19 dicembre 2005

Linguaggi di Programmazione

Nome e Cognome	
Matricola	
Corso di laurea	
Telefono	

1. Specificare la EBNF di un linguaggio funzionale *Scheme*-like, in cui ogni programma è composto da una sequenza non vuota di definizioni. Una definizione è introdotta dalla keyword `define`, seguita da un identificatore e da una espressione. Una espressione può essere semplice o una λ-espressione. Una λ-espressione è definita da una lista (non vuota) di parametri e da un corpo. Ecco un esempio di programma:

```
(define alfa 100)

(define beta 'fiore)

(define gamma '(1 2 ()))

(define delta (add (mul alfa 200) 25))

(define min (lambda (x y)(if (less x y) x y)))

(define sum (lambda (numeri)(if (null numeri) 0
                                (add (car numeri)
                                     (sum (cdr numeri))))))
```

2. È data la seguente tabella di operatori per la quale si assume priorità decrescente dall'alto verso il basso:

Operatore	*Tipo*	*Associatività*	*Ordine valutazione*	*Corto circuito*
^	binario	destra	da destra a sinistra	si
*****	binario	sinistra	da sinistra a destra	si
+	binario	sinistra	da sinistra a destra	no
-	binario	sinistra	da sinistra a destra	no

Sono stabilite le seguenti regole di corto-circuito:

- $expr_1$ **^** $expr_2 = 1$ quando $expr_2 = 0$.
- $expr_1$ ***** $expr_2 = 0$ quando $expr_1 = 0$.

Quindi, data la seguente istruzione di assegnamento,

```
x := a + b – c + d – e * f * g ^ (h * i) ^ m
```

a) Rappresentare l'albero della espressione di assegnamento.
b) Specificare la semantica operazionale dell'istruzione di assegnamento.

NB: Il linguaggio di specifica operazionale è così caratterizzato:

- Contiene gli operatori aritmetici **^,*,+,-**.
- Contiene gli operatori di assegnamento '**=**' e di confronto di uguaglianza '**==**'.
- Ogni operatore non può essere applicato ad espressioni, ma solo a variabili o costanti.
- Contiene le istruzioni condizionali (*if-then-endif* ed *if-then-else-endif*) i cui predicati possono essere solo semplici confronti di uguaglianza.

3. Definire nel linguaggio funzionale *Scheme* la funzione `cancella`, avente in ingresso un `elemento` ed una `lista`, che computa la lista risultante dalla cancellazione di tutte le istanze di `elemento` in `lista`, come nei seguenti esempi:

elemento	**lista**	(**cancella elemento lista**)
3	()	()
3	(3)	()
3	(1 2 3)	(1 2)
3	(1 2 3 4 3 5 3)	(1 2 4 5)
(1 2)	((1 2) (3 4) 5)	((3 4) 5)
()	(1 2 3 () (4 5) () 6)	(1 2 3 (4 5) 6)

4. Definire nel linguaggio *Haskell* la forma funzionale `computa`, avente in ingresso una funzione `f`, una funzione `g` ed una `lista` di `Integer`, che restituisce la lista ottenuta sommando l'applicazione di `f` e `g` ad ogni elemento di `lista`, come nei seguenti esempi:

f	**g**	**lista**	**computa f g lista**
quadrato	cubo	[1,2,3,4]	[2,12,36,80]
fattoriale	fibonacci	[2,4,3,0]	[3,27,8,1]

5. Specificare in *Prolog* il predicato `fib(N, F),` che risulta vero qualora `F` sia il valore della funzione di Fibonacci, definita (matematicamente) nel seguente modo:

$$\text{fib}(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ \text{fib}(n\text{-}2) + \text{fib}(n\text{-}1) & \text{altrimenti.} \end{cases}$$

6. Nell'ambito del paradigma orientato agli oggetti, definire e giustificare (sulla base di un semplice esempio) la regola di controvarianza dei parametri di ingresso nei metodi.