CS 511: Homework Assignment 4
Due: July 2, 11:55pm

Contents

1 Assignment Objectives and Policies

Get acquainted with

- simple model development in Promela;
- basic model checking in Spin;
- the use of LTL formulas for expressing properties of systems.

The following Assignment Policies should be upheld:

Collaboration Policy. This homework may be done in individually or in pairs. Use of the Internet is allowed, but should not include searching for existing solutions. In case it is done in pairs, only one of the members should submit through Canvas.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be reused. Violations will be penalized appropriately.

Late Policy. Late submissions are allowed with a penalty of 2 points per hour past the deadline.

Note: This assignment draws from [BK08]. In particular, the description given in Sec. 2 is taken from pp. 238-239.

2 Leader Election

In current distributed systems several services are offered by some dedicated process(es) in the system. Consider, for example, address assignment and registration, query coordination in a distributed database system, clock distribution, token regeneration after token loss in a token ring network, initiation of topology updates in a mobile network, load balancing, and so forth. Usually many processes in the system are potentially capable of providing these services. However, for consistency reasons it is usually the case that at any time only one process is allowed to actually provide a given service. This process – called the "leader" – is in fact elected. Leader election is a simple form of symmetry breaking. Sometimes it suffices to elect an arbitrary process, but for other services it is important to elect the process with the best capabilities for performing that service. Here we abstract from specific capabilities and use ranking on the basis of process identities. Each process has a unique identity, and it is assumed that a total ordering exists on these identities. The idea is therefore that the higher the process' identity, the better its capabilities.

The process of choosing a leader is known as *leader election*. A leader election algorithm (LEA):

- is decentralized;
- each node has the same local algorithm;
- each node has a unique ID and IDs form a totally ordered set;
- upon termination one node is 'leader' and the rest are 'lost'.

In this assignment we are going to look at two leader election algorithms on a ring topology:

- Part I: Chang-Roberts LEA or CR-LEA.
- Part II: Dolev-Klawe-Rodeh LEA or DKR-LEA (`http://w3.cs.huji.ac.il/~dolev/pubs/n-log-n.pdf`).

In each of these two algorithms, initially all processes are active. A process may become passive, in which case it is out of the race to become leader. The conditions under which it becomes passive depends on the algorithm and will be made precise below.

3 Part I: CR-LEA

This part of the assignment requests that you perform the task of implementing CR-LEA in Promela.

3.1 CR-LEA Description

The idea of the algorithm is that only the message with the highest id completes one round in the ring. Each node in the ring should behave as indicated by the following pseudocode:

```
send(id) to next process in ring;
while (true) do
  receive (m);
  if (m = id) then stop;    /* process is the leader */
  if (m > id) then send(m); /* forward identifier, id becomes passive */
  if (m < id) then do nothing; /* purge message */
od
```

Once the leader is elected, every node should be informed and print out the following message exactly once (per node): "I am node X and I know the leader is Y".

Finally, make sure that the execution of your implementation does not end in a timeout.

Note: CR-LEA has message complexity $\mathcal{O}(N^2)$ (it takes at most $\frac{n(n+1)}{2}$ messages to elect a leader). DKR-LEA has message complexity $\mathcal{O}(N \log N)$.

3.2 Implementation

Implement CR-LEA in Promela. Nodes should be connected in a ring topology. The buffer size for the Promela channels should be set to $2 * N$ where N is the number of processes in the network.

Hint: It is strongly recommended that, on a first reading, you continue with the next section. The implementation for DKR-LEA will be provided for you, and will serve as a guideline for implementing, the more simple, CR-LEA.

4 Part II: DKR-LEA

This part of the assignment requests that you perform just one task, namely verify properties of DKR-LEA using LTL formulae.

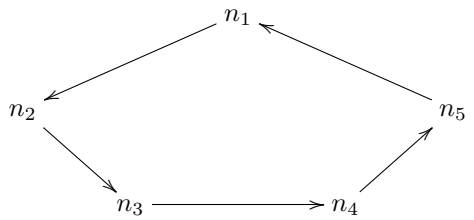
4.1 DKR-LEA Description

An implementation of DKR-LEA in Promela is provided for you below. First, an informal description. DKR adapts ideas from Franklin's algorithm (which applies to undirected rings) with the aim of improving the message complexity of the Chang-Roberts algorithm. The basic idea depends on the fact that, since order of communication is preserved, the effect of the algorithm is as though the activities of the system could be separated into synchronous phases. Initially all processes are active. In each phase, sufficient messages are passed so that each active process learns the current numbers of the two closest active processes on its left. With this information, each active process can determine if it is the active process immediately following a local maximum. If so, it assumes the local maximum as its own number, and continues as active on to the next phase. If not, it becomes passive and merely acts as a communication relay, passing on the messages initiated by active processes during later phases.

4.2 Implementation

Consider the following implementation[1] in Promela of DKR-LEA. It consists of two proctypes, `nnode` and `init`. The code for these proctypes are presented in two parts to ease readability. The former is the code that runs on each node in the ring; the latter initializes the ring.

It is suggested that you perform a symbolic execution of this algorithm in a ring with 5 nodes n_1 with id 1, ..., n_5 with id 5, connected as follows:



To simplify the state of the system you can write the messages that are exchanged on the arrows, the local state of each node next to the name of the node and you may assume that all nodes fire messages at the same time (although of course this is not necessarily the case in practice).

[1] The code itself is from the spin website.

4.2.1 Part 1

```
#define N       5       /* number of processes in the ring */
#define L       10      /* size of buffer  (>= 2*N) */
byte I;                 /* will be used in init for assigning ids to nodes */

mtype = { one, two, winner };       /* symb. Msg. Names */
chan q[N] = [L] of { mtype, byte};  /* asynchronous channels */

proctype nnode (chan inp, out; byte mynumber)
{   bit Active = 1, know_winner = 0;
    byte nr, maximum = mynumber, neighbourR;

    xr inp;  /* channel assertion: exclusive recv access to channel in */
    xs out;  /* channel assertion: exclusive send access to channel out */

    printf("MSC: %d\n", mynumber);
    out!one(mynumber);
end:    do
    :: inp?one(nr) ->
            if
            :: Active ->
                    if
                    :: nr != maximum ->
                            out!two(nr);
                            neighbourR = nr
                    :: else ->
                            know_winner = 1;
                            out!winner,nr;
                    fi
            :: else ->
                    out!one(nr)
            fi

    :: inp?two(nr) ->
            if
            :: Active ->
                    if
                    :: neighbourR > nr && neighbourR > maximum ->
                            maximum = neighbourR;
                            out!one(neighbourR)
                    :: else ->
                            Active = 0
                    fi
            :: else ->
                    out!two(nr)
            fi
    :: inp?winner,nr ->
            if
            :: nr != mynumber ->
                    printf("MSC: LOST\n");
            :: else ->
                    printf("MSC: LEADER\n");
            fi;
            if
            :: know_winner
            :: else -> out!winner,nr
            fi;
            break
    od
}
```

4.2.2 Part 2

The `init` process assigns random ids to each node and then starts them.

```
init {
    byte proc;
    byte Ini[6];        /* N<=6 randomize the process numbers */
    atomic {
        I = 1;  /* pick a number to be assigned 1..N */
        do
        :: I <= N ->
                if      /* non-deterministic choice */
                :: Ini[0] == 0 && N >= 1 -> Ini[0] = I
                :: Ini[1] == 0 && N >= 2 -> Ini[1] = I
                :: Ini[2] == 0 && N >= 3 -> Ini[2] = I
                :: Ini[3] == 0 && N >= 4 -> Ini[3] = I
                :: Ini[4] == 0 && N >= 5 -> Ini[4] = I
                :: Ini[5] == 0 && N >= 6 -> Ini[5] = I  /* works for up to N=6 */
                fi;
                I++
        :: I > N ->     /* assigned all numbers 1..N */
                break
        od;

       /* start all nodes in the ring */
        proc = 1;
        do
        :: proc <= N ->
                run nnode (q[proc-1], q[proc%N], Ini[proc-1]);
                proc++
        :: proc > N ->
                break
        od
    }
}
```

4.3 Verification

1. Using assertions:
   (a) Verify that if a node is selected as leader, then its id is the maximum one.
   (b) Verify that only one leader is elected. Hint: add a variable `nr_leaders` to count the number of processes that think they are leaders.
2. Using LTL formulae:
   (a) Some node is eventually elected as leader. Hint: use `nr_leaders`.
   (b) Eventually, it will always be true that there is exactly one elected leader
   (c) It is always the case that there are either 0 or 1 leaders chosen.

5 Submission Instructions

Submit your Promela source code via Canvas by the indicated date and time. Submit a zip file named `Assignment5_<Surnames>.zip` (where `<Surnames>` should be replaced by the surnames of the team members) containing:

1. The Promela source for the first exercise
2. The Promela source for each verification item together with the output from spin (save as text).

In total you should have 10 files.

1. CR-LEA source code
2. CR-LEA running output as txt
3. DKR-LEA source code with assertion statements
4. DKR-LEA running output as txt
5. DKR-LEA source code with ltl statement 1 at top of file
6. DKR-LEA source code with ltl statement 2 at top of file
7. DKR-LEA source code with ltl statement 3 at top of file
8. DKR-LEA ltl statement 1 verification output as txt
9. DKR-LEA ltl statement 2 verification output as txt
10. DKR-LEA ltl statement 3 verification output as txt

References

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008.