1. We want to design an asynchronous process `Split` that is the dual of `Merge`. The process `Split` has one input channel `in` and two output channels `out_1` and `out_2`. The messages received on the input channel should be routed to one of the output channels in a nondeterministic manner so that all possible splittings of the input stream are feasible executions. Describe all the components of the desired process `Split`.

Now suppose we want to capture the assumption that the distribution of messages among the two output channels should, while unspecified, fair in the sense that if infinitely many messages arrive on the input channel `in`, then both output channels `out_1` and `out_2` should have infinitely many messages transmitted. How would you add fairness assumptions to your design to capture this? If you are using strong fairness, then argue that weak fairness would not be enough (that is, describe an infinite execution that is weakly fair but the split of messages is not fair as desired).

2. Exercise 4.3 from the textbook.

3. Recall Peterson’s protocol that solves the mutual exclusion problem for two processes. Consider the following protocol that attempts to solve the mutual exclusion problem to resolve contention among three processes using two “copies” of the protocol: the first copy is used to resolve contention between `P_1` and `P_2`, and their winner then competes with `P_3` in the second copy using a separate set of shared registers.

More specifically, the protocol uses shared atomic registers `flag_1`, `flag_2`, `turn`, `flag'_1`, `flag'_2`, and `turn'` (all `flag` values are 0 initially). Processes `P_1` and `P_2` execute steps of Peterson’s protocol using the shared variables `flag_1`, `flag_2`, and `turn`. But when this protocol allows entry into critical section, now they execute steps of another copy of the protocol using the shared variables `flag'_1`, `flag'_2`, and `turn'`, where both `P_1` and `P_2` use the identifier 1 (that is, first set `flag'_1` to 1 and then set `turn'` to 1). Process `P_3` does not use the first copy, and executes the sequence of steps of the second copy of the protocol using the shared variables `flag'_1`, `flag'_2`, and `turn'` with the identifier 2 (that is, first set `flag'_2` to 1 and then set `turn'` to 2). When the second copy allows an entry to the critical section, then the corresponding process enters the real critical section. Upon leaving the critical section, the process needs to set the appropriate flag in the second copy to 0 and in case of `P_1` and `P_2`, also reset the appropriate flag of the first copy back to 0.

(a) Does the above three process protocol satisfy the requirement of mutual exclusion? Justify your answer.

(b) Does the above three process protocol satisfy the requirement of deadlock freedom (that is, if a process wants to enter critical section, then some process is allowed to enter the critical section)? Justify your answer.

4. Consider an asynchronous process `P` with two variables `x` and `y`, both of type `nat` and both initialized to 0. The behavior of the process is described by three tasks. The task `A_1` is always enabled and its update code is `x := x + 1`. The task `A_2` is always enabled and its update code is `x := 0`. The task `A_3` has the guard condition `x = 0` and the update code `y := y + 1`.

For each of the requirements below, state whether the process satisfies the requirement. If not, is there a suitable fairness assumption under which the requirement is satisfied. When adding fairness assumptions, clearly specify whether you are using strong fairness, or weak fairness, and for which tasks. Justify your answers with a brief explanation.

(a) The value of `x` eventually exceeds 0.

(b) The value of `x` eventually exceeds 5.

(c) The value of `y` eventually exceeds 5.

5. Exercise 4.19 from the textbook.