Problem 1.
Devise an obstruction-free, anonymous algorithm that implements binary consensus using a finite number of (unbounded) counters.

Reminder: a counter object implements two operations: inc, which increments the value of the counter and returns ok, and read, which returns the current value of the counter.

Solution

The following algorithm solves the problem:

uses: $C_0, C_1$ – counters

upon propose($v$) do

while true do

$(x_0, x_1) \leftarrow \text{readCounters}();$
if $x_0 > x_1$ then $v \leftarrow 0$;
else if $x_1 > x_0$ then $v \leftarrow 1$;
if $|x_0 - x_1| \geq n$ then return $v$;

$C_v.\text{inc}();$

The readCounters procedure atomically reads both counters $C_0$ and $C_1$. It can be implemented as follows:

upon readCounters() do

while true do

$x_0 \leftarrow C_0.\text{read}();$
$x_1 \leftarrow C_1.\text{read}();$
$x_0' \leftarrow C_0.\text{read}();$
if $x_0 = x_0'$ then return $(x_0, x_1);$
Problem 2.
Assume the non-responsive base objects model in which a process can execute several operations on base objects concurrently. In other words, a process does not have to wait for a response of its current operation on a base object to invoke another operation on a base object. Is it possible to implement a SWMR atomic register using (any number of) base SWMR atomic registers of which $t$ can be non-responsive?

Solution
We use $2t + 1$ base registers, so that always majority is correct. Read/write from/to majority of registers.

- **uses**: $R[1, \ldots, 2t + 1]$ – SWMR registers $t$ of which can be non-responsive

  - **upon** $write_1(v)$ **do**
    - $ts \leftarrow ts + 1$;
    - invoke $write_1(ts, v)$ on all $R[1, \ldots, 2t + 1]$;
    - wait for $t + 1$ responses;

  - **upon** $read_i$ **do**
    - invoke $read_i(v)$ on all $R[1, \ldots, 2t + 1]$;
    - wait for $t + 1$ responses;
    - return the value $v$ with the highest timestamp $ts$;

The presented algorithm implements a regular SWMR register. However, a regular register can be transformed into an atomic one (see the lecture slides about register transformations).

Problem 3. Assume the non-responsive base objects model in which a process can execute several operations on base objects concurrently. In other words, a process does not have to wait for a response of its current operation on a base object to invoke another operation on a base object. Is it possible to implement a consensus object using (any number of) base C&S objects of which $t$ can be non-responsive?

Solution
Let $P$ be the problem of implementing consensus using base C&S objects, one of which can be non-responsive, and registers (non-faulty). Let $Q$ be the problem of implement consensus using registers in a system of $n > 1$ processes, one of which can crash (we know this problem to be impossible). We perform our proof by contradiction: assume there exists an algorithm $A$ that solves $P$ using $k$ C&S objects, in a system of $n$ processes (one of which can crash). If we find an algorithm $B$ that solves problem $Q$, using $A$ we have reached a contradiction.

- **From registers to non-responsive C&S**: Each of $n$ processes emulates one base C&S object. The processes share a 2-dimensional array $CS$ of registers. When process $i$ wants to invoke the $CAS$ operation of C&S object $x$ it invokes the following:

  - **upon** $CAS_x(oldval,newval)$ **do**
    - $CS[x][i] \leftarrow (invocation, oldval, newval)$;
    - wait until $CS[x][i] = (response, retval)$;
    - return $retval$;

  Since one of the processes can fail, its corresponding C&S object becomes non-responsive. Each process $i$ reads invocations from locations $CS[i][∗]$ and applies them:
parallel task $C_i$
 Initially: $q = \bot$ (local variable)
 while true do
   for $j \leftarrow 1$ to $n$ do
     $(\text{type}, \text{oldval}, \text{newval}) \leftarrow \text{CS}[i][j]$;
     if type = invocation then
       if $q = \text{oldval}$ then $q \leftarrow \text{newval}$;
       $\text{CS}[i][j] \leftarrow (\text{response}, q)$;
   