Concurrence, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
April 27th 2015

Answer the questions concisely and precisely
Answer each problem in a separate page (remember to put your name)
Closed-book exam
Duration: 2 hour

Exercise 1 FSP (3 Points)
Draw the `LTS` for the `MICROWAVE` process below.

```
MICROWAVE = (put_food_in -> SETTINGS),
SETTINGS = (set_heat_level -> set_time -> COOK |set_time -> set_heat_level -> COOK),
COOK = (cook -> take_food_out -> MICROWAVE).
```

Model again the `MICROWAVE` process from above, this time using parallel composition.

Hint: You will need to use handshaking with shared actions, so that it is not possible to produce silly action traces. eg to cook after take food out.

We ask to complete the following three processes

```
COOK = ( put_food_in -> .... ->  take_food_out ->COOK).
SET_HEAT = ( put_food_in -> ... -> cook -> SET_HEAT).
SET_TIME = ...
```

such that

```
||MICROWAVE = ( COOK||SET_HEAT||SET_TIME).
```

Exercise 2 Safety (2 Points)
A lift has a maximum capacity of `N` people. In a model of the lift control system, passengers entering the lift are signalled by an `enter` action and passengers leaving the lift are signalled by an `exit` action. Specify a safety property in `FSP`, which when composed with the lift model, will check that the system never allows the lift that it controls to have more than `N` occupants.

Complete the following code:

```
const N = ...
property LIFTCAPACITY = LIFT[0],
LIFT[...] = (enter -> LIFT[i+1]
                |when(i>0) ....
                |...).
```

Draw the `LTS` for the `LIFTCAPACITY` when `N` is `2`.

Exercise 3 JAVA (3 Points)
Desing a class `Buffer` class to deal with a Producers/Consumers system. There should be a stack which is filled by Producers and emptied by consumers. Use the class `Stack` from JAVA, with MAX elements. The producers can fill elements while the stack is not full, whilst the consumers can take elements whilst it is not empty.

```
import java.util.Stack;
...

public class Buffer {
   private Stack<Integer> buffer = new Stack<Integer>();
     ...
}
```

Exercise 4 Erlang (2 Points)
Given the following module `kim_pmap` starting as:

```
-module(kim_pmap).
-compile(export_all).
%%--sequential version-------
map(F, L) -> [F(X) || X <- L].
...
```

Desing a call to this function on the list `[1,2,3]` such that the result is `[2,3,4]`

Now we ask to extend this module adding a parallel version `pmp` of the preceding `map` function. Complete the following code:

```
%%--pmap threaded version. Hands on unvalid data-------
pmap(F, L) ->
   Parent = self(),
   Pids = [spawn(fun() ->...) || ...],
          [receive {Pid,Res} -> ... end || ...<- Pids].
```