Laboratorio di Programmazione a.a. 2024-25

Esercitazione su temi d'esame

16 e 17 Dicembre 2024

1. Procedure in Scheme

Completa la procedura `match` che, date due stringhe di lettere *u* e *v*, restituisce la stringa delle corrispondenze *w* così definita: *w* ha la lunghezza della stringa più corta (fra *u* e *v*); se in una certa posizione *u* e *v* contengono la stessa lettera, allora anche *w* contiene quella lettera nella posizione corrispondente; se invece *u* e *v* contengono lettere diverse, *w* contiene il simbolo "asterisco" nella posizione corrispondente. Per esempio, il valore dell'espressione Scheme `(match "astrazione" "estremi")` è rappresentato dalla stringa delle corrispondenze `"*str**i"`.

```
(define match
  (lambda (u v)
    (if ( ..........  (string=? u "") (string=? v ""))
        ""
        (let ( (uh (string-ref u 0))  ..........
               (s  .......... )
             )
          (if ( ..........  uh vh)
              (string-append  ..........  s)
              (string-append "*" s)
              ))
    )))
```

2. Procedure in Scheme

Completa il programma `increment`, che calcola l'incremento di un numero naturale rappresentato come stringa di cifre in una base compresa fra *2* e *10*. Gli argomenti sono *num*, la stringa numerica, e *base*, di tipo intero; il valore restituito è una stringa numerica. Per esempio, il valore dell'espressione `(increment "1011" 2)` è `"1100"`, dove le stringhe rappresentano rispettivamente *11* e *12* in base *2*.

```
(define offset (char->integer #\0))

(define last-digit
  (lambda (base)  (integer->char (+ (- base 1) offset)) ))

(define next-digit
  (lambda (dgt)  ( ..........  (integer->char (+ (char->integer dgt) 1))) ))

(define increment
  (lambda (num base)  ; 2 <= base <= 10
    (let ((digits (string-length num)))
      (if (= digits 0)
          "1"
          (let ((dgt  .......... ))
            (if (char=? dgt (last-digit base))
                (string-append  ..........
                               "0")
                (string-append (substring num 0 (- digits 1))  .......... )
                ))
    ))))
```

3. Programmazione in Scheme

Date le stringhe `u`, `v`, la procedura `lcs` calcola una soluzione del problema della *sottosequenza comune più lunga*. Il risultato è rappresentato da una lista di terne, ciascuna delle quali contiene le posizioni in `u` e in `v` di un carattere comune che fa parte della sottosequenza più lunga, numerate a partire da 1, e la stringa costituita dal solo carattere comune. Esempi:

```
(lcs "pino" "pino")     →   ((1 1 "p") (2 2 "i") (3 3 "n") (4 4 "o"))
(lcs "pelo" "peso")     →   ((1 1 "p") (2 2 "e") (4 4 "o"))
(lcs "ala" "palato")    →   ((1 2 "a") (2 3 "l") (3 4 "a"))
(lcs "arto" "atrio")    →   ((1 1 "a") (3 2 "t") (4 5 "o"))
```

In particolare, nell'ultimo esempio `(3 2 "t")` contiene le posizioni di `'t'` rispettivamente in `"arto"` e `"atrio"`.

Completa il programma riportato nel riquadro introducendo opportune espressioni negli appositi spazi.

```
(define lcs       ; valore: lista di terne
  (lambda (u v)   ; u, v: stringhe

    (lcs-rec  ..........................................  u  ..........................................  v)
    ))

(define lcs-rec
  (lambda (i u j v)
    (cond ((or (string=? u "") (string=? v ""))

               ..........................................................................................  )

          ((char=?  ..........................................  ..........................................  )

           (cons  ..........................................................................................

                  (lcs-rec (+ i 1) (substring u 1) (+ j 1) (substring v 1)) ))

          (else
           (better  ..........................................................................................

                    ..........................................................................................  ))

          )))

(define better
  (lambda (x y)
    (if (< (length x) (length y))  y  x)
    ))
```

4. Definizione di procedure in Scheme

Definisci formalmente una procedura `cyclic-string` in Scheme che, dati come argomenti una stringa *pattern* e un numero naturale *length*, assuma come valore la stringa di lunghezza *length* risultante dalla ripetizione ciclica di *pattern*, eventualmente troncata a destra. Per esempio, nel caso dell'espressione `(cyclic-string "abcd" n)` il risultato della valutazione per $n = 0, 1, 2, 4, 5, 11$ deve essere, rispettivamente: `""`, `"a"`, `"ab"`, `"abcd"`, `"abcda"`, `"abcdabcdabc"`.

5. Definizione di procedure in Scheme

Definisci una procedura `av` in Scheme che, data una lista non vuota $(x_1\ x_2\ \ldots\ x_n)$ i cui n elementi x_i appartengono all'insieme $\{-1, 0, 1\}$, restituisca la lista $(y_1\ y_2\ \ldots\ y_{n-1})$ di $n-1$ elementi dello stesso insieme tale che $y_i = -1$ se $x_i + x_{i+1} < 0$, $y_i = 0$ se $x_i + x_{i+1} = 0$ e $y_i = 1$ se $x_i + x_{i+1} > 0$. Per esempio:

```
(av '(0 0 -1 -1 1 0 0 1 0))      →      (0 -1 -1 0 1 0 1 1)
```

6. Programmazione in Scheme

Definisci una procedura `shared` in Scheme che, date due liste *u*, *v* (strettamente) *ordinate* di numeri interi positivi, restituisca la lista ordinata degli elementi comuni a *u* e *v*. Per esempio:

```
(shared '(1 3 5 6 7 8 9 10) '(0 1 2 3 4 5 7 9))      →      (1 3 5 7 9)
```

7. Programmazione in Scheme

Una parola binaria, cioè una stringa composta esclusivamente dai simboli `0` e `1`, supera il *controllo di parità* se il numero di occorrenze di `1` è pari. Data una lista di parole binarie, la procedura `parity-check-failures` restituisce la lista delle posizioni delle parole che *non* superano il controllo di parità. Esempi:

```
(parity-check-failures '("0111" "1001" "0000" "1010"))  →  '(0)
(parity-check-failures '("0110" "1101" "0000" "1011"))  →  '(1 3)
(parity-check-failures '("0111" "1011" "0100" "1110"))  →  '(0 1 2 3)
(parity-check-failures '("0110" "1001" "0000" "1010"))  →  '()
```

Definisci un programma in Scheme che renda disponibile la procedura `parity-check-failures`.

8. Programmazione in Scheme

Scrivi un programma in Scheme basato sulla procedura `sorted-char-list` che, data una stringa, restituisce la lista dei caratteri che vi compaiono, ordinata in ordine alfabetico e senza ripetizioni. Esempi:

```
(sorted-char-list "")        →   ()
(sorted-char-list "abc")     →   (#\a #\b #\c)
(sorted-char-list "cba")     →   (#\a #\b #\c)
(sorted-char-list "list of chars that occur in this text")
             →   (#\space #\a #\c #\e #\f #\h #\i #\l #\n #\o #\r #\s #\t #\u #\x)
```

(Per il confronto alfabetico di caratteri puoi utilizzare le procedure predefinite `char=?`, `char<?`, `char<=?`, ecc.)

9. Programmazione in Scheme

Data una lista di stringhe *u*, la procedura `clean-up` restituisce la lista di tutti gli elementi di *u*, ma in cui ciascun elemento occorre una volta sola (senza eventuali ripetizioni). Per esempio:

```
(clean-up   '("rosa" "garofano" "pervinca" "ciclamino"
              "genziana" "pervinca" "fiordaliso" "rosa"))
          →      ("garofano" "ciclamino" "genziana" "pervinca" "fiordaliso" "rosa")
```

Scrivi un programma in Scheme per realizzare la procedura `clean-up`.

10. Programmazione in Scheme

Definisci in Scheme una procedura `longest-contiguous-repeat` che, data una lista (non vuota) `s` di stringhe, restituisce la stringa con il maggior numero di occorrenze contigue, cioè una dopo l'altra senza interruzioni, in `s`. Nei casi in cui la soluzione non sia unica, è indifferente quale fra le stringhe con questa proprietà venga restituita. Esempio:

```
(longest-contiguous-repeat '("a" "b" "b" "a" "a" "b" "c" "c" "c" "c" "b" "b"))  →  "c"
```