24 Gennaio 2014

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Specificare la definizione regolare relativa ad una tabella (non vuota), in cui ogni riga rappresenta codice, descrizione, dimensioni e peso di un prodotto, come nel seguente esempio:

```
324AX pila (2cm 8mm 1cm) 25g
158CE "lampada da tavolo" (20cm 18cm 28cm) 1.55Kg
208SG sedia (40cm 45cm 80cm) 6Kg
652PL "temperino in legno" (6mm 8cm 6mm) 22.5g
```

sulla base dei seguenti vincoli lessicali:

- Il codice inizia con tre cifre e termina con due lettere maiuscole, ma non può iniziare con uno zero.
- La descrizione è composta da una stringa non vuota di lettere minuscole, eventualmente separate da spazi. Nel caso in cui esistano spazi (e solo in tal caso), è richiesto l'uso dei doppi apici.
- Le dimensioni sono espresse da una terna di numeri interi (diversi da zero) racchiusa tra parentesi tonde, qualificati dall'unità di misura (`mm` o `cm`) e separati da uno spazio;
- Il peso si compone di una parte intera (diversa zero) ed opzionalmente da una parte decimale composta da una o due cifre. La parte decimale non può terminare con uno zero. Il peso è qualificato dall'unità di misura (`g` o `Kg`).
- Ogni elemento nella riga è separato dal successivo con uno spazio.
- Ogni riga, ad eccezione dell'ultima, è separata dalla successiva da un newline.

2. Specificare la grammatica EBNF di un linguaggio per la manipolazione di record, come nel seguente esempio:

```
r1, r2: (a: int, b: bool, c: string, d: (x: int, y: int));
r1.a = (r2.d.x - r2.d.y) / 10 + r1.a;
alfa: (n: int, s: string, w: bool);
r1 = r2;
alfa.s = r1.c;
r1.b = r2.b and (alfa.w or r2.b) or r1.b;
r2.b = true;
```

La frase può essere vuota. Esistono solo due tipi di istruzioni: dichiarazione di record e assegnamento. Ogni attributo di un record può essere un intero, un booleano, una stringa o un record (senza limiti di profondità). L'espressione di assegnamento non può mischiare elementi aritmetici (operatori e costanti numeriche) con elementi booleani (operatori e costanti booleane).

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

```
select [ a > 0 and (select [ b > c ] X == select [ d > e ] Y) ] R
```

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

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

L'operatore logico di congiunzione (**and**) è valutato in corto circuito. 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:

for-stat→ **foreach id$_1$ in id$_2$ do** *stat*

che corrisponde all'iterazione su tutti gli elementi di un array di nome **id$_2$**. Ad ogni iterazione, viene eseguita l'istruzione *stat* (che non altera l'array), in cui **id$_1$** rappresenta di volta in volta l'elemento corrente dell'array.

Si chiede di specificare la semantica denotazionale del corrispondente frammento di linguaggio, assumendo la disponibilità delle seguenti funzioni ausiliarie (in cui `s` rappresenta lo stato del programma):

- μ(**id**, `s`): restituisce la lista di elementi dell'array **id**, se questo è definito, altrimenti restituisce **undef**;
- M_{stat}(*stat*, `s`): restituisce lo stato raggiunto dalla computazione di *stat* (eventualmente **errore**).

In particolare, si richiede la specifica della funzione M_{for}(*for-stat*, `s`), che computa il nuovo stato, eventualmente **errore.** Nella specifica denotazionale è possibile utilizzare il pattern lista vuota `[]` ed il pattern `testa:coda`.

5. Specificare nel linguaggio *Scheme* la funzione `somma`, la quale riceve in ingresso due liste, `A` e `B`, e computa la somma vettoriale di `A` e `B`, rappresentata dalla lista in cui ogni elemento è la somma aritmetica dei due elementi che si corrispondono in `A` e `B` (gli elementi eccedenti vengono ignorati), come nel seguente esempio:

```
(somma '(1 2 3 4) '(5 2 6 9 0 28)) = (6 4 9 13).
```

6. É data la seguente dichiarazione nel linguaggio *Haskell,* relativa alla specifica di espressioni regolari:

```
data Regex = Symbol Char
           | Opt Regex
           | Star Regex
           | Range String
           | Cat Regex Regex
           | Alt Regex Regex
```

in cui `Symbol`, `Opt`, `Star`, `Range`, `Cat` ed `Alt` si riferiscono, rispettivamente, ad un carattere dell'alfabeto, l'opzionalità, la ripetizione zero o più volte, un range di caratteri, la concatenazione e l'alternativa. Si chiede di definire la funzione `mostra` (protocollo incluso) che, ricevendo in ingresso una espressione regolare, genera la stringa di testo che la rappresenta, come nel seguente esempio:

```
x :: Regex
x = (Alt (Star (Cat (Star (Alt (Symbol 'a') (Symbol 'b')))(Opt (Range "cde")))) (Symbol 'z'))
```

```
> mostra x
  (((a|b)*[cde]?)*|z)
```

É richiesto che le espressioni generate dagli operatori binari `Cat` ed `Alt` siano racchiuse tra parentesi tonde.

7. Specificare in *Prolog* il predicato `ordinata(L)`, in cui `L` è una lista (istanziata) di interi, che risulta vero quando `L` o contiene meno di tre numeri o ogni numero dal terzo in poi corrisponde alla somma dei due numeri precedenti, come nel seguente esempio: `[5,7,12,19,31,50]`.

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*:

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