Concurrent Algorithms November 24, 2020

Exercise 9

Problem 1. Consider the Disk Paxos algorithm in slides 14-15 of the lecture. The algorithm is reproduced below. If we omit line 11, is the algorithm still correct? Why or why not?

Algorithm 1 Obstruction-free consensus with Memory Failures

1: **procedure** PROPOSE(v)
2: **while** true **do**
3:   **for** every memory m in parallel **do**
4:    $Reg[m][i].T.write(ts)$
5:    $temp[m][1 \dots n] \leftarrow Reg[m][1 \dots n].read()$
6:   **until** completed for majority of memories
7:   $val \leftarrow temp[1..m][1..n].highestTspValue()$
8:   **if** $val = \perp$ **then** $val \leftarrow v$
9:   **for** every memory m in parallel **do**
10:    $Reg[m][i].V.write(val, ts)$
11:    $temp[m][1 \dots n] \leftarrow Reg[m][1..n].read()$
12:   **until** completed for majority of memories
13:   **if** $ts = temp[1 \dots m][1 \dots n].highestTsp()$ **then return** (val)
14:    $ts \leftarrow ts + n$

Problem 2. Consider the following variant of the Non-equivocating Broadcast algorithm seen today in class. Does this algorithm satisfy the Non-Equivocating Broadcast properties? Why or why not?

Algorithm 2 Non-equivocating Broadcast

procedure BROADCAST(m)
 $R[s].write(m)$
procedure RECEIVE
 $senderMsg = R[s].read()$
 for $i = 1 \dots n$ **do**
  $recvMsg = R[i].read()$
  if $recvMsg \neq \perp \wedge recvMsg \neq senderMsg$ **then**
   ▷ found conflicting values (Byzantine sender), don't deliver
   return
 $R[i].write(senderMsg)$
 $deliver(senderMsg)$