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.

Sequential Objects

A sequential object is a tuple $T = (Q, q_0, O, R, \Delta)$, where:

- Q is a set of *states*.
- $q_0 \in Q$ is an initial state.
- O is a set of operations.
- R is a set of responses.
- $\Delta \subseteq (Q \times \Pi \times O) \times (Q \times R)$ is a relation that associates a state, a process, and an operation to a set of possible new states and responses.

Processes invoke operations on the object. As a result, they get responses back, and the state of the object is updated to a new value, following from Δ.

Guided Exercise 4
(Asset Transfer Object)

Define a sequential object representing Asset Transfer, i.e., an object that allows processes to exchange *units of currency.*

Solution 4
(Asset Transfer Object)

- The set of states Q is the set of all possible maps $q : \mathcal{A} \to \mathbb{N}$. Intuitively, each state of the object assigns each account its *balance*.
- The initialization map $q_0 : \mathcal{A} \to \mathbb{N}$ assigns the initial balance to each account.
- Operations and responses of the type are defined as $O = \{transfer(a, b, x) : a, b \in \mathcal{A}, x \in \mathbb{N}\} \cup \{read(a) : a \in \mathcal{A}\}$ and $R = \{true, false\} \cup \mathbb{N}$.
- For a state $q \in Q$, a proces $p \in \Pi$, an operation $o \in O$, a response $r \in R$ and a new state $q' \in Q$, the tuple $(q, p, o, q', r) \in \Delta$ if and only if one of the following conditions is satisfied:
  - $o = transfer(a, b, x) \wedge p \in \mu(a) \wedge q(a) \geq x \wedge q'(a) = q(a) - x \wedge q'(b) = q(b) + x \wedge \forall c \in \mathcal{A} \setminus \{a, b\} : q'(c) = q(c)$ (all other accounts unchanged) $\wedge\ r = true$;
  - $o = transfer(a, b, x) \wedge (p \notin \mu(a) \vee q(a) < x) \wedge q' = q \wedge r = false$;
  - $o = read(a) \wedge q = q' \wedge r = q(a)$.

Bonus Exercise 5
(Total Order & Asset Transfer)

Use Total Order Broadcast to implement an Asset Transfer sequential object.

Solution 5

- Every process initializes a *balance[]* array with the initial, agreed upon balances of each process.
- Upon requesting a payment, a process TO-broadcasts a message *[PAY, source, recipient, amount]*.
- Upon TO-delivering a message *[PAY, source, recipient, amount]*, a process verifies if *balance[source]* is at least *amount*. If so, it subtracts *amount* from *balance[source]* and adds it to *balance[recipient]*.

Since every process receives the same sequence of operations, the outcome of each operation is the same at every process. Correct processes successfully issue operations and agree on the balance of every process at any point in time.