Principles of Programming Languages

2015.02.12

Notes

- 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 Split (4 points)

Define a tail-recursive function *split*, which, given a list L and a number n, returns a vector of two elements: the prefix sublist of L, with elements up to the n-th, and the remaining suffix.

E.g.

```
> (split '(0 1 2 3 4) 2)
'#((0 1) (2 3 4))
```

1.2 Factors (6 points)

Use *split* to define the function *3-factors*, which, given a list L, returns all the possible contiguous sublists A, B, C, such that `(equal? L (append A B C))`. A,B,C, cannot be empty.

E.g.

```
> (3-factors '(0 1 2 3 4))
'(((0 1 2) (3) (4))
  ((0 1) (2 3) (4))
  ((0 1) (2) (3 4))
  ((0) (1 2 3) (4))
  ((0) (1 2) (3 4))
  ((0) (1) (2 3 4)))
```

2 Haskell

2.1 Split (3 points)

Implement *split* in Haskell, noting that the Scheme version returns a vector: is there a more suitable type in Haskell for the job? If so, use it.

2.2 Factors (7 points)

Implement *3-factors* in Haskell, noting that the Scheme version returns a list of lists: is there a more suitable type in Haskell for the job? If so, use it.

2.3 Types (2 points)

Declare all the types of the defined functions.

3 Prolog

3.1 BetweenAB (5 points)

Define a predicate *betweenAB*, which, given a list L, checks if all the atoms a in L are at the beginning, and all the atoms b are at the end of L. E.g. `betweenAB([a,a,c,c,d,b,b,b])` is true.

3.2 DeepBetweenAB (6 points)

Define a "deep" version of *betweenAB*, which checks that all the lists in L have the same property of having all the atoms a at the beginning, and all the atoms b at the end.

E.g.

```
> deepBetweenAB([a,a,[a,[a,b],b],c,[a,a,c,c,[a,b,b,b],b],b,b]).
true
```

Solutions

Scheme

```
(define (split lst n)
  (define (split-h pre lst n)
    (if (< n 1)
        (vector pre lst)
        (split-h (append pre (list (car lst)))
                 (cdr lst)
                 (- n 1))))
  (split-h '() lst n))

(define (3-factors lst)
  (let ((out '())
        (n (- (length lst) 1)))
    (let outer ((i 1))
      (when (< i n)
        (let inner ((j (+ 1 i)))
          (when (<= j n)
            (let* ((v1 (split lst i))
                   (a  (vector-ref v1 0))
                   (b  (vector-ref v1 1))
                   (v2 (split b (- j i)))
                   (c  (vector-ref v2 0))
                   (d  (vector-ref v2 1)))
              (set! out (cons (list a c d) out))
              (inner (+ j 1)))))
        (outer (+ i 1))))
    out))
```

Haskell

```
split :: (Num a, Eq a) => [b] -> a -> ([b], [b])
split = split_h [] where
  split_h pre lst 0 = (pre, lst)
  split_h pre (x:xs) n = split_h (pre ++ [x]) xs (n-1)

threeFactors :: [a] -> [([a], [a], [a])]
threeFactors lst =
  let n = length lst - 1
  in let outer i out =
           if i < n then
             let inner j out' =
                   if j <= n then
                     let (a, b) = split lst i
                     in let (c, d) = split b (j-i)
                        in inner (j+1) ((a,c,d):out')
                   else out'
             in outer (i+1) (inner (i+1) out)
           else out
     in outer 1 []
```

Prolog

```
as([]) :- !.
as([a|Xs]) :- !, as(Xs).

bs([]) :- !.
bs([b|Xs]) :- !, bs(Xs).

betweenAB(X) :-
        append(A,Y,X),
        append(C,B,Y),
        as(A),
        bs(B),
        \+ member(a,C),
        \+ member(b,C).

atomOrAB([X]) :- !, (atom(X) ; deepBetweenAB(X)).
atomOrAB([X|Xs]) :- (atom(X) ; deepBetweenAB(X)), !, atomOrAB(Xs).

deepBetweenAB(X) :- betweenAB(X), !, atomOrAB(X).
```