Corso di Programmazione

I Prova di accertamento del 29 Gennaio 2021 / A

cognome e nome

Selezione degli esercizi proposti attraverso la piattaforma moodle.

1. Programmi in Scheme

Dati due numeri x e y, la procedura `pair` "costruisce" la coppia dei relativi valore medio e semidifferenza (non negativa), rispettivamente $(x+y)/2$ e $|x-y|/2$ (dove per *coppia* si intende una lista di due elementi).

Inoltre, date due liste numeriche della stessa lunghezza, la procedura `pair-list` applica `pair` e restituisce la lista delle coppie valore medio e semidifferenza relative agli elementi con la stessa posizione nelle due liste.

Esempi:

```
(pair 4.8 1.2)  →  (3.0 1.8)                    (pair 3.5 5.7)  →  (4.6 1.1)
(pair 8.1 0.3)  →  (4.2 3.9)                    (pair 5.0 3.0)  →  (4.0 1.0)
(pair-list '(4.8 3.5 8.1) '(1.2 5.7 0.3))   →   ((3.0 1.8) (4.6 1.1) (4.2 3.9))
(pair-list '(5.3 4.3 3.3) '(2.5 4.5 6.5))   →   ((3.9 1.4) (4.4 0.1) (4.9 1.6))
```

1.1. Definisci in Scheme la procedura `pair`:

```
(define pair
    (lambda (x y)    ;x y INTERI
        (list
            (/ (+ x y) 2)   ;(x+y)/2
            (/ (abs (- x y)) 2)   ;|x-y|/2
        )
    )
)
```

1.2. Utilizzando `pair`, definisci in Scheme la procedura `pair-list`:

```
(define pair-list
  (lambda (list1 list2)
      (if (or (null? list1) (null? list2) )
          ;UNA DELLE 2 è VUOTA
          null ; oppure '()
          ;ELSE
          (append
              (list (pair (car list1) (car list2))  ;crea una lista contenente la lista restituita da pair
              (pair-list (cdr list1) (cdr list1))   ;richiama pair-list togliendo il primo elemento delle liste
          )
      )
  )
)
```

2. Ricorsione ad albero

La procedura `lcs-align`, impostata qui sotto, risolve una variante del problema della *sottosequenza comune più lunga* (LCS), identificando quali caratteri devono essere rimossi dalle due stringhe *u* e *v* passate come argomento affinché le rispettive parti restanti, che costituiscono la LCS, possano essere "allineate". Più specificamente, `lcs-align` restituisce una coppia di liste, la prima contenente i caratteri da rimuovere da *u*, la seconda quelli da rimuovere da *v* (riportati nell'ordine in cui compaiono in *u* e *v*). Per esempio:

```
(lcs-align "ac" "bc")          →   ((#\a) (#\b))
(lcs-align "atrio" "arto")     →   ((#\t #\i) (#\t))
(lcs-align "epico" "esilio")   →   ((#\p #\c) (#\s #\l #\i))
```

2.1. Completa la definizione della procedura `lcs-align` inserendo opportune espressioni negli spazi indicati [`string->list` restituisce la lista dei caratteri di una stringa; `(cadr x)` è un'abbreviazione di `(car (cdr x))`]:

```
(define lcs-align  ; val: coppia di liste di caratteri
  (lambda (u v)    ; u, v: stringhe
    (let ((m (string-length u)) (n (string-length v))
          )
      (cond ((or (= m 0) (= n 0))
             (list (string->list u) (string->list v))
             )
            ((char=? (string-ref u 0) (string-ref v 0))

              .....
             )
            (else
             (let ((du (lcs-align (substring u 1) v))

                   (dv  ..... )
                   )
               (if (> (+ (length (car du)) (length (cadr du)))
                      (+ (length (car dv)) (length (cadr dv)))
                      )
                   (list (car dv) (cons (string-ref v 0) (cadr dv)))

                    .....
                   )))
            ))))
```

2.2. Con riferimento alla valutazione di `(lcs-align "ac" "bc")`, quali sono i valori associati alle variabili `du` e `dv` nella prima invocazione della procedura ricorsiva `lcs-align`?

`du` = ...

`dv` = ...

3. Verifica formale della correttezza

In relazione alla procedura `f` così definita:

```
(define f          ; val: intero
  (lambda (x y)    ; x, y: interi positivi,  tali che x ≤ y
    (if (= x y)
        (+ x y -1)
        (let ((z (quotient (+ x y) 2)))
          (+ (f x z) (f (+ z 1) y))
          ))
    ))
```

si può dimostrare per induzione sul valore della differenza $n–m$ che per ogni coppia di interi m, n tali che $0 < m \leq n$:

`(f m n)` $\rightarrow$ $n^2 – (m–1)^2$

3.1. In relazione alla dimostrazione impostata in questi termini:

- Formalizza la proprietà che esprime il caso / i casi base:

.....

.....

Assumendo come ipotesi induttiva che per un certo valore k intero non negativo si abbia che:

ogni coppia di interi positivi m, n tali che $n–m \leq k$: `(f m n)` $\rightarrow$ $n^2 – (m–1)^2$

- Formalizza la proprietà da dimostrare come passo induttivo:

.....

.....

3.2. Completa la definizione della procedura `sq` per calcolare il quadrato di un intero positivo riportando gli argomenti appropriati di `f`:

```
(define sq         ; val: intero
  (lambda (n)      ; n: intero positivo

    (f  .....  .....  )
    ))
```

4. Procedure con argomenti e valori procedurali

Il programma per realizzare la procedura `parity-check?`, impostato qui sotto, simula un controllo di parità relativo a una sequenza di parole binarie, tutte della stessa lunghezza, contenute in una lista. Immaginando di incolonnare le parole della sequenza, si tratta di verificare che il numero di 1 presenti in ciascuna colonna sia pari; se è effettivamente così per ogni colonna, allora il controllo ha successo e `parity-check?` restituisce il valore booleano *true*, altrimenti fallisce e `parity-check?` restituisce *false*.

Per esempio:

```
(parity-check? '("0110" "1000" "1011" "0101"))   →   true
(parity-check? '("0110" "1100" "1011" "0101"))   →   false
```

4.1. Completa il programma per realizzare `parity-check?` inserendo opportune espressioni negli spazi indicati:

```
(define parity-check?   ; val: booleano
  (lambda (words)       ; words: lista non vuota di stringhe di 0/1 della stessa lunghezza
    (rec-check? words 0 (string-length (car words)))
    ))

(define rec-check?
  (lambda (words k n)
    (if (< k n)
        (let ((kths (map (bit k) words))) ; kths: lista dei valori dei bit in posizione k nelle parole di words
          (if (even? (count-ones kths))
              (rec-check? words (+ k 1) n)
                    ..........
              ))
        true
        )))

(define bit
  ( ..........
    ..........
    ))

(define count-ones
  (lambda (cs)
    (if (null? cs)
          ..........
          (+ (car cs)  .......... )
          )))
```

4.2. Qual è il valore associato alla variabile *kths* nella prima invocazione di `rec-check?` nel corso della valutazione di `(parity-check? '("0110" "1100" "1011" "0101"))` ?

Valore associato a *kths*: ....

Corso di Programmazione

I Prova di accertamento del 29 Gennaio 2021 / B

cognome e nome

Selezione degli esercizi proposti attraverso la piattaforma moodle.

1. Programmi in Scheme

Dati due caratteri *x* e *y*, la procedura `pair` "costruisce" la coppia ordinata alfabeticamente dei due valori ricevuti come argomento (dove per *coppia* si intende una lista di due elementi).

Inoltre, date due liste di caratteri della stessa lunghezza, la procedura `pair-list` applica `pair` e restituisce la lista delle coppie ordinate alfabeticamente degli elementi che hanno la stessa posizione nelle due liste argomento.

Esempi:

```
(pair #\q #\n)  →  (#\n #\q)                    (pair #\c #\x)  →  (#\c #\x)
(pair #\t #\t)  →  (#\t #\t)                    (pair #\b #\a)  →  (#\a #\b)
(pair-list '(#\q #\c #\t) '(#\n #\x #\t))  →  ((#\n #\q) (#\c #\x) (#\t #\t))
(pair-list '(#\1 #\3 #\5) '(#\4 #\2 #\0))  →  ((#\1 #\4) (#\2 #\3) (#\0 #\5))
```

1.1. Definisci in Scheme la procedura `pair`:

1.2. Utilizzando `pair`, definisci in Scheme la procedura `pair-list`:

2. Ricorsione ad albero

La procedura `lcs-align`, impostata qui sotto, risolve una variante del problema della *sottosequenza comune più lunga* (LCS), identificando quali caratteri di *v* sono "allineati" con la LCS degli argomenti *u*, *v* e quali invece non lo sono, rimpiazzando questi ultimi con degli underscore "_". Più specificamente, `lcs-align` restituisce una coppia composta dall'intero che rappresenta la lunghezza della LCS di *u*, *v* e dalla rielaborazione di *v* in cui i caratteri che non fanno parte della LCS sono sostituiti da underscore (simbolo che si assume non faccia parte di *u* e *v*). Per esempio:

```
(lcs-align "ac" "bc")          →   (1 "_c")
(lcs-align "atrio" "arto")     →   (3 "ar_o")
(lcs-align "epico" "esilio")   →   (3 "e_i__o")
```

2.1. Completa la definizione della procedura `lcs-align` inserendo opportune espressioni negli spazi indicati [`(cadr x)` è un'abbreviazione di `(car (cdr x))`]:

```
(define lcs-align  ; val: coppia intero/stringa
  (lambda (u v)    ; u, v: stringhe
    (let ((m (string-length u)) (n (string-length v))
          )
      (cond ((= n 0) (list 0 ""))
            ((= m 0)
             (let ((w (lcs-align u (substring v 1))))
               (list 0 (string-append "_" (cadr w)))
               ))
            ((char=? (string-ref u 0) (string-ref v 0))

             (let ((dx ..... )
                   )
               (list (+ (car dx) 1) (string-append (substring v 0 1) (cadr dx))
                     )))
            (else
             (let ((du (lcs-align (substring u 1) v))

                   (dv ..... )
                   )
               (if (< (car du) (car dv))
                   (list (car dv) (string-append "_" (cadr dv)))

                   .....

                   )))
            ))))
```

2.2. Con riferimento alla valutazione di `(lcs-align "ac" "bc")`, quali sono i valori associati alle variabili `du` e `dv` nella prima invocazione della procedura ricorsiva `lcs-align`?

`du` = ...

`dv` = ...

3. Verifica formale della correttezza

In relazione alla procedura `f` così definita:

```
(define f          ; val: intero
  (lambda (x y)    ; x, y: interi positivi
    (if (= x 1)
        (+ y y -1)
        (let ((z (quotient x 2)))
          (+ (f z y) (f (- x z) (+ y z)))
          ))
    ))
```

si può dimostrare per induzione sul valore di n che per ogni coppia di interi m, $k > 0$ con $m \leq n$:

`(f m k)` $\rightarrow$ $(m+k-1)^2 - (k-1)^2$

3.1. In relazione alla dimostrazione impostata in questi termini:

- Formalizza la proprietà che esprime il caso / i casi base:

.....

.....

Assumendo come ipotesi induttiva che per un certo valore n intero positivo si abbia che:

ogni coppia di interi positivi $m \leq n$ e k : `(f m k)` $\rightarrow$ $(m+k-1)^2 - (k-1)^2$

- Formalizza la proprietà da dimostrare come passo induttivo:

.....

.....

3.2. Completa la definizione della procedura `sq` per calcolare il quadrato di un intero positivo riportando gli argomenti appropriati di `f`:

```
(define sq         ; val: intero
  (lambda (n)      ; n: intero positivo

    (f ..... ..... )
    ))
```

4. Procedure con argomenti e valori procedurali

Il programma per realizzare la procedura `parity-check?`, impostato qui sotto, simula un controllo di parità relativo a una sequenza di parole binarie, tutte della stessa lunghezza, contenute in una lista. Immaginando di incolonnare le parole della sequenza, si tratta di verificare che il numero di 1 presenti in ciascuna colonna sia pari; se è effettivamente così per ogni colonna, allora il controllo ha successo e `parity-check?` restituisce il valore booleano *true*, altrimenti fallisce e `parity-check?` restituisce *false*.

Per esempio:

```
(parity-check? '("0110" "1000" "1011" "0101"))   →   true
(parity-check? '("0110" "1100" "1011" "0101"))   →   false
```

4.1. Completa il programma per realizzare `parity-check?` inserendo opportune espressioni negli spazi indicati:

```
(define parity-check?   ; val: booleano
  (lambda (words)       ; words: lista non vuota di stringhe di 0/1 della stessa lunghezza
    (rec-check? words (- (string-length (car words)) 1))
    ))

(define rec-check?
  (lambda (words k)
    (if (< k 0)
        true
        (let ((kths (map (bit k) words))) ; kths: lista dei valori dei bit in posizione k nelle parole di words
          (if (check-ones? kths)
              (rec-check? words (- k 1))

                  .....
              ))
        )))

(define bit

  ( .....

    .....
    ))

(define check-ones?
  (lambda (cs)
    (cond ((null? cs)

             ..... )
          ((= (car cs) 1)
           (not (check-ones? (cdr cs))))
          (else

             ..... )
          )))
```

4.2. Qual è il valore associato alla variabile *kths* nella prima invocazione di `rec-check?` nel corso della valutazione di `(parity-check? '("0110" "1100" "1011" "0101"))` ?

Valore associato a *kths*: .....