Formal Methods in Software Development
Exercise 4 (July 5)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

May 23, 2007

The result is to be submitted to me by **July 5** (hard deadline) either as an email with a single PDF file as attachment or as a paper sent to me ("Wolfgang Schreiner, RISC") by university mail (you may use any university mail box).

1 Distributed Termination Detection

Consider the following problem:

> *Implement a distributed termination detection algorithm for a network of N nodes. Each node can be either in active or in passive state. Only an active node can send messages to other nodes; each message sent is received after some period of time later. After having received a message, a passive node becomes active; the receipt of a message is the only mechanism that triggers for a passive node its transition to activity. For each node, the transition from the active to the passive state may occur "spontaneously". The state in which all nodes are passive and no messages are on their way is stable: the distributed computation is said to have terminated. The purpose of the algorithm is to enable one of the nodes, say node 0, to detect that this stable state has been reached.*

The following algorithm (by Dijkstra and Safra) solves the problem:

- Every node maintains a counter c. Sending a message increases c by one; the receipt of a message decreases c by one. The sum of all counters thus equals the number of messages pending in the network.
- When node 0 initiates a detection probe, it sends a token with a value 0 to node $N-1$. Every node i keeps the token until it becomes passive; it then forwards the token to node $i-1$ increasing the token value by c.
- Every node and also the token has a color (initially all white). When a node receives a message, the node turns black. When a node forwards the token, the node turns white. If a black machine forwards the token, the token turns black; otherwise the token keeps its color.

- When node 0 receives the token again, it can conclude termination, if
  - node 0 is passive and white,
  - the token is white, and
  - the sum of the token value and c is 0.

  (and also inform the other nodes). Otherwise, it may start a new probe.

The core class of a Java program simulating this algorithm is attached.

Your tasks are the following:

1. Develop a formal model of a system of N nodes implementing this algorithm (in the style of the client/server system shown in class).

   Hint: each node may e.g. operate on variables *active*, *send*, *receive*, *term*, c, *color*, *probing*, *tsend*, *treceive* such that

   - *active* signals whether the node is active,
   - $send(i)$ represents the output buffer from this node to node i,
   - $receive(i)$ is the input buffer from node i to this node,
   - *term* is used by node 0 to signal the termination of the system,
   - c is the counter value of the node,
   - *color* is the color of the node,
   - *probing* is set by node 0 when it starts termination detection,
   - *tsend* represents the link for sending a token,
   - *treceive* represents the link for receiving a token.

   As shown in class, the system is defined by the set of external transitions and the composition of the system components; each component is described by it state space, internal transitions, initial state condition, and transition relation. Please note that also the communication channels between the nodes have to be modelled as components (the channels for the normal messages are different from the channels for passing the token).

2. Develop a Promela model of a system of 5 nodes implementing this algorithm, formulate in LTL the correctness property of the algorithm ("if node 0 concludes termination, the system is indeed terminated") in a suitable way and model-check the property. (If the model checking takes too long with 5 nodes, you may reduce the system to 4 nodes).

The result of this exercise consists of a PDF file or paper containing

1. the formal model,
2. the listings of the Promela model and of the LTL formula (including the definitions of the atomic predicates),
3. a screen shot of a (limited) simulation of the model and the output of model checking the LTL formula.

If Spin claims that the property is violated, investigate the generated counterexample in order to track down the bug in your model/formula; if you can't find the error, explain what you think has gone wrong.

```
class Prog extends Program
{
  public int i;                   // node number

  public boolean active = true;   // flag indicating the state of the process
  public int c = 0;               // messages sent - messages received
  public int color = Token.WHITE; // color of the node
  public Token token = null;      // token held by node
  public boolean probing = false; // flag representing state probing
  public boolean term = false;    // boolean flag indicating termination

  // called for the initialization of the program
  public Prog(int process)
    {
      i = process;
    }

  private final int PSEND = 3;    // inverse probability for sending message
  private final int PPASSIVE = 5; // inverse probability for becoming passive
  private final int PSTART = 3;   // inverse probability for starting probe

  // called for the execution of the program
  public void main()
    {
      // signal/message channels
      InChannelSet sigIn = new InChannelSet();
      InChannelSet msgIn = new InChannelSet();
      OutChannelSet sigOut = new OutChannelSet();
      OutChannelSet msgOut = new OutChannelSet();

      sigIn.addChannel(in(0));
      int is = in().getSize();
      for (int i = 1; i < is; i++)
        msgIn.addChannel(in(i));

      sigOut.addChannel(out(0));
      int os = out().getSize();
      for (int i = 1; i < os; i++)
        msgOut.addChannel(out(i));

      while (!term)
        {
          // an active node may become passive at its will
          if (active && random(PPASSIVE) == 0)
            {
              active = false;
              continue;
            }

          // an active node may send a message at its will
          if (active && random(PSEND) == 0)
            {
              int j = random(Main.PNUM-1);   // select channel and send msg
              msgOut.getChannel(j).send(new Msg(random(100)));
              c = c+1;                       // decrease message counter
              continue;
            }

          // node 0 may start a probe at its will
          if (i == 0 && !probing && random(PSTART) == 0)
            {
              probing = true;
```

```
            color = Token.WHITE;
            sigOut.getChannel(0).send(new Token(0, Token.WHITE));
            continue;
          }

        // any node may receive a message
        if (msgIn.select(1) != -1)
          {
            int j = msgIn.select();        // get channel
            Msg msg = (Msg)msgIn.getChannel(j).receive();
            c = c-1;                       // decrease message counter
            active = true;                 // switch active again
            color = Token.BLACK;           // switch color to black
            continue;
          }

        // process 0 may receive token again
        if (i == 0 && sigIn.select(1) != -1)
          {
            token = (Token)sigIn.getChannel(0).receive();
            term = token.color == Token.TERM;
            if (term) continue;
            if (!active && c+token.value == 0 &&
                color == Token.WHITE && token.color == Token.WHITE)
              sigOut.getChannel(0).send(new Token(0, Token.TERM));
            else
              probing = false;
            token = null;
            continue;
          }

        // inactive process may forward a token
        if (i != 0 && !active && sigIn.select(1) != -1)
          {
            token = (Token)sigIn.getChannel(0).receive();
            if (color == Token.BLACK)
              sigOut.getChannel(0).send(new Token(c+token.value,
                                                  Token.BLACK));
            else
              sigOut.getChannel(0).send(new Token(c+token.value,
                                                  token.color));
            term = token.color == Token.TERM;
            if (term) continue;
            color = Token.WHITE;
            token = null;
            continue;
          }
      }
    }

  // deliver random number 0 <= r < n
  Random rand = new Random();
  public int random(int n)
  {
    return (Math.abs(rand.nextInt()) % n);
  }
}
```