Concurrent Algorithms October 14, 2019

Solutions to Exercise 2

Problem 1. Omitted.

Problem 2. Figure 1 presents an example that violates atomicity. Such an execution can occur if the first *read* operation of p_2 gets 0 while retrieving $Reg[7]$ and gets 1 while retrieving $Reg[1000]$. This can occur since both $write(7)$ and $write(1000)$ are concurrent with the *read* operation. Afterwards, the second *read* operation of p_2 will return 7 since $write(1000)$ has not yet set $Reg[7]$ to zero.

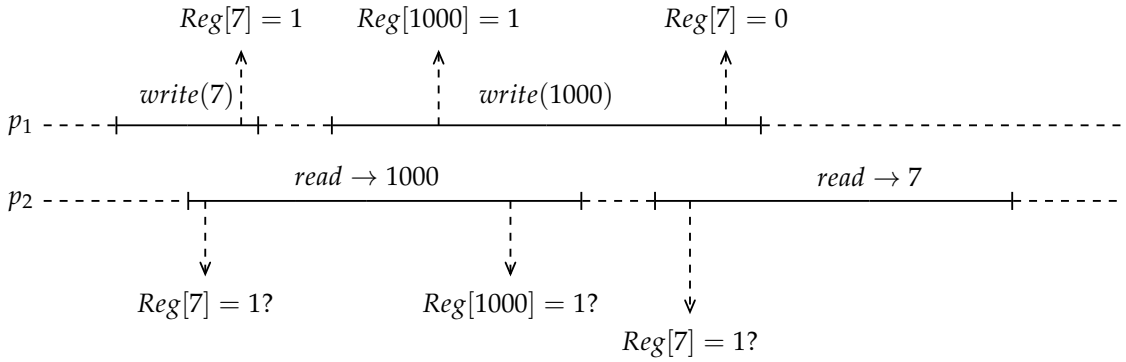

Figure 1: Execution that violates atomicity.

Problem 3. Hint: notice that if the writer first clears the array by writing 0's, it is possible for the value of the array to be all 0's, which is not a valid state. An example execution omitted.

Problem 4. The transformation does not work for multiple readers (the result is not an atomic register). The non-atomic execution in Figure 2 is possible in this case. Since the register is regular, the read by R1 may read the value 2 being concurrently written by W. Since this is R1's first read operation, the timestamp it obtains for value 2 is higher than its local timestamp. Later, the read by R2 (also concurrent with *Write(2)*) may read the previous value of the register (1). Since this is R2's first read operation, the timestamp it obtains for value 1 is also higher than its local timestamp.

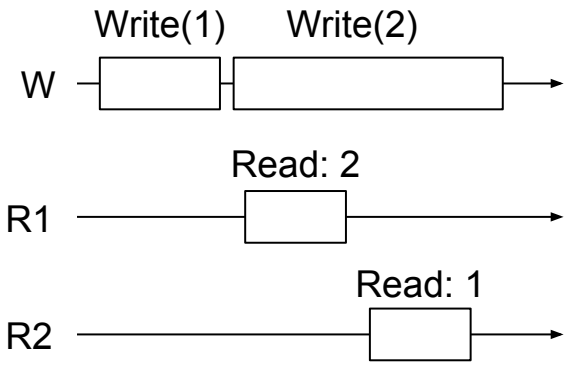

Figure 2: An execution that is possible with a regular register but not with an atomic register. There are three processes: a writer (W) and two readers (R1 and R2). The two reads are concurrent with the second write. The read by R1 completely precedes the read by R2. The execution is not atomic because it is impossible to assign linearization points to all operations: if the linearization point of *Write(2)* is before that of the read by R1, then the read by R2 cannot have a linearization point; if the linearization point of *Write(2)* is after that of the read by R1, then that read cannot have a linearization point.