Concurrent Algorithms September 23, 2025

Exercise 1.5 (Extra Exercise) Solution

Problem 1. In each of the following scenarios, is the shared object *atomic*, *regular*, *safe*, or *none-of-the-above*? Explain why.

1. *Atomic*. x86-64 processors guarantee that reads and writes by a single thread are observed consistently.
2. *None-of-the-above*. A read that is not concurrent with a write is not guaranteed to see the most recent value, since the latest write may still be buffered. This violates the guarantees required even of a safe register.
3. *Atomic*. (Here we assume the fence is part of the write operation, and that the write completes only once it has been flushed.)
4. *Safe*. A read concurrent with a write may observe the array in an intermediate state.
5. *Safe*. If two writes are concurrent with a read, the reader may miss the effect of the earlier write while still observing the effect of the later one. This depends on which array elements are modified and on the order in which the reader traverses the array—for example, if the later write modifies an element read after the element modified by the earlier write.

Problem 2.

1. When two reads are concurrent with the same write, the writer may have updated only one of the three unreliable registers so far. As a result, the value read can oscillate between the old and new values, depending on whether the register containing the new value is among the two read by the reader.
2. A simple modification is to maintain `latest_ts` and `latest_v` as persistent variables at the reader (i.e., initialized outside the read operation). By reusing them on each read, we ensure that once a new value has been observed, no older value can be returned later.
3. The above modification does not ensure atomicity when multiple readers are present.