# COMP9154

**Foundations of Concurrency**

*I declare that all of the work submitted for this exam is my own work, completed without assistance from anyone else.*

# Part I

## Question 1 (10 marks)

Answer the following, briefly and in your own words.

1. (2 marks) What is a linear-time property?

The linear time property specifies the subset of behaviours that should only occur for the program.

1. (2 marks) Why does Owicki-Gries require interference freedom checks?

Because even the correctness of each program running in parallel is proved, the state of one program may also be falsified by others if they are not interference free.

1. (2 marks) How do distributed consensus algorithms get around the FLP theorem?

They try to devise a message delivering system with synchronous or partially synchronous protocol achieved by for example: time-out mechanism, message communicating between nodes.

1. (2 marks) Why does the King algorithm require one more round than the number of traitors?

Because this ensures that there is at least one king that is not traitor. With the non-traitor king’s message, the generals can reach to an agreement.

1. (2 marks) What is the difference between a permission-based and a token-based distributed mutual exclusion algorithm?

Permission-based algorithm is implemented based on the communication of one node over all other nodes, once it gets the permission of others, it can proceed to the critical section. However, token-based algorithm only requires a node to receive one special permission, in other word, the token to enter the critical section.

## Question 2 (15 marks)

1. (5 marks) Of the four main critical section desiderata, which ones are satisfied by this algorithm? For any properties that are not satisfied, describe a behaviour that is in violation.

**Eventual entry** is violated in this algorithm.

One of the cases is that these programs repeat the bolded sequence forever thus q can never enter the critical section:

In this case, p executes the cs and set x false firstly, and then q passed for waiting x to be false, and then p finishes the non-cs rapidly and set x true before q proceeds to .

1. (5 marks) Suppose we rewrite process p to be exactly like process q, but with the roles of x and y swapped. Would this change your answer to the previous question?

**Eventual entry** and **mutual exclusion** are not satisfied.

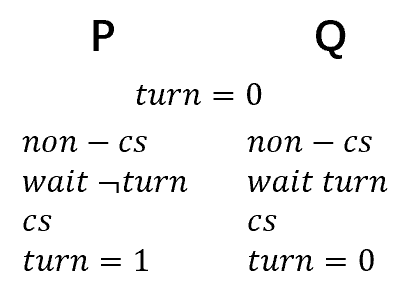
Assume that this time, process also consists of 8 lines, from to .

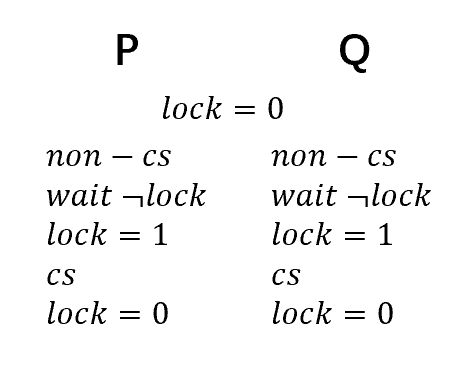
For eventual entry, the behaviour is similar to what we have in question a), rapidly pick up the permission to enter cs before q moves on indefinitely.

For mutual exclusion, there is a case that p and q run interleaved line by line in this pattern:

When they both arrive line 4 and set and false, and then move to line 3. Both of them will be permitted to jump to line 7, which is the critical section.

1. (5 marks) Why are there no algorithms with only a single bit of shared state? You don’t have to produce a formal proof, but try to make a convincing informal argument. For the purposes of this question, we consider an algorithm correct if it satisfies at least mutual exclusion and deadlock freedom.

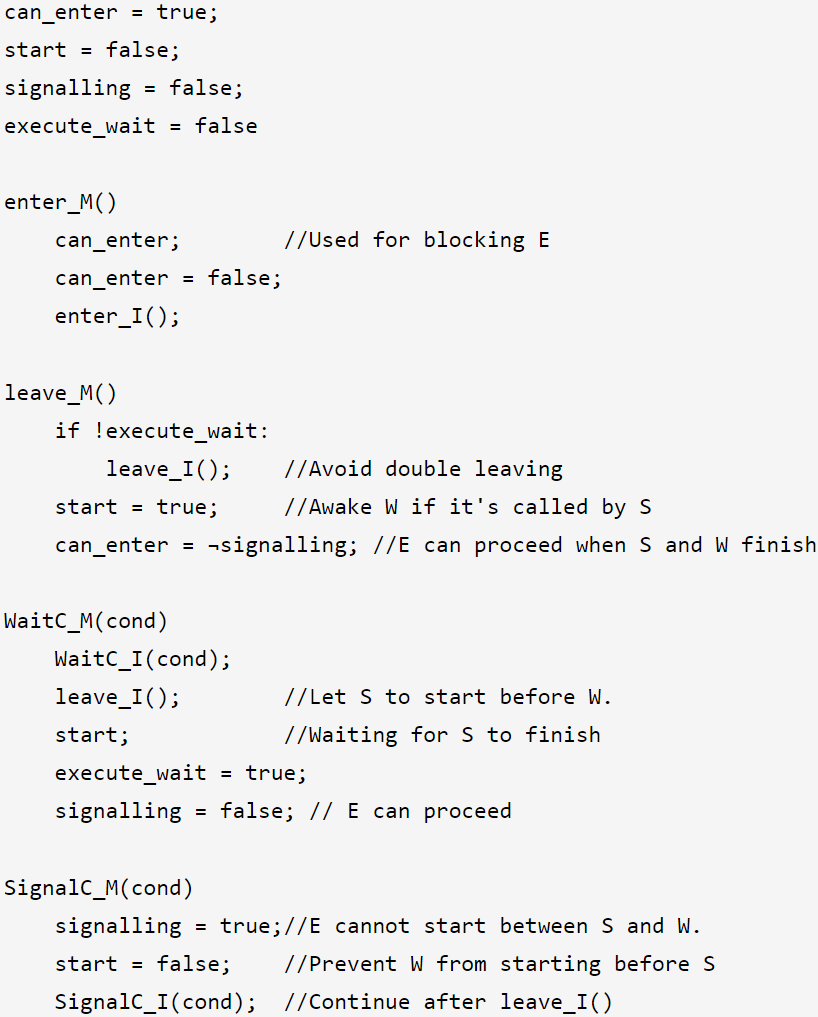
There are two potential algorithms for a single bit of shared state. One is to make the bit as a “lock” and the other is to treat the bit as “turn”.

With lock, and can pick up the lock together if they execute interleaved line by line. As a result, they will enter critical section at the same time and violate the mutual exclusion requirement. With turn, it is possible that one of them stuck in non-critical section and another one is trying to enter the critical section. For example, is waiting for turn to be 0 however is trapped in the non-cs forever then a dead lock occurs.

## Question 3 (15 marks)

Assume you have an underlying monitor implementation I with priority ordering E < S < W. Suppose we would like a monitor M to behave as if it had priority ordering E < W < S. Show how to implement our desired monitor M using our underlying implementation I.

The pseudo code for M is:



## Question 4 (10 marks)

1. (5 marks) Consider the LTL formula . Give a simpler but logically equivalent formula. Explain why it’s equivalent.

The equivalent formula is:

can be descripted as:

For all , there exists a so that happens after . （1）

can be descripted as:

There exists a , so that for all ,

there exists a so that happens after behaviour. （2）

It is straight forward that . For the other side, we can prove it by induction:

Let’s imagine a series of states:

1. Assume that in the base case that there exists , so that for all , there exists a so that happens.
2. Then we insert new states to the start of the series. Assume that when we have inserted nodes, there exists , so that for all , there exists a so that happens.
3. If we insert one more state to the beginning, then for all there is always an so that happens.
4. Starting from , we can deduce that for all , there is always an so that happens. Any series can be built by inserting elements to the beginning from the base case.

Thus, we have proved that by induction.

As a result, we have

1. (5 marks) Let the CCS process P be defined as follows: Simplify the following CCS expression step by step:

# Part II

## Question 5

1. (5 marks) Section 1 of the paper includes an extensive summary of the state of the art in 1996. How did Michael and Scott’s paper improve the state of the art?
2. (3 marks) What is the difference between a block-free algorithm and a wait-free algorithm? Of the two, which kind of algorithm did the specification of Assignment 1 ask for?

A wait-free algorithm is both block-free and starvation free. So, a wait-free algorithm can guarantee each process can enter critical section within finite steps which block-free only specifies that it is possible to enter if we put the time horizon infinitely long.

1. (2 marks) The performance analysis in Section 4 found no situation where Michael and Scott’s two-lock algorithm outperforms lock-free algorithms. What’s the benefit of the two-lock algorithm?
2. (5 marks) The compare-and-swap (CAS) on line E17 of the non-blocking algorithm has the accompanying comment “Try to swing Tail to the inserted node”. This suggests that, if the CAS fails, an enqueue operation can terminate even if its work is unfinished. Is this a problem? Explain why or why not.

## Question 6

1. (4 marks) What is the ABA problem? Explain informally and in your own words.
2. (3 marks) How does the authors’ use of modification counters help mitigate the ABA problem?
3. (5 marks) Ominously, the safety analysis in Section 3.1 is predicated upon the assumption that the ABA problem will never occur. Describe a scenario where, in the lock-free algorithm, the queue can end up corrupted if the ABA problem occurs.

Assume that the maximum value for an unsigned integer is , for a dequeue process, the recorded is . Before it executes the CAS operation in , there are already more than enqueues and dequeues successfully performed and the ptr recorded in Q->Head is the same as the recorded in variable that the process holds. This is possible because has been freed yet and the system allocates this piece of memory to the new node again. And this new created node with memory space is stored in Q->Head.ptr. In addition, the counter of Q->Head becomes for all that is a natural number. So, the processor would regard the value stored in &Q->Head and equal.

However, the next.ptr that the process holds has already been freed by other process long time ago. The resulting new Q->Head is not the actual Q->Head->next.ptr rather the old next.ptr more than steps ago. As a result, the queue will not be properly linked anymore.

## Question 7