i) Explain briefly how this operation can be extended to form a robust operation for placing holds.

(ii) How would the definition of *PlaceHold* be affected if *holds* had been defined as a partial function, so that a book b was only in $\operatorname{dom} holds$ is there was at least one hold on b?

(b) [14 marks] The following is the schema *CheckOutCopy* from Diller's specification:

$$
\begin{array}{|l}
\hline
\textit{CheckOutCopy} \\
\Delta \textit{LibraryState} \\
\Xi \textit{ParaLibrary} \\
n? : \textit{Person} \\
c? : \textit{Copy} \\
\hline
n? \in \textit{borrower} \\
c? \in \textit{available} \\
\#(\textit{borrowedby}^{\sim} (\!| \{n?\} |\!)) < \textit{MaxCopiesAllowed} \\
\textit{available}' = \textit{available} \setminus \{c?\} \\
\textit{checkedout}' = \textit{checkedout} \cup \{c?\} \\
\textit{borrowedby}' = \textit{borrowedby} \cup \{c? \mapsto n?\} \\
\textit{previouslyborrowedby}' = \textit{previouslyborrowedby} \\
\textit{borrower}' = \textit{borrower} \\
\textit{staff}' = \textit{staff} \\
\hline
\end{array}
$$

(i) Modify *CheckOutCopy* so that it handles the case of checking out a copy that is not currently on hold. The new version should not issue a copy to a borrower if there is a (different) copy of the same book being held for him/her. You only need to show the parts of the schema that are added/changed.

(ii) Write a schema to define a new operation *CheckOutHeldCopy* which issues a copy which is on hold to the borrower for whom it is being held, making sure that all of the constraints on *HoldsInfo* are maintained.

(c) [14 marks] The following is the schema *Return* from Diller's specification:

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{Return}} \\
\Delta \mathit{LibraryState} \\
\Xi \mathit{ParaLibrary} \\
c? : \mathit{Copy} \\
\hline
c? \in \mathit{checkedout} \\
\mathit{available}' = \mathit{available} \cup \{c?\} \\
\mathit{checkedout}' = \mathit{checkedout} \setminus \{c?\} \\
\mathit{borrowedby}' = \{c?\} \lhd \mathit{borrowedby} \\
\mathit{previouslyborrowedby}' = \\
\qquad \mathit{previouslyborrowedby} \oplus \{c? \mapsto \mathit{borrowedby}(c?)\} \\
\mathit{borrower}' = \mathit{borrower} \\
\mathit{staff}' = \mathit{staff} \\
\hline
\end{array}
$$

(i) Modify *Return* so that it handles the case of returning a copy when there are no outstanding holds on that book. You only need to show the parts of the schema that are added/changed.

(ii) Write a schema to define a new operation *ReturnAndHold* to handle the case where there is a hold on a book being returned. In this case, the returned copy should be placed on hold for the first borrower with a hold on that book, and that borrower's hold removed.

Question 2. **[20 marks]**

The following specification statement defines the problem of finding the number of distinct elements in a non-empty sequence, given that elements of the sequence are in (non-strictly) ascending order:

$$z:[\, n = \#A \;\wedge\; n \geq 1 \;\wedge\; asc(A)\,,\; z = |\, elts(A)\,|\,]$$

where $\#X$ is the length of sequence X, and asc and $elts$ are defined as follows:

$$
\begin{aligned}
asc(X) &\;\widehat{=}\; (\forall\, i, j : 1..\#X \bullet i < j \Rightarrow X[i] \leq X[j]) \\
elts(X) &\;\widehat{=}\; \{i : 1..\#X \bullet X[i]\}
\end{aligned}
$$

Derive an algorithm that satisfies this specification, using the refinement calculus. Your algorithm should work by comparing successive pairs of elements in A and incrementing z when they are different. At each step in the derivation you should indicate the refinement rule used, and give a brief justification of any applicability condition.

Question 3. **[20 marks]**

The "refinement" relation ($\sqsubseteq$) on programs is defined as follows:

$$S \sqsubseteq S' \text{ iff } wp(S,R) \Rightarrow wp(S',R) \text{ for any assertion } R.$$

(a) Explain what it means for a compound statement to be *monotonic* with respect to $\sqsubseteq$. Prove that the **if** statement, **if** $[\!]_i B_i \rightarrow S_i$ **if**, is monotonic with respect to $\sqsubseteq$.

(b) Show that **abort** and **magic** are extreme elements with respect to the refinement ordering, i.e. **abort** $\sqsubseteq S \sqsubseteq$ **magic** for any statement S.

(c) The *dual*, $\overline{S}$, of a statement S is defined as $wp(\overline{S},\phi) \;\hat{=}\; \neg wp(S,\neg\phi)$. Show that:

(i) $\overline{\textbf{abort}} = \textbf{magic}$

(ii) $\overline{\textbf{magic}} = \textbf{abort}$

(iii) $\overline{\textbf{skip}} = \textbf{skip}$

(iv) $\overline{|[\ \textbf{var}\ x \bullet S\ ]|} = |[\ \textbf{con}\ x \bullet \overline{S}\ ]|$

Question 4. **[25 marks]**

The "data refinement" relation ($\preceq$) on programs can be defined as follows:

$S \preceq_{rep} S'$ iff $rep \circ S \;\sqsubseteq\; S' \circ rep$
for some predicate transformer *rep* which is monotonic and $\vee$-distributive.

(a) Prove that data refinement distributes through sequential composition, i.e. show that if $S_1 \preceq_{rep} S_1'$ and $S_2 \preceq_{rep} S_2'$, then $S_1;\, S_2 \preceq_{rep} S_1';\, S_2'$.

(b) Prove that data refinement is semi-transitive, i.e. show that if $S_1 \preceq_{rep_1} S_2$ and $S_2 \preceq_{rep_2} S_3$, then $S_1 \preceq_{rep_3} S_3$, for some rep_3.

(c) Suppose that *rep* is defined as $rep\,\phi \;\hat{=}\; (\exists\, a \bullet I \wedge \phi)$, where I is a relation between the abstract variables a and concrete variables c. Show that the above definition reduces to the alternative definition of data refinement:

$S \preceq_{(a,I,c)} S'$ iff $(\exists\, a \bullet I \wedge wp(S,\phi)) \Rightarrow wp(S', (\exists\, a \bullet I \wedge \phi))$
for any assertion ϕ not containing any variables in c.

(d) Suppose that *rep* is defined as $rep\ \phi \;\hat{=}\; (\forall\, a \bullet I \Rightarrow \phi)$, where I is a relation between the abstract variables a and concrete variables c. Derive a definition of data refinement, comparable to the one in part (c), using this definition of *rep*.

(e) Explain what is meant by *adequacy* of a data representation. Discuss, with examples, whether adequacy is a necessary requirement for data refinement.

****