T–79.4301 **Autumn 2008**

Parallel and Distributed Systems

Home Exercise 1 - Deadline: 6th of November 2008 at 12:15 (strict deadline!)

Return your answer via email to `t794301-autumn08@tcs.hut.fi` with "Homework 1, [student number]" as the subject. Attach to the email a zip or tar file with a separate file for each part of the homework, named "`part-a.pml`", "`part-b.pml`", "`part-c.txt`", etc. A template tar file for the homework can be found from:

`https://noppa.tkk.fi/noppa/kurssi/t-79.4301/harjoitustyot`

under Home exercise 1. Download it, unpack the files, and modify them to contain your answers. When you are done, pack the files to a tar or zip file. On unix workstations this can be done with the command "`tar cvf homework1.tar homework1`", when you have a directory "`homework1`" with your answer files in it. Then send the package to the course email address with the subject above. Submissions that arrive late are not graded! Be sure to send your answer in time, and remember that it may take some time for email messages to arrive. You will receive a confirmation when your submission arrives.

The home exercises are personal, no group work allowed! There are three rounds of home exercises of 10 points each. To pass the home exercises ≥ 15 points are needed and ≥ 24 points gives a $+1$ to the exam grade (no effect to exam grades 0 or 5).

1. Consider the design of an elevator (lift) control logic for a building with four floors 0, 1, 2, and 3. We would like to model this logic in Promela and analyze some safety features of it using the Spin model checker.

   Each of the floors has just one elevator call button: `call_0`, `call_1`, `call_2`, and `call_3`. Inside the elevator there are also three buttons for selecting to which floor the user wants the elevator to take her/him: `go_0`, `go_1`, `go_2`, and `go_3`.

   The elevator doors can be opened by sending a message called `open` to the elevator and closed by sending the message `close`. The elevator can be commanded to move one floor up by sending it the message `up` and one floor down by sending the message `down`.

   In the initial state of the system the elevator is at floor 0 and its doors are closed.

   You are given a partial Promela model (below) where the button pushes at floors are transmitted to the `controller` through a channel called `floor_buttons`, button pushes in the elevator are transmitted to the

`controller` through a channel called `elevator_buttons` and messages to the elevator are sent to a channel called `commands`.

a) Add the elevator controller to the Promela model in the `controller proctype`. (Please include the full Promela model in your answer.) (5 p.)

b) Modify the Promela model for the `elevator proctype` to contain an assertion which triggers if your model sends the `up` command at floor 3 or the down command at floor 0. (Please include the full Promela model in your answer.) (1 p.)

c) Verify with Spin that the assertion does not trigger with your elevator controller. Hint: Using only the simulation mode of Spin is not sufficient here! (1 p.) (Please include a Spin run log in your answer.)

d) Modify the Promela model for the `elevator proctype` to contain an assertion which triggers if your model sends the `up` or `down` command while the elevator doors are open. (Please include the full Promela model in your answer.) (1 p.)

e) Verify with Spin that the assertion does not trigger with your elevator controller. (Please include a Spin run log in your answer.) (1 p.)

f) Is you controller fair: Is it possible in the Promela model that a repeated sequence of requests `call_i` for an elevator at a floor i is from some time point on ignored without the elevator ever stopping at floor i? (Answer: Just a short English language analysis of your model, four sentences at maximum.) (1 p.)

```
/* Partial Promela model of an elevator. */
/* Available through the course Noppa Assignments page: */
/* https://noppa.tkk.fi/noppa/kurssi/t-79.4301/harjoitustyot */

mtype = { call_0, call_1, call_2, call_3,
          go_0, go_1, go_2, go_3,
          open, close,
          up, down}

chan floor_buttons = [0] of { mtype };
chan elevator_buttons = [0] of { mtype };
chan commands = [0] of { mtype };

active proctype elevator() {
    do
        :: commands ? open -> printf("Elevator: opened doors.\n");
        :: commands ? close -> printf("Elevator: closed doors.\n");
        :: commands ? up -> printf("Elevator: moved up one floor.\n");
        :: commands ? down -> printf("Elevator: moved down one floor.\n");
    od
}

/* Simulates random pushing of call buttons. */
active proctype floor_button_input() {
    do
        :: floor_buttons ! call_0;
        :: floor_buttons ! call_1;
        :: floor_buttons ! call_2;
        :: floor_buttons ! call_3;
    od
}

/* Simulates random pushing of elevator buttons. */
active proctype elevator_button_input() {
    do
        :: elevator_buttons ! go_0;
        :: elevator_buttons ! go_1;
        :: elevator_buttons ! go_2;
        :: elevator_buttons ! go_3;
    od
}

active proctype controller() {
    int at = 0;
    bool closed = true;

    /* Implement your own elevator controller here! */

}
```