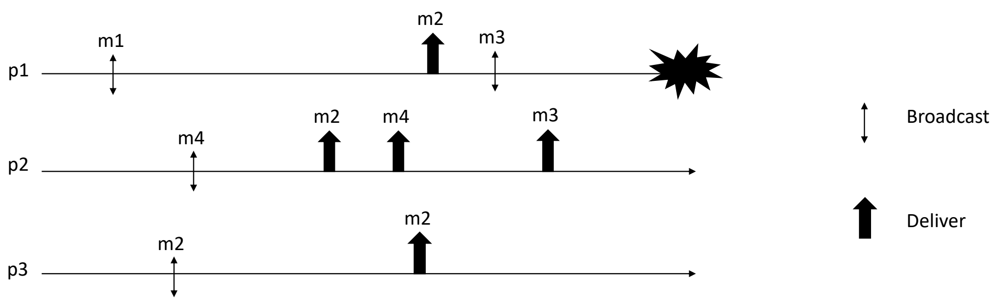
Distributed Systems (9 CFU)
05/07/2022

Family Name______________________Name__________________Student ID____________

Ex 1: Provide the specification of the (1, N) Regular Register and describe the majority voting algorithm discussed during the lectures.

Ex 2: Consider the message pattern shown in the Figure

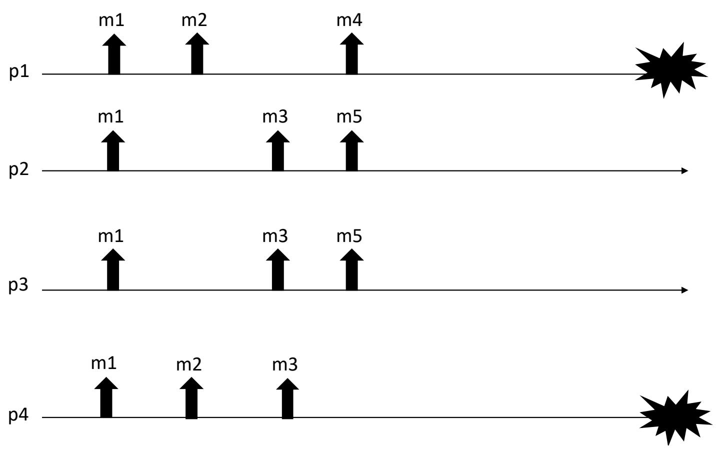


Answer to the following questions:

1. Complete the partial execution in order to obtain a run satisfying Uniform Reliable Broadcast
2. Complete the partial execution in order to obtain a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast
3. Complete the partial execution in order to obtain a run satisfying Best Effort Broadcast but not Regular Reliable Broadcast
4. List **ALL** the possible sequences satisfying both causal order and total order

NOTE: To solve the exercise you can just add deliveries of messages and new broadcast (if needed)

Ex 3: Consider the execution depicted in the Figure



Answer to the following questions:

1. Which is the strongest Total Order specification satisfied by the proposed run? Provide your answer by specifying both the agreement and the ordering property.
2. Modify the run in order to obtain an execution satisfying TO (UA, WUTO) but not TO (UA, SUTO)

3. Modify the run in order to obtain an execution satisfying TO (NUA, WNUTO) but not TO(NUA, WUTO).

NOTE: To solve the exercise you can just add deliveries of messages.

Ex 4: Let us consider a Regular Reliable Broadcast primitive satisfying the following properties:

- *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*.
- *Agreement:* If a message *m* is delivered by some correct process, then *m* is eventually delivered by every correct process.

Let us consider a distributed system composed of N processes executing the Eager algorithm (reported in figure)

Algorithm 3.3: Eager Reliable Broadcast

Implements:
    ReliableBroadcast, **instance** *rb*.

Uses:
    BestEffortBroadcast, **instance** *beb*.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
    delivered := $\emptyset$;

upon event $\langle$ *rb*, *Broadcast* $\mid m$ $\rangle$ **do**
    trigger $\langle$ *beb*, *Broadcast* $\mid$ [DATA, *self*, m] $\rangle$;

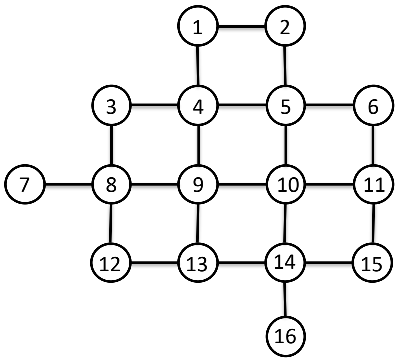
upon event $\langle$ *beb*, *Deliver* $\mid p$, [DATA, s, m] $\rangle$ **do**
    if $m \notin$ *delivered* **then**
        delivered := *delivered* $\cup \{m\}$;
        trigger $\langle$ *rb*, *Deliver* $\mid s, m$ $\rangle$;
        trigger $\langle$ *beb*, *Broadcast* $\mid$ [DATA, s, m] $\rangle$;

Answer to the following questions:

1. assuming that up to *f* processes may commit omission failures and no other failures may happen, discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).
1. assuming that up to *f* processes may be Byzantine faulty but constrained to have a symmetric behaviour[1], discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).

Ex 5: Consider a distributed system composed by *n* processes each one having a unique identifier. Processes communicate by exchanging messages through perfect point-to-point links and are connected through a grid (i.e., each process p_i can exchange messages only with processes located at *nord*, *sud*, *east* and *west* when they exist).
An example of such network is provided in the following figure:

[1] A Byzantine process has a symmetric behaviour if it can change the content of every message it is going to send, but it cannot send different values to different processes when invoking the bebBroascast. Summarizing, it can cheat but it will do it in a consistent way.



Processes are not going to fail, and they initially know only the number of processes in the system N and the identifiers of their neighbors.
Processes in the system must agree on a color assignment satisfying the following specification:

Module

Name: k-Color assignment

Events:

Request: ⟨ ca, Propose | c ⟩: Proposes a color to be adopted.
Indication: ⟨ ca, Decide | c ⟩: Outputs a decided color to be adopted by the process

Properties:

Termination: Every process eventually decides a color.
Validity: If a process decides a color *c*, then *c* was proposed by some process or c = *default*.
Integrity: No process decides twice.
Weak Agreement: If two processes decide c_i and c_j then either $c_i = c_j$ or one of the two is *default*
Color Selection: Let *C* be the set of proposed colors, if $|C| > 0$ then there exists at least a color $c_i \in C$ that is decided by k processes.

Assuming that 1< k <N and that k is known by every process, write the pseudo-code of an algorithm implementing the k-Color assignment primitive.

According to the Italian law 675 of the 31/12/96, I authorize the instructor of the course to publish on the web site of the course results of the exams.

Signature: ___________________________

Ex 1: Provide the specification of the (1, N) Regular Register and describe the majority voting algorithm discussed during the lectures.

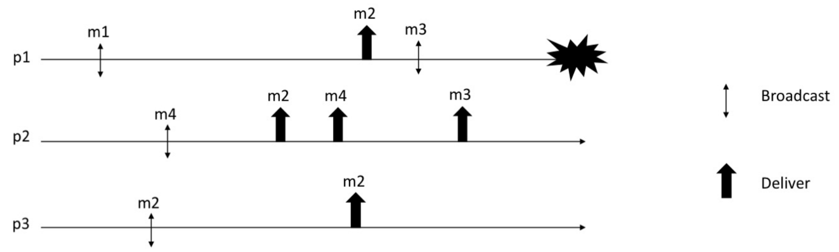
A REGULAR REGISTER (1,N) IS A STRUCTURE SHARED BY A WRITER PROCESS AND N READER PROCESSES. ITS MAIN PROPERTIES ARE:

TERMINATION: IF A CORRECT PROCESS INVOKE A READ OR WRITE OPERATION, THE OPERATION WILL COMPLETE SUCCESSFULLY.

VALIDITY: A READING CAN RETURN THE VALUE OF THE LAST COMPLETED WRITE OPERATION, OR A VALUE BEING WRITTEN IF THE READING IS CONCURRENT WITH THE WRITING.

THE MAJORITY VOTING ALG IS USED TO IMPLEMENT A REGULAR REGISTER (1,N) IN A FAIL-SILENT MODEL, WHERE PROCESSES CAN CRASH BUT CRASH CANNOT BE DETECTED.
THE WRITER SENDS <V, ts> TO ALL PROCESS VIA beb. EACH PROCESS RECEIVES THE MESSAGE AND UPDATES ITS LOCAL COPY OF THE REGISTRY IF THE ts IS GREATER THAN THE CURRENT ONE. THE WRITER WAITS FOR ACKS FROM A MAJORITY OF PROCESSES TO COMPLETE THE OPERATION.
THE READER SENDS READING REQUESTS TO ALL PROCESSES, WHICH RESPOND WITH <V, ts> OF THEIR LOCAL COPY. THE READER COLLECTS THE ANSWERS AND SELECTS THE VALUE WITH THE HIGHEST ts.
THE MAJORITY VOTING ALG GUARANTEES TERMINATION AND VALIDITY BY EXPLOITING UNIQUE QUORUM AND TIMESTAMP TO COORDINATE READINGS AND SCRIPTURES.

Ex 2: Consider the message pattern shown in the Figure

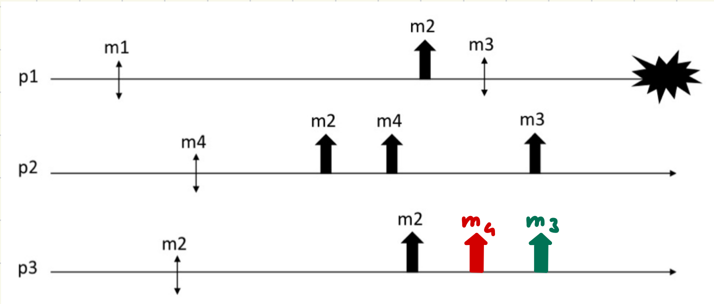


Answer to the following questions:

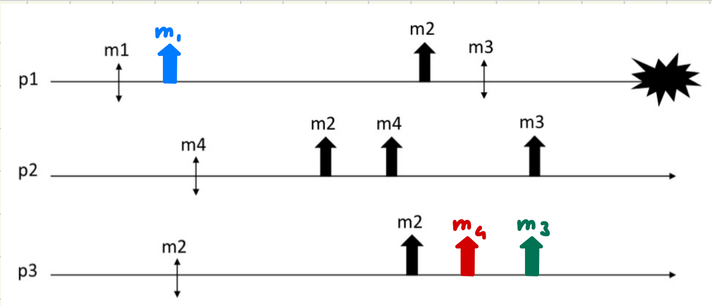
1. Complete the partial execution in order to obtain a run satisfying Uniform Reliable Broadcast
2. Complete the partial execution in order to obtain a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast
3. Complete the partial execution in order to obtain a run satisfying Best Effort Broadcast but not Regular Reliable Broadcast
4. List **ALL** the possible sequences satisfying both causal order and total order

NOTE: To solve the exercise you can just add deliveries of messages and new broadcast (if needed)

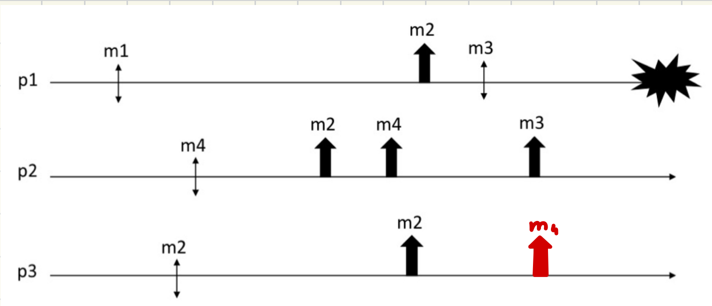
1) URB



2) RRB BUT NO URB



3) BEB BUT NO RRB



4) CASUAL ORDER = LOCAL ORDER + FIFO ORDER

$m_1 \rightarrow m_3$ FIFO

$m_2 \rightarrow m_3$ LOCAL

$m_2 \rightarrow m_4$ TOTAL

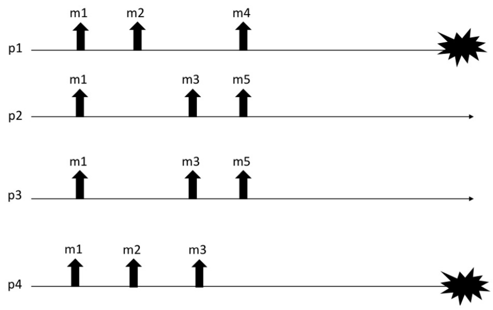
$m_4 \rightarrow m_3$ TOTAL

m_1, m_2, m_4, m_3

m_2, m_1, m_4, m_3

m_2, m_4, m_1, m_3

Ex 3: Consider the execution depicted in the Figure



Answer to the following questions:

1. Which is the strongest Total Order specification satisfied by the proposed run? Provide your answer by specifying both the agreement and the ordering property.
2. Modify the run in order to obtain an execution satisfying TO (UA, WUTO) but not TO (UA, SUTO)
3. Modify the run in order to obtain an execution satisfying TO (NUA, WNUTO) but not TO(NUA, WUTO).

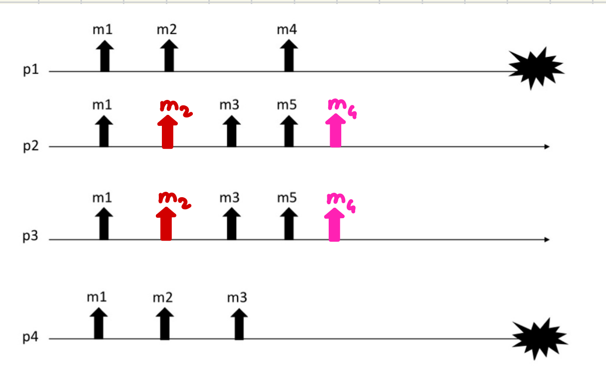
NOTE: To solve the exercise you can just add deliveries of messages.

1) TO (NUA, WUTO)

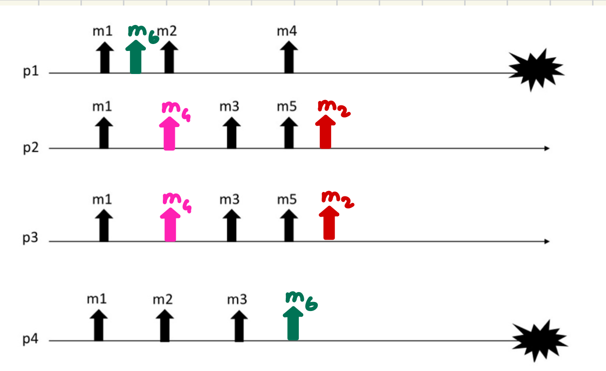
NUA: BECAUSE THERE ARE MSGS DELIVERED BY FAULTY PROCESSES THAT CORRECT PROCESSES DON'T DELIVER

WUTO: IF PROCESSES p AND q BOTH TODELIVER m AND m', THEN p TODELIVERS m BEFORE m' IF AND ONLY IF q TODELIVERS m BEFORE m'

2) TO (UA, WUTO) BUT NOT TO (UA, SUTO)



3) TO (NUA, WNUTO) BUT NOT TO (NUA, WUTO)



Ex 4: Let us consider a Regular Reliable Broadcast primitive satisfying the following properties:

- *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.
- *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

Let us consider a distributed system composed of N processes executing the Eager algorithm (reported in figure)

Algorithm 3.3: Eager Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*.

upon event ⟨ *rb*, *Init* ⟩ **do**
delivered := $\emptyset$;

upon event ⟨ *rb*, *Broadcast* | *m* ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, *m*] ⟩;

upon event ⟨ *beb*, *Deliver* | *p*, [DATA, *s*, *m*] ⟩ **do**
if $m \notin$ *delivered* **then**
delivered := *delivered* $\cup \{m\}$;
trigger ⟨ *rb*, *Deliver* | *s*, *m* ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, *s*, *m*] ⟩;

Answer to the following questions:

1. assuming that up to f processes may commit omission failures and no other failures may happen, discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).
1. assuming that up to f processes may be Byzantine faulty but constrained to have a symmetric behaviour[1], discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).

THIS ALG IS THE ASYNCHRONOUS RB PROTOCOL THAT CANNOT PROVIDE THE SET OF CORRECT PROCESSES AND CANNOT KNOW WHEN THE RETRANSMISSION IS NEEDED OR NOT. SO THE PROTOCOL WILL RETRANSMIT EVERY MSG THAT RECEIVE. THIS MEANS THAT THE FD IS NOT PERFECT.

1) VALIDITY: SATISFIED IF THERE IS AT LEAST ONE CORRECT p THAT RECEIVES m, AS IT WILL RETRANSMIT IT, ENSURING THAT IT'S RECEIVED BY THE MAJORITY.

NO DUPLICATION: SATISFIED BECAUSE THE PROTOCOL CHECK IF m IS DELIVERED ($m \notin$ DELIVERED) IN ORDER TO AVOID DUPLICATION.

NO CREATION: SATISFIED BECAUSE IN THE beb THE PROTOCOL SPECIFY THE SOURCE s. WE ARE SURE THAT m IS BROADCASTED BY THE SENDER s.

AGREEMENT: SATISFIED, SINCE THE PROTOCOL WILL RETRANSMIT EVERY MSG THAT RECEIVE, ALL PROCESS WILL RECEIVE AND DELIVER m.

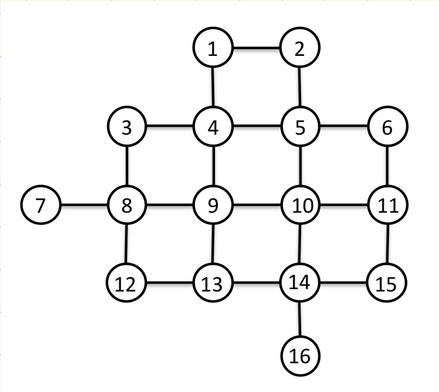
2) VALIDITY: SATISFIED IF AT LEAST ONE CORRECT p RECEIVES AND FORWARDS m, AS THE BYZANTINE PROCESSES COULD IGNORE m OR SEND CORRUPT MSGS, BUT AT LEAST A CORRECT PROCESS WILL RETRANSMIT IT.

NO DUPLICATION: SATISFIED BECAUSE THE PROTOCOL CHECK IF m IS DELIVERED ($m \notin$ DELIVERED) IN ORDER TO AVOID DUPLICATION, EVEN IF BYZANTINE SEND SEVERAL TIMES m.

NO CREATION: SATISFIED BECAUSE IN THE beb THE PROTOCOL SPECIFY THE SOURCE s. BYZANTINE COULD INVENT NEW MSGS, BUT CORRECT PROCESSES CAN RECOGNIZE AND IGNORE UNREALITIES.

AGREEMENT: IF TOO MANY PROCESSES ARE BYZANTINE THEY COULD BLOCK THE MSG, PREVENTING AGREEMENT.

Ex 5: Consider a distributed system composed by *n* processes each one having a unique identifier. Processes communicate by exchanging messages through perfect point-to-point links and are connected through a grid (i.e., each process p_i can exchange messages only with processes located at *nord*, *sud*, *east* and *west* when they exist).
An example of such network is provided in the following figure:



Processes are not going to fail, and they initially know only the number of processes in the system N and the identifiers of their neighbors.
Processes in the system must agree on a color assignment satisfying the following specification:

Module

Name: k-Color assignment

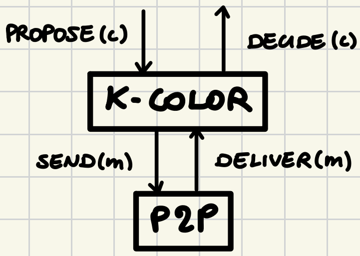
Events:

Request: ⟨ ca, Propose | c ⟩: Proposes a color to be adopted.
Indication: ⟨ ca, Decide | c ⟩: Outputs a decided color to be adopted by the process

Properties:

Termination: Every process eventually decides a color.
Validity: If a process decides a color *c*, then *c* was proposed by some process or c = *default*.
Integrity: No process decides twice.
Weak Agreement: If two processes decide c_i and c_j then either $c_i = c_j$ or one of the two is *default*
Color Selection: Let *C* be the set of proposed colors, if $|C| > 0$ then there exists at least a color $c_i \in C$ that is decided by k processes.

Assuming that $1 < k < N$ and that k is known by every process, write the pseudo-code of an algorithm implementing the k-Color assignment primitive.




```
UPON EVENT < K-COLOR, INIT > DO
    N = Π
    c = DEFAULT
    PROPOSAL_SET = ∅
    DECISION = ⊥
    NEIGHBOURS = {N, S, E, W}
    K = *   IT'S KNOWN

UPON EVENT < K-COLOR, PROPOSE | c > DO
    PROPOSAL = c
    PROPOSAL_SET = PROPOSAL_SET ∪ {c}
    FORALL pj IN NEIGHBOURS DO
        TRIGGER < P2P, SEND | (PROPOSAL, PROPOSAL_SET, pi) > TO pj

UPON EVENT < P2P, DELIVER | (PROPOSAL, PROPOSAL_SET, pi) > FROM pk
    IF N ⊆ PROPOSAL_SET AND DECISION = ⊥ DO
        COLOR = LIGHTER_COLOR(PROPOSAL_SET)
        IF SELF ≤ K DO
            DECISION = COLOR
            TRIGGER < K-COLOR, DECIDE | DECISION >
        ELSE
            DECISION = ⊥
            TRIGGER < K-COLOR, DECIDE | DECISION >
    ELSE
        PROPOSAL = c
        PROPOSAL_SET = PROPOSAL_SET ∪ {c}
        FORALL pj IN NEIGHBOURS DO
            TRIGGER < P2P, SEND | (PROPOSAL, PROPOSAL_SET, pi) > TO pj
```