Prova Scritta di Linguaggi di Programmazione I

05/12/2007

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

```
program esercizio;
  ...   ...
  function even( n: integer ) : boolean;

    function oddn( n: integer ) : boolean;

    begin
      if n = 0 then
        oddn := false
      else
        oddn := even( n-1 )
    end;

  begin
    if n = 0 then
      even := true
    else
      even := oddn( n-1 )
  end;

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

2. Sia $PE_{L_a}^{L_b}$ un valutatore parziale di L_b scritto in L_a e Q un programma scritto in L_c. Sia $[\![P]\!]$ la funzione calcolata dal programma P, $\forall P$. Si stabilisca un qualche programma R per cui abbia senso la valutazione $[\![PE_{L_a}^{L_b}]\!](R, Q)$ e si dica cosa produce.

Scegliendo $R = I_{L_b}^{L_c}$, un un interprete di L_c scritto in L_b,

$$[\![PE_{L_a}^{L_b}]\!](I_{L_b}^{L_c}, Q) = P \in L_b$$

con P tale che, $\forall X$

$$[\![P]\!](X) = [\![I_{L_b}^{L_c}]\!](Q, X) = [\![Q]\!](X)$$

Quindi P è una versione equivalente di Q ma scritto in L_b (il risultato della compilazione da L_c ad L_b di Q).

3. Sia L il linguaggio $\{xw \mid xw = w^R x, x \in \{\mathtt{a}, \mathtt{c}\}, w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*\}$, dove w^R è la stringa w rovesciata. Si diano le stringhe di L di lunghezza ≤ 4. Inoltre si completi la grammatica G individuata dalle produzioni

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

affinché generi L. Infine si dia l'albero di parsing di `cabbac`.

$\{w \in L \mid |w| \leq 4\} = \{\mathtt{a}, \mathtt{c}, \mathtt{aa}, \mathtt{cc}, \mathtt{aaa}, \mathtt{cac}, \mathtt{aba}, \mathtt{cbc}, \mathtt{aca}, \mathtt{ccc}, \mathtt{aaaa}, \mathtt{caac}, \mathtt{abba}, \mathtt{cbbc}, \mathtt{acca}, \mathtt{cccc}\}$.

$$S ::= \mathtt{a} \mid \mathtt{c} \mid \mathtt{a}\ A\ \mathtt{a} \mid \mathtt{c}\ A\ \mathtt{c} \qquad\qquad A ::= \varepsilon \mid \mathtt{a} \mid \mathtt{b} \mid \mathtt{c} \mid \mathtt{a}\ A\ \mathtt{a} \mid \mathtt{b}\ A\ \mathtt{b} \mid \mathtt{c}\ A\ \mathtt{c}$$

Prova Scritta di Linguaggi di Programmazione I

05/12/2007

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

```
int x[3] = {data_di_nascita};
int  i=1, k=2,
int mess(ref int z, int k) {
  x[--i] += z++;
  write(k, (x[--i] += k += x[k--]--));
  write(x[0]-x[2]);
  return x[i*k+1]-1;
}
write(mess(x[i++], k--));
write(x[k], k--);
```

> 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.

Negli Esercizi 6 e 7 si usino variabili anonime quando possibile e **si dia esplicitamente il tipo di tutte le funzioni**.

6. Rappresentando BST (Binary Search Tree) con il tipo di dato

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

   ed essendo $\max_n t$ il più grande dei valori strettamente minori di n di un BST t (se ne esistono), si scriva una funzione Haskell `boundedMaximum` che dato un numero n e una lista l di BST determina la lista dei $\max_n t$ che esistono, al variare di t in l.

   A titolo di esempio, `boundedMaximum 3 [Node 3 Void Void, Node 5 (Node 2 Void Void) Void]` vale `[2]`

7. Rappresentando QuadTrees con il tipo di dato

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

   si scriva una funzione Haskell `zipWith` che, data un'operazione binaria $\otimes$ e due QuadTrees q_1 e q_2 costruisce il QuadTree che codifica l'immagine risultante dall'applicazione di $\otimes$ a tutti i pixel della stessa posizione nelle immagini codificate da q_1 e q_2. Ad esempio

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

   restituisce Q (C 2) (C 3) (C 3) (C 3)

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

```
    }
    write(Q(x++ + z++, y, F));
    write(x,y);
}
int Q(name int v, ref int x, int R(valres int, name int)) {
    int w = R(y, v+x);
    y += w+x--; write(y);
    return (--x + w);
}
write(x,y);
```

$-2, 1, -1, 0, -2, -4, -1, -3, 5, 7$