

Εξέταση Γλώσσες Προγραμματισμού

Θοδωρής Φρίζος Παπαρρηγόπουλος
el18040

Θέμα 1

a) $\langle S \rangle \rightarrow (\langle L \rangle) \rightarrow (\langle L \rangle, \langle S \rangle) \rightarrow (\langle L \rangle, \langle S \rangle, \langle L \rangle) \rightarrow (\langle S \rangle, a, \langle S \rangle) \rightarrow$
 $((\langle L \rangle), a, (\langle L \rangle)) \rightarrow ((\langle L \rangle, \langle S \rangle), a, (a)) \rightarrow ((\langle S \rangle, a), a, (a)) \rightarrow ((a, a), a, (a)).$

b)

Η γραμματική δεν είναι *ambiguous* καθώς η γραμματική αυτή είναι αριστερά προσεταιριστική και ορίζει την προτεραιότητα στα “,” και “()”.

c) Η γραμματική παράγει είτε σκέτο a είτε *tuples* με τερματικό μόνο το a διαχωρισμένα με κόμα. Δηλαδή (a, a, a) ή $(a, (a), (a, a, a))$

Θέμα 2

a.

```
fun common.. x y =  
  let  
    fun aux(h1::t) (h2::t2) prefix =  
      if h1 = h2 then aux t1 t2 (h1::prefix)  
      else (prefix, (h1::t), (h2::t2))  
    | aux s1 s2 prefix = (prefix, s1, s2)  
  in aux x y []  
end;
```

Όταν δεν έχουμε ίδια στοιχεία επιστρέφει το *prefix* μαζί με τις άλλες λίστες

Όταν μια λίστα γίνει κενή ουσιαστικά πηγαίνει στο κάτω *pattern matching* και επιστρέφει το αποτέλεσμα.

b. Αν του περάσουμε για λίστα το $[1,1,2,3,4,5]$ τότε θα αληθεύσει. Θα *fail* αρι το 2ο *unique* όμως θα μπει στο 3ο.

`unique([]).`

`unique([Item | Rest]):-`

`\+ member(Item, Rest), unique(Rest).`

c.

To AM είναι 040 $\rightarrow AM1 = AM3 = 0$ και $AM2 = 4$

γ1. 4 17 0 42 4 17

γ2. 4 17 0 42 42

d. To AM είναι 040 $\rightarrow AM1 = AM3 = 0$ και $AM2 = 4$

δ1. 4 0 4 0

δ2. 4 4 0 0

`f(4) -> g(4,0) -> print (4,0) -> x = 0 -> print(4) -> print(0)`

```
f(4) -> g(4,0)-> print(4, 4) -> x_f = 0 -> print(x_f) = 0 -> print(x) = 0
```

Θέμα 3

```
fun nest x =  
  let  
    fun help 0 = 1  
      | help n = x + n + help (n-1)  
    in help 5  
  end;
```

Αν δεν υπήρχε το nesting link η συνάρτηση θα αποτύγχανε επειδή δεν θα μπορούσε αναδρομικά η help να χρησιμοποιήσει το x.

Θέμα 4

```
datatype 'a tree = Leaf | Node of 'a * 'a tree * 'a tree
```

```
fun trim Leaf = [Leaf]  
  | trim Node(value, left, right) =  
  let  
    fun is_different(value, Leaf) = false  
      | is_different(value, Node(v,l,r)) =  
        if value mod 2 = 1 then true  
        else false  
  
    fun help(Leaf, acc) = acc  
      | help(Node(n, l, r), acc) =  
        let  
          val left = is_different(n, l)  
          val right = is_different(n,r)  
  
        in  
          end  
  
  in  
    (help(tree, []))  
  end;
```

ο σκοπό ήταν η help να υπολογίζει το δέντρο ξεκινώντας από έναν κόμβο μέχρι να κόψει δηλαδή πχ όταν το left γίνει true τότε βάζουμε στο acc το υπόλοιπο που σχηματίζεται με το help (right) και αντίστοιχα με το δεξί. Δηλαδή θα έχουμε 4 περιπτώσεις για τα left & right

και μετά θα έχουμε μια άλλη συνάρτηση η οποία θα διαπερνά το δέντρο και μόλις βρει έναν κόμβο που δεν ικανοποιεί την συνθήκη του is_different θα τρέχει την help για να σχηματίσει το εκάστοτε δέντρο και θα συνεχίζει

Θέμα 5

a)

$n(_,_,_)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(A), \text{integer}(B), \text{integer}(C)$, $M1$ is $\text{max}(A,B)$, Res is $\text{max}(M1,C)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(A)$, $\text{integer}(B)$, $M1$ is $\text{max}(A,B)$, $\text{find_max}(C, M2)$, Res is $\text{max}(M1,M2)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(A)$, $\text{integer}(C)$, $M1$ is $\text{max}(A,C)$, $\text{find_max}(B, M2)$, Res is $\text{max}(M1,M2)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(B)$, $\text{integer}(C)$, $M1$ is $\text{max}(B,C)$, $\text{find_max}(A, M2)$, Res is $\text{max}(M1,M2)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(A)$, $\text{find_max}(B,M1)$, $\text{find_max}(C,M2)$, $M3$ is $\text{max}(M1,M2)$, Res is $\text{max}(A,M3)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(B)$, $\text{find_max}(A,M1)$, $\text{find_max}(C,M2)$, $M3$ is $\text{max}(M1,M2)$, Res is $\text{max}(B,M3)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{integer}(C)$, $\text{find_max}(A,M1)$, $\text{find_max}(B,M2)$, $M3$ is $\text{max}(M1,M2)$, Res is $\text{max}(C,M3)$.

$\text{find_max}(n(A,B,C), \text{Res})$:- $\text{find_max}(A,M1)$, $\text{find_max}(B,M2)$, $\text{find_max}(C,M3)$, $M4$ is $\text{max}(M1,M2)$, Res is $\text{max}(M4, M3)$.

$\text{maximize}(n(A,B,C), \text{MaxTree})$:-

$\text{find_max}(n(A,B,C), \text{Max})$,

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(A), \text{integer}(B), \text{integer}(C)$, $\text{MaxTree} = n(\text{Max}, \text{Max}, \text{Max})$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(A), \text{integer}(C)$, $\text{updateTree}(B, T, \text{Max})$, $\text{MaxTree} = n(\text{Max}, T, \text{Max})$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(A), \text{integer}(B)$, $\text{updateTree}(C, T, \text{Max})$, $\text{MaxTree} = n(\text{Max}, \text{Max}, T)$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(C), \text{integer}(B)$, $\text{updateTree}(A, T, \text{Max})$, $\text{MaxTree} = n(T, \text{Max}, \text{Max})$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(A)$, $\text{updateTree}(B, T1, \text{Max})$, $\text{updateTree}(C, T2, \text{Max})$, $\text{MaxTree} = n(\text{Max}, T1, T2)$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(B)$, $\text{updateTree}(A, T1, \text{Max})$, $\text{updateTree}(C, T2, \text{Max})$, $\text{MaxTree} = n(T1, \text{Max}, T2)$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{integer}(C)$, $\text{updateTree}(B, T1, \text{Max})$, $\text{updateTree}(A, T2, \text{Max})$, $\text{MaxTree} = n(T2, T1, \text{Max})$.

$\text{updateTree}(n(A,B,C), \text{MaxTree}, \text{Max})$:- $\text{updateTree}(A, T, \text{Max})$, $\text{updateTree}(B, T1, \text{Max})$, $\text{updateTree}(C, T2, \text{Max})$, $\text{MaxTree} = n(T, T1, T2)$.

b)
 $n(_,_,_)$.
 $is_odd_sum(n(A,B,C)):- integer(A),integer(B),integer(C), Sum\ is\ A + B + C, Sum\ mod\ 2\ =:=\ 1.$

```
unoddsun(n(A,B,C), Term):- integer(A),integer(B),integer(C),
(
    is_odd_sum(n(A,B,C)) -> Term is 17;
    Term = n(A,B,C)
).
unoddsun(n(A,B,C), Term):- integer(A),integer(B), unoddsun(C, T1),
(
    integer(T1), is_odd_sum(T1) ->
    (
        is_odd_sum(n(A,B,17))-> Term is 17;
        Term = n(A,B,17)
    );
    Term = n(A,B,T1)
).
unoddsun(n(A,B,C), Term):- integer(A),integer(C), unoddsun(B, T1),
(
    integer(T1), is_odd_sum(T1) ->
    (
        is_odd_sum(n(A,17,C))-> Term is 17;
        Term = n(A,17,C)
    );
    Term = n(A,T1,C)
).
unoddsun(n(A,B,C), Term):- integer(B),integer(C), unoddsun(A, T1),
(
    integer(T1), is_odd_sum(T1) ->
    (
        is_odd_sum(n(17,B,C))-> Term is 17;
        Term = n(17,B,c)
    );
    Term = n(T1,B,C)
).
```

```
unoddsun(n(A,B,C), Term):- integer(A), unoddsun(B, T1), unoddsun(C,T2). % check for 17
solutions and decide
unoddsun(n(A,B,C), Term):- integer(B), unoddsun(A, T1), unoddsun(C,T2),
unoddsun(n(T1,B,T2) Term). % check for 17 solutions and decide
unoddsun(n(A,B,C), Term):- integer(C), unoddsun(B, T2), unoddsun(A,T1),
unoddsun(n(T1,T2,C) Term). % check for 17 solutions and decide
unoddsun(n(A,B,C), Term):- unoddsun(A, T1), unoddsun(B,T2), unoddsun(C,T3),
unoddsun(n(T1,T2,T3) Term). % check for 17 solutions and decide
```

c) Ναι μπορούμε! Εστω συνάρτηση που επιστρέφει το αποτέλεσμα. Βελτιώνουμε με αυτή τα υποδέντρα και έπειτα να ανακατασκευάσουμε το δέντρο μας.

Θέμα 6

```
def sliding(list, K):
    sums = dict()

    sum = 0
    for i in range(K):
        sum += list[i]
    sums[sum] = 1

    for i in range(K, len(list)):
        sum += list[i] - list[i - K]
        if sum in sums:
            sums[sum] += 1
        else:
            sums[sum] = 1
    ans = -1
    max_sum = 0
    for a in sums:
        if sums[a] > ans:
            ans = sums[a]
            max_sum = a
        elif sums[a] == ans:
            if max_sum < a:
                ans = sums[a]
                max_sum = a

    print(max_sum, ans)

sliding([1,4,2,3,2,1,3,4,2],4)
sliding([1, 4, 2, 3, 2, 1, 3, 4, 2], 3)
```