Prova Scritta di Linguaggi di Programmazione II

11/07/2008

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.

```
part( _, [], [], [] ).

part( X, [Y|A], [Y|B], C ) :- Y =< X, part( X, A, B, C ).

part( X, [Y|A], B, [Y|C] ) :- Y > X, part( X, A, B, C ).
```

part/4:	try_me_else	$part_1$					
	get_var	X_5	A_1				
	get_str	null/0	A_2				
	get_str	null/0	A_3				
	get_str	null/0	A_4				
	preceed						
$part_1$:	retry_me_else		$part_2$	$part_2$:	trust_me		
	alloc		4		”		
(X)	get_var	Y_1	A_1		”		
	get_str	cons/2	A_2		”		
(Y)	uni_var		X_5		”		
(A)	uni_var		Y_2		”		
	get_str	cons/2	A_3	(B)	get_var	Y_3	A_2
	uni_val		X_5		get_str	cons/2	A_4
(B)	uni_var		Y_3		uni_val		X_5
(C)	get_var	Y_4	A_4	(C)	uni_var		Y_4
	put_val	X_5	A_1		”		
	put_val	Y_1	A_2		”		
	call		=</2		call		>/2
	put_val	Y_1	A_1		”		
	put_val	Y_2	A_2		”		
	put_val	Y_3	A_3		”		
	put_val	Y_4	A_4		”		
	call		part/4		”		
	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, G y]          | x =:= G z = [z,y]
f (K x : _ : xs)                = x : f xs
g (x:xs) (y:ys) = f (y:xs) ++ f (y:x:ys)
```

3. Sia dato il seguente programma Curry

```
data T a b = A | B a | C b a

f [x] A           = []
f xs (C True y)   = y:xs
f [] (C False _)  = []
```

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

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

Le risposte (delle implementazioni di Curry) sono

```
{y = C True v} v:v:x
{x = [], y = C False v} []
```

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

```
data P = Z | S P
data BST a = V | N a (BST a) (BST a)

bstIns x V = N x V V
bstIns x t@(N y _ _) | x=:=y = t
bstIns x (N y l r) | lt x y = N y (bstIns x l) r
bstIns x (N y l r) | lt y x = N y l (bstIns x r)

lt Z (S _)     = success
lt (S x) (S y) = lt x y
```

   Si assuma di aver esteso il Prelude di Haskell con le definizioni (=:=) = (==) e success = **True**.

   Si indichi il tipo di dato inferito nei due linguaggi.

   Come si comportano i due linguaggi per le queries `bstIns Z (N Z V V)` e `bstIns (S Z) (N Z V V)`?

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

5. Sia $\max_n t$ il più grande dei valori strettamente minori di n di un BST di interi t (se ne esistono), si scriva una funzione in Prolog `boundedMaximum/3` 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]`

6. Data la seguente definizione del tipo di dato astratto (polimorfo)

```
data Grph a = G [a] (a->[a])
```

   dove per ogni nodo (della lista dei nodi) la funzione restituisce la lista dei nodi adiacenti. Si scriva un predicato Curry `isConnected` per determinare se il grafo è connesso. Ad esempio

```
let f 1 = [2]
    f 2 = [3,1]
    f 3 = [1]
in isConnected $ G [1..3] f
```

   restituisce Success.

7. Si determini il programma Q per definire i predicati `=<` e `>` su numeri codificati con notazione di Peano (`0`/0 e `s`/1). Successivamente si calcoli $T_P^{ca} \uparrow 3$ per il programma P ottenuto aggiungendo Q al programma dell'Esercizio 1

```
0 =< A.
s(A) =< s(B) :- A =< B.
s(A) > 0.
s(A) > s(B) :- A > B.
```

$$
\begin{aligned}
T_P^{ca} \uparrow 1 = \cup\{ & s(A) > 0, \\
& 0 =< A, \\
& part(A, [], [], [])\} \\
T_P^{ca} \uparrow 2 = T_P^{ca} \uparrow 1 \cup \{ & s(s(A)) > s(0), \\
& s(0) =< s(A), \\
& part(0, [s(A)], [], [s(A)]), \\
& part(A, [0], [0], [])\} \\
T_P^{ca} \uparrow 3 = T_P^{ca} \uparrow 2 \cup \{ & s(s(s(A))) > s(s(0)), \\
& s(s(0)) =< s(s(A)), \\
& part(s(0), [s(s(A)), 0], [0], [s(s(A))]), \\
& part(s(0), [s(s(A))], [], [s(s(A))]), \\
& part(0, [s(A), s(B)], [], [s(A), s(B)]), \\
& part(0, [s(A), 0], [0], [s(A)]), \\
& part(s(A), [s(0), 0], [s(0), 0], []), \\
& part(s(A), [s(0)], [s(0)], []), \\
& part(0, [0, s(A)], [0], [s(A)]), \\
& part(A, [0, 0], [0, 0], [])\}
\end{aligned}
$$