Concurrent Algorithms October 25, 2016

Exercise 4

Problem 1. As given in the lecture, a queue is a shared object with two operations: enq(x) and deq(). Now we augment the queue object with an operation *peek* that returns but does not remove the first element in the queue. You task is to find an algorithm that implements consensus among n processes using the augmented queue object and atomic registers for an arbitrary n. (You may use any number of the augmented queue objects and atomic registers.)

Problem 2. A queue is implemented using two objects: fetch&inc (augmented with operation *read*) and *swap* as follows. A swap object is a shared object with an operation *swap*(v) that atomically replaces the current value of the object with v and returns the prior value. (The swap object has also operation *read*.)

```
Using a fetch&inc object head, initialized to 0.
Using an array of swap objects (of infinite length) items, initialized to ⊥.

enq(x):
    slot = head.fetch&inc();
    items[slot].swap(x);
    return ok

deq():
    while(true) do
        limit = head.read();
        for j = 0 to limit - 1 do
            y = items[j].swap(⊥);
            if (y is not ⊥) then
                return y
```

You may assume this queue is linearizable, and wait-free as long as deq() is never applied to an empty queue.

Consider the following sequence of statements:

- Both fetch&inc and swap objects have consensus number 2 (i.e., there is an algorithm that implements consensus using the shared object and atomic registers among two processes, while there is no algorithm that implements consensus using the shared object and atomic registers among three or more processes).
- We can add an operation peek() simply by taking a snapshot of the queue (using the snapshot algorithm studied in previous lectures) and returning the item at the head of the queue.
- Using the protocol devised for Problem 1, we can use the resulting queue to implement consensus among n processes for an arbitrary n.

We have just implemented consensus using only objects with consensus number 2. Your task is to identify the faulty step in this chain of reasoning, and explain what went wrong.

Problem 3. A *k-set-agreement* object is a generalization of a consensus object in which processes could decide up to k different values. Formally, k-set-agreement is defined as follows. It has an operation *propose*(v) that returns (or we say *decides*) a value, which satisfies the following properties:

1. *Validity:* Decided values are proposed values.
2. *Agreement:* At most k different values could be decided.
3. *Termination:* Every correct process eventually decides a value.

A *k-simultaneous-consensus* object is another generalization of a consensus object in which processes could decide k values simultaneously. Formally, k-simultaneous consensus is defined as follows. It has an operation *propose*($v_1, \dots, v_k$) that returns (or we say *decides*) a pair $(index, value)$ with $index \in \{1, \dots, k\}$, which satisfies the following properties:

1. *Validity:* If a process decides (i, v), then some process proposed $(v_1, \dots, v_k)$ with $v_i = v$.
2. *Agreement:* If two processes decide (i, v) and (i', v') with $i = i'$, then $v = v'$.
3. *Termination:* Every correct process eventually decides a value.

Your task is to show that k-set-agreement and k-simultaneous-consensus are equivalent. That is, you have to show that one implements the other.

Hint: When implementing k-consensus using k-set-agreement, an algorithm that solves the problem is the following:

```
1: function KSC.PROPOSE($v_1, \dots, v_k$)
2:     $V_i \leftarrow [v_1, \dots, v_k]$
3:     $dV_i \leftarrow kSA.\text{PROPOSE}(V_i)$
4:     $REG[i] \leftarrow dV_i$
5:     $snap_i \leftarrow REG.\text{snapshot}()$
6:     $c_i \leftarrow$ number of distinct (non-$\perp$) vectors in $snap_i$
7:     $d_i \leftarrow$ minimum (non-$\perp$) vector in $snap_i$
8:     return$\langle c_i, d_i[c_i] \rangle$
9: end function
```

Where $REG[0, \dots, n-1]$ is an array of single-writer multi-readers atomic registers initialized at $\perp$. Processes write atomically a *vector of values* in their register (Line 4). $REG.\text{snapshot}()$ returns an atomic snapshot of this array of registers. Consequently, $snap_i[0, \dots, n-1]$ is an array of vectors, possibly containing $\perp$ values for some indices. We suppose that there is an order on the set of values that can be proposed, and we use the induced *lexicographic order* on vectors at Line 7.

Your task is then to (1) prove that the algorithm above implements a k-simultaneous consensus from k-set agreement objects and atomic registers; and (2) find an algorithm that implements a k-set agreement object using k-simultaneous consensus objects and atomic registers.