Prova Scritta di Linguaggi di Programmazione II

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.

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

```
hanoi( [B], S,D,_, m(B,S,D) ).

hanoi( [B|H], S,D,T, h(m(B,S,D),L,R) ) :-
    hanoi( H, S,T,D, L ), hanoi( H, T,D,S, R ).
```

```
hanoi/5 :   try_me_else     hanoi'          get_str     h/3     A5
            get_str  cons/2     A1          uni_var             X7
            uni_var             X6          uni_var             X8
            uni_var             X7          uni_var             Y5
            get_str  null/0     X7          get_str     m/3     X7
            get_var      X8     A2          uni_val             X6
            get_var      X9     A3          uni_val             Y2
            get_var     X10     A4          uni_val             Y3
            get_str     m/3     A5          put_val      Y1     A1
            uni_val             X6          put_val      Y2     A2
            uni_val             X8          put_val      Y4     A3
            uni_val             X9          put_val      Y3     A4
            proceed                         put_val      X8     A5
                                            call           hanoi/5
hanoi' :    trust_me                        put_val      Y1     A1
            alloc                5          put_val      Y4     A2
            get_str  cons/2     A1          put_val      Y3     A3
            uni_var             X6          put_val      Y2     A4
            uni_var             Y1          put_val      Y5     A5
            get_var      Y2     A2          call           hanoi/5
            get_var      Y3     A3          dealloc
            get_var      Y4     A4
```

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.

```
p n xs = elem xs > n
elem (app [x,_] y _) = x+y/x
app []     y ys = y:ys
app (x:xs) y ys = x : app xs y ys
```

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

I simboli `p` e `app` *sono* constructor based mentre `elem` *non* lo è, dato che `app` sta nel pattern di R2.

IL TRS non è ortogonale, visto che R2 ed R4 si sovrappongono (su app [v,u] w ws).

Prova Scritta di Linguaggi di Programmazione II

01/07/2009

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

$$T_P \uparrow 1 = \{hanoi([A], B, C, D, m(A, B, C))\}$$
$$T_P \uparrow 2 = T_P \uparrow 1 \cup \{hanoi([A, B], C, D, E, h(m(A, C, D), m(B, C, E), m(B, E, D)))\}$$
$$T_P \uparrow 3 = T_P \uparrow 2 \cup \{hanoi([A, B, C], D, E, F, h(m(A, D, E), h(m(B, D, F), m(C, D, E), m(C, E, F)), h(m(B, F, E), m(C, F, D), m(C, D, E))))\}$$

4. Sia dato il seguente programma Curry

```
data P = Z | S P
data Q a = H a | K (Q a)

f x (S y@(S _)) = f x y
f (K x) (S Z)   = x
f x@(H _) (S Z) = K x

g (K (H _)) = H Nothing
g (H x)     = K (H $ Just x)
```

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 (g (f x (S y))) y where x,y free`.

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

Nei Definitional Tree di f e g non c'è bisogno necessariamente di nodi OR, quindi sia f che g sono induttivamente sequenziali e tutto il TRS lo è.

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
lookup x ((y,v):ys)
  | otherwise = lookup x ys
```

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 per le queries

1. `lookup 2 [(1,'a'), (1,'b')]` e
2. `lookup 1 [(1,'a'), (1,'b')]`?

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 Curry per ottenere gli stessi risultati di quella Haskell (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 Curry.

Si riscriva anche la versione Haskell se avesse un qualche effetto pratico.

Query 1, Haskell e Curry: Nothing

Query 2, Haskell: Just 'a'

Query 2, Curry: {} **Just** 'a', {} **Just** 'b', {} **Nothing**

In Haskell *non* si possono usare termini non-ground! Nel caso di Curry si possono usare queries non ground, si faccia girare la query `lookup x xs where x,xs free` per vedere le soluzioni generate.

La versione Haskell conviene cambiarla togliendo la testa superflua.

La versione Curry *va* riscritta, visto che genera soluzioni sbagliate. Si può rinunciare al funzionamento su query non-ground usando guardie booleane e correggerla. Altrimenti, se non si vuole perdere la generazione di istanze, non si può correggere in Curry standard, dato che non si ha la primitiva "complementare" di `=:=`. In MCC si può usare invece `=/=`.

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 superiore-sinistro, superiore-destro, inferiore-sinistro, inferiore-destro, rispettivamente. Chiamiamo *QT-termini* termini costruiti usando i costruttori `c/1` e `q/4`.

   Si scrivano due predicati PROLOG:

   1. `isProper/1` che determina se un QT-termine *ground* è un Quad Tree ben formato (cioè è una codifica secondo la regola). Ad esempio `isProper(q(c(1),c(1),c(1),c(1)))` è falso.
   2. `simplify/2` che dato un QT-termine *ground* restituisce il corrispondente Quad Tree ben formato. Ad esempio `simplify(q(c(1),c(1),c(1),c(1)),X)` restituisce `{ X = c(1) }`.

   Si discuta cosa potrebbe essere fatto (se possibile) per far funzionare i precedenti predicati su QT-termini non necessariamente ground.

7. Si vuole risolvere una generalizzazione del gioco del 15 con scacchiera $n \times n$. Codificando le mosse con il data-type **data** Move = Up | Dwn | Lft | Rght, si scriva una funzione Curry che dato n ed una configurazione iniziale della scacchiera determini una sequenza di mosse per risolvere il gioco.

   Si codifichi la scacchiera come si ritiene più opportuno.