Principles of Programming Languages, 2019.06.28

Notes

- Total available time: 1h 30'.
- 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, Scheme (12 pts)

Consider this data definition in Haskell: data Tree a = Leaf a | Branch (Tree a) a (Tree a)

Define an OO analogous of this data structure in Scheme using the technique of "closure as classes" as seen in class, defining the *map* and *print* methods, so that:

(define t1 (Branch (Branch (Leaf 1) -1 (Leaf 2)) -2 (Leaf 3)))

((t1 'map (lambda (x) (+ x 1))) 'print)

should display: *(Branch (Branch (Leaf 2) 0 (Leaf 3)) -1 (Leaf 4))*

Exercise 2, Haskell (12 pts)

1) Define a *Tritree* data structure, i.e. a tree where each node has at most 3 children, and every node contains a value.

2) Make *Tritree* an instance of Foldable and Functor.

3) Define a *Tritree* concatenation *t1 +++ t2*, where *t2* is appended at the bottom-rightmost position of *t1*.

4) Make Tritree an instance of Applicative.

Exercise 3, Erlang (8 pts)

Consider the following code:

```
bu(C) ->
    receive
        {get, From} ->
            if
                C =:= [] ->
                    From ! empty, bu([]);
                true ->
                    [H|T] = C,
                    From ! H, bu(T)
            end;
        {put, Data} -> bu([Data | C])
    end.
pr(From, To, B, Master) ->
    if
        From =< To ->
            B ! {put, From},
            io:format("~w in ~p~n", [self(), From]),
            pr(From+1, To, B, Master);
        true ->
            Master ! {self(), done}
    end.
```

```
cf(B) ->
    B ! {get, self()},
    receive
        empty ->
            io:format("~w: no data~n", [self()]),
            cf(B);
        V ->
            io:format("~w out ~p~n", [self(), V]),
            cf(B)
    end.
main() ->
    B  = spawn_link(?MODULE, bu, [[]]),
    P1 = spawn(?MODULE, pr, [0,4,B,self()]),
    P2 = spawn(?MODULE, pr, [5,9,B,self()]),
    P3 = spawn(?MODULE, pr, [10,15,B,self()]),
    spawn_link(?MODULE, cf, [B]),
    spawn_link(?MODULE, cf, [B]),
    receive
        {P3, done} ->
            receive
                {P2, done} ->
                    receive
                        {P1, done} -> ok
                    end
            end
    end.
```

1) Describe what this system does and how it works – a high level description is required, not a line-by-line

commentary.

2) Are there possible problems with it? Explain your answer and, if positive, give a possible fix.

3) Is this a possible output of the system (or the fixed system, if the answer at point 2 is positive)? Explain your answer

```
<0.68.0> in 0
<0.69.0> in 5
<0.70.0> in 10
<0.71.0> out 10
<0.72.0> out 5
<0.68.0> in 1
<0.69.0> in 6
<0.70.0> in 11
<0.71.0> out 11
<0.72.0> out 6
<0.68.0> in 2
<0.69.0> in 7
<0.70.0> in 12
<0.68.0> in 3
<0.69.0> in 8
<0.70.0> in 13
<0.71.0> out 12
<0.72.0> out 7
<0.68.0> in 4
<0.69.0> in 9
<0.70.0> in 14
<0.71.0> out 14
<0.72.0> out 9
<0.70.0> in 15
```

Solutions

Es 1

```
(define (Branch t1 x t2)
  (define (print)
    (display "(Branch ")
    (t1 'print)
    (display " ")
    (display x)
    (display " ")
    (t2 'print)
    (display ")"))
  (define (map f)
    (Branch (t1 'map f) (f x) (t2 'map f)))
  (lambda (message . args)
    (apply
     (case message
       ((print) print)
       ((map) map)
       (else (error "Unknown")))
     args)))

(define (Leaf x)
  (define (print)
    (display "(Leaf ")
    (display x)
    (display ")"))
  (define (map f)
    (Leaf (f x)))
 (lambda (message . args)
    (apply
     (case message
       ((print) print)
       ((map) map)
       (else (error "Unknown")))
     args)))
```

Es 2

```
data Tritree a = Nil | Tritree a (Tritree a)(Tritree a)(Tritree a) deriving (Eq, Show)

instance Functor Tritree where
  fmap f Nil = Nil
  fmap f (Tritree x t1 t2 t3) = Tritree (f x)(fmap f t1)(fmap f t2)(fmap f t3)

instance Foldable Tritree where
  foldr f i Nil = i
  foldr f i (Tritree x t1 t2 t3) = f x $ foldr f (foldr f (foldr f i t3) t2) t1

x +++ Nil = x
Nil +++ x = x
(Tritree x t1 t2 Nil) +++ t = (Tritree x t1 t2 t)
(Tritree x t1 t2 t3) +++ t =  (Tritree x t1 t2 (t3 +++ t))

ttconcat t = foldr (+++) Nil t
ttconcmap f t = ttconcat $ fmap f t

instance Applicative Tritree where
  pure x = (Tritree x Nil Nil Nil)
  x <*> y = ttconcmap (\f -> fmap f y) x
```

Es 3

Being bu and the two cf spawn-linked, we need to terminate the main process with exit to kill them. If we do not do so, we get an unending sequence of “no data”. Hence, we could change “{P1, done} -> ok” into {P1, done} -> exit(done)”.
The output of the system is correct, because it is possible that some elements remain in the buffer, being the buffer managed as a stack.