CMU 17-651 Models of Software Systems Fall 2018

Homework #9: Z Refinement

Dario A Lencina-Talarico Due: 29 October 2018

In a previous homework you developed a Z model of a class register. For the purposes of this homework, consider the following specification for this system: A teacher wants to keep a register of students in the class, and to record which students have completed their homework.

You may assume the given set *Student*:

$[Student]$

The teacher models the state of the register by recording the students who have enrolled in the class and, also, those who have completed the homework. Naturally, only students who are enrolled can have completed the homework.

Register

$$enrolled : \mathbb{P}\, Student$$
$$completed : \mathbb{P}\, Student$$
$$completed \subseteq enrolled$$

At the start of each course, no students are enrolled

InitRegister

$$Register$$
$$enrolled = \emptyset$$

which has a natural consequence of $completed = \emptyset$.

New students enroll in the class one at a time. Further, once students are enrolled in the class they cannot enroll again.

AddStudent

$$\Delta Register$$
$$name? : Student$$
$$name? \notin enrolled$$
$$enrolled' = enrolled \cup \{name?\}$$
$$completed' = completed$$

The existing homework completion status is unchanged with the assumption that new students will complete the homework at a later time.

Now, consider a concrete implementation of the register state space that represents the class register as two sequences: one that contains the enrolled students, and the other that is a sequence of *YES* or *NO*, indicating whether the corresponding student has completed the course. (You may do this either using infinite sequences as modeled in Spivey's example, or using Z's built-in notion of finite sequences.)

1. Write a schema to model the concrete state space. Think carefully about the representation invariant.
   Even when in modern programming languages like go, C++ or Haskell we have the data structures required to implemented the Register schema without transforming the model, I present a version of Register using array to illustrate the concept of a concrete state space.

   Similarly to Spivey's paper, I used a couple of arrays to represent the enrolled and completed Students, the only difference is that to represent students that completed the homework, I propose using an array of $\mathbb{N}$ where we will store the indexes of the enrolled students that have completed the homework.

$$
\begin{array}{|l}
\hline
Register1 \\
\hline
enrolled1 : \mathbb{N} \rightarrow Student \\
completed1 : \mathbb{N} \rightarrow \mathbb{N} \\
enrolled\_size : \mathbb{N} \\
completed\_size : \mathbb{N} \\
\hline
(\forall\, i, j : 1..enrolled\_size \bullet i \neq j \Rightarrow enrolled1(i) \neq enrolled1(j)) \land \\
(\forall\, i, j : 1..completed\_size \bullet i \neq j \Rightarrow completed1(i) \neq completed1(j)) \land \\
(completed\_size <= enrolled\_size) \land \\
(\forall\, i : 1..completed\_size \bullet completed1(i) <= enrolled\_size) \\
\hline
\end{array}
$$

2. Write a schema representing the initial state for the concrete state space. Argue (informally) that the initial state satisfies the state invariant.
   I decided to initialize both the enrolled and completed arrays to emptysets, and both sizes to 0.
   This initialization meets the invariant because *completed_size* is less than *enrolled_size*, each element in completed holds a value ¡= *enrolled_size*. Also, all elements in both enrolled and completed are different.

$$
\begin{array}{|l}
\hline
InitRegister1 \\
\hline
Register1 \\
\hline
enrolled1 : \emptyset \\
completed1 : \emptyset \\
enrolled\_size : 0 \\
completed\_size : 0 \\
\hline
\end{array}
$$

3. Write the abstraction function (as a schema) that relates the concrete state space and the previously defined abstract state space.

$$
\begin{array}{|l}
\hline
Abs \\
\hline
Register \\
Register1 \\
\hline
enrolled = \{i : 1..enrolled\_size \bullet enrolled1(i)\} \\
completed = \{i : 1..completed\_size \bullet enrolled1(completed1(i))\} \\
\hline
\end{array}
$$

4. Argue that the initial state is a reasonable concrete representation by showing that it corresponds to the initial state of the abstract state space.

$$Formally : InitRegister1 \land Abs \vdash InitRegister$$

$$\begin{array}{lll} \text{enrolled} & & \\ = & \{i : 1..enrolled\_size \bullet enrolled1(i)\} & [\text{Abs}] \\ = & \{i : 1..0 \bullet enrolled1(i)\} & [\text{enrolled\_size} = 0] \\ = \emptyset & & \\ \text{completed} & & \\ = & \{i : 1..completed\_size \bullet enrolled1(completed1(i))\} & [\text{Abs}] \\ = & \{i : 1..0 \bullet enrolled1(completed1(i))\} & [\text{complete\_size} = 0] \\ = \emptyset & & \end{array}$$

5. Produce a concrete version of *AddStudent*.

$$\begin{array}{|l} AddStudent1 \\ \Delta Register1 \\ student? : Student \\ \hline \forall i : 1..enrolled\_size \bullet student? \neq enrolled1(i) \\ enrolled\_size' = enrolled\_size + 1 \\ enrolled1' = enrolled1 \oplus \{student?\} \\ completed1' = completed1 \\ completed\_size' = completed\_size \\ \hline \end{array}$$

6. Argue informally (in the style of Spivey's paper, and the lectures slides) that this concrete operation is correct. (HINT: first show "applicability" — that it is defined over an appropriate set of concrete states; then show "correctness" — that it produces results that are consistent with the abstract operation.)
Huge disclaimer: I tried to follow a very similar format than Spivey's, sometimes it felt like I was just replacing values which is always weird:

The proposed AddStudent1 schema has the same inputs and outputs as AddStudent, but operates on the concrete instead of the abstract state. It is a correct implementation of AddStudent because of the following two facts:

(a) Whenever AddStudent is legal in some abstract state, the implementation AddStudent1 is legal in any corresponding concrete state.

(b) The final state which results from AddStudent1 represents an abstract state which AddStudent could produce.

Let us look at the reason why these two facts are true. The operation AddStudent is legal exactly if its pre-condition $student? \notin enrolled$ is satisfied, if this is so, the predicate
$enrolled = \{i : 1..enrolled\_size \bullet enrolled1(i)\}$
from Abs tell us that student? is not one of the elements enrolled1(i): $\forall i : 1..enrolled\_size \bullet student? \neq enrolled(i)$

This is the precondition of AddStudent1.
To prove the second fact we need to think about the concrete states before and after the execution of AddStudent1, and the abstract states they represent according to Abs. The two concrete states are related by AddStudent1, and we must show that the two abstract states are related as prescribed by AddStudent:
$enrolled' = enrolled \cup \{student?\}$

The domains of these two functions are the same, because

enrolled'

$\supseteq$ $completed'$	(invariant after)
$\supseteq$ $\{i : 1..completed\_size' \bullet enrolled1'(completed1'(i))\}$	(from Abs')
$\supseteq$ completed	

NOTE: Question 3 asks you to define the concrete initial state and show that it is a valid instance of your concrete state space, while 4 asks you to show that this initial state is not only possible, but also corresponds to the abstract initial state space.