CMU 17-651 Models of Software Systems Fall 2018

Homework #7: Invariants and Introduction to Z

Dario A Lencina Talarico Due: 15 October 2018

Part 1: Invariants

Consider the Diverging Counter example of Chapter 10 of GWC09. Prove that $x + y = 0$ is an invariant of the *DivergingCounter* state machine.

DivergingCounter = (
$[x, y : \mathbb{Z}]$,
$\{s : [x, y : \mathbb{Z}] \mid s(x) = -s(y)\}$,
$\{poke(i : \mathbb{Z})\}$,
δ ==

poke(i: $\mathbb{Z}$)
pre $i > 0$
post $x' = x + i \land y' = y - i$

).
1. Base case: show that θ holds in the initial state.
Here there's only one initial state:
$[x = 0;\ y = 0]$
Proof:
x + y = 0

= [initial state]
0 + 0 = 0
= [aritmetic]
0 = 0
2. Induction step on inc:

Show: $\theta(s), pre(s), post(s, s') \vdash \theta(s')$
That is, from $x' = x + i \land y' = y - i$

$\theta(s)$ == $x + y = 0$
prove that $x' + y' = 0$
Proof:

y’

= [postcondition]
y - i
= [hypothesis introduction y + x = 0, or y = -x]
-x - i
= [factorizing]
-(x + i)
= [replacing definition of x’]
-x’

y’ = -x’ is equivalent to y’ + x’ = 0
(NOTE: In your proof use style C (in Section 10.1.1) of reasoning about invariants and a similar degree of formalism as in the lecture on this topic.)

Part 2: Z

NOTE: For this part of the assignment you must format your answers using LaTeXand typecheck the answers using *fuzz*, Z-EVES, or the Community Z tools.

Write a Z specification of the following system. Your specification should include sufficient explanatory prose to make it easily understandable. (The prose is important—answers with little or no prose will receive a low grade.)

A teacher wants to keep a register of students in the class, and to record which students have completed their homework.

Let the given set *Student* represent the set of all students who might ever be enrolled in a class:

$[Student]$

Specify each of the following:

1. The state space for a register.
   Reasoning: I propose using two powersets of students, one to represent the enrolled students and another one for the students that completed their homework.
   I added the invariant completed ⊆ enrolled to capture the fact that all students that completed their homework have to be enrolled
   HINT: use two sets of students:

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

   $completed \subseteq enrolled$

2. An operation to enroll a new student.
   Reasoning: I propose using the union operation between the currently enrolled vehicles and the new student.
   It is important to mention that the union operation is defined for sets, that is why *student*? is wrapped in a set.

Enroll

$\Delta Register$
$student? : Student$

$student? \notin enrolled$
$enrolled' = enrolled \cup \{\text{student?}\}$
$completed' = completed$

3. The initial state(s) for your state space.

   Reasoning: The proposed z-spec is to initialize both sets as empty sets.

InitRegister

$Register$

$enrolled = \emptyset$
$completed = \emptyset$

4. An operation to record that a student (already enrolled in class) has completed the homework.
   Reasoning: The proposed Z spec adds student? to the completed power set by defininig a union.

RegisterHomeworkCompletion

$\Delta Register$
$student? : Student$

$student? \in enrolled$
$completed' = completed \cup \{\text{student?}\}$
$enrolled' = enrolled$

5. An operation to inquire whether a student (who must be enrolled) has completed the homework (the answer is to be either 'Yes' or 'No').
   Reasoning: The previous statement specifies the expected return values, to meet the requirements, we need to define a new type to answer:

   Answer ::= Yes | No
   HasCompletedTheHomeworkSuccess is used to express the successful HasCompletedTheHomework check. it returns answer:Yes, all it does is to check if the student is in the completed set.

HasCompletedTheHomeworkSuccess

$\Xi Register$
$student? : Student$
$answer! : Answer$

$student? \in completed$
$answer! : Yes$

   HasCompletedTheHomeworkErr is used to express the error scenario, meaning, student is not in the completed set, it returns answer:No

HasCompletedTheHomeworkErr

$\Xi Register$
$student? : Student$
$answer! : Answer$

$student? \notin completed$
$answer! : No$

Reasoning: Only one of the expressions will meet the preconditions so $\vee$ is the right relation to use:

HasCompletedTheHomework $\hat{=}$ *HasCompletedTheHomeworkSuccess* $\vee$ *HasCompletedTheHomeworkErr*

6. A robust version of the system. (Be sure to use the schema calculus, as illustrated by the class lecture and the paper by Spivey on Z.)
Declaring type to return a result:
REPORT ::= ok | already-enrolled | not-enrolled
Reasoning: We propose using REPORT to return the success or failure of each operation.
We also need to define some helper schemas to embed the RESULT.
Success is used when the operation has been successful.

Success

$result! : REPORT$

$result! : ok$

NotEnrolled is used to capture the scenario where the student is not enrolled in the register, all we do is checking if the student is in the enrolled set.

NotEnrolled

$\Xi Register$
$student? : Student$
$result! : REPORT$

$student? \notin enrolled$
$result!$:not-enrolled

AlreadyEnrolled is used to handle the error scenario of the enroll operation, all we do is a precondition that checks if the student is in the enrolled set.

AlreadyEnrolled

$\Xi Register$
$student? : Student$
$result! : REPORT$

$student? \in enrolled$
$result!$:already-enrolled

Robust versions of the operations:

RREnroll $\hat{=}$ (*Enroll* $\wedge$ *Success*) $\vee$ *AlreadyEnrolled*
RRegisterHomeworkCompletion $\hat{=}$ (*RegisterHomeworkCompletion* $\wedge$ *Success*) $\vee$ *NotEnrolled*
HasCompletedTheHomework is already a robust method so no need to modify.