Concurrent Algorithms October 8, 2019

Solutions to Exercise 1

Problem 1.
Part 1.a. Regular, not atomic.

Part 1.b. None of the above.

Part 1.c.
Atomic.

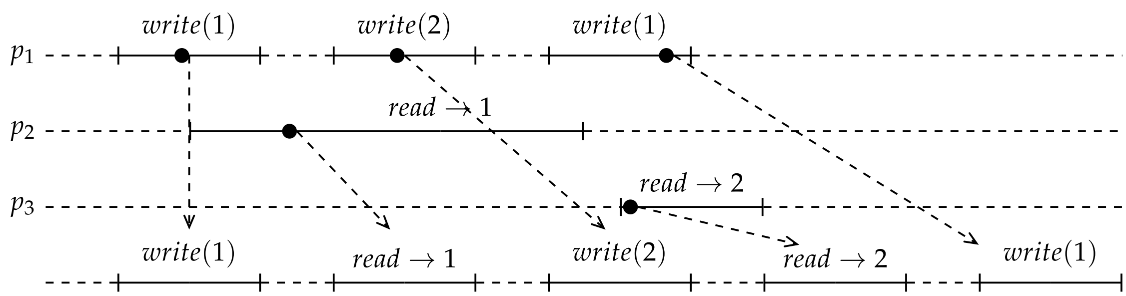

Figure 1: Serialization points and an equivalent sequential execution.

Problem 2. Consider the transformation from (binary) SRSW safe to (binary) MRSW safe registers given in class.

Part 2.a. Prove that the transformation works for multi-valued registers and regular registers.

When a process p_i reads the base regular register $Reg[i]$, p_i gets (a) the value of a concurrent write on $Reg[i]$ (if any) or (b) the last value written to $Reg[i]$ before such concurrent write operations. In case (a), the value v obtained is from a $R.write(v)$ that is concurrent with the *read* of p_i. In case (b), the value v obtained can either be (b.1) from a $R.write(v)$ that is concurrent with the *read* of p_i, or (b.2) from the last value written by a $R.write()$ before the *read* of p_i . Thus, the constructed register is regular.

Part 2.b. Also, prove that the transformation does not work for atomic registers (by providing a counterexample that breaks atomicity).

See execution in Figure 2.

Problem 3. Consider the transformation from binary MRSW safe registers to binary MRSW regular registers, given in class.

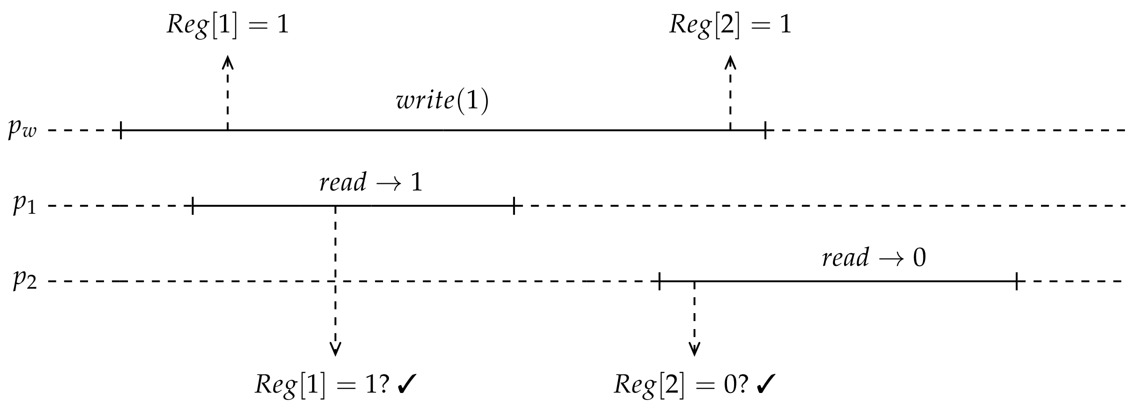

Figure 2: Execution that violates atomicity.

Part 3.a. Prove that the transformation does **not** generate multi-valued MRSW regular registers (by providing a counterexample that breaks regularity).
If the registers are multi-valued, then two consecutive reads on the safe register *Reg* may return arbitrary values, breaking regularity of the register implementation. Since the safe register is binary in the correct implementation (and thus limited to two values), this does not occur in the transformation given in class.

Part 3.b. Also, prove that the resulting registers are not binary atomic (by providing a counterexample that breaks atomicity).
The counterexample can be easily built by scheduling two distinct reads during a write(1) operation on the register. Since the underlying register is safe, we can ensure that the first operation returns 1, while the second (non-overlapping) operation returns 0, contradicting atomicity.