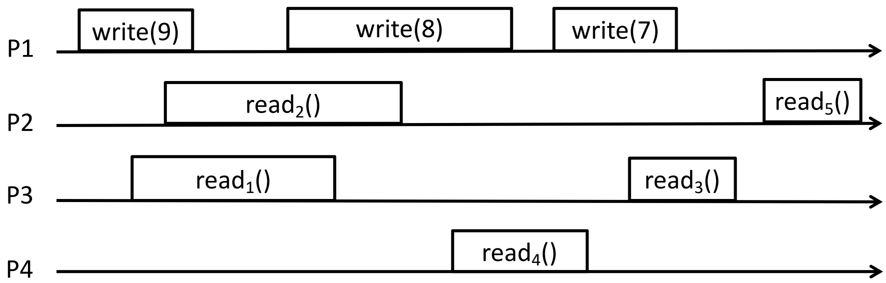
**Dependable Distributed Systems**
**Master of Science in Engineering in Computer Science**

**AA 2024/2025**

**Week 8 – Exercises**
**November 20h, 2024**

**Ex 1:** Consider the partial execution depicted in the Figure




Answer to the following questions:

1. Define <u>ALL</u> the values that can be returned by read operations (Rx) assuming that the run refers to a regular register.
2. Define <u>ALL</u> the values that can be returned by read operations (Rx) assuming that the run refers to an atomic register.
3. Assign to each read operations (Rx) a return value that makes the execution linearizable.

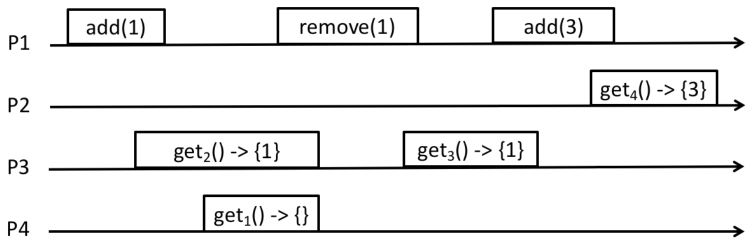
**Ex 2:** Consider a distributed system composed of *n* processes $\prod=\{p_1, p_2\dots p_n\}$ with unique identifiers that exchange messages through perfect point-to-point links. $\prod$ is not known to the processes. Processes are connected through a directed ring (i.e., each process $p_i$ can exchange messages only with processes and $p_{i+1(\text{mod } n)}$). Processes may crash and each process is equipped with a local perfect oracle (having the interface *new_next(p)*) reporting a new neighbor when the previous one is failing.
Write the pseudo-code of an algorithm implementing a Leader Election primitive at every process $p_i$.

**Ex 3:** Consider a set object that can be accessed by a set of processes. Processes may invoke the following operations on the object:

- add(v): it adds the value v in to the set
- remove(v) it removes the value v from the set
- get(): it returns the content of the set.

Informally, every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove().
For the sake of simplicity, assume that a value can be added/removed just once in the execution.

Consider the distributed execution depicted in the Figure




Answer to the following questions:

1. Is the proposed execution linearizable? Motivate your answer with examples.
2. Consider now the following property: "every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove(). If an add(v)/remove(v) operation is concurrent with the get, the value v may or may be not returned by the get()".
   Provide an execution that satisfy get validity and that is not linearizable.

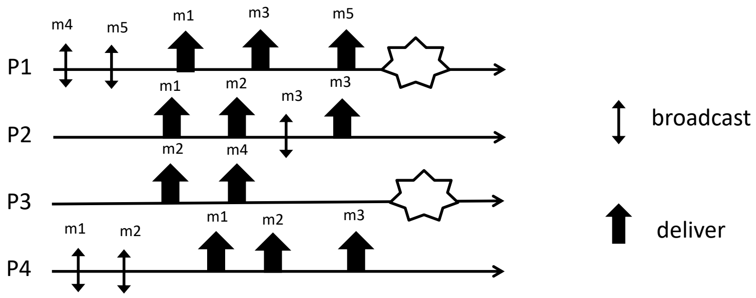
**Ex 4:** Consider the algorithm shown in the Figure

| | |
|---|---|
| **upon event** ⟨ Init ⟩ **do**<br>    *delivered* := ∅; *pending* := ∅; *correct* := Π;<br>    **forall** m **do** *ack*[m] := ∅;<br><br>**upon event** ⟨ *urb*, Broadcast \| m ⟩ **do**<br>    *pending* := *pending* ∪ {(*self*, m)};<br>    **trigger** ⟨ *beb*, Broadcast \| [DATA, *self*, m] ⟩;<br><br>**upon event** ⟨ *beb*, Deliver \| p, [DATA, s, m] ⟩ **do**<br>    *ack*[m] := *ack*[m] ∪ {p};<br>    **if** (s, m)/∈ *pending* **then**<br>      *pending* := *pending* ∪ {(s, m)};<br>      **trigger** ⟨ *beb*, Broadcast \| [DATA, s, m] ⟩; | **upon event** ⟨◊P,Suspect \|p⟩**do**<br>    *correct* := *correct* \ {p};<br><br>**upon event** ⟨◊P,Restore \|p⟩**do**<br>    *correct* := *correct* ∪ {p};<br><br>**function** candeliver(m) **returns** Boolean **is**<br>    **return** (*correct* ⊆ *ack*[m]);<br><br>**upon exists** (s, m) ∈ *pending* such that candeliver(m) **do**<br>    *delivered* := *delivered* ∪ {m};<br>    **trigger** ⟨ *urb*, Deliver \| s, m ⟩; |

Assuming that the algorithm is using a Best Effort Broadcast primitive and an Eventually Perfect Failure Detector ◊P discuss if the following properties are satisfied or not and motivate your answer

- *Validity*: If a correct process p broadcasts a message m, then p eventually delivers m.
- *No duplication*: No message is delivered more than once.
- *No creation*: If a process delivers a message m with sender s, then m was previously broadcast by process s.
- *Uniform agreement*: If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

**Ex 5:** Consider the execution depicted in the Figure

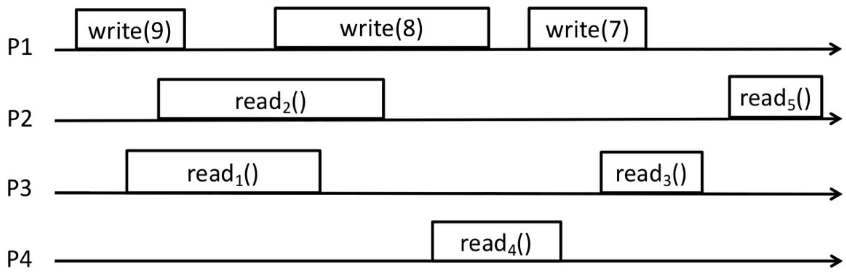



Answer to the following questions:

1. Which is the strongest TO specification satisfied by the proposed run? Motivate your answer.
2. Does the proposed execution satisfy Causal order Broadcast, FIFO Order Broadcast or none of them?
3. Modify the execution in order to satisfy TO(UA, WUTO) but not TO(UA, SUTO).
4. Modify the execution in order to satisfy TO(NUA, WNUTO) but not TO(UA, WNUTO).

**NOTE**: In order to solve point 3 and point 4 you can only add messages and/or failures.

**Ex 1:** Consider the partial execution depicted in the Figure




Answer to the following questions:

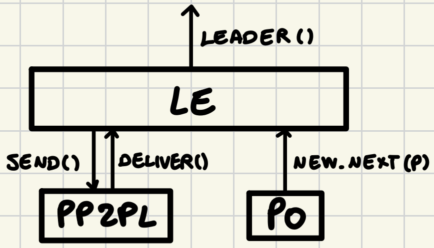
1. Define ALL the values that can be returned by read operations (Rx) assuming that the run refers to a regular register.
2. Define ALL the values that can be returned by read operations (Rx) assuming that the run refers to an atomic register.
3. Assign to each read operations (Rx) a return value that makes the execution linearizable.

1) $R_1()$: 0, 9, 8
$R_2()$: 0, 9, 8
$R_3()$. 8, 7
$R_4()$: 9, 8, 7
$R_5()$: 7

2) $R_1()$: 0, 9, 8
$R_2()$: 0, 9, 8
$R_4()$: IF $R_2() \rightarrow$ 0,9 THEN $R_4() \rightarrow$ 9,8,7
IF $R_2() \rightarrow$ 0,9,8 THEN $R_4() \rightarrow$ 8,7
SAME WITH $R_1()$ BECAUSE $R_1()$ AND $R_2()$ ARE CONCURRENT.
$R_3()$. IF $R_4() \rightarrow$ 9,8 THEN $R_3() \rightarrow$ 8,7
IF $R_4() \rightarrow$ 9,8,7 THEN $R_3() \rightarrow$ 7
$R_5()$: 7

3) W(9), $R_1(9)$, $R_2(9)$, W(8), $R_4(8)$, W(7), $R_3(7)$, $R_5(7)$

**Ex 2:** Consider a distributed system composed of $n$ processes $\prod=\{p_1, p_2\ldots p_n\}$ with unique identifiers that exchange messages through perfect point-to-point links. $\prod$ is not known to the processes. Processes are connected through a directed ring (i.e., each process $p_i$ can exchange messages only with processes and $p_{i+1(\text{mod } n)}$). Processes may crash and each process is equipped with a local perfect oracle (having the interface *new_next(p)*) reporting a new neighbor when the previous one is failing.
Write the pseudo-code of an algorithm implementing a Leader Election primitive at every process $p_i$.



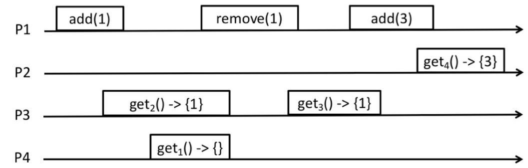

```
UPON EVENT < LE, INIT > DO
    NEXT = Pi+1 (MOD n)
    LEADER = ⊥
    SUSPECTED = ∅
    IF SELF = P1 THEN
        LEADER = SELF
        TRIGGER < PP2PL, SEND (LEADER, SELF) > TO NEXT

UPON EVENT < PO, NEW.NEXT (Pi) > DO
    FAIL = NEXT
    NEXT = Pi
    SUSPECTED = SUSPECTED ∪ {FAIL}
    IF FAIL = LEADER THEN
        LEADER = NEXT
        TRIGGER < PP2PL, SEND (LEADER, LEADER) > TO NEXT
    TRIGGER < PP2PL, SEND (CRASH, FAIL) > TO NEXT

UPON EVENT < PP2PL, DELIVER (CRASH, Pi) >
    SUSPECTED = SUSPECTED ∪ {Pi}
    TRIGGER < PP2PL, SEND (CRASH, Pi) > TO NEXT

UPON EVENT < PP2PL, DELIVER (LEADER, Pi) >
    LEADER = Pi
    TRIGGER < LE, LEADER (Pi) >
    TRIGGER < PP2PL, SEND (LEADER, Pi) > TO NEXT
```

**Ex 3:** Consider a set object that can be accessed by a set of processes. Processes may invoke the following operations on the object:

- add(v): it adds the value v in to the set
- remove(v) it removes the value v from the set
- get(): it returns the content of the set.

Informally, every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove().
For the sake of simplicity, assume that a value can be added/removed just once in the execution.

Consider the distributed execution depicted in the Figure




Answer to the following questions:

1. Is the proposed execution linearizable? Motivate your answer with examples.
2. Consider now the following property: "every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove(). If an add(v)/remove(v) operation is concurrent with the get, the value v may or may be not returned by the get()".
   Provide an execution that satisfy get validity and that is not linearizable.

1) NO BECAUSE:

ADD(1), $GET_2() \to \{1\}$, REMOVE(1), $GET_1() \to \{\}$, $GET_3() \to \{1\}$, ADD(3), $GET_4() \to \{3\}$

$GET_3() \to \{1\}$ IS NOT LEGAL BECAUSE IT DOESN'T RESPECT THE SEQUENCIAL SPECIFICATION OF THE SET. AFTER REMOVE(1) WE CANNOT GET $\{1\}$.

2) THE ORIGINAL SEQUENCE PROVIDE WHAT WE ARE LOOKING FOR:

ADD(1), $GET_2() \to \{1\}$, REMOVE(1), $GET_1() \to \{\}$, $GET_3() \to \{1\}$, ADD(3), $GET_4() \to \{3\}$

**Ex 4:** Consider the algorithm shown in the Figure

| | |
|---|---|
| **upon event** ⟨ Init ⟩ **do**<br>*delivered* := ∅; *pending* := ∅; *correct* := Π;<br>**forall** m **do** *ack*[m] := ∅;<br><br>**upon event** ⟨ *urb*, Broadcast \| m ⟩ **do**<br>*pending* := *pending* ∪ {(*self*, m)};<br>**trigger** ⟨ *beb*, Broadcast \| [DATA, *self*, m] ⟩;<br><br>**upon event** ⟨ *beb*, Deliver \| p, [DATA, s, m] ⟩ **do**<br>*ack*[m] := *ack*[m] ∪ {p};<br>**if** (s, m)∉ *pending* **then**<br>*pending* := *pending* ∪ {(s, m)};<br>**trigger** ⟨ *beb*, Broadcast \| [DATA, s, m] ⟩; | **upon event** ⟨◊P,Suspect \|p⟩**do**<br>*correct* := *correct* \ {p};<br><br>**upon event** ⟨◊P,Restore \|p⟩**do**<br>*correct* := *correct* ∪ {p};<br><br>**function** candeliver(m) **returns** Boolean **is**<br>**return** (*correct* ⊆ *ack*[m]);<br><br>**upon exists** (s, m) ∈ *pending* such that candeliver(m) **do**<br>*delivered* := *delivered* ∪ {m};<br>**trigger** ⟨ *urb*, Deliver \| s, m ⟩; |

Assuming that the algorithm is using a Best Effort Broadcast primitive and an Eventually Perfect Failure Detector ◊P discuss if the following properties are satisfied or not and motivate your answer

- *Validity*: If a correct process p broadcasts a message m, then p eventually delivers m.
- *No duplication*: No message is delivered more than once.
- *No creation*: If a process delivers a message m with sender s, then m was previously broadcast by process s.
- *Uniform agreement*: If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

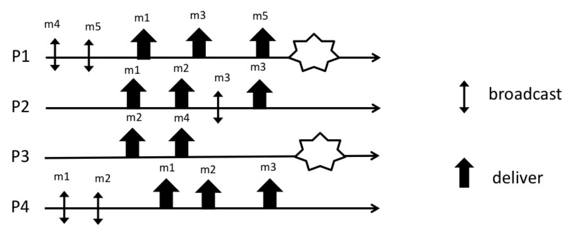
VALIDITY: SATISFIED BECAUSE WHEN WE BROADCAST A MSG, P IS ADDED IN PENDING. THE MSG IS DELIVERED ONLY WHEN ALL CORRECT PROCESS SENT THE ACK.

NO DUPLICATION. NOT SATISFIED BECAUSE THERE ISN'T "m ∉ DELIVERED" IN THE LAST "UPON EXIST...". CAN HAPPEN THAT A SUSPECTED PROCESS IS RESTORED AND THE MSG IS DELIVERED AGAIN.

NO CREATION: SATISFIED BECAUSE PENDING STORES (S, m), WHERE m IS THE MSG AND S THE PROCESS THAT WANTS DELIVER m.

UNIF AGREE: NOT SATISFIED BECAUSE A SUSPECTED PROCESS THAT IS RESTORED MAY NEVER HAVE SENT A MESSAGE WHEN IT WAS SUSPECTED.

**Ex 5:** Consider the execution depicted in the Figure




Answer to the following questions:

1. Which is the strongest TO specification satisfied by the proposed run? Motivate your answer.
2. Does the proposed execution satisfy Causal order Broadcast, FIFO Order Broadcast or none of them?
3. Modify the execution in order to satisfy TO(UA, WUTO) but not TO(UA, SUTO).
4. Modify the execution in order to satisfy TO(NUA, WNUTO) but not TO(UA, WNUTO).

**NOTE**: In order to solve point 3 and point 4 you can only add messages and/or failures.

1) TO (NUA, WUTO)

NUA: BECAUSE THE FAULTY PROCESSES DELIVER MESSAGES THAT CORRECT PROCESSES DON'T DELIVER ($m_4$, $m_5$).

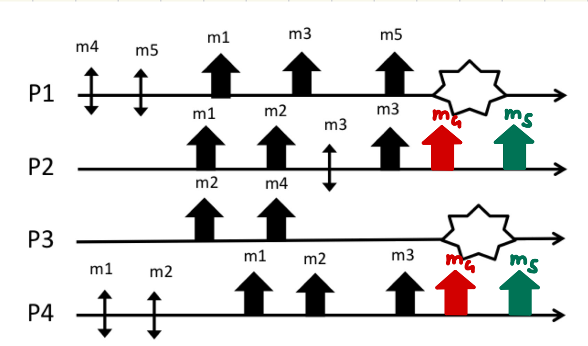
WUTO: BECAUSE THE MESSAGES THAT BOTH CORRECT AND FAULTY DELIVERS ARE IN THE COMMON ORDER.

2) CASUAL ORDER: FIFO + LOCAL ORDER

$m_2 \rightarrow m_3$ LOCAL
$m_1 \rightarrow m_2$ FIFO
$m_4 \rightarrow m_5$ FIFO

BOTH ARE SATISFIED IN THE CORRECT PROCESSES.

3) TO (UA, WUTO) BUT NO TO (UA, SUTO)




4) TO (NUA, WNUTO) BUT NO TO (UA, WNUTO)