22 Aprile 2014

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Specificare la definizione regolare dei simboli **identificatore** e **numero**, sulla base dei seguenti vincoli lessicali:

- Un **identificatore** è composto da almeno tre caratteri alfanumerici, ha un prefisso (anche vuoto) di cifre ed un suffisso (anche vuoto) di lettere, come nei seguenti esempi: `alfa`, `12345`, 3`beta`, `222AZ`, `12gamma`.
- Un **numero** è una costante intera o reale con segno opzionale. Nel primo caso, non sono inclusi zeri non significativi. Nel secondo caso, sia la parte intera che la parte decimale (composta da almeno due cifre) non include zeri non significativi, come nei seguenti esempi: `-9`, `0`, +`34`, `576`, `0.40`, 53`.98`, `-134.3560001`.

2. Specificare la grammatica BNF di un linguaggio in cui ogni frase è una sequenza (anche vuota) di dichiarazioni di variabili, come nel seguente esempio:

```
i, j, k: integer;
s1, s2: set of list of real;
frase: list of string;
gamma: list of set of string;
matrice: matrix [2,4,6] of integer;
alfa, beta: set of list of matrix [10] of string;
```

I tipi **integer**, **real**, **string** sono atomici, mentre **set** , **list** e **matrix** (matrice *n*-dimensionale, $n \geq 1$) rappresentano i costruttori di tipo, i quali sono ortogonali fra loro, con l'eccezione di **matrix**, i cui elementi possono essere solo atomici.

3. Specificare la semantica operazionale del seguente assegnamento (in cui **in** indica l'operatore di appartenenza):

```
R := select [ x in X and y ] R
```

assumendo che `R` sia una relazione complessa e non ordinata, così definita:

```
R: relation(x: integer, X: relation(k: integer), y: boolean);
```

L'operatore logico **and** è valutato in corto circuito, da sinistra a destra. Il linguaggio di specifica operazionale fornisce l'operatore di uguaglianza scalare (==) ma non l'operatore di appartenenza.

4. È dato il seguente frammento di grammatica BNF:

expr → *expr op expr* | **id**
op → + | **–**

in cui **id** è il nome di una lista (anche vuota) di interi, mentre gli operatori + e **–** rappresentano la somma e differenza vettoriale, il cui risultato è una lista (di lunghezza minore fra le due liste) in cui ogni elemento è il risultato della operazione aritmetica applicata agli elementi nella stessa posizione nelle due liste operando. Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità della funzione ausiliaria `istanza`(**id**, `s`), in cui `s` rappresenta lo stato del programma, la quale restituisce la sequenza di elementi della lista **id**, se questa è definita, altrimenti restituisce **errore.**

5. Specificare nel linguaggio *Scheme* la funzione `singletons`, avente in ingresso una lista (anche vuota), la quale computa la lista di liste di un elemento, come nei seguenti esempi:

```
(singletons '()) = ().
(singletons '(5)) = ((5)).
(singletons '(1 2 3 4)) = ((1) (2) (3) (4)).
(singletons '(a (2 3 4) b () 10)) = ((a) ((2 3 4)) (b) (()) (10)).
```

6. Dopo aver illustrato il significato della forma funzionale `foldr` in *Haskell*, indicare, sulla base della seguente definizione:

```
genera :: [Int] -> Int
genera = foldr (\x y -> x + 2 * y) 1
```

indicare l'espressione aritmetica computata dalla seguente applicazione ed il corrispondente risultato:

```
genera [1,2,3]
```

7. Specificare in *Prolog* il predicato `singletons(L,S)`, in cui `L` è una lista (anche vuota), mentre `S` è la lista annidata, come specificato al punto **5**.

8. Illustrare il meccanismo di passaggio dei parametri per nome e fornirne un esempio significativo.