Principles of Programming Languages, 2022.02.10

Important notes
- Total available time: 2h.
- You may use any written material you need, and write in Italian, if you prefer.
- You cannot use electronic devices during the exam: every phone must be turned off and kept on your table.
- You cannot use library functions not covered in class in your code.

Exercise 1, Scheme (8 pts)

Consider the following code:

```
(define (r x y . s)
  (set! s (if (cons? s) (car s) 1))
  (lambda ()
    (if (< x y)
        (let ((z x))
          (set! x (+ s x))
          z)
        y)))
```

1. What can we use *r* for? Describe how it works and give some useful examples of its usage.
2. It makes sense to create a version of *r* without the *y* parameter? If the answer is yes, implement such version; if no, explain why.

Exercise 2, Haskell (12 pts)

Consider a data structure Gtree for general trees, i.e. trees containg some data in each node, and a variable number of children.

1. Define the Gtree data structure.
2. Define gtree2list, i.e. a function which translates a Gtree to a list.
3. Make Gtree an instance of Functor, Foldable, and Applicative.

Exercise 3, Erlang (12 pts)

Define a parallel lexer, which takes as input a string *x* and a chunk size *n*, and translates all the words in the strings to atoms, sending to each worker a chunk of *x* of size *n* (the last chunk could be shorter than *n*). You can assume that the words in the string are separated only by space characters (they can be more than one - the ASCII code for ' ' is 32); it is ok also to split words, if they overlap on different chunks. E.g.

`plex("this is a nice  test", 6)` returns `[[this,i],[s,a,ni],[ce,te],[st]]`

For you convenience, you can use the library functions:
- `lists:sublist(List, Position, Size)` which returns the sublist of List of size Size from position Position (starting at 1);
- `list_to_atom(Word)` which translates the string Word into an atom.

Solutions

Es 1

It is a generator implemented as a closure, which returns the numbers from x to y with step s (+1 if s is not defined). When y is reached, it returns it indefinitely.

Yes, y is the upper limit and we could drop it:

```
(define (r1 x . s)
  (set! s (if (cons? s) (car s) 1))
  (lambda ()
    (let ((z x))
      (set! x (+ s x))
      z)))
```

Es 2

```
data Gtree a = Tnil | Gtree a [Gtree a] deriving Show

gtree2list :: Gtree a -> [a]
gtree2list Tnil = []
gtree2list (Gtree x xs) = x : concatMap gtree2list xs

instance Functor Gtree where
    fmap f Tnil = Tnil
    fmap f (Gtree x xs) = Gtree (f x) (fmap (fmap f) xs)

instance Foldable Gtree where
    foldr f i t = foldr f i $ gtree2list t

Tnil +++ x = x
x +++ Tnil = x
(Gtree x xs) +++ (Gtree y ys) = Gtree y ((Gtree x []:xs) ++ ys)

gtconcat = foldr (+++) Tnil
gtconcatMap f t = gtconcat $ fmap f t

instance Applicative Gtree where
    pure x = Gtree x []
    x <*> y = gtconcatMap (\f -> fmap f y) x
```

Es 3

```
split(List, Size, Pos, End) when Pos < End ->
    [lists:sublist(List, Pos, Size)] ++ split(List, Size, Pos+Size, End);
split(_, _, _, _) -> [].

lex([X|Xs], []) when X =:= 32 -> % 32 is ' '
    lex(Xs, []);
lex([X|Xs], Word) when X =:= 32 ->
    [list_to_atom(Word)] ++ lex(Xs, []);
lex([X|Xs], Word) ->
    lex(Xs, Word++[X]);
lex([], []) ->
    [];
lex([], Word) ->
    [list_to_atom(Word)].

run(Pid, Data) ->
    Pid!{self(), lex(Data, [])}.

plex(List, Size) ->
    Part = split(List, Size, 1, length(List)),
    W = lists:map(fun(X) ->
                      spawn(?MODULE, run, [self(), X])
                  end, Part),
    lists:map(fun (P) ->
                      receive
                          {P, V} -> V
                      end
              end, W).
```