T–79.4301 **Spring 2006**
Parallel and Distributed Systems
Home exercise 1 – Deadline: Friday, 17th of March 2006 at 12:15 (strict deadline!)

Please return your answer to the exercise before the deadline by e-mail to `Heikki.Tauriainen@tkk.fi` (include the course code T-79.4301 in the subject). Include in your answer: your name, student id, e-mail address, short answers to the exercise questions (plain ASCII text, PDF, or PostScript accepted). Return also the requested Promela models as additional e-mail attachments, clearly indicating at the beginning of each file (using Promela comments) the number of the question to which the file gives the answer. (We also accept ".tar", ".tar.gz", ".tar.bz2", or ".zip" archives if that is more convenient for you.)
The home exercises are personal, no group work allowed! There are three rounds of home exercises of 10 points each. To pass the course, you need to earn $\geq$ 15 points (out of a maximum of 30) from the three exercises. A score of $\geq$ 24 points raises a passing exam grade ($\geq$ 1) by 1 (no effect to exam grades 0 or 5).

1. Consider the design of an elevator (lift) control logic for a building with three floors 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 one elevator call button: `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_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 made 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 1 and its doors are closed.

   You are given a partial Promela model (below; also available for download at the course home page <`http://www.tcs.hut.fi/Studies/T-79.4301/`>) 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` without modifying the other `proctype` definitions. For simplicity, the controller is allowed to complete all operations caused by a button press before responding to another button press. (Please include the full Promela model in your answer as a separate e-mail attachment.) (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 1. (Hint: You may need to add also some internal state information to this `proctype`. Please include the full Promela model in your answer as a separate e-mail attachment.) (1 p.)

c) Verify with Spin that the assertion does not trigger with your elevator controller. (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 as a separate e-mail attachment) (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 your controller fair? In other words, is it possible in your model that a repeated sequence of requests `call_`i for an elevator at a floor i is ignored from some time point on without the elevator ever stopping at floor i? Give a short analysis (a few sentences in Finnish or English) of your model as your answer. (1 p.)

```
/* Partial Promela model of an elevator. */

mtype = { call_1, call_2, call_3,
          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_pusher() {
    do
        :: floor_buttons ! call_1;
        :: floor_buttons ! call_2;
        :: floor_buttons ! call_3;
    od
}

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

active proctype controller() {
    int at = 1;
    bool closed = true;

    /* Implement your own elevator controller here! */

}
```