Concurrency, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
November 10, 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 *Share a Car.* (5 Points). `SHARE_A_CAR` is a company having `M` cars and `N` customers. We assume that `N` is greater than `M`. When a client needs a car, she tries to issue a `request` action. A control process will engage a `request` whenever there is some car available. When a customer returns a car, a `done` action is issued. The alphabet of the control process is {`request, done`}. Cars are used by the clients in mutual exclusion. Assume that in the initial state all cars are available.

1. We ask to complete the following description for the process `SHARE_A_CAR` modeling the company behavior. First of all we consider

```
const M = some number // M= number of cars
const N = some number greater than M // N= number of clients (or customers)

CLIENT = (request -> car[i:1..M].acquire -> car[i].release -> done ->CLIENT).

CAR = (acquire -> release -> CAR).

||CARS = (car[1..M]:CAR).
```

   - First complete the process `CONTROL` whose actions are `request` and `done`.

     ```
     CONTROL = ...
     ```

     Hint: The control is designed in order to assure that the client `request` action will be only executed iff there is some car available. In order to assure this, `CONTROL` keeps a count of the number of available cars.

     Attention: Be careful to avoid unnecessary guards. Remind that guards are important for point 3 when you have to code a Java monitor for `CONTROL`.

   - Second, complete the following `SHARE_A_CAR` process

     ```
     ||SHARE_A_CAR = (client[1..N]:CLIENT ||    ....CARS || client[1..N]::CONTROL).
     ```

2. Assume in this question (and only here) that `N = 2` and `M = 1`. Draw the LTS of `SHARE_A_CAR`.

3. Write a Java monitor for `CONTROL`.

4. Do you think that traces of the SHARE_A_CAR process satisfy the property process UNKNOWN below? Provide either an explanation for an affirmative answer or a trace of SHARE_A_CAR as counterexample for a negative one.

```
property UNKNOWN =
  (client[i:1..N].car[j:1..M].acquire-> client[i].car[j].release -> UNKNOWN).
```

5. Assume that K clients are greedy, issuing the action request with high priority (note that K≤ N). Discuss the progress properties of the resulting process.

Exercise 2 *ohter_ pmap.* (5 Points). Remind the function map(F, L) applying the function F to each element of a list L defined as

```
map(_, []) -> [];
map(F, [H|T]) -> [F(H)|map(F,T)].
```

Assuming the module is called other_pmap we have for instance:

```
> other_pmap:map(fun(X)->X+1 end, [1,2,3]).
[2,3,4]
```

We are interested to develop parallel versions of map called pmap (without the use of list comprehension).

1. We ask to complete the following code:

```
pmap(F, L) ->
    S = self(),
    Pids = map(fun(I) -> spawn(fun() -> do_f(S, F, I) end) end, L),
    %% gather the results
    gather(...).

do_f(Parent, F, I) ->...

gather([Pid|T]) -> receive {..., Ret} -> ... end;
gather([]) ->....
```

2. Let us consider two experiments. We'll map two different functions F over a list of 100 elements.

   (a) The first computes this:

   ```
   L = [L1, L2, ..., L100],
   map(fun lists:sort/1, L)
   ```

   Each of the elements in L is a list of 1,000 random integers.

   (b) The second computes this:

   ```
   L = [27,27,..., 27],
   map(fun ptests:fib/1, L)
   ```

   Here, L is a list of 100 twenty-sevens, and we compute the list [fib(27), fib(27), ...] 100 times. (fib is the Fibonacci function.)

   Now we replace map with pmap. In which case the speed-up is better. Explain the intuitive reasons. Explain briefly (according to your opinion) when parallel programs will run efficiently in Multicore architectures. In which cases make sense to parallelize sequential programs?

3. The pmap version we used cared about the order of the elements in the return value (we used selective receive to do this). If we didn't care about the order of the return values, we can write a different program. Please complete the following code:

```
foreach(F, [H|T]) -> F(...), foreach(..., T);
foreach(_, []) -> ok.

pmap1(F, L) ->
    S = ...,
    foreach(fun(I) ->  spawn(fun() -> do_f1(S,F, I) end) end, L),
    %% gather the results
    gather1(length(L), []).

do_f1(Parent, F, I) ->...

gather1(0, L) -> ...;
gather1(N,  L) -> receive {Ret} -> gather1(... , ...) end.
```