6 Febbraio 2015

Linguaggi di Programmazione

Cognome e nome	
Matricola	

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

- Un **name**, comprendente almeno cinque caratteri alfanumerici di cui almeno due lettere e almeno due cifre, è diviso in due parti, di cui la prima solo di lettere e la seconda solo di cifre, come nei seguenti esempi: `alba25`, `Yk1123`, `mhp60`, `wQ201`, `odissea2001`.
- Un **number** è composto necessariamente da una parte intera (opzionalmente preceduta dal segno), opzionalmente seguita da una parte decimale. Entrambe le parti non includono zeri non significativi. È opzionalmente presente come suffisso anche una parte esponenziale, composta dal carattere `E` seguito da una parte intera (l'esponente) diversa da zero e senza zeri non significativi, come nei seguenti esempi: `0`, `-3`, `+0`, `+20`, `347E12`, `0.7E-3`, `12.09`, `-431.990002E+21`.

2. Specificare la grammatica BNF di un linguaggio per la definizione di una sequenza (anche vuota) di definizioni di funzioni *Scheme*, in cui ogni funzione è definita su uno o più parametri. Si assumono i seguenti requisiti:

- Il nome di una funzione o di un operatore è specificato dal terminale **function**;
- Un atomo è specificato dal terminale **atom**;
- Una lista quotata è preceduta dal terminale **quote**;
- Una lista può essere vuota.

3. Specificare la semantica operazionale della seguente espressione si selezione su una tabella:

```
select [ A ⊆ B and C ⊇ D ] T
```

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

```
T: table (A: table (a1: integer, a2: integer),
          B: table (b1: integer, b2: integer),
          C: table (c1: integer, c2: integer),
          D: table (d1: integer, d2: integer));
```

sulla base dei seguenti requisiti:

- L'operatore **and** è valutato in corto circuito, da destra a sinistra;
- L'uguaglianza è ti tipo strutturale;
- Oltre alle classiche istruzioni di controllo, il linguaggio di specifica operazionale fornisce l'appartenenza (**in**) e la negazione logica (**not**);
- È disponibile la funzione ausiliaria `insert(elem,set)`, che inserisce `elem` nell'insieme `set`.

4. È dato un linguaggio delle espressioni indicizzate definito dalla seguente grammatica BNF :

expr → **id**$_1$ **[** *expr* **]** | **id**$_2$ | **num**

in cui:

- **id**$_2$ rappresenta il nome di una variabile intera;
- **num** rappresenta una costante intera;
- **id**$_1$ **[** *expr* **]** rappresenta l'elemento della lista **id**$_1$ alla posizione *expr* (partendo da 1).

Si chiede di specificare la semantica denotazionale del linguaggio sulla base dei seguenti requisiti:

- Se la variabile indicizzata non è una lista o l'indice è minore di 1 o la lunghezza della lista indicizzata è minore dell'indice, la semantica è `ERRORE`.
- Sia disponibile una funzione `valnum`(**num**) che restituisce il valore di **num**;
- Sia disponibile una funzione `valid`(**id**, *s*) che restituisce il valore della variabile **id** nello stato *s* (nel caso in cui la variabile non abbia un valore, `valid` restituisce `ERRORE`);
- Sia disponibile una funzione booleana `list`(**id**, *s*) che stabilisce se la variabile **id** nello stato *s* sia una lista;
- Nella specifica, è possibile utilizzare il pattern lista, nelle forme `[]` e `(testa:coda)`.

5. Specificare in *Scheme* la funzione `power`, avente in ingresso due interi, $x \neq 0$ ed $n \geq 0$, la quale computa l'elevamento a potenza x^n. Quindi, specificare la funzione `sequenza`, avente in ingresso due interi, $x \neq 0$ ed $n \geq 0$, la quale computa la lista delle potenze x^1, x^2, ..., x^n. Nel caso di `n=0`, il risultato è la lista vuota.

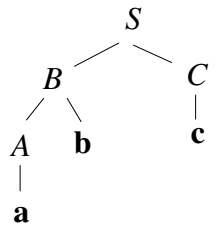
6. È data la seguente dichiarazione *Haskell*, relativa alla specifica di alberi sintattici:

```
data Syntree = Node String [Syntree]
```

Ecco un esempio di valore di `Syntree`:

```
tree = (Node "S" [(Node "B" [(Node "A" [(Node "a" [])]),(Node "b" [])]),
                  (Node "C" [(Node "c" [])])])
```

che corrisponde al seguente albero sintattico:



Si chiede di specificare la funzione `phrase`, che riceve in ingresso un `Syntree` e genera la stringa di terminali (separati fra loro da spazi) corrispondente alla frase generata dall'albero sintattico. Nel nostro esempio avremmo:

```
> phrase tree
" a  b  c "
```

7. Specificare in *Prolog* il predicato `power(X,N,P)`, in cui $X \neq 0$ ed $N \geq 0$ sono due interi, il quale risulta vero se $P = X^N$. Quindi, specificare il predicato `sequenza(X,N,S)`, il quale risulta vero se `S` è la lista X^N, X^{N-1}, X^{N-2}, ..., X^1. Nel caso di `N=0`, `sequenza` risulta vero se `S` è la lista vuota.

8. Mediante l'ausilio di semplici esempi, illustrare il linguaggio dei pattern in *Haskell*.