The University of New South Wales

Final Exam

2011/10/28

COMP3151/COMP9151

Foundations of Concurrency

Time allowed: **2 hours (within 9:45–12:00)**
Total number of questions: **2**
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 that they would be of any help.)

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 (15 Marks)

Question 1 (15 marks)

Recall Lamport's fast algorithm, here given for two processes:

Algorithm: Fast algorithm for two processes	
integer gate1 ← 0, gate2 ← 0 boolean wantp ← false, wantq ← false	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: gate1 ← p	q3: gate1 ← q
p4: if gate2 ≠ 0	q4: if gate2 ≠ 0
p5: wantp ← false	q5: wantq ← false
p6: await gate2 = 0	q6: await gate2 = 0
p7: goto p2	q7: goto q2
p8: gate2 ← p	q8: gate2 ← q
p9: if gate1 ≠ p	q9: if gate1 ≠ q
p10: wantp ← false	q10: wantq ← false
p11: await wantq = false	q11: await wantp = false
p12: if gate2 ≠ p	q12: if gate2 ≠ q
p13: await gate2 = 0	q13: await gate2 = 0
p14: goto p2	q14: goto q2
p15: critical section	q15: critical section
p16: gate2 ← 0	q16: gate2 ← 0
p17: wantp ← false	q17: wantq ← false

All but the first of the following questions suggest modifications of the algorithm. These modifications do *not* accumulate.

(a) Does the algorithm guarantee eventual entry?

(b) Does the algorithm still satisfy mutual exclusion if we swap the last two lines?

(c) Is the algorithm still deadlock-free if we remove the await statements in line 6?

(d) Is the algorithm still correct if we remove the await statements in line 13?

(e) Would the algorithm still be correct if line 16 were replaced by the following?

p16: if gate2 = p gate2 ← 0	q16: if gate2 = q gate2 ← 0

For positive answers, sketch a brief informal proof; for negative answers, provide a counter example behaviour.

Message-Passing Concurrency (30 Marks)

Question 2 (30 marks)

A *barrier* is a coordination mechanism that forces processes which participate in a concurrent algorithm to wait until each one of them has reached a certain point in its program. The collection of these coordination points is called the barrier. Once all processes have reached the barrier, they are all permitted to continue past the barrier.

In this question we investigate certain aspects of barriers (at first) built from the synchronous message-passing primitives (i.e., blocking sends and receives).

(a) Build a barrier for two fully connected processes.

(b) Formally specify what it means for such a 2-process barrier to work. This should include safety and liveness properties. Prove that your solution works.

(c) Build a barrier for n processes that are connected only by a unidirectional ring of channels.

(d) Repeat part (b) for your n-process barrier. (A proof sketch suffices.)

(e) Adapt your solutions to parts (a) and (c) to asynchronous message passing.

(f) Can you make your solutions to part (e) resilient to lossy channels? (You may assume that channels are lossy but fair in that no message can be sent infinitely often without ever being received. You may also use concurrency inside processes.)

Disclaimer: neither seniors nor cryptographers were harmed in the preparation of this exam.