Prova Scritta di Linguaggi di Programmazione I

17/09/2008

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 `gcdf`, cioè la sezione di codice all'interno del riquadro.

```
program esercizio;

  procedure gcdf( x, y: integer; var i, j: integer );
    var q, r, t: integer;
  begin
    q := x div y;   r := x mod y;
    if  r = 0  then
      begin  i := 0;  j := 1  end
    else
      begin  gcdf( y, r, i, j );  t := i - j*q;  i := j;  j := t  end
  end;

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

2. Sia $I_{L_1}^{L_2}$ un interprete di L_2 scritto in L_1 e $PE_{L_1}^{L_2}$ un valutatore parziale di L_2 scritto in L_1. Sia $[\![P]\!]$ la funzione calcolata dal programma P, $\forall P$. Si stabilisca un programma Q scritto in L_0 per cui la valutazione $[\![Q]\!](PE_{L_1}^{L_2}, I_{L_2}^{L_3})$ abbia senso e produca un compilatore.

Scegliendo $Q = PE_{L_0}^{L_1}$

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

con P tale che, $\forall T \in L_3$

$$[\![P]\!](T) = [\![PE_{L_1}^{L_2}]\!](I_{L_2}^{L_3}, T) = R \in L_2$$

con R tale che, $\forall X$

$$[\![R]\!](X) = [\![I_{L_2}^{L_3}]\!](T, X) = [\![T]\!](X)$$

il risultato P è quindi un compilatore da L_3 a L_2 scritto in L_1 ($C_{L_1}^{L_3 \to L_2}$).

3. Sia L_1 il linguaggio $\{xwz \mid xzw = (yzw)^R,\ x, y \in \{\mathtt{a}, \mathtt{b}\},\ z \in \{\mathtt{b}, \mathtt{c}\},\ w \in \{\mathtt{a}, \mathtt{c}\}^*\}$ e $L_2 := \{w \mid w = w^R,\ w \in \{\mathtt{a}, \mathtt{c}\}^*\}$, dove w^R è la stringa w rovesciata. Si diano le stringhe di L_1 ed L_2 di lunghezza ≤ 5 e 3 rispettivamente. 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 $S \to S_1 \mid \mathtt{a}S_2$ è ambigua (mostrando un testimone dell'ambiguità oppure argomentando opportunamente sulla non ambiguità).

$\{w \in L_1 \mid |w| \leq 5\} = \{\mathtt{bb}, \mathtt{aab}, \mathtt{aac}, \mathtt{acac}, \mathtt{aacac}, \mathtt{accac}\}$.
$\{w \in L_2 \mid |w| \leq 3\} = \{\varepsilon, \mathtt{a}, \mathtt{c}, \mathtt{aa}, \mathtt{cc}, \mathtt{aaa}, \mathtt{cac}, \mathtt{aca}, \mathtt{ccc}\}$.

$$S_1 ::= \mathtt{bb} \mid \mathtt{aab} \mid \mathtt{aac} \mid \mathtt{a}\ S_2\ \mathtt{cac}$$

$S_2 ::= \varepsilon \mid \texttt{a} \mid \texttt{c} \mid \texttt{a}\ S_2\ \texttt{a} \mid \texttt{c}\ S_2\ \texttt{c}$

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

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 1:

```
int x[3] = {data_di_nascita};
int  i=1, j=2;
int mess(int j, ref int z) {
 z++ -= z - x[i++];
 write(j, x[j++] *= x[--j] -= x[i++]++);
 write(x[1]+x[2]);
 return x[j]-1;
}
write(mess(j++, x[j++]));
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 F. La situazione incontrata è consistente con quanto dovrebbe succedere?

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

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

si scriva una funzione `transpose` che restituisce il trasposto di un albero (l'albero che si ottiene associando ad ogni nodo i trasposti dei figli in ordine inverso).

Ad esempio per `Node 1 [Node 2 [Node 3 [], Node 4 []], Node 5 []]` si ottiene `Node 1 [Node 5 [], Node 2 [Node 4 [], Node 3 []]]`

```
transpose t = treefoldl (flip Node) Void (:) [] t
```

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

7. Molte tecniche sviluppate per la compressione di immagini si basano su una codifica ad albero chiamata "Quad Tree". Si codificano in questo modo immagini quadrate il cui lato sia una potenza di 2. Se l'immagine è omogenea (stesso colore) 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 => QT a = C a | Q (QT a) (QT a) (QT a) (QT a)
```

si scriva una funzione Haskell `occurrencies` che dato un QuadTree ed un colore determina il numero (minimo) di pixel di quel colore. Ad esempio

```
let z = C 0; u = C 1; q = Q z u u u in occ (Q q (C 0) (C 2) q) 0
```

restituisce 6 (visto che il QuadTree codifica almeno 16 pixel).

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

```
int x = 7, y = 5;
int Q(name int v, ref int x) {
     int w = F(v-x, y);
     y += w+x--; write(y);
     return (++x + w);
}
int F(name int v, valres int z) {
  y += x++; write(x,y,z);
  z -= y; write(v);
  return z--;
}
{    int x = 1, y = -2, z = 3;
     write(Q(x++ + z++, y));
     write(x+2,y);
}
write(x,y);
```

Scegliendo shallow binding: $-3, -3, -2$ e poi non termina

Scegliendo deep binding: $-3, -3, -2, 7, 0, 2, 4, 1, 7, 5$