Final exam 2: Solution

Exam rules:

1. Exam time: from 14.15 to 17.15.
2. The exam is closed book. No electronic devices are allowed.
3. You can use any notation for algorithms, but remember to write which variables represent shared objects (e.g., registers) and which are process-local.
4. Describe shortly the main idea behind every algorithm you give.
5. Keep in mind that only one operation on one shared object (e.g., a read *or* a write of a register) can be executed by a process in a single step. To avoid confusion (and common mistakes) write only a single atomic step in each line of an algorithm.
6. The exam grade will be computed in the following way: 1.0 (for handing in the exam) plus the number of points obtained divided by 2.

Assumptions:

1. We assume an asynchronous, shared-memory system of n processes, out of which $n-1$ might crash.
2. Unless explicitly stated otherwise, we assume that every object is atomic (linearizable) and wait-free.

Good luck!

Problem 1 (2 points)

A *write-once register* is a shared object with the following sequential specification (x is initially equal to $\perp$ and v is always different than $\perp$):

```
upon write(v)
  if x = $\perp$ then x := v
  return ok

upon read()
  return x
```

Your tasks are to:

1. Give an algorithm that implements a consensus object using any number of write-once registers. **(1 point)**
2. Give an algorithm that implements a consensus object using one or more queue objects in a system of 2 processes. **(1 point)**

Remark: Besides write-once registers/queues, you can use any number of atomic, wait-free, MRMW, M-valued registers in your algorithms.

Solution. Refer to the solutions of the Exercise 4 and Exercise 5.

Problem 2 (2 points)

Give an algorithm that implements a regular, M-valued, MRSW, wait-free register using (any number of) safe, binary, MRSW, wait-free registers.

Solution. The required transformations are given in the lectures on register transformations.

Problem 3 (2 points)

An *augmented queue* is a shared object that implements a FIFO queue and provides the following operations:

1. *enqueue(v)* that puts element v at the end of the queue,
2. *dequeue*() that returns the first element from the queue and removes it from the queue, and

3. *peek*() that returns the first element from the queue *without* removing it from the queue.

Your task is to:

1. Give an algorithm that implements wait-free consensus using (any number of) augmented queues and registers (in a system with an arbitrary number of processes). **(1 point)**

2. What is the consensus number of an augmented queue? Explain why. **(1 point)**

Reminder. The consensus number of an object O is the maximum number of processes that can solve the consensus problem using any number of instances of object O and atomic registers.

Solution. Refer to "Wait-Free Synchronization" paper (second reference on the course web page), Section 3.4.

Problem 4 (1 point)

Consider the following (incorrect) implementation of a Fetch&Inc object out of Test&Set objects (infinite array T) and atomic register R (initialized to 0):

```
uses: R - atomic register,
      T[0..] - infinite array of Test&Set objects
initially: R = 0

upon inc()
  k := R.read();
  while T[k].test&set() = 1 do k := k + 1
  R.write(k + 1);
  return k + 1;
end
```

Describe an execution of the algorithm, in which either atomicity (i.e., linearizability) or wait-freedom is violated.

Solution. Assume process p reads 0 from the register. Just before it invokes *test&set*() on T[0] another process executes the whole *inc*(). function. p gets 1 from the *test&set* call and increments k to 1. Just before it invokes *test&set*() on T[1] another process executes the whole *inc*() function. p gets 1 from the *test&set* call and increments k to 2. This can go on forever, so the algorithm is not wait-free.

Problem 5 (1 point)

A *renaming* object is a shared object that provides operation *rename()*, which, when invoked by some process p_i, returns a new *unique* identifier of the process from some set $D = \{1, \ldots, l\}$. It is required that l (the size of set D) is a function of the number k of processes that actually took steps in a given execution. For example, if $l = k^2$ and only two processes invoke operation *rename()*, then (a) each process should get a value from set $\{1, 2, 3, 4\}$, and (b) the processes cannot get the same value.

1. Prove that wait-free renaming is impossible using only (atomic) registers if the new name space (set D) must be of size k (i.e., $l = k$). (Hint: there is a short answer to this question.) **(0.5 point)**

2. Give an algorithm that wait-free implements an atomic renaming object using atomic registers when l is some function of k and l does not depend on n. **(0.5 point)**

Solution.

1. If the wait-free renaming would be possible for $l = k$, it would be equivalent to a strong counter object. We cannot implement a strong counter object using only registers. This means that we cannot implement renaming with $l = k$ using only registers either.

2. The solution can be found in the lecture on renaming.

Problem 6 (2 points)

In the problem of agreement (i.e., consensus), each process p_i proposes a value v_i, and each process p_i later outputs a decision d_i such that the protocol satisfies agreement, validity, and termination. In this problem, we consider the problem of *weak agreement* in which each process has a choice: it can either *commit* to its decision, in which case the regular agreement property must be achieved; or it can *suggest* its decision, in which case disagreement is allowed.

More formally, each process p_i proposes a value v_i, and each process p_i outputs a decision d_i consisting of a pair (dec_i, val_i) where dec_i can be either *commit* or *suggest*. The protocol should satisfy the following properties:

1. Validity: Every dec_i is either *commit* or *suggest*; every val_i is a value v_j proposed by some process p_j.

2. Agreement: If any process decides $(commit, v)$, then every other decision is $d_i = (commit, v)$ or $(suggest, v)$.

3. Convergence: If every process proposes the same value v, then every process can only decide $d_i = (commit, v)$.

4. Termination: Every correct process that proposes a value eventually outputs a decision.

For example, if every process proposes 0, then we require every correct process to commit to 0. On the other hand, if some processes propose 0 and other processes propose 1, then it is possible that no process commits: some processes may suggest 0 and other processes may suggest 1. However, if any process commits to 1, then every other process must either commit to 1 or suggest 1.

Give an algorithm that implements weak agreement using two snapshot objects, S_1 and S_2, and prove the algorithm correct.

Solution. The pseudo code of the algorithm follows:

```
Local variables: ai, bi, arrays of size n

S1[i].write(v)
ai := S1.snap()
if every value in ai is v then
    x := (commit, v)
else
    x := (suggest, v)
S2[i].write(x)
bi := S2.snap()
if every value in bi is equal to (commit, v) then
    return (commit, v)
if any value in bi is equal to (commit, v  ) then
    return (suggest, v  )
return (suggest, v).
```