Principles of Programming Languages, 2017.02.27

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.
- You can choose to answer either to Exercise 3A or 3B, but not both.

Exercise 1, Scheme (5+5 pts)

Consider the following code:

```
(define (maki lst)
  (call/cc (lambda (exit)
             (for-each
              (lambda (x)
                (call/cc (lambda (yield)
                           (exit (cons x yield))
                           )))
              lst))))
(define (doit)
  (let ((x (maki '(a b c d))))
    (when (cons? x)
      (displayln (car x))
      ((cdr x)))))
```

1) Give a brief description on how it works, and report the output of running (doit).
2) Write a macro, called *curried*, for defining curried functions that works in this example way:
(define f (curried (x y) (+ x y)))
so that *f* is a function that can be used e.g. like this: *((f 3) 4)* returns 7.

Exercise 2, Haskell (3+3+9 pts)

Consider this data declaration: *data LolStream x = LolStream Int [x]*. The list [x] must always be an *infinite list* (also called a *stream*), while the first parameter, of type Int, when positive represents the fact that the stream is periodic, while it is not periodic if negative (0 is left unspecified). E.g. these are two valid LolStreams: *LolStream -1 [1,2,3...]; LolStream 2 [1,2,1,2,1,2…]*.

1) Define a function lol2lolstream which takes a finite list of finite lists $[h_1, h_2, \ldots h_n]$, and returns

$$LolStream\ (|h_1| + |h_2| + \ldots + |h_n|)\ \ (h_1 ++ h_2 ++ \ \ldots\ ++ h_n ++ h_1 ++ h_2 ++ \ \ldots)$$

2) Make LolStream an instance of Eq. (Note: == should terminate, when possible.)

3) Make LolStream an instance of Functor, Foldable, and Applicative.

Bonus (**+3 pts**) Make LolStream an instance of Monad.

Exercise 3A, Erlang (5 pts)

Define a parallel implementation of the classical functional higher order function *filter*.

Exercise 3B, Prolog (5 pts)

Define a *merge* predicate to merge two ordered lists of numbers like in the merge-sort algorithm.
E.g. *merge([1, 3, 7], [-2, 2], X). X = [-2, 1, 2, 3, 7]*
The correct usage of cut is required.

Solutions

Scheme

The continuation “yield” is the end of the body of the for-each, while the continuation “exit” is the when construct. Thus, the call ((cdr x)) will jump back to “yield” and “exit” to the when, making an implicit loop. The output is:

```
a
b
c
d
```

```
(define-syntax curried
  (syntax-rules ()
    ((_ (x) e1 ...)
     (lambda (x) e1 ...))
    ((_ (x1 x2 ...) e1 ...)
     (lambda (x1)
       (curried (x2 ...) e1 ...)))))
```

Haskell

```
lol2stream lol = let v = lol ++ v
                 in concat v
lol2lolstream = let v = lol2stream lol
                    n = sum $ map length lol
                in LolStream n v
-- utilities
destream (LolStream v l) = if v < 0 then l else take v l
periodic (LolStream v _) = v >= 0
-- instances
instance Eq a => Eq (LolStream a) where
  v == v1 = (destream v) == (destream v1)

instance Functor LolStream where
  fmap f (LolStream n l) = LolStream n (map f l)

instance Applicative LolStream where
  pure v = lol2lolstream [[v]]
  v1@(LolStream n1 f) <*> v2@(LolStream n2 g) =
    LolStream (n1 * n2) $ lol2stream [(destream v1) <*> (destream v2)]

instance Foldable LolStream where
  foldr f v t = foldr f v $ destream t

instance Monad LolStream where
  v >>= f = lol2lolstream [destream v >>= \x -> destream (f x)]
```

Erlang

```
runit(Proc, F, X) ->
    Proc ! {self(), X, F(X)}.

pfilter(F, L) ->
    W = lists:map(fun(X) ->
                          spawn(?MODULE, runit, [self(), F, X])
                  end, L),
    lists:foldl(fun (P, Vo) ->
                        receive
                            {P, X, true}  -> Vo ++ [X];
                            {P, _, false} -> Vo
                        end
                end, [], W).
```

Prolog

```
merge([],Ys,Ys) :- !.
merge(Xs,[],Xs) :- !.
merge([X|Xs],[Y|Ys],[X|Rs]) :-  X < Y, !, merge(Xs,[Y|Ys],Rs).
merge([X|Xs],[Y|Ys],[Y|Rs]) :- X >= Y, !, merge([X|Xs],Ys,Rs).
```