Corso di Programmazione

I Accertamento del 27 Gennaio 2015

cognome e nome

Risolvi i seguenti esercizi, riporta le soluzioni in modo chiaro negli appositi spazi e giustifica sinteticamente le risposte. Dovrai poi consegnare queste schede con le soluzioni, avendo cura di scrivere il tuo nome nell'intestazione e su ciascun eventuale foglio aggiuntivo.

1. Programmi in Scheme

Facendo riferimento al programma realizzato dalle procedure `f`, `g`, `h`:

```
(define f
  (lambda (x)   ; x > 0 intero
    (g x null)
    ))

(define g
  (lambda (x s)
    (if (> x 1)
        (g (quotient x 2) (cons x s))
        (h s 1)
        )))
```

```
(define h
  (lambda (s y)
    (if (null? s)
        y
        (let ((z (* 2 y)) (t (cdr s)))
          (if (even? (car s))
              (h t (- z 1))
              (h t (+ z 1))))
        )))
```

determina il risultato della valutazione di ciascuna delle seguenti espressioni:

`(f 1)` → `1`

`(f 2)` → `1`

`(f 3)` → `3`

`(f 5)` → `3`

`(f 7)` → `7`

`(f 8)` → `1`

`(f 12)` → `9`

`(f 16)` → `1`

2. Ricorsione e argomenti procedurali

Considera il problema di coprire un cordolo di lunghezza *n* e altezza 1 con piastrelle quadrate 1x1 di colore *rosso* e *blu*, rispettando il vincolo aggiuntivo che le piastrelle rosse non possono essere adiacenti. Dato un intero non negativo *n*, la procedura `rb-tilings` restituisce la lista di tutte le soluzioni possibili, codificate da una stringa di `r` e `b`, dove `r` rappresenta una piastrella rossa e `b` una piastrella blu. Per esempio, le 8 soluzioni diverse per un cordolo di lunghezza 4 risultano dalla valutazione dell'espressione:

```
(rb-tilings 4)   →   ("rbrb" "rbbr" "rbbb" "brbr" "brbb" "bbrb" "bbbr" "bbbb")
```

Completa la definizione della procedura `rb-tilings`.

```
(define rb-tilings  ; valore: lista di stringhe di r/b
  (lambda (n)       ; n: intero non negativo

    (cond ( (= n 0)                                                    (list ""))

          ((= n 1)  (list "r" "b")                                                )
          (else

           (append (map  (lambda (x) (string-append "rb" x))   (rb-tilings (- n 2)))

                   (map  (lambda (x) (string-append "b" x))  (rb-tilings (- n 1))  )
                   ))
          )))
```

3. Definizione di procedure in Scheme

La procedura `factors->number` restituisce un intero positivo data la sua fattorizzazione in fattori primi, rappresentata da due liste: la lista dei fattori primi e la lista dei corrispondenti gradi (esponenti dei fattori primi). Esempi:

```
(factors->number '(5) '(1))  →  5        (factors->number '(2 3) '(2 2))  →  36
(factors->number '(2) '(3))  →  8        (factors->number '(2 5) '(2 1))  →  20
(factors->number '(2 3 5) '(2 2 1))   →  180
(factors->number '(2 5 11) '(1 2 1))  →  550
```

Definisci in Scheme la procedura `factors->number`.

```
(define factors->number  ; valore: intero positivo
  (lambda (u v)          ; u, v: liste di interi positivi
    (cond ((null? u)
           1)
          ((= (car v) 1)
           (* (car u) (factors->number (cdr u) (cdr v))))
          (else
           (* (car u) (factors->number u (cons (- (car v) 1) (cdr v)))))
          )))
```

4. Verifica formale della correttezza

In relazione alla procedura `h` definita nell'esercizio **1** è possibile verificare che

$$(\texttt{h}\ '(2^k\ 2^{k-1}\ 2^{k-2}\ \ldots\ 2^2\ 2^1)\ n) \quad \rightarrow \quad (n-1)\cdot 2^k + 1$$

per qualsiasi coppia di interi positivi k, n. Dimostra per induzione questa proprietà; in particolare:

- Formalizza con precisione la proprietà generale che si vuole dimostrare:

  $\forall\, k, n \in \mathbb{N}^+ \;.\; (\texttt{h}\ '(2^k\ 2^{k-1}\ 2^{k-2}\ \ldots\ 2^2\ 2^1)\ n) \;\rightarrow\; (n-1)\cdot 2^k + 1$

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

  $\forall\, n \in \mathbb{N}^+ \;.\; (\texttt{h}\ '(2^1)\ n) \;\rightarrow\; (n-1)\cdot 2^1 + 1$

- Formalizza l'ipotesi induttiva: preso $k \in \mathbb{N}^+$

  $\forall\, n \in \mathbb{N}^+ \;.\; (\texttt{h}\ '(2^k\ 2^{k-1}\ 2^{k-2}\ \ldots\ 2^2\ 2^1)\ n) \;\rightarrow\; (n-1)\cdot 2^k + 1$

- Formalizza la proprietà da dimostrare come passo induttivo: per k scelto sopra

  $\forall\, n \in \mathbb{N}^+ \;.\; (\texttt{h}\ '(2^{k+1}\ 2^k\ 2^{k-1}\ \ldots\ 2^2\ 2^1)\ n) \;\rightarrow\; (n-1)\cdot 2^{k+1} + 1$

- Dimostra il caso / i casi base: [$\forall\, n \in \mathbb{N}^+$]

  $(\texttt{h}\ '(2)\ n) \;\rightarrow\; (\texttt{if (even? 2) (h '() (- 2n 1)) (h '() (+ 2n 1))})$

  $\rightarrow\; (\texttt{h}\ '()\ 2n-1)$

  $\rightarrow\; 2n-1 \;=\; 2(n-1) + 1$

- Dimostra il passo induttivo: [$\forall\, n \in \mathbb{N}^+$]

  $(\texttt{h}\ '(2^{k+1}\ 2^k\ \ldots\ 2^1)\ n)$

  $\rightarrow\; (\texttt{if (even?}\ 2^{k+1}\texttt{) (h '(}2^k\ \ldots\ 2^1\texttt{) (- 2n 1)) (h '(}2^k\ \ldots\ 2^1\texttt{) (+ 2n 1))})$

  $\rightarrow\; (\texttt{h}\ '(2^k\ 2^{k-1}\ \ldots\ 2^1)\ \texttt{(- 2n 1)})$

  $\rightarrow\; (\texttt{h}\ '(2^k\ 2^{k-1}\ \ldots\ 2^1)\ 2n-1)$

  $\rightarrow\; (2n-1-1)\cdot 2^k + 1$ per l'ipotesi induttiva

  $= 2(n-1)\cdot 2^k + 1 \;=\; (n-1)\cdot 2^{k+1} + 1$

5. Realizzazione di programmi in Scheme

Assumi che un testo sia rappresentato da una lista di stringhe dove ciascuna stringa corrisponde a una riga del testo. Considera quindi una versione del problema *LCS* per coppie di testi: dati due testi, si vuole conoscere un "sottotesto" comune con il maggior numero possibile di righe in corrispondenza (cioè righe uguali e che compaiono nello stesso ordine nei due testi dati). Eventualmente, a parità di numero di righe si sceglierà una soluzione complessivamente composta da più caratteri. Scrivi un programma che risolva il problema proposto nei termini precisati sopra.

```
(define lcs      ; valore: lista di stringhe
  (lambda (u v)  ; u, v: liste di stringhe
    (cond ((or (null? u) (null? v))
           null)
          ((string=? (car u) (car v))
           (cons (car u) (lcs (cdr u) (cdr v))))
          (else
           (longer (lcs (cdr u) v) (lcs u (cdr v))))
          )))


(define longer   ; valore: stringa
  (lambda (u v)  ; u, v: stringhe
    (cond ((< (length u) (length v))
           v)
          ((> (length u) (length v))
           u)
          ((< (text-length u) (text-length v))
           v)
          (else
           u)
          )))


(define text-length  ; valore: intero
  (lambda (u)        ; u: lista di stringhe
    (if (null? u)
        0
        (+ (string-length (car u)) (text-length (cdr u)))
        )))
```