26 gennaio 2011

Linguaggi di Programmazione

Nome e Cognome	
Corso di laurea	

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

- Una costante intera con più di una cifra non può iniziare con uno zero;
- Una costante intera diverse da zero è opzionalmente qualificata da un segno (parte integrante della costante);
- Un identificatore inizia con un carattere alfabetico ed è seguito da una sequenza (anche vuota) di caratteri alfanumerici, eventualmente separati da caratteri underscore _;
- In un identificatore, non sono ammesse sequenze di due o più underscore consecutivi e l'underscore non può terminare l'identificatore.

2. Specificare la grammatica BNF del linguaggio **R** per descrivere espressioni regolari in formato testuale, nel quale ogni frase è una definizione regolare, utilizzando per **R** (tra gli altri) i seguenti simboli: **id** (identificatore), **char** (carattere dell'alfabeto), **epsilon** (simbolo **ε**). Oltre alla concatenazione, si assumono per **R** i seguenti operatori:

*****	(ripetizione zero o più volte);
+	(ripetizione una o più volte);
|	(alternativa);
[... **]**	(un range di caratteri).

Un range di caratteri può essere espresso mediante enumerazione (ad esempio, **`[abc]`**) oppure mediante gli estremi del range, eventualmente multipli (ad esempio, **`[a-zA-Z]`**).
Infine, è possibile usare le parentesi tonde per raggruppare sottoespressioni regolari.

3. Specificare la semantica denotazionale di una istruzione condizionale a due vie definita dalla seguente grammatica:

if-stat → **if** *expr* **then** *stat-list*$_1$ **else** *stat-list*$_2$
stat-list → *stat stat-list*$_1$ | *stat*
expr → **true** | **false** | **id** | *expr*$_1$ **and** *expr*$_2$ | *expr*$_1$ **or** *expr*$_2$ | **not** *expr*$_1$

Si assume che:

- **id** rappresenti il nome di una variabile logica;
- la valutazione degli operandi nelle operazioni logiche sia da sinistra a destra;
- la valutazione della disgiunzione (**or**) sia in corto circuito;
- la valutazione della congiunzione (**and**) non sia in corto circuito;
- il linguaggio di specifica denotazionale disponga degli operatori logici ∧ (congiunzione), ∨ (disgiunzione), ¬ (negazione), e degli operatori aritmetici +, –, ∗, /, applicabili unicamente ad operandi semanticamente corretti;
- siano disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

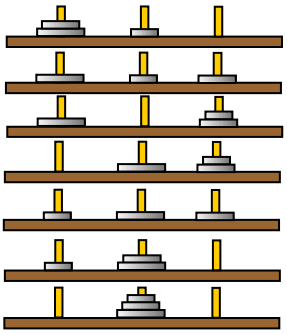
  μ(**id**, *s*): restituisce il valore della variabile **id** nello stato *s* (o **errore** nel caso **id** non sia istanziata);
  M_{stat}(*stat*, *s*): restituisce lo stato raggiunto dalla esecuzione di *stat* allo stato *s* (eventualmente **errore**).

4. Definire nel linguaggio *Scheme* la funzione **`hanoi`**, avente in ingresso un intero **`n`** > 0, che restituisce la lista di mosse che spostano la torre di **`n`** dischi dal piolo sorgente al piolo di destinazione, in modo che vengano rispettate le seguenti regole:

- Solo il disco superiore di una torre può essere rimosso ad ogni spostamento;
- Il disco rimosso da una torre non può essere spostato su un disco più piccolo di un'altra torre.

Ogni mossa è rappresentata dalla coppia (*da a*).
Ecco un esempio (**`n=3`**):

```
(hanoi 3) = ((sinistra centro)
             (sinistra destra)
             (centro destra)
             (sinistra centro)
             (destra sinistra)
             (destra centro)
             (sinistra centro))
```



5. È data la seguente dichiarazione nel linguaggio *Haskell*, relativa ad espressioni di numeri interi:

```
data Expr = Const Int
          | Var String
          | Plus Expr Expr
          | Minus Expr Expr
          | Mult Expr Expr

type Stato = [(String, Int)]
```

in cui **`Const`**, **`Var`**, **`Plus`**, **`Minus`** e **`Mult`** si riferiscono, rispettivamente, a una costante, una variabile, una somma, una differenza e una moltiplicazione, mentre **`Stato`** si riferisce alla associazione tra le variabili e i corrispondenti valori. Si chiede di definire in *Haskell*, mediante la notazione di pattern-matching, la funzione **`valexpr`** (protocollo incluso) che, ricevendo in ingresso una espressione **`e`** ed uno stato **`s`**, genera il valore di **`e`** nello stato **`s`**.

6. È data una base di fatti *Prolog* che specifica la grammatica EBNF di un linguaggio **L** in termini di assioma, simboli nonterminali, simboli terminali e produzioni. L'unico operatore esteso della grammatica è l'opzionalità, espressa dal funtore **`opt`**, con possibilità di innestamento. Ecco un esempio:

$S \rightarrow$ **a** A [**b**] | **c**
$A \rightarrow$ **b** [B **c** [**d**]] | **e**
$B \rightarrow$ **f** S | **a**

⇨

```
assioma('S').
terminali([a,b,c,d,e,f]).
nonterminali(['S','A','B']).
produzione('S',[a,'A',opt([b])]).
produzione('S',[c]).
produzione('A',[b,opt(['B',c,opt([d])])]).
produzione('A',[e]).
produzione('B',[f,'S']).
produzione('B',[a]).
```

Assumendo che la parte destra di ogni produzione inizi sempre con un simbolo terminale, si chiede di specificare in *Prolog* il predicato **`frase(F)`**, che risulta vero se e solo se **`F`** è una frase del linguaggio **L**.

7. Illustrare le scelte progettuali del linguaggio *Haskell* relative al polimorfismo e all'overloading delle funzioni.