Prova Scritta di Linguaggi di Programmazione I

01/07/2009

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. Si rappresenti il P-code relativo alla funzione `fmem`, cioè la sezione di codice Pascal all'interno del riquadro.

```
program esercizio;
  const  IMAX = 20;  UNKNOWN = 0;
  type   history = array[ 0 .. IMAX ] of integer;
  ...    ...

  function fmem( n: integer; var h: history ) : integer;

  begin
    if  h[n] = UNKNOWN  then
      begin
        if  n < 2  then  h[n] := 1
                   else  h[n] := fmem( n-2, h ) + fmem( n-1, h )
      end;
    fmem := h[n];
  end;


  function fact( n: integer ) : integer;
    var   i: integer;  h: history;
  begin
    for  i := 0  to  n  do  h[i] := UNKNOWN;
    fact := fmem( n, h )
  end;

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

```
fmem: ENT     0                IXA     1                IND
      LDA     0     5          LDC   int     1          LDC   int     1
      IND                      STO                      SUB   int
      LDA     0     4          UJP   end                LDA     0     5
      IND                else: LDA     0     5          IND
      IXA     1                IND                      CUP     2  fmem
      IND                      LDA     0     4          ADD   int
      LDC   int     0          IND                      STO
      EQL   int                IXA     1          end:  LDA     0     0
      FJP   end                MST     1                LDA     0     5
      LDA     0     4          LDA     0     4          IND
      IND                      IND                      LDA     0     4
      LDC   int     2          LDC   int     2          IND
      LES   int                SUB   int                IXA     1
      FJP  else                LDA     0     5          IND
then: LDA     0     5          IND                      STO
      IND                      CUP     2  fmem          RETF
      LDA     0     4          MST     1
      IND                      LDA     0     4
```

Prova Scritta di Linguaggi di Programmazione I

01/07/2009

2. Sia $PE_{L_1}^{L_2}$ un valutatore parziale di L_2 scritto in L_1. Sia $[\![P]\!]$ la funzione calcolata dal programma P, $\forall P$. Si stabiliscano un programma Q scritto in L_2 ed un R per cui la valutazione $[\![PE_{L_4}^{L_3}]\!](R, Q)$ abbia senso e produca un compilatore da L_1 ad L_2.

3. Sia G la grammatica individuata dalle produzioni

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

pensata per generare il linguaggio $L := \{v \mid v \in \{\mathtt{a}, \mathtt{c}\}^*,\ v = v^R,\ \#(\mathtt{a}, v) \geq \#(\mathtt{c}, v)\}$, dove w^R è la stringa w rovesciata e $\#(x, w)$ è il numero di occorrenze del simbolo x dentro w.

La grammatica G è ambigua?

Si diano le stringhe di L di lunghezza ≤ 4; l'albero di parsing di una stringa di $\mathcal{L}(G) \setminus L$; una stringa di $L \setminus \mathcal{L}(G)$. Qualora non fosse possibile si giustifichi il perchè.

> Non si riesce a generare 3 `c` consecutive, visto che l'unico modo per generare `c` passa per le regole di `A` che usano poi, per forza, quelle di `S` che inseriscono necessariamente delle `a`, quindi `aacccaa` $\in L \setminus \mathcal{L}(G)$.
>
> Viceversa si può dimostrare per induzione che $\mathcal{L}(G) \subseteq L$.

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 destra a sinistra*, argomenti chiamate *da destra a sinistra* e indici vettori inizianti da 0:

```
int v[3]={data_di_nascita},  i=1, j=2;
int mess(ref int z, int j) {
 while ( (v[j++] += v[j]--) < 70 )
   write(v[j]);
 v[--i] -= ++v[--j] - z;
 return i+j;
}
write(mess(v[j--],i) + 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 è (molto fortuitamente) consistente.

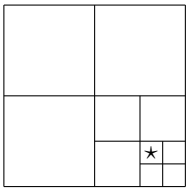
6. Rappresentando QuadTrees mediante il tipo di dato

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

si scriva una funzione Haskell `insertLogo` che dati i QuadTrees di due immagini q_l, q_p ed un QuadTree "maschera" a valori booleani, costruisce il QuadTree dell'immagine risultante inserendo la figura q_l all'interno del quadrante marcato $\star$ di q_p scegliendo i pixel di q_l o q_p in corrispondenza del valore True o False della maschera.

Prova Scritta di Linguaggi di Programmazione I

01/07/2009



Ad esempio

```
let m = Q ct cf ct cf
    cf = C False
    ct = C True
in insertLogo (C 4) m (C 1)
```

restituisce Q (C 1) (C 1) (C 1) (Q (C 1) (C 1) (C 1) (Q (Q (C 4) (C 1) (C 4) (C 1)) (C 1) (C 1) (C 1))).

```
insertLogo logo mask (Q q1 q2 q3 (Q r1 r2 r3 (Q s1 s2 s3 s4)))
 = buildNSimplify q1 q2 q3 qq
   where
     qq = buildNSimplify r1 r2 r3 rr
     rr = buildNSimplify xx s2 s3 s4
     xx = insertPict logo mask s1
insertLogo logo mask (Q q1 q2 q3 (Q r1 r2 r3 s@(C _)))
 = insertLogo logo mask (Q q1 q2 q3 (Q r1 r2 r3 (Q s s s s)))
insertLogo logo mask (Q q1 q2 q3 r@(C _))
 = insertLogo logo mask (Q q1 q2 q3 (Q r r r (Q r r r r)))
insertLogo logo mask r@(C _)
 = insertLogo logo mask (Q r r r (Q r r r (Q r r r r)))
```

con `buildNSimplify` e `insertPict` soluzioni degli appositi esercizi dell'eserciziario.

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

```
data (Eq a) => Tree a = V | N a [Tree a]
```

si scriva un predicato `iscorrect` che determina se un albero è un albero di parsing secondo le regole di una grammatica codificata mediante una funzione che, dato un simbolo, restituisce la lista delle possibili espansioni (stringhe di simboli) secondo le produzioni. Ad esempio è True

```
let r S = [ [], [A, S] ]
    r A = [ [] ]
in iscorrect r (N S [N A [], N S []])
```

8. Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma espresso in un linguaggio C-like con scoping *specificato a programma*, *shallow 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};
static-scope int H(ref int y) {
  int z = y--;
  x += z++;  write(x,y,z);
  return z--;
}
dynamic-scope int F(valres int v, int R(ref int)) {
    static-scope void G(name int w) {
        write(--x);  y[1] += w-x++; write(x,y);
```

```
    }
    int x = 3;
    G( v-- + R(v) );  return v--;
}
write(F(y[x++], H)); write(x,y);
```

$2, 8, 5, 7, 9, 1, 10, 7, 5, 3, 1, 10, 4$