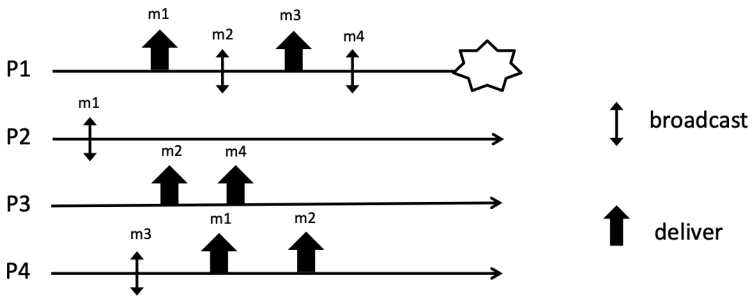
Ex 1: Consider the execution depicted in the Figure



Answer to the following questions:

1. Provide all the delivery sequences that satisfy both causal order and total order
2. Complete the execution in order to have a run satisfying TO(UA, WNUTO), FIFO order but not causal order

Ex_1_***Solution***:

1.1 Casual order relationships

Casual order is define on three fundamental properties. So, based on this idea, I will have:

- m2 -> m4 (FIFO order on p1)
- m1 -> m2 (local order on p1)
- m3 -> m4 (local order on p1)

Combining this three:

It follows that m1 -> m2 -> m4 and m3 -> m4

Concerning total order we have that:

- m2 -> m4 (due to deliveries on p3)
- m1-> m2 (due to deliveries on p3)
- m1 -> m3 (due to deliveries on p1 faulty)

It follows m1 -> m2 -> m4 and m1->m3

If we want to consider a uniform version of the total order then:

- m1, m3, m2, m4
- m1, m2, m3, m4

If we want to consider a non uniform version of the total order, we don't care about p1 and thus:

- m1, m3, m2, m4
- m1, m2, m3, m4
- m3, m1, m2, m4 (this is not delivered by p1)

1.2

We want basically TO (UA, WNUTO) + FIFO Order + non-casual order

- m2 -> m4 (FiFO order on p\1 and due to deliveries on p3)
- m1 -> m2 (due to deliveries on p2)

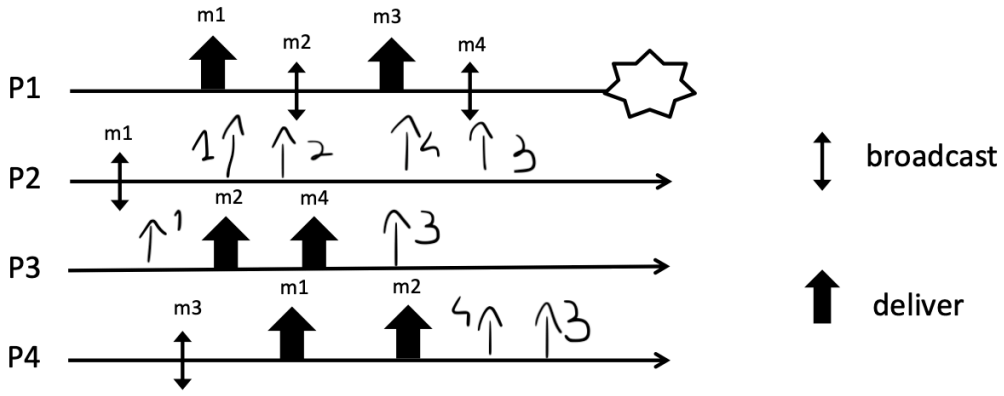
Thus it follows m1 -> m2 -> m4

To break casual order and keep FIFO I need to have:

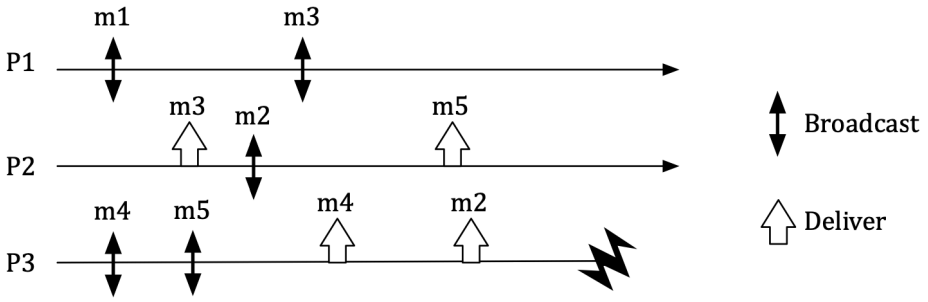
- m2-> m1 (to break first local order on p1)
- m4-> m3 (to break second local order on p1)

The second condition is the only one applicable and thus

m1, m2, m4, m3 can be delivered by all correct



Ex 2: Consider the partial execution shown in the Figure and answer to the following questions:

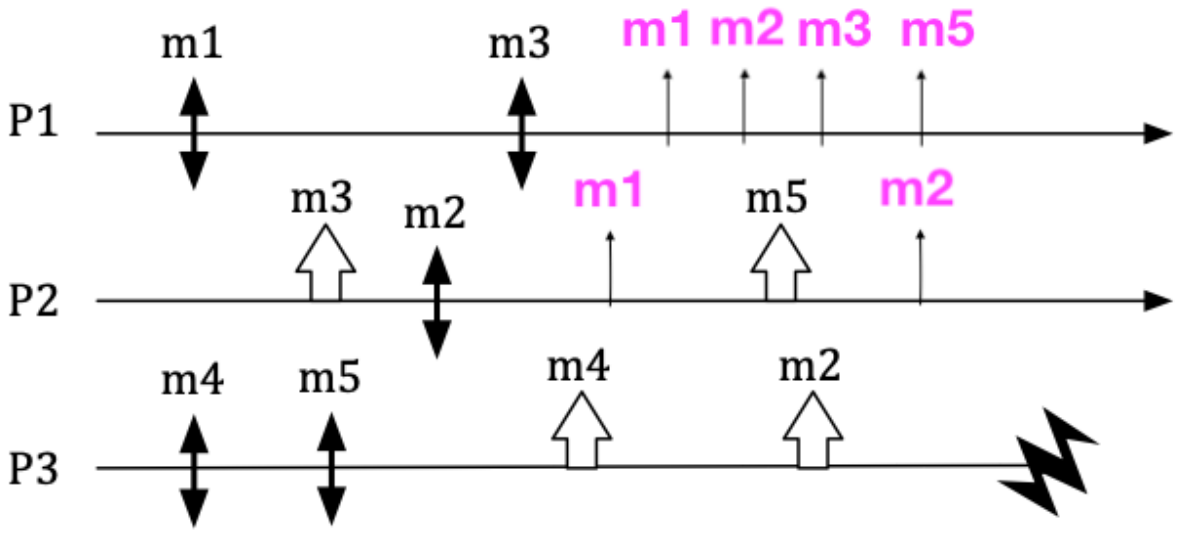


1. Complete the execution in order to have a run satisfying the Regular Reliable Broadcast specification but not Uniform Reliable Broadcast one.
2. For each process, provide ALL the delivery sequences satisfying FIFO Reliable Broadcast but not satisfying causal order.
3. For each process, provide ALL the delivery sequences satisfying total order and causal order.

Esercizio_2_Soluzione:

2.1

:



2.2

In order to get FIFO I need to ensure that

- m1 -> m3
- m4 -> m5

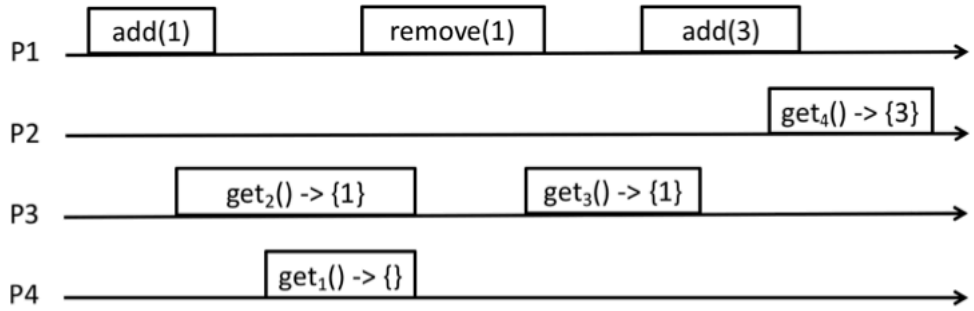
In order to have non casual I need m2->m3

So all the possible sequences are extracted from all the possible permutations of messages

2.3
To guarantee casual order, we can see m2->m3

Exercise_3_Solution:

3.1



S = { $add_1(1), get_2() -> \{1\}, \ remove_1(1), get_1() -> \{\}, add_1(3), get_4() -> \{3\}$ }

Given that I'm not able to place $get_2() -> \{1\}$ in the sequence satisfying precedence relationships and obtaining a legal history.

Exercise_4_Solution:

We have to check the algorithm and try to understand what this algorithm is doing

```
upon event Xbroadcast(m)
        mysn = mysn+1;
        ∀ p ∈ correct
                pp2pSend ("MSG", m, mysn, myId);

upon event pp2pReceive ("MSG", m, sn, i)
       mysn= mysn+1;
       if (m ∉ delivered)
          trigger XDeliver (m);
          delivered = delivered ∪ {m};

upon event crash (pi)
        correct = correct / {pi}
```

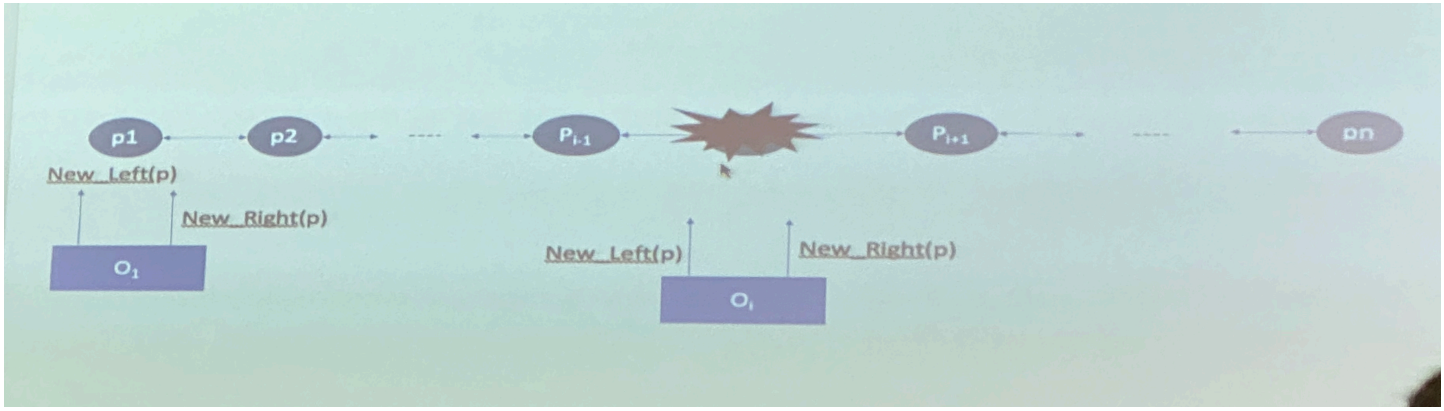
4.1

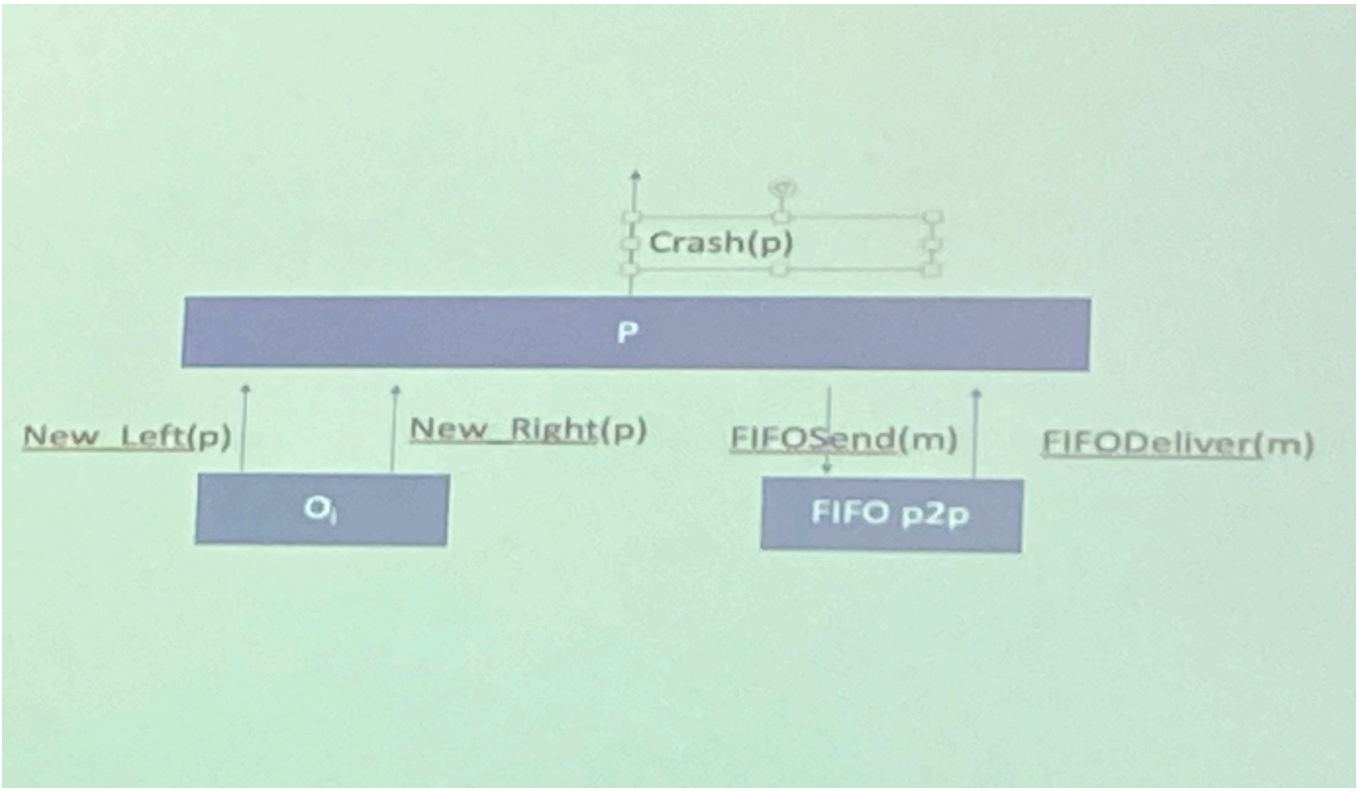
VALIDITY is satisfied so it is also Best Effort.
RELIABLE. It's easy to see that is no reliable. Given a failure, you only remove and anything else. We can see that creating the execution of the algorithm.

4.2

TO is not satisfied (we have a perfect link). Then let's check if we have FIFO. In fifo we are a local sequences number that we increment. If we look at the delivery side, the receiver isn't checking any type of delivery order condition.

Exercise_5_Solution:





Init

$correct_i$ = {p1, p2. … pn}

$detected_i$ = empty

$left_i$ = get_left(correct)

$right_i$ = get_right(correct)

upon event new_right(pj)

$correct_i = correct_i \setminus right_i$

$detected_i = detected_i \cup \{right_i\}$

trigger event crash($right_i$)
$right_i = p_j$

upon event new_left(pj)
$correct_i = correct_i \setminus left_i$
$detected_i = detected_i$ U {$left_i$}
trigger event crash($left_i$)
$left_i = p_j$

when $detected_i$ changes do:
foreach p_j in $detected_i$
if j > i :
trigger FIFOSend(CRASH, pj) to left
else:
trigger FIFOSend(CRASH, pj) to right

upon event FIFODeliver(CRASH, pj) from pk
if pj not in $detected_i$
$correct_i = correct_i \setminus$ pj
$detected_i = detected_i$ U {pj}
if k > i :
trigger FIFOSend(CRASH, pj) to left
else:
trigger FIFOSend(CRASH, pj) to right