

State-Based Testing

NEIL EDELMAN, McGill University (110121860)

CASIMIR DÉSARMEAUX, McGill University (260467441)

Assignment 1 report.

Categories and Subject Descriptors: D.1.3 [Software]: Concurrent Programming

General Terms: Software

Additional Key Words and Phrases: Subject

Reference Format:

N. Edelman, C. Désarmeaux. *State-Based Testing* (Fall 2015.)

304-429 (Section 001) A1

Instructor:	Prof. G. Mussbacher
Course:	ECSE 429 – Introduction to Software Quality Assurance
Date:	2015-10-05

Contents

1	Source Code	1
1.1	Source code	1
1.2	Generated code	1
2	Complete Test Class	1
3	Report	1
3.1	Description	1
3.2	Manual changes	1
3.3	Defects	2
3.4	Conformance	2

1. SOURCE CODE

1.1. Source code

The source code that generates the test class for the CCoinBox example given the state machine definition and implementation of the state machine.

The code is in the package `ca.mcgill.ecse429.conformancetest.nplus`.

1.2. Generated code

The result of the test class generation for the CCoinBox example without any manual changes after generation. This class must be called `GeneratedTestCCoinBox.java` and saved in the same package as the implementation of the state machine.

The generated code is in `test`, following the same directory structure.

2. COMPLETE TEST CLASS

The complete test class for the CCoinBox example with additional code added manually as needed to fully test the CCoinBox state machine based on the N+ Test Strategy (conformance tests only). This class must be called `TestCCoinBox.java` and saved in the same package as the implementation of the state machine. Any manual changes have to be clearly identified in the complete test class. Any complete test class that cannot be executed as a JUnit test will result in a mark of 0 for this part.

The generated code is in `test`, following the same directory structure.

3. REPORT

3.1. Description

Describe how to run your source code to generate the test class for a given state machine (xml file) and corresponding implementation of the state machine. This description should work for the CCoinBox example but also for the unknown state machine and its implementation.

From the root of the project,

```
java -cp bin:lib/xmlpull-1.1.3.1.jar:lib/xpp3_min-1.1.4c.jar:lib/xstream-1.4.7.jar
ca/mcgill/ecse429/conformancetest/nplus/Nplus <xmlfile>
```

For example,

```
java -cp bin:lib/xmlpull-1.1.3.1.jar:lib/xpp3_min-1.1.4c.jar:lib/xstream-1.4.7.jar
ca/mcgill/ecse429/conformancetest/nplus/Nplus ccoinbox.xml >
```

```
test/ca/mcgill/ecse429/conformancetest/ccoinbox/GeneratedTestCCoinBox.java
```

To add this to the Makefile, add another entry to the variable XML,

```
<xml file>:<path to java>
```

Eg,

```
legislation.xml:ca/mcgill/ecse429/conformancetest/legislation/Legislation.java
```

You will have to set `JUNITHOME` in your environment.

You can also work with Eclipse.

3.2. Manual changes

In a few paragraphs, discuss which manual changes you had to make to the generated test class to get the complete test class and why you had to make those manual changes instead of automatically generating the test code.

In order to make a complete test for the CCoinBox example, some manual code had to be added to the test class automatically generated. For every test case generated by the automated code, an instance of the test object was already created. At every creation of test objects, the constructor was first tested, which consists in making sure that the totalQuarters and curQuarters variables were set to 0, and that the allowVend boolean was set to false. As the exploration of the path carries on, the variables that were modified after every event applied on the test object. For the CCoinBox example, the incrementation of the current quarters after the addQtr() event was tested, as well as the decrease of the current quarters by 2 and increase of totalQuarters by 2 after every vend() event. Similarly, after every reset() and returnQtrs() event applied, the setting of current quarters back to 0 was also tested.

3.3. Defects

In one paragraph, describe whether you found any defects in the implementation of the CCoinBox example. For each found defect, describe how you fixed it.

Some test cases generated by the the Nplus.java class also lead to failure. Indeed, when the CCoinBox would reach the state allowed, it would then lead to 6 possibilities in the round trip path tree, but 3 of the 6 cases would fail.

- Allowed \rightarrow addQtrs() \rightarrow allowed: the condition where a state is allowed and the addQtr() event is applied should lead to another allowed state, however the CCoinBox.java implementation changes the state to notAllowed. To fix this issue, the CCoinBox.java has been modified so the it would keep the state to allowed
- Allowed \rightarrow vend() \rightarrow allowed: this is because the round trip path tree goes through 2 steps before reaching this one (empty \rightarrow notAllowed \rightarrow allowed). In these 2 steps, the addQtr() event is used as a transition, meaning that the count of current quarters when the allowed state is reached is 2. However, in order to go from an allowed state to another allowed state after the vend() event is applied, the current quarter count has to be 4. This cannot be done automatically, and has been fixed by using the setCurQtr(int) method, setting the current quarter count to 4 before the vend() event is applied
- Allowed \rightarrow vend() \rightarrow notAllowed: this is the same scenario as above, with one difference. In order to reach a notAllowed state, the current quarter count has to be 3, so that 1 quarter remains, which is conform the notAllowed state.

3.4. Conformance

In one paragraph, discuss what would be the main challenge to automate the generation of sneak path test cases from a given state machine conforming to the metamodel in Figure 1.

The above scenarios had to be fixed manually, because the generated code respects the round trip path tree. In these cases, the code implementing the depth first search to make the round trip path tree could not catch these flaws.

As mentionned above, there was a defect found in the implementation of the CCoinBox example, which concerns the addQtr() method. The defect came from the case when the state is allowed, and when a quarter is added, it is changed to not allowed. This has been fixed by simply changing the outcome of the case: when the state is allowed and a quarter is added, the state remains allowed.

Received 2015-10-05; revised 2015-10-05; accepted 2015-10-05