Concurrence, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
November 14, 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 (2 Points)
Consider the following FSP program where value 0 means `false` and 1 means `true`. Give a picture of the LTS corresponding to `TOGETHER` and argue your answer.

```
range B = 0..1

BOOLVAR        =  BOOLVAR[0],
BOOLVAR[u:B] = (read[u]   ->BOOLVAR[u]
                          |write[v:B]->BOOLVAR[v]
                          ).

SWAP = (read[b:B]->write[(b+1)%2]->SWAP).

||TOGETHER = (BOOLVAR||SWAP).
```

Exercise 2 (4 Points)
A professor needs some help to design a system to take care of groups of two students for virtual labs. He asks to the students of CPDS to design two safety properties. First let us consider the groups. A good group should contain two different members (in total there are 4 members). For instance in the following FSP, `GROUP` is an acceptable group because contains members 1 and 2, however `BAD_GRUP` is unacceptable because contains member 1 twice.

```
const N = 4
range R = 1..4
set MembersAlpha = {member[R], work[R][R]}

GROUP = MAKE_TEAM,
MAKE_TEAM = (member[1]->member[2]->WORK[1][2]),
WORK[1][2] = (work[1][2]->WORK[1][2]) + MembersAlpha.

BAD_GROUP= MAKE_TEAM,
MAKE_TEAM = (member[1]->member[1]->WORK[1][1]),
WORK[1][1] = (work[1][1]->WORK[1][1]) + MembersAlpha.
```

Consider the following items.

- (1 Point) Designs a safety property `DIFF_MEMBERS` that, provided a group has two members, it checks that members are different.

  o verify that each group contains exactly two different members. Use the following schema:

```
property DIFF_MEMBERS
  = (member[i:R]-> member[j:R] ->...),
WORK[i:R][j:R] =(work[i][j]->...).
```

- (2 Points) Design a safety property `ONCE` to verify that each member cannot be in more than one group (there are only two different groups). The property should run in the environment.

```
const N=4
range R = 1..4
set MembersApha={member[R], work[R][R]}

GROUP = MAKE_TEAM,
MAKE_TEAM = (member[1]->member[2]->WORK[1][2]),
WORK[1][2] = (work[1][2]->WORK[1][2])+MembersApha.

OTHER_GROUP = MAKE_TEAM,
MAKE_TEAM = (member[3]->member[4]->WORK[3][4]),
WORK[3][4] = (work[3][4]->WORK[3][4])+MembersApha.

NEW_GROUP = MAKE_TEAM,
MAKE_TEAM = (member[3]->member[1]->WORK[3][1]),
WORK[3][1] = (work[3][1]->WORK[3][1])+MembersApha.

||GROUPS =(a:GROUP||b:OTHER_GROUP||ONCE).

||BAD_GROUPS =(a:GROUP||b:NEW_GROUP||ONCE).
```

  Fill the code in the following schema:

```
property ONCE =(a.work[i:R][j:R]->A[i][j]| b.work[k:R][l:R]->B[k][l]),
A[i:R][j:R]=(...->A[i][j] |....->MEMBERS[i][j][k][l]),
B[k:R][l:R] =(...),
MEMBERS[i:R][j:R][k:R][l:R] =(when ...  b.work[k][l] -> WORK_FOREVER| ...),
WORK_FOREVER = (...).
```

- (1 Point) In the case of `BAD_GROUPS` give the shortest trace to the `ERROR` state.

Exercise 3 (3 Points)
The goal of this exercise is to construct a parallel version `par_sum_odds_evens(L)` of the sequential function `seq_odds_evens(L)`. Given a list of integers, for instance `L=[1,7,15,2,5,6,9])`, the function `seq_odds_evens(L)` splits `L` into two sub-lists that contain the odd and even integers `[1,7,15,5,9]`, `[2,6]` and returns the sums of elements in both lists,

```
poe:seq_sum_odds_evens([1,7,15,2,5,6,9]).
{37,8}
```

To solve this exercise you have to complete the following module **poe** (**p**arallel odds **e**vens) following guided steps

```
-module(poe).
-compile([export_all]).

sum(L) -> sum(L, 0).

sum([], N)    -> N;
sum([H|T], N) -> sum(T, H+N).

odds_evens(L) -> ...

seq_sum_odds_evens(L) -> ....

par_sum_odds_evens(L) ->...

 send_sum(P,L)-> ...

rcv(P) ->
    receive
        {P, X} -> X
    end.
```

- (1/2 Point) Complete the function `odds_evens(L)` splitting `L` into two sub-lists one containing the even numbers the other containing the odd numbers. The function returns both lists,

  ```
  poe:odds_evens([1,7,15,2,5,6,9]).
  {[1,7,15,5,9],[2,6]}
  ```

  Given an integer `X` in `L`, we check that `X` is even using `(X rem 2) =:= 0`. Please fill the blanks in the following code,

  ```
  odds_evens(L) ->
      Odds  = [X ||....],
      Evens = ...,
      {... , ....}.
  ```

- (1/2 Point) Give (a sequential version) of `seq_sum_odds_evens(L)`.
- (Points 2) Design a parallel version of `seq_sum_odds_evens(L)` called `par_sum_odds_evens(L)` based on the following schema:

  ```
  par_sum_odds_evens(L) ->
        ...
        P1 = spawn(..., send_sum, [..., Odds]),
        P2 = ...,
        ...

  send_sum(P,L)->....
  ```

  In this program **only two** processes `P1` and `P2` are spawned. Given `L = [1,7,15,2,5,6,9])`. Process `P1` receives `Odds = [1,7,15,5,9]` computes the sum `37` and send back this sum. Note that `P1` does not spawn any new process. Process `P2` deals with `[2,6]`.

Comment. This exercise is for academic purposes. The suggested parallel implementation is far from to be efficient.

Exercise 4 (1 Points)
Explain shorty the differences between Java and Erlang.