24 luglio 2007

# Linguaggi di Programmazione

| | |
|---|---|
| *Nome e Cognome* | |
| *Corso di laurea* | |
| *Telefono* | |
| *Email* | |

1. Specificare la grammatica BNF di un linguaggio per la definizione di moduli di programma, in cui ogni frase contiene una specifica di modulo, come nel seguente esempio:

```
module M is
  var a, b: integer;
      c, d: vector [10] of string;
      r: record (a: integer, b: string);
      x, y, alfa22: vector [5] of record (a: integer, b: vector [20] of string);
  body
    a := 1;
    b := 2;
    c[2] := "alfa";
    r.a := 3;
    x[2].b[3] = "beta";
  end
```

Un modulo ha un identificatore e contiene due sezioni. La prima sezione (introdotta dalla keyword **`var`**) specifica una serie di dichiarazioni di variabili con il loro tipo. I costruttori di tipo (ortogonali tra loro) sono **`vector`** e **`record`** . I tipi semplici sono **`integer`** e **`string`**. Nel caso di vettore, si indica la dimensione, mentre per il record si elencano gli attributi (almeno uno). La seconda sezione (introdotta dalla keyword **`body`**) specifica una lista di assegnamenti, in cui la parte sinistra è una espressione che rappresenta simbolicamente un indirizzo, mentre la parte destra può essere solo una costante semplice.

2. È dato il seguente frammento di codice in un linguaggio imperativo:

```
if a > 7 then
    if a > 5 then
        a := a - 3
    else
        a := a - 1
    end-if
else
    a := a - 4
end-if;
```

Nell'ambito della semantica assiomatica, assumendo che la postcondizione del frammento sia $Q$ = { `a > 0` }, determinare la precondizione più debole $P$ specificandone i passi computazionali.

3. Dopo aver definito in *Scheme* la funzione booleana `manca`, che stabilisce se `x` non è incluso nella lista `L`, come nei seguenti esempi,

| x | L | (manca x L) |
|---|---|---|
| 1 | (1 2 3) | #f |
| 4 | (a b c) | #t |
| (a) | (a b c) | #t |
| (b) | (a (b) c) | #f |
| () | () | #t |
| () | (1 2 ()) | #f |

   definire la funzione `unione`, avente in ingresso due liste (senza duplicati), `L1` e `L2`, che computa l'unione insiemistica (quindi, senza duplicati) `L1` ∪ `L2`.

4. Definire nel linguaggio *Haskell*, mediante la notazione di pattern-matching, le due funzioni definite al punto 3.

5. Implementare nel linguaggio *Prolog* il quesito al punto 3, specificando i seguenti predicati:

   `manca(X, Y)`: vero quando `X` non è incluso in `Y`;
   `unione(X, Y, Z)` : vero quando `Z` è l'unione insiemistica di `X` ed `Y`.

6. Nell'ambito del paradigma orientato agli oggetti, definire e giustificare (sulla base di un semplice esempio) la regola di controvarianza dei parametri di ingresso nei metodi.