CSEP590 – Problem Set 6
Summer 2003
Due: Thursday, August 14 by midnight.
Directions: Answer each of the following questions. Email your solutions in doc, pdf, or html format to evan@cs.washington.edu

Before starting this problem set, be sure to read the SPIN installation instructions posted as a link with this assignment. Once you have SPIN up and running and feel comfortable with its operation, then start on the problems.

This problem set will use SPIN to analyze and verify two mutual exclusion protocols. **IMPORTANT:** for each of the two problems below, email to the TA *all* Promela files that you use to solve the problem, along with your solution writeup. Be sure to indicate in comments at the top of each file what the file is used for. This will make it much easier and faster for Evan to grade your solutions.

1. Consider an arbitrary, but finite, number of identical processes that execute in parallel. Each process consists of a non-critical part and a critical part, usually called the *critical section*. In this problem we are concerned with the verification of a mutual exclusion protocol, that is, a protocol that should ensure that at any moment of time at most one process (among the N processes in our configuration) is in its critical section. There are many different mutual exclusion protocols developed in the literature. In this exercise we will be concerned with a particular protocol, developed in the 1980s. Assume there are N processes for some fixed value of $N > 0$. There is a global variable, referred to as *flag*, which is an array of length N, such that *flag[i]* is a value between 0 and 4 (for $0 \le i < N$). The idea is that *flag[i]* indicates the status of process i. The protocol executed by process i looks as follows:

```
l0: loop forever do
      begin
      l1: Non-critical section
      l2: flag[i] := 1;
      l3: wait until (flag[0] < 3 and flag[1] < 3 and ... and flag[N-1] < 3)
      l4: flag[i] := 3;
      l5: if (flag[0] = 1 or flag[1] = 1 or ...  or flag[N-1] = 1)
            then begin
                  l6: flag[i] := 2;
                  l7: wait until (flag[0] = 4 or flag[1] = 4 or … or flag[N-1] = 4)
            end
      l8: flag[i] := 4;
      l9: wait until (flag[0] < 2 and flag[1] < 2 and … and flag[i-1] < 2)
      l10: Critical section
      l11: wait until ((flag[i+1] ∈ {0,1,4}) and … and (flag[N-1] ∈ {0,1,4}))
      l12: flag[i] := 0;
end
```

Before doing any of the exercises listed below, try first to informally understand what the protocol is doing and why it could be correct in the sense that mutual exclusion is ensured. If you are convinced of the fact that the correctness of this protocol is not easy to see, (if not, let me know ☺) then start with the following questions.

1. Model this mutual exclusion protocol in PROMELA. Assume that all tests on the global variable *flag* (such as the one in statement l3) are *atomic*. Look carefully at the indices of the variable *flag* used in the tests. Make the protocol description modular such that the number of processes can be changed easily.
2. Check for several values of N (N ≥ 2) that the protocol indeed ensures mutual exclusion. Report your results for N ranging from 2 to the maximum value for which verification is still manageable. Produce a table indicating for each checked value of N, the size of the state vector, the number of states, the run time, and the state space (in bytes).
3. The code that a process has to go through before reaching the critical section can be divided into several segments. We refer to statement l4 as the *doorway*, to segment l5, l6, and l7, as the *waiting room* and to segment l8 through l12 (which contains the critical section) as the *inner sanctum*. Your task is to check the following basic claims using assertions. Give for each case the changes to your original PROMELA specification for this protocol and present the verification results. In case of negative results, simulate the counter-example by means of guided simulation. Assume for this part of the problem that N = 2.
    a. Whenever some process is in the inner sanctum, the doorway is locked, that is, no process is at location l4.
    b. If a process i is at l10, l11 or l12, then it has the least index of all the processes in the waiting room and the inner sanctum.
    c. If some process is at l12, then all processes in the waiting room and in the inner sanctum must have flag value 4.

2. A popular mechanism that is used in many protocols to establish mutual exclusion is the use of *semaphores*, in particular binary semaphores. A binary semaphore s is an integer variable that can only take the values 0 or 1. Once s has been given an initial value, there are two operations permitted on s: P(s) and V(s). P(s) decrements the value of s by 1, but only if s equals 1. In case the value of s equals 0, the process that wants to call P(s) on s waits (ie, blocks its execution) until s equals 1. V(s) increments the value of s by 1. Notice that V(s) may lead to the unblocking of a process that is waiting to P(s) on s.

In this problem, we will look at a mutual exclusion protocol that uses 2 semaphores, *s.true* and *s.false*. The idea behind using these two semaphores is that a process selects one of the semaphores on the basis of the value of a local Boolean variable: if the value of that variable is *true*, then it tries to acquire (ie, P(s)) the semaphore *s.true*, otherwise it tries to acquire *s.false*. Both semaphores are initially 1. Assume there are N processes, for some fixed N > 0. There are two global Boolean variables *b* and *w*, whose initial value is irrelevant. Each process i also has a local Boolean variable c_i. The program for process i looks as follows:

```
while true do
    c_i := w;
    P(s.c_i);
    if c_i = w then
        P(s.¬c_i);
        b := true;
        while b do
            b := false;
            V(s.¬c_i);
            P(s.¬c_i);
        end;
        w := ¬w;
        V(s.¬c_i);
    fi;
    Critical section;
    b := true;
    V(s.c_i);
end
```

Answer the following 2 questions concerning this protocol:

1. Model this mutual exclusion protocol in PROMELA. In particular consider the modeling of the binary semaphore carefully, and motivate your description of such semaphore.
2. Check using assertions whether the protocol indeed establishes mutual exclusion among N processes, for some fixed N (for instance 4 or 5). Clarify the necessary extensions to your original protocol description.

3. Now that you have modeled, simulated, and verified each of these protocols, do you have any better understanding of why/how they achieve mutual exclusion? If so, explain (in a few short sentences) how each protocol works. If not, explain what still confuses you. This question is of course very open-ended, so don't panic if you aren't 100% sure.