Concurrent Computing November 25, 2025

Solutions to Exercise 9

Problem 1.

Consider the atomic *commit-adopt* object, which has the following specification. Every process p proposes an input value v to such an object and obtains an output, which consists of a pair (dec, val); dec can be either *commit* or *adopt*. The following properties are satisfied:

- **Validity:** If a process obtains output (commit, v) or (adopt, v), then v was proposed by some process.
- **Commitment:** If every process proposes the same value, then no process may output (adopt, v) for any value v.
- **Agreement:** If a process p outputs (commit, v) and a process q outputs (commit, v') or (adopt, v'), then $v = v'$.
- **Termination:** Every correct process eventually obtains an output.

Consider the following implementation of an atomic *commit-adopt* object from atomic wait-free snapshot objects and atomic MRMW registers:

- Using two shared snapshot objects: S_1 and S_2 of size n, initialized to $(\perp, \perp, \ldots, \perp)$;
- Using two local arrays of registers: a_i and b_i of size n.

The implementation is as follows:

```
propose(v)
    S_1.update(i, v);

    a_i := S_1.snapshot();
    if every non-⊥ value in a_i is v then
        x := (true, v);
    else
        v := max(a_i); // max(arr) returns the greatest non-⊥ element in array arr
        x := (false, v);

    S_2.update(i, x);

    b_i := S_2.snapshot();
    if every non-⊥ value in b_i is equal to (true, v) then
        return (commit, v);
    if some value in b_i is equal to (true, val) for some val then
        return (adopt, val);
    return (adopt, v);
```

Is the above implementation correct (does it satisfy the *commit-adopt* properties)? Justify your answer.

Solution

Yes, the implementation is correct.

Validity All non-$\perp$ values in S_1 are proposed values. Therefore, if a process p writes (true, v) or (false, v) in S_2, then v must have been proposed by some process q (possibly by p itself). Since the output value of a process is taken either from S_1 or from S_2, validity is satisfied.

Lemma 1 *If S_2 contains two entries $(true, v_1)$ and $(true, v_2)$, then $v_1 = v_2$.*

Proof Assume not. Since every process writes in S_1 and S_2 at most once, it must be that some process p_1 wrote (true, v_1) and some other process p_2 wrote (true, v_2). Thus, it must be that p_1 wrote v_1 in S_1, took a snapshot of S_1 and only saw v_1 in that snapshot. Similarly, it must be that p_2 wrote v_2 in S_1, took a snapshot of S_1 and only saw v_2 in that snapshot. This is impossible: since the snapshot object is atomic and the processes update S_1 before scanning, it must be that either p_1 saw p_2's value, or vice-versa. We have reached a contradiction. □

Commitment Assume all proposed values are equal. Then no process can write $(\text{false}, \cdot)$ in S_2; S_2 contains only entries of the form $(\text{true}, \cdot)$. By Lemma 1, all such entries have equal values, so all processes that return must commit.

Agreement In order for a process p to commit v, p must write v to S_1, scan S_1 and see only entries equal to v; p must then write (true, v) to S_2, scan S_2 and see only entries equal to (true, v) and finally return (commit, v).

Assume by contradiction that process p commits v and some process q commits or adopts $v' \neq v$. q's scan of S_2 cannot include the (true, v) entry written by p, otherwise q would adopt v (remember that by Lemma 1, q cannot see any entry (true, v') with $v' \neq v$ in S_2 if p has already written (true, v) to S_2). Therefore, q's scan of S_2 must happen before p's write to S_2. Furthermore, q's scan of S_2 must include some entry $e = (\cdot, v')$ with $v' \neq v$ (written either by q or some other process). But then p's scan of S_2 (which is after p's write to S_2 and therefore after q's scan of S_2) will also include e, and thus p cannot commit v. We have reached a contradiction.

Termination The code does not contain any waiting, loops, or goto statements, and the snapshot objects are wait-free, so every correct process will return in a finite number of steps.