Principles of Programming Languages

2015.09.22

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 Co-sublist (5 points)

Consider a list $(x_0 x_1 \dots x_n)$. Its *sublist* from i to j is the list $(x_i, x_{i+1} \dots x_j)$. Define the procedure `co-sublist` which, given a list L and two indexes i and j, $i \leq j$, returns the list of ordered elements of L that are not in the sublist from i to j. You cannot use procedures with side effects in your code (e.g. `set!`).

E.g. `(co-sublist '(1 2 3 4 5 6) 1 3)` should be `(1 5 6)`.

1.2 Fancy Sublist (5 points)

Define this construct: `(subl` $e_1 e_2 \dots$ `->` $e_i \dots e_j$ `<-` $e_{j+1} \dots e_n$`)`; its evalutation returns the sublist $(e_i \dots e_j)$.

E.g. `(subl 1 -> 2 3 4 <- 5 6)` should be `(2 3 4)`.

2 Haskell

2.1 Type definition and accessor (3 points)

Define the `Bilist` data-type, which is a container of two homogeneous lists. Define an accessor for `Blist`, called `bilist_ref`, that, given an index i, returns the pair of values at position i in both lists.

E.g. `bilist_ref (Bilist [1,2,3] [4,5,6]) 1` should return (2,5).

2.2 Oddeven (5 points)

Define a function, called `oddeven`, that is used to build a `Bilist x y` from a simple list. `oddeven` takes all the elements at odd positions and put them in `y`, while all the other elements are put in `x`, maintaining their order. You may assume that the given list has an even length (or 0). Write also all the types of the functions you define.

E.g. `oddeven [1,2,3,4]` must be `Bilist [1,3] [2,4]`.

2.3 Inverse oddeven (5 points)

Define an inverse of `oddeven`, e.g. `inv_oddeven $ oddeven [1,2,3,4]` must be `[1,2,3,4]`. Write also all the types of the functions you define.

2.4 Position of maximum (5 points)

Define a function, called `bilist_max`, that given an input `Bilist` $[x_1, x_2, \ldots, x_n]$ $[y_1, y_2, \ldots, y_n]$, where $x_k + y_k$, for $1 \leq k \leq n$, is the maximum, returns k.

E.g.

```
> bilist_max (Bilist [3,2,-1] [2,1,7])
2
```

3 Prolog (5 points)

Given a pair of lists $[x_1, x_2, \ldots, x_n]$ and $[y_1, y_2, \ldots, y_n]$, define a deterministic predicate `maxsum` to obtain the maximum value of $x_k + y_k$. The two lists are assumed to have the same length.

E.g.

```
?- maxsum([3,2,-1],[2,1,7],X).
X = 6.
```

Solutions

Scheme

```
(define (co-sublist L start end)
  (let loop ((p 0)
             (res '())
             (ls L))
    (cond
      ((null? ls)
       res)
      ((or (< p start)(> p end))
       (loop (+ p 1)
             (append res (list (car ls)))
             (cdr ls)))
      (else
       (loop (+ p 1)
             res
             (cdr ls))))))

(define -> '->)
(define <- '<-)

(define (subl . args)
  (let loop ((state #f)
             (res '())
             (ls args))
    (cond
      ((null? ls)
       res)
      ((eq? (car ls) '->)
       (loop #t res (cdr ls)))
      ((eq? (car ls) '<-)
       (loop #f res (cdr ls)))
      (state
        (loop state (append res (list (car ls))) (cdr ls)))
      ((not state)
        (loop state res (cdr ls))))))
```

Haskell

```
data Bilist a = Bilist [a] [a] deriving (Show, Eq)

bilist_ref (Bilist l r) pos = (l !! pos, r !! pos)

oddevenh :: [a] -> [a] -> [a] -> Bilist a
oddevenh [] ev od = Bilist ev od
oddevenh (x:xs) ev od = oddevenh xs od (ev++[x])

oddeven :: [a] -> Bilist a
oddeven l = oddevenh l [] []
```

```
inv_oddeven :: Bilist a -> [a]
inv_oddeven (Bilist l r) = foldl (++) [] $ map (\(x,y) -> [x,y]) $ zip l r

bilist_maxh (Bilist (l:ls) (r:rs)) pos curmax maxpos | l+r > curmax =
        bilist_maxh (Bilist ls rs) (pos+1) (l+r) pos
bilist_maxh (Bilist (l:ls) (r:rs)) pos curmax maxpos =
        bilist_maxh (Bilist ls rs) (pos+1) curmax maxpos
bilist_maxh _ _ _ maxpos = maxpos

bilist_max (Bilist (l:ls) (r:rs)) = bilist_maxh (Bilist ls rs) 1 (l+r) 0
```

Prolog

```
maxsum([X], [Y], M) :- !, M is X+Y.
maxsum([X|Xs], [Y|Ys], V) :- V is X+Y, maxsum(Xs, Ys, M), V > M, !.
maxsum([X|Xs], [Y|Ys], M) :- maxsum(Xs, Ys, M).
```