9 febbraio 2011

Linguaggi di Programmazione

Cognome e nome	
N° fogli utilizzati	

1. Specificare la definizione regolare relativa ai simboli lessicali **realconst** (costante reale) e **id** (identificatore), sulla base dei seguenti vincoli:

- Una costante reale si compone di una parte intera (obbligatoria) ed una parte decimale (opzionale);
- La parte intera è separata dalla parte decimale (se espressa) da un punto;
- Una costante reale include il segno '–' se negativa (ma non il segno '+', se positiva);
- La parte intera con più di una cifra non può iniziare con uno zero;
- Un identificatore inizia con un carattere alfabetico ed è seguito da una sequenza (anche vuota) di caratteri alfanumerici;
- Un identificatore non può essere più lungo di quattro caratteri.

2. Specificare la grammatica EBNF di un sottoinsieme **P** del linguaggio *Prolog*, in cui ogni frase è una sequenza non vuota di clausole, come nel seguente esempio (non esaustivo):

```
alfa(a,X,_).
beta([1,2,[H|T]]).
gamma(opt([X,Y,Z|W],p(5))).
r(X,Y) :- alfa(_,X,2), beta([1,2,Y]), gamma(_).
r(_,[]) :- alfa(a,b,c).
```

Il linguaggio **P** comprende strutture (con uno o più argomenti), liste (eventualmente rappresentate mediante pattern), atomi (numeri e stringhe), variabili e il carattere '_' (variabile anonima). Strutture e liste sono ortogonali fra loro. Nella specifica della EBNF si fa uso (tra gli altri) dei simboli lessicali **atom** (atomo) e **var** (variabile).

3. Specificare la semantica denotazionale di una funzione *Haskell*-like definita dalla seguente grammatica (mediante il costrutto a guardie):

function → **id** *param-list equation-list*
param-list → **id** *param-list*$_1$ | **id**
equation-list → *equation equation-list*$_1$ | *equation*
equation → '|' *cond* = *expr*

esempio di frase

```
alfa x y z
  | x > y    = x + y
  | x == z   = z - 1
  | y < z    = x * (y -z)
```

Si richiede la specifica del valore computato dalla funzione, assumendo che:

- non ci siano errori semantici nella valutazione delle condizioni (guardie) e delle espressioni;
- i parametri della funzione abbiano un valore intero;
- siano disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

  M_{expr}(*expr*): restituisce il valore (intero) di *expr*,
  M_{cond}(*cond*): restituisce il valore (booleano) di *cond*;

- il valore computato dalla funzione corrisponda alla prima espressione la cui condizione (guardia) risulti vera.
- nel caso in cui nessuna condizione (guardia) risulti vera, il valore computato sia **nil**.

4. Definire nel linguaggio *Scheme* la funzione **`maxval`**, avente in ingresso una funzione unaria **`f`** (che computa un valore intero) ed una lista (non vuota) di valori denominata **`dominio`**. La funzione **`maxval`** restituisce il massimo tra i valori computati da **`f`** quando viene applicata agli elementi di **`dominio`**.

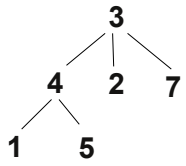
5. È data la seguente dichiarazione nel linguaggio *Haskell*, relativa ad alberi *n*-ari di interi:

```
data Albero = Nodo Int [Albero]
```

Ecco un esempio di valore di `Albero`:

```
frassino :: Albero
frassino = (Nodo 3 [(Nodo 4 [(Nodo 1 []),(Nodo 5 [])]),(Nodo 2 []),(Nodo 7 [])])
```

che corrisponde al seguente albero:



Si chiede di definire in *Haskell*, mediante la notazione di pattern-matching, la funzione **`sommalbero`** (protocollo incluso) che, ricevendo in ingresso un `Albero`, genera la somma di tutti i valori contenuti nell'albero. Nel nostro esempio avremmo:

```
sommalbero frassino
22
```

6. Specificare nel linguaggio *Prolog* il predicato **`specchio(L, S)`**, che risulta vero quando **`S`** è la lista speculare di **`L`**, come nel seguente esempio:

```
specchio([1, 2, 3, [a, b, c], [], x, y, z], S).
S = [z, y, x, [], [c, b, a], 3, 2, 1].
```

(Si consiglia di definire il predicato ausiliario **`lista(X)`**, che risulta vero quando **`X`** è una lista).

7. Dopo aver illustrato il significato del predicato cut (**`!`**) nel linguaggio *Prolog*, si analizzi il seguente programma:

```
alfa(abete).
alfa(quercia).
beta(abete).
beta(quercia).
gamma(abete).
gamma(quercia).
delta(faggio).
omega(A) :- alfa(A), beta(A), !, gamma(A).
omega(B) :- delta(B).
omega(C) :- alfa(C).
```

Quindi, spiegandone le ragioni, si indichino tutte le possibili risposte dell'interprete *Prolog* alla seguente query:

```
?- omega(X).
```