Concurrency, Parallelism and Distributed Systems (CPDS)
Module I: Concurrency
Facultat d'Informàtica de Barcelona
Final Exam
April 12, 2019

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)*Eating in rounds*
Consider three friends, Alice, Bob and Mary in a lunch eating in rounds

```
a_eat, b_eat, m_eat, a_eat, b_eat, m_eat, a_eat, b_eat, m_eat, ...
```

`LUNCH` process is defined as:

```
ALICE =(a_eat->ALICE).
BOB =(b_eat->BOB).
MARY = (m_eat->MARY).

FIRST_CONTROL =(a_eat->b_eat->m_eat->FIRST_CONTROL).

||LUNCH =(ALICE||BOB||MARY||FIRST_CONTROL).
```

Remember the M&K approach to design a monitor, sketched as follows:

```
FSP: when (cond)  act-> NEWSTAT

JAVA: public synchronized void act()
              throws InterruptedException{
          while.....
          .....
      }
```

- (1 Point). To design a monitor we ask you to redesign `FIRST_CONTROL` into a `SECOND_CONTROL` with explicit `when` guards. Please complete the following LTS:

  ```
  SECOND_CONTROL = TURN[1],
  TURN[i: 1..3]  = (when (...) a_eat->TURN[...] | ... |...).
  ```

  such that `||OTHER_LUNCH = (ALICE||BOB||MARY||SECOND_CONTROL)` works correctly.

- (1 Point). Design a monitor `SecondControl` corresponding to `SECOND_CONTROL`. Please follow the M&K schema:

```
class SecondControl{
   protected int turn = 1;
   public ... a_eat() throws InterruptedException{...}
   public ... b_eat( ) throws InterruptedException{...}
   public ... m_eat() throws InterruptedException{...}
```

Exercise 2 (6 Points) *All Crossing*
Consider the process `ALL_CROSSING` defined as follows:

```
ROAD = (car -> up -> car_cross -> down -> ROAD).
RAIL = (train -> green -> train_cross -> red -> RAIL).
SIGNAL = (green -> red -> SIGNAL | up -> down -> SIGNAL).

||ALL_CROSSING = (ROAD||RAIL||SIGNAL).
```

1. (2 Points) Complete the labeled transition system of figure 1. There is a total of 20 transitions.

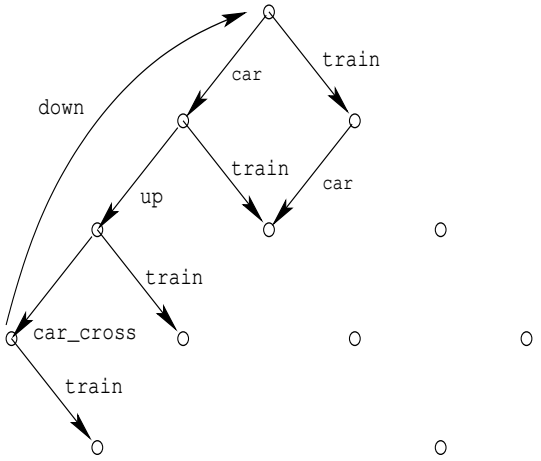

Figure 1: Labeled transition system `ALL_CROSSING`

2. (1 Point) Consider the property process `CAR` defined as:

   ```
   property CAR = (car -> car_cross -> CAR).
   ```

   Draw the transition system of `CAR` and discuss whether `ALL_CROSSING` satisfies `CAR`.

3. (1 Point) Consider the property process `BOTH` defined as:

   ```
   property BOTH = (car -> car_cross -> BOTH
                   |train -> train_cross -> BOTH
                   ).
   ```

   Draw the transition system of `BOTH` and discuss whether `ALL_CROSSING` satisfies `BOTH`.

4. (1 Point) Let `PRIOR_TRAIN` be:

   ```
   ||PRIOR_TRAIN = ALL_CROSSING << {train}.
   ```

   Discuss whether `PRIOR_TRAIN` satisfies:

   ```
   progress CAR_CROSS = {car_cross}.
   ```

5. (1 Point) Let `PRIOR_GREEN_TRAIN` be:

```
||PRIOR_GREEN_TRAIN = ALL_CROSSING << {green, train}.
```

Disscuss whether `PRIOR_GREEN_TRAIN` satisfies `CAR_CROSS`.

Exercise 3 (2 Points) *Erlang Processes.*
Given the following module

```
-module(tut14).
-export([start/0, say_something/2]).

say_something(What, 0) ->
    done;
say_something(What, Times) ->
    io:format("~p~n", [What]),
    say_something(What, Times - 1).
start() ->
    spawn(tut14, say_something, [hello, 3]),
    spawn(tut14, say_something, [goodbye, 3]).
```

Answer to the following questions.

- (1/2 Point). After compiling with `c(tut14)`, what is the trace corresponding to execute

  ```
  tut14:say_something(hello, 3).
  ```

- (1/2 Point). What is the trace corresponding to execute

  ```
  tut14:start().
  ```

- (1 Point). Explain shortly the answers.