Dairon Andrés Benites Aldaz
DD2443 HT22 (52168) Parallel and Distributed Computing
September 9, 2022

Lecture 4. Exercise

1. Hand-in

Question 1. HSLS 4.5. Consider the atomic MRSW construction shown in Fig. 4.12. True or false: If we replace the atomic SRSW registers with regular SRSW registers, then the construction still yields an atomic MRSW register. Justify your answer.

```
1  public class AtomicMRSWRegister<T> implements Register<T> {
2    ThreadLocal<Long> lastStamp;
3    private StampedValue<T>[][] a_table; // each entry is an atomic SRSW register
4    public AtomicMRSWRegister(T init, int readers) {
5      lastStamp = new ThreadLocal<Long>() {
6         protected Long initialValue() { return 0; };
7      };
8      a_table = (StampedValue<T>[][]) new StampedValue[readers][readers];
9      StampedValue<T> value = new StampedValue<T>(init);
10     for (int i = 0; i < readers; i++) {
11       for (int j = 0; j < readers; j++) {
12         a_table[i][j] = value;
13       }
14     }
15   }
16   public T read() {
17     int me = ThreadID.get();
18     StampedValue<T> value = a_table[me][me];
19     for (int i = 0; i < a_table.length; i++) {
20       value = StampedValue.max(value, a_table[i][me]);
21     }
22     for (int i = 0; i < a_table.length; i++) {
23       if (i == me) continue;
24       a_table[me][i] = value;
25     }
26     return value;
27   }
28   public void write(T v) {
29     long stamp = lastStamp.get() + 1;
30     lastStamp.set(stamp);
31     StampedValue<T> value = new StampedValue<T>(stamp, v);
32     for (int i = 0; i < a_table.length; i++) {
33       a_table[i][i] = value;
34     }
35   }
36 }
```

FIGURE 4.12

The `AtomicMRSWRegister` class: an atomic MRSW register constructed from atomic SRSW registers.

Atomic registers satisfy the following conditions, regular registers only the first two:

(1) $$\text{Never: } R^i \to W^i$$

(2) $$\text{Never for some: } j : W^i \to W^j \to R^i$$

(3) $$\text{If: } R^i \to R^j \text{ then } i \leq j$$

The construction which we arrive at by substituting the array of atomic registers `a_table` with an array of regular registers `r_table` yields an atomic MRSW register.

The only way that regular and atomic registers diverge is in how they handle reads and writes that overlap. The revised architecture also meets the second criterion because the second property deals with non-overlapping writes and the original construction from the figure is an atomic MRSW register.

The third condition, which describes how non-overlapping reads behave, needs to be demonstrated. A read must yield a value that is earlier than a previous, non-overlapping read in order to breach this requirement. This is not possible because each read inserts its value into an entire row. This requirement is also satisfied because the later read will retrieve the most recent value read by any previous read by fetching the maximum `StampedValue` over a column.

We only need to show that the third condition is not satisfied; `read()` determines some value from `r_table`, which is returned as the result. Given that the value is taken from a regular register (which does not satisify the third condition), it is not guaranteed to comply with it.

Question 2. HSLS 4.6. Give an example of a quiescently consistent register execution that is not regular.

A quiescently consistent behavior that cannot be generated by a regular register occurs when a thread reads a value from the past write calls instead of the most recent one.

$Thread_a := writes(1)$ before it finishes $Thread_b := reads(1)$ Before $Thread_b$ finishes reading $Thread_a$ starts to write 2 after write 1 instructions has finished. Then once write 2 is done thread B reads once more, but it reads 1 again instead of 2. It follows the rules to satisfy quiescent consistency but doesn't behave like some regular register.

Question 3. HSLS 4.10. Consider the following implementation of a register in a distributed, message passing system. There are n processors P0,...,Pn1 arranged in a ring, where Pi can send messages only to Pi+1 mod n. Messages are delivered in FIFO order along each link. Each processor keeps a copy of the shared register.

- To read the register, the processor reads the copy in its local memory.

- A processor Pi starts a write() call of value v to register x, by sending the message "Pi: write v to x" to Pi+1 mod n.
- IfPi receivesamessage"Pj:writevtox,"fori=j,thenitwritesvtoitslocal copy of x, and forwards the message to Pi+1 mod n.
- If Pi receives a message "Pi: write v to x," then it writes v to its local copy of x, and discards the message. The write() call is now complete. Give a short justification or counterexample.

If write() calls never overlap,

- Is this register implementation regular?

Because no read call can return a value from the future, the implementation is **regular**. Because the registers won't contain any future values until the write instruction begins, it can return a value from the future. Additionally, they are unable to return values from the past due to the fact that after the write instruction completes, the registers are completely modified and contain no residual values.

- Is it atomic?

It is not atomic. Since, This implementation is not atomic since later reads from P_{i+1}. can see the written old values. while P_i observes the new value being written, in the register. Thus, the register is not atomic and thus not linearizable.

If multiple processors call write(), • is this register implementation safe?

It is not.

Since, $Multiple - processors :_{(write)(some_n) \Longleftrightarrow Some-operation::(read_i)}$ which returns a value $R_j, i <= j$