The University of New South Wales

Final Exam

2005/11/15

COMP3151/COMP9151

Foundations of Concurrency

Time allowed: **3 hours (8:45–12:00)**
Total number of questions: **8**
Total number of marks: **45**

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 *not* take this question paper out of the exam.

Except for Question 3, write your answers into the answer booklet provided. Be concise — *excessively verbose answers will be penalised*. Use a pencil or the back of the booklet for rough work. Your rough work will not be marked.

Family Name:	
Other Names:	
Signature:	
Student Number:	

Shared-Variable Concurrency (15 Marks)

Question 1 (6 marks)

Give all possible final values of variable x in the following program. Prove your answer correct in Andrews' *PL*.

```
int x = 0;
sem s1 = 1, s2 = 0;
co P(s2); P(s1); x = x * 2; V(s1);
// P(s1); x = x * x; V(s1);
// P(s1); x = x + 3; V(s2); V(s1);
oc
```

Question 2 (9 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[1] to solve this problem. The monitor should have two procedures: deposit(amount) and withdraw(amount). First specify a monitor invariant. Assume the arguments to deposit and withdraw are positive. Use the *Signal-and-Continue* discipline. Explain your solution.

Message-Passing Concurrency (22 Marks)

Question 3 (4 marks)

Fill in the four gaps in the following program such that it becomes a filter process that merges two EOS-terminated and sorted input streams of integer messages into an EOS-terminated and sorted output stream of integer messages. (Write your answer to this question directly on this page, not into the answer booklet.)

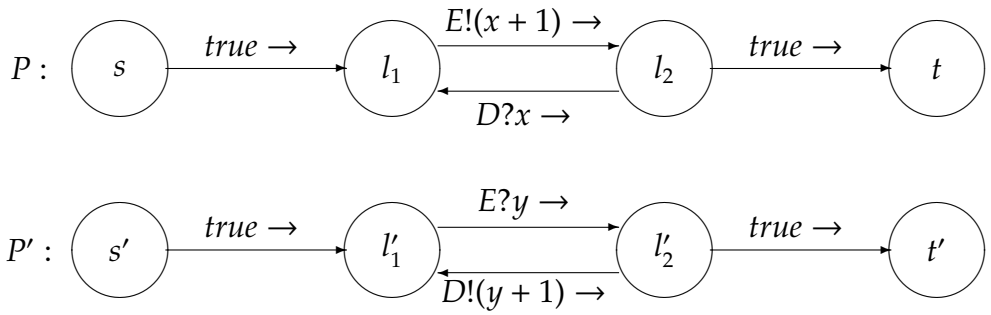
```
op in1(int), in2(int), out(int);
process Merge {
  int v1, v2;
  receive in1(v1); receive in2(v2);
  while (v1 != EOS and v2 != EOS) {
    if (v1 <= v2) {
                                                    # gap 1
    } else {
                                                    # gap 2
```

[1] You may use pseudo-MPD monitor notation similar to the one used in the textbook rather than the idiosyncratic `m2mpd` syntax.

```
    }
  }
  if (v1 == EOS) {
    while (v2 != EOS) {
                                                        # gap 3
    }
  } else {
      while (v1 != EOS) {
                                                        # gap 4
      }
  }
  send out(EOS);
}
```

Question 4 (4 marks)

Prove that $\{true\}\ P \parallel P'\ \{x + 1 = y\}$ holds for the synchronous transition diagram:



Question 5 (7 marks)

The following program sketch attempts to implement a (bounded) queue.

```
global Queue
  op deposit(int item), fetch(ref int item);
body
  int buf[0:n−1];
  int front = 0, rear = 0, count = 0;

  proc deposit(item) {
    if (count < n) {
      buf[rear] = item;
      rear = (rear+1) % n; count++;
    } else
      # take actions appropriate for overflow
  }

  proc fetch(item) {
    if (count > 0) {
      item = buf[front];
      front = (front+1) % n; count−−;
    } else
      # take actions appropriate for underflow
```

}
end Queue

a) *(3 marks)* Explain why it should not be shared by more than one process. Provide a counterexample to prove your point.

b) *(4 marks)* Rewrite the **global** such that it can be shared by an arbitrary number of processes. *Hint:* the **in** statement might be helpful.

Question 6 (7 marks)

Meeting scheduler. Consider three processes, each having a local array of integers already sorted in ascending order. At least one integer value is contained in all three arrays. Develop an MPD program or an asynchronous transition diagram for the three processes. The three processes exchange messages until each has determined the smallest common value. Provide key assertions in each process and give pre- and postconditions for the core of the whole program.

Restrictions: Messages contain only one integer value at a time. Processes should limit the number of messages they send in order to save bandwidth. For instance, it would be unreasonable to send messages containing array values beyond the earliest common meeting time. (This restriction is imposed to preclude solutions in which some processes simply send all their array content.)

Discuss termination and the role of fairness.

Automata and Logic (8 Marks)

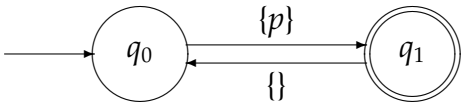
Appendix B contains a summary of LTL and Büchi automata.

Question 7 (4 marks)

Construct a Büchi automaton recognising exactly the models of the LTL formula $\Box(p \rightarrow \Diamond q)$.

Question 8 (4 marks)

Consider the Büchi automaton $\mathcal{A} = (\{q_0, q_1\}, \{\{\}, \{p\}\}, q_0, \{(q_0, \{p\}, q_1), (q_1, \{\}, q_0), q_1\})$ depicted below. Characterise the infinite words $\mathcal{A}$ recognises (a) informally and (b) with an LTL formula.



A Andrews' *PL* (a Proof System for MPD Annotations)

Assignment axiom

$$\left\{\phi[^{e}/_{x}]\right\} x \;=\; e \left\{\phi\right\} \qquad \textbf{ass}$$

Composition rule

$$\frac{\left\{\phi\right\} S_1 \left\{\psi\right\},\ \{\psi\}\, S_2\, \{\psi'\}}{\left\{\phi\right\} S_1 ; S_2 \left\{\psi'\right\}} \qquad \textbf{comp}$$

If-Else statement rule

$$\frac{\left\{\phi \wedge b\right\} S_1 \left\{\psi\right\},\ \left\{\phi \wedge \neg b\right\} S_2 \left\{\psi\right\}}{\left\{\phi\right\} \texttt{if (}b\texttt{) } S_1 \texttt{ else } S_2 \left\{\psi\right\}} \qquad \textbf{if}$$

While statement rule

$$\frac{\left\{\phi \wedge b\right\} S \left\{\phi\right\}}{\left\{\phi\right\} \texttt{while (}b\texttt{) } S \left\{\phi \wedge \neg b\right\}} \qquad \textbf{while}$$

Rule of consequence

$$\frac{\phi' \rightarrow \phi,\ \left\{\phi\right\} S \left\{\psi\right\},\ \psi \rightarrow \psi'}{\left\{\phi'\right\} S \left\{\psi'\right\}} \qquad \textbf{cons}$$

Await statement rule

$$\frac{\left\{\phi \wedge b\right\} S \left\{\psi\right\}}{\left\{\phi\right\} \texttt{<await (}b\texttt{)}S\texttt{>} \left\{\psi\right\}} \qquad \textbf{await}$$

Co statement rule

$$\frac{\left\{\phi_i\right\} S_i \left\{\psi_i\right\} \text{ hold and are interference free}}{\left\{\bigwedge_i \phi_i\right\} \texttt{co } S_1 \texttt{ //...// } S_n \texttt{ oc} \left\{\bigwedge_i \psi_i\right\}} \qquad \textbf{co}$$

Semaphore wait rule

$$\frac{\phi \wedge s > 0 \rightarrow \psi[^{s-1}/_{s}]}{\left\{\phi\right\} \texttt{P(}s\texttt{)} \left\{\psi\right\}} \qquad \textbf{P}$$

Semaphore signal rule

$$\frac{\phi \rightarrow \psi[^{s+1}/_{s}]}{\left\{\phi\right\} \texttt{V(}s\texttt{)} \left\{\psi\right\}} \qquad \textbf{V}$$

Simplifying assumption: arithmetic on bounded types such as `int` does not wrap around silently. Overflow and underflow errors lead to abnormal termination which renders program behaviours irrelevant to partial correctness arguments such as proofs in *PL*.

B LTL and Büchi Automata

Recall the syntax of **L**inear time **T**emporal **L**ogic:

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \vee \phi \mid \bigcirc\phi \mid \phi \,\mathcal{U}\, \phi$$

where p is proposition symbol drawn from some finite set $\mathcal{P}$.

We employ the usual abbreviations for "truth": $\mathit{true} = p \vee \neg p$, "falsehood": $\mathit{false} = \neg \mathit{true}$, "conjunction": $\phi \wedge \psi = \neg(\neg\phi \vee \neg\psi)$, "eventually": $\Diamond\phi = \mathit{true}\,\mathcal{U}\,\phi$, "henceforth": $\Box\phi = \neg\Diamond\neg\phi$.

LTL Semantics

LTL formulae are evaluated over infinite state sequences (or *behaviours*), where states are identified with sets of propositional symbols. Thus the state space is $\Sigma = 2^{\mathcal{P}}$ ($= \{ P \mid P \subseteq \mathcal{P} \}$).

The satisfaction relation $\models$ between behaviours $\sigma = (s_i)_{i\in\mathbb{N}}$ and LTL formulae is defined inductively by:

- $\sigma \models p$ (where $p \in \mathcal{P}$) iff $p \in s_0$.
- $\sigma \models \neg\psi$ iff $\sigma \not\models \psi$.
- $\sigma \models \psi \vee \psi'$ iff $\sigma \models \psi$ or $\sigma \models \psi'$.
- $\sigma \models \bigcirc\psi$ iff $(s_{i+1})_{i\in\mathbb{N}} \models \psi$.
- $\sigma \models \psi \,\mathcal{U}\, \psi'$ iff there exists $k \in \mathbb{N}$ such that $(s_{i+k})_{i\in\mathbb{N}} \models \psi'$, and, for all $j \in \mathbb{N}$ such that $0 \leq j < k$, we have that $(s_{i+j})_{i\in\mathbb{N}} \models \psi$.

Büchi automata

A *Büchi automaton* is an NFA $\mathcal{A} = (Q, A, q_0, \Delta, F)$ understood as an acceptor of behaviours, not finite words over its alphabet A. Formally, $\mathcal{A}$ *accepts* $(s_i)_{i\in\mathbb{N}}$ iff there exists an infinite sequence $(l_i)_{i\in\mathbb{N}}$ of automaton states such that

- $l_0 = q_0$,
- $(l_i, s_i, l_{i+1}) \in \Delta$, for all $i \in \mathbb{N}$, and
- some element of the final state set F occurs infinitely often in $(l_i)_{i\in\mathbb{N}}$.