Concurrent Programming

Exercise Booklet 9: Promela and Spin[1]

Solutions to selected exercises (◊) are provided at the end of this document. Important: You should first try solving them before looking at the solutions. You will otherwise learn **nothing**.

1 Basic Promela

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.

```
int turn = 1;

Thread.start { // P
  while (true) {
     await (turn==1);
     turn = 2;
  }
}

Thread.start { // Q
  while (true) {
     await (turn==2);
     turn = 1;
  }
}
```

Exercise 2.

Draw the transition system of the following two programs separately and then compare them (note: you can use SpinSpider as a guide):

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

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

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

```
8       (flag[1-_pid] == false || turn == 1-_pid)

10  crit:   skip    // critical section

12      flag[_pid] = false
    }
```

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:

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

```
Thread.start { // P
  while (true) {
    // non-critical section
    np = nq + 1;
    await nq==0 or np<=nq;
    // CRITICAL SECTION
    np = 0;
    // non-critical section
  }
}
```

```
Thread.start { // Q
  while (true) {
    // non-critical section
    nq = np + 1;
    await np==0 or nq<np;
    // CRITICAL SECTION
    nq = 0;
    // non-critical section
  }
}
```

1. Encode it in Promela. Note that the assignment has to be split into two operations in Promela to reflect that it is not atomic.
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. The following extension to 3 processes of Dekker's algorithm is known to deadlock.

```
global int turn=0;
global bool[] flags = {false, false, false};

thread (id): {
  int left = (id+2)%3;
  int right = (id+1)%3;
  flags[id] = true;
  while (flags[left] || flags[right])
    if (turn == left) {
      flags[id] = false;
      await (turn==id);
      flags[id] = true;
    }
  }
  // Critical Section
  turn = right;
  flags[id]=false;
}
```

Encode it in Promela and verify where it deadlocks (there are multiple traces that lead to deadlock, exhibiting one suffices). Here is some code that you can start from.

```
byte turn;
bool flags[3];


proctype P() {
  byte myId = _pid-1;
  /* complete here */
}

init {
  turn=0;
  byte i;
  for(i:0..2) { flags[i] = false; }
  atomic {
    for (i:0..2) { run P(); }
  }
}
```

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 providing an example of its use.

Exercise 8. Implement the Bar exercise from eb5, in Promela, using the semaphore encodings from Exercise. 7. Use Spin to show that the required invariant is upheld, i.e. that only one Jets fan goes in for every Patriot fans, by inserting an appropriate assertion. In your example use 20 Jets fans and 20 Patriot fans.

2 Channels

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();
}
```

Exercise 11. Implement the turnstile example from class using channels. Here is some code to get you started:

```
mtype { bump , read  };
chan counter = [0] of { mtype, chan }
chan reply[2] = [0] of { byte }

proctype Counter() {
   /* complete */
}

proctype Turnstile() {
  /* complete */
}

init {
  byte x;
  atomic {
    run Counter();
    run Turnstile();
    run Turnstile();
  }
  _nr_pr==2;
  counter!read(reply[0]);
  reply[0]?x;
  printf("%d\n",x) /* should print 20 */
}
```

3 Solutions to Selected Exercises

Answer to exercise 1

```
byte turn=1;
```

```
active proctype  P() {
  do
    :: turn==1;
       printf("P went in \n");
       turn = 2
  od
}

active proctype Q() {
  do
    :: turn==2;
       printf("Q went in \n");
       turn = 1
       od
}
```

Answer to exercise 7

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

Answer to exercise 9

- Init sends qforb to A on qname
- A sends 123 to qforb
- B receives 123 on qforb and prints it

Answer to exercise 10

```
#define acquire 0
#define release 1
chan sema = [0] of { bit };
proctype semaphore() {
  byte count = 1;
  do
  :: (count == 1) -> sema!acquire; count = 0
  :: (count == 0) -> sema?release; count = 1
  od
}

proctype user() {
  do
  :: sema?acquire;
     /* crit. sect */
    sema!release;
     /* non-crit. sect. */
  od
}
init {
```

```
21    run semaphore();
      run user();
23    run user();
}
```