4 aprile 2005

Linguaggi di Programmazione

Nome e Cognome	
Matricola	
Anno di corso	
Telefono	

1. Specificare la EBNF di un linguaggio per definire e manipolare tabelle complesse, come nel seguente esempio:

```
def T1:(a: int, b: string, c: bool, d: int);
def T2:(alfa: int, beta: int, gamma:(x: int, y: string));
def T3:(m: int, r:(v: int, w: string, s:(p: string, q: string), n: int), t: bool);
…
retrieve [a=d] T1;
retrieve [a=d and b="sole"] T1;
retrieve [a=d and b="sole" or c=false] T1;
retrieve [alfa=15 and retrieve [x=4 or y="stella"] gamma = T5] T2;
retrieve [beta=10 and retrieve [x=20] gamma = retrieve [y="luna"] gamma] T2;
…
```

   Un programma è una lista non vuota di istruzioni di definizione e/o di manipolazione. Ogni istruzione è terminata dal simbolo '**;**'. I due tipi di istruzioni possono essere specificati in qualsiasi ordine reciproco. Una istruzione di definizione specifica il nome di una tabella e la lista (non vuota) di attributi con i relativi domini. I domini atomici sono **`int`**, **`string`** e **`bool`**. Un attributo può essere a sua volta una tabella (anche complessa). Ogni istruzione di manipolazione è introdotta dalla keyword **`retrieve`**, seguita da un predicato racchiuso tra parentesi quadre e dal nome della tabella operando. Il predicato è una lista di condizioni logiche (**`and`**, **`or`**) in cui ogni condizione è un confronto di uguaglianza. Possono essere confrontati attributi semplici con altri attributi semplici o con costanti. Possono essere confrontati anche attributi complessi con altri attributi complessi (o tabelle) o con il risultato di una manipolazione di attributi complessi (o tabelle), senza limiti di innestamento delle **`retrieve`** nei predicati.

2. Specificare la semantica denotazionale della seguente istruzione (ciclo a condizione finale):

   `repeat` L **`until`** B

   in cui la lista L di istruzioni viene ripetuta (almeno una volta) finchè la condizione booleana B risulta vera (il ciclo termina quando B = **`true`**). Specificatamente, si definisca la funzione M_r(**`repeat`** L **`until`** B, *s*), in cui *s* rappresenta lo stato del programma, assumendo di aver a disposizione le seguenti funzioni:

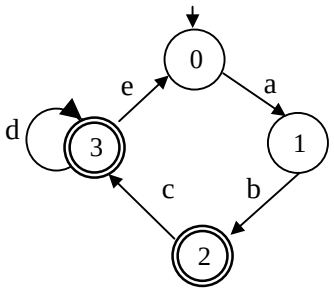
   - M_b(B, *s*) : espressione booleana → {**`true`**, **`false`**, **`errore`**}
   - M_l(L, *s*): lista di istruzioni → nuovo stato o **`errore`**.

3. Definire nel linguaggio funzionale *Scheme* la funzione **`occ`**, avente in ingresso un elemento **`x`** ed una **`lista`**, che restituisce il numero di occorrenze di **`x`** in **`lista`**:

L	**x**	**occ x lista**
()	a	0
(a)	a	1
(b a b)	b	2
(c (c d) a c)	c	2
(a (b c) (c b) (d (b c)) b (b c) (b c) a)	(b c)	3
(a () (b ()) () c)	()	2

4. Data una base di fatti *Prolog* relativa alla specifica di un automa, come nel seguente esempio:

```
state(0).
state(1).
state(2).
state(3).
initial(0).
final(2).
final(3).
trans(0,a,1).
trans(1,b,2).
trans(2,c,3).
trans(3,d,3).
trans(3,e,0).
```



in cui l'automa ha uno ed un solo stato iniziale ed almeno uno stato finale, definire le regole per i seguenti predicati:

a) **`gen(S, L)`**
in cui la lista (eventualmente vuota) **`L`** di simboli è generata da un cammino (eventualmente vuoto) che parte dallo stato **`S`** e termina in uno stato finale;

b) **`generates(L)`**
in cui la lista **`L`** è generata da un cammino che parte dallo stato iniziale e termina in uno stato finale.

Ecco alcuni esempi:

`S`	`L`	`gen(S, L)`	`generates(L)`
1	[b]	true	false
1	[b,c]	true	false
2	[c,d,d]	true	false
2	[c,d,e]	false	false
2	[c,e,a,b]	true	false
3	[]	true	false
2	[a,b]	false	true
0	[a,b,c,d]	true	true

5. Discutere le scelte progettuali relative alle espressioni aritmetiche in un linguaggio imperativo.

6. Illustrare le caratteristiche fondamentali della programmazione funzionale.