25 Gennaio 2012

Linguaggi di Programmazione

Cognome e nome	
Corso di laurea	

1. Specificare la definizione regolare relativa ad una lista (anche vuota) di elementi, in cui ogni elemento è un identificatore o un numero, come nel seguente esempio,

```
[a2b, 25, 0, AAzxy, 0.12, -13.00, stella, 100, S289aa, 123.65, zeta]
```

sulla base dei seguenti vincoli lessicali:

- Un identificatore inizia con una lettera ed è seguito da almeno due caratteri alfanumerici.
- Un numero è composto da una parte intera (una o più cifre) ed opzionalmente da una parte decimale (un punto seguito da due cifre); la parte intera con più di una cifra non può iniziare con uno zero; un numero può avere un segno negativo (ma non positivo).
- Gli elementi della lista sono separati tra loro da una virgola e da uno spazio.

2. Specificare la grammatica BNF di un linguaggio per la manipolazione di tabelle, in cui ogni frase è una lista (anche vuota) di istruzioni, come nel seguente esempio:

```
def R (a: int, b: string)
R = [(1, "alfa"), (2, "beta")]
def T (x: string, y: string, z: int)
def V (elem: int)
V = []
T = [("sole", "luna", 25)]
select [elem >= 0 ] V
select [a > 1 and (b == "beta" or "gamma" != b)] select [a != 2 ] R
```

Esistono tre tipi di istruzioni: definizione di tabella, istanziazione di tabella e interrogazione. Ogni definizione coinvolge almeno un attributo (di tipo `int` o `string`). Nella istanziazione, la parte destra (istanza della tabella) è una lista (anche vuota) di tuple. Una interrogazione può semplicemente essere una tabella o, più in generale, la selezione di una tabella (o di un'altra selezione, senza limiti di innestamento). Il predicato di selezione (racchiuso tra parentesi quadre) può essere una comparazione (che coinvolge gli operatori `==`, `!=`, `<`, `<=`, `>`, `>=`) o, più in generale, una espressione logica che coinvolge gli operatori (ortogonali tra loro) `and` ed `or`. Si possono comparare due attributi o un attributo ed un valore, ma non due valori.

3. È data la seguente tabella di operatori, per la quale si assume priorità decrescente dall'alto verso il basso,

Operatore	*Associatività*	*Ordine valutazione*
`+`, `-`, `*`, `/`	destra	da destra a sinistra
`>`, `<`, `=`	nonassoc	da sinistra a destra
`and`	sinistra	da destra a sinistra
`or`	destra	da destra a sinistra

e la seguente espressione:

```
a + b - c * d / f > (a - b) * z + w and x = y or x > y
```

Assumendo che l'operatore **and**, a differenza di **or**, sia valutato in corto circuito, si chiede di:

- Rappresentare l'albero della espressione;
- Specificare la semantica operazionale della espressione;
- Decorare l'albero della espressione con le variabili intermedie introdotte nella specifica operazionale.

Il linguaggio di specifica operazionale è così caratterizzato:

- Contiene l'operatore di assegnamento (`:=`), gli operatori aritmetici (`+`, `-`, `*`, `/`), gli operatori di confronto (`>`, `<`, `=`) e gli operatori logici (`&&`, `||`, **not**);
- Ogni operatore non può essere applicato ad espressioni, ma solo a variabili o costanti;
- Contiene le istruzioni condizionali (*if-then-endif* ed *if-then-else-endif*) e l'istruzione *return*, che restituisce il risultato e termina l'esecuzione.

4. Specificare la semantica denotazionale del ciclo a conteggio definito dalla seguente BNF:

loop-stat → **for id = num₁ to num₂ do** *stat*

in cui [**num**$_1$ .. **num**$_2$] costituisce il range di valori interi per la variabile di conteggio **id**. In particolare, si richiede la specifica delle funzione semantica M_{loop}(*loop-stat*, `s`), in cui `s` rappresenta lo stato del programma. Sono disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- `name`(**id**): restituisce il nome (attributo lessicale) di **id**;
- `value`(**num**): restituisce il valore (attributo lessicale) di **num.**
- `value`(`nome`, `s`): restituisce il valore della variabile `nome` nello stato `s`.
- `inscope`(`nome`, `s`): restituisce un booleano, vero se e solo se la variabile `nome` è visibile nello stato `s`.
- M_{assign}(`nome`, `val`, `s`): restituisce lo stato raggiunto dopo l'assegnamento della variabile `nome` (nello stato `s`) con il valore `val`.
- M_{stat}(*stat*, `s`): restituisce lo stato raggiunto (eventualmente `error`) dopo l'esecuzione di *stat*.

5. Definire nel linguaggio *Scheme* la funzione `ordinate`, avente in ingresso una `lista` (anche vuota) di coppie di numeri, che seleziona da `lista` le coppie strettamente ordinate, quelle cioè in cui il primo elemento è maggiore del secondo.

6. È data la seguente dichiarazione nel linguaggio *Haskell*, relativa ad espressioni su insiemi di interi:

```
data Expr = Set [Int]
          | Var String
          | Union Expr Expr
          | Inter Expr Expr
          | Select (Int -> Bool) Expr

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

in cui `Set`, `Var`, `Union`, `Inter` e `Select` si riferiscono, rispettivamente, ad una istanza, una variabile, una unione insiemistica, una intersezione insiemistica e una selezione (il cui primo argomento è la funzione filtro), mentre `State` si riferisce alla associazione tra le variabili e le corrispondenti istanze (stato). Si chiede di definire in *Haskell*, mediante la notazione di pattern-matching, la funzione `computa` (protocollo incluso) che, ricevendo in ingresso una espressione di insiemi di interi ed uno stato, genera il valore della espressione in quello stato. Sono disponibili le seguenti funzioni ausiliarie (di cui non è richiesta la codifica): `head`, `tail` ed `elem` (appartenenza di un elemento ad una lista).

7. È data una base di fatti *Prolog* che specifica la grammatica BNF di un linguaggio in termini di assioma, simboli terminali, simboli nonterminali e produzioni, come nel seguente esempio:

$S \rightarrow \mathbf{x}\ A\ \mathbf{y}$
$A \rightarrow \mathbf{y}\ S\ \mathbf{z} \mid C\ \mathbf{y}$
$B \rightarrow A\ \mathbf{w} \mid \mathbf{x}\ B$
$C \rightarrow B\ \mathbf{z} \mid \mathbf{x}$

```
assioma('S').
terminale(x).
terminale(y).
terminale(z).
terminale(w).
nonterminale('S').
nonterminale('A').
nonterminale('B').
nonterminale('C').
produzione('S',[x,'A',y]).
produzione('A',[y,'S',z]).
produzione('A',['C',y]).
produzione('B',['A',w]).
produzione('B',[x,'B']).
produzione('C',['B',z]).
produzione('C',[x]).
```

Si chiede di specificare in *Prolog* il predicato `ricorsione(N)`, che risulta vero se e solo se esiste una produzione del nonterminale `N` che è (direttamente o indirettamente) ricorsiva, cioè quando è possibile derivare da `N` una forma sentenziale che inizia con il simbolo `N`. Nel nostro esempio si avrebbe:

```
?- ricorsione(N).
N = 'A' ;
N = 'B' ;
N = 'C'.
```

8. È dato il seguente frammento di codice:

```
a, b: integer;
v = array [0..5] of integer;
a := 1;
b := 2;
v := [2,4,6,8,10,12];

procedure scambia(n: integer, m: integer)
  local t: integer;
begin
  t := n;
  n:= m;
  m := t
end;
```

Assumendo il passaggio dei parametri per nome (partendo comunque dallo stato del programma dato nel frammento di codice), illustrare la computazione delle seguenti due chiamate, indicando il nuovo stato raggiunto:

```
scambia(a, b)
```

```
scambia(a, v[a])
```