1 dicembre 2008

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	
Telefono	
Email	

1. Specificare la grammatica EBNF di un linguaggio logico *Prolog*-like, come nel seguente esempio di frase:

```
alfa(lio2,3).
beta(teo,rio(X,Y)).
gamma(_,Z,[1,2,meo(25,X,beta(Y,rio(a,b))),legge(rino,libro),10]).
zeta(X,Y) :- beta(X,Z1), alfa(Y,Z1).
omega([a,b,X|Y]) :- tilde([X|[alfa(X,Y),beta(Z,W)]]), beta(_,W).
epsilon(geo(neo(X,Y),Z,[])) :- omega(X), alfa(Y,Z).
```

Una frase è costituita da una lista (non vuota) di fatti e/o regole. Gli argomenti di ogni predicato nei fatti e nelle regole possono essere semplici o complessi. Nel primo caso si tratta di atomi (stringhe alfanumeriche), numeri, variabili con nome o variabili anonime (rappresentate dal simbolo underscore '_'). Nel secondo caso, si tratta di strutture o liste. I costruttori struttura e lista sono perfettamente ortogonali fra loro. Inoltre, una lista può essere espressa mediante il pattern `[` *prefisso* | *coda* `]` (come nell'esempio). Nella EBNF, si richiede l'uso (fra gli altri) dei seguenti terminali:

- **atom**: stringa alfanumerica (con lettera iniziale minuscola)
- **num**: numero (stringa di cifre decimali)
- **var**: nome di variabile (con lettera iniziale maiuscola)
- **underscore**: simbolo di underscore (variabile anonima)
- **id**: nome di predicato.

La parte destra di ogni regola è rappresentata da una lista di predicati separati da una virgola e terminata da un punto. Tutti i predicati, strutture e fatti hanno almeno un argomento.

2. Specificare la semantica denotazionale di una espressione relazionale di selezione definita dalla seguente BNF:

relexpr → **select [** *pred* **]** *relexpr* | **id**

in cui *relexpr* rappresenta una operazione di selezione, *pred* un predicato di selezione, ed **id** il nome di una tabella. Si assume che l'istanza (anche vuota) della tabella sia rappresentata da una lista (anche vuota) di tuple. Il risultato della selezione è cositituito dalla sottolista di tuple della tabella per le quali il predicato di selezione risulta vero.

In particolare, si chiede di specificare la funzione semantica **Mr**(*relexpr*, s), in cui s rappresenta lo stato del programma, assumendo di avere a disposizione le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- μ(**id**, s): computa la lista di tuple della tabella di nome **id** (se è definita) oppure **error** (se non è definita).
- σ(*p*, *t*): computa il valore booleano del predicato *p* applicato alla tupla *t*.
- `empty(lista)`: stabilisce se `lista` è vuota.
- `head(lista)`: restituisce la testa di `lista`.
- `tail(lista)`: restituisce la coda di `lista`.
- `cons(testa,coda)`: restituisce la lista (`testa:coda`), in cui `testa` è una tupla e `coda` una lista.

3. Definire nel linguaggio *Scheme* la funzione **`iniziale`**, la quale, ricevendo in ingresso una **`lista`** , computa la lista ottenuta eliminando l'ultimo elemento di **`lista`**, come nei seguenti esempi:

lista	**(iniziale lista)**
(a b c)	(a b)
(() 1 (2 3 4) (3 a))	(() 1 (2 3 4))
(())	()

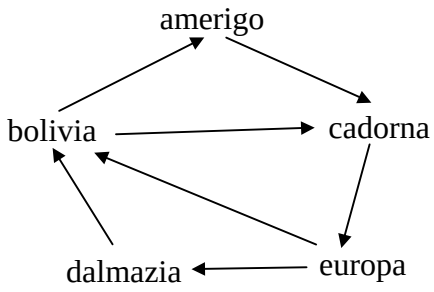
4. Definire nel linguaggio *Haskell* la funzione **`coppie`** (protocollo incluso) che, avente in ingresso una **`lista`** **(**senza duplicati) ed una funzione booleana **`f`** applicabile a due elementi di **`lista`**, computa la lista delle coppie di elementi di **`lista`** per le quali la funzione **`f`** risulta vera, come nei seguenti esempi:

lista	**f**		**coppie lista f**
[1..10]	(==)		[(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7),(8,8),(9,9),(10,10)]
"tre"	(/=)		[('t','r'),('t','e'),('r','t'),('r','e'),('e','t'),('e','r')]

5. E' data una base di fatti *Prolog* relativa ad una rete stradale urbana che specifica il collegamento tra piazze mediante strade a senso unico, come nel seguente esempio:

```
piazze([amerigo,bolivia,cadorna,dalmazia,europa]).

strade([strada(bolivia,amerigo),
        strada(amerigo,cadorna),
        strada(bolivia,cadorna),
        strada(cadorna,europa),
        strada(europa,dalmazia),
        strada(dalmazia,bolivia),
        strada(europa,bolivia)]).
```



Si chiede di specificare il predicato **`percorso(Piazza)`**, che risulta vero qualora **`Piazza`** sia una piazza dalla quale sia possibile percorrere in sequenza tutte le strade (ognuna nella rispettiva direzione) una sola volta.

6. Illustrare la politica di gestione dell'overloading delle funzioni nel linguaggio funzionale *Haskell*.