Tutorial 6 Problems

Problem 1 Concurrent reads in *Split-SC*

In class, we discussed a version of *SC*, *Split-SC*, that together with *Env* behaves like *CM*. This was achieved through the use of locks. Each read/write access to a location sets a lock on that location and no further read/writes are possible on that location till the lock is released. Although this works correctly, it is inefficient as it forces the reads to happen sequentially, which is really not necessary (convince yourself about this). Modify the behavior of *Env* to allow concurrent reads.

The original specification of *Env* is reproduced below:

Q: *locks*, a mapping from A to I,
indicating for each memory address, which caches have currently active operations
Q_0: for all a, $locks(a) = \emptyset$
L: Input and output reverse those of *Split-SC*:

Input:

$respond(v, i)$, $v \in V$, $i \in I$

$respond(OK, i)$, $i \in I$

Output:

$request(read(a), i)$, $a \in A$, $i \in I$

$request(write(a, v), i)$, $a \in A$, $v \in V$, $i \in I$

Internal: None

Transitions:

$respond(*, i)$

Can occur: anytime

Effect: for all a, $locks(a) := locks(a) - \{i\}$

(That is, throw away locks held by i.)

$request(read(a), i)$

Can occur: $locks(a) = \emptyset$

Effect: $locks(a) := \{i\}$

$request(write(a, v), i)$

Can occur: $locks(a) = \emptyset$

Effect: $locks(a) := \{i\}$

Solution:

The only transition of *Env* that needs to be changed is:

$request(read(a),i)$

Can occur: $locks(\mathrm{a}) = \emptyset$
Effect: Nothing

In other words, now we no longer place a lock on a location if it's a read operation. It's clear that the composition of *Split-SC* and *Env* still implements *CM*.