Principles of Programming Languages

2013.07.24

Notes

• Total available time: 2h.
• You may use any written material you need.
• You cannot use computers or phones during the exam.

1 Prolog (11 points)

1. Define the \texttt{prefix} predicate that holds iff its second argument is a prefix of the first argument.
   E.g. \texttt{prefix("Hello world", "Hello")} is true, while \texttt{prefix("Hello world", "wor")(false).}

2. Define an analogous predicate for suffixes.
   E.g. \texttt{suffix("Hello world", "world")} is true, while \texttt{suffix("Hello world", "Hello")} is not.

3. Define the \texttt{infix} predicate that holds iff its second argument is a substring of the first argument.
   (Hint: an infix is a prefix of a suffix.)

4. Define the \texttt{overlap} predicate that holds iff its two argument strings actually overlap, i.e. either one is
   an infix of the other, or one’s prefix is a suffix of the other.

2 Haskell (11 points)

Consider an immutable doubly linked list datatype (DList), where each node has two pointers, one to the
previous node (\texttt{prev}) and another to the next node (\texttt{next}), together with its local datum. There is a special
value \texttt{Nil}, denoting the empty DList. A well-formed DList has always the first node with \texttt{prev} set to \texttt{Nil},
and the last node with \texttt{next} set to \texttt{Nil}.

1. Define the \texttt{DList} datatype. DList must be an instance of the Eq class, and \texttt{==} must always terminate
   for every well-formed DList.

2. Define the \texttt{car} and \texttt{cdr} functions for DLists. The latter must return well-formed DLists, if not called
   on \texttt{Nil}. Errors must be managed in the Maybe monad.

3. Define the \texttt{cons} function for DLists.

3 Scheme/Ruby (10 points)

Define a mutable variant of DList either in Scheme or in Ruby. You are requested to define the \texttt{DList}
datatype: \texttt{Dcar}, \texttt{Dcdr}, and \texttt{Dcons}, i.e. \texttt{car}, \texttt{cdr}, \texttt{cons} variants for DLists; and \texttt{DList=\texttt{?}} that holds if both its
arguments are equal.
Solutions

Prolog

prefix([X|_], [X]).
prefix([X|Xs], [X|Z]) :- prefix(Xs, Z).

suffix(X,X) :- \+ X = [].
suffix([_|Xs], S) :- suffix(Xs, S).

infix(X,Y) :- suffix(X, SuffX), prefix(SuffX, Y).
overlaph(X,Y) :- suffix(X, SuffX), prefix(Y, SuffX).
overlap(X,Y) :- overlaph(X, Y); overlaph(Y, X); infix(X, Y); infix(Y, X).

Haskell

data DList a = Nil | Node (DList a) a (DList a)

instance Eq a => Eq (DList a) where
  Nil == Nil = True
  (Node p c n) == (Node p’ c’ n’) = c == c’ && n == n’
  _ == _ = False

car Nil = Nothing
car (Node prev head next) = Just head

cdr Nil = Nothing
cdr (Node prev head next) =
  let Node p c n = next
  in Just $ Node Nil c n

cons x Nil = Node Nil x Nil
-- cons exploits call by need: new's definition is recursive:
cons x (Node Nil cur next) = let new = Node Nil x (Node new cur next)
in new
Scheme

(struct DList (prev
    curr
    next) #:mutable)

(define Nil (DList #f #f #f)) ;; the Nil object
(define (Nil? x) (eq? x Nil)) ;; just for convenience

(define (Dcons item node)
    (if (Nil? node)
        (DList Nil item Nil)
        (let* ((newcar (DList Nil (DList-curr node) (DList-next node)))
            (newnode (DList Nil item newcar)))
            (set-DList-prev! newcar newnode)
            newnode)))

(define (Dcar node)
    (if (Nil? node)
        (error "Dcar of Nil")
        (DList-curr node)))

(define (Dcdr node)
    (if (Nil? node)
        (error "Dcdr of Nil")
        (let ((next (DList-next node)))
            (DList Nil (DList-curr next) (DList-next next)))))

(define (DList=? node1 node2)
    (cond
        ((and (Nil? node1) (Nil? node2)) #t)
        ((or (Nil? node1) (Nil? node2)) #f)
        (else
            (and (equal? (DList-curr node1)
                (DList-curr node2))
                (DList=? (DList-next node1)
                    (DList-next node2))))))