# Prova Scritta di Linguaggi di Programmazione II

07/09/2009

> 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.

```
sorted( [] ).

sorted( [_] ).

sorted( [X,Y|V] ) :- X =< Y, sorted( [Y|V] ).
```

```
sorted/1 :  try_me_else     sorted'
            get_str  null/0      A1
            proceed

sorted' :   retry_me_else   sorted"
            get_str  cons/2      A1
            uni_var              X2
            uni_var              X3
            get_str  null/0      X3
            proceed

sorted" :   trust_me
            alloc                 2
            get_str  cons/2      A1
            uni_var              X3
            uni_var              X4
            get_str  cons/2      X4
            uni_var              Y1
            uni_var              Y2
            put_val      X3      A1
            put_val      Y1      A2
            call              leq/2
            put_str  cons/2      A1
            set_val              Y1
            set_val              Y2
            call           sorted/1
            dealloc
```

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).

```
R1.   lookup _ Void = Nothing
R2.   lookup x (Node x v _ _) = Just v
      lookup x (Node y _ l r)
R3a.    | x < y      = lookup x l
R3b.    | otherwise = lookup x r
```

Considerando le guardie parte del corpo.

| | sx | dx | dup | el | col |
|---|---|---|---|---|---|
| R1 | • | • | | • | |
| R2 | | • | | • | |
| R3a | • | | • | • | |
| R3b | • | • | | • | |
| TRS | no | no | sì | sì | no |

L'unico simbolo definito `lookup` è constructor based.

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

R2 ed R3a, così come R2 ed R3b, si sovrappongono (su **lookup** x (Node x v y z)).

R3a ed R3b si sovrappongono (su **lookup** x (Node y v w z)).

3. Si scriva un programma $Q$ che definisce il predicato `=<` su numeri in notazione di Peano.

   Si calcoli $T_P^{ca} \uparrow 4$ per il programma $P$ ottenuto unendo $Q$ con il programma dell'Esercizio 1

# Prova Scritta di Linguaggi di Programmazione II

07/09/2009

```
0 =< Y.
s(X) =< s(Y) :- X =< Y.
```

$$
\begin{aligned}
T_P^{ca} \uparrow 1 &= \{0 =< A, \quad sorted([A]), \quad sorted([])\} \\
T_P^{ca} \uparrow 2 &= T_P^{ca} \uparrow 1 \cup \{s(0) =< s(A), \quad sorted([0, A])\} \\
T_P^{ca} \uparrow 3 &= T_P^{ca} \uparrow 2 \cup \{s(s(0)) =< s(s(A)), \quad sorted([s(0), s(A)]), \quad sorted([0, 0, A])\} \\
T_P^{ca} \uparrow 4 &= T_P^{ca} \uparrow 3 \cup \{s(s(s(0))) =< s(s(s(A))), \quad sorted([s(s(0)), s(s(A))]), \\
&\qquad sorted([s(0), s(0), s(A)]), \quad sorted([0, s(0), s(A)]), \\
&\qquad sorted([0, 0, 0, A])\}
\end{aligned}
$$

4. Sia dato il seguente programma Curry

```
f x        Nothing   _ = x
f []       (Just _) z = [Just z]
f (x:xs) (Just _) z = f xs (Just z) 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*.

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 quarta regola è in overlap sia con la seconda che con la terza regola), quindi $f$ non è induttivamente sequenziale.

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

```
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 l _) | x < y = lkup x l
lkup x (N y _ r) | x > y = lkup x r
```

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 1 (V 'a') (V 'b')`, come si comportano i due linguaggi per le queries `lkup 1` $t$ e `lkup 0` $t$?

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. Rappresentando Alberi "generici" con i costruttori `void/0` e `node/2`, dove `node` mantiene il dato di un nodo e la lista dei figli, si scriva una funzione `diameter/2` che determina il diametro di un albero. Il diametro di un albero è la lunghezza del massimo cammino fra due nodi, indipendentemente dall'orientamento degli archi.

   A titolo di esempio, `diameter( node(f, [ node(c, [node(b,[node(a,[])]), node(d, [node(e,[])]) ]) ]), D)` restituisce `D = 4`.

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

```
diameterGen(T, D) :- treefoldr_aggr1_aggr2(T, _,D).

treefoldr_aggr1_aggr2(void, -1,-1).
treefoldr_aggr1_aggr2(node(_,Ts), H,D) :-
    foldr_g_tfr(Ts, HM1,HM2,DM),
    H is 1+HM1,
    D is max(DM,2+HM1+HM2).

foldr_g_tfr([],-1,-1,-1).
foldr_g_tfr([T|Ts], NHM1,NHM2,NDM) :-
    foldr_g_tfr(Ts, HM1,HM2,DM),
    treefoldr_aggr1_aggr2(T,HT,DT),
    updateH(HT,HM1,HM2, NHM1,NHM2),
    NDM is max(DM,DT).

updateH(H,HM1,HM2, HM1,HM2) :- H =< HM2.
updateH(H,HM1,_,    H,   HM1) :- H > HM1.
updateH(H,HM1,HM2, HM1,H)    :- H > HM2,H =< HM1.
```

   Non ci sono possibilità in Prolog standard di avere predicati di disuguaglianza numerica che funzionino su variabili libere, e in questo programma non si può fare a meno del controllo di disuguaglianza.

7. Si vuole risolvere il seguente gioco: in uno spazio $n$ dimensionale, dato un ipercubo che ha sui vertici dei pulsanti luminosi, con alcuni pulsanti inizialmente accesi, si deve trovare (se esiste) un **insieme** di mosse che spenga tutto l'ipercubo. Una mossa consiste nel premere uno dei pulsanti. Premendo un pulsante viene invertito il suo stato (da acceso diventa spento e viceversa) e quello degli $n$ pulsanti ad esso adiacenti (lungo gli spigoli).

   A titolo d'esempio, per $n = 2$ la configurazione $\begin{bmatrix} \boxed{1} & \boxed{2} \\ 3 & 4 \end{bmatrix}$ ammette come soluzione l'insieme $\{3, 4\}$.

   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.