7 Aprile 2015

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Specificare la definizione regolare relativa al simbolo **identificatore**, composto da lettere, cifre e underscore, come nei seguenti esempi: `alfa`, `a25`, `a_bc_def`, `alfa_beta_1_23_456`, `xy_zwq10_658_20`. Gli underscore non possono iniziare ne terminare l'identificatore, e nemmeno essere accostati l'uno all'altro. Il prefisso (non vuoto) dell'identificatore non può contenere cifre. Il suffisso (anche vuoto) dell'identificatore non può contenere lettere.

2. Specificare la grammatica BNF di un linguaggio per la definizione dei pattern in *Haskell*, sulla base dei seguenti requisiti:

- Una frase è composta da un solo pattern;
- I pattern atomici possono essere identificatori (**id**), costanti intere (**num**) e underscore (_);
- I pattern strutturati sono le tuple, le liste ed il costruttore lista.

3. Specificare la semantica operazionale dell'operatore relazionale di prodotto cartesiano $R \times S$ mediante una notazione imperativa, sulla base dei seguenti requisiti:

- Lo schema di una relazione è una lista di coppie (*attributo*, *tipo*);
- L'istanza di una relazione è un insieme (anche vuoto) di tuple di costanti scalari;
- Sono disponibili le seguenti funzioni ausiliarie:
  - `schema(X)`: restituisce lo schema della relazione `X`,
  - `instance(X)`: restituisce l'istanza della relazione `X`,
  - `insert(elem,set)`: inserisce `elem` nell'insieme `set`,
  - `||`: operatore binario infisso di concatenazione di liste.

4. È dato un linguaggio di espressioni insiemistiche:

$$expr \rightarrow expr \times expr \mid \textbf{id}$$

in cui **id** rappresenta il nome di una variabile di tipo lista, mentre × rappresenta l'operatore di prodotto cartesiano.
Si chiede di specificare la semantica denotazionale del linguaggio sulla base dei seguenti requisiti:

- Sia disponibile una funzione `instance`(**id**, *s*) che restituisce il valore (lista di elementi) della variabile **id** nello stato *s* (nel caso in cui la variabile non abbia un valore, `instance` restituisce `ERRORE`);
- Nella specifica denotazionale è possibile utilizzare i costrutti *Haskell*-like per la manipolazione di liste.

5. Specificare in *Scheme* la funzione `indexing`, avente in ingresso una lista `V` (vettore) ed un intero `k` ≥ `0`, la quale computa l'elemento `V[k]` del vettore (assumendo che il primo elemento di `V` sia indicizzato da `0`). Quindi, specificare la funzione `element`, avente in ingresso una lista di liste `M` (matrice) e due interi `i` ≥ `0` e `j` ≥ `0`, la quale computa l'elemento `M[i][j]` della matrice. Nel caso in cui l'indicizzazione sia errata, entrambe le funzioni restituiscono l'atomo `exception`. Ad esempio:

```
(indexing '(5 12 33 alfa beta gamma) 3) = alfa
(indexing '(5 12 33 alfa beta gamma) 6) = exception
(element '((1 2 3) (4 5 6) (7 8 9 10) (alfa beta gamma)) 2 3) = 10
(element '((1 2 3) (4 5 6) (7 8 9 10) (alfa beta gamma)) 4 0) = exception
(element '((1 2 3) (4 5 6) (7 8 9 10) (alfa beta gamma)) 2 4) = exception
```

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

```
computa :: [[a]] -> [a]
computa = foldr (\x y -> reverse (x++y)) []
```

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

```
computa ["alfa", "beta", "gamma"]
```

7. Con riferimento al punto 5, specificare in *Prolog* il predicato `indexing(V,I,X)`, in cui `V` è una lista ed `I` è un intero maggiore o uguale a zero. Il predicato risulta vero quando `X` è l'elemento di `V` nella posizione `I` (partendo dalla posizione `0`), oppure quando `I` esce dal range di `V` ed `X=exception`.

8. Con il supporto di semplici esempi, illustrare la relazione che sussiste in *Haskell* tra funzioni polimorfe e funzioni overloaded.