12 Febbraio 2014

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Specificare la definizione regolare relativa ai simboli **identifier** e **num**, sulla base dei seguenti vincoli lessicali:

- Un **identifier** è composto da almeno tre caratteri alfanumerici, ha un prefisso (non vuoto) di lettere ed un suffisso (anche vuoto) di cifre, come nei seguenti esempi: `alfa`, `ab3`, `abc`, `a34`, `M240`, `Lt2`, `gH27`, `z32158`.
- Un **num** è una costante (senza segno) intera o reale. Nel primo caso, non sono inclusi zeri non significativi. Nel secondo caso, sia la parte intera che la parte decimale (composta da almeno una cifra) non include zeri non significativi, come nei seguenti esempi: `0`, `2`, `10`, `124`, `0.2`, `12.01`, `345.3456700001`

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

```
[(1, 12.35),(-2.034, +10, 20.0001),(+0.138),(0,1,2,3,-44.999103,124,-2)]
```

sulla base sei seguenti vincoli:

- Ogni tupla include almeno una costante numerica.
- Una costante numerica (con segno opzionale) è composta da una parte intera ed opzionalmente da una parte decimale.
- La parte intera e la parte decimale (composta almeno da una cifra) non contengono zeri non significativi.
- Si assumono (unicamente) i seguenti elementi lessicali: **`0 1 2 3 4 5 6 7 8 9 0 [ ] ( ) , + - .`**

3. Specificare la semantica operazionale della seguente espressione di selezione:

```
select [ X ⊆ Y or w = z ] R
```

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

```
R: table(X: table(a: integer, b: integer),
         Y: table(c: integer, d: integer),
         w: integer,
         z: integer);
```

L'operatore logico **`or`** è valutato in corto circuito, da destra a sinistra. Il linguaggio di specifica operazionale fornisce l'operatore di uguaglianza scalare (`==`) e gli operatori insiemistici di appartenenza (`∈`) e non-appartenenza (`∉`).

4. È dato il seguente frammento di grammatica BNF:

vexpr→ *vexpr* + *vexpr* | **id**

in cui **id** è il nome di un vettori di numeri. L'operazione di somma vettoriale genera un nuovo vettore con dimensione pari alla dimensione minore dei due vettori, in cui ogni elemento corrisponde alla somma aritmetica dei due elementi nella nella stessa posizione nei due vettori. Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità della funzione ausiliaria `instance`(**id**, `s`), in cui `s` rappresenta lo stato del programma, la quale restituisce la lista di numeri del vettore **id**, se questo è definito, altrimenti restituisce **errore.** Nella specifica denotazionale è possibile utilizzare il pattern lista vuota `[]` ed il pattern `testa:coda`.

5. Specificare nel linguaggio *Scheme* la funzione `quadrati`, avente in ingresso un intero `n` ≥`0`, la quale computa la lista dei quadrati dei numeri interi da `1` a `n`, come nel seguente esempio:

```
(quadrati 5) = (1 4 9 16 25).
```

6. Specificare in *Haskell* la classe di tipi `Expr`, nella quale è definito l'operatore binario `(#)`, che rappresenta l'elevamento a potenza (quest'ultima rappresentata sempre da un intero `p`≥`0`). Quindi, istanziare la classe `Expr` mediante i seguenti tipi:

- `Int`: in cui `(#)` è la classica potenza a base intera;
- `String`: in cui `(#)` rappresenta la ripetizione `p` volte della stringa.

7. Specificare in *Prolog* il predicato `quadrati(N,Q)`, in cui `N` è un intero maggiore o uguale a zero, mentre `Q` è la lista dei quadrati dei numeri interi da `1` a `N`. Si assume che `N` sia istanziato.

8. Dopo aver definito il concetto di applicazione stretta, tracciare (fornendo una spiegazione) la lazy evaluation della seguente espressione in *Haskell* (assumendo la funzione `quadrato n = n * n`):

```
quadrato (quadrato $! (2 + 3))
```