26 luglio 2006

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	
Telefono	
Email	

1. Specificare la BNF di un linguaggio per definire ed istanziare relazioni complesse (non in prima forma normale), come nel seguente esempio:

```
relation R: [a: int, b: string, c: [d: int, e: real]];

relation S: [num: int, r: [t: [delta: int]]];

R := [(3, "alfa", [(10, 23.12)(12, 1.44)])
      (4, "beta", [(16, 3.56)])
      (5, "gamma", [])];

S := [(2, [([(10)(20)(30)])])
      (8, [([(124)(25)])])
      (28, [([])])
      (45, [])];

relation Delta: [x: int, y: int];

Delta := [];
```

Ogni programma contiene almeno una istruzione. Le istruzioni di definizione ed assegnamento delle relazioni possono essere specificate in qualsiasi ordine. Gli attributi atomici sono **`int`**, **`real`** e **`string`**. Non esiste limite di innestamento delle relazioni. Una relazione (o un attributo di tipo relazione) può essere istanziato con la relazione vuota **`[]`**.

2. Specificare la semantica denotazionale di una espressione aritmetica che coinvolge gli operatori di moltiplicazione ed esponenziazione:

   $expr \rightarrow expr_1 * expr_2 \mid expr_1 \wedge expr_2 \mid$ **id** | **num**

   sulla base delle seguenti assunzioni:

   a) **num** rappresenta una costante intera;
   b) **id** rappresenta il nome di una variabile intera;
   c) la valutazione degli operandi è da destra a sinistra;
   d) la valutazione degli operatori è in corto circuito, secondo le seguenti regole:

      - $expr_1 * expr_2 = 0$ quando $expr_2 = 0$
      - $expr_1 \wedge expr_2 = 1$ quando $expr_2 = 0$

   e) è disponibile una funzione μ(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s*;
   f) μ(**id**, *s*) = `UNDEF`, qualora il valore della variabile **id** non sia definito;
   g) è disponibile una funzione M_n(**num**) che restituisce il valore di **num**;
   h) il linguaggio di specifica denotazionale dispone degli operatori * e ^.

3. Definire nel linguaggio funzionale *Scheme* la funzione `remdup`, avente in ingresso una lista `L`, che computa la lista degli elementi di `L` senza duplicazioni, come nei seguenti esempi

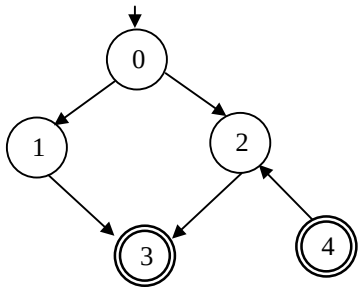
L	(**remdup L**)
()	()
(a)	(a)
(a b)	(a b)
(a b a)	(b a)
(a b a b a c d a)	(b c d a)
((a b)(a b c)(a b) 3 4 ())	((a b c) (a b) 3 4 ())
((a b)(2 (a b)) c (a b) 10)	((2 (a b)) c (a b) 10)

4. Definire nel linguaggio *Haskell* la funzione `campionato`, avente in ingresso una lista di squadre di calcio, che computa la lista di tutte le possibili partite del campionato, come nel seguente esempio:

```
campionato ["inter", "milan", "iuve"] =
    [("inter","milan"),("inter","iuve"),("milan","inter"),("milan","iuve"),("iuve","inter"),("iuve","milan")]
```

5. È data una base di fatti *Prolog* relativa alla specifica di un grafo aciclico diretto, come nel seguente esempio,

```
stati([0,1,2,3,4]).
iniziale(0).
finali([3,4]).
arco(0,1).
arco(0,2).
arco(1,3).
arco(2,3).
arco(4,2).
```



in cui il grafo ha uno ed un solo nodo iniziale ed almeno un nodo finale. Si chiede di definire il predicato `connesso` (senza argomenti), che risulta vero qualora tutti gli stati finali siano raggiungibili dallo stato iniziale (si noti che, nell'esempio, `connesso` risulta falso).

6. Illustrare le differenze sostanziali tra il concetto di funzione in un linguaggio imperativo (come il C) ed il concetto di funzione in un linguaggio funzionale (come Haskell).