Corso di Programmazione

Esame del 28 Luglio 2015

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Ricorsione di coda

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

```
(define f
  (lambda (x y)
    (g (cons (list x y) null) 0)
    ))
```

```
(define g
  (lambda (s k)
    (if (null? s)
        k
        (let ((x (caar s)) (y (cadar s))
              (t (cdr s)))
          (if (or (= x 0) (= y 0))
              (g t (+ k 1))
              (g (cons (list (- x 1) y)
                       (cons (list x (- y 1)) t))
                 k)))
        )))
```

determina il risultato della valutazione di ciascuna delle seguenti espressioni:

`(f 0 3)` → .....

`(f 1 1)` → .....

`(f 3 1)` → .....

`(f 2 2)` → .....

`(f 2 3)` → .....

`(f 3 2)` → .....

`(f 3 3)` → .....

`(f 4 4)` → .....

2. Ricorsione e argomenti procedurali

Considera il problema di coprire un cordolo di lunghezza *n* e altezza 1 con piastrelle rettangolari di dimensione 2x1 e quadrate 1x1. Dato un intero non negativo *n*, la procedura `two-tilings` restituisce la lista di tutte le soluzioni possibili, codificate da una stringa di simboli `<>` e `o`, dove `<>` rappresenta una piastrella rettangolare e `o` una piastrella quadrata. Per esempio, le 8 soluzioni diverse per un cordolo di lunghezza 5 risultano dalla valutazione dell'espressione:

```
(two-tilings 5)  →  ("ooooo" "ooo<>" "oo<>o" "o<>oo" "o<><>" "<>ooo" "<>o<>" "<><>o")
```

Completa la definizione della procedura `two-tilings`.

```
(define two-tilings  ; valore: lista di stringhe di o/<>
  (lambda (n)        ; n: intero non negativo
    (cond ((= n 0)  ..... )
          ( ..... (list "o"))
          (else
           (append
            (map  ..... (two-tilings (- n 1)))
            (map  ..... )
            ))
          )))
```

3. Programmazione Dinamica

Si vuole applicare la tecnica *bottom-up* di *programmazione dinamica* al programma dell'esercizio **2**. Assumi di avere a disposizione la classe `StringList`, già utilizzata in laboratorio, che rende disponibili strumenti analoghi alle primitive Scheme per operare con liste di stringhe, specificamente una costante `NULL` per la lista vuota e i metodi statici `listNull`, `listCar`, `listCdr` e `listCons`, corrispondenti a `null?`, `car`, `cdr` e `cons`. Assumi inoltre che siano già state realizzate le procedure `append` e `prefix`: la prima restituisce l'*append* delle due liste passate come argomento; la seconda restituisce una lista in cui a tutte le stringhe della lista passata come secondo argomento viene premesso il prefisso passato come primo argomento. Definisci un metodo statico *non ricorsivo* `twoTilings` che realizza la funzione di `two-tilings` in modo più efficiente tramite programmazione dinamica.

```
public static StringList append( StringList s1, StringList s2 ) { ... }

public static StringList prefix( String p, StringList s ) { ... }
```

4. Verifica formale della correttezza

Dato un intero $n \geq 0$, il seguente metodo statico calcola la quarta potenza di n utilizzando solo somme e confronti. Nel programma sono riportate precondizione, postcondizione e invariante. Dimostra formalmente la correttezza parziale del programma.

```
public static int pow4( int n ) {            // Pre: n ≥ 0
  int  u = 0,  v = 0;
  int  x = 0,  y = 0;
  while ( (x < v) || (u < n) ) {             // Inv: 0 ≤ u ≤ n, 0 ≤ x ≤ v = u², y = x²
    if ( x < v ) {
      y = y + x;  x = x + 1;  y = y + x;
    } else {
      v = v + u;  u = u + 1;  v = v + u;
  }}
  return y;                                  // Post: y = n⁴
}
```

5. Oggetti in Java

Il modello della scacchiera realizzato dalla classe `Board` per affrontare il rompicapo delle *n* regine deve essere integrato introducendo un metodo `where` e una costante `NONE` (vedi sotto a sinistra): dato un indice di riga *i*, compreso fra 1 e la dimensione della scacchiera, `where(i)` restituirà l'indice di colonna in cui è collocata una regina nella riga *i* oppure `NONE` se la riga *i* è libera. Inoltre, `addQueen(i,j)` non deve modificare lo stato della scacchiera se c'è già una regina nella riga *i*, garantendo così che non si possa collocare più di una regina per riga. Per esempio, il metodo statico `listOfCompletions`, la cui definizione è riportata sotto a destra, stamperà tutte le soluzioni del rompicapo, se ve ne sono, compatibili con un'arbitraria disposizione iniziale di regine in righe diverse di una scacchiera.

```
Board( int n )  // costruttore

void addQueen( int i, int j )
void removeQueen( int i, int j )

int size()
int queensOn()
boolean underAttack( int i, int j )
String arrangement()

int where( int i )

static final int NONE
```

```
public static void listOfCompletions( Board board ) {
  int n = board.size();
  int q = board.queensOn();
  if ( q == n ) {
    System.out.println( board.arrangement() );
  } else {
    int i = 1;
    while ( board.where(i) != Board.NONE ) {
      i = i + 1;      // ricerca di una riga libera
    }
    for ( int j=1; j<=n; j=j+1 ) {
      if ( ! board.underAttack(i,j) ) {
        board.addQueen( i, j );
        listOfCompletions( board );
        board.removeQueen( i, j );
  }}}
}
```

In base a quanto specificato sopra, proponi opportune integrazioni della classe `Board`. (Riporta solo le modifiche che si rendono necessarie rispetto alla versione discussa a lezione.)