

---

## Homework 4

Here are two problems. The first is from hw2 and has additional requirements. The second is from hw3 and additional info is on the next page.

1. Define a **reliable broadcast channel** to be a channel such that: (1) every message sent is eventually delivered to all addresses (including the sender's address); and (2) all addresses receive messages in the same order. Develop a service program for such a channel. A skeleton is provided below.

```

service RelBroadcastChannel(Addr, v) { // reliable broadcast channel
  // v[j]: channel subsystem at address j
  ia {And(Addr.size ≥ 2, Map<Addr, Sys*> v, distinctNonNull(v,mysid))
  ....
  input void v[Addr j].tx(Seq msg) {...}
  input Seq v[Addr j].rx() {...}
  atomicity assumption {...}
  progress assumption {...}
} // RelBroadcastChannel

```

Your program must allow users at different addresses to call tx simultaneously. It must not require messages to be delivered in the global order in which they were sent; for example, given a sequence of sends,  $v1.tx(m1)$ ,  $v2.tx(m2)$ ,  $v3.tx(m3)$ , it should allow the messages  $m1$ ,  $m2$ ,  $m3$  to be received in any order. (Requiring receptions in the global send order would be horribly inefficient.)

2. In the analysis of the sliding window protocol with lossy channel (in section 4.4), we gave an operational argument that  $A_5$  is an invariant of program  $X$ , where  $X$  is the closed system of the sliding window protocol and the point-to-point fifo channel inverse. Provide an assertional proof of this. Specifically, obtain predicate(s)  $B$  such that
  - (1)  $B$  holds immediately after the initialization of  $X$ .
  - (2) for every atomic step  $e$  of  $X$ : executing  $e$  in a state satisfying  $B$  and  $A_2, \dots, A_6$  results in a state that satisfies  $B$  and  $A_5$ .

(Hint: Identify every atomic step  $e$  of  $X$  that affects  $A_5$ . For each such step  $e$ , identify the predicate that must hold (immediately) before  $e$ 's execution in order for  $A_5$  to hold (immediately) after the step.)

---

## Making headway on the second problem

$A_5 : ((k \text{ in } rbuff.keys) \text{ and } (data \text{ msg } j \text{ receivable})) \Rightarrow j \geq k+1-N+RW$

The atomic steps that affect  $A_5$  can be classified as follows:

- E1. Transmission of a data message.
- E2. Reception of a data message that does not change rbuff.
- E3. Reception of a data message that changes rbuff.

### Making headway with E1

We want a predicate  $B_1$  such that  $\{B_1, A_{2-6}\} E1 \{A_5\}$  holds; that is, if E1 executes in *any* state where  $B_1$  and  $A_{2-6}$  hold, the resulting state satisfies  $A_5$ .

E1 introduces a new data message, say with unbounded sequence number  $j1$ . In order for this to not falsify  $A_5$ , we want  $j1 \geq k+1-N+RW$  to hold. All we know about  $j1$  is that it is at least  $na$ . So it suffices if  $na \geq k+1-N+RW$  holds before E1. So  $\{B_1, A_5\} E1 \{A_5\}$  holds, where  $B_1$  is defined as follows:

$B_1 : ((k \text{ in } rbuff.keys) \text{ and } (ns > na)) \Rightarrow na \geq k+1-N+RW$

A little reflection tells us that we expect the consequent to hold even if  $ns$  equals  $na$ . So we can simplify  $B_1$  to the following:

$B_1 : (k \text{ in } rbuff.keys) \Rightarrow na \geq k+1-N+RW$

Now do the same thing with  $B_1$ : find out which atomic steps affect it and what is needed to preserve it.

### Making headway with E2

What must hold before this step in order for  $A_5$  to continue to hold after the step? The *only* change in the entire system is that a data msg is received ( $nr$  changes only if rbuff changes). So the remaining data messages continue to satisfy whatever they did before. So  $\{A_5\} E2 \{A_5\}$  holds. So E2 makes no contribution to  $B$ .

### Making headway with E3

Suppose E3 data msg  $k1$  is received (and it was not previously in rbuff). What must hold just before in order for  $A_5$  to hold just after? If  $k1$  is less than some  $k$  that was in rbuff before the step, then  $A_5$  before the step implies  $A_5$  after the step. So let  $k1$  be the highest key in rbuff after the step....

### And so on

So you keep doing this, identifying new predicates that need to hold. At some point, if you are doing things correctly, one of three things happen:

- You have a list of predicates, say  $B_1, B_2, \dots$ , whose conjunction is the desired  $B$ .
- Or one of your predicates, say  $B_j$ , is obviously not invariant, i.e., you can write down an evolution of the system that ends in a state that satisfies *not* $B_j$ . In this case, working backwards from where you obtained  $B_j$ , you can find an evolution that does not satisfy  $A_5$ . So you would have proved that  $A_5$  is not invariant.
- Or list of predicates keeps growing in complexity and becomes unmanageable. Then either the problem you are trying to solve is very hard or you are not simplifying expressions as you go along.

Of course, you know the desired outcome...