Prova Scritta di Linguaggi di Programmazione II

28/03/2008

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

```
avl( e, z ).

avl( t(n,L,R), s(H) ) :- avl( L, H ), avl( R, H ).

avl( t(n,L,R), s(s(H)) ) :- avl( L, s(H) ), avl( R, H ).

avl( t(n,L,R), s(s(H)) ) :- avl( L, H ), avl( R, s(H) ).
```

```
avl/2:   try_me_else          avl1
         get_str      e/0     A1
         get_str      z/0     A2
         proceed
avl1:    retry_me_else        avl2
         alloc                2          ⎫
         get_str      t/3     A1         ⎪
         uni_var              X3         ⎪
L':      uni_var              X4         ⎪ ⋆
R':      uni_var              Y1         ⎪
         get_str      n/0     X3         ⎭
         get_str      s/1     A2
H':      uni_var              Y2
         put_val      X4      A1
         put_val      Y2      A2
         call                 avl/2
         put_val      Y1      A1         ⎫
         put_val      Y2      A2         ⎪ ⋆⋆
         call                 avl/2      ⎪
         dealloc                         ⎭

avl2:    retry_me_else        avl3
         [⋆]                             ⎫
         get_str      s/1     A2         ⎪
         uni_var              X5         ⎪
         get_str      s/1     X5         ⎪ ⋆⋆⋆
H":      uni_var              Y2         ⎪
         put_val      X4      A1         ⎭
         put_str      s/1     A2
         set_val              Y2
         call                 avl/2
         [⋆⋆]
avl3:    trust_me
         [⋆⋆⋆]
         put_val      Y2      A2
         call                 avl/2
         put_val      Y1      A1
         put_str      s/1     A2
         set_val              Y2
         call                 avl/2
         dealloc
```

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

```
f x y []       | x=:=y = [x]
f x y (_:zs)   | y+x>0 = x:zs
g (f x 0 zs)   | x<0   = x:zs
```

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

Il TRS non è constructor based ($f \in \mathcal{D} \cap \mathcal{C}$).

IL TRS non è ortogonale, visto che R2 ed R3 si sovrappongono.

3. Sia dato il seguente programma Curry

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

Dopo averlo esplicitamente tipato, si stabilisca se è induttivamente sequenziale e si calcolino tutte le derivazioni di *needed* narrowing per il termine `f x y (f x y z) where x,y,z free`.

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

Le risposte (delle implementazioni di Curry) sono

```
{y = [v], z = Just vs} Just (v:v:vs)
{x = True, y = []} z
```

Le derivazioni *non sono* tutte basic.

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

```
palindrome xs = xs =:= reverse xs
```

Si assuma di aver esteso il Prelude di Haskell con la definizione (=:=) = (==).

Si indichi il tipo di dato inferito nei due linguaggi.

Come si comportano i due linguaggi chiamando `palindrome` su `"aba"` e su `"abc"`?

Si possono usare in input termini non-ground? E in caso quali risultati vengono forniti?

5. Sia $f_1, \ldots, f_i, \ldots$ la successione di Fibonacci.

Scrivere un predicato `fiboddlist/2` in PROLOG per costruire, a partire da un numero intero n, la lista $[f_1, f_3, \ldots, f_{2n-1}]$.

6. Preso un gruppo $r_1, \ldots, r_n$ di rettangoli si vuole determinare la massima sequenza di incapsulamento, cioè la più lunga sequenza $r_{x_1}, \ldots, r_{x_k}$ per cui il rettangolo r_{x_i} è strettamente contenuto nel rettangolo $r_{x_{i+1}}$, eventualmente ruotando i rettangoli di $\pi/2$.

Si scriva una funzione Curry `maxEncapsulation` che data una lista di coppie che rappresenta i rettangoli determina la (una) massima sequenza di incapsulamento. Ad esempio

```
maxEncapsulation [(6,4),(5,7),(8,4),(2,2)] = [(2,2),(6,4),(7,5)]
```

7. Si calcoli $T_P^{ca} \uparrow 3$ per il programma

```
avl(v,z).
avl(n(N,L,R), s(H)) :- avl(L, H),avl(R, H).
avl(n(N,L,R), s(s(H))) :- avl(L, s(H)), avl(R, H).
avl(n(N,L,R), s(s(H))) :- avl(L, H), avl(R, s(H)).

p(n(X,L,R)) :- L=n(X,R,_), avl(n(X,L,_),N), avl(R,N).
```

$T_P^{ca} \uparrow 1 = \{avl(v, z)\}$
$T_P^{ca} \uparrow 2 = T_P^{ca} \uparrow 1 \cup \{avl(n(A, v, v), s(z))\}$
$T_P^{ca} \uparrow 3 = T_P^{ca} \uparrow 2 \cup \{avl(n(A, v, n(B, v, v)), s(s(z))), avl(n(A, n(B, v, v), v), s(s(z))),$
$avl(n(A, n(B, v, v), n(C, v, v)), s(s(z)))\}$