Prova Scritta di Linguaggi di Programmazione II

17/07/2009

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

```
brep( z, [0] ).

brep( s(z), [1] ).

brep( s(s(N)), [0|B] ) :- half( s(s(N)), Q, z ), brep( Q, B ).

brep( s(s(N)), [1|B] ) :- half( s(s(N)), Q, s(z) ), brep( Q, B ).
```

2. Si stabilisca se il seguente programma è lineare sx/dx, duplicante, eliminante, collassante; se è constructor-based (specificando anche quali funzioni lo sono) e se è ortogonale. Si specifichi quali regole sono in overlap (mostrandone un testimone).

```
app []     ys = ys
app (x:xs) ys = x : app xs ys
p (app xs [x]) = x:x:xs
q [x,x] ys = app ys (x:ys)
```

	sx	dx	dup	el	col
R1	•	•			•
R2	•	•			
R3	•		•		
R4			•		
TRS	no	no	sì	no	sì

I simboli `q` e `app` *sono* constructor based mentre `p` *non* lo è, dato che `app` sta nel pattern di R3.

IL TRS non è ortogonale, visto che non è lineare sinistro.

R1 ed R3 si sovrappongono (su app [] [v]).

R2 ed R3 si sovrappongono (su app (v:vs) [w]).

3. Si scriva un programma Q che definisce il predicato `half/3` che dato un numero in notazione di Peano restituisce quoziente e resto della divisione per due.

   Si calcoli $T_P^{ca} \uparrow 3$ per il programma P ottenuto unendo Q con il programma dell'Esercizio 1

```
half(z,z,z).
half(s(z),z,s(z)).
half(s(s(X)),s(Y),Z) :- half(X,Y,Z).
```

$$T_P^{ca} \uparrow 1 = \{brep(s(z),[1]),\ brep(z,[0]),\ half(s(z),z,s(z)),\ half(z,z,z)\}$$

$$T_P^{ca} \uparrow 2 = T_P^{ca} \uparrow 1 \cup \{half(s(s(s(z))),s(z),s(z)),\ half(s(s(z)),s(z),z)\}$$

$$T_P^{ca} \uparrow 3 = T_P^{ca} \uparrow 2 \cup \{brep(s(s(s(z))),[1,1]),\ brep(s(s(z)),[0,1]),\ half(s(s(s(s(s(z))))),s(s(z)),s(z)),\ half(s(s(s(s(z)))),s(s(z)),z)\}$$

Prova Scritta di Linguaggi di Programmazione II

17/07/2009

4. Sia dato il seguente programma Curry

```
g []     _           = Nothing
g (x:xs) (Just y)    = Just (f (x:y:xs) (Right (g xs Nothing)))

f []     (Left y)    = y
f (_:xs) (Left _)    = f xs (Left True)
f [x,_]  _           = x
f (x:xs) (Right Nothing) = f xs (Left x)
```

Dopo averlo esplicitamente tipato, si stabilisca quali funzioni sono induttivamente sequenziali e si calcolino le derivazioni di *Needed* narrowing per il termine

```
f xs (Right (g xs x)) where x,xs free
```

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

> Per i tipi si vedano i tipi inferiti da un'implementazione di Curry.
>
> Nel Definitional Tree di f c'è bisogno necessariamente di un nodo OR (la terza regola è in overlap sia con la seconda che con la quarta regola), quindi f non è induttivamente sequenziale. La funzione g ammette un Definitional Tree privo di nodi OR.

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

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

lkup _ (V y) = Right y
lkup x (N y _ _) | x=:=y = Left x
lkup x (N y _ r) | lt y x = lkup x r
lkup x (N y l _) | lt x y = lkup x l

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.

> Per i tipi si vedano i tipi inferiti da qualunque implementazione dei 2 linguaggi.

Preso t = `N (S Z) (V 1) (V 2)`, come si comportano i due linguaggi per le queries `lkup (S Z)` t e `lkup Z` t?

Si possono usare in input termini non-ground? E in caso quali risultati vengono forniti per una query con sole variabili libere?

Se ha senso si riscriva la versione di un linguaggio per ottenere gli stessi risultati di quella dell'altro (su queries ammissibili da entrambi i linguaggi) o si spieghi perché ciò non sia possibile. Oppure si riscriva se avesse un qualche effetto pratico per queries ammissibili solo in un linguaggio.

6. Mediante la codifica ad albero chiamata "Quad Tree" si codificano immagini quadrate il cui lato sia una potenza di 2. Se l'immagine è omogenea la si codifica, indipendentemente dalle sue dimensioni, con una foglia contenente il colore. Se l'immagine è eterogenea allora si utilizza un nodo i cui figli contengono le codifiche dei quadranti sup-sx, sup-dx, inf-sx, inf-dx, rispettivamente.

   Si scriva un predicato PROLOG `upLimit/3` che dato un colore ed un QuadTree (ground) ne costruisce uno che codifica l'immagine i cui pixels sono limitati al colore c (pixel originale se il colore è $< c$, c altrimenti).

Si discuta cosa potrebbe essere fatto (se possibile) per far funzionare il precedente predicato su termini non necessariamente ground.

7. Si vuole risolvere il seguente gioco: data una pulsantiera luminosa $n \times n$ con alcuni pulsanti accesi, si deve trovare (se esiste) un **insieme** di mosse che spenga l'intera pulsantiera. Una mossa consiste nel premere uno dei pulsanti. Premendo un pulsante viene invertito il suo stato (da acceso diventa spento e viceversa) e quello dei quattro pulsanti ad esso adiacenti (sopra, sotto, destra e sinistra), sempre che ci siano.

   A titolo d'esempio, la configurazione $\begin{bmatrix} \boxed{1} & 2 & 3 \\ 4 & 5 & 6 \\ \boxed{7} & \boxed{8} & 9 \end{bmatrix}$ ammette come soluzione l'insieme $\{1, 5, 8, 9\}$.

   Codificando i dati come si ritiene piú opportuno, si scriva una funzione Curry che, data una configurazione iniziale della pulsantiera, restituisce un **insieme** di mosse che risolvono il gioco.