Software Validation
ECSE-429
Fall 2015

Assignment 1
State-Based Testing

Tuan-Kiet Luu 260479318
Genevieve Nantel 260481768
Group 23

1- How to run our source code:

- In Eclipse, open our project and navigate to the file called "RoundTripPath.java".
- At the beginning of the RoundTripPath class, there is a static string called "StateMachineXML". This is where you enter the name of the xml file you would like to input. For example, for the "ccoinbox.xml" file, put the following:

```
static String StateMachineXML = "ccoinbox.xml";
```

- Click the green "Run" button to run "RoundTripPath.java".

- This will generate a file called "GeneratedTestCCoinBox.java". This is the generated test class without the manual changes.

- To run the tests without the manual changes, simply click the green run (Run As.. JUnit) button when you are on the "GeneratedTestCCoinBox.java" file to run the JUnit tests. This should result in 10 tests where 3 out of 10 fails.

To test with the manual changes, open the file called "TestCCoinBox.java" and click the green run button (Run as .. JUnit). This will run the tests where the manual changes were added and they should now all pass. There are comments where the manual changes were added. Also, we added extra assertions to check that the values are well set.

2- Discussion of manual changes:

When we ran the initial generated test file "GeneratedTestCCoinBox.java" we noticed that we had three failing tests:

Conformance test 6:
(empty) —> addQtr —> (notAllowed)—>addQtr—> (allowed)—>addQtr—>(allowed)

Conformance test 8:
(empty) —> addQtr —> (notAllowed)—>addQtr—> (allowed)—> vend—>(notAllowed)

Conformance test 9:
(empty) —> addQtr —> (notAllowed)—>addQtr—> (allowed)—> vend—>(allowed)

Because our algorithm does not check the conditions, we see that some of the transitions that have a specific conditions fail. For example:

- Conformance test 8 is when we are in state "Allowed" and we call vend() with curQtrs== 3 then we go to state "notAllowed". However, with our tree, when we go through that transition, our curQtrs was only equal to 2. We therefore added manually an addQtr to have our curQtrs==3.

- Conformance test 9 is when we are in state “Allowed” and we call vend() with curQtrs> 3 then we go to state “allowed”. However, with our tree, when we go through that transition, our curQtrs was only equal to 2. We therefore added manually two time addQtr to have our curQtrs==4.
- Conformance test 6, is the result of a mistake found in the CCoinbox.java file for which we also added an addQtr as explained in section 3 on this report.

3- Discussion of any defects in CCoinBox

In the CCoinBox.java file there seems to be a mistake as we discovered through our conformance test 6. If you are in the state “Allowed” and you add another quarter, you should be able to stay in state “Allowed” as described in figure 2. State machine of the CCoinBox found in the assignment description. However, in the addQtr function, there is a case that if you are in state “Allowed” and you add another quarter then you are set to state “notAllowed” as you can see in the snippet of code found below:

```
case allowed:
        curQtrs = curQtrs + 1;
        setState(State.notAllowed);
        wasEventProcessed = true;
        break;
      default:
```

For the returnQtrs() and reset() functions, they both set the state to empty (“setState(State.empty);”). However, they do not change the property “AllowVend” to false. Logically, when you reset, everything should be reinitialized (totalQtrs = 0, curQtrs = 0, and allowVend = false). Similarly, when you have curQtrs=2, and you vend(), it should also put allowVend to false, since you have no more coins (curQtrs=0).

4- Discussion of generation of sneak path test cases

The main challenge to automate the generation of the sneak paths test is the complexity. Indeed, sneak paths testing requires the user to test all illegal transitions that is not specified for the state. Therefore, if we want to automate the generation of the the sneak path tests with the metamodel given in Figure 1, we would have to test all transitions on every given state, and come up with an error message for each cases. Furthermore, we have seen an example in class where we had a sneak path test that would test the CCoinBox in state empty, and use pop(). In that case, we can test the expected result which is stack.getNbElements() would give 0 again. However, if we are given a metamodel like Figure 1, we cannot always fully understand the logic of every state and transition, and therefore test the illegal transitions by checking the resulting conditions and actions. Also, a sneak path is possible for each unspecified transition and this would increase the complexity of the analysis.