CSSE 373 - Formal Methods in Spec. and Design

Homework 5

You will practice modeling some complex problems using Alloy in this lab/homework. This is an individual work. You are allowed to discuss this work with your classmates; however, you must write your own model and turn in your work separately.

Please create **./tunrins/Homework5.pdf** to answer the questions asked in this lab/homework.

Lab: Protocol Modeling

Step 1: Read Chapter 6.1 and re-write the model given in Figure 6.1 - 6.2.

Step 2: Additionally you will provide source code comments for each signature, facts, predicates, assertions, and check and run commands. [**5 points**]

Step 3: You will use the ***pdf*** file to answer the following: [**15 points**]

1. After completing the model you will add the following command and execute it:

   pred show { **some** elected }
   run show **for** 3 Process, 4 Time

   This command generates an instance similar to Figure 6.4. Project the instance on Time, capture the 4 snapshots, and explain what is happening in each snapshot.

2. Result of "AtMostOneElected" [Snapshot of counterexample or just state that the assertion is valid]

3. Result of "AtLeastOneElectedWithAnIssue". Use the following assertion and check command:

   assert AtLeastOneElectedWithAnIssue {
   some t: Time | **some** elected.t
   }
   check AtLeastOneElectedWithAnIssue for 3 **but** 7 Time

   It should give you a counterexample. Explain what went wrong here.

4. Result of "AtLeastOneElected" as provided in the book's model. Explain why this assertion is valid.

5. Run ***looplessPath*** **for 3 Process, 12 Time** versus **3 Process, 13 Time**. What is the significance of not getting an instance for the scope of 13 Time?

Homework: Modeling a New Problem from Scratch

All of you must have played the Tic-Tac-Toe game before. (If you have not, then stop by my office; we can play a couple of games!) You will model the Tic-Tac-Toe game in Alloy and verify few properties of the game for this homework. The questions that follow will help you incrementally build your model and arrive at a correct solution. Remember, there is no one right answer to these problems, so feel free to use your creativity to come up with an elegant solution.

Problems

Q1: Modeling the State of the Game (10 points)

X		O
	O	
O		X

Figure 1: A game state where O won.
(X = Cross, O = Nought)

Here are a few things that you need to consider to model the state of the game.

a. Each game state must somehow represent the configuration of the game board.
b. There are altogether 9 positions that can be assigned a marker (Cross or Nought).
c. A game state may also need to keep track of next turn (either Cross's or Nought's turn)

Q2: Specifying a Win State (10 points)

Write a predicate that takes two parameters, a game sate and a marker (Cross or Nought), and checks if that marker won the game. Here are two example headers from which you could choose:

pred Win[s: GameState, t: Marker] { … } OR
pred GameState.Win[t: Marker] { … }

Remember there are 8 different ways to win a game. You need to specify all of them. Figure 1 shows one way in which all of the cells in a diagonal are equal and hence, *Nought* is the winner. You will check to see if your predicate is working by running the predicate for exactly 1 *GameState*.

As an answer to this question, you will code the *Win* predicate in the Alloy file and provide a **snapshot** of the instance generated by Alloy Analyzer in the pdf file. You will further interpret the instance and provide a **tabular representation** as shown in Figure 1 for the instance in the pdf file.

Q3: Specifying a Draw State (10 points)

Write a predicate that takes a game state as its parameter and checks if the game is a draw. Again, here are two headers that you could use:

pred Draw[s: GameState] { … } OR
pred GameState.Draw[] { … }

As an answer to this question, you will code the predicate and also show the **snapshot** of an instance. Furthermore, you will interpret the instance by providing a **tabular representation** as shown in Figure 1.

Q4: Specifying the Next Turn (5 points)

Your game should advance by changing one position at a time in the game board. Since there are two players (*Cross* and *Nought*), they need to switch turns in each move. Write a helper function that would take a game state as a parameter and returns the next of the *next* turn for the supplied state. Assuming *Noughts* and *Crosss* are abstracted as *Marker*, here is its header:

fun NextTurn[s: GameState]: Marker { … }

If next turn for *s* is *Nought*, then the function should return *Cross* and vice versa.

Q5: Initial State (5 points)

The initial state of the game should have an empty board and should set the turn (a *Marker*) for the next move. Specify these constraints in a predicate. Your predicate could take two parameters, a *GameState* and a *Marker* for the next turn. Here are the two headers that you could use:

pred Init[s: GameState, t: Marker] { … }
pred GameState.Init[t: Marker] { … }

Q6: Transition from One state to Another (10 points)

Write a predicate that takes two game states (*s* and *s'*) as its parameter and specifies the constraints that associates one empty cell location of s in *s'* with either *Cross* or *Nought* based on the next turn of *s*, after which it will set the next turn for *s'*. Note that nothing except one cell position should change between *s* and *s'*. Here is the header that you could use:

pred Transition[s, s': GameState] { … }

Q7: Game Trace (10 points)

You will specify a fact that traces a game from start to end.

Q8: Winning Trace (10 points)

Write a predicate that specifies that a game was eventually won by a marker (either *Nought* or *Cross*). Run a command that shows that the game can be won within 5 moves (or 6 game states including the initial empty state). You will include **snapshots of instances projected over *GameState*** with corresponding **tabular representation** of each game state in the pdf file.

Q9: Trace Ending in a Draw (10 points)

Write a predicate that specifies that a game eventually resulted in a draw. Run a command that shows an instance of such a game for 9 moves (or 10 game states include the initial state). You will include both **snapshots** and **tabular representations** of the instance in the pdf file (showing the first two and the last two game states with corresponding tables is just fine for this one).

Note that you may need to define more functions or predicates than specified in this instruction to achieve your goal. Hope you will enjoy this assignment! Good luck!