Exercise Session 4
Broadcast – Reliable, Uniform, Causal, and Total-Order

Exercise 1

Sketch an execution history with two processes p1 and p2, that illustrates the following case: the execution should satisfy the properties of **Reliable Causal Broadcast** but it should not satisfy **Uniform Causal Broadcast**.

Such an execution may happen when the uniformity of agreement is broken. This means that some process should deliver a message and then crash, before relaying it to the rest of the processes.

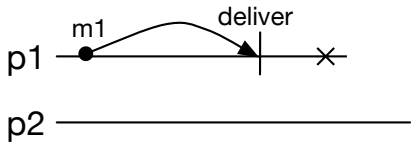


Figure 1: An execution that satisfies **Reliable Causal Broadcast** but does not satisfy **Uniform Causal Broadcast**.

Note: This execution represents a more general case (not specific to Causal-ordering), illustrating the difference between **Uniform Broadcast** and **Non-Uniform Broadcast**.

Exercise 2

If an algorithm implements Total Order broadcast, does it also satisfy the properties of the following?

1. Causal broadcast
2. Uniform Reliable broadcast

For each of the two (separately), either explain why it does, or give an execution that is allowed by total order broadcast, but is not allowed by the corresponding broadcast abstraction.

See Figure 2 and Figure 3.

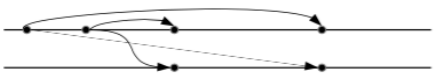


Figure 2: An execution that satisfies **Total Order Broadcast** but does not satisfy **Causal Broadcast**.

Figure 3: An execution that satisfies **Total Order Broadcast** but does not satisfy **Uniform Reliable Broadcast**.

Exercise 3

Consider a broadcast algorithm that has the following properties:

Validity: For any two processes pi and pj, if pi and pj are correct, then every message broadcast by pi is eventually delivered by pj.

No duplication: No message is delivered more than once.

No creation: If a message m is delivered by some process pj, then m was previously broadcast by some process pi.

Causal delivery: No process pi delivers a message m2 unless pi has already delivered every message m1 such that m1 $\longrightarrow$ m2.

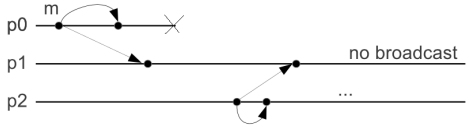
Does this broadcast algorithm satisfy the agreement property (if a message m is delivered by some correct process, them m is eventually delivered by every correct process)? Motivate your answer.

This is a best-effort causal broadcast abstraction. Accordingly, on a crash-free execution (all processes are correct) agreement is guaranteed due to validity.

Of course, the crash-free case is not that interesting, so lets discuss what happens in case there are crashes. What happens when a message is broadcast? A broadcast of message m by process p enforces all other processes to receive all the messages that belong to the causal past of m. [1] This, of course, includes the messages that were delivered and the messages that were broadcast by p before message m. It should be clear that this is a direct consequence of the definition of causality: A message m1 causally precedes a message m2 (m1 $\longrightarrow$ m2) when:

1. both are broadcasts of the same process and m1 was broadcast before m2
2. m1 is a broadcast of p1 and m2 is a broadcast of p2 and m2 was sent after p2 delivered m1
3. m1 $\longrightarrow$ m' and m' $\longrightarrow$ m2 entails m1 $\longrightarrow$ m2 (transitivity)

So, in an execution where the correct processes keep broadcasting messages, the causal delivery property ensures that all the delivered messages of a process before m will be delivered before delivering m, ensuring agreement even in the case of crashes. However, we cannot guarantee that every process will send an infinite number of messages, so the following execution is possible:



As you can see, p1 delivers message m sent by p0 just before p0 crashed. Due to the crash, m was not delivered by p2. If p1 stays inactive (as it happens in the execution above), p2 is not guaranteed to deliver message m, hence violating agreement. Consequently, the broadcast algorithm of the question does not guarantee the agreement property in executions that there are crashes.

[1] If p does not crash while broadcasting m, case that could lead to m not being delivered by every process