Exercise 6

1 Emulating a $(1, N)$ Register from $(1, 1)$ Registers

Consider the implementation in Algorithm 1 of a $(1, N)$ register, instance onr, from an array of N instances of so-called *base registers*. This algorithm sends no messages explicitly, it merely reduces one abstraction to another one. It is an example of an algorithm in the so-called *shared memory*-model.

The unique writer process of the $(1, N)$ register is p. The base registers are $(1, 1)$ registers, denoted *br*.q for $q \in \Pi$, such that only process p may write to instance *br*.q and only process q may read from it. (Recall that *self* denotes the process executing the algorithm. The consistency property of the registers, whether they are safe, regular, or atomic, is specified later.)

Algorithm 1: Multi-Reader Emulation

Implements:
  $(1, N)$-Register, **instance** *onr*. // the writer is p

Uses:
  $(1, 1)$-Register (multiple instances).

upon event ⟨ *onr*, *Init* ⟩ **do**
  writeset := ∅;
  forall $q \in \Pi$ **do**
    Initialize a new instance *br*.q of $(1, 1)$-Register with writer p and reader q;

upon event ⟨ *onr*, *Read* ⟩ **do**
  trigger ⟨ *br.self*, *Read* ⟩;

upon event ⟨ *br.self*, *ReadReturn* | v ⟩ **do**
  trigger ⟨ *onr*, *ReadReturn* | v ⟩;

upon event ⟨ *onr*, *Write* | v ⟩ **do** // only the writer p
  forall $q \in \Pi$ **do**
    trigger ⟨ *br*.q, *Write* | v) ⟩;

upon event ⟨ *br*.q, *WriteReturn* ⟩ **do** // only the writer p
  writeset := *writeset* ∪ $\{q\}$;
  if *writeset* $= \Pi$ **then**
    writeset := ∅;
    trigger ⟨ *onr*, *WriteReturn* ⟩;

Answer these questions and justify your answers:

(a) Let the array *br*.q for $q \in \Pi$ be *safe* binary $(1, 1)$-registers. Show that the emulation produces a *safe* binary $(1, N)$-register instance *onr*.

(b) If we replace the N safe registers *br*.q for $q \in \Pi$ with an array of *regular binary* $(1, 1)$-registers (i.e., registers that only store one bit), does the algorithm implement a *regular binary* $(1, N)$-register?

(c) If we replace the N safe registers *br*.q for $q \in \Pi$ with an array of *regular multi-valued* $(1, 1)$-registers, does the algorithm implement a *regular multi-valued* $(1, N)$-register?

2 Reliable Storage with Crashes and Recoveries

In the fail-recovery model we consider crash-recovery process failures [CGR11, Section 2.2.4]. This means that also a correct process may crash, as long as it recovers later. To be more precise, a *correct* process in this model is one that either never crashes or one that eventually recovers and never crashes again. All other processes are *faulty*.

When a process recovers, a special $\langle$ *Recovery* $\rangle$ event is triggered by the runtime system; an algorithm can react accordingly. All local state of a process is lost after a crash, apart from data in *stable storage*. A process has two operations, called *store* and *retrieve*, for writing to and reading from stable storage. The content of stable storage is not affected by crashes.

Modify the "Majority Voting" algorithm for a $(1, N)$ regular register, which was discussed in class, so that it works in the fail-recovery model. Try to store as few variables in stable storage as needed.

Your algorithm should use *stubborn point-to-point links* and *stubborn best-effort broadcast* primitives [CGR11, Sections 2.4.3 and 3.5.1], which are defined just like point-to-point links and best-effort broadcast primitives, except that they deliver every message sent on them not only once (as usual) but over and over (infinitely many times). This is needed for the fail-recovery model, and implies your algorithm should filter duplicate messages.

3 Flooding Uniform Consensus

Can we optimize Algorithm 5.3 [CGR11] (Flooding Uniform Consensus) to save one or more communication rounds? More preicsely, can it be modified such that all correct processes always decide after $N - 1$ or fewer rounds? (Consider a system of two processes only.)