Solutions for CP Exercise Class 7

1. Solution for Andrews Ex. 8.14

```
module Account
  op Deposit(d : posinteger);
  op Withdraw(w : posinteger);
body

  var balance : integer;

  process AccountServer =
    repeat
      in Deposit(d)                          → balance := balance + d
      [] Withdraw(w) and w ≤ balance → balance := balance − w
      ni
    forever;

end Account;
```

2. Solution for Andrews Ex. 8.15

(a)

```
module ABmeeting
  op MeetA();
  op MeetB();
body

  process AccountServer =
    repeat
      in MeetA() →
        in MeetB() → skip ni;
        in MeetB() → skip ni;
      ni
    forever;

end ABmeeting;
```

(b)

```
module ABmeeting
  op MeetA();
  op MeetB();
body

  process AccountServer =
    repeat
      in MeetA() →
        in MeetB() →
          in MeetB() → skip ni
        ni
      ni
    forever;

end ABmeeting;
```

3.

```
module Event
  op Pass();
  op Clear(var r : integer);
  op Release(v : posinteger);
body

  var S : integer;

  process EventServer =
    repeat
      in Pass() and S > 0 → skip
      [] Clear(var r)     → r := S;  S := 0
      [] Release(v)       → S := S + v;
                            for i in 1..?Pass do
                              in Pass() → skip ni
      ni
    forever;

end Event;
```

[The loop in the *Release* branch ensures that all current calls of *Pass* are processed before a possible call of *Clear* as in the monitor version.]

4. The semaphore operation P(s) is implemented by:

```
var l : integer;
repeat
  e.Pass;
  e.Clear(l)
until l > 0;
if l > 1 then e.Release(l − 1)
```

[The call of *Pass* ensures that the semaphore value is not tested (with *Clear*) until it is known to have been positive. Hereby a busy wait is reduced to a semi-busy one being much less resource demanding.]

5. Solution for Andrews Ex. 8.12

If we can assume that a guard using the function that gives the number of pending operation calls is re-eavaluated whenever a call is made, we can do with:

```
module Barrier
  op arrive();
body

  process Control =
    var nr : integer := 0;
    repeat
      in arrive() and ?arrive >= n → for i in 1..n − 1 do
                                       in arrive() → skip ni
      ni
    forever;

end Barrier;
```

If the guard is not reevaluated, we can instead nest n accepts within each other by recursion:

```
module Barrier
  op arrive();
body

  procedure meet(k : integer)
    in arrive() → if k > 1 then meet(k − 1) ni

  process Control =
    var nr : integer := 0;
    repeat
      meet(n)
    forever;

end Barrier;
```

In Ada neither of these solutions are possible since the *count*-attribute is not reevaluated and **accept**-statements can occur only in the main loop of a task (not within procedures). Instead the *requeue* facility must be used.