Principles of Programming Languages

2014.09.08

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

1.1 Multiple Apply (3 pts)

Define a procedure called `multiple-apply` which takes another procedure f, a natural number n and an item x, and applies f n times to x, i.e. it should return $f^n(x)$.

1.2 Position of Max (4 pts)

Define a procedure called `position-of-max`, that takes a list l and returns the position of l which contains the maximum value present in l. E.g. `(position-of-max '(2 3 1 -2))` is 1.
Note: remember that `max` in Scheme accepts a variable number of arguments, at least one. E.g. `(max 2 3 1 -2)` is 3.

1.3 Max of the Longest (6 pts)

Consider a definition of *norm*, where the norm of a number is the number itself, while the norm of a string is its length. Write a procedure called `max-of-the-longest`, that takes a list of lists, containing either strings or numbers, and returns the maximum norm of the elements in the longest of the lists.
E.g. `(max-of-the-longest '((99 0) (2 3 "hi, there!") (3 "hi there" 1 -1 -1)))` is 8.

2 Haskell

2.1 Part I (8 pts)

Translate every procedure of the Scheme part into Haskell, assuming that the list of lists contains either `String`s or `Int`s and defining suitable data structures, if needed.
Note: `max` in Haskell has type `Ord a => a -> a -> a`.

2.2 Part II (5 pts)

Declare all the types of the functions defined in Part I.

3 Prolog (6 pts)

Define `multiple-apply` in Prolog, using cut if possible.

Solutions

Scheme

```
(define (multiple-apply fun k L)
  (if (<= k 0)
      L
      (multiple-apply fun (- k 1) (fun L))))

(define (position-of-max L)
  (let ((max (car L))
        (pos 0)
        (p   0))
    (for-each (lambda (x)
                (when (> x max)
                    (set! max x)
                    (set! pos p))
                (set! p (+ 1 p)))
              L)
    pos))

(define (norm x)
  (cond
   ((number? x) x)
   ((string? x) (string-length x))
   (else (error "wrong type"))))

(define (max-of-the-longest L)
  (apply max
         (map norm
              (list-ref L
                        (position-of-max (map length L))))))
```

Haskell

```
multipleApply :: (Eq a, Num a) => (t -> t) -> a -> t -> t
multipleApply f 0 lst = lst
multipleApply f k lst = multipleApply f (k-1) (f lst)

positionOfMax :: (Num b, Ord a) => [a] -> b
positionOfMax lst = posHelper lst (head lst) 0 0
  where
    posHelper [] mx mp p = mp
    posHelper (v:vs) mx mp p = posHelper vs
                               (if v>mx then v else mx)
                               (if v>mx then p else mp)
                               (p+1)

data StrNum = S String | N Int deriving (Show, Eq)

norm :: StrNum -> Int
norm (S x) = length x
norm (N x) = x
```

```
maxOfTheLongest :: [[StrNum]] -> Int
maxOfTheLongest lst =
  let (x:xs) = (map norm (lst !! positionOfMax (map length lst)))
  in foldl max x xs
```

Prolog

```
multipleapply(_,0,L,L) :- !.
multipleapply(F,K,X,Y) :- K > 0, !, K1 is K-1,
                          multipleapply(F,K1,X,Y1), call(F,Y1,Y).
```