Heriot Watt University

Rigorous Methods for Software Engineering

Coursework 2

By

Vytautas Tumas

4th December 2015

Introduction

The aim of the coursework is to write an Alloy model for one of the given systems. I chose to model Tic-Tac-Toe game. The model is made up of five objects, four facts, six predicates and three assertions. The model uses two utility libraries. To show that moves are an ordered set I used the built in *"ordering"* library which creates a linear ordering over supplied element. The second library used is *'relation'* which allows me to easily access range and the domain of any relation.

Model

The Tic-Tac-Toe game has two player X and O, these are represented by Player signature. Each cell on the game board extends an abstract Cell sig, and is given a unique name ranging from *C1* to *C9*. The abstract cell contains a player field, which is a binary relationship between a cell and at most one player. This filed will hold the 'owner' of the cell. In addition, there is a constraint ***"Domain"*** which specifies that if the cell is not empty, the domain if the player field is the cell itself.
The board rows are modelled similarly to cells. There are three unique rows, which extend the abstract row signature. The abstract row signature contains a cell field, which is a sequence of cells. To ensure that a row has only three cells I have added a constraint on the cardinality of the row. In addition, I had to define a fact ***"OneCell"*** which ensures that each cell appears in at most row, i.e. one cell cannot be in two rows.
Finally the game board is defined by the Grid signature. The object has a row field, which is a sequence of rows, the field has a cardinality constraint to ensure that the grid has only three rows. The ***"OneRow"*** fact ensures that a row appears in at most one grid.
Player move is represented by the Move signature. It contains three fields: *p* – a player currently making a move, *pos* – the cell which player will occupy, and the game board. The player move has only a single constraint, which says that *iff* the *pos* cell is empty it can have a player assigned to it.
To show the player move I wrote a ***"Movement"*** fact. The fact specifies that for all movements, alternate players move each turn, and no two moves exist with the same *pos* value.

Assertions

I developed three assertions for the model. First, I assert that the board is full, i.e. there are no empty cells, after **n** moves. The assertion is true only if we run the example for 9 movements, if there are any less, Alloy will easily find an example

where there are empty cells.
Second developed assertion was a win condition. It specifies that there exist a lone player, who makes a winning move. A move is a wining move only if one of the predicates are true: ***"WinRow"*** **-** all cells in a row belong to the same player, ***"WinColumn"*** – cells in each column are the same, and ***"WinDiag"*** – which checks that the diagonal cells belong to the same player.
The final assertion I made was that there can be no winners after less than 5 moves. However, this assertion proved my model to be quite inconsistent. It turned out that my game model is missing a clause which would specify that at the beginning the board is empty, as the result the ***"NoWin"*** assertion turned out to be false just after one move, as Alloy would randomly fill some of the cells. This bug can be fixed by adding a fact, which states that at the first move the board is empty. But here I realised a much deeper issue with the model, my move representation does not show how the board changes, i.e. there is no way to check that a board is empty before the first move is made. This issue can be solved by introducing a new field *board'* which represents the board after a move has been made, thus allowing me to specify that the initial board is empty. However, due to some unforeseen circumstances I was not able to implement this property.

Summary

The Alloy modelling tool is a suitable choice to model a simple Tic-Tac-Toe game, it allowed me to precisely model the game board, players and the rules. In addition, the model visualisation helps to identify the flaws in the model and fix them. However, I believe that models of complicated software would turn out to be difficult to analyse as the more objects there in the model, the more cluttered the visual aid is, thus making it difficult to understand.