

Question 1. In the paper “A simple bivalency proof that t -resilient consensus requires $t + 1$ rounds” (available through the course web page), the following result is shown:

Theorem. Consider a synchronous round-based system \mathcal{S} with n processes and at most t crash failures *such that at most one process crashes in each round*. If $n > t + 1$ then there is no algorithm that solves consensus in t rounds in \mathcal{S} .

(a) Prove that the same theorem holds if we replace the word “consensus” by “weak consensus”, where the weak consensus problem is the weaker version of consensus with the validity property replaced by

- **Nontriviality:** There is an execution in which a correct process decides 0 and there is an execution in which a correct process decides 1.

You need only describe in detail those parts of your proof that are different from the proof given in the above-cited paper.

(b) Consider a synchronous round-based system \mathcal{S}' with n processes and at most t crash failures. Unlike the system \mathcal{S} in the previously stated theorem, in \mathcal{S}' we do *not* restrict failures to occur at most one per round.

- (i) Give a 2-round algorithm that solves weak consensus in \mathcal{S}' .
- (ii) Is there a 2-round algorithm that solves consensus in \mathcal{S}' ? Justify your answer.

Question 2. Components of a distributed system occasionally (though infrequently) experience failures. In practice, however, in most runs of an algorithm there are no failures. Thus, it makes sense to design fault-tolerant algorithms that are optimised for the *failure-free runs*. In this problem you will prove a lower bound on the number of messages in the failure-free runs of binary TRB protocols that tolerate arbitrary failures.

Let \mathcal{P} be any t -tolerant n -process *binary* TRB algorithm (i.e., where the sender’s input message may be 0 or 1) for *arbitrary failures* in the synchronous round model. Let R_v denote the failure-free run in which sender’s initial message is v ($v \in \{0, 1\}$). Let n_v denote the number of messages sent in R_v . Prove that $n_0 + n_1 \geq \lceil \frac{n(t+1)}{2} \rceil$.

Hint: Given an algorithm for binary TRB in the synchronous round model for arbitrary failures, define an undirected graph G as follows: G has n nodes, one for each process. There is an edge between p and q in G if and only if processes p and q exchange at least one message (it does not matter whether the message is from p to q or from q to p) in at least one of runs R_0 and R_1 . Prove that G is $t + 1$ connected (i.e., the removal of *any* t nodes does not cause the graph G to be divided into two or more disconnected components). Use this fact to prove the desired result.

It turns out that this lower bound is tight: there is a binary TRB algorithm for arbitrary failures in the synchronous model that uses a total of just $\lceil \frac{n(t+1)}{2} \rceil$ messages in the two failure-free runs. You don’t have to prove this here.

Question 3. Suppose we want to take a consistent snapshot of the state of the n processes of an asynchronous system (we don’t care about recording the state of the channels). Assume that there are no failures and all the channels are FIFO. Pierre and Paul propose two different algorithms for doing this. These snapshot algorithms, which use markers as in Chandy-Lamport’s algorithm, are as follows:

- (a) In Pierre’s algorithm: (i) when a process receives the marker for the *first time*, it sends the marker to all its neighbours; (ii) when a process receives the marker *from all its neighbours*, it records its local state (i.e., it takes a snapshot of its own state).
- (b) In Paul’s algorithm, the processes are named $0, 1, \dots, n - 1$ and are placed around a unidirectional “virtual ring”: the successor of process i is process $(i + 1) \bmod n$. When a process i receives the marker for the *first time*, in one atomic action, it records its local state and sends the marker to $(i + 1) \bmod n$. Note that the ring is used *only* for the messages of the snapshot algorithm. In the underlying distributed system there is a direct link between any two processes.

As with Chandy-Lamport, assume that: (1) there is only one process that wants to take a global snapshot, and (2) it starts the algorithm by simulating the receipt of a marker from “god”.

For each of these two algorithms, say whether it works, and briefly justify your answer.

Question 4. Consider an asynchronous system \mathcal{S} with crash failures. Let \mathcal{A} be *any* algorithm that solves consensus in \mathcal{S} when processes have access to the *eventually perfect* failure detector $\diamond P$. (Recall that $\diamond P$ satisfies the following properties: (i) eventually each crashed process is permanently suspected by each correct process; and (ii) there is a time after which no correct process is suspected by any correct process.)

- (a) Giancarlo Giannini says that \mathcal{A} must solve *uniform* consensus. Is he right? Briefly justify your answer.
- (b) Let D be a failure detector that returns an *arbitrary* list of suspects. Thus, we cannot make any assumption on the quality of D ’s output.

Suppose we execute algorithm \mathcal{A} with failure detector D (instead of $\diamond P$). Francesco Giacobazzi says that \mathcal{A} may now violate the agreement property of consensus. Is he right? Briefly justify your answer.

- (c) Revisit the above two questions in case \mathcal{A} is an arbitrary algorithm that solves consensus in \mathcal{S} when processes have access to the *perfect* failure detector P (instead of the eventually perfect failure detector $\diamond P$). (Recall that P satisfies the following two properties: (i) eventually each crashed process is permanently suspected by each correct process, and (ii) no process is suspected before it crashes.)

Question 5. Let \mathcal{S} be an asynchronous system with n processes, t of which may crash, and where links are reliable. Assume $n > 3t$. Consider the following implementation of a failure detector in \mathcal{S} .

- Every process p periodically:
 - (i) Sends an “are you alive?” message to every process in the system.
 - (ii) Waits to receive an “I am alive” message from $n - t$ distinct processes.
 - (iii) Suspects the t processes from which it did not receive the “I am alive” message; this list of suspects is used until the next period.
- Every process that receives “are you alive?” from a process q , replies with “I am alive” to q .

- (a) Does this failure detector satisfy the *strong completeness* property? Briefly justify your answer.
- (b) Does it satisfy the following property: At each time τ and for each correct process p , a majority of the correct processes is not suspected by p ? Briefly justify your answer.
- (c) Can one transform this failure detector into $\diamond S$ in system \mathcal{S} ? If so, give an algorithm that effects this transformation and justify its correctness. If not, explain why.

Question 6. A single-writer, multi-reader register R is *less-than-safe* if it satisfies the following property:

- If a read operation on R is not concurrent with *any* other operation on R then it returns the value last written into R .

A less-than-safe register is weaker than a safe register, where a read operation returns the value last written as long as it is not concurrent with any *write* operation on the register.

Lamport showed that it is possible to solve the mutual exclusion problem in an asynchronous shared-memory system using only safe registers. Is it possible to do so using only less-than-safe registers? Justify your answer.

Question 7. Shown below is a proposed mutual exclusion algorithm for the asynchronous model where processes communicate via shared registers with write and read operations. Prove or disprove:

- (a) The algorithm satisfies mutual exclusion and deadlock freedom if the shared registers are atomic.
- (b) The algorithm satisfies mutual exclusion and deadlock freedom if the shared registers are safe.

communication var:

control _{i} : integer in range 0..2 initially 0

local var:

j : integer in range 1.. N

```
1.  repeat
2.      NCS          % non-critical section %
3.      L0: control $i$  := 1
4.      L1: for  $j := 1$  to  $i - 1$  do
5.          if control $j$   $\neq 0$  then goto L1
6.          control $i$  := 2
7.          for  $j := 1$  to  $N$  do
8.              if  $j \neq i$  and control $j$  = 2 then goto L0
9.          CS          % critical section %
10.         control $i$  := 0
11.  forever
```