# Conformance Test Report

**ECSE 429: Fall 2015**

**Group 19**

**Geoffrey Long:**

**Kelly Morrison: 260459446**

# How to Run our Source Code

Import the ConformanceTest.zip file into Eclipse using the *Import* feature of Eclipse (File – Import – General – Existing Projects into Workspace – Select archive file). To run the code on a given state machine (xml format), add the path of the xml file to the argument of the project (Run – Run Configurations – Arguments – Program Arguments) and then run the project (Run). This will run the main class, ConformanceTest, which automatically generates the JUnit Test 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 – Run As – JUnit Test).

# 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 and these transitions depend on the value of curQtrs. Although one conformance test is generated for each of these paths, the value of curQtrs will be the same for all three tests because the round trip path up until the “allowed” state is the same for all three transitions. In this case, the value of curQtrs=2 for all three transitions, so the tests that check if curQtrs==3 and curQtrs>3 will always fail.

To fix this, in the test class we implemented manually, TestCCoinBox, we added a while loop to increment the value of curQtrs until the desired value is reached. 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 means that our automatically generated test class will not be able to handle the 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. We see similar behavior 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. It would be hard to automatically generate a test class that takes care of these cases when we do not know the nature of the state machine.

# Discussion of any Defects in CCoinBox

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 method, when you are in the “allowed” state and the new state is “empty”, the allowVend Boolean should be set to false, 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).

# Discussion of Generation of Sneak Path Test Cases

The main challenge of automatically generating the sneak path test cases from a given state machine conforming to the metamodel give 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.