More SPIN and PROMELA. Checking Liveness

David Kendall

1 Objectives

- Make further progress with SPIN and PROMELA in modelling and analysing some simple concurrent systems
- Use PROMELA and SPIN to model and analyse simple mutex protocols with respect to liveness properties

2 Checking Liveness Properties of Mutex using SPIN

Use Promela/Spin to model and verify the algorithms below for maintaining mutually exclusive access to a critical region in a 2 process system.

You should check for the following properties:

- Mutual exclusion is preserved. i.e. at most one process is in its critical section at any one time.
- Deadlock is avoided.
- No process is permanently blocked (i.e. is waiting for an event which can never occur).
- No process suffers from starvation (i.e. it is not possible for a process which is seeking entry to its critical section to be prevented forever from entering it)

You can assume that no process will remain in its critical section forever. You can not assume that a process will ever try to enter its critical section.

Verification of these properties can be carried out with Spin by using the following techniques:

Assertions – for checking for preservation of mutual exclusion. You will need to think of an assertion (or assertions) which you can include in your program, whose truth guarantees that mutual exclusion is preserved. Use Spin to check that your assertions are not violated.

Invalid end-states – by default Spin will check for invalid end states (i.e. states from which it is not possible to make further progress, which either a) do not occur as the result of termination, or b) are not labelled as valid end states (by using an "end:" label in Promela)). The existence of an invalid end-state indicates the possibility of deadlock.

Progress – The absence of permanent blocking and/or starvation can be checked by the use progress labels. You need to identify one or more states which must be visited repeatedly in order for the system to be considered "live". Label these states with a progress label.

You should refer to the Basic SPIN Manual, if necessary, for an introduction to the use of assertions and end-labels and progress-labels.

The algorithms are described below using pseudo-code.

Concentrate initially on checking the liveness property (No starvation). Then check all properties. If you are pushed for time, then omit algorithm 3 and just do algorithms 4 and 5.

Algorithm 3 Uses a single shared variable 'turn' which is set to 1 when it is the turn of process 1 to be allowed to enter its critical section, and is set to 2 when it is the turn of process 2. Initially turn = 1.

Process 1

```
repeat
  while turn = 2 do
    /* nothing */
  end while;
  /* do critical section */
  turn = 2;
  /* do processing outside critical section */
forever
```

Process 2

```
repeat
  while turn = 1 do
    /* nothing */
  end while;
  /* do critical section */
  turn = 1;
  /* do processing outside critical section */
forever
```

Algorithm 4 An attempt to improve algorithm 2.

Process 1

```
need1 = false;
repeat
```

```
    need1 = true;
    while need2 do
      need1 = false;
      /* burn some time */
      need1 = true;
    end while;
    /* do critical section */
    need1 = false;
    /* do processing outside critical section */
  forever
```

Process 2

```
  need2 = false;
  repeat
    need2 = true;
    while need1 do
      need2 = false;
      /* burn some time */
      need2 = true;
    end while;
    /* do critical section */
    need2 = false;
    /* do processing outside critical section */
  forever
```

Algorithm 5 Uses shared variables ‘need1’, ‘need2’ and ‘turn’. Assume initially need1 = false, need2 = false and turn = 1.

Process 1

```
  repeat
    need1 = true;
    turn = 2;
    while (need2 and (turn = 2)) do
      /* nothing */
    end while
    /* do critical section */
    need1 = false;
    /* do processing outside critical section */
  forever
```

Process 2

```
  repeat
    need2 = true;
    turn = 1;
    while (need1 and (turn = 1)) do
```

```
      /* nothing */
    end while
    /* do critical section */
    need2 = false;
    /* do processing outside critical section */
  forever
```