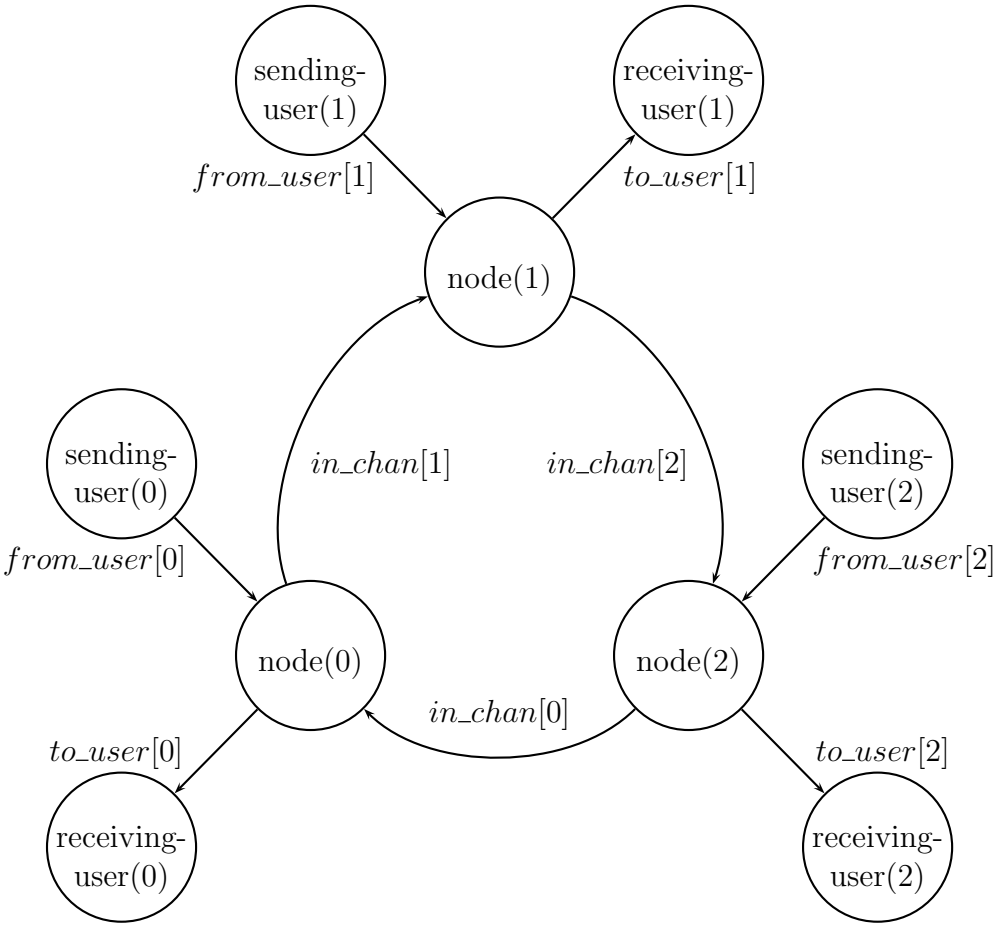
T–79.4302 **Autumn 2013**
Parallel and Distributed Systems
Assignment 2 - Deadline: 20th of November 2013 at 10:15 (strict deadline!)

Return your answer via email to `t794302@ics.aalto.fi` with "assignment 2, [student number]" as the subject. Attach to the email a zip or tar file with a separate file for each part of the assignment, named "`part1-a.pml`", "`part1-b.txt`", "`part2-a.pml`", etc. A template tar file for the assignment can be found from:
`https://noppa.tkk.fi/noppa/kurssi/t-79.4302/harjoitustyot`
under Assignment 2. 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 assignment2.tar assignment2`", when you have a directory "`assignment2`" 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. The course assistant will manually send you a confirmation once he receives your submission.
The assignments are personal, no group work allowed! There are three rounds of assignments of 8 points each. To pass the assignments ≥ 12 points are needed. The grade for this course depends both on the result of the exam and on the result for the assignments.

1. We are designing a communication protocol with a unidirectional ring topology and want to model the initial design with Promela. The system consists of three nodes numbered 0, 1, and 2 and denoted `node(0)`, `node(1)`, and `node(2)`, respectively.

   The only means of communication in the system between nodes is provided by a unidirectional ring of message channels. The node i can only directly send messages to its following neighbour node $i + 1$ (modulo 3). Thus the node `node(i)` can thus send messages to a message channel `in_chan[(i+1)% 3]` which are read by node `node((i+1)% 3)`. All channels have a capacity of 1 message, and the send mechanism is such that sending to a full channel will block inside the sender procedure and prohibit the sending node from proceeding. In addition, there is no direct capability to check whether a send would block as this is not implemented in the underlying hardware system.

   Each node has a sending user attached to it which wants to send messages consisting of two fields: `(data, to_node)`, where `data` denotes



that the packet to be sent is a data packet destined to the receiving user attached to node `to_node`. The actual data sent is abstracted away from the Promela model of the protocol. The nodes must now relay packets to other nodes in order to implement a usable data transport between all the sending and receiving users attached to the nodes.

(a) Create a Promela model of a ring protocol containing no mechanism to avoid deadlocking of the system due to all the message channels being full. Please return your final Promela model.
(b) Find the deadlock of your system using Spin and give a log of the verification run.
(3 p.)

2. In the second phase we add a distributed flow-control mechanism to the protocol working as follows. A new protocol message `(hole,0)` called a *hole* is added to the system and initially the message channels

are initialized with 2 holes (see Promela code of the `init proctype` below). They are used to count the amount of free capacity available in the three channels in a distributed fashion.

If a node does not want to send anything and receives a hole message, it forwards the hole to the next node like any other messages not destined to it. If the node wants to send a new message from the user attached to it, it must wait until it receives a hole message. When a hole is received, it is not forwarded but instead the data message is sent in its place. Only when a message is delivered to the receiving user at the destination node, a new hole is generated.

(a) Model this ring protocol with the distributed flow-control using holes as described above in Promela. Please return also your final Promela model.
(b) Prove that your system is deadlock free using Spin and give a log of the verification run.
(4 p.)

Hint: It is easy to overflow the capacity of Spin with this part. Be very careful in your modelling decisions (no extra variables, using `atomic` if you run into capacity problems, etc.).

3. Consider the following generic question: Suppose you have some unidirectional ring protocol which you have model checked to be deadlock free for $N = 3$ nodes. (Any protocol, that is not the one discussed above.) Is it the case that the protocol is also deadlock free for $N = 4$ nodes? (1p.)

```
/* Message types used by the protocol. Do not change. */
mtype = {data, hole};

chan in_chan[3] = [1] of { mtype, byte };
chan from_user[3] = [0] of { mtype, byte };
chan to_user[3] = [0] of { mtype, byte };

/* A process modelling the sending user. Do not change. */

proctype sending_user(byte id) {
    /* Send a message randomly to one of the nodes. */
    do
        :: from_user[id] ! data, 0
        :: from_user[id] ! data, 1
        :: from_user[id] ! data, 2
    od
}

/* A process modelling the receiving user. Do not change. */

proctype receiving_user(byte id) {
    byte message_to;
    mtype message_type;
    do
        :: to_user[id] ? message_type, message_to;
           atomic {
               assert((message_type == data) && (message_to == id));
               message_to = 0;
               message_type = 0;
           }
    od
}

/* A simple ring protocol in Promela */

proctype node(byte id) {
    /* Add your Promela model of the node here. */

    /* Following line added to make a syntactically valid Promela */
    /* model. Remove it first.                                    */
    skip
}
```

```
/* The init process for starting all other processes. */

init {
    int id;
    atomic {
        id = 0;
        do
            :: (id < 3) ->
                   run node(id);
                   run receiving_user(id);
                   run sending_user(id);
                   id++
            :: (id == 3) -> break
        od;

/* Uncomment the following #define to generate the initial holes in the */
/* ring in question 2.                                                  */

/* #define QUESTION2 */
#ifdef QUESTION2
        /* Send 2 holes to the channels 0,1. */
        id = 0;
        do
            :: (id < 2) -> in_chan[id] ! hole, 0; id++
            :: (id == 2) -> break
        od
#endif
    }
}
```