Solutions for CP Exercises, Week 4

1. Solution for Andrews Ex. 3.3

(a)

```
var l : integer := 1;

process P[i : 1..n] =
  var r : integer := 0;
  repeat
    nc1:  non-critical section_i;
          repeat
              Swap(r, l);
          until r = 1;
    cs1:  critical section_i;
          Swap(r, l);
  forever;
```

We are now going to prove that the above solution does ensure mutual exclusion.

First, we assume that the local variables r are renamed to global variables r_i $(i = 1..n)$ that are all initialized to 0;

Next, we prove some auxiliary invariants:

$$\begin{array}{lcll} F_i & \triangleq & in\ cs_i \Rightarrow r_i = 1 & i = 1..n \\ G & \triangleq & l \in 0, 1 & \\ H_i & \triangleq & r_i \in 0, 1 & i = 1..n \end{array}$$

Since r_i is changed only in P_i, F_i is a local invariant By induction, H_i and G are easily seen to be invariants since 0 and 1 are the only values being swapped around.

Now we define

$$I \triangleq r_1 + r_2 + \ldots + r_n + l = 1$$

This holds intially and since any of the variables are changed only by atomic swapping of two of them, their sum will remain constant. Therefore, I is an invariant of the program.

Now, if two of the processes P_i and P_j $(j \neq i)$ should be in their critical sections at the same time, F_i and together with G and H_i would give us

$$r_1 + r_2 + \ldots + r_n + l \geq 2$$

contradicting the invariant I. Thus, we conclude that this cannot be the case, ie. the algorithm ensures mutual exclusion.

If two or more processes execute $Swap(r_i, l)$ at the same time, one of the will get the "token" first and thereby obtain access to the region. Which of them it is not determined. Thus, the algorithm cannot deadlock nor livlock, but it is *not fair*. Starvation can occur if other process manages to enter the region inbetween a given process attempts to execute $Swap(r_i, l)$.

(b) To avoid memory contention by writing to l, its value may be checked before an attempt is made to change it with *Swap*:

```
repeat
    while l = 0 do skip;
    Swap(r, l);
until r = 1;
```

This will not effect the proof in (a).

(c) *Not included*

2. Solution for Andrews Ex. 3.2

```
var l : integer := 1;

process P[i : 1..n] =
  var s : integer;
  repeat
    non-critical section_i;
    DEC(l, s);
    while s > 0 do {
      INC(l, s);
      delay;
      DEC(l, s);
    }
    critical section_i;
    INC(l, s);
  forever;
```

Here, the lock l is used as in the test-and-set solution. However, if the lock is already "set" ($l < 1$), the effect of *DEC* must be undone by *INC*, before trying again. The correctness argument (or proof) follows the same line as for the test-and-set solution.