EPFL

X

Teacher: Professor Rachid Guerraoui
Exam: CS-451 - Distributed Algorithms
Exam date: December 19th, 2023

X

SCIPER : **X** Signature :

Time Limit 3:00 hours.

Instructions

- This exam is closed-book: no notes, electronics, nor cheat sheets are allowed.
- Read through each problem before starting to solve it.
- Whenever presenting an algorithm, be as precise as possible, and preferentially use the pseudo-code notation used in class.
- When solving a problem, do not assume any known result from the lectures, unless it is explicitly stated that you might use some known result.

Important (read with care!)

Unless a problem explicitly states otherwise:

- Assume that communication links are **perfect**.
- Assume that **all** processes can fail.
- Assume that processes only fail by **crashing**.
- **Do not** assume that a **Failure Detector** is available.

The maximum score for this exam is 7. The grade for this exam is equal to the score achieved, capped at 6.
This means that a perfect score is not necessary to achieve the maximum grade!

Good luck!

Question 1 - *Interfaces and properties*

Properties Menu What follows is list of interface events (marked with letters) and liveness and safety properties (marked with numbers). Some apply to distributed abstractions we have seen in class, some do not. Use letters and numbers to fill in the questions below. For example:

x. *(0 points)* State the interface and properties of Example Broadcast.

Interface	B	C	L			
Properties	3	4	8	12		

Important: The answer box has room for up to six interface events and six properties, but that does not mean all abstractions have six events and six properties! Leave the cells you don't need empty, just like in the example above.

Nice to have: Please list interfaces and properties in ascending order if you can. Obviously, if your list is correct but out of order, your answer will still be considered 100% correct!

Interface events Pick your interface events from the list below.

A. **(Request)** `Broadcast(message)` - Broadcasts message `message`.

B. **(Request)** `Broadcast(message, destination)` - Broadcasts message `message` to process `destination`.

C. **(Request)** `Broadcast(message, source)` - Broadcasts message `message` from process `source`.

D. **(Request)** `Crash(process)` - Crashes process `process`.

E. **(Request)** `Propose(value)` - Proposes value `value`.

F. **(Request)** `Propose(values)` - Proposes a set of values `values`.

G. **(Indication)** `Crash(process)` - Indicates that process `process` crashed.

H. **(Indication)** `Decide(proposer, value)` - Decides value `value`, as proposed by process `proposer`.

I. **(Indication)** `Decide(value)` - Decides value `value`.

J. **(Indication)** `Deliver(message)` - Delivers message `message`.

K. **(Indication)** `Deliver(source, message)` - Delivers message `message`, broadcast by `source`.

L. **(Indication)** `Restore(process)` - Retracts a previously indicated suspicion of process `process` having crashed.

M. **(Indication)** `Restore` - Retracts a previously indicated suspicion of the local process having crashed.

N. **(Indication)** `Suspect(process)` - Suspects process `process` of having crashed.

O. **(Indication)** `Suspect` - Suspects the local process of having crashed.

Safety and liveness properties Pick your properties from the list below.

1. **Abort-validity**: If any process decides 0, then all processes crashed.
2. **Abort-validity**: If any process decides 0, then some process crashed or proposed 0.
3. **Agreement**: If any correct process delivers a message m, then eventually every correct process delivers m.
4. **Agreement**: If any correct process delivers a message m, then eventually every process delivers m.
5. **Causal delivery**: A message is received only if it was sent.
6. **Causal delivery**: If a process p delivers a message m', then p previously delivered every message m such that the sender of m is the same as m'.

7. **Causal order**: If a process p delivers a message m', then p previously delivered every message m such that $m \to m'$ (i.e., m *causally precedes* m'). For all m, m' we have $m \to m'$ if and only if: *(FIFO order)* some process broadcast m before m'; *(Local order)* some process delivered m before broadcasting m'; or *(Transitivity)* some message $\bar{m}$ exists such that $m \to \bar{m}$ and $\bar{m} \to m'$.
8. **Causal order**: If a process p delivers a message m', then p previously delivered every message m such that $m \to m'$ (i.e., m *causally precedes* m'). For all m, m' we have $m \to m'$ if and only if: *(FIFO order)* some process broadcast m' before m; *(Local order)* some process delivered m' before broadcasting m; or *(Transitivity)* some message $\bar{m}$ exists such that $m \leftarrow \bar{m}$ and $\bar{m} \to m'$.
9. **Commit-validity**: If any process decides 1, then all processes are correct.
10. **Commit-validity**: If any process decides 1, then all processes proposed 1.
11. **Eventual strong accuracy**: Eventually, no correct process is suspected by any correct process.
12. **Integrity**: All processes decide simultaneously.
13. **Integrity**: No process decides twice.
14. **Integrity**: No process decides.
15. **No creation**: If process p delivers message m from process q, then q broadcast m.
16. **No creation**: If process p delivers message m from process q, then q is correct and broadcast m.
17. **No duplication**: No correct process delivers the same message more than once.
18. **No duplication**: No process delivers the same message more than once.
19. **No duplication**: No two processes deliver the same message.
20. **Reliable delivery**: If a correct process p sends a message m to a correct process q, then eventually q delivers m from p.
21. **Reliable delivery**: If a correct process p sends a message m to a process q, then eventually q delivers m from p.
22. **Strong completeness**: Every crashed process is eventually permanently suspected by every correct process.
23. **Termination**: Every correct process eventually crashes.
24. **Termination**: Every correct process eventually decides.
25. **Termination**: Every correct process eventually proposes.
26. **Termination**: Every correct process eventually terminates.
27. **Total order**: An enumeration $p_1, \ldots, p_N$ exists of all correct processes such that, for all $i < j$, all messages from p_i are delivered before all messages from p_j.
28. **Total order**: Let m, m' be messages. If any correct process delivers m without having delivered m', then no correct process delivers m' before m.
29. **Total order**: Let p, q be processes. If p proposes before q, then p decides before q.
30. **Uniform agreement**: If any process delivers a message m, then eventually every correct process delivers m.
31. **Uniform agreement**: If any process delivers a message m, then eventually every process delivers m.
32. **Uniform agreement**: No two processes decide different values.
33. **Uniform agreement**: No two processes propose different values.
34. **Validity**: If a process decides a value v, then v was proposed by a correct process.
35. **Validity**: If a process decides a value v, then v was proposed by all correct processes.
36. **Validity**: If a process decides a value v, then v was proposed by some process.
37. **Validity**: If processes p and q are correct, then every message broadcast by p is eventually delivered by q.
38. **Validity**: Let p, q be processes. Every message broadcast by p is eventually delivered by q.
39. **Validity**: Let p, q be processes. If p is correct, then every message broadcast by p is eventually delivered by q.
40. **Validity**: Let p, q be processes. If q is correct, then every message broadcast by p is eventually delivered by q.

Questions

a. *(0.2 points)* State the interface and properties of Best-Effort Broadcast (BEB).

Interface						
Properties						

b. *(0.2 points)* State the interface and properties of Reliable Broadcast (RB).

Interface						
Properties						

c. *(0.2 points)* State the interface and properties of Uniform Reliable Broadcast (URB).

Interface						
Properties						

d. *(0.2 points)* State the interface and properties of Uniform Causal Broadcast (UCB).

Interface						
Properties						

e. *(0.2 points)* State the interface and properties of Total Order Broadcast (TOB).

Interface						
Properties						

f. *(0.2 points)* State the interface and properties of Uniform Consensus (UC).

Interface						
Properties						

g. *(0.2 points)* State the interface and properties of Non-Blocking Atomic Commit (NBAC).

Interface						
Properties						

h. *(0.2 points)* State the interface and properties of the Eventually Perfect Failure Detector ($\diamond\mathcal{P}$).

Interface						
Properties						

Question 2 *Multiple-choice questions*

Select the correct answer. Each question has exactly one correct answer.

a. *(0.1 points)* Assuming that no process can crash, can a Perfect Failure Detector be implemented in the asynchronous setting?

- ☐ Yes.
- ☐ No.

b. *(0.1 points)* Can a Perfect Failure Detector be implemented in a synchronous system?

- ☐ Yes.
- ☐ No.

c. *(0.1 points)* Can Terminating Reliable Broadcast be implemented using an Eventually Perfect Failure Detector?

- ☐ Yes.
- ☐ No.

d. *(0.1 points)* Can Uniform Consensus be implemented in a setting where every process is guaranteed to eventually crash?

- ☐ Yes.
- ☐ No.

e. *(0.2 points)* Is it possible to implement Consensus using Total Order Broadcast?

- ☐ No. Unlike Total Order Broadcast, Consensus cannot be implemented without a Perfect Failure Detector.
- ☐ No. Both Total Order Broadcast and Consensus only allow for asynchronous implementations.
- ☐ Yes. Every process broadcasts its proposal. Upon delivering the proposal of every other process, a correct process decides the most represented proposal.
- ☐ Yes. Every process broadcasts its proposal. Upon delivering the first value v, a correct process decides v.
- ☐ Yes. Every process broadcasts its identifier. The process whose identifier is first delivered by all correct processes is the leader. The leader sends its proposal v to all processes. Upon receiving v from the leader, a process delivers v.
- ☐ Yes. Total-Order Broadcast can be used to implement a Perfect Failure Detector, which can be used to implement Consensus.

f. *(0.2 points)* Let Knock-Off Best-Effort Broadcast be a variant of Best-Effort Broadcast that does not provide Validity. What is the implementation of Knock-Off Best-Effort Broadcast that minimizes the number of messages sent by a broadcasting process?

- ☐ Trick question. Without Validity, Best-Effort Broadcast cannot be implemented.
- ☐ Trick question. Without Validity, Best-Effort Broadcast can be implemented only using a Perfect Failure Detector.
- ☐ The implementation of Best-Effort Broadcast we saw in class also solves Knock-Off Best-Effort Broadcast, all while minimizing the number of messages sent by a broadcasting process.
- ☐ Upon broadcasting, a process does not send out any message. No process ever delivers any message.
- ☐ Upon broadcasting a message m, a process sends m only to itself, then delivers m.

g. *(0.3 points)* Let Knock-Off Consensus be a variant of Consensus that does not provide Agreement. What is the implementation of Knock-Off Consensus that minimizes the total number of messages sent by all processes?

- ☐ Trick question. Without Agreement, Knock-Off Consensus cannot implement a Perfect Failure detector, which makes it impossible to implement Total-Order Broadcast, which is required to solve Consensus in the first place.
- ☐ Trick question. Knock-Off Consensus is not weaker than Consensus, so the implementation of Consensus we saw in class is already the one that minimizes the total number of messages sent by all processes.
- ☐ No process sends any message. Every process decides the constant value “0".
- ☐ No process sends any message. Each process decides its own proposal.
- ☐ No process sends any message. No process ever decides.
- ☐ No process sends any message. Every process sequentially decides every finite string of bits, thus eventually ensuring Validity.

h. *(0.3 points)* Let N denote the number of processes in a system. What is the minimum number of correct processes required for a wait-free, asynchronous implementation of a regular register to be possible? (In what follows, assume that N is divisible by 6).

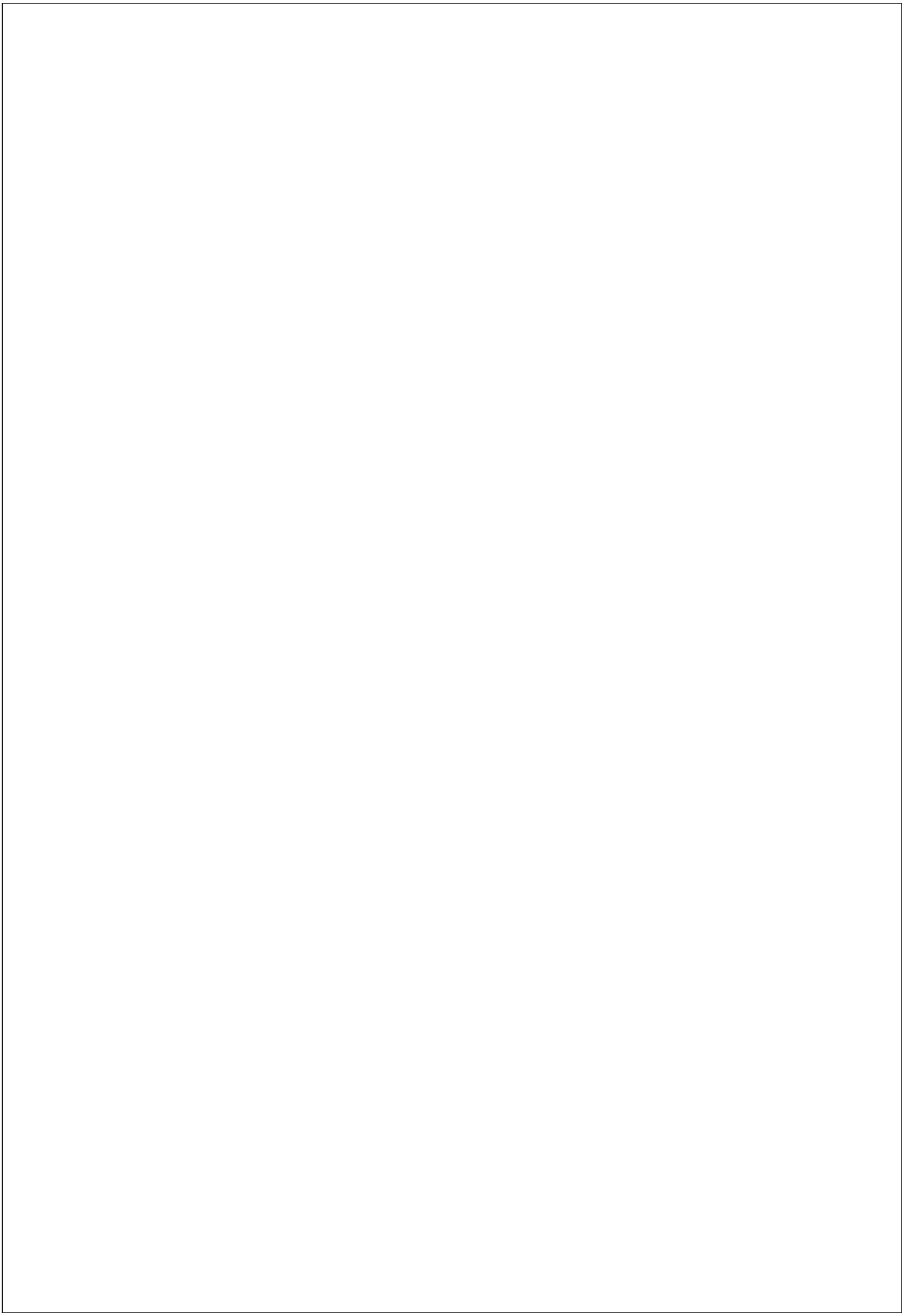
- ☐ 1
- ☐ N / 3
- ☐ N / 3 + 1
- ☐ N / 2
- ☐ N / 2 + 1
- ☐ N - 1

Question 3 *Consensus and Total-Order Broadcast*

a. *(0.5 points)* Implement Consensus using a Perfect Failure Detector and Best Effort Broadcast. Prove the correctness of your implementation.

b. *(0.5 points)* Implement Total-Order Broadcast using Reliable Broadcast and (one or more instances of) Consensus. Prove the correctness of your implementation.

Please answer inside the box!



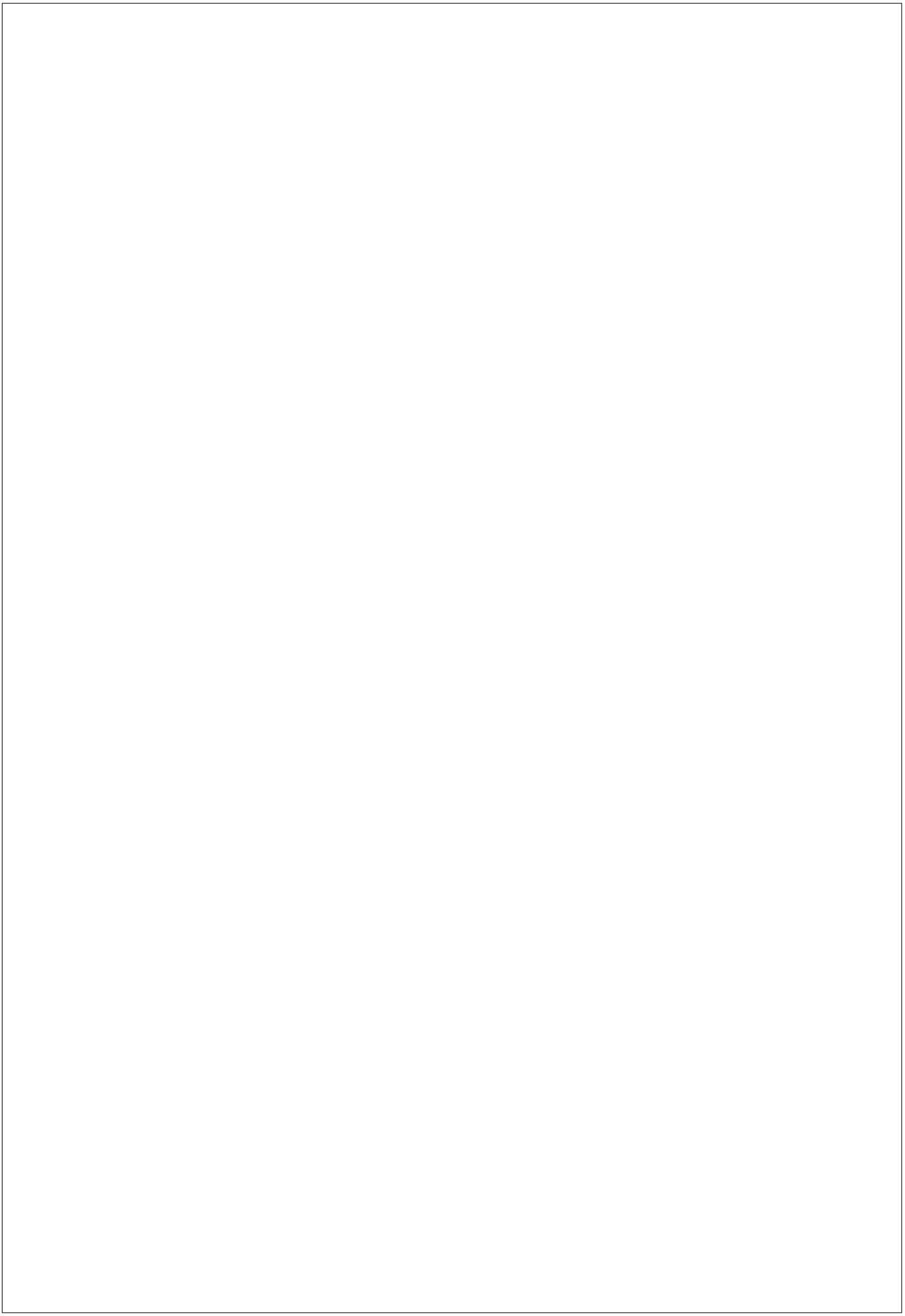
Question 4 *Non-Blocking Atomic Commit and Consensus*

---

a. *(0.75 points)* Prove that, if one process can fail, Non-Blocking Atomic Commit cannot be implemented unless a Perfect Failure Detector is available (or can be implemented).

b. *(0.75 points)* Prove that any algorithm which solves Consensus using an Eventually Perfect Failure Detector also solves Uniform Consensus.

Please answer inside the box!



Question 5 *Binary Consensus*

Binary Consensus The Binary Consensus abstraction is identical to Consensus, but the only values that can be proposed are 0 and 1. More formally, Binary Consensus exposes the following interface events:

- **(Request)** `Propose(bit)` - Proposes value `bit`, with `bit` $\in$ `{0, 1}`.
- **(Indication)** `Decide(bit)` - Decides value `bit`.

The properties of Binary Consensus are the same as Consensus.

---

a. *(1.5 points)* Consider a system of N processes. Processes have access to:

- N instances of Binary Consensus (each process can propose a value to each instance)
- One instance of Uniform Reliable Broadcast (each process can use the instance to broadcast any number of messages)

Implement Consensus (the multi-valued version we have seen in class). Prove the correctness of your algorithm.

Please answer inside the box!