Corso di Programmazione

Prova scritta del 19 Luglio 2005

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Astrazione procedurale

Definisci una procedura *altern* in Scheme che, date tre funzioni $f : \mathbf{N} \rightarrow \mathbf{N}$, $g : \mathbf{N} \rightarrow \mathbf{N}$ e $h : \mathbf{N} \rightarrow \mathbf{N}$ come parametri, assuma come valore la funzione $q : \mathbf{N} \rightarrow \mathbf{N}$ tale che $q(3n) = f(n)$, $q(3n+1) = g(n)$ e $q(3n+2) = h(n)$.

2. Astrazione sui dati

Considera alberi binari di ricerca (BST) i cui nodi hanno valori interi e per i quali sono definite le seguenti operazioni: *(empty-tree)* per creare l'albero vuoto; *(empty? T)* per verificare se l'albero *T* è vuoto; *(root-node T)* per determinare il valore del nodo radice di un albero non vuoto *T*; *(left T)*, *(right T)* per determinare i sottoalberi sinistro e destro di un albero non vuoto *T*; *(grow-tree n L R)* che assume come valore l'albero con radice di valore *n* e con sottoalberi sinistro *L* e destro *R*. Utilizzando opportunamente il protocollo introdotto sopra, definisci in Scheme una procedura a valori booleani *nodes-in-interval* che, dato un albero binario di ricerca *T* e una coppia di interi *x*, *y*, restituisca la lista *ordinata* dei nodi di *T* il cui valore cade nell'intervallo *[x, y]*.

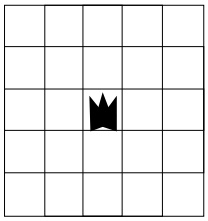
3. Astrazione sui dati

Questo esercizio fa riferimento alla soluzione del problema delle *n* regine, di cui si riporta qui di seguito il programma discusso durante il corso per calcolare il numero di soluzioni:

```
(define queens-arrangements
  (lambda (n)
    (queens-completions (empty-board n))
    ))

(define queens-completions
  (lambda (board)                    ; scacchiera completa?
    (if (= (assigned-rows board) (board-size board))
        1                            ; si: e' stata trovata una soluzione
        (next-row-trials 1 board)    ; no: considera la riga successiva
        )
    ))

(define next-row-trials
  (lambda (c board)                  ; sulla riga successiva
    (let ((depth-completions         ; a partire da colonna c
           (if (safe-next? board c)  ; prima: tentativo in posizione c
               (queens-completions (add-next-queen board c))
               0)
           ))
      (if (< c (board-size board))   ; quindi: colonne successive
          (+ depth-completions (next-row-trials (+ c 1) board))
          depth-completions
          ))
    ))
```



A partire da una data disposizione di *k* regine nella scacchiera *n* x *n*, si vuole conoscere in quali modi sia possibile aggiungere le rimanenti $n-k$ regine, senza spostare le prime *k*, in modo da pervenire a una soluzione del problema. La disposizione iniziale è descritta da una lista di lunghezza *n*, con l'*i*-imo elemento uguale a *0* se non c'è una regina nella *i*-ima riga, oppure a *j* se c'è una regina nella *j*-ima colonna della *i*-ima riga (quindi per una disposizione iniziale di *k* regine la lista contiene $n-k$ elementi uguali a zero, non necessariamente contigui). Per esempio, la lista *(0 0 3 0 0)* descrive la disposizione illustrata dalla figura qui a fianco, a partire dalla quale si possono ricavare due soluzioni diverse.

Dopo aver introdotto un ulteriore parametro *constraints* di queens-arrangements per la lista che rappresenta la disposizione iniziale, nei termini specificati sopra, apporta le modifiche che si rendono necessarie, salvaguardando il più possibile la struttura del programma, affinché *(queens-arrangements n constraints)* assuma come valore la lista di completamenti della scacchiera che risolvono il problema delle *n* regine a partire dalla disposizione iniziale data (la lista sarà vuota se non ci sono soluzioni). A tal fine numera le righe che intendi modificare nel codice formalizzato sopra e riporta i numeri e le corrispondenti modifiche sul foglio con la soluzione che proponi.

4. Asserzioni e invarianti

Il metodo statico *inside*, definito in Java qui sotto, applica la ricerca binaria per verificare se qualche componente del vettore *v* cade nell'intervallo *[x, y]*, dove *v* si presume ordinato in ordine crescente.

```
public static boolean inside( int x, int y, int[] v ) {

  /**  require ... ...   **/

  int l = 0;
  int r = v.length - 1;

  while ( l <= r )  /**  invariant ... ... **/
                    /**  variant ... ...   **/  {

    int m = (l + r) / 2;

    if ( v[m] < x ) {
      l = m + 1;
    } else if ( y < v[m] ) {
      r = m - 1;
    } else {
      return true;
    }
  }                 /**  check ... ...     **/

  return false;     /**  ensure ... ...    **/
}
```

Riporta le asserzioni appropriate al posto dei puntini per esprimere precondizioni, postcondizioni, proprietà invarianti e funzione di terminazione del comando iterativo ed altri eventuali controlli. (Puoi esprimere le asserzioni in Jass o nel linguaggio matematico, come preferisci.)

5. Classi in Java

Definisci una classe *AgendaMensile* in Java per rappresentare informazioni sintetiche sugli impegni nel corso del mese. L'estensione dell'agenda è di un solo mese. Ciascuna informazione, espressa da una semplice stringa, è associata a un intervallo orario (ore intere) di un particolare giorno. Il protocollo di *AgendaMensile* deve essere caratterizzato come segue:

```
public class AgendaMensile {

  public static final int BUSY = -1;

    ...  ...

  // Costruttore: agenda priva di annotazioni

  public AgendaMensile()  { ... }


  // Metodo per annotare un impegno (giorno, ora di inizio, ora di fine, nota)
  // giorno: [1,31];  ini: [0,23]; fin: [1,24]
  // nota di tipo String (stringa)

  public void annota( int giorno, int ini, int fin, String nota )  { ... }


  // Metodo per verificare gli impegni
  // restituisce la nota corrispondente nel caso di impegno preso fra ora e ora+1
  // restituisce una stringa "libero" altrimenti

  public String verifica( int giorno, int ora )  { ... }


  // Metodo pr conoscere la prima ora libera da impegni
  // nel corso della giornata selezionata e a partire dall'ora indicata
  // restituisce BUSY se non ci sono intervalli liberi a partire da ora

  public int libero( int giorno, int ora )  { ... }

}
```