Concurrent Algorithms September 23, 2025

Exercise 1.5 (Extra Exercise)

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

1. On x86-64, consider a 64-bit integer stored in RAM that is read and modified by a single thread (the same thread both reads and writes).
2. The same as (1), but the value can be read by multiple threads. (Hint: x86-64 allows writes to be buffered locally, making them immediately visible locally but visible only later from other cores, potentially after subsequent read instructions. Search for TSO and x86 store buffer for more details.)
3. The same as (2), but with an MFENCE after each write operation (MFENCE ensures that prior writes are visible to other cores (flushed) before any subsequent operation is executed).
4. An array of integers, modified by a single writer and read by multiple readers, where each individual integer is accessed atomically.
5. The same as (4), except that each write affects at most one element of the array, atomically.

Problem 2. You have access to three multi-reader multi-writer (MRMW) atomic multi-valued registers, but at any time, one of them will be unavailable, either temporarily or indefinitely, and the one being unavailable can change over time. Using those 3 unreliable registers, you want to build a single-writer reliable atomic multi-valued register that is always available. You are given access to an oracle which can predict which two registers will be available for the next two operations (one operation on each register).

Each unreliable register is initialized to `(0, ⊥)`. You are given the following implementation of read and write operations:

```
 // A value local to the writer:
timestamp := 0

operation write(v):
  timestamp := timestamp + 1
  for reg in oracle.get_two_available_registers():
    reg.write((timestamp, v))

operation read():
  latest_ts := 0
  latest_v := ⊥
  for reg in oracle.get_two_available_registers():
    (ts, v) := reg.read()
    if ts > latest_ts:
      latest_ts := ts
      latest_v := v
  return latest_v
```

1. Give a counterexample showing that this implementation is not atomic with a single reader.

2. Propose a simple modification that makes the register atomic when there is a single reader.
3. Does your modification in (2) also ensure atomicity with multiple readers?