EFREI M1: Distributed Algorthms 2019
Solutions for Quiz 4

1 "One-Shot" Atomic Snapshots

In *one-shot* atomic snapshot, every process p_i performs $update_i(v_i)$ followed by $snapshot()$, let S_i denote the result of the snapshot. Prove that every run of one-shot atomic snapshot satisfies the following properties:

Self-Inclusion $\forall i$: $v_i \in S_i$

Containment $\forall i, j$: $(S_i \subseteq S_j) \vee (S_j \subseteq S_i)$

Here we assume that the initial value of each memory location i is $\perp$ and we say that $S_i \subseteq S_j$ if $\forall k : (S_i[k] \neq \perp) \Rightarrow (S_i[k] = S_j[k])$.

Solution. Self-Inclusion is immediate: since p_i first performs $update_i(v_i)$ and then $snapshot()$ to obtain S_i, S_i must necessarily contains v_i in position i.

Now suppose that p_i and p_j obtained snapshots S_i and S_j, respectively, in a given run. Let L be any linearization of the corresponding history. Suppose that the snapshot operation of p_i precedes the snapshot operation of p_j in L. Since L is legal, for every non-$\perp$ position k in S_i, $update_k(v_k)$ precedes $snapshot_i()$ and, thus, $snapshot_j()$ in L. Since there is exactly one update performed by p_k in this run, we have $S_j[k] = S_i[k] = v_k$. The case when S_j precedes S_i in L is symmetric. Thus, Containment is also satisfied.

The Immediacy property is violated in the run presented in slide 21 of lecture 5. Here $v_2 \in S_1$, but $S_2 \not\subseteq S_1$.

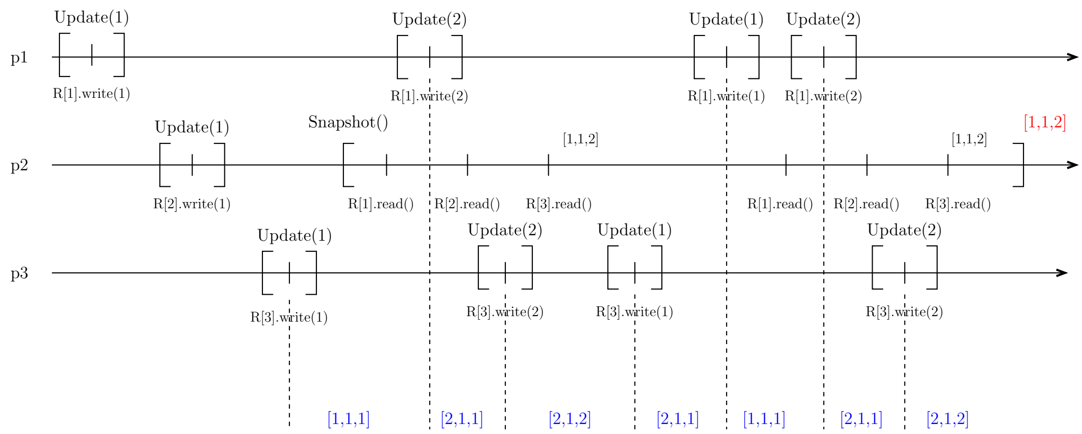

Figure 1: ABA in atomic snapshots: p_2 gets two identical scans, but the scan outcome (in red) does not belong to the set of allowed snapshots (in blue).

2 Atomic Snapshots and the ABA Problem

Show that our atomic snapshot algorithm fails if a process may perform multiple update operations with identical parameters.

Solution. Figure 1 gives an example of a run in which p_1 and p_2 update the memory concurrently with a snapshot taken by p_2. In the first scan, p_2 sees the old value od p_1 (1) and the new value of p_3 (2), then p_3 and p_1 write back their "old" values (in this order), and then we repeat this scenario with the second scan of p_2.

The resulting execution is not linearizable: there is no place between the updates where we can linearize the snapshot operation by p_2.

3 Extending ABD

- Does the ABD algorithm run by one writer and *multiple* readers implement an atomic (linearizable) register?

  The answer is no. Consider the following scenario. The writer invokes operation *write*(1) (assuming the the initial register value is 0). At the moment when the corresponding message only reaches a minority of the processes, reader 1 issues a *read* operation, reaches a process in this minority and returns 1. Then reader 2 issues a *read* operation, but this time only reaches a majority that is not aware on the new value. Reader 2 must return 0, which implies a new-old inverstion.

- If not, can it be turned in an atomic one?

  Here we need to add a *writing phase* to the reader's algorithm. Before returning a value v, the reader must make sure that a majority of processes has v or a newer value.

- How to support multiple writers?

  With multiple writers, we need to make sure that the timestamps associated with the values respect the order in which write operations are perfromed.

  For this, every writer should start with a *reading phase* in which the writer queries a majority of processes to get the maximal sequence number t used by them. The new sequence number is then chosen as $t + 1$.

  Of course, two concurrent writers can compute identical sequence number. To break ties, as in the Bakery algorithm, we define the timesamp to be the pair (sequence number, process identifier). Timestamps are then compared lexicographically.