Distributed Algorithms, Final Exam Solution

January 2016

Part I
Broadcast (12 points)

Answer the following questions in the provided spaces only.

Question 1 (5x1 points)

Which of the following properties are safety properties and which are liveness? Mark an "S" or "L" next to each property **clearly** with a *one-line* explanation.

1. For any message m_1 delivered by a process p_1, if there exists another process p_2 that delivered m_1 and then delivered a message m_2, then p_1 also delivers m_2. ______
   Explanation:

2. For every pair of processes p_1 and p_2 that deliver a message m_1, if p_1 delivers a message m_2 before delivering m_1, then p_2 also delivers m_2 before delivering m_1. ______
   Explanation:

3. Every process that crashes is eventually detected. ______
   Explanation:

4. No process is detected before it crashes. ______
   Explanation:

5. No two processes deliver the same message. ______
   Explanation:

Question 2 (3x1 points)

Give a brief explanation of the properties for each of the following abstractions.
Note: The notation for each abstraction will be used in Questions 3 and 4.

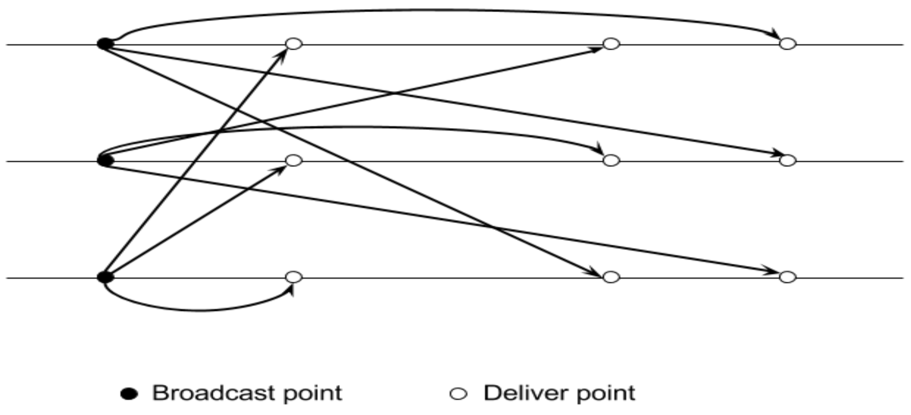
1. Causal broadcast (Notation: C)
   Answer:

2. Total order broadcast (Notation: T)
   Answer:

3. Uniform reliable broadcast (Notation: U)
   Answer:

Question 3 (2 points)

Consider the broadcast executions shown below. Which properties does it satisfy? Use the notations from Question 2. (Choose only a single option, no explanation required.)

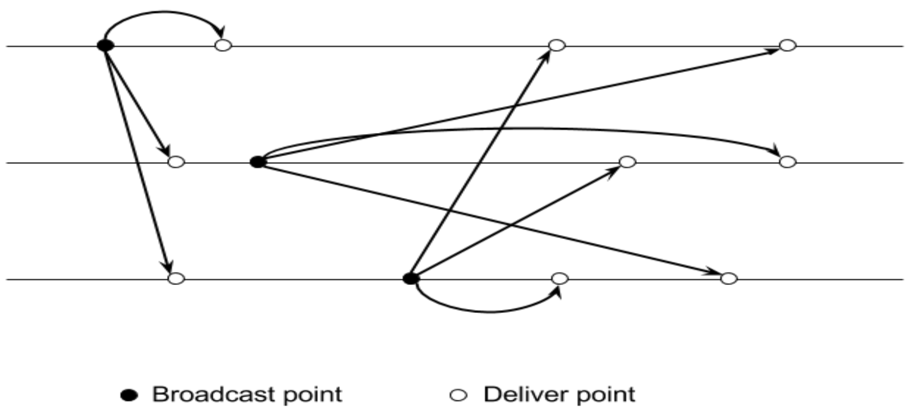


1. C
2. T
3. U
4. C & T
5. T & U
6. C & U
7. C & T & U
8. None of the above.

*Answer:*______

Question 4 (2 points)

Consider the broadcast executions shown below. Which properties does it satisfy? (Choose only a single option, no explanation required.)



1. C
2. T
3. U
4. C & T
5. T & U
6. C & U
7. C & T & U
8. None of the above.

*Answer:*______

Part II

Consensus, NBAC, TRB, GM (15 points)

Question 1 (5x2 points)

Indicate which sentence is true and which is false with a T and F about three different consensus algorithms, respectively. Explain your answer.

Algorithm I (nonuniform consensus with P): The processes exchange and update proposals in rounds and decide on the value of the non-suspected process with the smallest id.

Algorithm II (uniform consensus with P): The processes exchange and update proposal in rounds, and decide after n rounds.

Algorithm III (uniform consensus with majority): The processes alternate in the role of a coordinator until one of them succeeds in imposing a decision.

1. Algorithm I can satisfy uniform consensus if we use Uniform Reliable Broadcast instead of Best Effort Broadcast.
2. In algorithm II, the decided value is proposed by the correct process with lowest ID.
3. Using a $\diamond P$ in algorithm II does not violate Agreement property but Termination.
4. In algorithm III, if p_i is leader and decides, the decided value is actually proposed by p_i.
5. Algorithm III finishes in at least N rounds because a process processes its value in a round only if it is the leader in that round.

Question 2 (4x0.5 point)

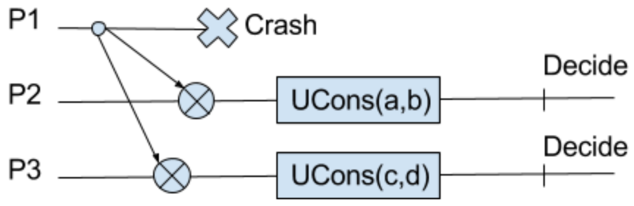
Explain the difference between each of the following pairs of abstractions in terms of their respective properties.

1. Consensus and Non Blocking Atomic Commit (NBAC).
2. Reliable Broadcast and Terminating Reliable Broadcast.
3. Group Membership and perfect failure detector (P).
4. View Synchrony and uniform view Synchrony.

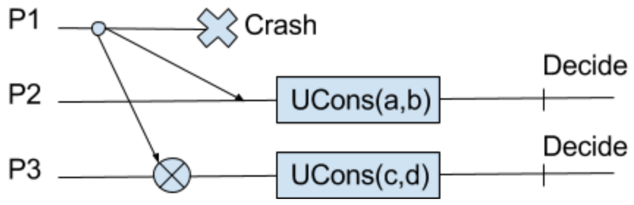
Question 3 (3x1 point)

Each of the below executions represents a NBAC abstraction implemented using Uniform Consensus. All the processes propose 1 in the beginning. Write all the possible values (0, 1, or 0/1) for **a, b, c** and **d** in each execution. (⊗ shows the sender is detected to have crashed and the message is lost.)

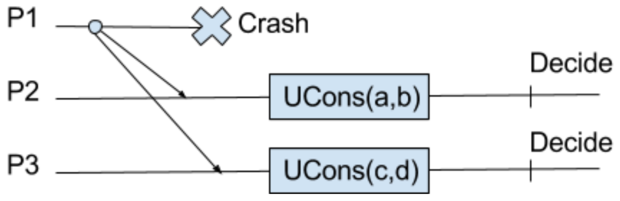
1.



2.



3.



Part III

Process Failures and Shared Memory (13 points)

Question 1. Failures and process majorities.

(2x3 points)

Consider the fail-silent system model, i.e., where processes may fail by crashing, and the failure detection *cannot* be detected, so no failure detector can be used.

1. Does any implementation of a regular register require a majority of correct processes in this case? Explain your answer.
2. Now suppose an eventually perfect failure detector $\diamond P$ can be used – is a correct majority still required? Explain your answer.

Question 2. Single-choice statements.

(5x0.5 points)

Mark each of the following statements with either `T`, if the statement is true, or with `F`, if that statement is false. No explanation is required.

1. Consider any distributed algorithm running in a system of n total processes. The algorithm does not make use of any failure detector, thus this system must comprise at least $n = 2 \cdot f + 1$ processes in order to tolerate a predefined subset of f processes crashing.
2. The Uniform Reliable Broadcast abstraction ensures that, despite a process crashing while it broadcasts a message m, the correct processes still eventually deliver m.
3. A regular register execution where no crashes occur is equivalent with an atomic register execution.
4. Consider a Reliable Broadcast algorithm. If all sending processes are assumed to be correct, then this algorithm actually satisfies the Uniform variant of Agreement.
5. If sending processes in a Causal Broadcast algorithm are assumed to be correct, then this algorithm actually satisfies the Uniform Causal Broadcast specification.

Question 3. A regular register array.

(4.5 points)

Traditional registers, as defined in the class, are meant to store one single object (e.g., a value). A *register array*, in contrast, is able to store multiple objects.

We consider a regular register array algorithm $\mathcal{A}$ which relies on m underlying regular registers, $[R_0, ..., R_{m-1}]$. Each underlying register stores an object. We assume that all objects have the form of $\langle key_i/value_i \rangle$ tuples, with $i \in \{0, .., m-1\}$. Thus, each object is uniquely associated with one of the underlying registers, such that object with key key_i is associated with register R_i.

Additionally, we require that $\mathcal{A}$ runs on N processes, $p_0, .., p_{N-1}$, where $N = m \cdot 2$. To ensure reliability, this algorithm guarantees that every underlying register is replicated at two processes, as follows. Processes are grouped in pairs, $\{p_0, p_1\}, \{p_2, p_3\}$, ..., $\{p_{N-2}, p_{N-1}\}$; and each process pair replicates one of the registers, such that the first pair of processes handles (i.e., replicates) register R_0, the second pair handles register R_1, and so on.

For handling any `read` or `write` operation on a key key_i, a process either triggers that operation on its underlying register (if key_i is associated to its underlying register), or it forwards the operation to one of the two processes handling the register associated with key_i.

The specification of a regular register array is as follows:
Name: Regular register array.
Properties: Same as properties *Termination* and *Validity* of regular registers.
Requests:
`read(`k_i`)` $\rightarrow v_i$,
`write(`k_i, v_i`)`, where k_i is they key of the object, and v_i is the value.

Give an implementation of $\mathcal{A}$, assuming that the m underlying regular registers are already provided. Algorithm 1 provides a skeleton for $\mathcal{A}$.

Algorithm 1 A regular register array.

```
1: Implements:
2:     Regular register array ($\mathcal{A}$)
3: Uses:
4:     RegularRegister ($rr_i$)
5: procedure $\langle read \mid k_i \rangle$ at process $p_x$ do
6:     ???

7: procedure $\langle write \mid k_i, v \rangle$ at process $p_x$ do
8:     ??
```

Part IV

Population Protocols and Self-Stabilization (15 points)

Throughout this part, we will use the following fairness assumption:

Definition 1 (Fairness)**.** *An infinite execution E is fair iff, for every configuration C occurring infinitely often in E, for every possible transition $C \to C'$, C' also occurs infinitely often in E.*

Unless stated otherwise, all infinite executions in this part are assumed to be fair.

Question 1 (3 points)

Consider a population of n agents. Every two agents can meet. We define $\mathcal{Q} = \{\circ, \bullet\}$, and the following protocols:

$$\mathcal{A} : \begin{cases} \bullet, \bullet & \to \bullet, \circ \\ \bullet, \circ & \to \circ, \bullet \\ \circ, \bullet & \to \bullet, \circ \\ \circ, \circ & \to \circ, \circ \end{cases} \qquad \mathcal{B} : \begin{cases} \bullet, \bullet & \to \bullet, \bullet \\ \bullet, \circ & \to \bullet, \bullet \\ \circ, \bullet & \to \bullet, \bullet \\ \circ, \circ & \to \circ, \circ \end{cases}$$

$$\mathcal{C} : \begin{cases} \bullet, \bullet & \to \circ, \circ \\ \bullet, \circ & \to \circ, \bullet \\ \circ, \bullet & \to \bullet, \circ \\ \circ, \circ & \to \circ, \circ \end{cases} \qquad \mathcal{D} : \begin{cases} \bullet, \bullet & \to \circ, \circ \\ \bullet, \circ & \to \bullet, \bullet \\ \circ, \bullet & \to \bullet, \circ \\ \circ, \circ & \to \bullet, \circ \end{cases}$$

For each predicate below, indicate the (possibly empty) list of protocols which satisfy the predicate.

1. If all the agents start with the initial state $\circ$, then all the agents eventually have permanently the same state $\circ$.
2. If all the agents start with the initial state $\bullet$, then all the agents eventually have permanently the same state $\circ$.
3. Eventually, there is always exactly one agent in state $\bullet$.
4. If all agents start with the initial state $\bullet$, and if n is odd, then eventually, there is always exactly one agent in state $\bullet$.
5. The number of agents in state $\bullet$ does not increase along the execution.
6. For any $0 \le k \le n$, in any suffix of the execution, there is eventually at least k agents in state $\bullet$.

Question 2 (3 points)

Consider a population of n agents. The possibilities of interactions between the agents are summed up as a graph G with n vertices. Each edge $e = (x, y)$ represents a possible interaction where x is the initiator, and y is the responder. It is assumed that (x, y) is an edge iff (y, x) is an edge (the graph is bidirectional).

Consider the protocol with state space $\{\circ, \bullet\}$:

$$\mathcal{U} \ : \ \begin{cases} \bullet, \bullet & \to \bullet, \circ \\ \bullet, \circ & \to \bullet, \circ \\ \circ, \bullet & \to \circ, \bullet \\ \circ, \circ & \to \circ, \circ \end{cases}$$

It is assumed that all the agents start in the same initial state $\bullet$.

1. Show that, eventually, the state of every agent p remains constant, equal to, say, $\gamma(p)$.
2. Show that the set I of agents p such that $\gamma(p) = \bullet$ forms an independent set in G, i.e, no two agents in I are neighbours in G.
3. Show that there exists a graph G, and a fair execution of $\mathcal{U}$ on G such that the corresponding set I is not a *maximum independent set*, i.e., there exists an independent set K in G of size strictly greater than the size of I.

Question 3 (4 points)

The leader election problem in the context of population protocols is specified as follows. The state of each agent comprises an output flag which indicates whether the agent is currently a leader (flag set to $\bullet$), or a non-leader (flag set to $\circ$). The specification of leader election (LE) requires that, in any (fair) execution, eventually there is a unique agent which permanently outputs $\bullet$, whereas the other agents permanently output $\circ$.

We study LE over the family $\mathcal{K}$ of complete (bidirectional) graphs, i.e., the graphs in which any two agents can interact.

1. Consider the protocol $\mathcal{U}$ from Question 2. Show that, for any graph $G \in \mathcal{K}$, any fair execution in which the agents all start in the same initial state $\bullet$, solves LE.

We now proceed to show that no single population protocol can solve LE over the whole graph family $\mathcal{K}$ in a self-stabilizing way. Recall that a self-stabilizing protocol is required to solve the problem at hand whatever the initial configuration is. We prove the claim by contradiction. Assume there exists a protocol $\mathcal{V}$ such that, for any graph $G \in \mathcal{K}$, any fair execution (starting from an arbitrary configuration), solves LE.

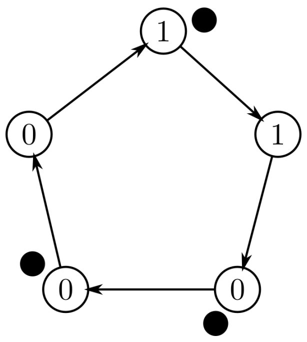


Figure 1: Oriented ring - Agents with • hold a token.

2. Consider a (sufficiently large) graph $G \in \mathcal{K}$, and any fair execution E of $\mathcal{V}$ on G. Show that there exists a configuration C occurring infinitely often in E, and a strict subgraph H of G such that all agents from H output $\circ$ in C, and $H \in \mathcal{K}$.
3. Explain why there exists a possible sequence $C \to C_1 \to \cdots \to C'$ of transitions between configurations on G such that some agent from H outputs $\bullet$ in C'.
4. Derive a contradiction.

Question 4 (5 points)

Consider an oriented ring $R = \{0, 1, \ldots, n-1\}$ as in Figure 1 with n an *odd integer*. The orientation means that an agent x is an initiator in some interaction iff the responder is agent $x + 1 \mod n$.

We define a protocol $\mathcal{W}$, with state space $\mathbb{Z}/2 = \{0, 1\}$ (the integers modulo 2), by the following rules:

$$\mathcal{W} \ : \ \forall a, b \in \mathbb{Z}/2, \ a, b \to a, a+1$$

Let C be a configuration. We say that agent x *holds a token* in configuration C if agent x and agent $x+1$ (modulo n) have the same state in C, i.e., $C(x+1) = C(x)$.

1. Let C_0 be an arbitrary initial configuration. Show that, in any fair execution starting from C_0, eventually, there is always exactly one token present in the system.
2. Show moreover that every agent holds the token infinitely often.

In other words, we have just shown that $\mathcal{W}$ is a self-stabilizing solution to the problem of token circulation.

1. Adapt the protocol to the case where the size n of the ring is only assumed to not be a multiple of 3. Generalize.

Appendix

For the reader's convenience, we recall here some basic formal definitions about population protocols. A graph is a couple $G = (G_V, G_E)$ where G_V is a set of vertices (representing the agents of the population), and $G_E \subseteq G_V \times G_V$ is a set of edges (representing the possible interactions between the agents). The graph is bidirectional when: (x, y) is an edge iff (y, x) is an edge. The edge (x, y) represents the interaction in which x plays the role of the initiator, and y the role of the responder.

A protocol is a tuple $\mathcal{A} = (\mathcal{Q}, \delta)$ where $\mathcal{Q}$ is a finite set (the state space), and $\delta : \mathcal{Q} \times \mathcal{Q} \to \mathcal{Q} \times \mathcal{Q}$ is the transition function. The transition function is to be thought as a finite set of rules $p, q \to p', q'$. Such a transition means that, if the initiator (resp. responder) is in state p (resp. q), then after the interaction, the state of the initiator (resp. responder) becomes p' (resp. q').

A configuration C of $\mathcal{A}$ on G is map that assigns a state $C(x)$ to each agent x in G. Given an edge (x, y) in G, and two configurations C, C', we write $C \xrightarrow{x,y} C'$ when, for all agent $z \notin \{x, y\}$, $C(z) = C'(z)$, and $C(x), C(y) \to C'(x), C'(y)$ is a rule of the protocol. Intuitively, this relation means that C' is obtained from C by activating the interaction (x, y). We write $C \to C'$ if $C \xrightarrow{x,y} C'$ for some edge (x, y) in $\mathcal{G}$.

An execution is a (finite or infinite) sequence $C_1 \to C_2 \to \dots$ of transitions between configurations.