Corso di Programmazione

Esame del 17 Luglio 2009

cognome e nome

Risolvi gli esercizi giustificando sinteticamente le risposte.

1. Procedure in Scheme

Considera la procedura così definita:

```
(define t
  (lambda (x y)
    (let ((r (remainder x y)) (q (quotient x y)))
      (cons r (if (= q 0) null (t q y)))
      )))
```

Scrivi il risultato di ciascuna delle seguenti valutazioni:

`(t 1 2)` → ..... `(t 15 2)` → .....

`(t 2 2)` → ..... `(t 15 4)` → .....

`(t 6 2)` → ..... `(t 15 10)` → .....

`(t 11 2)` → ..... `(t 2857 10)` → .....

2. Procedure con valori procedurali

Dato un valore intero compreso nell'intervallo [0, 35], la procedura `digit` restituisce la corrispondente cifra, rappresentata da una stringa di un solo carattere, utilizzando la notazione consueta per le cifre 0÷9 e le 26 lettere dell'alfabeto inglese per quelle successive, di valore maggiore o uguale a 10. Per esempio: `(digit 14)` → `"E"`.

```
(define digit
  (lambda (v)     ; 0 <= v < 36
    (if (< v 10)
        (string (integer->char (+ v code0)))
        (string (integer->char (+ (- v 10) codeA)))
        )))
```

```
(define code0 (char->integer #\0))
(define codeA (char->integer #\A))
```

Completa la definizione in Scheme della procedura `converter` che, dato il valore $b \leq 36$ della base, restituisce una *procedura* per calcolare la rappresentazione (stringa di cifre) in base b del numero naturale passato come argomento. Per esempio, una procedura per rappresentare i numeri naturali in notazione binaria può essere definita in questo modo: `(define bin-rep (converter 2))`; dopodiché si avrà: `(bin-rep 6)` → `"110"`, `(bin-rep 11)` → `"1011"`, ecc.

```
(define converter
  (lambda (b)      ; 2 <= b <= 36

    ( .....
      (if (< n b)
          (digit n)

          ( .....

            .....

            .....

            ..... )
          ))))
```

3. Verifica formale della correttezza

Dimostra per induzione che la procedura riportata a fianco calcola la potenza intera di base intera, o più precisamente:

$$\forall b \in \mathbf{N}^{+} . \forall e \in \mathbf{N} . \texttt{(power } b \ e\texttt{)} \rightarrow b^{e}$$

Sviluppa la dimostrazione, seguendo i passi indicati qui sotto:

```
(define power
  (lambda (x y)
    (let ((z (quotient y 2)))
      (cond ((= y 0) 1)
            ((= y 1) x)
            ((even? y) (power (* x x) z))
            (else (* (power x (+ z 1)) (power x z)))
            ))))
```

Formalizza il caso base / i casi base:

Formalizza l'ipotesi induttiva:

Formalizza la proprietà da dimostrare nel passo induttivo:

Dimostra il caso / i casi base:

Dimostra il passo induttivo:

4. Memoization

Considera il seguente metodo formalizzato nel linguaggio *Java*:

```
public static int f( int i, int j ) {  // i, j >= 0
  if ( i * j == 0 ) {
    return  i + j;
  } else {
    return  f( i+1, j-1 ) + f( i-1, j );
  }
}
```

Trasforma il programma ricorsivo applicando opportunamente la tecnica di *memoization*.

5. Oggetti in Java

Supponi che sia disponibile una classe `Stack`, simile a quella discussa a lezione, per rappresentare stack i cui elementi sono di tipo `String`. Il protocollo di questa classe comprende: un costruttore che crea una struttura vuota; il metodo `empty()` che consente di determinare se lo stack è vuoto; il metodo `push(`*s*`)` che aggiunge la stringa *s* in cima allo stack; il metodo `pop()` che rimuove dalla struttura e restituisce la stringa in cima allo stack.

Attraverso un procedimento del tutto analogo a quello seguito per valutare numericamente un'espressione in "forma polacca inversa" (RPN), è possibile *tradurre* un'espressione RPN nell'espressione Scheme equivalente, cioè nell'espressione Scheme la cui valutazione determina l'applicazione delle stesse operazioni (dell'aritmetica) agli stessi operandi e nello stesso ordine. Per esempio, l'espressione RPN `"2 3 * 12 3 + 18 24 6 / - * +"` ha come equivalente l'espressione Scheme `"(+ (* 2 3) (* (+ 12 3) (- 18 (/ 24 6))))"`.

Per ottenere questo risultato basta inserire nello stack le stringhe che codificano (sotto)espressioni Scheme, in sostituzione dei corrispondenti valori interi, a partire da quelle più elementari che rappresentano direttamente numeri. All'occorrenza di ciascuna operazione nell'espressione RPN, invece di procedere alla valutazione numerica, si combinano opportunamente le stringhe recuperate dallo stack, che rappresentano le sottoespressioni a cui l'operazione si applica, per formare un'espressione Scheme più complessa.

Formalizza in Java un metodo `public static String rpnToScheme( String rpn )` che, data un'espressione RPN, restituisce l'espressione Scheme equivalente (dove argomento e valore sono rappresentati da oggetti di tipo `String`), calcolata con l'ausilio di uno stack secondo le indicazioni fornite sopra. Per esempio, si vuole che:

```
rpnToScheme( "2 3 * 12 3 + 18 24 6 / - * +" ) → "(+ (* 2 3) (* (+ 12 3) (- 18 (/ 24 6))))"
```