SLR206: 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 $S_2 \not\subseteq S_1$.

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.

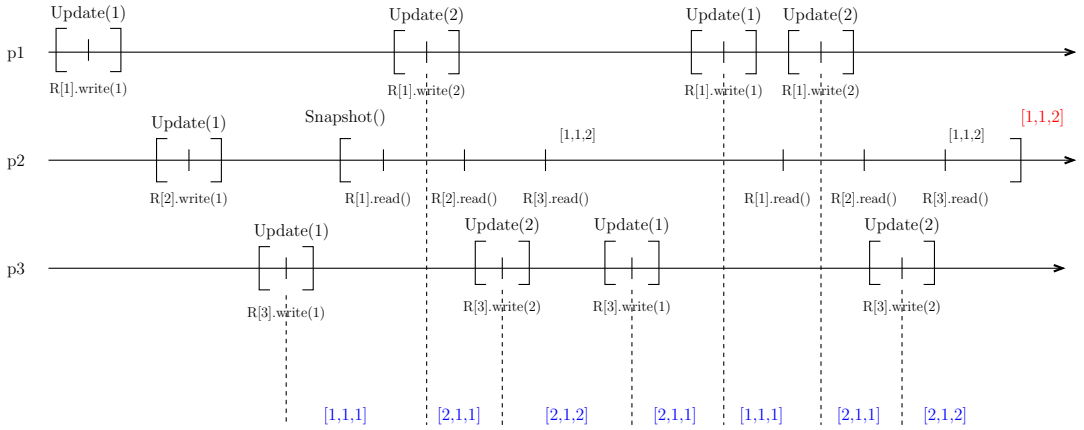

Figure 1: ABA in atomic snapshots: p_2 gets two identical scans, but the scan outcome (in red) does not belong to the set of allowed snapshots (in blue).

3 Immedite snapshot: using atomic registers instead of atomic snapshots

Would the one-shot IS algorithm (cf. the next page) be correct if we replace $A_r.update_i(v_i)$ with $U_r[i].write(v_i)$ and $A_r.snapshot()$ with $scan(U_r[1], \ldots, U_r[N])$? Here for each level r, instead of an atomic snapshot object A_r, we use N atomic registers $U_r[1],\ldots,U_r[N]$. Justify your answer.

Solution. The properties of atomic snapshot are not used by the algorithm. In particular, we do not need the containment property of AS to be satisfied by the snapshots returned by A_r, $r = 1, \ldots, N$, except for those of size r.

Using the same arguments as for the original algorithm, we can show that at most r processes can reach level r. Indeed, among N processes that can participate, at least one will output at level N: at least the one which was the last to perform the write to a register in $U_N[1], \ldots, U_N[N]$. By induction, the invatiant holds for every lower level. Hence, every process that returns at level r returns the values of the set of exactly r processes that reached that level.

Thus, we indeed can use arrays of atomic registers instead of atomic snapshot objects. Moreover, we can even use *regular* registers instead of atomic ones (please check).

4 Immedite snapshot: using just one array of atomic registers

Would it be possible to use only one array of N registers in the IS implementation?

Solution. By the algorithm if a process reached level r, then it earlier reached all levels in $\{r, \ldots, N\}$. Thus, to decide if a process can output at a given level, it can check if exactly r processes are in level r *or lower*.

We can therefore use just one array of registers $U[1], \ldots, U[N]$. To start a level r, any process p_i can simply write its value together with the level, $[v_i, r]$, to $U[i]$. And to decide if it can output at this level, p_i can count the number of processes that have written values $[*, r']$ such that $r' \leq r$. Please check if the resulting algorithm is correct.