Prof. Bernd Finkbeiner, Ph.D.
Peter Faymonville, M.Sc.
Michael Gerke, B.Sc.

Winter term 2011/2012
Problem Set 5

Verification

Please write the names of all group members on the solutions you hand in.

Problem 1: Fairness and Liveness with SPIN

Recall our exercise 2.2 (see below for a partial solution) where non-starvation of a channel-based mutual exclusion protocol was not provable. The reason was that one process could play unfair and take the token infinitely many times even though the other process was waiting for it. This kind of unfair behaviour can be ruled out by fairness assumptions.

In order to check liveness properties in SPIN, assertions are not powerful enough. Therefore, you should use either the builtin *ltl*-primitive or specify your properties as *never*-claims (i.e. a non-deterministic Büchi automaton).

- Prove non-starvation for both processes under suitable (weak or strong) fairness assumptions. Hint: You should prove LTL-formulas of the form fairness assumption $\Rightarrow$ non-starvation. The fairness assumption may use the propositions enP0, enP1 execP0, execP1 below.
- Prove that your fairness assumptions are not too strong. That is, prove that the system has fair traces.

```
mtype {TOKEN}; /* token to be passed */
  chan ch = [1] of {mtype}; /* global channel containing token */

proctype P(byte i) {
  think:  printf("MSC: P(\%d) thinking.\n", i);
  wait: ch?TOKEN;
  use:  printf("MSC: P(\%d) enters CS.\n", i);
      /* critical section */
      printf("MSC: P(\%d) leaves CS.\n", i);
  leave:  ch!TOKEN;
      goto think
}

byte pidP0, pidP1; /* process IDs of P(0) and P(1) */
init {
  atomic { ch!TOKEN; pidP0 = run P(0); pidP1 = run P(1); }
}
/* non-starvation for P(0): [] (waitP0 -> <> useP0) *
* non-starvation for P(1): [] (waitP1 -> <> useP1) */

#define useP0 (P[pidP0]@use)
#define useP1 (P[pidP1]@use)
#define waitP0 (P[pidP0]@wait)
#define waitP1 (P[pidP1]@wait)
```

```
#define enP0 (enabled(pidP0))
#define enP1 (enabled(pidP1))
#define execP0 (_last == pidP0)
#define execP1 (_last == pidP1)
```

Problem 2: A Lift System

Consider a lift system that services $N > 0$ floors numbered 0 to $N - 1$. There is a lift door at each floor with a call-button and an indicator light that signals whether or not the lift has been called. In the lift cabin there are N send-buttons (one per floor) and N indicator lights that inform which floors are requested. For simplicity, consider $N = 3$. Present a set of atomic propositions - try to minimize the number of propositions - that are needed to describe the following properties of the lift system as LTL-formulas and give the corresponding LTL-formulas:

- The doors are "safe", i.e., a Floor door is never open if the cabin is not present at the given Floor.
- A requested Floor will be served eventually.
- Again and again the lift returns to Floor 0.
- When the top Floor is requested, the lift serves it immediately and does not stop on the way there.
- The cabin is motionless unless there is some request.

Consider the elevator system from above.

- Model such a system in Promela. Your system should consist of three components: The environment controlling the call and send buttons, the lift motors moving the cabin up and down and opening/closing doors, and the controller controlling the indicator lights and driving the lift motors. Each of these components may be realized by one or more processes.
- Verify that your model is free from deadlocks (i.e., make sure that there are no invalid end states). Verify that every floor is reachable by the elevator.
- Express the following properties in LTL and verify (fairness assumptions may be needed for liveness properties):
  - The elevator does not drive beyond the top floor.
  - A floor door is never open if the cabin is not present at the given floor.
  - The cabin is motionless unless there is some request.
  - No indicator light is on without a corresponding request.
  - A requested floor will be served eventually.
  - After a request, the corresponding indicator light goes on before serving the request is completed.