Exam of Principles of Programming Languages, 2017.01.30

Notes:
- Total available time: **partial exam***:* 1h 20' (Exercises 1 and 2); **complete exam***:* 2h.
- You may use any written material you need, and write in Italian, if you prefer.
- You cannot use electronic devices during the exam.

Exercise 1 - Haskell (14 points)

1.1) Define a data structure, called Lt, for generic list of lists, where each list has a fixed length and such number is stored in the data structure.
1.2) Define a function, called checkLt, that takes an Lt and returns true if it is valid (i.e. every list in it have the correct size), false otherwise.
1.3) Define a function, called checklist, that takes a list t and an Lt, checks if all the sublists of t are in the given Lt, and uses Maybe to return the list of sublists of t that are not present in Lt.
Note: sublists must be contiguous, e.g. the sublists of size 2 of [1,2,3] are [1,2], [2,3].
1.4) Make Lt an instance of Functor.
Note: state all the types of the defined functions.

Exercise 2 - Erlang (6 points)

Consider the following Erlang code, implementing a parallel version of **map**:

```
-module(pmap).

-export([map/2, runit/3]).

runit(Proc, F, X) -> Proc ! F(X).

map(F, L) ->
    W = lists:map(fun(X) -> spawn(?MODULE, runit, [self(), F, X]) end, L),
    lists:map(fun (P) ->
                      receive
                          V -> V
                      end
              end, W).
```

Is this code correct? Explain briefly how it works, and, if it is not correct, how to fix it.

Exercise 3 - Scheme (5+6 points)

3.1) Consider a list L and a natural number k. Define an iterator with a closure, which returns, in turn, all the contiguous sublists of L of length k, and a symbol 'end at the end. It is possible to use the function take provided by Racket, e.g. (take '(a b c) 2) is '(a b).

3.2) Define a function checklist analogous to the one of Exercise 1.3, by using only a foldl as the main loop.

For instance a call (checklist '(b b a a b b b c) '((a b)(b a)(b b))) should return ((b c) (a a)).

Solutions

Ex 1

```haskell
data Lt a = Lt Int [[a]] deriving (Show, Eq)

checkLt :: Lt a -> Bool
checkLt (Lt _ [])     = True
checkLt (Lt k (x:xs)) = length x == k && checkLt (Lt k xs)

sublists :: Int -> [a] -> [[a]] -> [[a]]
sublists size lst res =
    let factor = take size lst in
        if length factor == size
        then sublists size (tail lst) (factor:res)
        else res

checklist :: Eq a => [a] -> Lt a -> Maybe [[a]]
checklist lst (Lt size ltf) =
    let factors  = sublists size lst []
        nfactors = [x | x <- factors, not (x `elem` ltf)]
    in if nfactors == [] then Nothing else Just nfactors

instance Functor Lt where
    fmap f (Lt k lst) = Lt k $ map (\x -> map f x) lst
```

Ex 2

The code is wrong, since we could get a list which is not [F(X) | X in L], but a permutation thereof.
To fix it, we could send each worker's pid, and selectively check it to build the properly ordered list:

```erlang
runit(Proc, F, X) -> Proc ! {self(), F(X)}.
map(F, L) ->
    W = lists:map(fun(X) -> spawn(?MODULE, runit, [self(), F, X]) end, L),
    lists:map(fun (P) -> receive
                           {P, V} -> V
                         end
              end, W).
```

Ex 3

```scheme
(define (sublists lst k)
  (let ((curr lst)
        (size (length lst)))
    (lambda ()
      (if (< size k)
        'end
        (begin
          (let ((v (take curr k)))
            (set! size (- size 1))
            (set! curr (cdr curr))
            v))))))

(define (checklist lst factors)
  (define k (length (car factors)))
  (define iter (sublists lst k))
  (foldl
    (lambda (x r)
      (let ((curr (iter)))
        (if (member curr (cons 'end factors))
          r
          (cons curr r))))
    '()
    lst))
```