The University of New South Wales

COMP3151/9154
Foundations of Concurrency

Final Exam

Term 2, 2021

Time Allowed: 24 Hours. Submit by 8AM Sydney time on the 25:th of August.

Total Number of Questions: 6

Total Number of Pages: 4

Total Marks Available: 100

Brief answers are **strongly** preferred. Extreme verbosity may cost marks.

Produce a typeset PDF file, via LaTeX or otherwise, with your answers.

Submit with `give cs3151 exam exam.pdf` or with the `give` web interface.

The exam is open-book. You may read anything you like, and in general use any passive resource.

You **may not** use active resources—don't solicit, offer, or accept help of any kind, with one exception: you may ask private questions on Ed. Johannes will monitor Ed regularly, except when he sleeps (around 10PM–6AM).

Include the following statement in your PDF file:
I declare that all of the work submitted for this exam is my own work, completed without assistance from anyone else.

You must adhere to the UNSW student conduct requirements listed at `https://student.unsw.edu.au/conduct`.

Part I

Question 1 (20 marks)

Explain, informally and in your own words, the difference between the following:

(a) (2 marks) Concurrency and sequential computation.

(b) (2 marks) A lock and a semaphore.

(c) (2 marks) A strong semaphore and a weak semaphore.

(d) (2 marks) Strong and weak fairness.

(e) (2 marks) Synchronous and asynchronous message passing.

(f) (2 marks) Shared-variable concurrent programming and distributed programming.

(g) (2 marks) Classical monitors à la Brinch Hansen & Hoare, and monitors as implemented in Java.

(h) (2 marks) A safety property and a liveness property.

(i) (2 marks) The meaning of `if` statements in Promela and in Java.

(j) (2 marks) The setting in which the FLP theorem holds, and the setting in which the Byzantine Generals algorithm is resilient to crash failures.

Question 2 (10 marks)

Progress is the assumption that a concurrent system does not terminate prematurely. In other words, it is always true that if at least one process has an enabled transition, then some transition will eventually be taken. (We've tacitly assumed this through most of the course).

Assuming only progress, and introducing no fairness assumptions,[1] show that no two-process mutual exclusion algorithm satisfies eventual entry.

Question 3 (20 marks)

Suppose $P = (L, T, s, t)$ is a synchronous transition diagram. Assume that all transitions in T are either input or output transitions; there are no internal transitions. Furthermore, assume that the transitions are unguarded, perform no state update, and that output transitions always send the contents of a single variable. Thus all transitions in T are of the following two forms:

$$\ell_i \xrightarrow{\top;C \Leftarrow x;I} \ell_j \qquad\qquad \ell_i \xrightarrow{\top;C \Rightarrow x;I} \ell_j$$

A *dual* of P, written $\mathcal{D}(P)$, is obtained by (a) replacing all input actions in P with output actions, and vice versa; and (b) renaming all local variables so that the variables of $\mathcal{D}(P)$ are disjoint from those of P.[2] For example, these transition systems are each others' duals:

Are the following claims true? For each claim, explain why it's true, or give a counterexample.

The closed product of P and $\mathcal{D}(P)$ is...

(a) (5 marks) ...convergent if the graph of P is acyclic.

(b) (5 marks) ...convergent for all P.

(c) (5 marks) ...deadlock-free if the graph of P is acyclic.

(d) (5 marks) ...deadlock-free if, for every location ℓ and channel C, ℓ has at most one outgoing transition involving C.

[1] For those who are familiar with the paper from the 2020 exam: justness counts as a fairness assumption for the purposes of this question.

[2] The exact naming scheme is unimportant, so long as it generates fresh names. The whole renaming business is just a technicality to make sure the closed product is well-formed.

Part II

These questions are about the paper *Time, Clocks, and the Ordering of Events in a Distributed System* by Leslie Lamport (Commun. ACM 21(7): 558-565, 1978).

Question 4 (15 marks)
For each of the following subquestions, write a paragraph of text to answer it, in your own words.

(a) (5 marks) What is the scientific contribution of the paper?

(b) (5 marks) What is an anomalous behaviour?

(c) (5 marks) Lamport made the following remark about the Chandy-Lamport algorithm for taking distributed snapshots:[3]

> "I consider the algorithm to be a straightforward application of the basic ideas from [27]"

...where [27] is, of course, *Time, Clocks and the Ordering of Events in a Distributed System*. What do you think he meant by this remark? How would you explain the connection?

Question 5 (25 marks)
In the acknowledgements, Lamport gives credit to Paul Johnson and Robert Thomas for the use of timestamps to order operations, and the concept of anomalous behaviour. They are the authors of RFC 677, which outlines a method for maintaining consistent replicas of a distributed database. You don't have to read the RFC! Instead, here is a summary of the relevant parts:

1. The system consists of n independent *database management processes* (DBMPs) with unique PIDs. They communicate with each other using asynchronous message passing over reliable channels with in-order delivery.
2. Each DBMP maintains a local copy of the database, as a mapping from *keys* $k \in \mathbf{K}$ to pairs $(v, (t, p))$ of *values* $v \in \mathbf{V}$ and *timestamps* $(t, p) \in (\mathbb{N} \times \mathbb{N})$. In the initial state, all the DBMPs have identical local copies.
3. Each DBMP with PID i maintains a logical clock $C_i \in \mathbb{N}$ that is incremented with every action performed by the DBMP.
4. At any point, a DBMP with PID i may initiate a *modification request* to update the database, requesting a new value v to be assigned to key k. When it does so, it sends a message $(k, v, (C_i, i))$ to every other DBMP.
5. When a process receives (or initiates) a modification request for key k, it compares the timestamp of the request with the timestamp of the pre-existing entry for k in its local database. If the timestamp of the modification is lexicographically larger, the database entry is updated with the new (value, timestamp); otherwise, the modification is discarded.

That's it! Now answer the following questions:

(a) (3 marks) This algorithm guarantees that if two modification requests have the same timestamp, the modification requests are equal. How?

(b) (2 marks) The system of local clocks here does *not* satisfy Lamport's clock condition. Which item from the above list corresponds to an implementation requirement {`IR1`, `IR2`} from the paper? Which implementation requirement is absent?

[3] https://lamport.azurewebsites.net/pubs/pubs.html#chandy

(c) (5 marks) Hence, this algorithm may result in causally inconsistent behaviour. Give an example execution where a modification request m_1 is discarded in favour of another modification m_2, even though m_2 happens before m_1. Note that *happens before* refers to the technical term introduced in Lamport's paper (written $m_1 \to m_2$).

Hint: the example can be very small.

(d) (5 marks) Propose a modification of the algorithm that makes it satisfy Lamport's clock condition. Explain why the example from the previous question is no longer possible with your modification.

(e) (10 marks) We have seen that the original algorithm is not *causally* consistent. It is still *eventually* consistent, in the following sense: if modifications cease and all outstanding messages have been received and acted upon, all DBMPs will have identical local copies of the database.

Define an invariant that can be used to prove eventual consistency. The invariant must:

- be true in the initial state,
- be preserved by the acts of sending and receiving modification requests, and
- imply eventual consistency once there are no more in-flight messages.

Prove these points. You don't need to use any particular formalism; feel free to state (and prove) your invariant in prose.

As usual in distributed algorithms, you may assume atomicity of local computation. In particular, you may assume that rule 5 is executed atomically.

Question 6 (10 marks)

The papers considers a mutual exclusion algorithm as a case study.

(a) (5 marks) The paper presents three requirements, numbered (I-III), on a correct mutual exclusion algorithm. Compare and contrast these requirements with the ones we have seen in the course (mutual exclusion, eventual entry, deadlock freedom, absence of unnecessary delay, bounded waiting, linear waiting). Which of Lamport's requirements correspond to which of our requirements? Are there any that don't correspond to our requirements?

(b) (5 marks) Why are the acknowledgement messages in rule 2 of the algorithm necessary? Show an example where one of Lamport's requirements is violated if the algorithm is modified to never send these acknowledgements.