Manual for the ltioco Tool

Lars Luthmann   Hendrik Göttmann   Malte Lochau

April 12, 2019

Contents

1 About

The purpose of this tool is to provide a demonstration of the results obtained in the paper *On Input/Output Conformance Testing of Live Timed Systems*. The tool implements an Input/Output Conformance (ioco) Testing relation for Timed Input/Output Automata (TIOA) generated with UPPAAL. Of course, it is able to read in every TIOA generated with UPPAAL. Thus, the user can select a TIOA as implementation and one as specification. The tool then automatically generates the corresponding zone graphs which can be investigated by the user. With the zone graphs as a basis, the conformance testing is performed.

This tool is developed at the Real-Time Systems Lab of TU Darmstadt.

2 Installation

For installing the tool you have two possibilities. **The easier and faster approach** is to download the jar-file from the website[1]. In this case, you only need *Java Runtime Environment* (JRE) at version 1.8 or higher. If you do not already have a JRE installed, please visit `https://java.com` and follow the instructions. The other approach is to compile the tool from the source code. This procedure is explained in detail in the remainder of this section.

2.1 Compiling From Source on Linux

In this part it is described how to compile the tool from source with the help of the build system Gradle.

2.1.1 Installation Requirements

Before you are able to compile the source code, you have to install a few required programs first. These programs can be easily installed by following the instructions on their corresponding websites.

- Java Development Kit (version 8 or higher)[2]
- Git (`https://git-scm.com/downloads`)
- Gradle (`https://gradle.org/install`)

After the successful installation of the mentioned programs, you can proceed with the next step.

[1] `http://www.es.tu-darmstadt.de/es/team/lars-luthmann/icst18`
[2] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

2.1.2 Checking Out the Git Repository

Check out the git repository like explained below.

1. Open a terminal.
2. Navigate to the folder where you want to place the source files.
3. Type in the command
   `git clone --recursive https://github.com/hendrikgoettmann/bachelorthesis.git your-folder`
   and press enter. Replace `your-folder` with a folder name on your system.

The usage of the parameter `--recursive` is crucial because the given repository contains submodules which must be initialized.

2.1.3 Compiling and Running the Project Using Gradle

After checking out the repository, you can compile and run the project with Gradle. Simply open a terminal, navigate to the root directory of the project and execute `gradle run`.

1. Open a terminal.
2. Navigate to the root directory of the project. Here you find a file named *build.gradle*.
3. Type in `gradle run` and press enter.

Gradle will then resolve all dependencies automatically and run the tool directly afterwards. If you want to run the supplied unit tests, simply type `gradle test` into the terminal.

2.1.4 Eclipse Import

For importing the project into Eclipse please perform the following steps:

1. Open a terminal.
2. Navigate to the root directory of the project. Here you find a file named *build.gradle*.
3. Execute `gradle eclipse` and open Eclipse afterwards.
4. Go to *File* → *Import* → *Existing Projects into Workspace*.
5. Choose under *Select root directory* "project-root/juppaal" and click *Finish*.
6. Repeat step 4, choose "project-root" as *root directory* this time and click *Finish*.

It is important that both projects are imported into Eclipse. Otherwise, Eclipse will produce countless error messages.

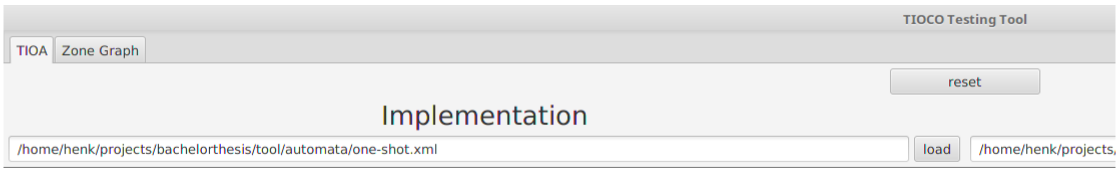

Figure 1: TIOA Selection

3 Usage

To run this tool, it is required that you have two TIOA created with UPPAAL (one represents the implementation and the other one the specification). If you do not have a TIOA at hand, you can take one from the provided examples in the *automata* directory.

3.1 Choosing Implementation and Specification

After starting the tool, the first thing to do is to choose the implementation and the specification. This is done via the respective *load* buttons as depicted in Figure 1. After clicking the button, a dialog will pop up. Here, you can select the XML file of the TIOA. The chosen file is shown in the text field next to the *load* button. The actual automaton is displayed in the window below the text field (not depicted in Figure 1).

3.2 Analyzing the Zone Graph

The next step after loading the TIOA is to switch to the *Zone Graph* tab where the zone graphs are displayed. It is possible to inspect the graphs by zooming in and out (mouse wheel), or by dragging symbolic states around with the mouse. Additionally you can save a zone graph as image by clicking the right mouse button on the graph.

3.3 Testing with tioco

Besides generating zone graphs from TIOA, the automated testing in terms of $\mathbf{ltioco}_{\mathrm{z}}$ is the second main feature of this tool. For testing an implementation against the corresponding specification, three modes are available: *Manual*, *Depth-First Search* and *Random*. They all have in common, that after each action / delay of a trace, the sets of output actions are compared. If the set of output actions of the implementation is not a subset of the set of output actions of the specification, an alert will pop up to show that $\mathbf{ltioco}_{\mathrm{z}}$ is not satisfied. The three modes differ in the way in which zone graphs are traversed. This is explained in the following.

3.3.1 Manual

The *manual* mode delegates the decision, which traces are tested, to the user. Via the respective input fields, the user can select the next action / delay which will be simulated (depicted in Figure 2). Only actions, which are reachable without any delay

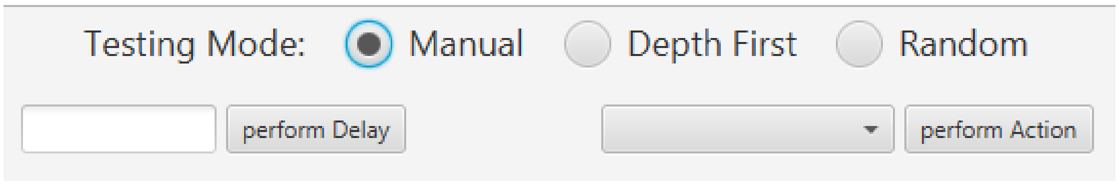

Figure 2: Manual Mode

from the current state of the specification, can be selected. Delays, however, may be chosen arbitrarily.

3.3.2 Random

The *random* mode generates random traces from the specification. At each step it determines which delays and actions are allowed. From all possible alternatives one is chosen arbitrarily. This action / delay is displayed on the button labeled with *Next step*. By clicking the button the respective action / delay is executed.

3.3.3 Depth-First Search

The *depth-first search* mode generates traces until all edges of the zone graph of the specification are visited at least once. The traces are chosen in a deterministic manner, such that the algorithm will always produce the same traces for the same specification.