Testing Techniques

Assignment 3

Sasja Gillissen, Martin Huijben, Martijn Terpstra

December 17, 2016

1 Model-Based Testing

1.1 MBT Tool Selection

For our tool we have chosen OSMO MBT TOOL[1]. This tool seemed to be the most appropriate from the list of tools given in the assignment desription. Firstly the tool is free software under the LGPL License. This allows us to freely use this model. Many existing tools are proprietary, requiring us to buy a license for continued use. Secondly the chosen tool, and its tests are written in Java. Our SUT is also written in Java so integration is easier than it would have been if our SUT and Testing Tool were written in different languages.

1.2 MBT Modeling

Our model is basically the same as the TorXakis model. When it starts, it repeatedly chooses an action from the following list and tests it.

Expression Expressions can be generated out of any combination of +,-,* and numbers. It is checked that the SUT returns the right solution to this expression.

1/0 TorXakis checks if the SUT really gives an error when presented with this input.

equality Our model generates two expressions and asks the SUT if the first is greater, equal or smaller than the second. It then checks if the answer given back is correct.

exit This tests that giving the 'exit' commando to the SUT returns 'bye!' and that the SUT does not respond to input afterwards.

pi Checks if the definition of pi is 3.141592653589793

[1] `https://github.com/mukatee/osmo`

e Checks if the definition of e is 2.718281828459045

function definition Generates one of three premade functions and remembers it.

function application There are two options. Either the function has already been defined or not. If the function is defined, TorXakis checks if the SUT gives the right answer. Otherwise it is checked that the SUT gives the corresponding error.

variable definition Generates one of three premade variable definitions and remembers it.

variable application Checks if the SUT gives the right answer if the variable has already been defined. Otherwise it checks if the SUT gives the corresponding error.

1.3 MBT Test Environment

1.4 MBT Testing

1.5 Deliverable

The code for both the SUT, the wrapper and the TorXakis models is publicly available at `https://github.com/Witik/CommandlineCalculator`.

To start the proxy service and SUT simply run the Sockserv.jar with as arguments the port to run on and command that you would otherwise execute. If you are in the root directory of the git project you can use the command

2 Comparison

2.1 implementation relation

Unlike TorXakis, With OSMO we have to manually implement some functionality.

In TorXakis all we have to do is describe states and state transitions. After that we can run TorXakis and check our model against our SUT.

In OSMO we have to manually generate some testing functionality. For instance TorXakis has support to automatic expression building (from simpler expressions) while with OSMO we have to write our own expression building algorithm.

2.2 support for test input generation as well as output checking

1. In TorXakis it is easier to generate input. However, this is not hard in OSMO. When generating tests, one should give OSMO a number on which it generates random steps. This number can also be used as a generator number for generating random input. Then you just do

`Random(number).nextInt(size)` to choose what to do next.

2. Both TorXakis and OSMO allow random input generation.

In TorXakis there is builtin functionality to generate input. In OSMO we have full acces to all java libraries. While OSMO does not come with such expression building tools we can and have implemented the generation in java.

Output checking is easier in OSMO.

In OSMO

Output checking is easier in OSMO. In TorXakis we had to extend our test adapter to filter the output to something TorXakis could match, since the regexp expressions TorXakis offered where not strong enough.

2.3 support for non-determinism

Our model does not need non-determinism.

2.4 method of test selection

Both models allow random test selection, redoing previous random generated test selection, and walking through the model.

2.5 modeling notation: its expressiveness and ease of use

2.6 on-line vs. off-line testing, i.e., on-the-fly vs. batch.