Question 1:

This question relates to quorum replication (of R/W registers).

a) The implementers of a certain key-value datastore decided to skip line 4 of the client read protocol that we have seen in class, claiming that in the workloads they envision, the atomic register semantics would be preserved even without it, so it is not worth the extra latency. Explain what properties should such a workload exhibit in order for their claim to be correct? Explain your answer.

As we have seen in class, an example scenario that shows why without Line 4 the register would not be atomic involves a write operation \( W(x,v) \) and two read operations \( R1(x) \) and \( R2(x) \) such that \( W \) overlaps with both \( R1 \) and \( R2 \) but \( R2 \) starts after \( R1 \) ends.

Reminder about the example we have seen in class: Suppose \( W \) updates a single server \( S1 \) and now \( R1 \) receives \( r \)-replies from a READ quorum that includes \( S1 \). Hence, \( R1 \) will return \( v \) and terminate. Since it does not wait for \( w \)-replies here, it is possible that \( R2 \) finishes its Line 2 before any additional servers are updated with \( v \). In this case, \( R2 \) will only find previously written values and return the one with highest timestamp, but it will be a previous value to \( v \).

So, obviously the required workload should avoid having such combinations of write and 2 (or more) reads. We need to be convinced that this is the only required limitation on the workload. Clearly by the protocol when there is no overlap between any read to any write the atomic semantic is preserved – writes can be ordered according to their timestamp and a read would always return a value of a previous write. If only a single read may overlap any write operation (or all such reads are overlapping), then it is clear that the read would return either the previous value or the value of a concurrent write.

b) Suppose the timestamp in the client write protocol is obtained from an atomic clock (plus the process id to break symmetry) – in this case line “A” becomes setting the timestamp to the output of the atomic clock and lines 1&2 (in the following slide) are not needed. Is line 4 still needed in the client read protocol to ensure atomic register semantics? Explain your answer.

Line 4 is still required. The counter example we have seen in class works with a single write and is unrelated to the accuracy of its timestamp.
**Question 2:**

The following variant of Mostefaoui & Raynal has been proposed:

Code for i

1. **init:** \( r_i \rightarrow 0, \) est\( i \rightarrow v_i \)
2. **while** didn’t decide **do**
   3. \( c \rightarrow \Omega; \) est\( c \rightarrow \text{nil}; \) \( r_i \rightarrow r_i + 1 \)
   4. **if** \( i = c \) **then**
      5. est\( c \rightarrow \text{ets} i \)
   6. **else**
      7. **wait until** \(<\text{EST}, r_i, v>\) is received from \( c \) or \( c \notin \Omega \)
      8. **if** \(<\text{EST}, r_i, v>\) is received from \( c \) **then**
         9. est\( c \rightarrow v \)
      10. **end if**
     11. **end if**
   12. broadcast\( <\text{EST}, r_i, \text{est} c>\)
   13. **wait until** \(<\text{EST}, r’i, \text{est}’c>\) messages was received from \( n - f \) nodes
   14. rec\( i \leftarrow \{\text{est}’c \mid <\text{EST}, r’i, \text{est}’c> \text{ was received}\}\)
   15. **if** rec\( i = \{v’\} \) **then**
      16. decide \( v’ \) and broadcast\( <\text{DECIDE}, v'>\)
   17. **else if** rec\( i = \{v’, \text{nil}\} \) **then**
      18. est\( i \rightarrow v’ \)
     19. **end if**
   20. **end while**
21. **Upon receiving** \(<\text{DECIDE}, v 0 >\) from \( q \) **do**
22. decide \( v 0 \) and broadcast\( <\text{DECIDE}, v 0 >\)

Does it solve the consensus problem? Explain your answer – in case your answer is positive, NO NEED for a formal proof.

**No,** the new algorithm does not solve consensus – there can be a violation of agreement. The reason is that during some unbounded initial period, \( \Omega \) might return different identifiers to different processes. Hence, it is possible that in the first round all processes but \( p \) think that \( q \) is the leader. Process \( q \) sends it values est\( q = v \) to all who reply \(<\text{EST}, 1, v>\). Process \( q \) receives \( n-f \) of these messages (including itself) and decides \( v \).

However, \( p \) that thinks it is the coordinator too sends its value est\( p = v 1 (\neq v) \) in a \(<\text{EST}, 1, v 1>\) message, and this message is included in the first \( n-f \) messages of all processes (except \( q \)). Hence, all processes find rec\( i = \{v, v 1\} \) and therefore do not decide and keep their value. In round 2 the output of \( \Omega \) for everyone is \( p \) and thus everyone decide \( v 1 \).
**Question 3:**

In this question we will prove the following lemma regarding Paxos.

**Lemma:** Suppose a quorum of acceptors have sent (ACCEPTED,r,ACK) messages for some value v in some round r. Then no leader with a round number r1>r would send an (ACCEPT,r1,v1) message with v1≠v.

**Proof:** Assume, by way of contradiction, that the theorem does not hold. **Complete the proof from here.**

Hint, think of a minimality argument regarding a leader that sends (ACCEPT,r1,v1) with a value v1≠v.

Assume by way of contradiction that the lemma does not hold, and let q be the leader with minimal r1 that sends an (ACCEPT,r1,v1) message with v1≠v. Hence, q receives at least one (PROMISE,r1,ACK,r',v') message such that r' was maximal among received PROMISE messages and v'=v1.

Due to the quorum intersection property, at least one of the PROMISE messages received by q was from a process that sent an (ACCEPTED,r,ACK) message to p, adopted v as its value in round r, and set last_good_round to r.

Due to the minimality of r1 and the code, all r' values in the (PROMISE,r1,ACK,r',v') messages received by q for which v1≠v must be smaller than r. A contradiction to the assumption that q chose v1≠v for its (ACCEPT,r1,v1) message.
Question 4:

a) Explain intuitively the following claim: “$S$ based consensus protocols require all participants to agree on the entire membership while $\Omega$ based consensus protocols can solve consensus with weaker knowledge guarantees about the membership”.

The reason is that $\Omega$ ensures that eventually there will be only one leader/coordinator who will be able to dictate its round proposal to all other nodes.
On the other hand, with $S$ the list of trusted/suspected nodes can be different forever, meaning that the nodes need another mechanism to ensure that (at least eventually) there is a single round coordinator/leader who can enforce its value on all correct nodes. Setting the round leader to be $(r \mod n)$ does the trick, but requires exact membership knowledge.

b) Can Paxos be used as is to solve the Non-Blocking Atomic Commit problem, meaning that each process invokes Paxos with “Abort” or “Commit” and decides on the output of Paxos? Explain your answer.

No. The safety requirements of NBAC are different, and in particular they require that if at least one node votes for abort, then only abort can be decided. In contrast, in consensus (and in particular Paxos) any proposed value can be decided, so if one process votes for abort and another for commit, the end result might be commit.

Submission instructions:

You should solve this exercise alone – submissions are individual. Solutions must be submitted through the course web site – either printed or a high-resolution scan of handwriting. Solutions must be written in Hebrew unless you get an authorization from Prof. Friedman to submit in English.
Try to be brief. If your answer is very lengthy, it could be a sign that it is wrong.

The submission date is Sunday 02/12/2018 before 12:00 noon.

Good luck!