The University of New South Wales

Final Exam

2006/11/13

COMP3151/COMP9151

Foundations of Concurrency

Time allowed: **3 hours (8:45–12:00)**
Total number of questions: **6**
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 (8 marks)

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

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

Question 2 (7 marks)

Hyman's Algorithm. The following algorithm was published in the Communications of the ACM in January 1966. Does it solve the critical section problem for two processes? Prove your answer.

```
1   # assume a type bit with the two values 0 and 1
2   bit turn, flag[0:1] = ([2] 0);
3   process HIA [i = 0 to 1] {
4     while (true) {
5       flag[i] = 1;
6       while (turn != i) {
7         <await (flag[1−i] == 0);>
8         turn = i;
9       }
10      # critical section
11      flag[i] = 0;
12    }
13  }
```

Message-Passing Concurrency (30 Marks)

Question 3 (7 marks)

Suppose a computer center has two printers, *A* and *B*, that are similar but not identical. Three kinds of client processes use the printers: those that must use *A*, those that must use *B*, and those that can use either one. Using the multiple primitives notation (i.e., **in** .. −> .. []..**ni**), fill in the gap in the following program such that it becomes a fair solution, assuming that clients eventually release printers.

(Write your answer to this question directly on this page.)

```
global PrintMoron
  type prType = enum (prA, prB);              # printer type
  type reqType = enum (reqA, reqB, reqD);  # print request type; D = don't care
```

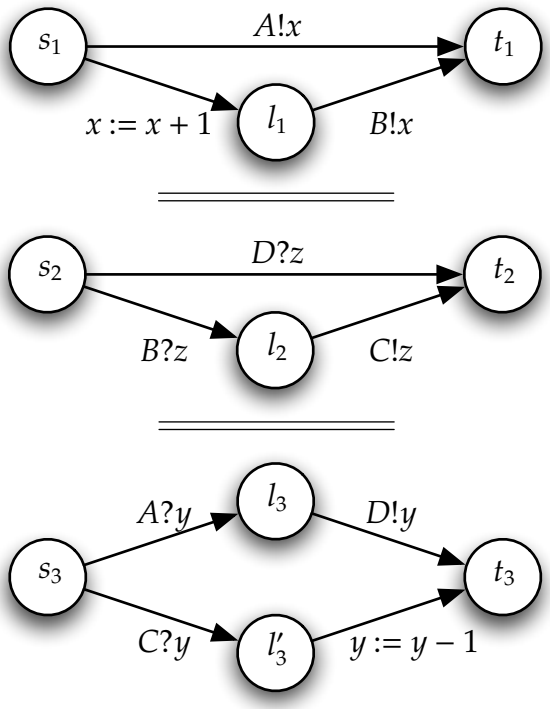
```
  op request (int pid, reqType r, ref prType p) {call };
  op release (int pid, prType p);
body PrintMoron
  bool Afree = true, Bfree = true;          # printer availability
  int Auser = −1, Buser = −1;               # who uses the printer; -1 = nobody

  process Granter {
    while (true) {
                                                          # gap
                                                          # gap
                                                          # gap
                                                          # gap
                                                          # gap
                                                          # gap
    }
  }
  process Releaser {
    while (true) {
                                                          # gap
                                                          # gap
                                                          # gap
                                                          # gap
                                                          # gap
                                                          # gap
  }
end PrintMoron

resource main ()
  import PrintMoron;
  int numRequesters; getarg (1, numRequesters);
  int numRounds; getarg (2, numRounds);
  reqType rqt[0:2] = (reqA, reqB, reqD);

  process Requester[i = 1 to numRequesters] {
    prType p;
    reqType r;
    for [j = 1 to numRounds] {
      nap (int (random () * 100));
      r = rqt[int (random (3))];
      write ("Process", i, "requesting print service", r);
      PrintMoron.request (i, r, p);
      write ("Process", i, "using printer", p, "after requesting", r);
      nap (int (random () * 100));
      PrintMoron.release (i, p);
      write ("Process", i, "releasing printer", p);
    }
  }
end main
```

Question 4 (7 marks)

Hamming's problem. Develop an MPD program whose output is the sequence of all multiples of 2, 3, and 5 in ascending order. The first elements of the sequence are 0, 2, 3, 4, 5, 6, 8, 9, 10, 12, 14. There will be four concurrent processes: one each to calculate the multiples of the numbers 2, 3, and 5, respectively, and a fourth process to merge the results.

Question 5 (7 marks)

Prove $\{true\}\, P\, \{x = z \wedge y \leq z\}$ for the synchronous transition diagram P depicted below.



Question 6 (9 marks)

Partitioning a set. Give two disjoint sets of integers S_0 and P_0, their union $S_0 \cup P_0$ has to be partitioned into two subsets S and P such that $|S| = |S_0|$ and $|P| = |P_0|$, and every element of S is smaller than any element of P.

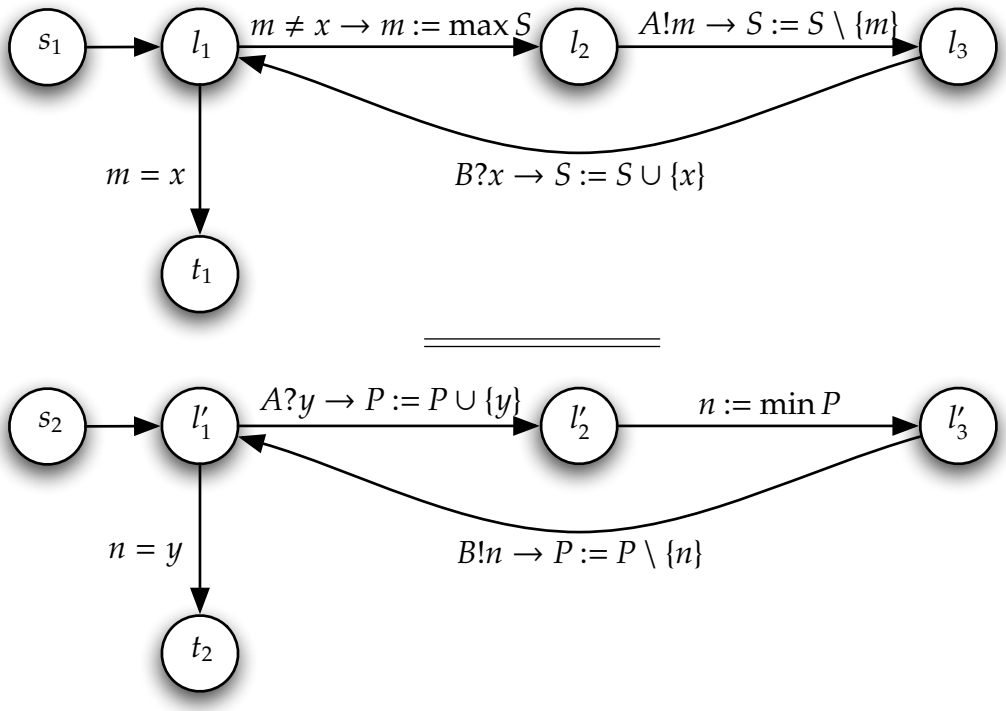
Prove the synchronous transition diagram given below correct with respect to precondition

$$S = S_0 \wedge P = P_0 \wedge S \neq \emptyset \wedge S \cap P = \emptyset \wedge x \neq m \wedge y \neq n$$

and postcondition

$$S \cup P = S_0 \cup P_0 \wedge S \cap P = \emptyset \wedge |S| = |S_0| \wedge |P| = |P_0| \wedge \max S < \min P$$

where, by convention, $\min \emptyset = +\infty$.



Discuss termination for 3 bonus marks.

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*.