Corso di Programmazione

Esame del 9 Settembre 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 (list x) (list y) 0)
    ))
```

```
(define g
  (lambda (u v k)
    (if (null? u)
        k
        (let ((x (car u)) (y (car v))
              (s (cdr u)) (t (cdr v)))
          (if (or (= x 0) (= y 0))
              (g s t (+ k 1))
              (g (cons (- x 1) (cons x s))
                 (cons y (cons (- y 1) t)) k)
              ))
        )))
```

determina il risultato della valutazione di ciascuna delle seguenti espressioni:

`(f 5 0)` → .....

`(f 1 2)` → .....

`(f 1 3)` → .....

`(f 3 1)` → .....

`(f 2 2)` → .....

`(f 2 4)` → .....

`(f 4 2)` → .....

`(f 2 5)` → .....

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 3x1. Dato un intero $n \geq 2$, la procedura `two-tilings` restituisce la lista di tutte le soluzioni possibili, codificate da una stringa di simboli `<>` e `<O>`, dove `<>` rappresenta una piastrella di lunghezza 2 e `<O>` una di lunghezza 3. Per esempio, la valutazione dell'espressione `(two-tilings 10)` restituisce la lista di soluzioni:

```
("<><><><><>" "<><><O><O>" "<><O><><O>" "<><O><O><>" "<O><><><O>" "<O><><O><>" "<O><O><><>")
```

Completa la definizione della procedura `two-tilings`.

```
(define two-tilings  ; valore: lista di stringhe di <>/<O>
  (lambda (n)        ; n ≥ 2 intero

    (cond ((= n 2)  ..... )

          ( ..... (list "<O>"))

          ( ..... )
          (else
           (append

            (map  ..... (two-tilings (- n 2)))

            (map  ..... )
            ))
          )))
```

3. Memoization

Si vuole applicare la tecnica *top-down* di *memoization* 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 memoization.

```
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 riportati 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 ( (u < n) || (x < v) ) {                 // Inv: 0 ≤ u ≤ n, 0 ≤ x ≤ v = u², y = x²
    if ( (u < n) && (Math.random() < 0.5) ) {
      v = v + u;  u = u + 1;  v = v + u;
    }
    if ( (x < v) && (Math.random() < 0.5) ) {
      y = y + x;  x = x + 1;  y = y + x;
  }}
  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 due nuovi metodi: `isFreeRow(int)`, che dato un indice di riga *i* compreso fra 1 e la dimensione *n* della scacchiera, restituirà *true* se e solo se nella riga *i* non c'è alcuna regina; `addQueen(String)`, che svolge la stessa funzione di `addQueen(int,int)` ma ricevendo come argomento la codifica della posizione tramite una stringa di due caratteri (overloading), una lettera per la colonna e una cifra per la riga secondo le convenzioni consuete. Inoltre, `addQueen` e `removeQueen` non devono modificare lo stato della scacchiera se l'operazione è inconsistente perché due regine verrebbero a trovarsi sulla stessa casella oppure perché nella posizione data non c'è una regina da rimuovere. Per esempio, il metodo statico `listOfCompletions`, definito sotto a destra, stamperà tutte le soluzioni del rompicapo, se ve ne sono, compatibili con una disposizione iniziale di regine che non si minacciano reciprocamente.
In base a quanto specificato sopra, proponi opportune integrazioni della classe `Board` e riporta solo le modifiche che si rendono necessarie rispetto alla versione discussa a lezione.

```
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()

boolean isFreeRow( int i )
void addQueen( String pos )

// esempio:  b.addQueen( "b6" )
```

```
public static void listOfCompletions( Board b ) {
  int n = b.size();  int q = b.queensOn();
  if ( q == n ) {
    System.out.println( b.arrangement() );
  } else {
    int i = 1;
    while ( !b.isFreeRow(i) ) {
      i = i + 1;      // ricerca di una riga libera
    }
    for ( int j=1; j<=n; j=j+1 ) {
      if ( ! b.underAttack(i,j) ) {
        b.addQueen( i, j );
        listOfCompletions( b );
        b.removeQueen( i, j );
  }}}
}
```