Distributed Algorithms

Consensus - solutions
4th exercise session

Matteo Monti <matteo.monti@epfl.ch>
Jovan Komatovic <jovan.komatovic@epfl.ch>

Exercise 1
(Consensus & Perfect failure detector)

Consider our fail-stop consensus algorithms (Consensus Algorithm I and Consensus Algorithm II). Explain why none of those algorithms would be correct if the failure detector turned out not to be perfect.

Solution 1 - Algorithm I

We consider the case where one correct process ρ mistakenly triggers a $<crash, \pi>$ event for some correct process π.

In the case where $id(\pi) = 0$, $id(\rho) = 1$, π proposes A, ρ proposes B:

- π broadcasts its proposal A and immediately decides for A.
- ρ believes that π crashed, and broadcasts and decides B.

This breaks the agreement property of consensus.

Solution 1 - Algorithm II

We consider the case where one faulty process ρ mistakenly triggers a $\langle crash, \pi \rangle$ event for some correct process π.

In the case where $N = 2$, $id(\pi) = 0$, $id(\rho) = 1$, π proposes A, ρ proposes B:

- π broadcasts its proposal A.
- ρ believes that π crashed, and broadcasts its proposal B.

Solution 1 - Algorithm II

- *ρ* delivers its own proposal *B*. Since *N* = *2*, *ρ* decides *B*.
- *ρ* crashes before *π* gets its proposal. *π* correctly triggers <*crash, ρ*> and, since *N* = *2*, decides *A*.

This breaks the uniform agreement property of uniform consensus.

Exercise 2
(Consensus & Eventually perfect failure detector)

Explain why any fail-noisy consensus algorithm (one that uses an eventually perfect failure detector ◇P) actually solves uniform consensus (and not only the non-uniform variant).

Solution 2

- We consider an algorithm that uses an eventually perfect failure detector. By contradiction, we assume that the algorithm satisfies agreement, but not uniform agreement. We consider two executions A and B of the algorithm.
- In A, two processes π and ρ decide differently, and π crashes. Let t denote the time when ρ decides.
- In B, π does not crash, but every process that suspects π in A also suspects π in B at the same moment of its execution. No process that suspects π restores π before t. All messages from π are delayed: none of its messages is delivered before t.

Solution 2

- It is easy to see that ρ receives exactly the same messages and indications from the failure detector in A and B, and thus decides differently from π also in B.
- However, in B, π never failed. Therefore, if the algorithm violates uniform agreement, it also violates agreement.

Exercise 3
(Consensus & Correct majority)

Explain why any fail-noisy consensus algorithm (one that uses an eventually perfect failure detector ◇P) requires a majority of the processes to be correct. More precisely, provide a "bad run" in the case where the majority of processes is faulty.

Solution 3

Consider a system with an even number *N* of processes. Let *A*, *B* denote two distinct subsets of *N/2* processes that propose values *A* and *B* respectively. By contradiction, let us assume that an algorithm exists that achieves consensus when *N/2* processes fail. The two following executions are valid:

- Execution 1. All processes in *A* crash at the beginning without issuing any message. All the processes in *B* still achieve consensus and, by validity, decide *B*. Let T_B denote the time when the last process decides on a value.
- Execution 2. All processes in *B* crash at the beginning without issuing any message. All the processes in *A* still achieve consensus and, by validity, decide *A*. Let T_A denote the time when the last process decides on a value.

Solution 3

Let us now consider the following Execution 3. All the processes in *A* are suspected by each process in *B*, and vice versa (at the same time as Executions 1 and 2, respectively). No message between a process in *A* and a process in *B* is delivered before $max(T_A, T_B)$. No process restores any process in the other group before $max(T_A, T_B)$.

It is immediate to see that a process in *A* cannot distinguish between Executions 2 and 3. Similarly, a process in *B* cannot distinguish between Executions 1 and 3. Therefore, all processes in *A* decide *A*, all processes in *B* decide *B*, and agreement is violated.