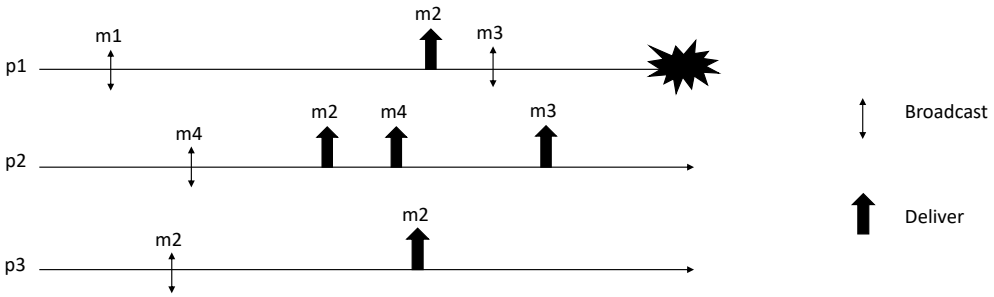
FATTO

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.

X **Ex 2:** Consider the message pattern shown in the Figure

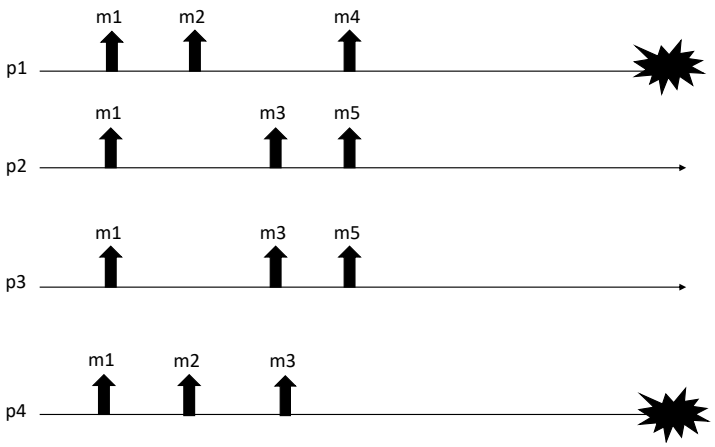


Answer to the following questions:

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)

X **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 ⟨ *rb*, *Init* ⟩ **do**
delivered := ∅;

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

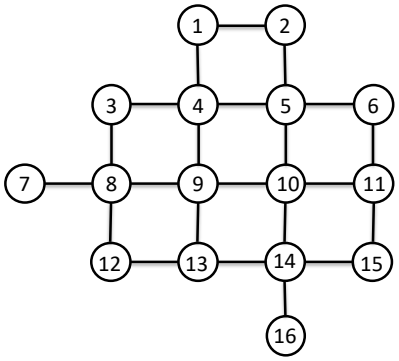
upon event ⟨ *beb*, *Deliver* | *p*, [DATA, *s*, *m*] ⟩ **do**
if *m* ∉ *delivered* **then**
delivered := *delivered* ∪ {*m*};
trigger ⟨ *rb*, *Deliver* | *s*, *m* ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, *s*, *m*] ⟩;

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).

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:

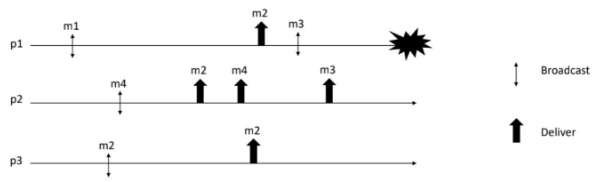
Termination: Every process eventually decides a color.
Validity: If a process decides a color *c*, then *c* was proposed by some process or c = *default*.
Integrity: No process decides twice.
Weak Agreement: If two processes decide c_i and c_j then either $c_i = c_j$ or one of the two is *default*
Color 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: _____

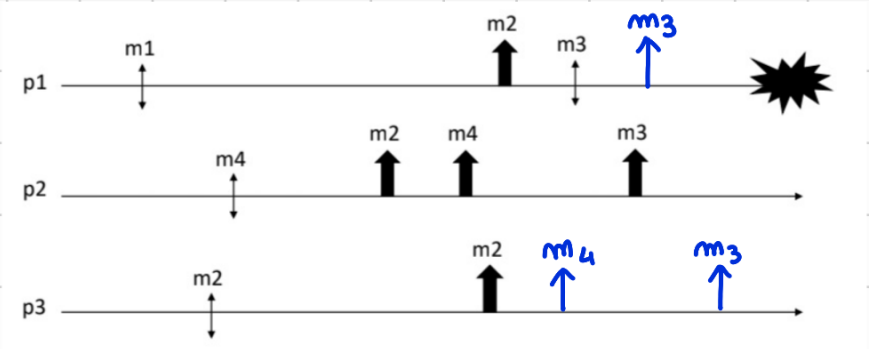
Ex 2: Consider the message pattern shown in the Figure



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

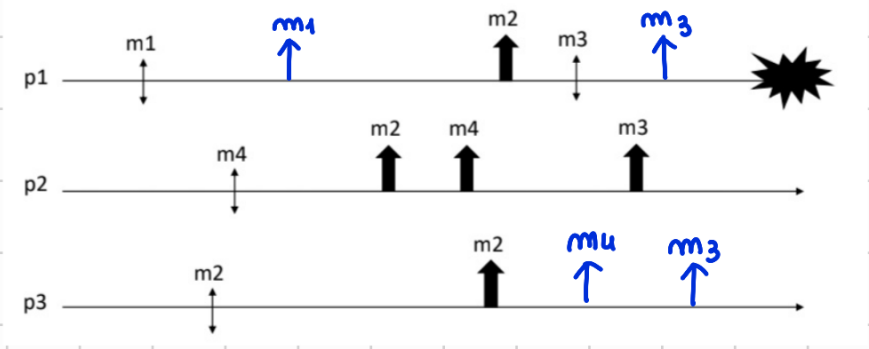
1) URB : faulty subset of correct



C: m_2 m_4 m_3

F: m_2 m_3

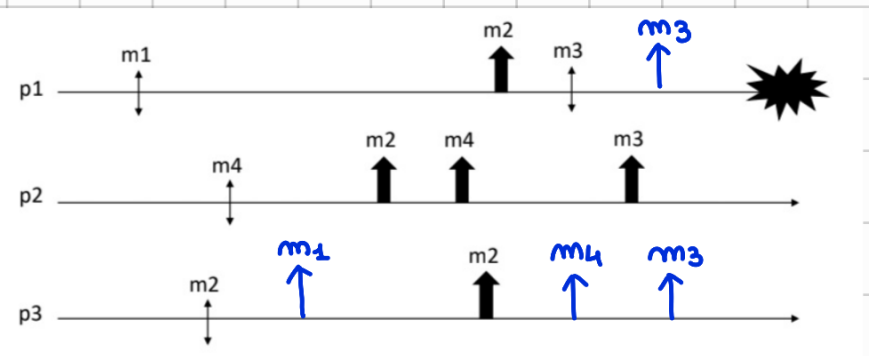
2) RB, not URB : correct same set



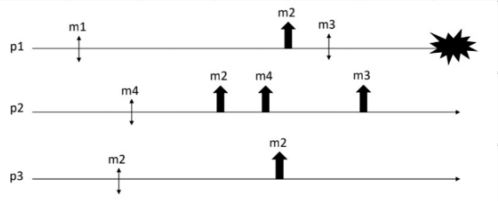
C: m_2 m_3 m_4

F: m_2 m_3 m_1

3) BEB, not RB : validity m_4 m_2



4) Casual and Total

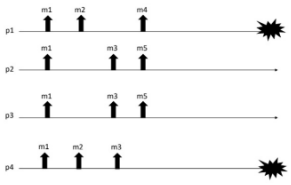


FIFO + LOCAL : $m_1 \rightarrow m_3$
$m_2 \rightarrow m_3$

TOTAL : $m_2 \rightarrow m_4 \rightarrow m_3$

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



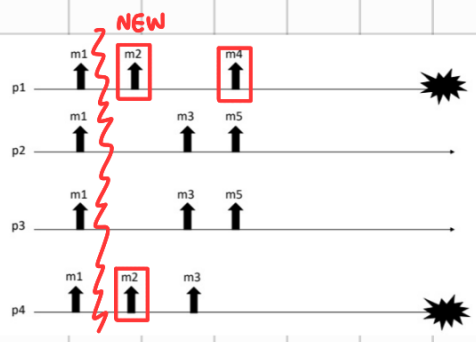
Answer to the following questions:

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.

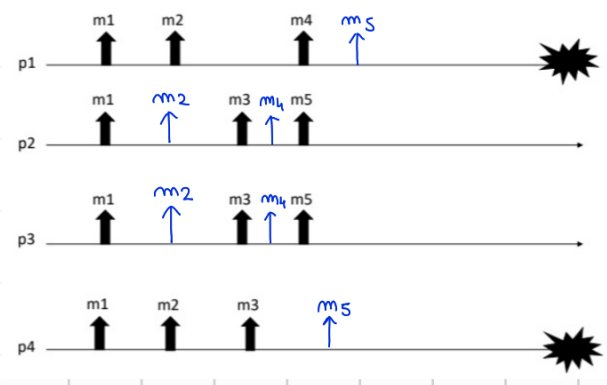
1) NUA: no agreement (F no subset of C).
WUTO → TO(NUA, WUTO)



p_1 m_1 (m_2 m_4)

p_4 m_1 (m_2) m_3

2) TO(UA, WUTO), not TO(UA, SUTO)



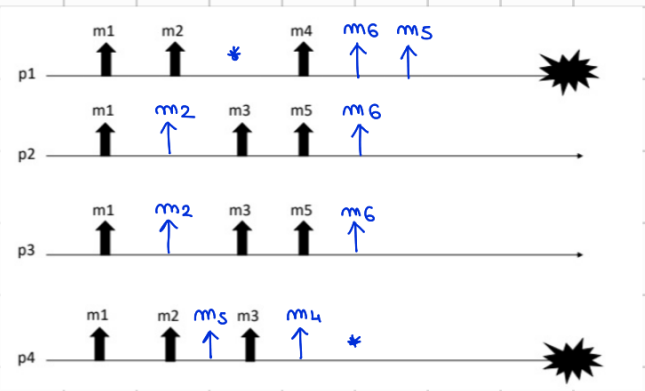
m_1 m_2 m_4 → SUBSET

m_1 m_2 m_3 m_4 m_5

m_1 m_2 m_3 m_4 m_5

m_1 m_2 m_3 → SUBSET

3) TO(NUA, WNUTO), mais TO(NUA, WUTO)



*: omission

1 2 * 4 6 5

1 2 3 5 6

1 2 3 5 6

1 2 5 3 4

WUTO: 1 2 3 5
1 3 4 5 *

WNUTO: 1 2 3 4 6
1 4 3 5 6 *

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 := ∅;

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

upon event ⟨ *beb*, *Deliver* | *p*, [DATA, *s*, *m*] ⟩ **do**
if *m* ∉ *delivered* **then**
delivered := *delivered* ∪ {*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).

f: omit failures

The eager reliable broadcast is the asynchronous RB protocol that cannot provide the list / set of correct processes and cannot know when the retransmission is needed or not. The protocol (to stay safe) will retransmit every message that receive. This means that the FD is not perfect.

(1)

Validity: yes it is still valid. Because if a correct process broadcast a message [DATA, s, m], p received the m and if m ∉ delivered, p delivered [DATA, s, m] through perfect link.

No duplication: yes it is valid because the protocol check if the message is delivered (m ∉ delivered) in order to avoid duplication of messages.

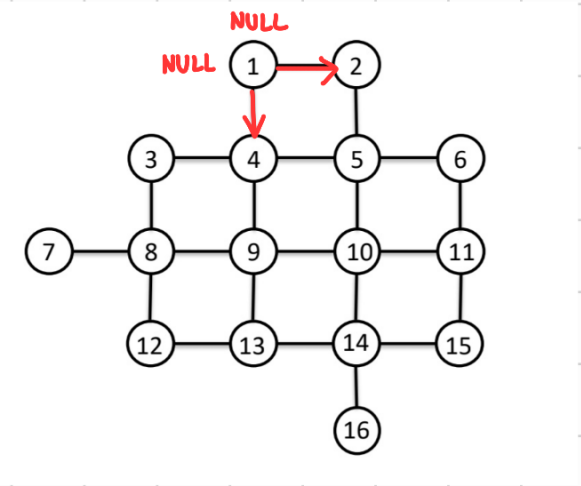
No creation: yes it is valid because in the broadcast of a message the protocol specify the source s, that

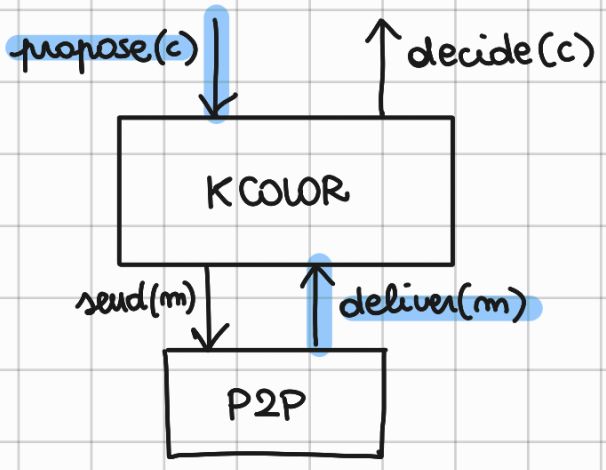
is the sender. We a message is delivered we are sure that the m is broadcasted by the sender s. The f processes can't generate fake messages.

Agreement: yes it is valid, because can happen that faulty processes omit a msg but the correct still broadcast and deliver msgs from correct processes.

(2) NOT DONE

5) n processes, unique ID
P2P links
grid (N, S, E, W)
N all correct
get_neighbors
no failures





We have a grid of processes that are connected through perfect link. A process is connected to its neighbours and can communicate only with p_N, p_S, p_E and p_W. Assuming that p are not going to fail, each process start with a value c and have to exchange this value c with the other p.
At first we start a round, the p_i starts to propose the value c to its neighbours (if they exist) and c is added to proposal set. When the set is full of value c and all the correct process have proposed, start the decide part. The final value c have to be proposed by K processes accordingly to $1 < K < N$.

upon event < Kcolor, INIT > do
    N = Π
    c = default
    proposal_set = ∅
    decision = ⊥
    neighbours = { N, S, E, W }
    K = * it is known

upon event < Kcolor, PROPOSE | c > do
    proposal = c
    proposal_set = proposal_set ∪ { c }
    for each $p_j \in$ neighbours do
        trigger < p2p, p2p_SEND | (PROPOSAL, proposal_set, p_i) > to p_j

upon event < p2p, p2p_DELIVER | (PROPOSAL, proposal_set, p_t) > from p_K

    if N ⊆ proposal_set
        if decision = ⊥ do
            decision = FUNCTION FOR DECISION

    else
        proposal_set = ps ∪ ps update of proposal set
        for each $p_j \in$ neighbours do
            trigger < p2p, p2p_SEND | (PROPOSAL, proposal_set, p_t) > to p_x

upon event FUNCTION FOR DECISION

```
color = lighter_color (proposal_set)
if self ≤ k do
    decision = color
    trigger <kcolor, DECIDE | decision>
else
    decision = ⊥
    trigger <kcolor, DECIDE | decision>
```

OTHERWISE I can insert the function in the p2p deliver:

upon event <p2p, p2p_DELIVER | (PROPOSAL, proposal_set, p_t) > from p_k

```
if N ≤ proposal_set
    if decision = ⊥ do
        color = lighter_color (proposal_set)
        if self ≤ k do
            decision = color
            trigger <kcolor, DECIDE | decision>
        else
            decision = ⊥
            trigger <kcolor, DECIDE | decision>
else
    proposal_set = ps ∪ ps   update of proposal set
    for each p_i ∈ neighbours do
        trigger <p2p, p2p_SEND | (PROPOSAL, proposal_set, p_t) > to p_x
```