Corso di Programmazione

Esame del 2 Febbraio 2009

cognome e nome

Risolvi i seguenti esercizi, riporta le soluzioni in modo chiaro negli appositi spazi e giustifica sinteticamente le risposte. Dovrai poi consegnare queste schede con le soluzioni, avendo cura di scrivere il tuo nome nell'intestazione e su ciascun eventuale foglio aggiuntivo.

1. Definizione di procedure in Scheme

Derfinisci una procedura `prime-factors` in Scheme che, dato un numero intero positivo *n*, restituisce la lista dei fattori primi di *n*. In tale lista ciascun fattore primo compare tante volte quanto è il relativo esponente nella fattorizzazione di *n*. Per esempio:

```
(prime-factors 1)      →      '()
(prime-factors 15)     →      '(3 5)
(prime-factors 64)     →      '(2 2 2 2 2 2)
(prime-factors 180)    →      '(2 2 3 3 5)
```

2. Procedure in Scheme

Con riferimento alla procedura `balanced-str` così definita:

```
(define balanced-str
  (lambda (lv)
    (if (= lv 0)
        ""
        (let ((q? (even? lv)) (pn (balanced-str (- lv 1))))
          (string-append (if q? "[" "(") pn pn (if q? "]" ")"))
          ))))
```

calcola i risultati della valutazione di ciascuna delle seguenti espressioni Scheme:

`(balanced-str 0)` → _____

`(balanced-str 1)` → _____

`(balanced-str 2)` → _____

`(balanced-str 3)` → _____

`(balanced-str 4)` → _____

3. Procedure con argomenti procedurali

Considera il seguente programma in Scheme:

```
(define nxt
  (lambda (sq op)
    (if (null? (cdr sq))
        null
        (cons (op (car sq) (cadr sq))
              (nxt (cdr sq) op))
        )))
```

```
(define tri
  (lambda (sq op)
    (if (null? (cdr sq))
        sq
        (cons sq
              (tri (nxt sq op) op))
        )))
```

Assumi che il primo argomento della procedura `tri` sia una sequenza binaria, cioè una lista di `0` e `1`, non vuota. Quale espressione definisce il secondo argomento in modo tale che il valore restituito dalla procedura sia il *triangolo di Steinhaus* costruito a partire dalla sequenza binaria data? Formalizza un'opportuna espressione per il secondo argomento completando l'applicazione di `tri` riportata come esempio nel riquadro.

```
(tri '(1 1 0 1)  _____ )

                                  → '((1 1 0 1) (0 1 1) (1 0) 1)
```

4. Verifica formale della correttezza

```
(define ufo            ; valore: intero
  (lambda (x)          ; x > 0 intero
    (cond ((= x 1) 1)
          ((even? x)
           (- (* 2 (ufo (quotient x 2))) 1))
          (else
           (+ (* 2 (ufo (quotient x 2))) 1))
          )))
```

In relazione alla procedura in Scheme definita sopra è possibile dimostrare che:

$$\forall\, k \in \mathbf{N}^{+} .\ \texttt{(ufo}\ \ 3{\cdot}2^{k-1}\texttt{)} \quad \rightarrow \quad 2^{k} + 1$$

Imposta e sviluppa la dimostrazione per induzione di questa proprietà; in particolare:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il/i caso/i base:

- Dimostra il passo induttivo:

5. Astrazione sui dati

In relazione al *problema di Giuseppe Flavio*, supponi che la configurazione del gioco sia rappresentata da una classe `RoundTable`, definita in Java, le cui istanze sono accessibili attraverso il costruttore e i metodi seguenti:

`RoundTable( int n )`	costruttore: crea la condifgurazione iniziale della tavola con *n* commensali
`boolean lastPlayer()`	restituisce *true* se in tavola è rimasto solo l'ultimo commensale, *false* altrimenti
`exitingPlayer()`	restituisce il numero che identifica il commensale che sta per uscire
`nextTable()`	modifica la configurazione della tavola, determinando l'uscita di un commensale e il passaggio di mano della moka

Utilizzando questo protocollo, definisci in Java un metodo

```
public static boolean josephusTest( int n, int k )
```

che, dati il numero iniziale n di commensali e il numero k che identifica uno di essi, con $1 \leq k \leq n$, verifica se il k-imo commensale resterà a tavola per ultimo quando tutti gli altri saranno usciti.