Logic Model Checking, CS 118
Fourth Assignment

Maximum score is 100 points.
Late submissions lose 10 pts, cumulatively each day at noon

Solutions are due: **Noon, Tuesday 17 February 2015**
via email to gh@caltech.edu
(plain ascii text files only)

1. (30 Points) Consider the following concurrent algorithm for controlling access to the critical section of two processes, specified informally in the following table.

A Critical Section Algorithm	
`bool wantp = false, wantq =false`	
`Process p`	`Process q`
`p1: while true`	`q1: while true`
`p2:` *`non-critical section`*	`q2:` *`non-critical section`*
`p3: wantp = true`	`q3: wantq = true`
`p4: await wantq == false`	`q4: await wantp == false`
`p5:` *`critical section`*	`q5:` *`critical section`*
`p6: wantp = false`	`q6: wantq = false`

   Each numbered line corresponds to an indivisible statement execution. The keyword "await" is used to indicate that the process will wait for the condition that follows it to become true before proceeding. Model the algorithm in Promela, and verify it with Spin.

2. (30 Points) Consider a distributed system that consists of five producers, one scheduler, and two consumers. Each producer has two tasks that need to be completed by a consumer. For each task, a producer sends a request to the scheduler. The scheduler has a queue to store up to five incoming producer requests. The scheduler, upon receiving a request, assigns the task to one of the consumers. The two consumers have their own separate queue of outstanding tasks. A consumer can only service one request at a time, but it can have up to two outstanding tasks waiting in the queue. Consumers do not acknowledge completion of tasks.

   The definitions for proctypes `Producer`, `Consumer`, and `init` are provided below. Write the definition for proctype `Scheduler` for assigning tasks to the consumers. The scheduler is constrained by the following property when assigning tasks: it must balance the workloads of the two consumers. That is, a new task is always assigned to the consumer who has the fewest number of outstanding tasks in its queue. If the number of tasks currently assigned is equal, then the task could be assigned to either consumer. Prove with Spin that your solution has the right properties.

```
mtype = { task }
chan prodToSched = [5] of {mtype, byte}
```

```
chan schedToCons[2] = [10] of {mtype, byte}

proctype Producer(byte me)
{
    prodToSched!task(me)
    prodToSched!task(me)
}

proctype Consumer(byte me)
{   byte you

    do
    :: schedToCons[me]?task(you)
    od
}

init
{   byte i
    atomic
    { for (i : 0 .. 4) {
            run Producer(i)
      }
      run Consumer(0)
      run Consumer(1)
    }
}

active proctype Scheduler ()
{  /* you write this */
}
```

3. (40 Points) Five concurrent algorithms are sketched below, each defining the behavior of two concurrent processes named p and q. There is an array of bit values called a[] with 8 elements. Assume that there is at least one integer value for which array element a[i] equals 0. Formalize each algorithm in Promela and use Spin to perform the verification. You will need to extend or modify each algorithm to prevent array indexing errors. You should also add an initial process that initializes the array a[] with nondeterministically chosen 0 or 1, while preventing that all eight array elements are set to 1. This initialization should be completed before processes p and q start executing.

   An algorithm is considered correct if for all possible scenarios, both processes *terminate* after one of them has found a zero. Each numbered line for each process corresponds to an indivisible action. For each algorithm, show that it is correct or find a counterexample. For each counterexample, explain what the problem is that causes it.

   These are the global variable declarations to be used for all five algorithms.

```
bool found = false
int i, j, turn   // turn is only used in the last two algorithms
bit a[8]         // to be initialized by you before p and q run
```

   The informal specifications of the five algorithms follows.

=== Algorithm 1

```
p0: i = 0
p1: found = false
p2: while !found {
p3:   i++
p4:   found = (a[i] == 0)
p5: }

q0: j = 1
q1: found = false
q2: while !found {
q3:   j--
q4:   found = (a[j] == 0)
q5: }
```

=== Algorithm 2

```
p0: i = 0
p1: while !found {
p2:   i++
p3:   found = (a[i] == 0)
p4: }

q0: j = 1
q1: while !found {
q2:   j--
q3:   found = (a[j] == 0)
q4: }
```

=== Algorithm 3

```
p0: i = 0
p1: while !found {
p2:   i++
p3:   if a[i] == 0
p4:     found = true
p5: }

q0: j = 1
q1: while !found {
q2:   j--
q3:   if a[j] == 0
q4:     found = true
q4: }
```

=== Algorithm 4

```
p0: i = 0
p1: while !found {
p2:   await turn == 1 -> turn = 2
p3:   i++
p4:   if a[i] == 0
```

```
p5:     found = true
p6: }

q0: j = 1
q1: while !found {
q2:   await turn == 2 -> turn = 1
q3:   j--
q4:   if a[j] == 0
q5:     found = true
q6: }
```

=== Algorithm 5

```
p0: i = 0
p1: while !found {
p2:   await turn == 1 -> turn = 2
p3:   i++
p4:   if a[i] == 0
p5:     found = true
p6: turn = 2
p7: }

q0: j = 1
q1: while !found {
q2:   await turn == 2 -> turn = 1
q3:   j--
q4:   if a[j] == 0
q5:     found = true
q6: turn = 1
q7: }
```