Assignment 5
MBT from EFSMs - Executable tests

Model-Based Testing
DIT848/GU and TDA260/Chalmers

April, 2012

1 Introduction

The goals of this assignment are for you to learn how to design and write extended simple finite state machine (EFSM) models, and use ModelJUnit for generating tests from EFSM models. In this assignment ModelJUnit will automatically generate test sequences from your EFSM model, but you need to write an adaptor that connects the model to the SUT.

2 Submitting your work

If you want to have feedback on your assignment, check with Hamid Ebadi (`hamide #@# student.chalmers.se`) on how (and when) to submit. If you want to submit, please attach a .zip or .tar.gz archive, containing your source code and .txt, .pdf or .doc file describing your answers. Please name your file with the number of assignment and your (last) name as in the following example: `ebadi_assignment05.zip`.

3 Modelling

In this assignment, you are given an implementation of a calculator (`Calculator3.java`) based on the specification in section 4. Design one or more EFSM models for this calculator assuming that it is your system under test (SUT). Each state of your EFSM will correspond to a particular state of the calculator implementation. For example, you might start with just four EFSM states as follows:

- Initial State
- Reading number
- Reading Operator
- Error state

The initial state should model the state where the calculator is ready to accept a new expresion (or subexpression). The error state represents an invalid operation or input; no further input is allowed in this state until the system is restarted.

4 Specification

This calculator is an extension of the one that is used in the first assignment where it is also possible to have paranthesized expressions. The informal specification for this calculator is explained in what follows. The calculator has the following buttons:

- digits: 0-9
- operations: +-*/
- execute: =
- reset: C
- parantheses: ()

The calculator behaves mostly like the one from previous assignments except for the following points.

- If an open parenthesis button is pressed a new subexpression is started, where both operands and operators are entered normally. When a close parenthesis is pressed. this subexpression is evaluated and then the result is stored as an operand.
- Open parenthesis is not allowed after a digit, and close parenthesis is not allowed after an operator. An operator is not allowed after an open parenthesis, and a digit is not allowed after a close parenthesis.
- If the open parenthesis button is immediately followed by a close parenthesis, it should be an error.
- It is not possible to press "=" within a subexpression.
- If an operation button is pressed while there is an operand and operation memorized, the calculator acts as if '=' was pressed before the operation button.
- If an operation button is pressed exactly after another operation button, the last one will be used for calculation and the first operation button will be ignored.

Each transition of your EFSM should model an action that can be implemented by calling one of the methods in the `Calculator` class, or a sequence of those methods.

You can design your EFSM on paper, or using a drawing program such as the drawing tools in Microsoft Word.

You may want to design more than one model, in order to test different aspects of the behaviour of the `Calculator` interface.

5 Test Generation

Generate tests from your models by using ModelJUnit. Write an implementation of your EFSMs in Java as classes that inherit from `FsmModel`. Then, write a JUnit class that instantiates these EFSMs and the SUT, write an *adaptor* to connect the model with the SUT, and try out the available ModelJUnit traversal algorithms. Report state and transition coverage for each of them.