Possible flow of any tutorial:

i) Discussion of any topics in past lectures which students found hard
ii) Discussion of some exercises on the recent lectures. These exercises are given on-the-spot and discussed. In other words, they are *not* for grading the students – they are for online discussion.
iii) Any questions the students have about assignments (not applicable for Tut 1)

Content:

The general idea of concurrency, and Promela – a modeling language for concurrent systems (the first two lectures in cs3211 – check Lesson Plan and Workbin in IVLE please).

Sample Exercises:

1. Consider the following Promela specification of a simple producer and consumer. Assume that both of them access a buffer which serves as a shared data structure.

```
toktype = {P, C};
toktype turn = P;

active proctype producer()                   active proctype consumer()
{                                            {
  do                                           do
  :: (turn == P) -> /*produce 1 item*/         :: (turn == C) -> /* consume 1 item */
                    turn = C                                      turn = P
  od                                           od
}                                            }
```

If there is only one producer process and one consumer process, will an interleaved execution violate mutual exclusion of access to the shared buffer ? To answer this question, construct the global state transition system by hand after assigning labels to the control locations of the processes.

Answer: To answer this question, we can simply try to understand the "states" of the program and then study the possible interleavings. The global "state" of a program is the valuation of the different program counters and variables. Now, what is a program counter? A program counter captures the progress in control flow in any thread. If there are n threads in a concurrent program, there will be n program counters. So, in the above concurrent program there will be two program counters. Now, what are the variables? The threads do not have any local variables. There is a single global variable called turn. So, a single state in the above concurrent program is a valuation of

(program counter of producer, program counter of consumer, turn)

Now that we have fixed what all constitutes a state, we need to know the possible values of these. The possible values of turn are P or C (depending on whether it is the producer's turn or consumer's turn). What are the possible values of the two program counters?

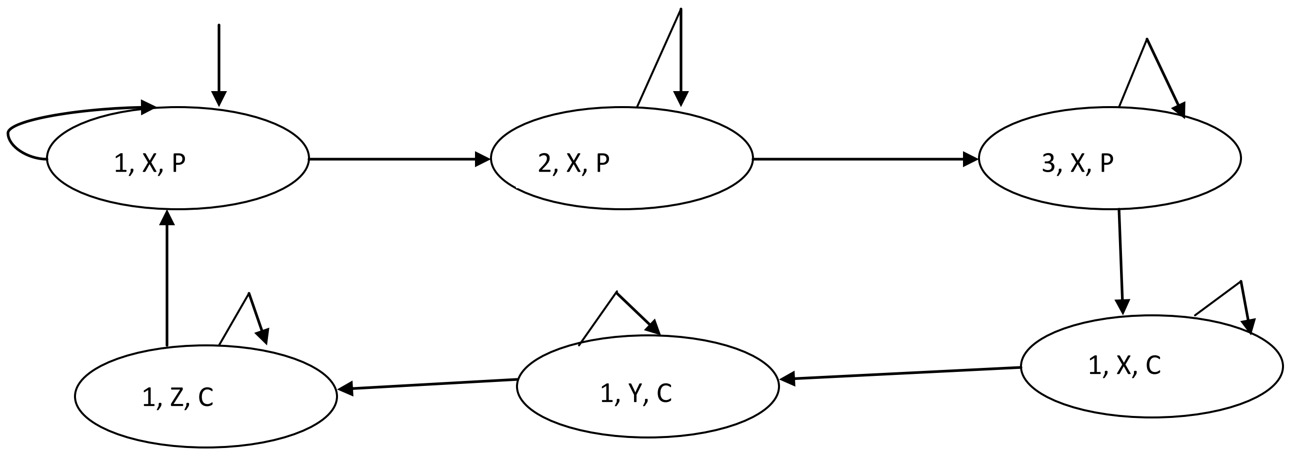
```
active proctype producer()
  do
  :: 1 (turn == P) ->
                 2 /*produce 1 item*/
                 3  turn = C
  od
}
```

The possible program counter values for the producer are shown in bold red. So, the program counter being equal to 1 means that the control flow is before (turn == P), that is the check (turn == P) has not yet been executed. Similarly, the program counter of the consumer can also take on three values.

```
active proctype consumer()
  do
  :: X (turn == C) ->
                 Y /*consumer 1 item*/
                 Z  turn = P
  od
}
```

I just call them {X, Y, Z} to avoid any confusion with the program counter values of the producer.

Now, we can understanding all possible interleavings of the concurrent program as follows.



This graph could be understood as follows. Initially, the program counter of producer thread is at 1, the program counter of consumer thread is at X, and the value of turn is P. This accounts for the initial state of the program is

(1, X, P)

Now from the initial state, either the producer could be scheduled to execute or the consumer could be scheduled to execute. If the consumer is scheduled, the program state remains at (1, X, P) - if the producer is scheduled it moves to (2, X,P).

```
   2. Suppose we instantiate the producer and consumer modules to
concrete processes in SPIN as follows.

init{
  atomic{ run producer(); run producer(); run consumer(); run consumer(); }
}

Can any two of the above SPIN processes be allowed to
access the shared buffer at the same time – that is, violation of
mutual exclusion? Explain your answer.
```

Answer: This is, in fact, possible since two producers might access the shared buffer at the same time. The turn only distinguishes between producers and consumers.

3. Consider an asynchronous composition of two processes described in the Promela modeling language.

```
byte x;
proctype myproc(bool i) {
      do
      :: x = x + x;
      od
}
init{ atomic{ run myproc(0); run myproc(1) } }
```

Use the semantics of Promela to find out the values taken by x during the program execution. Initially x is equal to 1.

Answer:

Every execution of the loop duplicates the value of x. So, x will assume the values of the form

2^n for $n \geq 0$

4. Suppose the infinite loop in previous question is compiled into the following assembly code. The processes are running asynchronously, and each time a process is scheduled, only its next instruction is executed atomically. Initially x = 1, and we use a hypothetical machine with infi nite memory. Note that x is a shared global variable and reg_{iA} , reg_{iB} are local registers in processes A and B respectively. Which of the following values can x reach in the execution of the above program -- 2,3,5? Explain your answer in each case. If the value is possible, show a detailed interleaving producing the value. If the value is not possible, argue why it is not possible.

```
            Process A                     Process B
LoopA:   regA^1 = x             LoopB:   regB^1 = x
         regA^2 = x                      regB^2 = x
         regA^3 = regA^1 + regA^2        regB^3 = regB^1 + regB^2
         x = regA^3                      x = regB^3
         go to LoopA                     go to LoopB
```

Answer: Getting the value 2

```
regA1 = x (gets 1)
regA2 = x (gets 1)
regA3 = regA1 + regA2 (gets 2)
x = regA3  (gets 2)
                // the other thread has not even been scheduled at this point
```

Getting the value 3

```
                                    regB1 = x   (gets 1)
regA1 = x (gets 1)
regA2 = x (gets 1)
regA3 = regA1 + regA2 (gets 2)
x = regA3  (gets 2)
                                     regB2 = x   (gets 2)
                                     regB3 = regB1 + regB2   (gets 3)
                                     x = regB3   (gets 3)
```

getting the value 5

```
                                     regB1 = x   (gets 1)
regA1 = x (gets 1)
regA2 = x (gets 1)
regA3 = regA1 + regA2 (gets 2)
x = regA3  (gets 2)

regA1 = x (gets 2)
                                     regB2 = x   (gets 2)
                                     regB3 = regB1 + regB2   (gets 3)
                                     x = regB3   (gets 3)
regA2 = x (gets 3)
regA3 = regA1 + regA2 (gets 5)
x = regA3  (gets 5)
```