Prof. Christian Cachin
Jovana Mićić

Solution for exercise 9

9.1 Flooding Consensus (3pt)

Consider *Flooding Uniform Consensus* [CGR11, Algorithm 5.3], in which every process always uses N rounds before deciding.

a) Can one reduce the number of rounds for some process? Consider a system with two processes p and q.

b) Is this algorithm correct if the failure detector is not perfect ($\mathcal{P}$) but is only the eventually perfect one ($\Diamond\mathcal{P}$)? Justify your answer.

Solution.

a) The answer is no. In the case of two processes, the algorithm needs two communication steps because a decision cannot be reached by all correct processes after one step. Consider a system with two processes p and q and assume that every process must decide after executing only one round in the algorithm. We describe an execution where the processes do not reach uniform agreement; thus, the algorithm needs at least two rounds. Suppose that p and q propose two different values v and w, respectively. Without loss of generality, assume that $v < w$. In the following execution, process p is faulty. During the first round, p and q send their values to each other. Process p receives its own value and q's value and decides at the end of the round by our assumption. It decides v, the smaller value, and then crashes. Now, assume that the message from p to q in round one is delayed arbitrarily. There is a time after which q detects p to have crashed because of the strong completeness property of the perfect failure detector. As q cannot know that p actually did send a message, q reaches the end of the first round and must decide. Process q decides its own value w, which violates *uniform agreement*. Note that in the original algorithm, where the processes decide only after two rounds, the above scenario cannot occur. This is because p crashes before it can decide (in fact, it never decides); later on, q decides w.

b) A violation of *strong completeness* property of the perfect failure detector could lead to the violation of the *termination* property of consensus as follows. There is an execution in which a process p waits to deliver a message from a process q or to detect the crash of process q. Should q crash and p never detect the crash of q, p would remain blocked forever and never decide. Consider now *strong accuracy*. If it does not hold, it could violate the *agreement* property. It can happen that if process q crashes after deciding x, and p is falsely suspected to have crashed by processes r and q, then r and q will decide y.

9.2 Leader-Driven Consensus (4pt)

Study the *Leader-Driven Consensus* [CGR11, Chap. 5.3], where epoch-change and epoch consensus are implemented as described.

a) Why does this algorithm require a majority of correct processes?

b) Which property is violated if there is no majority of correct processes?

c) Draw or describe an execution with four processes p, q, r, s that justifies your previous answers.

Solution. We explain this for the case of a system of four processes p, q, r, and s. Assume by contradiction that the algorithm tolerates the crash of two processes. Assume that p and q propose a value v, whereas r and s propose a different value v'. Consider an execution (Figure 1) where p and q crash initially: in this execution, r and s decide v' to respect the *validity* property of consensus. However, processes r and s cannot distinguish this execution form one where p and q continue to run and eventually decide, but never receive any message from r and s (this execution must terminate to satisfy consensus). Then, p and q decide v by the assumption on the protocol and violate the *agreement* property. With a majority of correct processes, the issue does not occur.

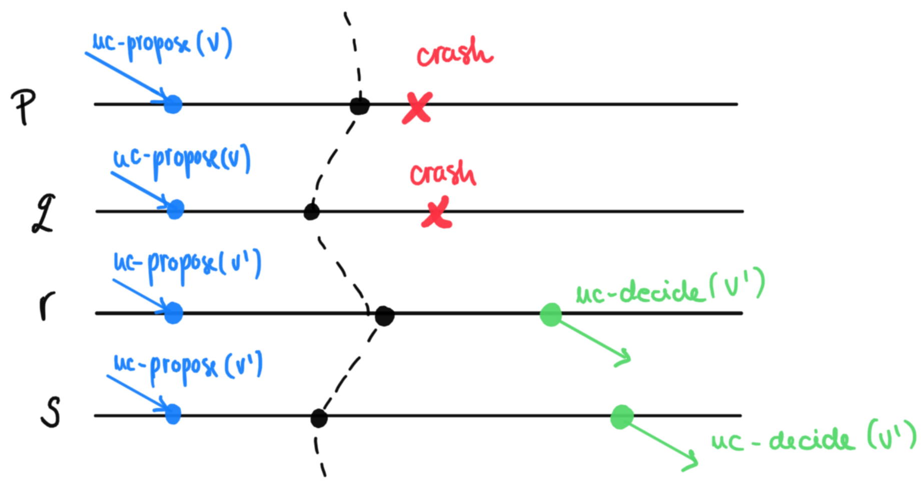

Figure 1. Example of an execution of *Leader-Driven Consensus* algorithm.

9.3 Leader-Driven Consensus, optimized (3pt)

In practice, *Leader-Driven Consensus* [CGR11, Chap. 5.3] protocol is often implemented with slightly fewer messages. This optimiziation assumes that all processes start with the same process ℓ_0 as the first leader and that ℓ_0 is correct. Describe why the first epoch consensus instance may omit the first round of message exchange (of the READ and STATE messages) between ℓ_0 and the other processes.

Solution. It may omit the first round of message exchanges for reading because in the initial epoch, process ℓ_0 knows that no decision could have been made in a previous epoch (as there is no previous epoch). This first round in every epoch consensus instance is actually only needed to make sure that the leader will propose a value that might have been decided (more precisely, to ensure the lock-in property of epoch consensus). The algorithm, therefore, saves one communication phase by directly having ℓ_0 write v_ℓ and all correct processes decide after three communication steps.

References

[CGR11] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming (Second Edition)*, Springer, 2011.