Principles of Programming Languages, 2023.09.12

Important notes
- Total available time: 2h.
- You may use any written material you need, and write in English or in Italian.
- 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 (11 pts)

1. Design a construct to define multiple functions with the same number of arguments at the same time. The proposed syntax is the following:

   (multifun <list of function names> <list of parameters> <list of bodies>).

   E.g.
   ```
   (multifun (f g) (x)
          ((+ x x x)
           (* x x)))
   ```
   defines the two functions *f* with body *(+ x x x)* and g with body *(* x x)*, respectively.
2. Would be possible to define something similar, but using a procedure and lambda functions instead of a macro? If yes, do it; if no, explain why.

Exercise 2, Haskell (11 pts)

Consider the *binary tree* data structure as seen in class.

1. Define a function *btrees* which takes a value *x* and returns an infinite list of binary trees, where:
   1. all the leaves contain *x*,
   2. each tree is complete,
   3. the first tree is a single leaf, and each tree has one level more than its previous one in the list.
2. Define an infinite list of binary trees, which is like the previous one, but the first leaf contains the integer 1, and each subsequent tree contains leaves that have the value of the previous one incremented by one.
   E.g. *[Leaf 1, (Branch (Leaf 2)(Leaf 2), ...]*
3. Define an infinite list containing the count of nodes of the trees in the infinite list of the previous point.
   E.g. *[1, 3, ...]*

Write the signatures of all the functions you define.

Exercise 3, Erlang (11 pts)

Consider the infinite list of binary trees of Exercise 2: instead of infinite lists, we want to create processes which return the current element of the "virtual infinite list" with the message *next*, and terminate with the message *stop*.

1. Define a function *btrees* to create a process corresponding to the infinite tree of Exercise 2.1.
2. Define a function *incbtrees* to create a process corresponding to the infinite tree of Exercise 2.2.

Notes: for security reasons, processes must only answer to their creating process; to define trees, you can use suitable tuples with atoms as customary in Erlang (e.g. *{branch, {leaf, 1}, {leaf, 1}}*).

Note: multichance students do not need to solve Exercise 1.

Solutions

Ex 1

1.

```scheme
(define-syntax multifun
  (syntax-rules ()
    ((_ (f) (x ...) (b))
     (define (f x ...) b))
    ((_ (f . fs) (x ...) (b . bs))
     (begin
       (define (f x ...) b)
       (multifun fs (x ...) bs)))))
```

2.

Of course we can use a list of lambda functions instead of the list of bodies (alas, we need to replicate the list of parameters on each body). The main problem is that we cannot bind the top-level function names from inside a procedure, so the answer is no.

Ex 2

```haskell
data Btree a = Leaf a | Branch (Btree a)(Btree a) deriving (Show, Eq)

instance Functor Btree where
    fmap f (Leaf x) = Leaf (f x)
    fmap f (Branch x y) = Branch (fmap f x) (fmap f y)

addLevel :: Btree a -> Btree a
addLevel t = Branch t t

btrees :: a -> [(Btree a)]
btrees x = (Leaf x) : [ addLevel t | t <- btrees x]

incBtrees :: [Btree Integer]
incBtrees = (Leaf 1) : [ addLevel $ fmap (+1) t | t <- incBtrees]

counts :: [Integer]
counts = map (\x -> 2^x - 1) [1..]
```

Ex 3

```erlang
btrees_body(T, Pid) ->
    receive
        next ->
            T1 = {branch, T, T},
            Pid ! T1,
            btrees_body(T1, Pid);
        stop ->
            T
    end.

btrees(N, Pid) ->
    spawn(?MODULE, btrees_body, [{leaf, N}, Pid]).


inctree({leaf, X}) ->
    {leaf, X+1};
inctree({branch, X, Y}) ->
    {branch, inctree(X), inctree(Y)}.

incbtrees_body(T, Pid) ->
    receive
        next ->
            T1 = inctree(T),
            T2 = {branch, T1, T1},
            Pid ! T2,
            incbtrees_body(T2, Pid);
        stop ->
            T
    end.

incbtrees(Pid) ->
    spawn(?MODULE, incbtrees_body, [{leaf, 0}, Pid]).
```