School of Innovation, Design and Engineering

5. Modeling and Verification of Finite State Machines

This document provides the laboratory instructions for the first session on Model-based testing. The objective of the lab is (i) for the course participants to get acquainted with a modeling and verification tool and (ii) to get experience of modeling and verifying properties of finite state machines.

Exercise 1 – Modeling and verification of an untimed system

In this first exercise, we shall work with a version of the alternating bit protocol - a well-known communication protocol, based on the sliding window technique. It provides reliable transfer on a lossy and noisy channel.

The exercises are performed using the UPPAAL tool and a set of pre-implemented models.

Please use the following link section to install UPPAAL 4.0:

http://www.it.uu.se/research/group/darts/uppaal/download.shtml

To find out more about UPPAAL, read this short introduction:

http://www.it.uu.se/research/group/darts/uppaal/about.shtml#introduction

Alternating Bit Protocol description

The model of alternating bit protocol consists of three entities: Sender, Medium and Receiver. The Sender sends messages and receives acknowledgments, Medium retransmits messages and acknowledgments, and Receiver receives messages and transmits acknowledgments.

Since the Medium can lose messages, to ensure that the messages have been properly transmitted, a bit is added to each message. The Sender will try to retransmit message with the bit `0` as long as it does not receive corresponding acknowledgment containing the bit `0`. Similarly for bit `1`.

The Sender sends messages `s0` or `s1` to the Medium which is supposed to deliver them to the Receiver. After sending the message `s0`, the Sender expects to receive `sack0` acknowledgment from the Receiver. It will proceed with the sending of the next message only once the acknowledgment `sack0` is received. If for some certain amount of time an acknowledgment is not received, it will try resending the message `s0`. When it receives `sack0`, the Sender will go on to send the message `s1` and wait for the corresponding acknowledgment `rack1` while possibly resending the message `s1` if timeout occurs or wrong acknowledgment is received. After successfully getting the acknowledgment for `s1`, it will start sending `s0` again.

The receiver starts by waiting to receive one of the messages `r0` or `r1` from the Medium (a forwarded `s0` or `s1`) and, after receiving it, sends the acknowledgment `rack0` or `rack1` to the Medium, corresponding to the incoming message. Internally, the Receiver is keeping track of whether it is time to receive message `r0` or `r1` to be able to discard duplicate messages cause by a faulty channel.

The Medium receives the messages `s0` and `s1` and sometimes forwards them as `r0` and `r1` to the Receiver. It also receives acknowledgments `rack0` and `rack1` from the Receiver and occasionally forwards them to the Sender as `sack0` and `sack1`, respectively. If it does not forward the messages or acknowledgments, it receives them ignores them.

Web:
http://www.es.mdh.se/staff/349-Eduard_Paul_Enoiu

E-mail: eduard.paul.enoiu@mdh.se

Tasks:

1. Unzip the pre-implemented models from the following link: http://www.promptedu.se/promptwp/wp-content/uploads/2014/10/5.Lab1_.zip
2. Start the UPPAAL tool. It will open up with a default workspace. Select the "File" menu and click on the "Open System" item. A drop-down menu will appear. Select the folder in which you unzipped the pre-implemented models `and` select `abp0.xml` file. Click "Open". It will open up with a workspace, including a navigation panel and an already modeled system.
3. Take a look at the three automata that are included in the project. Navigate between them using the project tree on the left side of the user interface.
4. Note the "Declarations" part of the project tree where the synchronization channels are declared.
5. Switch to the second tab, the "Simulator" and use "Next" or "Random" button to simulate execution of the system. You can navigate forward and backwards through the simulation by selecting the appropriate step in the "Simulation Trace". On the bottom right part of the screen, you can see synchronization channel activity. It might be required to expand that part of the window.
6. Switch to the third tab, the "Verifier". You should be presented by a single verification query: `E<> sender.WaitAck`. Test the query by clicking the "Check" button.
7. Add another query by clicking on the "Insert" button in "Verifier" tab. In the middle of the window is the space to write the query itself. Use the query provided to build a query that will check whether the model is deadlock free. Test it.
8. Notice that the model provided always sends the message `s0`, and therefore does not implement the ABP protocol. Update the model so that it corresponds to the specification above while assuming that:
    a. The medium always forwards all of the messages.
    b. There are no timeouts in the system.
9. Verify that the new model is deadlock free.
10. Use the verifier to check whether every time the sender starts sending, the receiver will eventually receive the message ("leads to" `-->` operator is useful for this purpose.
11. Send the solutions containing the abp0.xml and abp0.q files to eduard.paul.enoiu@mdh.se.

Help: The commonly used operators for queries in Uppaal

p is reachable	*E<> p*
p is an invariant	*A[] p*
p leads to q	*p --> q*

Example:

System is never in state Sender.a	*A[] not Sender.a*
System can deadlock	*E<> deadlock*