Exercise 1

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.

Answer. 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 1 and Figure 2.

![Figure 1](image1.png)

Figure 1: An execution that satisfies Total Order Broadcast but does not satisfy Causal Broadcast.

![Figure 2](image2.png)

Figure 2: An execution that satisfies Total Order Broadcast but does not satisfy Uniform Reliable Broadcast.

Exercise 2

Consider a broadcast algorithm that has the following properties:

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

No duplication: No message is delivered more than once.

No creation: If a message $m$ is delivered by some process $p_j$, then $m$ was previously broadcast by some process $p_i$.

Causal delivery: No process $p_i$ delivers a message $m_2$ unless $p_i$ has already delivered every message $m_1$ such that $m_1 \rightarrow m_2$.

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

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 let's 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 $m$.

This, of course, includes the messages that were delivered and

\[1\] If $p$ does not crash while broadcasting $m$, case that could lead to $m$ not being delivered by every process.
Distributed Algorithms 2016/2017

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 $m_1$ causally precedes a message $m_2$ ($m_1 \Rightarrow m_2$) when:

1. both are broadcasts of the same process and $m_1$ was broadcast before $m_2$
2. $m_1$ is a broadcast of $p_1$ and $m_2$ is a broadcast of $p_2$ and $m_2$ was sent after $p_2$ delivered $m_1$
3. $m_1 \Rightarrow m'$ and $m' \Rightarrow m_2$ entails $m_1 \Rightarrow m_2$ (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, $p_1$ delivers message $m$ sent by $p_0$ just before $p_0$ crashed. Due to the crash, $m$ was not delivered by $p_2$. If $p_1$ stays inactive (as it happens in the execution above), $p_2$ 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.