Concurrent Programming

Exercise Booklet 9: Promela and Spin[1]

Exercise 1

Implement the following entry/exit protocol (Attempt I) seen in class, in Promela. Use the Promela code in the slides as an aid in understanding Promela syntax.

```
global int turn = 1;

thread P: {                  thread Q: {
  while (true) {               while (true) {
     await (turn==1);             await (turn==2);
     turn = 2;                    turn = 1;
  }                             }
}                            }
```

Exercise 2

Draw the transition system of the following two programs (note: you can use SpinSpider as a guide):

```
% Program 1                  % Program 2
  byte state = 1;              byte state = 1;
  active proctype A(){         active proctype A(){
    atomic {                       (state==1) ->
      (state==1) ->                  state = state+1
        state = state+1        }
    }                          active proctype B() {
  }                               (state==1) ->
  active proctype B() {              state = state-1
    atomic {                   }
      (state==1) ->
        state = state-1
    }
  }
```

Exercise 3

Check whether the following algorithm guarantees mutual exclusion by adding an auxiliary variable `critical` and appropriate statements. Do you recognize this algorithm?

```
bool flag[2]
bool turn

active [2] proctype user()
{
        flag[_pid] = true
        turn = _pid
        (flag[1-_pid] == false || turn == 1-_pid)

crit:   skip    // critical section

        flag[_pid] = false
}
```

[1]Sources include: `http://www.cs.toronto.edu/~chechik/courses01/csc2108/lectures/spin.2up.pdf`

Exercise 4

What happens in the previous algorithm if you exchange the line `flag[_pid] = true` with the line `turn = _pid`?

Exercise 5

Consider the following simplified presentation of the Bakery Algorithm for two processes:

```
1  global int np,nq =0;
```

```
 2 thread P: {
 3   while (true) {
 4     // non-critical section
 5     np = nq + 1;
 6     await nq==0 or np<=nq;
 7     // CRITICAL SECTION
 8     np = 0;
 9     // non-critical section
10   }
11 }
```

```
 2 thread Q: {
 3   while (true) {
 4     // non-critical section
 5     nq = np + 1;
 6     await np==0 or nq<np;
 7     // CRITICAL SECTION
 8     nq = 0;
 9     // non-critical section
10   }
11 }
```

1. Encode it in Promela.

2. Show that it does not guarantee mutual exclusion.

3. Add the command `np = 1` in thread P just before line 4 and add `nq=1` in thread Q just before line 4. Show, using Spin, that the resulting program does enjoy mutual exclusion.

Exercise 6

Show that this algorithm for mutual exclusion is faulty. Produce a counterexample trace.

```
byte cnt
byte x, y, z

active [2] proctype user()
{       byte me = _pid+1                        // 1 or 2

L1:     x = me
        if
        :: (y != 0 && y != me) -> goto L1       // try again
        :: (y == 0 || y == me)
        fi
        z = me
        if
        :: (x != me)  -> goto L1                // try again
        :: (x == me)
        fi
        y = me
        if
        :: (z != me) -> goto L1                 // try again
        :: (z == me)
        fi                                      // success
        cnt++
        assert(cnt == 1)                        // critical section
        cnt--
        goto L1
}
```

Exercise 7

1. Define a general semaphore with operations `acquire(sem)` and `release(sem)`, where `sem` is a numeric shared global variable. Since there are no functions in Promela, you can declare macros using **`inline`**. Hint: use **`atomic`** and blocking expressions.

2. Check that your solution is correct by verifying, using assertions, that mutual exclusion is guaranteed in the following code:

```
byte sem = 1;

inline acquire(sem) {
  atomic {
    sem>0;
    sem--
  }
}

inline release(sem) {
  sem++
}

active [2] proctype  user() {
  do
    :: acquire(sem);
       printf("%d is in the CS\n",_pid);
       release(sem)
  od
}
```

Exercise 8

Prove that the following solution in Hydra is correct in the sense that only one man goes in for every two woman.

```
thread Man: {
    mutex.acquire();
    ticket.acquire();
    ticket.acquire();
    mutex.release();
}

thread Woman: {
    ticket.release();
}
```

Encode the solution in Promela and verify it. Spawn 20 men and 20 women.

Exercise 9

What does the following program print?

```
proctype A(chan q1) {
  chan q2;
  q1?q2;
  q2!123
}
proctype B(chan qforb) {
  int x;
```

```
  qforb?x;
  printf("x=%d\n"x)
}
init {
  chan qname = [1] of { chan };
  chan qforb = [1] of { int };
  run A(qname);
  run B(qforb);
  qname!qforb
}
```

Exercise 10

Implement a binary semaphore in Promela using message passing and synchronous communication. You should have two proctypes, `semaphore` and `user`. Here is the code for the user:

```
#define acquire 0
#define release 1
chan sema = [0] of { bit };  /* synchronous channel */
proctype semaphore() {
  /* complete this */
}
proctype user() {
  do
  :: sema?acquire;
     /* crit. sect */
     sema!release;
     /* non-crit. sect. */
  od
}
init {
  run semaphore();
  run user();
  run user();
}
```