Concurrent Computing October 28, 2024

Exercise 4 Solution

Problem 1. We can use a single write-once register r to implement binary consensus as follows.

```
upon propose(v)
  r.write(v)
  return r.read()
```

A decided value is a value stored in r that was proposed by some process (validity). Register r changes only once (i.e., from $\perp$ to $\neq \perp$), so no two processes can read different values from r (agreement).

Problem 2. Assume for the sake of contradiction that there exists an algorithm $\mathcal{A}$ that solves consensus in an asynchronous system using MRMW atomic registers. We prove that in this case there exists an algorithm $\mathcal{A}'$ that solves consensus in an asyncronous system using SRSW safe registers, which contradicts the FLP theorem.

Let us write down the code of algorithm $\mathcal{A}$. The algorithm makes use of a (possibly infinite) number of MRMW atomic registers $R_1, R_2, \ldots$. We know from the course that there exists an implementation of MRMW atomic registers using (an infinite number of) SRSW safe registers. Therefore, for each call of R_i.read or R_i.write that the algorithm $\mathcal{A}$ makes, we replace the call with the implementation of R_i using only SRSW safe registers. We call the resulting algorithm code $\mathcal{A}'$.

To conclude the proof, we notice that algorithm $\mathcal{A}'$ implements consensus using only SRSW registers. The correctness of $\mathcal{A}'$ follows from the (assumed) correctness of $\mathcal{A}$ and the correctness of the register implementation, which has been shown in class.

Problem 3. Follows from 1 and 2.

Problem 4. Shared object X has consensus number ∞. Figure 1 presents an algorithm that solves binary consensus among an arbitrary number n of processes, using X (and arbitrarily many shared registers). The algorithm is wait-free as it terminates in a finite number of steps. We can show that the algorithm solves consensus by verifying the two properties of consensus. By contradiction, suppose that the algorithm violates validity. Thus, some process decides $b, b \in \{0, 1\}$ while all processes propose $1 - b$. According to the algorithm, however, if all processes propose $1 - b$, then all processes follow the same code. Either every process does Test&Set() and every one sees $a \in \{0, 1\}$, or every process does Fetch&Add2() and every one sees $a \mod 2 = 0$. Thus validity can not be violated Now suppose that the algorithm violates agreement. Thus, some process p decides 0 while some other process $q, q \neq p$ decides 1. By validity, both 0 and 1 are proposed. Then there must be at least two processes one of which does Test&Set() and the other of which does Fetch&Add2(). According to the linearization of object X, if Test&Set() is the first operation in the linearization, then the value x maintained by X can either be 0 (the initial value) or satisfy $x \mod 2 = 1$; if Fetch&Add2() is the first operation in the linearization, then the value x maintained by X satisfies $x \mod 2 = 0$. In either case, no process returns a different value from the proposal of the process who does the first operation in the linearization of object X. Therefore, agreement is satisfied.

```
Using X;
propose(v) {
    if(v = 1)
        a = X.Test&Set();
        if(a = 0 or a mod 2 = 1)
            return 1;
        else
            return 0;
    else if(v = 0)
        a = X.Fetch&Add2();
        if(a mod 2 = 0)
            return 0;
        else
            return 1;
}
```

Figure 1: Shared object X has ∞ consensus number.