Homework (Week 2)

Submission: Due on Friday, 17th of June, 11am Sydney Time. Please submit using the CSE Give System either online or via this command on a CSE terminal:

```
give cs3151 hw2 hw2.pdf hw2.pml
```

Late submissions are accepted up to five days after the deadline, but at a penalty: 5% off your total mark per day.

No skip (4 marks)

Consider the following trivial Promela model:

```
byte x = 0;

active proctype P() {
  x = 1;
  x = 2;
  x = 3;
}

ltl skip_2 { <>(x == 1 until x == 3) }
```

Here we would be surprised if the value of `x` becomes `3` immediately after being `1`. Yet Spin verifies that the property `skip_2` holds.

Explain why this model nonetheless satisfies `skip_2`. Also, give an LTL formula that better captures the intent stated above.

Submit your answer in a PDF file called `hw2.pdf`. Use of LaTeX is encouraged but not required. Please make your answers as **concise as possible**

Hint: the issue is to do with the semantics of LTL; it's not about any implementation detail of Spin. You can work out the answer without ever running Spin at all.

Dekker's Algorithm (8 marks)

Dekker's algorithm (aka attempt 5 from the lectures) was presented with two processes. But does the algorithm work for three or more processes?

1. Write a Promela model for Dekker's algorithm with three processes. Use the same general idea for the pre- and post-protocol with `wantX` variables and a `turn` counter (you'll need three `wantX` :s and three distinct `turn` values). Make sure your Promela model obeys the limited critical reference restriction.
2. Use Spin to check if your generalisation of Dekker's algorithm satisfies mutual exclusion and eventual entry. Include the LTL properties you used in your Promela file, along with `/* comments */` explaining what verification options you used and what the result was.

Submit your answer in a Promela file called `hw2.pml`

Hint 1: Feel free to assume that this Promela header file is present in the same directory as your .pml file, if you want to use the same critical section boilerplate as in the lectures.

Hint 2: The limited critical reference restriction also applies to the guards of `do` and `if` statements. In particular, the following does *not* obey limited critical reference, because `wantp` and `wantq` cannot be checked atomically.

```
do
:: wantp -> ...
:: wantq -> ...
:: else  -> ...
od
```