Prova Scritta di Linguaggi di Programmazione I

26/02/2009

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 `vsum`, cioè la sezione di codice all'interno del riquadro. (il simbolo *cappelletto*, in sostituzione di ↑, si riferisce ai puntatori.)

```
program esercizio;

  const  IMAX = ...;
  type   v2 = array[ 0 .. 1 ] of real;   vect = ^v2;   ...

  procedure vsum( k: integer; u, v: vect; var s: vect );

  begin
    if  k = 0  then  new( s );
    if  k <= IMAX  then
      begin  s^[k] := u^[k] + v^[k];   vsum( k+1, u, v, s )  end
  end;

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

2. Sia $PE_{L_1}^{L_2}$ un valutatore parziale di L_2 scritto in L_1, $C_{L_1}^{L_2 \to L_3}$ un compilatore da L_2 a L_3 scritto in L_1 e $I_{L_1}^{L_2}$ un interprete di L_2 scritto in L_1. Sia $[\![P]\!]$ la funzione calcolata dal programma P. Si determinino L_a e L_b affinché la valutazione $[\![PE_{L_0}^{L_1}]\!](I_{L_a}^{L_3}, C_{L_b}^{L_2 \to L_4})$ abbia senso e si dica cosa tale valutazione produce.

Per avere $I_{L_a}^{L_3}$ in input a $PE_{L_0}^{L_1}$ deve essere $L_a = L_1$. Quindi

$$[\![PE_{L_0}^{L_1}]\!](I_{L_1}^{L_3}, C_{L_b}^{L_2 \to L_4}) = P \in L_1$$

con P tale che, $\forall Q$

$$[\![P]\!](Q) = [\![I_{L_1}^{L_3}]\!](C_{L_b}^{L_2 \to L_4}, Q)$$

per avere $C_{L_b}^{L_2 \to L_4}$ input di $I_{L_1}^{L_3}$ deve essere $L_b = L_3$. Quindi

$$[\![P]\!](Q) = [\![I_{L_1}^{L_3}]\!](C_{L_3}^{L_2 \to L_4}, Q) = [\![C_{L_3}^{L_2 \to L_4}]\!](Q)$$

Da cui deduciamo che $Q \in L_2$ e che P è un compilatore da L_2 ad L_4 scritto in L_1.

3. Sia G la grammatica individuata dalle produzioni

$$S ::= \mathtt{a}\ B \mid \mathtt{b}\ A \mid \mathtt{b} \mid \mathtt{ba}\ C \qquad A ::= \mathtt{b}\ S \mid \mathtt{a}\ C$$
$$C ::= \mathtt{a}\ A \mid \mathtt{b}\ B \mid \mathtt{a} \qquad B ::= \mathtt{a}\ S \mid \mathtt{b}\ C$$

e sia L il linguaggio $\{v \mid vw^R = wv^R,\ v \in \{\mathtt{a},\mathtt{b}\}^*,\ w \in \{\mathtt{a},\mathtt{c}\}^*\}$, dove w^R è la stringa w rovesciata.

La grammatica G è ambigua? (si mostrino i 2 alberi di parsing testimoni dell'ambiguità oppure si argomenti opportunamente sulla non ambiguità)

In ogni stringa di $\mathcal{L}(G)$ quante `a` e `b` si possono avere?

Si diano le stringhe di L di lunghezza ≤ 3, una stringa di $\mathcal{L}(G) \setminus L$ e una di $L \setminus \mathcal{L}(G)$.

G è ambigua, basta mostrare i 2 alberi di parsing di `baa`.

Ogni stringa di $\mathcal{L}(G)$ ha un numero pari di `a` e dispari di `b`.

$\{w \in L \mid |w| \leq 3\} = \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bb}, \mathtt{aaa}, \mathtt{bab}, \mathtt{aba}, \mathtt{bbb}, \mathtt{ab}, \mathtt{abb}, \mathtt{aab}\}$.

$\mathtt{baa} \in \mathcal{L}(G) \setminus L$, $\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{bb}, \mathtt{ab}, \mathtt{aaa}, \mathtt{abb} \in L \setminus \mathcal{L}(G)$

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 *dopo* *r*-value, valutazione espressioni *da sinistra a destra*, argomenti chiamate *da destra a sinistra* e indici vettori inizianti da 1:

```
int i=1, j=2, v[3]={data_di_nascita};
int mess(int i, ref int z) {
 while ( (v[i++] += v[i]--) < 70 );
 v[--i] -= v[++j]++ - z;
 return i;
}
write(mess(j, v[j--]));
write(v[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 **lista** delle associazioni, si mostri schematicamente la situazione quando entra in esecuzione H. La situazione incontrata è consistente con quanto dovrebbe succedere?

La situazione non è consistente. La variabile x vista da H è quella di F invece che quella del blocco principale.

6. Mediante la codifica ad albero chiamata "Quad Tree" si codificano immagini quadrate il cui lato sia una potenza di 2. Se l'immagine è omogenea la si codifica, indipendentemente dalle sue dimensioni, con una foglia contenente il colore. Se l'immagine è eterogenea allora si utilizza un nodo i cui figli contengono le codifiche dei quadranti superiore-sinistro, superiore-destro, inferiore-sinistro, inferiore-destro, rispettivamente. Usando il tipo di dato

```
data (Eq a,Show a) => QT a = C a | Q (QT a) (QT a) (QT a) (QT a)
  deriving (Eq,Show)
```

si scriva una funzione Haskell `commonPoints` che data una lista non-vuota di QuadTrees l costruisce il QuadTree "maschera", a valori booleani, che ha "un pixel" a `True` se nella medesima posizione tutte le immagini di l hanno pixels uguali, `False` altrimenti. Ad esempio

```
let q2 = Q z u u u; z = C 0; u = C 1
in commonPoints [u,q2]
```

restituisce Q (C **False**) (C **True**) (C **True**) (C **True**).

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

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

si scriva una funzione `frontier` che restituisce la frontiera di un albero (la lista degli elementi delle foglie, rispettando l'ordine).

Quadratica semplicissima

```
frontier t = treefoldr aggr1 [] (++) [] t
  where
    aggr1 a [] = [a]
    aggr1 _ xs = xs
```

con `treefoldr` soluzione dell'apposito esercizio dell'eserciziario.

La variante lineare si fa passando dall'object level al function level. Qui non posso mostrarla.

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, valutazione delle espressioni da sinistra a destra e indici vettori inizianti da 0:

```
int x = 2, y[] = {1,4,7};
int H(ref int y) {
  int z = y--;
  x += z++;  write(x,y,z);
  return z--;
}
int F(valres int v, int R(ref int)) {
    void G(name int v) {
        write(--x);  y[x++] += v-x; write(x,y);
    }
    int x = 3;
    G( v-- + R(v) );  return v--;
}
write(F(y[x++], H));
write(x,y);
```

2, 9, 5, 7, 3, 1, 4, 19, 5, 9, 1, 4, 4