24 Giugno 2015

Linguaggi di Programmazione

Cognome e nome	
Matricola	

1. Dato un alfabeto composto dai tre caratteri *x*, *y*, *z,* specificare l'espressione regolare delle stringhe che contengono due o tre *y* non consecutivi.

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

```
alfa, beta, gamma: struct {a: int,
                           b: string,
                           c: struct {d: bool, e: int}};
alfa = beta;
delta: struct {x: int, y: string, z: bool};
alfa.a = delta.x;
beta.c.d = true;
gamma.c.e = alfa.a;
delta.x = 20;
delta.y = "lupo";
```

La frase si compone almeno di una istruzione. Ci sono due tipi di istruzioni: definizione di struttura ed assegnamento. Nell'assegnamento, la parte sinistra può solo essere una struttura o un suo campo interno. La parte destra può anche essere una costante atomica (intero, stringa, o booleano).

3. Specificare la semantica operazionale della seguente selezione su una relazione:

```
select [ X ⊆ Y or a = b ] T
```

assumendo che `T` sia una relazione complessa così definita:

```
T: table (X: table (x: integer), Y: table (y: integer), a: string, b: string)
```

sulla base dei seguenti requisiti:

- L'operatore **`or`** è valutato in modo completo (non in corto circuito);
- Oltre alle classiche istruzioni di controllo, il linguaggio di specifica operazionale fornisce l'appartenenza (**`in`**), la negazione logica (**`!`**), la congiunzione logica (**`&&`**) e la disgiunzione logica (**`||`**).
- È disponibile la procedura ausiliaria `insert(elemento, insieme)`, che inserisce `elemento` nell'`insieme`.

4. È dato il seguente frammento di grammatica BNF, relativo alla specifica del ciclo *loop-until* in un linguaggio imperativo (il ciclo termina quando la condizione risulta vera):

loop-stat → **loop** *stat* **until** *expr*
expr → *expr* **and** *expr* | **not** *expr* | **id** | **true** | **false**
stat → *assign-stat* | *loop-stat*

Specificare la semantica denotazionale del frammento di linguaggio, assumendo che **id** rappresenti il nome di una variabile logica, la congiunzione logica sia valutata in modo completo (non in corto circuito), il linguaggio di specifica disponga degli operatori logici **and** ed **or**, (ma non l'operatore di negazione), siano disponibili le funzioni ausiliarie M_{id}(**id**, `s`), che restituisce il valore di **id** allo stato `s` (eventualmente **errore**), e M_{assign}(*assign-stat*, `s`). In particolare, specificare la funzione semantica M_{stat}(*stat*, `s`).

5. Specificare in *Scheme* la funzione booleana `superlist`, avente in ingresso due liste, `S` e `L`, che risulta vera quando `S` ⊃ `L`, cioè quando `S` contiene strettamente `L` (partendo dal primo elemento). Ad esempio:

```
(superlist '(a b c d) '(a b c)) = #t
(superlist '(a b c d) '(a b c d)) = #f
(superlist '(a b c d) '(a c b)) = #f
(superlist '(1 2 a b c) '(a b c)) = #f
(superlist '(a) '() = #t
(superlist '() '() = #f
```

6. Specificare in *Haskell* la funzione `scindi_quadruple` (protocollo compreso), avente in ingresso una lista (anche vuota) di quadruple, la quale computa la quadrupla di liste in ordine speculare, come nei seguenti esempi:

```
scindi_quadruple [] = ([],[],[],[])
scindi_quadruple [(1, "alfa", True, "beta")] = (["beta"],[True],["alfa"],[1])
scindi_quadruple [(1, "alfa", True, "beta"),(2, "gamma", False, "delta")] =
                 (["beta","delta"],[True,False],["alfa","gamma"],[1,2])
```

7. Con riferimento al punto 5, specificare in *Prolog* il predicato `superlist(S,L)`.

8. Illustrare il modo in cui in *Haskell* si possono definire le funzioni overloaded.