**Foundations of Software Testing, 2E**

**Test Generation Using the W Method**

Java Programs for the Algorithms in Chapter 5: Test Generation From Finite State Models

Updated: August 4, 2013

Table of Contents

1. How to generate a test set? 2

2. Encoding an FSM 3

3. Folders and files 4

4. Debugging switches 4

5. Executing an FSM 5

6. Bugs 5

# How to generate a test set?

1. Compile and run WMethod.java in your favorite IDE.
2. Enter the file name when prompted. This file must contain the FSM in the format described later.
3. The program will then display the FSM, the W-set, and the tests generated. A sample interaction follows.

run WMethod  
Enter filename: [TestFSMs/mathur-test]

FSM input from: TestFSMs/mathur-test

States: 5

Edges 10

Input alphabet:

a

b

Output alphabet:

0

1

From Input/Output To

1 a/0 1

1 b/1 4

2 b/1 5

2 a/0 1

3 a/0 5

3 b/1 1

4 b/1 4

4 a/1 3

5 a/1 2

5 b/1 5

Transition cover set (P). 11 entries.

Empty a b ba baa baaa baaaa baaab baab bab bb

W Set. 4 entries.

a aa aaa baaa

Number of Test Cases: 29

Test cases: [a, aa, aaa, aaaa, abaaa, ba, baa, baaa, baaaa, baaaaa, baaaaaa, baaaaaaa, baaaabaaa, baaaba, baaabaa, baaabaaa, baaabbaaa, baaba, baabaa, baabaaa, baabbaaa, baba, babaa, babaaa, babbaaa, bba, bbaa, bbaaa, bbbaaa]

# Encoding an FSM

Each FSM needs to be encoded and saved in a plain text file. The test generation algorithm reads the FSM from a specified file and encodes it for internal use. The format of an FSM file is as follows.

Data for each edge occupies one line. Lines are separated by CRLF. Each line contains, in order, the label of the source state, the label of the destination state, and the edge label. Edge labels are treated internally as strings of characters. While the state labels must be positive integers. The edge label must be of the form x/y where x denotes the input to the FSM and y the corresponding output.

**Example FSM**:

**Example input: FSM coding for Figure 5.13 on page 235 of the textbook (reproduced above).**

1 1 a/0

1 4 b/1

2 5 b/1

2 1 a/0

3 1 b/1

3 5 a/0

4 4 b/1

4 3 a/1

5 5 b/1

5 2 a/1

Several FSMs are located in the W-Code/TestFSMs folder.

# Folders and files

|  |  |
| --- | --- |
| **Class** | **Purpose** |
| Edge.java | Denotes an edge in the FSM. |
| EdgeSet.java | Collection of edges each of type Edge. An EdgeSet object is is a collection of Edge objects denoting edges going out of a State object. |
| InvalidEdgeException.java | This exception is raised when an invalid edge is detected while inputting an FSM. |
| LabelAndEdgeSet | This object denotes the label of an edge, e.g. a/0, denotes input a and output 0. |
| NoNextStateException.java | This exception is raised when there is no destination state specified for an edge. |
| pTable.java | Denotes a P-table that arises during the construction of k-equivalence partitions (see Section 5.5.5 of the textbook). |
| pTableEntry.java | Denotes one entry in a pTable object. |
| pTableManager.java | Determines the W-set using k-equivalence partitions. |
| State.java | Defines the state of an FSM. |
| TestingTree.java | Denotes the testing tree object. |
| TestingTreeNode.java | Denotes a node in the testing tree. |
| Utilities.java | Contains several utility methods. |
| WMethod | Contains the main() method and some supporting methods. |

# Debugging switches

Following is a list of switches and purpose. All switches are located in Utiliyties.java. These could be turned ON/OFF to enable/disable printing of various intermediate data generated during the execution of any of the three class test sequence generation algorithms.

|  |  |
| --- | --- |
| **Switch** | **Set this to true to …..** |
| fsmPrintSw | Debug cycle breaking. |
| pTableDebugSw | Print P-tables as they are generated. |
| testingTreeDebugSw | Print information as a testing tree is built. |
| transitionCoverSetBebugSw | Print information while the transition cover set is being computed. |
| fsmCreationDebugSw | Print states, edges and labels while FSM is input. |

# Executing an FSM

Utilities.java contains the runFSM() method with the following signature

public static void runFSM(State [] FSM, int stateID, String input, String separator),

where FSM is an array containing the states of the FSM to be executed, stateID is the ID of the state from the execution is to begin, input is the string of input symbols to be applied to the FSM, and separator is a string that separates the elements in the input string. For example, given an FSM, the following command executes it against the input "a a b a b".

Utilities.runFSM(FSM, 1, "a a b a b", " ");

For the FSM shown earlier, the above command generates the following output.

FSM execution begins. Input: a a b a b Initial state: 1

Current state: 1

Input: a Next state: 1 Output: 0

Current state: 1

Input: a Next state: 1 Output: 0

Current state: 1

Input: b Next state: 4 Output: 1

Current state: 4

Input: a Next state: 3 Output: 1

Current state: 3

Input: b Next state: 1 Output: 1

FSM execution completed. Final state: 1

Output pattern:00111

# Bugs

The tool currently generates tests only for m=n where m is the number of states in the FSM representing the code and n that representing the design against which the code is being tested..

Report bugs to [apm@cs.purdue.edu](mailto:apm@cs.purdue.edu). However, it would be great if you can fix the bug and send me the modified set of files.

<End of document>