Corso di Programmazione

Esame del 22 Luglio 2011

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmi in Scheme

Data una stringa *txt*, composta esclusivamente da lettere minuscole e spazi bianchi, la procedura `word-list` restituisce la lista delle parole che occorrono in *txt*, ordinata in ordine alfabetico e senza ripetizioni. Completa il programma impostato nel riquadro definendo le procedure `first-word` e `word-add`. A tal fine puoi utilizzare i predicati predefiniti per confrontare coppie di stringhe e posizioni relative secondo l'ordine alfabetico/lessicografico: `string=?`, `string<?`, `string<=?`, `string>?`, `string>=?`. Assumi che qualunque sequenza di lettere senza spazi bianchi intermedi sia una parola. Per esempio:

```
(word-list "il sole illumina il giglio e due api sole")
```

→ `'("api" "due" "e" "giglio" "il" "illumina" "sole")`

```
(define word-list                                    ; valore: lista
  (lambda (txt)                                      ; txt: stringa
    (cond ((string=? txt "")
           null)
          ((char=? (string-ref txt 0) #\space)  ; #\space è il carattere spazio bianco
           (word-list (substring txt 1)))
          (else
           (let ((w (first-word txt)))
             (word-add w (word-list (substring txt (string-length w))))
             ))
          )))
```

2. Procedure con valori e argomenti procedurali

Date una base intera $b \geq 2$ e una funzione f che mappa i valori delle cifre appartenenti all'intervallo $[0, b[$ nei simboli che li rappresentano (stringhe di un carattere), la procedura `h` restituisce la funzione che converte gli interi non negativi nelle corrispondenti rappresentazioni in base b. Per esempio, utilizzando le definizioni

```
(define d (lambda (v) (if (= v 0) "0" "1")))
(define t (h 2 d))
```

si hanno le seguenti valutazioni:

```
(t 0)   →    "0"        (t 1)   →    "1"        (t 5)   →   "101"
(t 12)  →  "1100"       (t 16)  →  "10000"      (t 23)  →  "10111"
```

Completa la definizione della procedura `h` riportata qui sotto, introducendo il codice Scheme appropriato negli spazi indicati a tratto punteggiato.

```
(define h
  (lambda (b f)    ; b ≥ 2,  f : [0, b[ → stringa

    ( ..........................................................................

      (if (< n b)

          ..........................................................................

          (let ((g  ..........................................................  ))

            (string-append (g (quotient n b)) (f (remainder n b)))
            )))
    ))
```

3. Oggetti in Java

Considera il modello della scacchiera realizzato dalla classe `Board` per affrontare il rompicapo delle N regine. Per le istanze di `Board` è definito il protocollo richiamato qui di seguito:

```
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()
```

Utilizzando esclusivamente il protocollo sopra specificato, definisci in Java un metodo statico che, data un'istanza di Board, restituisce il numero di posizioni della scacchiera che non sono minacciate dalle regine collocate su di essa.

4. Memoization

Considera la seguente procedura formalizzata in Scheme:

```
(define g
  (lambda (x y)                ; x, y ≥ 0
    (if (or (< x 2) (< y 2))
        (* x y)
        (+ (g (- x 1) y) (g x (- y 2)) (g (+ x 1) (- y 1)))
        )))
```

Applicando la tecnica di *memoization*, trasforma la procedura ricorsiva in un programma in Java che consenta di ridurre drasticamente i tempi di calcolo. Per quanto possibile, organizza la rappresentazione dello stato in modo tale che non vengano allocate componenti dell'array destinate a rimanere inutilizzate.

5. Verifica formale della correttezza

```
(define magic                    ; valore intero
  (lambda (x)                    ; x > 0  intero
    (let ((r (remainder x 3)))
      (cond ((= r 0)
             (- (* 3 (magic (quotient x 3))) 2))
            ((= r 1)
             (if (= x r) 1 (* 3 (magic (quotient x 3)))))
            ((= r 2)
             (if (= x r) 1 (+ (* 3 (magic (quotient x 3))) 2)))
            ))))
```

In relazione alla procedura definita sopra è possibile dimostrare che per qualsiasi $k \geq 0$ intero:

$$\texttt{(magic } 3^k\texttt{)} \quad \rightarrow \quad 1$$

Dimostra per induzione questa proprietà; in particolare:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il caso / i casi base:

- Dimostra il passo induttivo: