Corso di Programmazione

Esame del 17 Settembre 2008

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Procedure in Scheme

Considera la seguente procedura in Scheme:

```
(define f
  (lambda (u)
    (cond ((or (null? u) (null? (cdr u))) u)
          ((equal? (car u) (cadr u)) (f (cdr u)))
          (else (cons (car u) (f (cdr u))))
          )))
```

Completa le seguenti valutazioni, riportando il risultato nell'apposito spazio:

`(f '())` → _____

`(f '(a a))` → _____

`(f '(a b c))` → _____

`(f '(a b b c c c d d))` → _____

`(f '(a b b a c c a a))` → _____

2. Programmazione in Scheme

Definisci in Scheme una procedura `funlist-comp` che, data una lista *funs* di funzioni dagli interi agli interi, restituisca la composizione delle funzioni di *funs*, considerate nell'ordine e dove la prima funzione della lista è quella applicata più esternamente. Più precisamente:

`( (funlist-comp '(`$f_1\ f_2$ `...` f_n`))` x `)` → $f_1(f_2(\ldots f_n(x)\ldots))$

(Se *funs* è la lista vuota viene restituita la funzione identica; se contiene un solo elemento *f*, viene restituita *f*.)

3. Verifica formale della correttezza

Dimostra formalmente correttezza e terminazione del seguente programma iterativo in *Java* per calcolare il prodotto di due numeri naturali, utilizzando le asserzioni, gli invarianti e la funzione di terminazione indicati nei commenti.

```
public static int product( int a, int b ) {
  /**  require  a >= 0;  b >= 0;  **/
  int x = a,  y = b,  z = 0,  r = y % 3;
  while ( y > 0 )
    /**  invariant  x*y + z == a*b;  r == y % 3;  y >= 0;  **/
    /**  variant  3*y + 2*r  **/  {
    if ( r == 0 ) {
      x = 3 * x;  y = y / 3;  r = y % 3;
    } else if ( r == 2 ) {
      y = y + 1;  z = z - x;  r = 0;
    } else { // r == 1
      y = y - 1;  z = z + x;  r = 0;
  }}
  return z;
  /**  ensure  Result == a*b;  **/
}
```

4. Memoization

Trasforma la seguente procedura, definita in *Scheme*, e realizza un programma equivalente applicando opportunamente la tecnica di *memoization*. A tua discrezione, puoi formalizzare la soluzione proposta in *Scheme* oppure in *Java*.

```
(define f
  (lambda (x y)  ; x, y naturali
    (cond ((or (= y 0) (> y x))  0)
          ((or (= y 1) (= y x))  1)
          (else
           (+ (f (- x 2) (- y 2))
              (* (- (* 2 y) 1) (f (- x 2) (- y 1)))
              (* (* y y) (f (- x 2) y))
              ))
          )))
```

5. Programmazione orientata agli oggetti in Java

Per consentire una maggiore libertà nella disposizione delle regine sulla scacchiera e nell'ordine secondo cui vengono introdotte o rimosse, si apportano alcune modifiche al protocollo e alla rappresentazione interna della classe `Board`. In particolare, i metodi `underAttack(int)`, `addNextQueen(int)` e `removeLastQueen()` vengono sostituiti dai metodi `underAttack(int,int)`, `addQueen(int,int)`, `removeQueen(int,int)`, rispettivamente, dove la coppia di parametri interi rappresenta gli indici di riga e di colonna di una casella. Risulta così possibile disporre una regina in qualunque casella della scacchiera, purché non sia già occupata, anche se questa è "sotto scacco" da parte di un'altra regina. È inoltre possibile rimuovere qualunque regina che sia stata disposta sulla scacchiera.

```
public class Board {

  private boolean[][] config;
  private int[] rowUnderAttack;
  private int[] columnUnderAttack;
  private int[] diagDwUnderAttack;
  private int[] diagUpUnderAttack;
  private int queens;
  private int n;

  public Board( int size ) {
    config = new boolean[size][size];
    for ( int i=0; i<size; i=i+1 ) {
      for ( int j=0; j<size; j=j+1 ) {
        config[i][j] = false;
    }}
    rowUnderAttack = new int[size];
    for ( int i=0; i<size; i=i+1 ) {
      rowUnderAttack[i] = 0;
    }
    columnUnderAttack = new int[size];
    for ( int i=0; i<size; i=i+1 ) {
      columnUnderAttack[i] = 0;
    }
    diagDwUnderAttack = new int[2*size-1];
    for ( int i=0; i<2*size-1; i=i+1 ) {
      diagDwUnderAttack[i] = 0;
    }
    diagUpUnderAttack = new int[2*size-1];
    for ( int i=0; i<2*size-1; i=i+1 ) {
      diagUpUnderAttack[i] = 0;
    }
    queens = 0;
    n = size;
  }

  public int boardSize() {
    return n;
  }

  public int queensOnBoard() {
    return queens;
  }

  public boolean underAttack(int row,int col) {

    return (
      (rowUnderAttack[row-1] > 0)
      || (columnUnderAttack[col-1] > 0)
      || (diagDwUnderAttack[row-col+n-1] > 0)
      || (diagUpUnderAttack[row+col-2] > 0)
      );
  }

  public void addQueen( int row, int col ) {

    if ( config[row-1][col-1] ) { return; }
    config[row-1][col-1] = true;
    rowUnderAttack[row-1]
      = rowUnderAttack[row-1] + 1;
    columnUnderAttack[col-1]
      = columnUnderAttack[col-1] + 1;
    diagDwUnderAttack[row-col+n-1]
      = diagDwUnderAttack[row-col+n-1] + 1;
    diagUpUnderAttack[row+col-2]
      = diagUpUnderAttack[row+col-2] + 1;
    queens = queens + 1;
  }

  public void removeQueen( int row, int col ) {

    if ( !config[row-1][col-1] ) { return; }
    config[row-1][col-1] = false;
    rowUnderAttack[row-1]
      = rowUnderAttack[row-1] - 1;
    columnUnderAttack[col-1]
      = columnUnderAttack[col-1] - 1;
    diagDwUnderAttack[row-col+n-1]
      = diagDwUnderAttack[row-col+n-1] - 1;
    diagUpUnderAttack[row+col-2]
      = diagUpUnderAttack[row+col-2] - 1;
    queens = queens - 1;
  }

  public boolean[][] arrangement() {
    return config;
  }

}  // class Board
```

Il programma in *Java* impostato qui di seguito serve a calcolare il numero di soluzioni del problema delle regine per una scacchiera *n* x *n*. Completane il codice in accordo con il protocollo della classe `Board` specificato sopra.

```
public static int queensArrangements( int n ) {
  return queensCompletions( new Board(n) );
}

public static int queensCompletions( Board board ) {
  int r = board.queensOnBoard() + 1;
  if ( r > board.boardSize() ) {
    return 1;
  } else {
    int q = ____________________________________________________________________ ;
    for ( int c=1; c<=board.boardSize(); c=c+1 ) {

       if  ____________________________________________________________________ {

         board.addQueen( r, c );

         ____________________________________________________________________

         ____________________________________________________________________

    }}
    return q;
  }
}
```