Model-based testing of timed systems with *DIVERSITY*

12/12/2016

Introduction Model-based testing (MBT) uses models of systems for the derivation of test cases. We use timed automata as models that manipulate variables to abstractly denote system states (we call them data variables) and variables to capture timing constraints (we call them clocks) on system executions. The objective is to get familiar with model-based testing using symbolic techniques. Symbolic execution is a technique used here to analyze model behavior. It works by using symbols, instead of concrete values. Hence, variables during the execution are assigned with expressions based on those symbols rather than concrete values.

This exercise sheet is divided in two parts.

- **Part I Symbolic execution** provides an initiation to the symbolic execution. Possible symbolic executions are explored: some coverage criteria are investigated to achieve structural coverage.

DIVERSITY module	exploration strategy	coverage
symbolic execution	BFS/HIT-OR-JUMP	transition, state

- **Part II Off-line testing** is about offline testing process where test data are precomputed from the model (which contrasts with online testing where the test data are generated while executing the test). In offline testing, real executions of the System Under Test (SUT) are post-processed on the model to deliver a verdict.

DIVERSITY module	algorithm	verdicts
testing	off-line	PASS, FAIL, Inconclusive, WeakPass

DIVERSITY is a symbolic automatic analysis and testing tool that integrates different sat-solvers. DIVERSITY generates a symbolic tree by symbolically executing a system specification and offers numerous exploration/coverage strategies together with test data generation capabilities. DIVERSITY supports offline testing.

Running example: Thermostat system We consider a small example of a thermostat system which automatically controls heating or/and cooling equipment. The objective is to maintain the temperature within a certain range $T_{min} \leq t \leq T_{max}$. The system consists of two communicating components as follows:

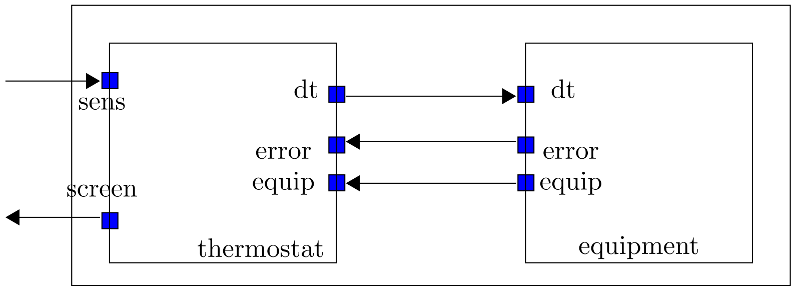


Figure 1: Thermostat system architecture

- the thermostat control component receives from the environment (here a sensor) a measure of the ambient temperature through the channel *sens*. When this temperature is not within the range defined by T_{min} and T_{max}, the control asks the equipment to adjust the temperature: it transmits through channel *dt* the difference between the sensed temperature and the exceeded threshold (either $T_{min} - t$ or $T_{max} - t$). The thermostat control also displays on a screen the heating/cooling/off status of the equipment.
- the equipment component heats/cools at ± 4 degrees/time unit. It then notifies the thermostat either of its successful action on channel *equip* or of a failure on channel *error*.

Part I
Symbolic framework

1 The model

Consider the automaton of the thermostat control component in Figure 2.

1. [**Understanding the example**] The automaton has two operating modes: cooling and heating. Given that we want to keep the temperature between 20 and 25 degrees ($T_{min} = 20, T_{max} = 25$), identify the transitions of the equipment TIOSTS which are associated with the cooling (respectively heating) mode and explain the purpose of each of them. *Hint: the sensed temperature is stored in the variable t of the thermostat TIOSTS.*

   Notes: Consider now the TIOSTS of the equipment component in Figure 3. In fact, the thermostat system is modeled by means of *two* TIOSTSs

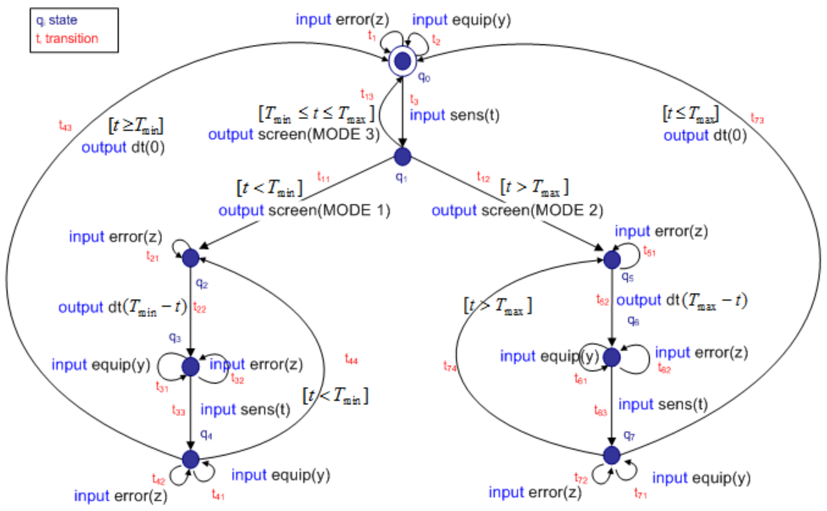


Figure 2: TIOSTS of the thermostat component

which communicate through shared channels which are internal to the system as illustrated in Figure 1. For instance, consider the action $dt!T_{max} - t$ of the thermostat: this action can be fired if and only if the equipment is ready to receive it thanks to the action $dt?x$, and in this case the value of x is immediately assigned to $T_{max} - t$. Transitions labeled by those two actions are thus considered to be executed at the same time.

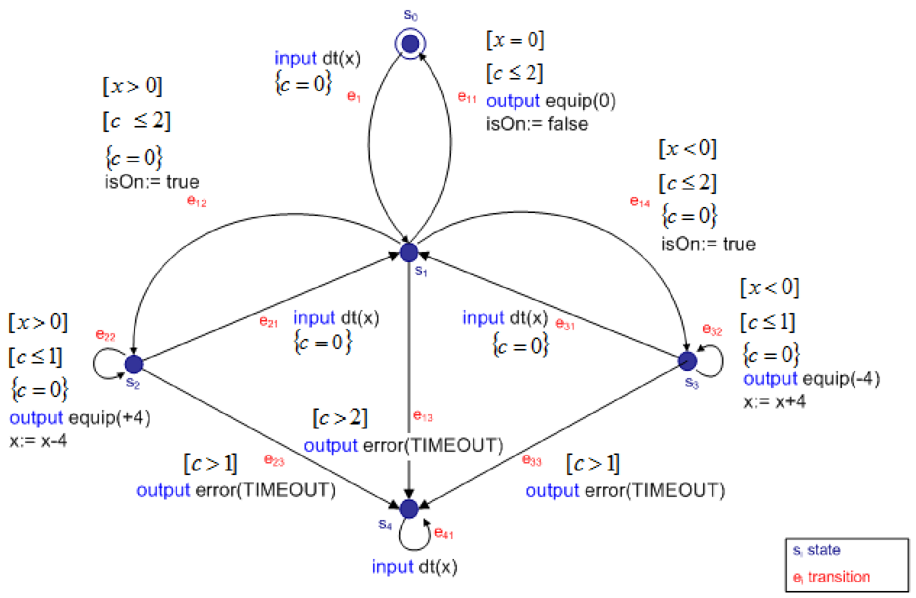


Figure 3: TIOSTS of the equipment component

2. [**Coding in DIVERSITY**] Complete the implementation of the system given in the file `Diversity/Specification/thermostatSystem.xlia`[1] by adding the transitions missing from the equipment TIOSTS.

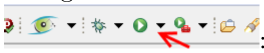
   Recall that $\{c\}$ is an action that resets clock c, in Diversity this is executed by the code `c = 0;` (Timed) guards are denoted as parameters of the command (`tguard`) `guard`.

2 Exercise: Symbolic execution

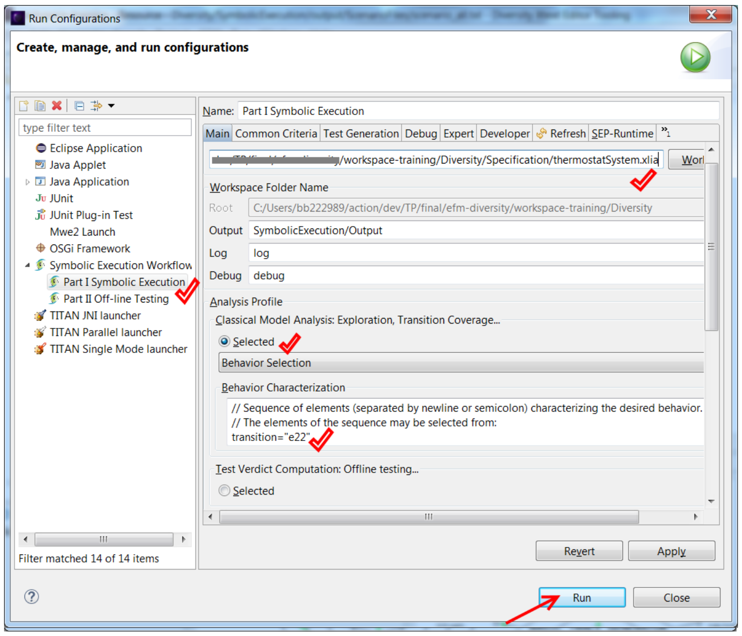
The idea is to compute all possibles executions of the thermostat system as a symbolic tree using Diversity. Symbolic execution trees may be infinite structures (due to some repetitive behavior in the specification). In this exercise, we use the so-called *Hit-Or-Jump* coverage heuristic to build a symbolic tree (in fact reduced to a path) covering a declared sequence of (possibly non consecutive) transitions. First, a symbolic tree is computed in width for a given maximal depth. Once it is computed, an analysis is realized to study whether or not a part of the sequence has been covered in this tree:

- if some non empty prefixes of the sequence have been covered, the tool identifies the set of paths that covered the greatest prefix, and chooses one among them in a random way,
- else the tool chooses one path in a random way.

Once a path is chosen all other paths are erased and the whole process starts again from the last symbolic state of the path (*i.e.* the target state of the last symbolic transition of the path) until the sequence is fully covered. Of course nothing ensures that the strategy will succeed to cover the whole sequence of transitions, therefore a stopping criteria is also defined (in terms of maximal number of computed symbolic states in our case).

1. [**Test selection**] As a first step, the objective is to cover the heating mode of the equipment. For this purpose, you need to run the configuration of Diversity that has been parametrized (with the Hit-Or-Jump coverage) in order to cover the transition of the equipment TIOSTS $e_{22} = (s_2, c \leq 1, x > 0, \{c\}, equip!4, x := x - 4, s_2)$. First click on the small black triangle button in the upper menu bar indicated by the red arrow here : a menu is displayed. In this menu, choose "Run Configurations...". You will obtain the following dialog window:

[1] *"xLIA" which stands for eXecutable Language for Interaction Assemblage* is the entry language of the DIVERSITY tool.



Run Configurations [redacted]

Create, manage, and run configurations

type filter text

- Eclipse Application
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Mwe2 Launch
- OSGi Framework
- Symbolic Execution Workflow
  - Part I Symbolic Execution
  - Part II Off-line Testing
- TITAN JNI launcher
- TITAN Parallel launcher
- TITAN Single Mode launcher

Filter matched 14 of 14 items

Name: Part I Symbolic Execution

Main | Common Criteria | Test Generation | Debug | Expert | Developer | Refresh | SEP-Runtime | »1

[redacted]/workspace-training/Diversity/Specification/thermostatSystem.xlia

Workspace Folder Name

Root: C:/Users/bb222989/action/dev/TP/final/efm-diversity/workspace-training/Diversity

Output: SymbolicExecution/Output

Log: log

Debug: debug

Analysis Profile

Classical Model Analysis: Exploration, Transition Coverage...

[✓ Selected]

Behavior Selection

Behavior Characterization

```
// Sequence of elements (separated by newline or semicolon) characterizing the desired behavior.
// The elements of the sequence may be selected from:
transition="e22"
```

Test Verdict Computation: Offline testing...

Selected

Finally, click on "Run" (be sure that the launch configuration "Part I Symbolic Execution" is active and parametrized to cover the transition e_{22}). The goal is now to identify the succession of transitions covered by the tool, to identify the corresponding symbolic trace and the path conditions over time and data. To reach that goal, analyze the symbolic tree. It is generated as a file in the `gv` graphical format here

`Diversity/SymbolicExecution/Output/symbex_output.gv` - open it by clicking right on this file and successively choosing "Symbolic Execution Workflow" then "GraphViz" (Before, it may be necessary to refresh the folder `Diversity/SymbolicExecution/` by using the F5 keyboard key).

Now, choose the adequate transition (and change the parameter the launch configuration "Part I Symbolic Execution" accordingly) in order to cover:

- the cooling mode of the equipment,
- a TIMEOUT error of the equipment,

and identify again the corresponding succession of transitions covered by the tool, the corresponding symbolic trace and the path conditions over time and data.

2. [**Test data generation**] Symbolic paths represent classes of behaviors. To generate a concrete one, Diversity calls SAT-solvers.

   The launch configuration "Part I Symbolic Execution" has been already configured for that (in Tab "Trace Generation"). Look at the generated trace and deduce how it is structured in terms of actions and durations (file `scenario_all.txt` in `Diversity/SymbolicExecution/Output/ScenarioFiles`).

Part II

Off-line testing

In black box testing, the behavior of system under test (SUT) can only be observed by building traces while interacting with it. The test harness illustrated in Figure 4 allows to send test data on the input channel *sens* and observe, besides the output channel *screen*, all the internal channels of the system *dt*, *equip*, and *error*.

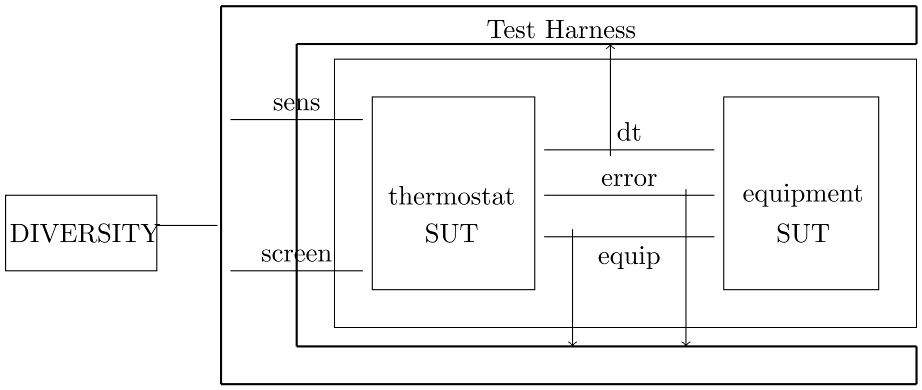


Figure 4: Thermostat system test architecture

The off-line testing process consists in three steps:

(1) first, test input sequences are extracted from paths of the symbolic execution tree (those traces are obtained as in exercise 2 but they contain only input actions and durations in between);

(2) test input sequences are submitted to SUT, i.e. test execution, which produces output sequences that are merged with input sequences to form input-output traces;

(3) resulting traces are analyzed in order to provide verdicts.

3 Exercise: Input sequence selection

Compute a test input sequence which covers the heating mode (respectively cooling mode) by configuring the launch configuration "Part I Symbolic Execution" (in Tab "Test Generation", section "Observable Traces Selection"), the channel(s) to keep.

4 Exercise: Test execution

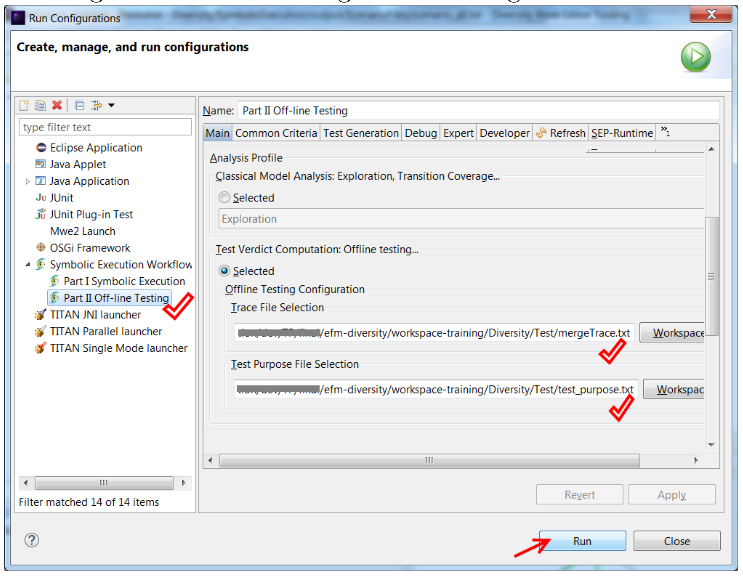
We have provided a java simulator of the thermostat system together with the test harness of Figure 4 which takes an input sequence and generates a merged input-output trace.

Submit the trace you have computed in exercise 3 for the heating mode (respectively cooling mode): is the resulting trace a valid behavior of the system, knowing that in the simulator $T_{min} = 20$ and $T_{max} = 25$?

The input trace should be entered in the file `inputTrace.txt` within the folder `Thermostatsystem_SUT`. The execution of the trace is performed by clicking on the button . The resulting trace will be computed in the `mergeTrace.txt` file in the same repertory. It may be necessary to refresh the file by using the F5 keyboard key.

5 Exercise: Verdict computation

The testing data (test purpose, verdict computation, . . .) are located in the repertory Test. Two data have to be initialized : the trace to be analyzed (in the mergeTrace.txt file) and the transition targeted by the testing process, the last transition of the test purpose to be reached, in the test_purpose.txt file. The computation of the verdict is obtained the launch configuration "Part II Off-line Testing" and the verdict is given on the right bottom window.



Name: Part II Off-line Testing

Analysis Profile

Classical Model Analysis: Exploration, Transition Coverage...

Selected

Exploration

Test Verdict Computation: Offline testing...

Selected

Offline Testing Configuration

Trace File Selection

[redacted]/efm-diversity/workspace-training/Diversity/Test/mergeTrace.txt

Test Purpose File Selection

[redacted]/efm-diversity/workspace-training/Diversity/Test/test_purpose.txt

1. Consider the following collected trace of the SUT simulating the thermostat system:

```
delta = 1
INPUT sens(0)
delta = 1
OUTPUT screen(1)
delta = 1
OUTPUT dt(20)
delta = 2
OUTPUT equip(4)
```

Use Diversity to show that this is a valid behavior of SUT for $T_{min} = 20$ and $T_{max} = 25$. *Hint: The objective is to test the heating functionality, choose the transition* $e_{22} = (s_2, c \leq 1, x > 0, \{c\}, equip!4, x := x - 4, s_2)$ *as a test purpose in order to obtain a verdict* `PASS`.

2. Consider now the following collected trace of an SUT implementing the thermostat system:

```
delta = 1
INPUT sens(26)
delta = 1
OUTPUT screen(2)
delta = 1
OUTPUT dt(-1)
delta = 2
OUTPUT equip(4)
```

   Use Diversity to show that this test of SUT reveals a failure for $T_{min} = 20$ and $T_{max} = 25$. *Hint: The objective is to test the cooling functionality, choose the transition* $e_{32} = (s_3, c \leq 1, x < 0, \{c\}, equip! - 4, x := x - 4, s_3)$ *as a test purpose in order to obtain a verdict* `FAIL`.

3. Same question for the following trace of SUT:

```
delta = 1
INPUT sens(26)
delta = 1
OUTPUT screen(2)
delta = 1
OUTPUT dt(-1)
delta = 2
OUTPUT equip(-4)
delta = 1
INPUT sens(19)
delta = 1
OUTPUT screen(1)
delta = 1
OUTPUT dt(1)
delta = 2
OUTPUT equip(4)
```

   Deduce an interesting property of the thermostat system. *Hint: Explain the failure despite the proper functioning of the cooling/heating mode.*

4. Suggest a faulty SUT trace due to timing noncompliance.

5. Suggest SUT traces which cover the verdicts `INCONCi`, `INCONCr` and `WEAKPASS`.