Prova Scritta di Linguaggi di Programmazione II

17/06/2009

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 codice della macchina di Warren relativo al seguente programma in Prolog.

```
add( X, nil , bst(X, nil , nil) ).

add( X, bst(X,L,R), bst(X,L,R) ).

add( X, bst(Y,L,R), bst(Y,T,R) ) :- X < Y, add( X, L, T ).

add( X, bst(Y,L,R), bst(Y,L,T) ) :- Y < X, add( X, R, T ).
```

Le ‘‘ significano operazione uguale a quella a dx.

```
add/3:  try_me_else        add'        add':   retry_me_else      add"
        get_var      X4      A1                ‘‘
        get_str   nil/0      A2                get_str   bst/3      A2
        get_str   bst/3      A3
        uni_val              X4                ‘‘
        uni_var              X5                ‘‘
        uni_var              X6                ‘‘
        get_str   nil/0      X5                get_str   bst/3      A3
        get_str   nil/0      X6                uni_val              X4
                                               uni_val              X5
                                               uni_val              X6
        proceed                                ‘‘

add":   retry_me_else      add*        add*:   trust_me
        alloc                 3                ‘‘
        get_var      Y1       A1               ‘‘
        get_str   bst/3       A2               ‘‘
        uni_var               X4               ‘‘
        uni_var               Y2               uni_var              X5
        uni_var               X5               uni_var              Y2
        get_str   bst/3       A3               ‘‘
        uni_val               X4               ‘‘
        uni_var               Y3               uni_val              X5
        uni_val               X5               uni_var              Y3
        put_val      Y1       A1               put_val      X4      A1
        put_val      X4       A2               put_val      Y1      A2
        call                 </2               ‘‘
        put_val      Y1       A1               ‘‘
        put_val      Y2       A2               ‘‘
        put_val      Y3       A3               ‘‘
        call              add/3                ‘‘
        dealloc                                ‘‘
```

2. Si stabilisca (argomentando opportunamente) se il seguente programma è lineare destro/sinistro, duplicante, eliminante, collassante; se è constructor-based (specificando anche quali funzioni lo sono) e se è ortogonale.

```
f (A y:_:zs) = y : f y zs
f [y, A x] | y =:= B y = h y x
g (_:xs) (f [A _]) = xs
```

Prova Scritta di Linguaggi di Programmazione II

17/06/2009

	sx	dx	dup	el	col
R1	•		•	•	
R2	•		•		
R3	•	•		•	•
TRS	sì	no	sì	sì	sì

Il simbolo f *è* constructor based mentre g *non* lo è, dato che f sta nei pattern di R3. Quindi il TRS *non* è constructor based.

IL TRS non è ortogonale, R1 ed R2 si sovrappongono (su f [A v,A w]).

3. Si calcoli $T_P^{ca} \uparrow 3$ per il programma P ottenuto aggiungendo al programma dell'Esercizio 1 la clausola

   g (X) < f (X, _) .

$$
\begin{aligned}
&T_P^{ca} \uparrow 1 = \{lt(g(A), f(A,B)),\\
&\quad add(A, bst(A,B,C), bst(A,B,C)),\\
&\quad add(A, nil, bst(A, nil, nil))\}\\
&T_P^{ca} \uparrow 2 = T_P^{ca} \uparrow 1 \cup \{\\
&\quad add(f(A,B), bst(g(A), C, bst(f(A,B), D, E)), bst(g(A), C, bst(f(A,B), D, E))),\\
&\quad add(f(A,B), bst(g(A), C, nil), bst(g(A), C, bst(f(A,B), nil, nil))),\\
&\quad add(g(A), bst(f(A,B), bst(g(A), C, D), E), bst(f(A,B), bst(g(A), C, D), E)),\\
&\quad add(g(A), bst(f(A,B), nil, C), bst(f(A,B), bst(g(A), nil, nil), C))\}\\
&T_P^{ca} \uparrow 3 = T_P^{ca} \uparrow 2 \cup \{\\
&\quad add(f(A,B), bst(g(A), C, bst(g(A), D, bst(f(A,B), E, F))),\\
&\qquad bst(g(A), C, bst(g(A), D, bst(f(A,B), E, F)))),\\
&\quad add(f(A,B), bst(g(A), C, bst(g(A), D, nil)), bst(g(A), C, bst(g(A), D, bst(f(A,B), nil, nil)))),\\
&\quad add(g(A), bst(f(A,B), bst(f(A,C), bst(g(A), D, E), F), G),\\
&\qquad bst(f(A,B), bst(f(A,C), bst(g(A), D, E), F), G)),\\
&\quad add(g(A), bst(f(A,B), bst(f(A,C), nil, D), E), bst(f(A,B), bst(f(A,C), bst(g(A), nil, nil), D), E))\}
\end{aligned}
$$

4. Sia dato il seguente programma Curry

```
f x Nothing _ = x
f [] (Just _) z = [Just z]
f _ y@(Just _) True = [y]
```

   Dopo averlo esplicitamente tipato, si stabilisca se è induttivamente sequenziale e si calcolino i primi 2 livelli delle derivazioni di *needed* narrowing per il termine `f (f [x] y z) y z where x,y,z free`.

   Si dica inoltre se le posizioni selezionate nei vari passi delle derivazioni calcolate sono *Basic*.

Nel Definitional Tree di f c'è bisogno necessariamente di un nodo OR, quindi f non è induttivamente sequenziale.

5. Si consideri il seguente frammento di codice sintatticamente ammissibile sia in Haskell che in Curry.

```
lookup _ [] = Nothing
lookup x ((y,v):ys)
  | x == y    = Just v
  | otherwise = lookup x ys
```

Il tipo di dato inferito nei due linguaggi è lo stesso?

Ci si può aspettare di ottenere i medesimi risultati nei 2 linguaggi usando la funzione su dati di tipo `[(Int,a)]`?

Se sì sebbene si abbiano gli stessi risultati, potrebbe avere un qualche effetto pratico una eventuale riscrittura del codice?

Se no si mostri come andrebbe riscritta la versione Curry per ottenere gli stessi risultati o si spieghi perché ciò non sia possibile.

6. Si vuole risolvere il seguente gioco matematico. Data una matrice M di numeri interi $n \times 5$ ed un numero intero "goal" v_g, si deve restituire un'espressione aritmetica "sequenziale" della forma

   $$((((v_1\ o_1\ v_2)\ o_2\ v_3)\ o_3\ v_4)\ o_4\ v_5)$$

   dove $\{o_1, \ldots, o_4\} = \{+, -, \cdot, \div\}$ mentre i v_i sono 5 numeri ognuno scelto da una colonna *distinta* di M (cioè non devono esistere $i \neq j, k, p \neq q$ per cui $v_i = M_{pk}$ e $v_j = M_{qk}$).

   L'espressione cercata deve essere quella, fra tutte le espressioni sequenziali, il cui valore si avvicina maggiormente, in valore assoluto, al goal v_g (in caso di piú espressioni con medesimo valore una qualsiasi).

   L'operazione $\div$ va calcolata in virgola fissa, arrotondando al secondo decimale (e quindi tutti i valori intermedi dell'espressione son pure da considerare in virgola fissa).

   Si faccia attenzione al fatto che l'espressione $e = (167 \div 331) \cdot 374 - 379 + 248$, a causa di questa regola, non valuta a $57.6948\ldots$ ma a 56 e quindi e è una soluzione esatta per il goal 56 e la matrice

   $$\begin{pmatrix} 36 & 49 & 79 & 20 & 22 \\ 174 & 85 & 189 & \mathbf{167} & 187 \\ \mathbf{248} & 210 & 234 & 293 & 298 \\ 314 & \mathbf{374} & \mathbf{379} & 382 & \mathbf{331} \end{pmatrix}$$

   Codificando le espressioni aritmetiche mediante il tipo di dato

   ```
   data Expr = N Int | O Op Expr Int
   data Op = Add | Mul | Sub | Div
   ```

   si scriva una funzione Curry che dato un intero v_g e la matrice M, codificata come si ritiene piú opportuno, restituisce la soluzione (o una qualunque tra quelle equivalenti).

7. Si scriva un predicato PROLOG `bst2stream/2` che, dato un albero binario di ricerca t, i cui elementi (in ordine) siano $x_1, \ldots, x_n$ costruisce il termine `stream`$(x_1, [x_2 - x_1, \ldots, x_n - x_{n-1}])$. A titolo di esempio, `bst2stream(bst(5,bst(2,void,void), bst(6,void,void)),Xs)` restituisce la soluzione `Xs = stream(2,[3,1])`.