Institute of Computer Science, University of Bern
Distributed Algorithms, Spring 2020

Prof. Christian Cachin
Jovana Mićić

Exercise 4

4.1 Emulating a $(1, N)$ register from $(1, 1)$ registers (3pt)

Recall that a (W, R) register, for $W, R \in \{1, N\}$, permits R processes as readers and W processes as writers, in a system with N processes in total.

Consider the implementation of a $(1, N)$ register, instance *onr*, shown below. It uses an array of N so-called *base register* instances. This algorithm sends no messages explicitly, it merely reduces one abstraction to another one and uses 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.)

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

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

upon event $\langle$ *onr-Init* $\rangle$ **do**
    writeset $\leftarrow \emptyset$;
    forall $q \in \Pi$ **do**
        Initialize a new instance *br*.q of $(1, 1)$-Register with writer p and reader q;

upon event $\langle$ *onr-Read* $\rangle$ **do**
    trigger $\langle$ *br.self-Read* $\rangle$;

upon event $\langle$ *br.self-ReadReturn* $\mid v \rangle$ **do**
    trigger $\langle$ *onr-ReadReturn* $\mid v \rangle$;

upon event $\langle$ *onr-Write* $\mid v \rangle$ **do** // only the writer p
    forall $q \in \Pi$ **do**
        trigger $\langle$ *br.q-Write* $\mid v) \rangle$;

upon event $\langle$ *br.q-WriteReturn* $\rangle$ **do** // only the writer p
    writeset $\leftarrow$ *writeset* $\cup \{q\}$;
    if *writeset* $= \Pi$ **then**
        writeset $\leftarrow \emptyset$;
        trigger $\langle$ *onr-WriteReturn* $\rangle$;

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?

4.2 Multivalued register from binary registers (4pt)

Suppose there are any number of $(1, 1)$ *binary* regular base-registers (*br*) available. Implement a $(1, 1)$ *multivalued* regular register (*rr*).

The challenge with this reduction is the safety property of the regular register. It implies that only the most recently written value or the (unique) concurrently written value may be returned. Suppose the domain of *rr* is $\{1, \ldots, k\}$. The idea is to use an array of k binary base-registers and to define a *unary* encoding of the value in *rr*. The *read* and *write* operations of *rr* traverse the array in opposite directions.

4.3 Register emulations without correct majority? (3pt)

Consider the asynchronous protocol for emulating a $(1, N)$ regular register using majority voting (Algorithm 4.2 in [CGR11]). It relies on a majority of the processes being correct and does not use any failure detector. Suppose there is an eventually perfect failure detector ($\Diamond\mathcal{P}$) available. Can one then relax the assumption of a correct majority? (Either provide a protocol using $\Diamond\mathcal{P}$ or show that it is not possible.)