Formal Methods in Software Development
Exercise 8 (January 27)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

The result is to be submitted by the deadline stated above *via the Moodle interface* of the course as a `.zip` or `.tgz` file which contains

1. a PDF file with
   - a cover page with the course title, your name, Matrikelnummer, and email address,
   - a section for each part of the exercise with the requested deliverables and optionally any explanations or comments you would like to make;
2. the file with the Promela model used in the exercise.
3. the files with the LTL properties (Button "Save As" in the LTL Property Manager).

Email submissions are *not* accepted.

Exercise 8: Model Checking Leader Election in Spin

We consider a system of n processes $p_0, \ldots, p_{n-1}$ and n channels $c_0, \ldots, c_{n-1}$ each of which may hold m messages. Each process p_i can receive a message from channel c_i and send a message to channel $c_{(i+1) \mod n}$, i.e., the processes are organized in a unidirectional "ring".

From time to time a process may desire to be elected the "leader" of the ring. The core requirement is that at every moment the ring has at most one leader. We want to ensure this by the following protocol:

- Every process nondeterministically cycles through the states "idle" (no activity), "waiting" (having requested to be elected the leader), or "leader" (having been successfully elected the leader).
- If process p_i is in state "idle", it may switch to state "waiting", i.e., it may request its election to the leader. For this purpose, it sends its identifier i to its successor in the ring.
- Every process p_i in state "idle" or "waiting" is ready to receive a message from its predecessor; if such a message (a process identifier) j is received, it is handled as follows:
  - If p_i is in state "idle", it forwards j to its successor.
  - If p_i is in state "waiting", there are three cases:
    1. If $j < i$, then p_i does nothing (i.e., it discards j).
    2. If $j > i$, then p_i forwards j to its successor and switches to state "idle" (i.e., it has lost the election).
    3. If $j = i$, then p_i switches to state "leader" (i.e., it has won the election).
- Every process p_i in state "leader" does not receive or send any message but only flips the truth value of a Boolean variable b (from "true" to "false" and vice versa). At any time, however, it may also switch to state "idle" and thus make room for another leader.

By this protocol, if there are multiple processes simultaneously competing for leadership, the one with the highest process identifier "wins" the election and becomes the leader.

Your tasks are as follows:

1. Implement above model for process number $n = 4$ and $m = 2$ in Promela[1]. Take attached file `LeaderElection.txt` as the starting point of your implementation (it already implements two of the transitions described above

   Make sure that your model does not allow repeated transitions where nothing changes (i.e., no variable is changed and no message is received or sent); such "stuttering steps" unnecessarily violate the progress properties given below. In particular, the main loop must not contain `true -> skip` transitions; if no incoming message is available, the loop must be blocked.

2. Run a simulation for several hundred steps. The simulation must not run into a deadlock.

[1] If model checking with $m = 2$ is not feasible, choose $m = 1$ (you may then also try $n = 5$); generally choose the largest model that you can reasonably check.

3. Formulate in Spin LTL the property

   > Always, if process i is the "leader", no other process is also leader.

   and check it for $i = 0$ and $i = N - 1$. Analyze the results in detail and explain whether they indicate an error in your model or not.

4. Formulate in Spin LTL the property

   > Some process is infinitely often the "leader".

   and check it. Analyze the results in detail and explain whether they indicate an error in your model or not.

5. Formulate in Spin LTL the property

   > Every process is infinitely often *not* the "leader".

   and check it. Analyze the results in detail and explain whether they indicate an error in your model or not.

6. Formulate in Spin LTL the property

   > If process i infinitely often wants to become the "leader", it eventually becomes the leader.

   and check it for $i = 0$ and $i = N - 1$. Analyze the results in detail and explain whether they indicate an error in your model or not.

If property 5 does not hold, change it such that it assumes a suitable fairness condition and show that this makes the property true. Does weak fairness suffice or is strong fairness required? Why?

Please use sufficiently many parentheses to make the parsing of formulas unique (do e.g. not write `[]p->q` but write `([](p))->(q)` or write `[]((p)->(q))`; in particular always use parentheses for the bodies of temporal formulas (`[](x>0)` or `<>(x==y)`).

Check the output of Spin carefully to determine whether an error has occurred during model checking (the message `error:0` *may be even given, if the model checking has been prematurely aborted or not all of the state space has been explored). If the message "error: max search depth too small" appears, increase in the "Advanced Parameter Settings" the parameter "Maximum Search Depth". If the message "pan: reached -DMEMLIM bound" appears, increase the parameter "physical memory available".*

The deliverables of the exercise consist of

- The completed Promela model.
- Screenshots of (the final parts of) the simulation runs.
- The LTL properties (PLTL formulas plus definitions of the predicates).
- The output of Spin for each model check.
- Screenshots of counterexample simulations (if any).
- For each model check, an interpretation (did the requested property hold or not and why)?

Some hints/reminders on Promela are given below:

- The Promela version of `if (E) C1 else C2` is

  ```
  if
  :: E -> C1
  :: else -> C2
  fi
  ```

  The Promela version of `if (E) C` is

  ```
  if
  :: E -> C
  :: else -> skip
  fi
  ```

  The Promela version of `while (E) C` is

  ```
  do
  :: E -> C
  :: else -> break
  od
  ```

  In all cases, if you omit the `else` branch, the process will *block* in a state that is not allowed by all the conditions in the other branches.

- The expression c `?` `[` M `]` is true if and only if a channel c holds a message of type M. The statement c `?` M will then remove the message. A typical application is in

  ```
  do/if
  :: cond && c ? [ M ] ->
     c ? M;
     ...
  :: ...
  od/fi;
  ```

  where in a certain situation only a certain kind of message may be accepted.

- In the attached Promela model, the processes receive identifiers 1,2,3,... i.e.

  `p[1]@label`

  indicates that `p(0)` is at the position indicated by `label` (see also the simulations for the identifiers of the individual processes).

Alternative: Alternatively to the assignment given above, you may also use RISCAL to write a `shared system` model (in the style given in the lecture), take file `LeaderElection2.txt` as a starting point (it already contains 2 transitions, 5 more are needed). Run the system to elaborate its state space and make sure that the execution does not crash (show the output of the execution). Perform the checks as described above, but show the properties *for all* processes (not just processes 0 and $N-1$) using quantifiers in the LTL formulas. Analyze and interpret the results as above. As for the additional question related to fairness: the fairness constraint shall not be encoded in LTL but by annotating the corresponding transition with a suitable `fairness` clause and declaring the LTL formula as `ltl[fairness]`.