6 Febbraio 2013

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Specificare la definizione regolare relativa ai seguenti simboli lessicali di un linguaggio di programmazione:

- *Commento*: stringa (non vuota) di caratteri che inizia con -- e termina con newline;
- *Costante intera*: sequenza di cifre priva di zeri non significativi, con segno opzionale;
- *Costante reale*: costituita da una parte intera (obbligatoria, priva di zeri non significativi), un punto ed una parte reale (opzionale, priva di zeri non significativi), con segno opzionale;
- *Identificatore*: sequenza di caratteri alfanumerici, eventualmente separati dal carattere underscore, il quale però non può stare né all'inizio né alla fine dell'identificatore, e nemmeno dopo un altro underscore.

2. Specificare la grammatica EBNF di un linguaggio di espressioni relazionali su tabelle complesse (non in prima forma normale), come nel seguente esempio:

```
select [ A == select [ B != C and C > select [ D == E ] F ] H ] R;

S;

select [ A < B or select [ C == D ] E >= F ]
  select [ G <= select [ H != L ] select [ K > N ] W ] T;
```

Ogni frase si compone di almeno una espressione su tabella, il cui effetto è la visualizzazione del risultato. È possibile visualizzare una intera tabella o una selezione (eventualmente multipla) di una tabella. Il predicato di selezione (racchiuso tra parentesi quadre) si compone di una serie di confronti fra espressioni, collegati dagli operatori logici **and** ed **or** (senza uso di parentesi).

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

```
select [ select [a > b] X = select [c > d] Y ] 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));
```

La selezione restituisce le tuple di `R` che soddisfano l'uguaglianza delle selezioni interne sugli attributi complessi `X` ed `Y`. Il linguaggio di specifica operazionale fornisce l'operatore polimorfo di uguaglianza (==), applicabile a qualsiasi tipo, e gli operatori di confronto (<, >), applicabili solo agli interi.

4. È dato il seguente frammento di grammatica BNF, relativo alla specifica di una espressione di tipo intero in un linguaggio funzionale, che coinvolge costanti (**num**), identificatori (**id**) che hanno un binding con un valore intero, somme e chiamate di funzioni unarie (**id(***expr***)**):

expr → **num** | **id** | *expr* + *expr* | **id(***expr***)**

Assumendo che l'esecuzione di una funzione possa generare errore, si chiede di specificare la funzione semantica $M_e(expr)$, relativa al corrispondente frammento di linguaggio, assumendo la disponibilità delle seguenti funzioni ausiliarie (di cui non è richiesta la specifica):

- M_n(**num**), che restituisce il valore lessicale (intero) di **num**;
- M_{id}(**id**), che restituisce il valore lessicale (stringa) di **id**;
- M_s(`s`), che restituisce il valore intero associato alla costante simbolica di nome `s`, oppure **errore** (nel caso in cui `s` non sia definita).
- M_f(`f`), che restituisce la λ-espressione associata al nome `f`, oppure errore (nel caso in cui `f` non sia definito).

5. Specificare nel linguaggio *Scheme* la funzione `cartesiano`, che computa il prodotto cartesiano di due liste in ingresso, come nel seguente esempio:

```
(cartesiano '(1 2) '(a b c)) = ((1 a)(1 b)(1 c)(2 a)(2 b)(2 c))
```

6. In riferimento alla BNF definita al punto 4, è data la seguente dichiarazione *Haskell,* relativa ad espressioni di interi:

```
data Expr = Number Int
          | Const String
          | Plus Expr Expr
          | Fun (Int->Int) Expr

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

in cui `Bindings` associa ad ogni costante simbolica un numero. Si chiede di definire mediante la notazione di pattern matching la funzione `computa` (protocollo incluso), avente in ingresso una espressione ed una lista di bindings, la quale computa il risultato della espressione.

7. Specificare in *Prolog* il predicato `quadrupla(X)`, in cui `X` è una lista (istanziata) di interi, che risulta vero quando `X` contiene una sequenza strettamente ordinata di quattro numeri, in cui il quarto numero è la somma dei primi due numeri (ad esempio `[... 8,10,12,18,...]`).

8. Dopo aver definito il concetto di forma funzionale, stabilire (fornendo una spiegazione) quali sono le forme funzionali associate ai seguenti protocolli nel linguaggio *Haskell*:

```
Int -> Int
Eq a => a -> b
Int -> Int -> Int
a -> b -> c
Int -> (Int, Int)
Num a => a -> (b, c)
(Int -> String) -> Int
(a -> b) -> c
a -> (b -> c)
```