Concurrent Algorithms 2019
Final Exam

January 27th, 2020

Time: 12h15 - 15h15 (3 hours)

Instructions:

• This exam is “closed book”: no notes, electronics, or cheat sheets are 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 variable represents which shared object (e.g., registers).
• Unless otherwise stated, we assume atomic multi-valued MRMW shared registers.
• Unless otherwise stated, we ask for linearizable and wait-free algorithms.
• Unless otherwise stated, we assume a system of \( n \) asynchronous processes which might crash.

• 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 by the TAs (available upon request).

Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1  (6 points)

Your tasks:

1. (2 points) Explain the difference between a safe register and an atomic register. Provide an example execution that is allowed for a safe register but not allowed for an atomic register.

2. (4 points) Write an algorithm that implements an $M$-valued MRMW atomic register using (any number of) $M$-valued SR5W atomic registers.
Problem 2  (2 points)

You are given the pseudocode of the write operation of a write-once multi-valued SRSW atomic register that uses single-bit (binary) atomic registers (see below). Remember that the multi-valued register is write-once, meaning that your read will overlap at most one write. Furthermore, note that function Base2Conversion converts from decimal to binary system. You can assume the existence of a function Base10Conversion that converts from binary to decimal system.

Using: $b[3 \times M]$ array of atomic single-bit registers initialized to 0;

```c
write(x) { // x \geq 0 and log_2(x) < M
    v \leftarrow \text{Base2Conversion}(x);
    \text{for } i = 1 \text{ to } M \text{ do }
        b[i] \leftarrow v[i];

    \text{for } i = 1 \text{ to } M \text{ do }
        b[M + i] \leftarrow v[i];

    \text{for } i = 1 \text{ to } M \text{ do }
        b[2 \times M + i] \leftarrow v[i];
}
```

Your task:
Devise the read operation of this register and justify its correctness.
**Problem 3  (2 points)**

Consider the following modified implementation of the obstruction-free consensus object taught in class. The following algorithm uses atomic multi-valued MRMW shared registers in a system of $n$ processes. A process’s id is known to itself as $i$.

Using: an array of atomic multi-valued MRMW shared registers $T[1, 2, ..., n]$, initialized to 0;
Using: an array of atomic multi-valued MRMW shared registers $V[1, 2, ..., n]$, initialized to $(\bot, 0)$;

propose($v$) {
    $ts := i$;
    while (true) do{
        $T[i].write(ts)$;
        $maxts := 0$;
        $val := \bot$;
        for $j = 1$ to $n$ do 
            $(vt, t) := V[j].read()$;
            if $maxts < t$ then
                $maxts := t$;
                $val := vt$;
        
        if $val = \bot$ then $val := v$;
        $V[i].write(val, ts)$;
        $maxts := 0$;
        for $j = 1$ to $n$ do 
            $t := T[j].read()$;
            if $maxts < t$ then $maxts := t$;
        
        if $ts = maxts$ then
            return($val$);
        
        $ts := ts + 1$;
    }
}

Your tasks:

1. (1 point) Give the definitions (i.e., respective properties) of obstruction-free consensus, lock-free consensus, and wait-free consensus.

2. (1 point) If the algorithm is correct prove its correctness. Otherwise, provide an execution in which some property of obstruction-free consensus is violated.
Problem 4  (6 points)

Your tasks:

1. (2 points) Give the sequential specification of a snapshot object.

2. (3 points) Give an algorithm that implements an atomic lock-free snapshot object using (any number of) atomic MRMW registers.

3. (1 point) In your algorithm above, if you replace the base registers (atomic MRMW) by regular MRMW registers, does the algorithm still correctly implement an atomic lock-free snapshot object? Justify your answer.
Problem 5  (2 points)

Consider the atomic commit-adopt object, which has the following specification. Every process $p$ proposes an input value $v$ to such an object and obtains an output, which consists of a pair $(dec, val)$; $dec$ can be either commit or adopt. The following properties are satisfied:

- **Validity**: If a process obtains output $(commit, v)$ or $(adopt, v)$, then $v$ was proposed by some process.

- **Agreement**: If a process $p$ outputs $(commit, v)$ and a process $q$ outputs $(commit, v')$ or $(adopt, v')$, then $v = v'$.

- **Commitment**: If every process proposes the same value, then no process may output $(adopt, v)$ for any value $v$.

- **Termination**: Every correct process eventually obtains an output.

Consider the following implementation of an atomic commit-adopt object from atomic wait-free snapshot objects and atomic MRMW registers:

Using two shared snapshot objects: $S_1$ and $S_2$ of size $n$, initialized to $(\bot, \bot, ..., \bot)$;
Using two local array of registers: $a_i$ and $b_i$ of size $n$;

```c
propose_i(v) {
    S_1.update(i, v);
    a_i := S_1.snapshot();
    if every non-$\bot$ value in $a_i$ is $v$ then
        x := (true, v);
    else
        v := max(a_i);  // max(arr) returns the greatest non-$\bot$ element in array $arr$
        x := (false, v);

    S_2.update(i, x);
    b_i := S_2.snapshot();
    if every non-$\bot$ value in $b_i$ is equal to $(true, v)$ then
        return $(commit, v)$;
    if some value in $b_i$ is equal to $(true, val)$ for some val then
        return $(adopt, val)$;
    return $(adopt, v)$;
}
```

Your task:
Is the above implementation correct (does it satisfy the commit-adopt properties)? Justify your answer.
Problem 6  (2 points)

An atomic 0-set-once object is a shared object that has three states ⊥, 0, and 1. ⊥ is the initial state. It provides only one operation set(v), where \( v \in \{0, 1\} \), such that:

- If the object is in state ⊥, then set(v) changes the state of the object to v and returns v.
- If the object is in state s, where s ∈ \{0, 1\}, then set(v) changes the state of the object to s ∧ ¬v and returns the new state of the object (i.e. s ∧ ¬v).

Your tasks:

1. (1 point) Explain what it means for a shared object to have infinite consensus number.
2. (1 point) Prove that the atomic 0-set-once object has infinite consensus number.