Principles of Programming Languages

2015.07.06

Notes

- NAME: ______________________________
- Did you present a small project? YES / NO
- Total available time: 2h.
- You may use any written material you need.
- You cannot use computers or phones during the exam.

1 Scheme

1.1 Closures (6 points)

Consider this code:

```
(define (producer ag1 ag2)
  (let loop ((i 1))
   (if (< i 10)
     (begin
       ((if (odd? i) ag1 ag2) i)
       (loop (+ i 1)))
     (cons
       (ag2 'end)
       (ag1 'end)))))
```

Define two closures `clos1` and `clos2` for passing them to `producer` such that it returns the list `'(20 9 7 5 3 1)` (note that $20 = 2 + 4 + 6 + 8$).

1.2 Macro (6 points)

Define a macro `multiple-apply`, `(multiple-apply (fun-1 fun-2 ...) to list-1 list-2 ...)` where `fun-i` are functions and `to` is a keyword, which returns a list containing the result of applying `fun-i` to `list-i`.

E.g.

```
> (multiple-apply (+ - *) to '(1 2 3) '(3 4) '(9 -2))
(6 -1 -18)
```

2 Haskell

2.1 Split (5 points)

Assume this is a possible Haskell variant of `producer` in Exercise 1, in which the closures' states are made explicit.

```
producer ag1 ag2 = prod’ ag1 [] ag2 0 1
    where
        prod’ ag1 st1 ag2 st2 i | i >= 10 = (st2:st1)
        prod’ ag1 st1 ag2 st2 i | odd  i = prod’ ag1 (ag1 i st1) ag2 st2 (i+1)
        prod’ ag1 st1 ag2 st2 i | even i = prod’ ag1 st1 ag2 (ag2 i st2) (i+1)
```

1. Define two suitable functions for `producer`'s two arguments, such that its call returns [20,9,7,5,3,1].

2. Write the type of `prod’`, assuming that all the numbers have type Int.

2.2 Duo-fold (6 points)

Define an higher-order function called `duofold`, which takes two binary functions f and g, a starting value t and a (finite) list $[e_1, e_2, \ldots]$, and returns $\ldots f(g(f(t, e_1), e_2), e_3), \ldots$. Please, write also its type.

Example: `duofold (+) (-) 0 [1,2,3,4]` returns -2 (i.e. $0 + 1 - 2 + 3 - 4$).

3 Prolog

3.1 Check length (4 points)

Define a predicate `lile`, which, given a list, check if it contains its own length. E.g. `lile([2,1])` is true, while `lile([1,2,1])` is not.

3.2 Deep check length (6 points)

Define a version of the predicate `lile`, called `lileg`, which takes a list of any depth, and check if all the lists in it contain their length. E.g. `lileg([2,[2,[1]],3])` is true.

Solutions

Scheme

```
(define clos1
  (let ((data '()))
   (lambda (i)
     (if (eq? i 'end)
       data
       (set! data (cons i data))
       ))))

(define clos2
  (let ((data 0))
   (lambda (i)
     (if (eq? i 'end)
       data
       (set! data (+ i data))
       ))))

(define-syntax multiple-apply
  (syntax-rules (to)
    ((_ (f1 ...) to l1 ...)
     (list (apply f1 l1) ...))))
```

Haskell

```
prod' :: (Int -> [Int] -> [Int]) -> [Int] -> (Int -> Int -> Int) -> Int -> Int -> [Int]
producer (:) (+)

duofold :: (t1 -> t -> t1) -> (t1 -> t -> t1) -> t1 -> [t] -> t1
duofold f g v [] = v
duofold f g k (x:xs) = duofold g f (f k x) xs
```

Prolog

```
lile(L) :- length(L,N), !, member(N,L).

iflc([]) :- !.
iflc([X|Xs]) :- atomic(X), !, iflc(Xs).
iflc([X|Xs]) :- lileg(X),  !, iflc(Xs).

lileg(L) :- lile(L), !, iflc(L).
```