7 aprile 2009

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	

1. Specificare la grammatica BNF di un linguaggio in cui ogni frase è composta da una lista (anche vuota) di classi, come nel seguente esempio:

```
class alfa is
  attributes a: int;
             b: string;
             c: bool;
             d: record(x: bool, y: record(z1: string, num: int));
             t: table(n: int, descrizione: string);
  methods lookup(p1: int, p2: table(a: string, b: int)): int;
          remdup(a1: string, b2: int): table(a: int, b: string);
end alfa;

...

class omega inherits alfa, beta, gamma23 is
  methods ...

end omega;

...
```

Le sezioni degli attributi e dei metodi sono opzionali (ad esempio, la calsse `omega` non contiene la sezione degli attributi). Ogni metodo ha almeno un parametro formale e restituisce un valore. I possibili tipi sono **`int`**, **`string`**, **`bool`**, **`record`** e **`table`**. A differenza di **`record`** (i cui campi possono essere di qualsiasi tipo), il tipo **`table`** può contenere solo campi di tipo **`int`**, **`string`**, o **`bool`**. Una classe può opzionalmente ereditare da altre classi.

2. Specificare la semantica denotazionale di una espressione insiemistica che coinvolge operatori di unione ($\cup$) e differenza ($-$), il cui linguaggio è definito dalla seguente BNF:

$$\textit{expr} \rightarrow \textbf{(}\ \textit{expr} \cup \textit{expr}\ \textbf{)} \mid \textbf{(}\ \textit{expr} - \textit{expr}\ \textbf{)} \mid \textbf{set}$$

in cui *expr* rappresenta una espressione insiemistica e **set** il nome di un insieme. Si assume che l'istanza (anche vuota) di ogni insieme sia rappresentata da una lista (anche vuota) di elementi senza duplicazioni.

In particolare, si chiede di specificare la funzione semantica $\mathbf{M}_e(\textit{expr})$ assumendo di avere a disposizione la funzione ausiliaria $\mathbf{M}_{id}(\textbf{set})$, di cui non è richiesta la specifica, che restituisce la lista di elementi dell'insieme di nome **set** (se è definito) oppure **errore** (se non è definito). Si richiede che la specifica denotazionale sia definita mediante la notazione di pattern-matching. In particolare, è possibile utilizzare il pattern `(testa:coda)` per una lista non vuota, ed il pattern `[]` per una lista vuota. Il linguaggio di specifica denotazionale non contiene operatori insiemistici, ad eccezione dell'appartenenza, $\in$. Non è richiesto il controllo di compatibilità dei due operandi.

3. Dopo aver specificato la funzione fattoriale di un intero **n** ≥ 0 **(fact n)**, specificare nel linguaggio *Scheme* la funzione **(fattoriali n)** che, ricevendo in ingresso un intero **n** ≥ 0, genera la lista dei fattoriali degli interi compresi tra 0 ed **n**, come nei seguenti esempi:

n	(fattoriali n)
0	(1)
1	(1 1)
2	(1 1 2)
3	(1 1 2 6)
5	(1 1 2 6 24 120)

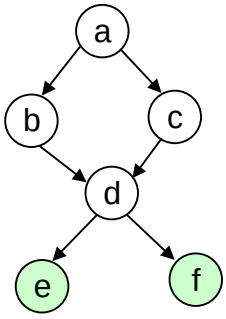
4. Dopo aver definito nel linguaggio *Haskell* le seguenti tre strutture tabellari,

- `Libri(titolo, autore, anno_pubblicazione)`
- `Prestiti(libro, lettore)`
- `Lettori(nome, anno_nascita, citta)`

specificare la funzione **autolettori** che, ricevendo in ingresso tali strutture tabellari, una **citta** ed un **anno**, resituisce la lista dei nomi dei lettori di **citta** nati prima dell'**anno**, che hanno in prestito un libro di cui sono loro stessi autori.

5. La base dei fatti di un programma *Prolog* descrive un grafo aciciclo e diretto, come nel seguente esempio:

```
arco(a,b).
arco(a,c).
arco(b,d).
arco(c,d).
arco(d,e).
arco(d,f).
```



Si chiede di specificare il predicato **foglie(F)**, che risulta vero qualora **F** sia una lista costituita unicamente da nodi foglia del grafo (nell'esempio, i nodi foglia sono colorati in verde).

6. Illustrare, sulla base di alcuni semplici esempi, il meccanismo di interpretazione di un programma *Prolog* ed il modo in cui il costrutto *cut* permette di alterare tale meccanismo.