

# Protocol Verification using UPPAAL: Exercises\*

Lab assistant: Alexandre David

Department of Computer Systems (room 1237, mailbox 26), Uppsala University,  
Box 325, S751 05, Uppsala. Phone: 018-18 73 41. E-mail: [adavid@docs.uu.se](mailto:adavid@docs.uu.se).

## 1 Introduction

This note describes a serie of exercises on UPPAAL. It is recommended to solve them in groups of two students. The purpose of the exercise is to demonstrate how behaviours can be modeled in CCS, and to explore relatively simple ways of automatic verification.

In the following we assume that the reader is familiar with the basics of CCS, and has access to the paper “Verifying a CSMA/CD-protocol with CCS” [Par88].

## 2 Presentation of Results

You should hand in the following results:

- A cover page with the following information:
  - complete filename (i.e. `~pelle/procalg/foo.atg`, not just `foo.atg`),
  - your name and *personal number*,
  - the name of the course, i.e. *Process Algebra Spring 2001*.
- A one page explanation (possibly on the cover page) of how you solved the problems given in this note.
- Printouts of:
  - your automata,
  - your well commented property files.

### 2.1 Results

The results will be available locally from my web page URL: <http://www.docs.uu.se/~adavid/>.

### 2.2 Tutorial

You will be given a tutorial for UPPAAL. Get familiar with it first. It presents more features that you need for this assignment but they are relevant to the course so it is better to know them. No report for this part. You can do this before or during the lab session. You can consult the course directory `/stud/docs/kurs/process_algebra`, where you will find a tutorial directory and the models of the tutorial. You are invited however to draw them by yourself.

You will find the **script** to start UPPAAL in the course directory `/stud/docs/kurs/process_algebra`.

## 3 A Simple Protocol

Read the introduction to “Verifying a CSMA/CD-protocol with CCS”. The first exercises are to model and verify a simple data link protocol, where some unrealistic assumptions will be made in order to keep the exercise simple.

The protocol specification is as follows: there are a *sender*, a *receiver* and a *medium*. The sender transmits messages (without any sequence numbers) to the medium. The medium can,

\* Based on the document *Using the Concurrency Workbench: Exercises*.

after reception of a message from the sender, do one of two things: either the message is delivered to the receiver, or it is lost. The loss of a message should be modeled with a  $\tau$ -transition (omitted action label in UPPAAL), since the loss itself does not constitute an observable event. When a message is lost a timeout in the sender will occur — model this timeout as a communication signal sent from the medium (in case of message loss) to the sender. In case of a timeout, the sender will retransmit the message. When the receiver gets a message an acknowledgement is sent to the sender. Assume that the acknowledgement is sent *directly* to the sender and not through any medium.

*Exercise 1.* Model the above protocol in UPPAAL, and validate using the simulator that it is functionally correct. You can check deadlocks by using the command line verifier `verifyta`. Export first your model to `.ta` format to do this. Check some simple properties.  $\square$

Redefine the protocol by modifying the receiver so that it reports to the layer above when a message is received. Add a test process (to model the above layer) that receives the messages from the receiver. Modify the sender to receive messages from the layer above. Add another test process that generates messages to the sender (modeling the above layer).

*Exercise 2.* Validate that the functionality of the refined protocol is correct in the simulator. Check for deadlocks and simple properties.  $\square$

Now modify the test environment so that the test sender increments a shared integer variable  $i$  whenever a message is sent. Modify the test receiver so that it decrements the variable  $i$  whenever a message is received.

*Exercise 3.* Validate the protocol as before and verify that the value of the variable  $i$  is always one or zero, i.e.:  $A[] ( i==0 \text{ or } i==1 )$ .  $\square$

*Exercise 4.* What happens if the protocol is redefined in the following way: acknowledgements are not sent at all from the receiver to the sender? What happens if additionally the sender never expects acknowledgements? Validate, verify and describe in words the behaviour of the resulting protocol. Print out the automata.

*Exercise 5.* Take again the working model but model the time-out using clocks and by bounding the transmission time, i.e. the medium does not send a signal to the sender. Revalidate the protocol.

## 4 The CSMA/CD-protocol

Read the rest of “Verifying a CSMA/CD-protocol with CCS”. As a preliminary step, draw up the automata  $MAC_1$ ,  $MAC_2$  and  $M$  in UPPAAL.

*Exercise 6.* Validate and verify that the system is functionally correct. Print out the automata and your property file.  $\square$

*Exercise 7.* In the exercise you have verified a number of invariant properties of the CSMA/CD-protocol. Do your properties ensure that the protocol is correct? Is the system ensured to be dead-lock and live-lock<sup>1</sup> free? In particular you will want to check that one MAC side is not in a state where a message is incoming and it is sending one at the same time. Check also the capacity of the medium, i.e. two messages can be in progress in one direction.  $\square$

As mentioned in the discussion, the model of the MAC is slightly inaccurate. In reality, the MAC would be two processes: one *sender* which performs actions  $send!$ ,  $b!$ ,  $e!$ ,  $c?$  and one *receiver* which performs actions  $rec!$ ,  $br?$  and  $er?$

<sup>1</sup> UPPAAL does not check for livelocks

*Exercise 8.* Redefine the protocol by letting each *MAC* consist of two processes, one sender (*S*) and one receiver (*R*). The idea is to let *S* perform all actions involving message transition, i.e. the “horizontal” transitions in the *MAC* diagram, while *R* performs all actions involving message reception, i.e. the “vertical” transitions. Replace *MAC* with *S* and *R* run in parallel. Print out the automata. Use UPPAAL to prove that the modified protocol is no longer correct and explain the behaviour of the protocol in words<sup>2</sup>. □

*Exercise 9.* Redefine the protocol once again so that it works, while still keeping the *S* and *R* separate. You may need to add some channels to achieve synchronisation between *S* and *R*, or between *S*, *R* and the medium<sup>3</sup>. □

---

<sup>2</sup> Hint: exhibit a livelock by decoration of the model and loop detection

<sup>3</sup> Hint: think of semaphores

## References

- [BRRdS96] Amar Bouali, Annie Ressouche, Valérie Roy, and Robert de Simone. The FC2TOOLS set. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of 8th Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 441–445. Springer-Verlag, 1996.
- [JLS96] H.E. Jensen, K.G. Larsen, and A. Skou. Modelling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL. In *Proc. of 2nd International Workshop on the SPIN Verification System*, pages 1–20, August 1996.
- [LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-Checking for Real-Time Systems. In *Proc. of Fundamentals of Computation Theory*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88, August 1995.
- [LPY97] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear-Box Controller: an Industrial Case Study using UPPAAL. In preparation, 1997.
- [Par88] Joachim Parrow. Verifying a csma/cd-protocol with ccs. In Aggarwal and Sabnani, editors, *Proc. of the 8th IFIP Symposium on Protocol Specification, Testing and Verification*, pages 373–384. North-Holland (1988), June 1988.
- [Yi91] Wang Yi. *A Calculus of Real Time Systems*. PhD thesis, Chalmers University of Technology, 1991.
- [YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the 7th International Conference on Formal Description Techniques*, 1994.

