Parallel and Distributed Computing
DD2443 - Pardis24
Exercises for Lecture 4

Name: Casper Kristiansson

September 1, 2024

Exercise 1

Question

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.

Answer

If we replace the atomic SRSW registers with regular SRSW registers, then the constriction will not yield an atomic MRSW register. The reason behind this is that an atomic SRSW register guarantees that every read is reflected on the most recent write that has happened. Using this allows for consistency and correct order of operations on parallel systems.

A regular SRSW register does not provide this type of guarantee. This means that on a regular register, an operation such as read could return an outdated value. The fundamentals of an MRSW register are that all reads and writes need to be consistent. This means that replacing an atomic SRSW register with a regular register would break the consistency and therefore the statement is false.

Exercise 2

Question

Give an example of a quiescently consistent register execution that is not regular.

Answer

A quiescent consistency is when a register returns values of a read operation that is in a consistent state. A regular register in this case is a register that on a read operation returns either a value of the most recent write operation or the value of some earlier write operation (due to the lack of constituency).

An example situation of quiescently consistent but not regular register execution could for example happen in the following scenario:

- Register initial state is: 1
- Thread1 Write the value 2 to the register
- Thread2 performs a read operation after thread1 is done writing and therefore reads the value 1
- Thread3 writes the value 3 to the register after thread1's write operation
- Thread4 reads the value 3 from the register

In this scenario, it is quiescently consistent because any more read operations will return the latest value which in this case is 3. The operation is not regular because thread2 reads the value 1 after a write has occurred but no finished.

Exercise 3

Question

Consider the following implementation of a register in a distributed, message-passing system. There are n processors $P_0, \ldots, P_{n-1}$ arranged in a ring,

where P_i can send messages only to $P_{(i+1) \bmod n}$. 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 P_i starts a `write()` call of value v to register x, by sending the message "P_i: write v to x" to $P_{(i+1) \bmod n}$.**
- **If P_i receives a message "P_j: write v to x," for $i \neq j$, then it writes v to its local copy of x, and forwards the message to $P_{(i+1) \bmod n}$.**
- **If P_i receives a message "P_i: 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?**
- **is it atomic?**

If multiple processors call `write()`,

- **is this register implementation safe?**

Answer

A regular register in this case is a register that on a read operation returns either a value of the most recent write operation or the value of some earlier write operation (due to the lack of constituency). An atomic register is a register where every read and write operation happens instantaneously meaning it is consistent and you will always have access to its latest value. A register is considered safe where every read operation returns the most recent written value but if it is happening at the same time as a write it could return an older value.

For the first situation where the `Write()` calls never overlap the register implementation can be considered to be regular. The reason behind this is that when the `Write()` calls never overlap the process will be fully completed before the next one happens. Doing it this way will make sure that the register is consistent and each read will return the most recent value. The action can though not be considered atomic because for it to be considered atomic the write operation has to happen in a single instant. With the current setup, a read might not get the latest write value.

For the second situation where the `Write()` calls happen at the same time. With this setup, the register is not considered safe with this implementation. The reason for this is that if multiple writes are happening at the same time the processor might receive multiple writes to different parts of it. The issue with this is that when the inner processors might share a value they might be inconsistent. If any read operations happen during this time it will return a wrong value which isn't the most recent. A safe register requires that the latest read should always return the most recent value.