CMU 17-651 Models of Software Systems Fall 2018

Homework #8: More Z

Dario A Lencina-Talarico Due: 17 October 2018

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) Write an operation to see if a book is currently checked out.
   Assuming that we need to find if a particular copy of a book is checked out and that we do not need to do handle the scenario when the book is not in the library.
   Using the definitions of *Library*, *BookOp*, *Copy* provided in the Lecture:

   Defining type to answer:
   Answer ::= Yes | No

$$\begin{array}{|l}
\textit{IsBookCheckedOut} \\
\hline
\Xi BookOp \\
book? : Copy \\
answer! : Answer \\
\hline
book? \in books \\
(\#\{b : books \mid (records\ b).status = out\} = 1) \Rightarrow answer! = Yes) \land \\
(\#\{b : books \mid (records\ b).status = out\} = 0) \Rightarrow answer! = No) \\
\hline
\end{array}$$

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

   i. 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.
   Given the definition of Data, which is the type used to store status and lastuser, it is not possible to preserve the history of the checkouts because every new book checkout overwrites the *lastuser* user field.
   We would have to modify our records data structure to create a new record every time a book is Checked in or out and if someone wanted to pull the history of a book, they would have to query all records for a given book, the last known record would be the current state.

   ii. 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.
   Given the previous answer, it is possible, here's the operation:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\_\ \mathit{GetAllTheBooksThatUserHasCheckedOutAndLaterReturned}\ \_\_\_\_\_\_}\\
\Xi \mathit{BookOp} \\
\mathit{person}? : \mathit{REAL\_PERSON} \\
\mathit{user\_books}! : \mathbb{F}\,\mathit{Copy} \\
\hline
\mathit{user\_books} = \{b : \mathit{books} \mid (\mathit{records}\ b).\mathit{lastuser} = \mathit{person}? \land (\mathit{records}\ b).\mathit{status} = \mathit{in}\} \\
\hline
\end{array}
$$

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

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

   According to the CONNECTIONS set that was defined in the Lecture, it is possible: CONNECTIONS $= \{\{\}, \{1\}, \{2\}, \{3\}, \{1,2\}\{1,3\}, \{2,3\}, \{1,2,3\}\}$

   There's is no side effect of calling *Busy* in the sense that the state of the Telephone Net is not changed, this is guaranteed by the invariant and the fact that part of the *Busy* spec says that $reqs' = reqs$

   A new set $\{ph?, dialled?\}$ will be added to the reqs and cons sets where $ph? = dialled?$.

   (b) 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.)
   Did not try :(

   (c) Is it possible to place a call from a phone that's already busy?
   Lets consider the *Call* operation defined in the lecture, $ph?, dialled? : PHONE$ are the inputs to the Z operation.

   The backing data structure of the TelephonyNet Schema is $\mathbb{P}$ CONNECTION

   Placing a call is defined as $reqs` = reqs \cup \{\{ph?, dialled?\}\}$

   It is possible to place a call from a phone that is connected and it has to be made to a phone that is different than the one that we are already connected to.

   In order to make this a more realistic model, I propose that we do at least 2 things:
   1. Transitioning a request to a connection should include the possibility of a failure due to various factor such as network quality, electrical, noise etc.
   2. When a phone disconnects from a conference call (more than 2 peers connected) we should not drop all the participants like we do right now.

   (d) 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? I was unable to find this scenario, probably I did not explore enough.

   (e) 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.

The hangoup operation is modeled as
$reqs' = reqs \setminus \{c : const | ph? \in c\}$
If ph is not in $const$ then it returns an empty set, then $reqs' = reqs \setminus \emptyset = reqs$

In order to make it more realistic, I propose adding a new output called success! with type $Success ::== YES | NO$
so that we can be more explicit about the success or failure of this opearation.

(f) 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?
I if not connected, dialled will return a $\emptyset$, else it will return a set with all the connections that ph has.
YesOrNo::= Yes | No

BusyModified

$Event$
$ph? : PHONE$
$activeConnections! : \mathbb{P} CONNECTION$
$busy! : YesOrNo$

$reqs' = reqs$
$activeConnections! = \{con : cons \mid ph \in con\}$
$busy! = Yes \Leftrightarrow ph? \in \bigcup cons$

3. The following scenario describes a typical classroom situation.

(a) 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.

$[ID, STUDENTNAME, HOMEWORK]$

ClassRecords

$student : ID \rightarrow\!\!\!\!\!\!+ \; STUDENTNAME$
$turnedIn : STUDENTNAME \rightarrow\!\!\!\!\!\!+ \; \mathbb{P}\, HOMEWORK$

$dom\ turnedIn \subseteq range\ student$

Complete the schema with an appropriate invariant.

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

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

- No two students are assigned the same *ID*.
  This can be inferred since the definition of student as $ID \rightarrow\!\!\!\!\!\!+ \; STUDENTNAME$ guarantees that.

  Meaning, it is not possible to map the same id to two different students.

- A given student may have more than one ID. The proposed definition does allow that if the enrollment people happen to use a variation of the student's name for each id.
- There may be some IDs that are not used by the system. To the best of my knowledge, the assignement does not provide a very specific definition of the ID type or domain, I think it is safe to say that there will be many ids that wont be used by the system.
- All students that have an ID also have a set of $HOMEWORK$s. The current definition of the system does not provide this guarantee because the turnedIn partial function is not initialized
- Any student who has a set of $HOMEWORK$s also has an ID. Yes, the invariant that I introduced focuses on this property by defining that the domain of turnedIn has to be in or equal the range of student, that implies that the sutdentname has to be associated with an id.

(c) 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. I could not find how to init partial functions, is the intention to enroll all the students here?

If that is not the case then an Empty Initializer will respect the invariant as both the *stuent* and *turnedIn* partial functions are empty, meaning, no range and no domain.

InitClassRecords

$ClassRecords$

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

AddStudent

$\Delta ClassRecords$

$id? : ID$

$student\_name? : STUDENTNAME$

$id \notin dom\ student$

$student' = student \cup \{id? \mapsto student\_name?\} \land turnedIn' = turnedIn$

(e) 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.

Declaring return type to encapsulate success/error.

$ADD\_STUDENT\_RESULT$ ::= success | $student\_already\_enrolled$

AddStudentSuccess

$result!$:ADD_STUDENT_RESULT

$result! : success$

$$\begin{array}{l}
AddStudentError \\
\hline
result! : \text{ADD\_STUDENT\_RESULT} \\
\hline
id \in dom\ student \\
result! : student\_already\_enrolled
\end{array}$$

With the defined schemas we can proceed to create the robust AddStudent:
$RAddStudent \mathrel{\widehat{=}} AddStudent \wedge AddStudentSuccess \vee AddStudentError$

(f) Write an operation, $DeadBeats$, that returns the set of IDs (not student names!) of students who have not turned in more than one homework assignment.

$$\begin{array}{l}
DeadBeats \\
\hline
\Xi ClassRecords \\
dead\_beats! : \mathbb{P} ID \\
\hline
\forall\, id : (domstudent) \bullet \\
\#\{turnedIn(student(id)\} < 2 \wedge dead\_beats = dead\_beats \cup id, student' = student \wedge student' = student
\end{array}$$

(g) 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.

$$STATUS ::= ok \mid not\_ok$$

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

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

I know that this is a recursive function but I did not understand when the iteration stop, maybe when $\#(total\ \ student) < 2$? What does the X operator mean?