Principles of Programming Languages

2013.09.19

Notes

- Total available time: 1h 30'.
- You may use any written material you need.
- You cannot use computers or phones during the exam.

1 Scheme (9 points)

Define an object, using the "closures as objects" technique seen in class, that works as a simple immutable container of integer numbers. It must offer two methods: `member?`, that checks if a number is contained in the object; and `subsetsum`, that checks if a given number is the sum of elements contained in the object (at most each element must be taken once).

For instance, if you define `(define ob (make-object '(3 2 7)))`, then `(ob 'member? 9)` is false, while `(ob 'subsetsum 9)` is true.

Hint: you can call this procedure in your code:

```
(define (subsets e)
  (let loop ((l e)
             (out '(())))
    (if (null? l)
        out
        (loop (cdr l)
              (append out
                      (map (lambda (x) (cons (car l) x)) out))))))
```

2 Haskell (11 points)

Define the function `infixes`, which takes a list g as input and returns the list of all infixes (i.e. non-empty contiguous sublists) of g.

For instance, `infixes "ciao"` is the list `["o","ao","iao","ciao","a","ia","cia","i","ci","c"]` (remember that a string is a list of characters in Haskell).

3 Prolog (11 points)

Consider binary trees represented as a hierarchic lists, where each node is a list [node, subtree1, subtree2]. Leaves are just symbols. In the *colored subtree problem*, we take as input a tree, and put into each internal node a number representing the number of different leaves present in its subtrees.

E.g. given this tree: `[R,[X,yellow,brown],[Y,blue,yellow]]` the solution is: $R = 3$, $X = Y = 2$.

Define the `col_tree` predicate, that solves the colored subtree problem.

Hint: the predicate `union(X,Y,Z)` holds if the list Z is the union of X and Y, seen as sets.

Solutions

Scheme

```
(define (make-object lst)
  (let ((sum-of-subsets (map (lambda (x) (foldl + 0 x))
                             (subsets lst))))
    (define my-member (lambda (x)
                        (list? (member x lst))))
    (define subsetsum (lambda (val)
                        (list? (member val sum-of-subsets))))

    (lambda (msg . args)
      (apply (case msg
               ((member?) my-member)
               ((subsetsum) subsetsum)
               (else (error "unknown method" msg)))
             args))))
```

Haskell

Idea: as hinted in another exam, the infixes are the suffixes of the prefixes.

```
suffixes lst = suf lst []
               where
                 suf [] res = res
                 suf (x:xs) res = suf xs ((x:xs) : res)

prefixes lst = pre lst []
               where
                 pre [] res = res
                 pre (x:xs) [] = pre xs [[x]]
                 pre (x:xs) res = pre xs $ ((head res) ++ [x]) : res

-- A less efficient but one-line version:
prefixes' l = map reverse $ suffixes $ reverse l

infixes lst = foldl (++) [] $
              map suffixes (prefixes lst)
```

Prolog

The main idea is to use the second argument to keep track of all the symbols used in the subtrees.

```
col_tree([1, X, X], [X]) :- atomic(X), !.

col_tree([2, X, Y], [X,Y]) :- atomic(X), atomic(Y), !.

col_tree([N, Tree1, Tree2], Colors) :-
        atomic(Tree1),
        col_tree(Tree2, Col2), !,
        union([Tree1], Col2, Colors),
        length(Colors, N).

col_tree([N, Tree1, Tree2], Colors) :-
        col_tree(Tree1, Col1),
        atomic(Tree2), !,
        union(Col1, [Tree2], Colors),
        length(Colors, N).

col_tree([N, Tree1, Tree2], Colors) :-
        col_tree(Tree1, Col1),
        col_tree(Tree2, Col2), !,
        union(Col1, Col2, Colors),
        length(Colors, N).
```