Concurrency, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
November 14, 2016

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 *Museum Audit* (6 Points). Inspired in M&K 3.6. A museum allows visitors to enter through the east entrance and leave through its west exit. Arrivals and departures are signaled to the museum controller by the turnstiles at the entrance and exit. At opening time, the museum director signals the controller that the museum is open and then the controller permits both arrivals and departures. At closing time, the director signals that the museum is closed, at which point only departures are permitted by the controller. The FSP description of the concurrent process `MUSEUM` is provided below in terms of the sequential processes `EAST`, `WEST`, `CONTROL` and `DIRECTOR`.

```
const N = 2

EAST = (arrive -> EAST).

WEST = (leave  -> WEST).

DIRECTOR = (open -> close -> DIRECTOR).

CONTROL        = CLOSED[0],
CLOSED[i: 0..N] = (when (i == 0) open -> OPENED[0]
                  |when (i > 0) leave -> CLOSED[i - 1]
                  ),
OPENED[i: 0..N] = (close -> CLOSED[i]
                  |when (i < N) arrive -> OPENED[i + 1]
                  |when (i > 0)  leave  -> OPENED[i - 1]
                  ).

||MUSEUM = (EAST || WEST || DIRECTOR || CONTROL).
```

1. (1 Point). Give a picture of the LST of `MUSEUM` for `N=2`. It has 6 states. Please try to get a picture easy to understand.

2. (2 Point) Define a safety property to prove that the predicate *the museum is empty* is a necessary condition to engage action `open`. Simpler definitions will obtain higher scores. How do you test at the analyzer that `MUSEUM` satisfies this property?

3. (1 Point) We define `SUCCESS_MUSEUM` as follows:

   ```
   ||SUCCESS_MUSEUM = MUSEUM << {arrive}.
   ```

   Give a picture of the LST corresponding to `SUCCESS_MUSEUM`. Explain shorty how `SUCCESS_MUSEUM` can be obtained from `MUSEUM` just deleting some some transitions (give examples).

4. (1 Point) Discuss the following two progress properties on a successful museum process `SUCCESS_MUSEUM`:

   ```
   progress ONE = {leave}
   progress TWO = {open}
   ```

5. (1 Point) Write a Java monitor corresponding to `CONTROL`. Solutions using `wait()` and `notifyAll()` will obtain higher scores.

Exercise 2 *Ping Pong* (4 Points). In the following example two processes are created and they send messages to each other a number of times.

1. (2 Points) In order to get the following execution:

   ```
   10> pingpong:start().
   Pong received ping
   Ping received pong
   Pong received ping
   Ping received pong
   Pong received ping
   Ping received pong
   Ping finished
   Pong finished
   ```

   please complete the following code (**in the special sheet at the end of this exam**). Note that, we assume that `pong` is first created to be able to give the identity of `pong` when `ping` was started. That is, `ping` must be able to know the identity of `pong` to be able to send a message to it.

   ```
   -module(pingpong).
   -export([start/0, ping/2, pong/0]).

   start() ->
       Pong_PID = spawn(..., pong, []),
       spawn(..., ping, [3, Pong_PID]).

   ping(0, Pong_PID) ->
       Pong_PID ! finished,
       io:format("Ping finished~n", []);

   ping(N, Pong_PID) ->
       Pong_PID ! {ping, ...},
       receive
           pong ->
               io:format("Ping received pong~n", [])
       end,
       ping(N - 1, ...).

   pong() ->
       receive
           finished ->
               io:format("Pong finished~n", []);
           {ping, ...} ->
               io:format("Pong received ping~n", []),
               Ping_PID ! ...,
               pong()
       end.
   ```

2. (2 Points). Remind that Erlang thus provides a mechanism for processes to be given names so that these names can be used as identities instead of `pids`. This is done by using:

   ```
   register(some_atom, Pid)
   ```

   Let us now rewrite the `pingpong` example using this and give the name `pong` to the `pong` process. Please fill the following code (**in the special sheet at the end of this exam**)

   ```
   -module(regpingpong).

   -export([start/0, ping/1, pong/0]).

   start() ->
       register(..., spawn(regpingpong, pong, [])),
       spawn(..., ping, [3]).


   ping(0) ->
       pong ! finished,
       io:format("Ping finished~n", []);

   ping(...) ->
       pong ! {ping, ...},
       receive
           pong ->
               io:format("Ping received pong~n", [])
       end,
       ping(...).

   pong() ->
       receive
           finished ->
               io:format("Pong finished~n", []);
           {ping, ...} ->
               io:format("Pong received ping~n", []),
               Ping_PID ! ...,
               pong()
       end.
   ```

Name:

ID:

Ex2.1

```
-module(pingpong).
-export([start/0, ping/2, pong/0]).

start() ->
    Pong_PID = spawn(pingpong, pong, []),
    spawn(..., ping, [3, Pong_PID]).

ping(0, Pong_PID) ->
    Pong_PID ! finished,
    io:format("Ping finished~n", []);

ping(N, Pong_PID) ->
    Pong_PID ! {ping, ...},
    receive
        pong ->
            io:format("Ping received pong~n", [])
    end,
    ping(N - 1, ...).

pong() ->
    receive
        finished ->
            io:format("Pong finished~n", []);
        {ping, ...} ->
            io:format("Pong received ping~n", []),
            Ping_PID ! ...,
            pong()
    end.
```

Name:

ID:

Ex2.2

```
-module(regpingpong).

-export([start/0, ping/1, pong/0]).

start() ->
    register(..., spawn(regpingpong, pong, [])),
    spawn(..., ping, [3]).


ping(0) ->
    pong ! finished,
    io:format("Ping finished~n", []);

ping(...) ->
    pong ! {ping, ...},
    receive
        pong ->
            io:format("Ping received pong~n", [])
    end,
    ping(...).

pong() ->
    receive
        finished ->
            io:format("Pong finished~n", []);
        {ping, ...} ->
            io:format("Pong received ping~n", []),
            Ping_PID ! ...,
            pong()
    end.
```