Concurrent Algorithms 2018
Midterm Exam

November 26th, 2018

Time: 1h45

Instructions:

- This midterm is "closed book": no notes, electronics, or cheat sheets allowed.
- When solving a problem, do not assume any known result from the lectures, unless we explicitly state that you might use some known result.
- 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.
- Remember to write which variables represent shared objects (e.g., registers).
- Unless otherwise stated, we assume atomic multi-valued MRMW shared registers.
- Unless otherwise stated, we ask for *wait-free* algorithms.
- Unless otherwise stated, we assume a system of n asynchronous processes which might crash.
- For every algorithm you write, provide a short explanation of why the algorithm is correct.

- Make sure that your name and SCIPER number appear on **every** sheet of paper you hand in.
- You are **only** allowed to use additional pages handed to you upon request by the TAs.

Good luck!

Problem	Max Points	Score
1	2	
2	3	
3	3	
4	2	
Total	10	

Problem 1 (2 points)

Task. Write an algorithm that implements a MRSW atomic multi-valued register using (any number of) SRSW regular multi-valued registers.

Problem 2 (3 points)

The register-swap operation is an atomic operation with the following sequential specification.

```
1 Variables:
2 Shared MWMR atomic registers A and B.
3 procedure register-swap(A, B)
4     tempA ← A.read()
5     tempB ← B.read()
6     A.write(tempB)
7     B.write(tempA)
```

Figure 1: Sequential specification of the register-swap atomic operation.

Consider an asynchronous shared-memory system with n processes, $n-1$ of which may fail by crashing. Assume atomic MRMW registers, on which you can perform atomic read, write, and register-swap operations.

Task. Write an algorithm that implements wait-free consensus for n processes in this setting.

Problem 3 (3 points)

The sequential specification of a test-and-set object is the following.

```
1 Variables:
2 Binary Register V, initially 0.
3 procedure test-and-set()
4     temp ← V.read()
5     if temp = 0 then
6         V.write(1)
7     return temp
```

Figure 2: Sequential specification of test-and-set.

Consider an asynchronous shared-memory system with three processes, two of which may fail by crashing. Assume that atomic multi-reader multi-writer registers and test-and-set objects are available.

Task. Prove that one cannot solve wait-free consensus in this system.

Problem 4 (2 points)

The augmented double-ended queue object is the same as the queue object except that it supports `enqueue` and `dequeue` operations on both of its ends (HEAD or TAIL). In addition, processes can invoke a total of 3 `peek` operations. Each of the first 3 `peek` calls returns the value at the selected end of the queue. Every subsequent `peek` call returns $\perp$.

The sequential specification of the object using a doubly linked list is given in Figure 3.

```
1  Variables:
2  elements := List(), the list of elements in the queue
3  peeks_invoked := 0, the number of invoked peeks
4  procedure enqueue(end,val)
5      if end = HEAD
6          elements.add_first(val)
7      else
8          elements.add_last(val)
9  procedure dequeue(end)
10     if end = HEAD
11         val ← elements.first()
12         elements.remove_first()
13     else
14         val ← elements.last()
15         elements.remove_last()
16     return val
17     procedure peek(end)
18         if peeks_invoked = 3
19             return ⊥
20         peeks_invoked ← peeks_invoked + 1
21         if end = HEAD
22             return elements.first()
23         else
24             return elements.last()
```

Figure 3: Sequential specification of the augmented double-ended queue.

Task. Write an algorithm that implements a wait-free consensus object in a system of at most 4 processes using augmented double-ended queue objects that support at most 3 `peek` operations and registers.