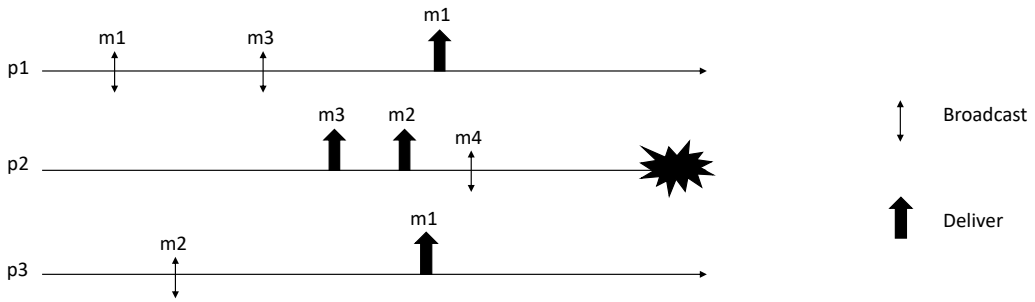
FATTO

Distributed Systems
29/01/2021
Exam A

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

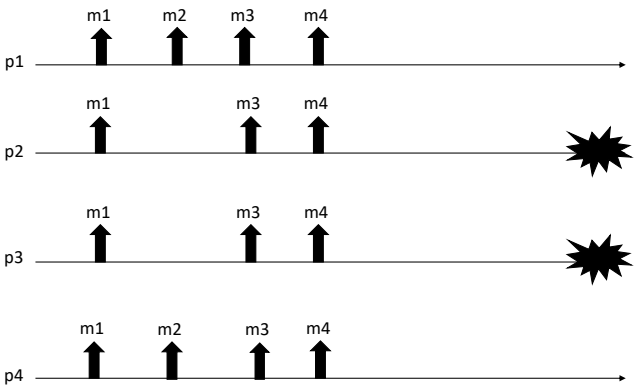


Answer to the following questions:

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, WNUTO) but not TO(UA, WUTO)

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

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

Ex 4: Let us consider a distributed system composed of N processes executing the algorithm reported in figure

Algorithm 3.12: Broadcast with Sequence Number

Implements:
FIFOReliableBroadcast, **instance** *frb*.

Uses:
ReliableBroadcast, **instance** *rb*.

upon event ⟨ *frb*, *Init* ⟩ **do**
lsn := 0;
pending := ∅;
next := $[1]^N$;

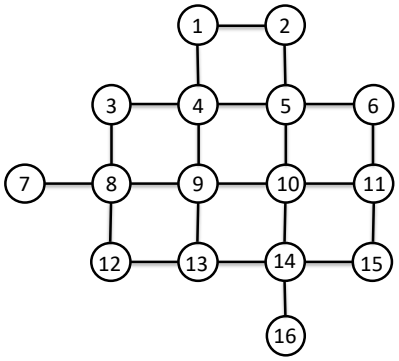
upon event ⟨ *frb*, *Broadcast* | m ⟩ **do**
lsn := *lsn* + 1;
trigger ⟨ *rb*, *Broadcast* | [DATA, *self*, m, *lsn*] ⟩;

upon event ⟨ *rb*, *Deliver* | p, [DATA, s, m, sn] ⟩ **do**
pending := *pending* ∪ {(s, m, sn)};
while exists $(s, m', sn') \in$ *pending* such that $sn' = next[s]$ **do**
$next[s] := next[s] + 1$;
pending := *pending* \ {(s, m', sn')};
trigger ⟨ *frb*, *Deliver* | s, m' ⟩;

Let us assume that
1. up to f processes may be Byzantine faulty and
2. A Byzantine process is not able to compromise the underline Reliable Broadcast primitive (i.e., when a Byzantine process sends a message through the rbBroadcast interface, the message will be reliably delivered to every correct process).

For each of the following properties, discuss if it can be guaranteed when f=1 and motivate your answer (also by using examples)
- *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.
- *FIFO delivery:* If some process broadcasts message *m1* before it broadcasts message *m2*, then no correct process delivers *m2* unless it has already delivered *m1*.

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: 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:

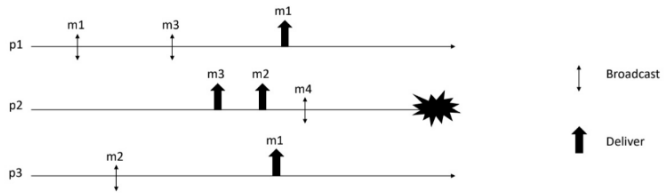
Termination: Every process eventually decides a color.
Validity: If a process decides a color *c*, then *c* was proposed by some process.
Integrity: No process decides twice.
Fairness: If a color *c* is proposed by a process p_i, *c* will be eventually decided by some process p_j
Color Balance: Let *C* be the set of proposed colors, every $c_i \in C$ is decided by at most k= $N/|C|$ processes.

Write the pseudo-code of an algorithm implementing the 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

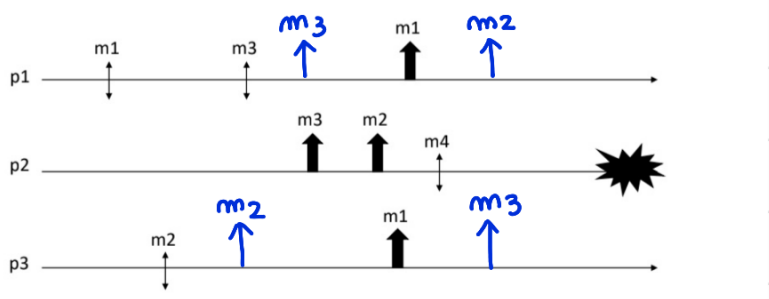


Answer to the following questions:

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

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

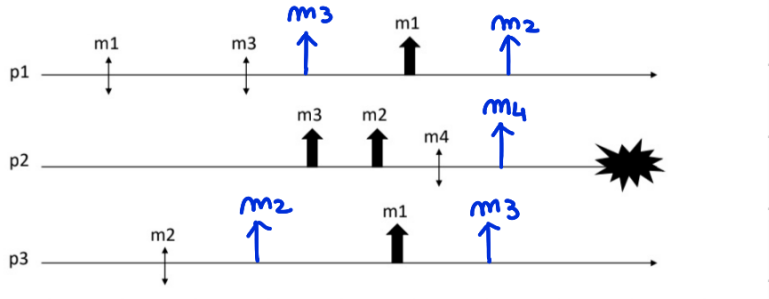
1) URB



correct: m_1 m_2 m_3

faulty subset : m_3 m_2

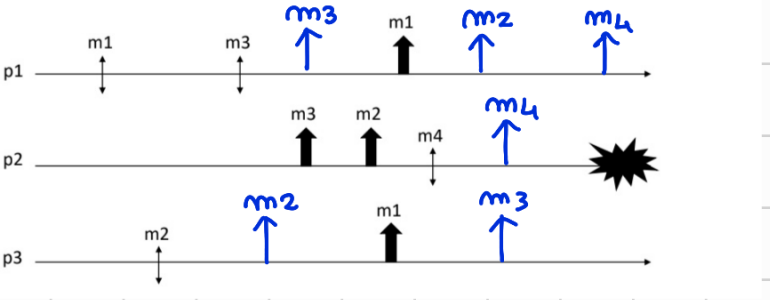
2) RB, not URB



correct: m_3 m_1 m_2
m_2 m_1 m_3

faulty: m_3 m_2 m_4

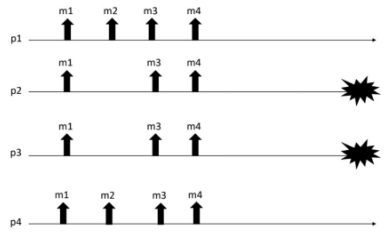
3) BEB, not RB



correct: m_3 m_1 m_2 m_4
m_2 m_1 m_3

faulty: m_3 m_2 m_4

Ex 3: Consider the execution depicted in the Figure



Answer to the following questions:

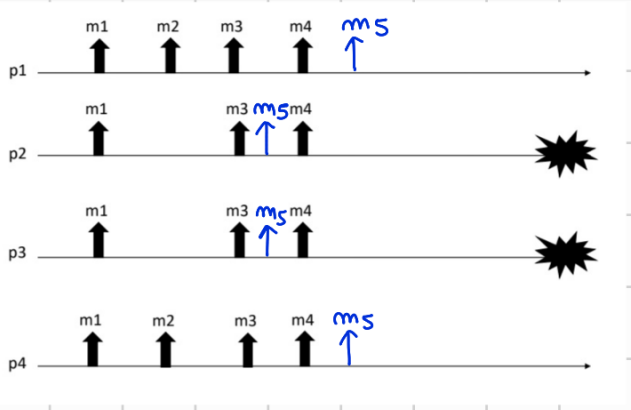
1. Which is the strongest Total Order specification satisfied by the proposed run? Provide your answer by specifying both the agreement and the ordering property.
2. Modify the run in order to obtain an execution satisfying TO(UA, WNUTO) but not TO(UA, WUTO)
3. Modify the run in order to obtain an execution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

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

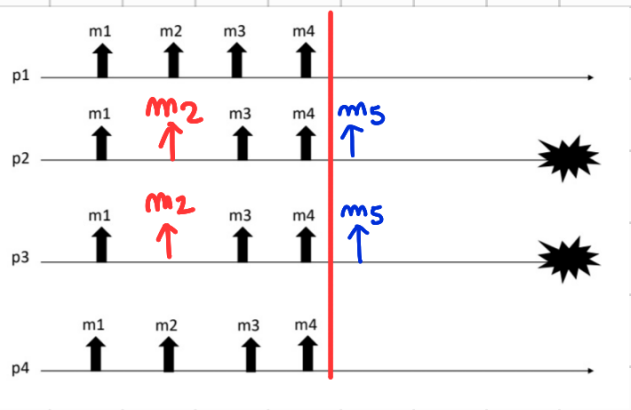
1) UA: the faulty are a subset of correct
WUTO: no restrictions on the set of delivered message

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

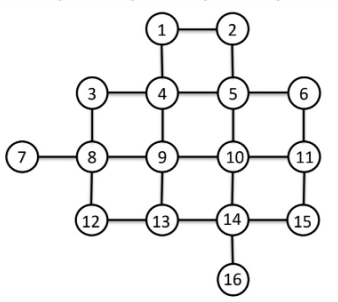
WNUTO
1 2 3 4
1 4 3



3) TO(NUA, SUTO), not TO(UA, SUTO)



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:

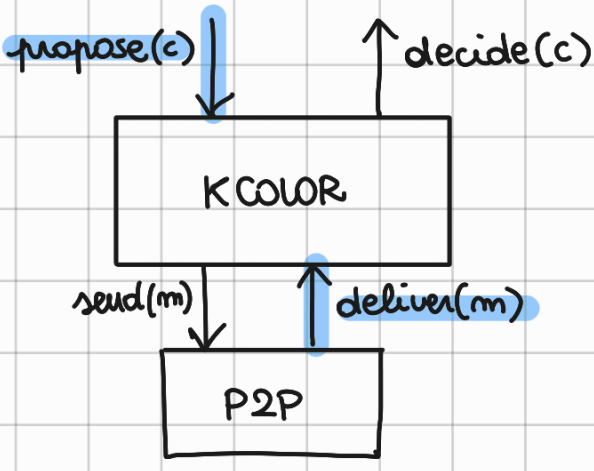
Module
Name: Color assignment

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.
Integrity: No process decides twice.
Fairness: If a color *c* is proposed by a process p_i, *c* will be eventually decided by some process p_j
Color Balance: Let *C* be the set of proposed colors, every $c_i \in C$ is decided by at most k= $N/|C|$ processes.

Write the pseudo-code of an algorithm implementing the Color assignment primitive.

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 $\frac{N}{|C|}$.

upon event < Kcolor, INIT > do
    N = Π
    proposal_set = ∅
    decision = ⊥
    neighbours = { N, S, E, W }

$$K = \frac{N}{\text{proposal\_set}}$$

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>
```