ECSE 429 (Software Validation) Assignment One Report

Geoffrey Long (*260403840*)
Kelly Morrison (*260459446*)

October 4, 2015

1 Running the Source Code

Import the ConformanceTest.zip file into Eclipse using the Import feature of Eclipse (File $\rightarrow$ Import $\rightarrow$ General $\rightarrow$ Existing Projects into Workspace $\rightarrow$ Select archive file). To generate tests for a given state machine (xml format), add the full path of the xml file to the argument of the project (Run $\rightarrow$ Run Configurations $\rightarrow$ Arguments $\rightarrow$ Program Arguments), then click "apply" then click "run". This will run the main class, ConformanceTest, which automatically generates the JUnit Test file, which is called Generate$< ClassName >$ where ClassName is the name of the class associated with the xml file. You will find this Test class saved in the same package as the implementation of the state machine. Open this class and run it (Run $\rightarrow$ Run As $\rightarrow$ JUnit Test).

2 Discussion of Manual Changes

When we run our automatically generated test class on the CCoinBox state machine, we have 2/10 tests that fail. This is because certain conditions are not met. Specifically, when you are in the allowed state of the machine, there are three transitions that execute the vend event. These three transitions all depend on the value of curQtrs and should execute only when the condition allows. Although a conformance test is generated for each of these paths, the round trip path up to the "allowed" state, and therefore the value of curQtrs, will be the same for all three tests. In this case, the value of curQtrs is 2 for all three transitions. The tests checking the other paths - one that requires curQtrs==3 and another for curQtrs¿3 - will always fail.

This issue was dealt with in the manual implementation of the test class, TestCCoinBox, via manipulation of the condition variable. In this class, we added a while loop to increment the value of curQtrs until its value satisfies the condition for the transition being tested. We also have an if statement that verifies that the value is correct, to ensure the assertTrue statement will pass. After adding these manual changes, our test class passes for all ten conformance tests.

This shows that our automatically generated test class will not be able to handle cases where there are multiple conditions for a given event at a specific state. Based on the previous transitions in the round trip path, the condition variable will have a specific value and will only meet certain conditions for the same event. Although it is possible to automatically generate code to ”force” the values to match the conditions, this might not be desirable. It is difficult to say if a condition is failing because it is supposed to fail, or because it simply didn’t take the right path through the tree. Essentially, it would be hard to automatically generate a test class that takes care of these cases without knowing the nature of the state machine.

It is worth noting that similar behavior can be seen in the automatically generated test class of the Legislative state machine. There are 2/6 tests that fail because the value of the variable isCommonBill does not match the expected value.

3 Discussion of CCoinBox Defects

In the addQtr() method of the CCoinBox.java class, there is an error in the case allowed section (line 142). According to the diagram on the PDF and the XML file, when you are in the allowed state, and you call the addQtr() method, the new state should be allowed, but in the java code, the new state is set to notAllowed. We updated the value of the new state to allowed to match the state machine from the XML file.

Similarly, in the reset() method and the returnQtrs() method of the CCoinBox.java class, there are errors in the allowed case. For both methods, when you are in the allowed state and the new state is empty, the allowVend Boolean should be set to false. This is again according to the XML file and the PDF. We added this statement to both the returnsQtrs() and reset() methods in the allowed cases (lines 113 and 175 respectively).

4 Generating Sneak Path Test Cases

The main challenge of automatically generating the sneak path test cases from a given state machine conforming to the metamodel given to us is that we do not know the nature of the state model so it would be hard to determine which events are illegal for a given state.