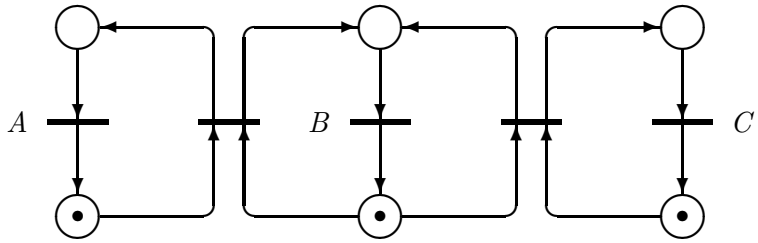
Solutions for Exercises, Week 10

1. Solution for Exam June 1994, Problem 3

Question 3.1

Before each round, P_2 must synchronize with *either* P_1 or P_3. A Petri-net expressing this is:



Question 3.2

[A synchronization can be implemented by signalling forth and back using two semaphores. A choice between two synchronizations is then implemented by using a common semaphore for the signalling:]

var S_{AC}, S_B : *semaphore* := 0;

process P_A =	**process** P_B =	**process** P_C =
repeat	**repeat**	**repeat**
wait(S_{AC});	*signal*(S_{AC});	*wait*(S_{AC});
signal(S_B);	*wait*(S_B);	*signal*(S_B);
A	B	C
forever;	**forever**;	**forever**;

[Alternatively, *wait* and *signal* may be exchanged in all three processes.]

2. Solution for Andrews Ex. 8.9

(a) A solution giving priority to writers is shown in [Andrews p.388] using the facility to query the number pending calls of an operation *op* to block readers when there are pending writers.

(b) A fair solution can be obtained from the above solution by explicitly processing the readers inbetween the writers. Using our notation, we get:

```
module ReadersWriters
  op read(var T);
  op write(T);
body

  op startread();
  op endread();

  var val : T;

  proc read(var r : T)
    startread();
    r := val;
    endread()

  process Writer =
    var nr : integer := 0;
    repeat
      in startread() and ?write = 0 → nr := nr + 1
      [] endread()                  → nr := nr − 1
      [] write(v : T) and nr = 0    → val := v
                                      while ?startread > 0 do
                                        in startread() → nr := nr + 1 ni
      ni
    forever;

end ReadersWriters;
```

Although extra readers may slip through while the reader group is started in the *write* branch, they cannot recur at the *startread* queue as they cannot pass *endread*. So for a finite number of readers, the *startread* queue will eventually be emptied.

If only the readers waiting when the write has ended should be started, the **while** loop may be replaced by:

```
for i in 1..?startread do
  in startread() → nr := nr + 1 ni
```

3. Solution for Andrews Ex. 8.12

Assuming 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 such size-dependent guards were not reevaluated, we could 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;
```

Aside: 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.

[Solutions for Exam Dec. 2018 and Dec. 2020 will appear on the material page.]