CS 511: Homework Assignment 5
Due: 12 December, 11:59pm

1 Assignment Objectives and Policies

The following Assignment Policies should be upheld:

Collaboration Policy. This homework must be done individually. Use of the Internet is allowed, but should not include searching for existing solutions.

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 services are offered by one or more dedicated processes in the system. Usually many processes in the system are potentially capable of providing these services. However, for consistency reasons it is typically 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. This assignment is thus interested in *leader election*. Leader election is a simple form of symmetry breaking. Sometimes it suffices to elect an arbitrary process as leader, 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.

As mentioned, the process of choosing a leader is known as *leader election*. A leader election algorithm (LEA):

- is decentralized;
- each node executes the same code;
- each node has a unique ID and IDs form a totally ordered set;
- upon termination one node is "leader" and the rest "lose".

In this assignment we are going to:

- Part I: Complete the code for a simple leader election protocol (Chang-Roberts LEA or CR-LEA) for a set of nodes connected via a ring topology.
- Part II: Verify three simple properties of the algorithm from Part I by proposing LTL formula for them and then verifying that they hold using Spin.

3 Part I: CR-LEA

This part of the assignment requests that you perform the task of implementing CR-LEA in Promela. The algorithm proceeds in two phases of message passing. In the first phase (round 1), only the message with the highest id completes a roundtrip in the ring. This one message is then sent around the ring for a second round (round 2) to inform all nodes about the winner.

Each node in the ring should behave as indicated by the following pseudocode:

```
send(Id,round1) to next process in ring;
do
  ::  receive (M,round1) from previous node in the ring ->
      if
         :: M = id -> send(M,round2); /* process is the leader, start round2 */
         :: M > id -> send(M,round1); /* forward identifier to next node */
         :: M < id -> do nothing;     /* purge message */
      id
  ::  receive (M,round2) from previous node in the ring ->
      if
        :: M != Id -> send(Id,round2) to next process in ring;
        :: else -> do nothing /* I am leader */
      fi;
      break;
od
```

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

3.1 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. Use the stub from Figure 1 as guideline. Remember to rename it to `cr.pml`. You only have to add code where it says "complete here".

Each node has an id held in the variable `mynumber`. Also, each node has two channels, an input channel `inp` for receiving messages from its "left" neighbor and an output channel `out` for sending messages to its "right" neighbor. Messages are sequences of fields. In this case all messages have two fields: the first is an enum value (`one` or `winner`) and the second is a number. For example, `out!one(mynumber)` (or equivalently, `out!{one,mynumber}`) sends the message `{one,mynumber}` via the channel `out`.

4 Part II: Verification

Propose LTL formulae for the properties listed below. Recall the informal meaning of the two modal operators:

$\Box A$ "Always A"
$\Diamond A$ "Eventually A"

1. Property `p1`: Some node is eventually elected as leader. Hint: use an auxiliary variable `nr_leaders`.
2. Property `p2`: Eventually, it will always be true that there is exactly one elected leader
3. Property `p3`: It is always the case that there are either 0 or 1 leaders chosen.

For each of these properties you must set `jSpin` to `Acceptance` and make sure that `Weak Fairness` is checked. You can also use the command line:

```
# spin -a cr.pml
ltl p2: your formula here
# gcc -o pan pan.c
# ./pan -a -f

(Spin Version 6.5.1 -- 20 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p2)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 168 byte, depth reached 260, errors: 0
...
```

5 Submission Instructions

Submit your solution via Canvas by the indicated date and time. Submit a zip file named `Assignment5.zip` containing four files:

```
#define N       4
#define L       10

byte nr_leaders;

/* ltl p1 { your_ltl_formula_1_here }; */
/* ltl p2 { your_ltl_formula_2_here }; */
/* ltl p3 { your_ltl_formula_3_here }; */

mtype = { one, winner };
chan q[N] = [L] of {mtype, byte};

proctype nnode (chan inp, out; byte mynumber)
{
  byte nr, neighbor;

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

  printf("NNode: %d\n", mynumber);

  out ! one(mynumber);

  end: do
      :: inp ? one(nr) ->
         /* complete here */
      :: inp ? winner(nr) ->
         /* complete here */
         break;
  od
  do /* dummy loop to ensure non-termination */
    :: true -> skip
  od
}

init {
  byte Ini[6];
  byte I;

  for (I : 1 .. N) {
     if
       :: 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
    fi;
  }
  atomic {
    int proc;
    for (proc : 1 .. N ) {
       run nnode (q[proc-1],q[proc%N], Ini[proc-1]);
       printf("Initializing %d %d %d \n", q[proc-1],q[proc%N], Ini[proc-1])
    }
  }
}
```

Figure 1: Stub to complete

1. The Promela source for the first exercise **including the three ltl formulae** (file `cr.pml`).
2. The output of spin for property `p1` (put the text in a file `output_p1.txt`).
3. The output of spin for property `p2` (put the text in a file `output_p2.txt`).
4. The output of spin for property `p3` (put the text in a file `output_p3.txt`).

References

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