Concurrence, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
April 22th 2013

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 (3 Points)
As we are in a period of economical crisis we will consider the bridge problem in the case of a small number of cars. Here there are 2 blue cars (`const NB = 2`) and 1 red car (`const NR = 1`). We make the "minimum" number of changes to the program given in the course to adapt to this new situation.

```
const NB = 2 // number of blue cars
const NR = 1 // number of red cars
range IDB= 1..NB // blue car identities
range IDR= 1..NR // red car identities (one by default)

range TB= 0..NB // blue car count
range TR= 0..NR // red car count

CAR = (enter->exit->CAR).

BNOPASS1   = C[1],
C[i:IDB]   = ([i].enter -> C[i%NB+1]).

BNOPASS2   = C[1],
C[i:IDB]   = ([i].exit -> C[i%NB+1]).

RNOPASS1   = C[1],
C[i:IDR]   = ([i].enter -> C[i%NR+1]).

RNOPASS2   = C[1],
C[i:IDR]   = ([i].exit -> C[i%NR+1]).

||BCONVOY= ([IDB]:CAR || BNOPASS1 || BNOPASS2).

||RCONVOY= ([IDR]:CAR || RNOPASS1 || RNOPASS2).

BRIDGE = BRIDGE[0][0],  //initially empty
BRIDGE[nr:TR][nb:TB] =    //nr is the red count, nb the blue count
   (when (nb==0) red[IDR].enter  -> BRIDGE[nr+1][nb]
    |red[IDR].exit    -> BRIDGE[nr-1][nb]
    |when (nr==0) blue[IDB].enter -> BRIDGE[nr][nb+1]
    |blue[IDB].exit   -> BRIDGE[nr][nb-1]
   ).

||CARS = (red:RCONVOY || blue:BCONVOY).

||SingleLaneBridge = (CARS || BRIDGE).
```

Answer to the following questions:

- Give the LTS corresponding to `BCONVOY` (6 states). Explain how this labelled transition system is build through shared actions (just give a small number of cases). Give an intuitive explanation of the result.
- Give the LTS corresponding to `SingleLaneBridge` (8 states). Give an intuitive explanation of the result.

Exercise 2 (2 points). Given the `SingleLaneBridge` given in Exercise 1 consider the stressed system and the following two progress property:

```
||CongestedBridge = SingleLaneBridge >> {red[IDR].exit,blue[IDB].exit}.

progress BLUECROSS = {blue[IDB].enter}
progress REDCROSS =  {red[IDR].enter}
```

Answer to the following questions:

- Give the LTS corresponding to `CongestedBridge`.
- Does `CongestedBridge` verify the `BLUECROSS` property? Justify the answer. In the case of `Progress violation` give a trace to the terminal set of states and the terminal set of states.
- Does `CongestedBridge` verify the `REDCROSS` property? Justify the answer. In the case of `Progress violation` give a trace to the terminal set of states and the terminal set of states.

Exercise 3 (2 Points). JAVA
In the chapter 5 of the *Concurrency* book you find the following (slightly modified) design of a semaphore:

```
const Max = 1
range Int = 0..Max

SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]    = (up->SEMA[v+1]
                 |when(v>0) down->SEMA[v-1]
                 ).

LOOP = (mutex.down -> critical -> mutex.up -> LOOP).

||SEMADEMO = (alice:LOOP || bob:LOOP || {alice, bob}::mutex:SEMAPHORE(1)).
```

Given the following implementations of `Alice`, `Bob` and `Semademo`

```
public class Alice extends Thread{
   Mutex mutex;

   public Alice(Mutex mutex){
      System.out.println("Hello, I'am Alice");
      this.mutex = mutex;
   }

   public void run(){
      try{
         for(int cuenta=0;cuenta<10;++cuenta){
            Thread.sleep(500);
            mutex.down();
            System.out.println("Alice is in the critical section");
            mutex.up();
         }
      } catch (InterruptedException e) {}
   }
}

public class Bob extends Thread{
   Mutex mutex;

   public Bob(Mutex mutex){
      System.out.println("Hello, I'am Bob");
      this.mutex = mutex;
   }

   public void run(){
      try{
         for(int cuenta=0;cuenta<10;++cuenta){
            Thread.sleep(1500);
            mutex.down();
            System.out.println("Bob is in the critical section");
            mutex.up();
         }
      } catch (InterruptedException e) {}
   }
}

public class Semademo {
   public static void main(String[] args){
      Mutex mutex = new Mutex();
      Thread alice = new Alice(mutex);
      Thread bob = new Bob(mutex);
      alice.start();
      bob.start();
   }
}
```

You have to

- Design the `Mutex` class following the the FST (this class should guaranty mutual exclusion).

  ```
  public class Mutex {
     ...
  }
  ```

- A possible execution of `Semademo` can be:

  ```
  Hello, I'am Alice
  Hello, I'am Bob
  Alice is in the critical section
  Alice is in the critical section
  Bob is in the critical section
  Alice is in the critical section
  Alice is in the critical section
  Alice is in the critical section
  Bob is in the critical section
  Alice is in the critical section
  Alice is in the critical section
  Alice is in the critical section
  Bob is in the critical section
  Alice is in the critical section
  Alice is in the critical section
  Bob is in the critical section
  Bob is in the critical section
  Bob is in the critical section
  Bob is in the critical section
  Bob is in the critical section
  Bob is in the critical section
  Bob is in the critical section
  ```

Explain why there is no alternation between Alice and Bob entering into the critical region.

Exercise 4 (2 Points). Following we ask to develop two versions of the merge sort (one sequential and the other parallel). We will sort floating-point numbers in order to avoid printing problems [1]. Following we use the list :

```
L = [27.0, 82.0, 43.0, 15.0, 10.0, 38.0, 9.0, 8.0].
```

Suppose that the module `msort` contains several functions the following two funcrions:

- Function `sep(L, N)` returns {`L1, L2`} so that `L1++L2 == L` and `length(L1)=N`. For instance

  ```
  32> msort:sep(L, 3).
  {[27.0,82.0,43.0],[15.0,10.0,38.0,9.0,3.0]}
  ```

  To divide `L` in two half use `msort:sep(L, length(L) div 2).`

- Function `merge(L1, L2)` returns the merge of two sorted lists, for instance:

  ```
  34> L1= [27.0, 43,0, 82.0].
  ...
  35> L2= [3.0, 9.0, 10.0, 15.0, 38.0].
  ...
  36> msort:merge(L1,L2).
  [3.0,9.0,10.0,15.0,27.0,38.0,43,0,82.0]
  ```

Now **you** have to:

- Complete the following sequential version of the merge sort function `ms(L)` returning `L` sorted.

  ```
  ms([]) -> ...;
  ms([A]) ->...];
  ms(L) ->
       {L1, L2} = sep(..., length(L) div 2),
       SL1 = ms(L1),
       SL2 = ...,
       ...
  ```

- Design a parallel version `pms` of the merge sort along the lines suggested going from `qs` to `pqs`.

Exercise 5 (1 Points). Compare the Java and Erlang programming languages.

- Explain the main differences between both programming languages.
- Explain how threads (in Java) and processes (in Erlang) are created. Give an easy example for each case.

[1] From Armstrong book (pag 29) we read: Remind: When the shell prints the value of a list it prints the list as a string, but only if all the integers in the list represent printable values. Given `L= [83, 117, 114, 112, 114, 105, 115, 101]`, if we ask execute `L` we get the string `"Surprise"`.