12 settembre 2005

Linguaggi di Programmazione

Nome e Cognome	
Matricola	
Anno di corso	
Telefono	

1. Specificare la EBNF di un linguaggio funzionale per definire una sequenza (non vuota) di funzioni, come nel seguente esempio:

```
max :: Int -> Int -> Int -> Int
max x y z
  | x >= y and x >= z     =     x,
  | y >= x and y >= z     =     y,
  | otherwise             =     z;

alfa :: Int -> Int -> String -> String -> Int
alfa i j str1 str2
  | i > j and str1 == str2 or i != j          =     i + j  + z – w,
  | i == j or j < k and k != t                =     i – j * k + 5,
  | i != j                                    =     j – 10,
  | otherwise                                 =     i * j / 2;
```

Ogni funzione è definita da un protocollo (nome, lista dei tipi dei parametri formali e tipo del valore di ritorno), dalla lista degli identificatori dei parametri formali e da un corpo. Esistono solo i tipi predefiniti **`Int`** e **`String`**. Il corpo è definito mediante il costrutto delle "guardie" in cui ogni clausola è costituita da un predicato e dalla corrispondente espressione aritmetica (che rappresenta il valore di ritorno qualora il predicato risulti vero). Un predicato e' una sequenza di operazioni di confronto connesse (se più d'una) da operatori logici **`and`** ed **`or`**. Si possono confrontare solo variabli (mediante gli operatori **`==, !=`**, **`>`**, <, **`>=`**, **`<=`**). L'ultima clausola è necessariamente quella di default (**`otherwise`**). Una espressione aritmetica coinvolge gli operatori **`+`**, **`–`**, **`*`**, **`/`**, i cui operandi possono essere variabili o costanti intere.

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*
`not`	unario	destra	-	-
`and`	binario	sinistra	da sinistra a destra	si
`or`	binario	sinistra	da destra a sinistra	no

e la seguente istruzione di assegnamento:

```
x := a and not b or c and d and e
```

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 di negazione (**`!`**), disgiunzione (**`||`**) ed assegnamento (←).
- Non contiene l'operatore di congiunzione, né le costanti logiche **`true`**, **`false`**;
- Ogni operatore non può essere applicato ad espressioni, ma solo a variabili;
- Contiene le istruzioni condizionali (*if-then* ed *if-then-else*) i cui predicati possono essere solo variabili.

3. Definire nel linguaggio funzionale *Scheme* la funzione **`select`**, avente in ingresso una funzione **`fun`** ed una sequenza **`lista`**. Si assume che la funzione **`fun`** sia unaria e computi un valore booleano. La funzione **`select`** restituisce la sotto-sequenza (eventualmente vuota) di **`lista`** comprendente gli elementi che rendono la funzione **`fun`** vera, come nei seguenti esempi:

`lista`	**`fun`**	**`(select fun lista)`**
`()`	`number?`	`()`
`(a 3 125 (x y))`	`number?`	`(3 125)`
`(a 3 () x (2 d 12) 125)`	`atom?`	`(a 3 x 125)`
`(c (c d) a c (2 3 4))`	`list?`	`((c d) (2 3 4))`
`(1 (2 (3 4 x) z) 34 () y)`	`list?`	`((2 (3 4 x) z) ())`
`(1 2 3 x y z)`	`list?`	`()`

4. Specificare in *Prolog* il predicato `fattoriale(N, F),` che risulta vero qualora `F` sia il fattoriale di `N`. Si assume di disporre dei seguenti predicati (che quindi non devono essere definiti):

   - `mul(X, Y, Z)` : vero qualora `Z` sia il prodotto aritmetico di `X` ed `Y`;
   - `sub(X, Y, Z)` : vero qualora `Z` sia la differenza aritmetica di `X` ed `Y`.

5. Descrivere il concetto di *forma funzionale* nel contesto dei linguaggi funzionali.

6. Illustrare e giustificare le regole di covarianza e controvarianza nel paradigma orientato agli oggetti.