Testing Techniques

Assignment 3

Sasja Gillissen, Martin Huijben, Martijn Terpstra

December 19, 2016

1 Model-Based Testing

For this assignment we continued with the test cases used in the previous assignments

Category	Input	Expected Output
Exiting	`exit`	Nothing
Function Assignment	`f(x):= x + 1 ; f(7)`	`8`
Builtin Functions (1)	`sqrt(ln(log(100)))`	`0.83`
Builtin Functions (2)	`floor(0.5) + ceil(0.5) + round(0.5)`	`2`
Variable Assignment	`k = 3 ; k`	`3`
Builtin Variables	`e * pi`	`8.5397342226735656`
Expression (1)	`(((0 + 1) * 2) - 3) / 4`	`-0.2500000000000000`
Expression (2)	`1 / 0`	Error
Equality (1)	`3 = 3`	`0`
Equality (2)	`2 = 1`	`1`
Equality (3)	`2 = 3`	`-1`

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 software. 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] `https://github.com/mukatee/osmo`

1.2 MBT Modeling

Our model is basically the same as the TorXakis model. The biggest difference is that here we can easily work with float values, where in TorXakis this was too troublesome to implement. This means we can actually test the build in variables within epxressions and test the built-in functions. All built-in functions use floats.

When our model 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 OSMO 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

e Checks if the definition of e is 2.718281828459045

function definition Generates a random function and remembers it.

built-in functions The built-in functions are checked with random input.

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

variable definition Generates a random variable definition 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

With OSMO we could easily interact with our system under test.

Unlike the previous assignment we did not have to create a special interface to test our program.

Instead we add the testing environment to our SUT directly. Our testing tool, and its model are written in java so we could develop it the same way the program works.

1.4 MBT Testing

Our testing is done at compile time. When we build our SUT using `mvn install` our tests are also run and the build fails should the tests fail.

Many random tests are generated during compile time. For the sake of brevity I have include a small section of the testing output below.

```
MDPW=(1 * (((8 + 2) * 1) + 9))
8 + 2 = 10
10 * 1 = 10
10 + 9 = 19
1 * 19 = 19
added variable MDPW to memory
3 * 4 = 12
3 * 12 = 36
5 * 9 = 45
5 * 45 = 225
36 + 225 = 261
3 + 7 = 10
8 * 9 = 72
10 + 72 = 82
3 * 6 = 18
4 + 18 = 22
82 + 22 = 104
261 - 104 = 157
3 - 7 = -4
9 + 7 = 16
-4 * 16 = -64
7 - 6 = 1
2 + 3 = 5
1 - 5 = -4
-64 * -4 = 256
3 + 5 = 8
9 - 8 = 1
1 * 9 = 9
```

```
256 - 9 = 247
[157 = 247] = -1
hUz=((0 + 4) + ((2 + (5 + 7)) * 0))
0 + 4 = 4
5 + 7 = 12
2 + 12 = 14
14 * 0 = 0
4 + 0 = 4
added variable hUz to memory
added function XIf to memory
XIf() = (8 + (((4 * 5) + 3) + ((5 - 1) + (8 - 1))))
```

In the end we could test all our use cases successfully and even more rigorously than we did with TorXakis in the previous assignment.

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

Both tools miss functionality that the other tool has. However, the functionality that OSMO misses is easier to overcome.

For example, describing states and state transitions in OSMO is tedious, but our SUT does not have complicated state transitions. Another example, OSMO does not have automatic expression building, but this was easy to implement.

The problems in TorXakis are difficulties when extracting input (regexp is just not strong enough), difficulties connecting to the SUT and difficulties working with floats. All three of them heavily limited the ease of implementation.

2.2 support for test input generation as well as output checking

Both TorXakis and OSMO allow random input generation. In TorXakis it is easier to generate input. However, this is not hard in OSMO. In TorXakis there is builtin functionality to generate input.

While OSMO does not come with such expression building tools we can and have implemented the generation in java. 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.

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 SUT is fully deterministic. Therefore 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

The models written with OSMO are java programs. While the syntax may be different from TorXakis it was significantly easier to setup because we were already familiar with java.

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

OSMO does off-line batch testing, just like TorXakis. In OSMO we define a model and let OSMO compare our model with our SUT.

We do not need anything more complicated. Firstly because our system is lightweight enough that we can easily have a session running just for testing. On-line testing would not allow us to test anything we could not easily test offline. Secondly because our system does not need to operate in real time. Restarting our system just to test it does not cause any problems. Lastly we assume our SUT is deterministic and only depends on user input.