Prova Scritta di Linguaggi di Programmazione I

11/07/2008

Si noti che quanto messo nei riquadri è una *bozza* fornita solo a *titolo indicativo*.
Quindi **non è** un modello di soluzione completa che ci si aspetta ad un esame.

1. Con riferimento al seguente programma in Pascal, si rappresenti il P-code relativo alla funzione `gcd`, cioè la sezione di codice all'interno del riquadro.

```
program esercizio;

  var v: array [ 0 .. 255 ] of integer;
    ...    ...
  procedure stp( var x, y, r: integer );
  begin  r := x mod y;  x := y;  y := r  end;

  function gcd( n: integer ) : integer;
    var r: integer;
  begin
    if  n = 0  then
      gcd := v[n]
    else
      begin
        stp( v[n-1], v[n], r );
        if  r = 0  then
          gcd := gcd( n-1 )
        else
          gcd := gcd( n )
      end
  end;

  begin  ...  ...  end.
```

2. Sia $I_{L_1}^{L_2}$ un interprete di L_2 scritto in L_1. Sia $[\![P]\!]$ la funzione calcolata dal programma P, $\forall P$. Si stabiliscano un programma Q scritto in L_3 ed un R scritto in L_2 per cui la valutazione $[\![I_{L_1}^{L_2}]\!](R, Q)$ abbia senso e produca un compilatore da L_3 ad L_2.

3. Sia L_1 il linguaggio $\{xyw \mid xw = w^R y,\ x, y \in \{\mathtt{a}, \mathtt{b}\},\ w \in \{\mathtt{b}, \mathtt{c}\}^*\}$ e $L_2 := \{\mathtt{a}w \mid w = w^R,\ w \in \{\mathtt{a}, \mathtt{c}\}^*\}$, dove w^R è la stringa w rovesciata. Si diano le stringhe di L_1 ed L_2 di lunghezza ≤ 4. Inoltre si diano due grammatiche non ambigue, con simboli iniziali S_1 ed S_2 per generare L_1 ed L_2. Infine si dica se la grammatica ottenuta unendo le due precedenti e la produzione $S \rightarrow S_1 \mid S_2\mathtt{a}$ è ambigua (mostrando un testimone dell'ambiguità oppure argomentando opportunamente sulla non ambiguità).

$\{w \in L_1 \mid |w| \leq 4\} = \{\mathtt{aa}, \mathtt{bb}, \mathtt{bbb}, \mathtt{bbbb}, \mathtt{bbcb}\}$.

$\{w \in L_2 \mid |w| \leq 4\} = \{\mathtt{a}, \mathtt{aa}, \mathtt{ac}, \mathtt{aaa}, \mathtt{acc}, \mathtt{aaaa}, \mathtt{acac}, \mathtt{aaca}, \mathtt{accc}\}$.

S ambigua: ad esempio si possono far vedere i due alberi di parsing di `aa`.

4. Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma in un linguaggio C-like con assegnamento che calcola *l*-value *prima* di *r*-value, valutazione argomenti chiamate *da destra a sinistra* e indici vettori inizianti da 0:

```
int x[3] = {data_di_nascita};
int  i=2, k=1;
int mess(ref int z, int i) {
```

```
    x[i--] += z++ + i++;
    write(i, (x[k++] += k *= x[i++]--));
    write(x[1]-x[2]);
    return x[i]-1;
}
write(mess(x[i++], k--));
write(x[i], i--);
```

> Attenzione che l'ordine di valutazione degli argomenti delle chiamate *non ha nulla a che vedere* con quello che le procedure poi fanno con gli argomenti. In particolare una `write`(e_1, e_2) stamperà sempre prima (il valore di) e_1 e poi e_2.

5. Assumendo di utilizzare nell'Esercizio 8 la tecnica di implementazione CRT con pila nascosta, si mostri schematicamente la situazione sullo stack nascosto e nel vettore centralizzato quando entra in esecuzione G. La situazione incontrata è consistente con quanto dovrebbe succedere?

6. Rappresentando Binary Search Trees con il tipo di dato

   ```
   data (Ord a,Show a) => BST a = Void | Node a (BST a) (BST a)
   ```

   si scriva una funzione `diff2next` che, dato un BST, costruisce un BST (annotato) di coppie dove il primo elemento di ogni coppia è l'elemento dell'albero originale mentre il secondo elemento è `Just(`*la differenza rispetto al valore successivo*`)`, secondo l'ordinamento dei valori contenuti, oppure `Nothing` per il nodo di valore massimo. A titolo di esempio, `Node 4 Void (Node 7 (Node 5 Void Void) Void)` restituisce la soluzione `Node (4,Just 1) Void (Node (7,Nothing) (Node (5,Just 2) Void Void) Void)`.

7. Rappresentando Alberi "generici" con il tipo di dato

   ```
   data (Eq a,Show a) => Tree a = Void | Node a [Tree a]
   ```

   si scriva una funzione `degree` che restituisce il grado di un albero (il massimo del numero di figli di ogni nodo). Ad esempio `degree (Node '+' [Node '*' [Node 'x' [], Node 'y' []], Node 'z' []])` restituisce 2

8. Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma espresso in un linguaggio C-like con scoping *dinamico*, *deep binding*, assegnamento che calcola *l*-value *dopo* *r*-value e valutazione delle espressioni da sinistra a destra:

```
int x = 3, y = -2, z = 2;
int F(ref int y, name int e, int H(valres int, name int)) {
    int w = H(z, y++ + e);
    z += y--+w; write(z);
    return (--y + w);
}
{   int x = -3, y = 1;
    int G(valres int z, name int e) {
      z += ++x; write(x,y,z);
      y += e; write(y);
      return z++;
    }
    write(F(x, y++ + z++, G));
    write(x,y);
}
write(x,y);
```

$-2, 1, 0, 3, 0, -3, -3, 3, 3, -2$