CMU 17-651 Models of Software Systems Fall 2014

Homework #8: More Z

McLaughlin Due: October 13, 2014, 2014

Total Points:
NOTE: For this assignment you must format your answers using LaTeX and typecheck your answer to question 3 using *fuzz*, Z-EVES, or the Community Z tools.

1. The following questions refer to the handout on the Library Problem.

   (a) **5 points** Write an operation to see if a book is currently checked out.
   We are just reporting on the state, not modifying it. We need an input to identify the book of interest, and will output the status directly, for any book in the collection:

   StatusOfBook

   $\Xi Library$
   $book? : Book$
   $status! : STATUS$

   $book? \in books$
   $status! = (records\ book?).status$

   (b) Suppose you are curious to find out whether other people are interested in the same books as you.

   i. **5 points** Is it possible to write an operation that returns the set of books that you have checked out and later returned, and that were checked out by someone else after you returned them? If so, write the operation. If not, say why.
   No, not possible. Only the last user is remembered, so once someone else checks it out, you can no longer check previous borrowers.

   ii. **5 points** Is it possible to write an operation that returns the set of books that you have checked out and later returned, but that were NEVER checked out by someone else after you returned them? If so, write the operation. If not, say why.
   It's definitely possible. We won't be changing the state, just reporting on it. We need one input—the ID of the borrower in question:

   BooksReturnedByBorrower

   $\Xi Library$
   $borrower? : REAL\_PERSON$
   $books! : \mathbb{P}\ Book$

   $books! = \{b : books \mid (records\ b).lastuser = borrower? \land (records\ b).status = in\}$

2. The following questions refer to the handout on the Telephone Net.

   (a) **2 points** Can a telephone Call itself? If so, what is the effect of a subsequent Busy operation?

Yes, a telephone can Call itself.

A subsequent Busy operation will always return YES, since either the request didn't get connected because the phone was already in a connection, or the connection was made because it wasn't in a connection.

(b) **5 points** Write an explicit Connect operation to connect a pair of phones. (HINT: Connect is different from Call; Connect establishes a connection among phones on an outstanding satisfiable request. Note also that you can't use the Event framing schema here because it assumes that the starting state is an efficient net.)

The important things to include here are appropriate pre-conditions: that there be a pending request, and that the individual phones should not already be in connections, as connecting them to each other would then violate the *disj* invariant. The operation could have called *Event* in its declaration (since the problem description doesn't actually constrain the version(s) of the state space over which the operation should be defined), but an *Event* on the *EfficientTN* will result in the connection being made implicitly. As such, ΔTN is used, instead—*Connect* thus does for *TN* exactly what happens as a result of the additional invariants in *EfficientTN*.

$$
\begin{array}{|l}
\textit{Connect} \\
\hline
\Delta TN \\
ph? : PHONE \\
dialed? : PHONE \\
\hline
\{ph?, dialed?\} \in reqs \\
ph? \notin \bigcup cons \\
dialed? \notin \bigcup cons \\
cons' = cons \cup \{\{ph?, dialed?\}\} \\
reqs' = reqs \\
\hline
\end{array}
$$

(c) **2 points** Is it possible to place a call from a phone that's already busy?

Yes, it is possible to place a call from a phone that's already busy, though the new call won't become a connection until after the original connection is terminated.

(d) **2 points** Give an example that illustrates a situation in which a Hangup implies that a new connection will be made. Does the specification say which connection will be made if more than one is possible?

If phones A and B are connected, and phone C calls phone A, then when A and B hang up, C will be connected with A.

The specification does not state which connection will be made if more than one is possible (for example, if D also calls A in the above example).

(e) **2 points** If a Hangup operation is applied to a ph? that is not yet connected, what happens? Briefly speculate on how the specification would have to be changed to make this more realistic.

Nothing happens, i.e., it fails to cancel the request.

The system should be modified so that it can determine which phone in a request is the dialer and which the diallee. This way, if the dialer of an as-yet-unconnected hangs up prior to a connection being made, the system can remove the request.

(f) **5 points** Modify Busy so that it also indicates which phones ph? is connected to. What is the output if the input ph? is not connected at all?

All we need to do is add the additional output:

MoreInformativeBusy

$$\begin{array}{|l}
\textit{Busy} \\
\textit{connected}! : \mathbb{P}\,\textit{PHONE} \\
\hline
\textit{connected}! = \{c : \textit{PHONE} \mid \{\textit{ph}?, c\} \in \textit{cons} \land c \neq \textit{ph}?\} \\
\hline
\end{array}$$

The output will be the empty set if the phone is not connected.

3. The following scenario describes a typical classroom situation.

   (a) **5 points** A teacher needs to keep track of which homework assignments each student in the class has turned in. Each student in the class is given an ID, and at any time each student in the class has a (possibly empty) set of homeworks that have been turned in. The system should only keep records of the students in *this* class.

   Complete the schema with an appropriate invariant.

   $[\textit{ID}, \textit{STUDENTNAME}, \textit{HOMEWORK}]$

   ClassRecords

   $$\begin{array}{|l}
   \textit{student} : \textit{ID} \pfun \textit{STUDENTNAME} \\
   \textit{turnedIn} : \textit{STUDENTNAME} \pfun \mathbb{P}\,\textit{HOMEWORK} \\
   \hline
   \forall\, i, d : \textit{ID} \bullet i \neq d \Rightarrow \textit{student}(i) \neq \textit{student}(d) \\
   \operatorname{ran} \textit{student} = \operatorname{dom} \textit{turnedIn} \\
   \hline
   \end{array}$$

   Note: You are not allowed to change the state variables in the schema. Supply only an appropriate invariant.

   (b) **5 points (1 each)** Which of the following can be inferred from your definition of *ClassRecords*? Briefly justify your answer.

   Answers below are based on the schema above. Score based on actual given answer. Where correct for their schema, but not aboslutely, note that in comments.

   - No two students are assigned the same *ID*.
     True, the first line of the invariant says that.
   - A given student may have more than one *ID*.
     True, but then they would also have two different names. The student-to-ID relationship is a real world one, not somethig captured in this system.
   - There may be some *ID*s that are not used by the system.
     True, since *student* is declared to be a partial function.
   - All students that have an *ID* also have a set of *HOMEWORK*s.
     True, the second line of the invariant lets us infer that.
   - Any student who has a set of *HOMEWORK*s also has an *ID*.
     Yes, since we specified that the range of *student* is the same as the domain of *turnedIn*, every student with homework recorded has an ID.

   (c) **5 points** Write a schema *InitClassRecords* that defines an appropriate initial state space for the system. Explain why the initial state space is consistent with the state space invariant that you defined earlier.

   We init the system as you'd expect, with no students or homework.

$$\begin{array}{|l}
\textit{InitClassRecords} \\
\hline
\textit{ClassRecords}' \\
\hline
\textit{student} = \emptyset \\
\textit{turnedIn} = \emptyset \\
\hline
\end{array}$$

(d) **5 points** Write an operation to add a student to the class, provided that the student is not already a member of the class.

The operation will first check to make sure both ID and name are new (since either one matching would indicate the student was already in the class, as unique IDs get assigned unique names), and if so, add the new student to the roster and record initially that they have completed no homework:

$$\begin{array}{|l}
\textit{AddStudent} \\
\hline
\Delta \textit{ClassRecords} \\
\textit{student}? : \textit{ID} \\
\textit{name}? : \textit{STUDENTNAME} \\
\hline
\textit{student}? \notin \operatorname{dom} \textit{student} \\
\textit{name}? \notin \operatorname{ran} \textit{student} \\
\textit{student}' = \textit{student} \cup \{\textit{student}? \mapsto \textit{name}?\} \\
\textit{turnedIn}' = \textit{turnedIn} \cup \{\textit{name}? \mapsto \emptyset\} \\
\hline
\end{array}$$

(e) **5 points** Write a robust version of the operation that returns an error value if the student is already a member of the class. Use the schema calculus: you should not need to rewrite the original operation.

First, we add the output messages we'll need:

$$\textit{MESSAGE} ::= \textit{ID\_exists} \mid \textit{Name\_exists} \mid \textit{Success}$$

Next, we add a report that indicates that an operation has completed successfully:

$$\begin{array}{|l}
\textit{Success} \\
\hline
\textit{answer}! : \textit{MESSAGE} \\
\hline
\textit{answer}! = \textit{Success} \\
\hline
\end{array}$$

We'll report success in the event that an operation goes through without any difficulties, i.e., in any case where we are not reporting an error.

We also need two error schemata:

$$\begin{array}{|l}
\textit{IDExists} \\
\hline
\Xi \textit{ClassRecords} \\
\textit{id}? : \textit{ID} \\
\textit{answer}! : \textit{MESSAGE} \\
\hline
\textit{id}? \in \operatorname{dom} \textit{student} \\
\textit{answer}! = \textit{ID\_exists} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\hline \textit{NameExists} \\
\Xi \textit{ClassRecords} \\
\textit{name}? : \textit{STUDENTNAME} \\
\textit{answer}! : \textit{MESSAGE} \\
\hline
\textit{name}? \in \operatorname{ran} \textit{student} \\
\textit{answer}! = \textit{Name\_exists} \\
\hline
\end{array}$$

That gives us the robust operation:

$$\textit{RobustAddStudent} \mathrel{\widehat=} (\textit{AddStudent} \land \textit{Success}) \lor \textit{IDExists} \lor \textit{NameExists}$$

(f) **5 points** Write an operation, *DeadBeats*, that returns the set of *ID*s (not student names!) of students who have not turned in more than one homework assignment. We just need one output, which will be the set of IDs associated with a name that is associated with fewer than two assignments.

$$\begin{array}{|l}
\hline \textit{AddStudent} \\
\Xi \textit{DeadBeats} \\
\textit{answer}! : \mathbb{P}\, \textit{ID} \\
\hline
\textit{answer}! = \{ \textit{id} : \textit{ID} \mid \#\textit{turnedIn}(\textit{student}(\textit{id})) < 2 \} \\
\hline
\end{array}$$

(g) **2 points** Consider the following globally-defined function that determines whether a student's status is ok or not, based on a comparison with the set of total assignments that could have been turned in.

$$\textit{STATUS} ::= \textit{ok} \mid \textit{not\_ok}$$

$$\begin{array}{|l}
\textit{StatusOf} : \mathbb{P}\, \textit{HOMEWORK} \times \mathbb{P}\, \textit{HOMEWORK} \to \textit{STATUS} \\
\hline
\forall\, \textit{student}, \textit{total} : \mathbb{P}\, \textit{HOMEWORK} \bullet \\
\quad (\textit{StatusOf}(\textit{student}, \textit{total}) = \textit{ok}) \Leftrightarrow (\#(\textit{total} \setminus \textit{student}) < 2)
\end{array}$$

Explain in informal terms when the status of a student is not ok.

A student is not ok if there are two or more assignments they have not yet turned in.