Network Algorithms: Exercise 4
Consensus

Dr. Stefan Schmid, Arne Ludwig, Srivatsan Ravi, Lalith Suresh

1 Impossibility of Consensus under Failures (15 + 15 + 25 = 55 points)

Recall that a *register* is an object that exports a *read* and *write* operation. Oversimplify, for this exercise, assume that read and write events are atomic in the sense that they take zero time, and no two events happen at the same time (they are "linearized"). Consider a model (Consensus #2 from the lecture) where n processes with $n > 1$ and atomic registers are trying to reach a consenus in a wait-free manner, and where either none or one process may fail. The proof for 2 processes and 2 registers for this model was covered in the lecture (see also Hagit Attiya and Jennifer Welch, "Distributed Computing Fundamentals, Simulations, and Advanced Topics"). Give a similar proof for 3 processes under the following guidelines:

a) Show that there exists a bivalent initial configuration for (Consensus #2) with 3 processes and binary inputs when either none or one process may fail.

b) Argue why there must exist a critical configuration.

c) Show that **a)** and **b)** leads to a contradiction regarding consensus.

2 Consensus with Queues (15 + 30 = 45 points)

Assume that you can use atomic queues in addition to the atomic registers in order to find consensus on one of the input values of 2 processes. The queue offers atomic *enque* and atomic *deque* operations. Suppose that up to 1 process may fail.

a) Assume that you are already given an initialized queue with a winner and a loser ball in it. Present an algorithm that is using this queue and atomic registers in order to solve consensus for 2 processes where up to 1 process may fail.

b) Assume now that you do not have this initialized queue, but rather you have *two* initially empty queues. Take a look at the (incomplete) algorithm $propose_i(v)$. Each process initially has a private input value v. In the algorithm, each process first enques a winner and then a loser ball in "its queue". Complete the algorithm with the notation of the function $func_i(Q_j, R_j, v)$ so that $propose_i(v)$ reaches consensus even if up to 1 process may fail. Make sure that you go through all the different executions and check whether they give a correct result.

Algorithm 1: $propose_i(v)$

Input: $v \in V, i \in \{1, 2\}$; processes p_i; empty queues Q_i; register array of size 2 $R_i[1, 2]$; initially false flags $flag_i$

```
1 Q_i.enq("winner");
2 Q_i.enq("loser");
3 flag_i.write(true);
4 result_i = v;
5 for j = 1, 2 do
6     if flag_j == true then
7         result_i ← func_i(Q_j, R_j, result_i);
8 return result_i
```