The University of New South Wales

Final Exam

13s2

COMP3151/COMP9151

Foundations of Concurrency

Time allowed: **2 hour**
Total number of questions: **5**
Total number of marks: **50**

Textbooks, lecture notes, etc. are not permitted, except for 2 double-sided A4 sheets of hand-written notes.

Calculators may not be used.

Not all questions are worth equal marks.

Answer all questions.

Answers must be written in ink.

You can answer the questions in any order.

You may take this question paper out of the exam.

Write your answers into the answer booklet provided. Use a pencil or the back of the booklet for rough work. Your rough work will not be marked.

Shared-Variable Concurrency (20 Marks)

Question 1 (10 marks)

Algorithm 6 below is a close relative of Peterson's algorithm for the critical section problem. It solves the critical section problem.

Algorithm 6	
boolean b[0..1] ← {false, false} integer turn	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: b[0] ← true	q2: b[1] ← true
p3: turn ← 0	q3: turn ← 1
p4: await b[1] = false or turn = 1	q4: await b[0] = false or turn = 0
p5: critical section	q5: critical section
p6: b[0] ← false	q6: b[1] ← false

(a) Does Algorithm 6 still solve the critical section problem if we swap lines 2 and 3, that is, exchange statement p2 with statement p3 and exchange statement q2 with statement q3?

(b) Consider another variant of Algorithm 6 above where the statement await b[1] = false is added at the end of the code of process p. Does the modified algorithm satisfy deadlock freedom?

(c) Consider the same variant of Algorithm 6 as in part b. Does the modified algorithm satisfy eventual entry?

Justify each of your answers briefly.

Question 2 (10 marks)

The Savings Account Problem. A savings account is shared by several people (processes). Each person may deposit or withdraw funds from the account. The current balance in the account is the sum of all deposits to date minus the sum of all withdrawals to date. The balance must never become negative. A deposit never has to delay (except for mutual exclusion), but a withdrawal has to wait until there are sufficient funds. Withdrawals have to be serviced first-come-first-served.

Develop a monitor to solve this problem.[1] The monitor should have two procedures: deposit(amount) and withdraw(amount). Specify a monitor invariant. Assume the arguments to deposit and withdraw are positive. Use the *Signal-and-Continue* discipline. Explain how your solution implements each of the requirements.

[1] You may use pseudo monitor notation similar to the one used in the textbook, that is, you have at your disposal variables of type condition and functions waitC, signalC, and empty to access condition variables.

Message-Passing Concurrency (30 Marks)

Question 3 (8 marks)

Two kinds of processes, `A`'s and `B`'s, enter a room. An `B` process cannot leave until it meets two `A` processes and an `A` process cannot leave until it meets a `B` process. Each process leaves the room—without meeting any other processes—once it has met the required number of other processes.

(a) Develop a server process to implement this synchronisation. Show the interface of the processes modelling individual `A` and `B` processes.

(b) Modify your answer to a so that the first of the two `A` processes that meets a `B` process does not leave the room until after the `B` process meets a second `A` process.

Use Promela (with message passing primitives only) to express your solutions.

Question 4 (10 marks)

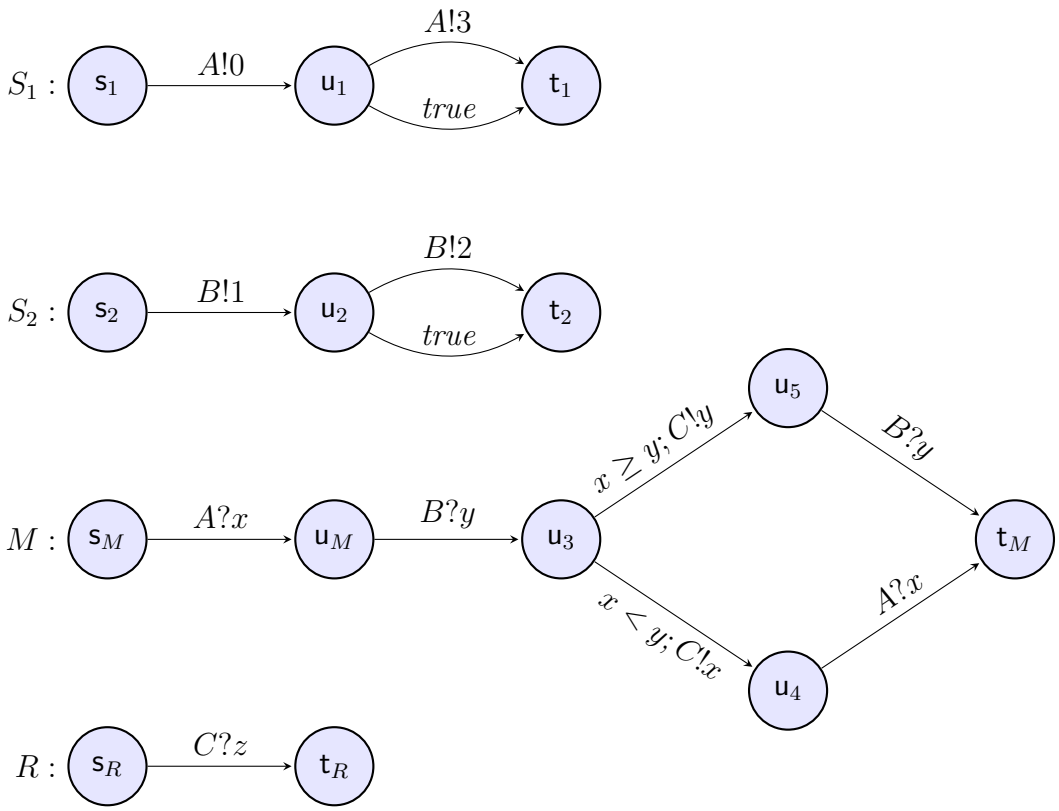

Figure 1: The synchronous transition diagram $S_1 \| S_2 \| M \| R$.

Consider the synchronous transition diagram in Fig. 1 and prove

$$\{true\}\, S_1 \| S_2 \| M \| R \,\{x = 3 \wedge y = 1 \wedge z = 0\}$$

with Levin & Gries or AFR for full marks or via semantics and Floyd for half marks.

Question 5 (12 marks)

System model: The system comprises of $n \in \mathbb{N}$ nodes named $1, \ldots, n$. Nodes communicate by passing messages over asynchronous, reliable channels. Every node can send messages to every other node. Nodes have unique IDs. Neither nodes nor channels crash.

Each process has a *clock* CL_i that ranges over the non-negative integers and starts at 0. A *timestamp* is a pair (cv, i) consisting of a clock value and a process ID. Timestamps are ordered lexicographically, that is, we use the process IDs to break ties to have a total order on process' clock values.

Nodes execute local actions, sends, and receives. These actions are augmented as follows to accommodate timestamping.

- Every local action increments the clock by 1.
- Every send of a message m of node i is replaced by (a) incrementing the clock, $\text{CL}_i{+}{+}$, followed by (b) sending $(m, (\text{CL}_i, i))$.
- Whenever a node i receives any message timestamped with (v, k), it updates its clock CL_i to $\max(\text{CL}_i, v) + 1$.

Each node i maintains an initially empty *request queue* rq_i consisting of timestamped messages, ordered lexicographically by their timestamps.

The following protocol is proposed to solve the critical section problem with the aid of timestamps. For simplicity, the individual actions such as **R1** can be assumed to be atomic.

Requesting access.

R1 When node i wants to enter its CS, it sends a REQUEST message to all other nodes, and places the request $(\text{REQUEST}, (\text{CL}_i, i))$ on its own request queue rq_i. (Recall that first the clock is incremented and that all the messages are timestamped.)

R2 When a node k receives a message REQUEST from i, it places node i's request on its request queue rq_k and sends a REPLY message to i.

Gaining access. Node i enters its CS when both

A1 i has received from every other node a message with timestamp lexicographically greater than the timestamp of its own REQUEST, and

A2 i's own request is at the head of rq_i.

Releasing the CS.

E1 Upon exiting the CS, node i removes its request from rq_i and sends a timestamped RELEASE message to all other nodes.

E2 When node k receives RELEASE from i, it removes i's request from its request queue rq_k.

In the context of the model and algorithm outlined above, answer the following questions. Justify your answers.

(a) Show that this algorithm does not guarantee mutual exclusion.

(b) Show that this algorithm guarantees mutual exclusion if channels are not only reliable but also FIFO.

(c) Assuming weak fairness, does the algorithm guarantee eventual entry if the channels are not only reliable but also FIFO?