Distributed Algorithms, Final Exam

January 16, 2015

Name:

Sciper number:

Time Limit: 3 hours

Instructions:

- This exam is closed book: no notes or cheat sheets are allowed.
- Write your name on *each* page of the exam.
- If you need additional paper, please ask one of the TAs.
- Read through each problem before starting solving.
- Partial credit will be awarded, so explain your thinking carefully.
- State clearly any additional assumptions that you use which are not stated in the question.

Good Luck!

Points:

Part 1 (17)	Part 2 (17)	Part 3 (8)	Part 4 (13)	**Total (55)**

1 Reliable Broadcast (17 points)

Question 1. (4 points)

Mark each of the following properties

with `S`, if it is a safety property,
with `L`, if it is a liveness property,
or with `X`, if it is neither a safety property nor a liveness property.

1. If a correct process p broadcasts a message m, then p eventually delivers m.
2. If a correct process p broadcasts a message m, then every process eventually delivers m.
3. Every correct process broadcasts a message at least every 10 seconds.
4. Let p and q be two processes that both deliver messages m_1 and m_2. If p delivers m_1 before m_2, then q delivers m_1 before m_2.

Question 2. (4 points)

Mark with `T` all that applies.

1. It is possible to implement Reliable Broadcast with only Best Effort Broadcast and
   (a) Perfect Failure Detectors
   (b) Eventually Perfect Failure Detectors
   (c) A majority of correct processes
2. It is possible to implement Uniform Reliable Broadcast with only Best Effort Broadcast and
   (a) Perfect Failure Detectors
   (b) Eventually Perfect Failure Detectors
   (c) A majority of correct processes

Question 3. (2 points)

Give the complete specification of Causal Broadcast (as presented in the class).

Question 4. (3 points)

1. Construct an execution that satisfies Total Order Broadcast, but not Causal Broadcast.
2. Construct an execution that satisfies Causal Broadcast, but not Total Order Broadcast.
3. Construct an execution that satisfies Reliable Broadcast, but neither FIFO Order (Property below) nor Total Order.

Property 1 FIFO Order

1: **FIFO Order**: If some process broadcasts message m_1 before it broadcasts message m_2 , then no process delivers m_2 unless it has already delivered m_1.

Question 5. (4 points)

Prove that any execution of Total Order Broadcast which satisfies FIFO Order (Property above) also satisfies Causal Order.

2 Consensus (17 points)

Question 1. (3 points)

Give an implementation of the following three abstractions:

1. Consensus, with the *Validity* property left out,
2. Consensus, with the *Agreement* property left out,
3. Consensus, with the *Termination* property left out.

Question 2. (2 points)

Give the complete specification of Group Membership (as presented in the class).

Question 3. (2 points)

The implementation of Group Membership presented in class is given in Algorithm 2. Would the algorithm still implement Group Membership, if it used a non-uniform version of Consensus instead of Uniform Consensus? If yes, explain why. If not, which properties of Group Membership could be violated?

Algorithm 2 Group Membership implementation

```
Implements
    GroupMembership (gmp)

Uses
    PerfectFailureDetector (P)
    UniformConsensus (Ucons)

upon event <Init> do
    view := (0, S);
    correct := S;
    wait := false;

upon event <crash, $p_i$> do
    correct := correct \ {$p_i$};

upon event (correct <view.memb) and (wait = false) do
    wait := true;
    trigger <Ucons.propose, (view.id, correct)>;

upon event <Ucons.decide, (id, memb)> do
    view := (id, memb);
    wait := false;
    trigger <gmp.membView, view)>;
```

Question 4. (5 points)

Consider a new abstraction called Fair Multi-Consensus (Module 3) that provides fairness across several instances of a variant of consensus. The abstraction is defined as follows:

Module 3 Fair Multi-Consensus

```
1: Module:
2:     Name: FairMultiConsensus, instance fmc.

3: Events:
4:     Request: $\langle fmc, Propose \mid i, v \rangle$: Propose value $v$ for instance $i$.
5:     Indication: $\langle fmc, Decide \mid i, v \rangle$: Decide value $v$ for instance $i$.

6: Properties:
7:     Validity: Any value decided in instance $i$ is a value proposed in instance $i$.
8:     Uniform Agreement: No two processes decide differently in the same instance.
9:     Integrity: No process decides more than once in each instance.

10:    Weak Termination: If all correct processes propose for instance $i$, then every correct process eventually decides
    for instance $i$.

11:    Fairness: We say that process $p$ wins instance $i$ if the value proposed by $p$ is decided in instance $i$ at some
    process. Let $w_p(i)$ be the number of instances between 0 and $i$ won by $p$. If any process decided for all instances
    between 0 and $i$, then the following holds for all correct processes $p, p'$: $|w_p(i) - w_{p'}(i)| \leq 1$.
```

It makes sure that all correct processes are granted an equal "amount of decisions". The numbers of won instances for two correct processes must not differ by more than one. For example, consider a system of two correct processes p_1 and p_2, where p_1 proposes x in the first three instances and p_2 proposes y in the first three instances. If x is decided in the first instance, then y must be decided in the second instance. (If x was decided again, the fairness property would be violated.) In the third instance, x or y may be decided.

Your task is to give an algorithm that implements a Perfect Failure Detector using only Fair Multi-Consensus in a system of n processes.

Question 5. (5 points)

Consider an asynchronous system of only two processes. In such a system, is it possible to implement Uniform Consensus using only Perfect Point-to-Point Links and Non-Blocking Atomic Commit? Explain why.

3 Shared memory (8 points)

Question 1 (3 points)

Consider a system with two processes p_1 and p_2. Give a register execution such that each process performs exactly two operations and the execution is

1. not safe
2. not regular but safe
3. not atomic but regular

Question 2 (5 points)

Give an algorithm that implements a $1-1$ atomic register using any number of $1-N$ regular registers (without any failure detectors).

4 Self-stabilizing Mutual Exclusion (13 points)

In this exercice, we study another solution of the self-stabilizing mutual exclusion problem. We consider a ring of N processes denoted by $P_0, \ldots, P_{N-1}$ (connected in this order). There are two distinguished processes: P_0, referred to as the *bottom* process, and P_{N-1}, referred to as the *top* process. The other processes are referred to as the *normal* processes. The state of each process P_i is an integer variable x_i taking values in $\{0, 1, 2\}$. *All arithmetic operations are done modulo* 3. Each process can instantaneously read the states of its neighbour processes. The following defines the rules of the algorithm:

- At the bottom process P_0: if $x_0 + 1 = x_1$ then $x_0 := x_0 + 2$.
- A normal process P_i $(0 < i < N - 1)$: if $x_{i-1} = x_i + 1$ or $x_i + 1 = x_{i+1}$ then $x_i := x_i + 1$.
- At the top process P_{N-1}: if $x_{N-2} = x_0$ and $x_{N-1} \neq x_0 + 1$ then $x_{N-1} := x_0 + 1$.

A process is said to *hold a token* when the corresponding precondition in the algorithm above holds. We say there is a *move* at process P if process P is activated while holding a token. The activation of a process not having a token has no effect.

We recall that the Mutual Exclusion problem consists in the following: *(i)* at each point of the execution, there is a unique token, *(ii)* every process gets the token infinitely often.

Question 1. (4 points)

We represent a configuration C of the system as the string $x_0 \ldots x_{N-1}$ of state values of the processes from bottom to top. Moreover, in that string, we place between neighbours whose states differ an arrow such that, in the direction of the arrow, the state decreases (modulo 3) by 1. For instance, the configuration $C = 0\ 0\ 0\ 1\ 0\ 0\ 2\ 1\ 1$ is decorated as $0\ 0\ 0 \leftarrow 1 \rightarrow 0\ 0 \rightarrow 2 \rightarrow 1\ 1$. Note that if there are no arrows between x_i and x_{i+1}, then $x_i = x_{i+1}$. We introduce

$$f(C) = \text{number of left-pointing arrows } + 2 \cdot \text{number of right-pointing arrows}$$
$$g(C) = \text{number of left-pointing arrows } + \text{number of right-pointing arrows}$$

We consider configurations C, C' such that some process has a token in C, and C' is obtained from C by activating such a process. The following table describes the effect of this activation on the possible arrows around this process. The moves (0), (1) and (3) are completely filled, as well as the last column of the move (4). Fill in the remaining gaps represented by "?" (no justifications required).

Move	Activated process	Before	After	$f(C') - f(C)$	$g(C') - g(C)$
(0)	Bottom P_0	$x_0 \leftarrow x_1$	$x_0 \rightarrow x_1$	+1	0
(1)	Normal P_i	$x_{i-1} \rightarrow x_i\ x_{i+1}$	$x_{i-1}\ x_i \rightarrow x_{i+1}$	0	0
(2)	Normal P_i	$x_{i-1}\ x_i \leftarrow x_{i+1}$	x_{i-1} ? x_i ? x_{i+1}	?	?
(3)	Normal P_i	$x_{i-1} \rightarrow x_i \leftarrow x_{i+1}$	$x_{i-1}\ x_i\ x_{i+1}$	-3	-2
(4)	Normal P_i	$x_{i-1} \rightarrow x_i \rightarrow x_{i+1}$	x_{i-1} ? x_i ? x_{i+1}	?	-1
(5)	Normal P_i	$x_{i-1} \leftarrow x_i \leftarrow x_{i+1}$	x_{i-1} ? x_i ? x_{i+1}	?	?
(6)	Top P_{N-1}	$x_{N-2} \rightarrow x_{N-1}$	x_{N-2} ? x_{N-1}	?	?
(7)	Top P_{N-1}	$x_{N-2}\ x_{N-1}$	x_{N-2} ? x_{N-1}	?	?

In the sequel we assume the following facts (you are *not* required to prove them).

Fact 1. *If there is a unique arrow in the system at some point of the execution, then this arrow remains unique and travels back and forth from the bottom P_0 to the top P_{N-1}. Moreover, there is a unique token.*

Fact 2. *In any execution starting from an arbitrary configuration, there is at least one move of the bottom process P_0 between two moves of the top process P_{N-1} (recall that a move is the activation of a process holding a token).*

Question 2. (4 points)

Consider an execution with infinitely many moves starting. Assume that the bottom process never performs a move in the whole execution. We will derive a contradiction.

- (a) Show that the top process performs at most one move. Thus, eventually, only normal processes make moves.
- (b) Considering the quantity g, show that there is a finite number of moves of type (3), (4), (5).
- (c) Show that there is only a finite number of moves of type (1) and (2). Derive the contradiction.

Question 3. (5 points)

We now prove that, in an execution with infinitely many moves, eventually there is a unique arrow. We have just seen that the bottom process P_0 makes infinitely often a move. Consider a fragment E of execution starting with a move of the bottom process P_0 and ending with the last move before the next move of P_0. We say that the event $A =$ "the leftmost arrow exists and points to the right" is falsified during a move from configuration C to C' if A holds in C but not in C'.

- (d) Show that the event A is falsified at least once during the fragment E.
- (e) Note that this falsification necessarily occurs during moves of type (3), (4) or (6). Show that, if this falsification occurs in a move of type (6), then there is a unique arrow by the end of E.
- (f) Using Fact 2, show that, if during E the falsification of A occurs only in moves of type (3) and (4), then the quantity f is decreased by at least 1 during E. Conclude that the algorithm is self-stabilizing.