ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΜΑΘΗΜΑ: ΕΡΓΑΣΤΗΡΙΟ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ (Prolog)
ΕΞΑΜΗΝΟ: ΣΤ΄- Εαρινό 2019-20
ΚΑΘΗΓΗΤΗΣ: ΔΗΜΟΣΘΕΝΗΣ .ΣΤΑΜΑΤΗΣ

ΑΣΚΗΣΗ ΠΡΑΞΗΣ 3: ΛΙΣΤΕΣ ΚΑΙ ΑΝΑΔΡΟΜΗ

Ενα είδος σύνθετου όρου που χρησιμοποιείται πάρα πολύ στην Prolog είναι η λίστα. Μια λίστα είναι μια διατεταγμένη σειρά από άτομα ή σύνθετους όρους ή και από άλλες λίστες. Μια λίστα, της οποίας είναι γνωστά τα στοιχεία που την αποτελούν, συμβολίζεται από αγκύλες `[]` μέσα στις οποίες γράφονται αυτά τα στοιχεία χωρισμένα με κόμματα, π.χ.

```
[a, b, c, d]          (λίστα με τέσσερα ατομικά στοιχεία: a,b,c,d)
[a, 3, name(kostas,antoniou)]  (λίστα με τρία στοιχεία a, 3 και το σύνθετο όρο name(kostas, antoniou)
[[a,b], [c,d]]        (λίστα με στοιχεία δύο άλλες λίστες: [a,b] και [c,d])
```

Η λίστα στην Prolog είναι δυναμική δομή δεδομένων και σε αντιστοιχία και με τις άλλες γλώσσες, όπου ορίζεται η συνδεδεμένη λίστα, δεν μπορεί να υπάρχει άμεση πρόσβαση στο n-στό στοιχείο της. Άμεση πρόσβαση υπάρχει μόνον στην **κεφαλή** (δηλαδή το πρώτο στοιχείο μιας λίστας) και στην **ουρά** (δηλαδή στη λίστα που απομένει αν αφαιρέσουμε την κεφαλή). Ο συμβολισμός

```
[H | T]
```

χωρίζει μια λίστα σε κεφαλή (`H`) και ουρά (`T`). Η κεφαλή `H` της λίστας είναι το πρώτο στοιχείο της. Η ουρά `T` είναι πάντα λίστα και προκύπτει αν από την αρχική λίστα αφαιρέσουμε την κεφαλή. Αν η λίστα αποτελείται από ένα μόνο στοιχείο τότε η ουρά της είναι η κενή λίστα `[]`. Για παράδειγμα,

```
?- [H|T] = [a,b,c,d].
H = a
T = [b,c,d]

?- [H|T] = [[a,b],[c,d]].
H = [a, b]
T = [[c, d]]

?- [H|T] = [a].
H = a
T = []
```

Αν μια λίστα είναι κενή τότε η ουρά της δεν υπάρχει και το κόψιμο της λίστας αυτής σε κεφαλή και ουρά αποτυγχάνει:

```
?- [H|T] = [].
No (False)
```

Θέματα συμβολισμού: Διαφορά μεταξύ "," (κόμμα) και "|" (κάθετος)

Το σύμβολο "," (κόμμα) διαχωρίζει τα στοιχεία μιας λίστας. Το σύμβολο "|" (κάθετος) διαχωρίζει την κεφαλή από την ουρά. Παραδείγματα:

```
?- [A,B,C] = [1,2,3].
A=1
B=2
C=3

?- [A|B|C] = [1,2,3].
No (False)

?- [A|[B|C]] = [1,2,3].
A=1
B=2
C=[3]
```

Υπάρχει δυνατότητα να παραθέσουμε κάποια στοιχεία στην αρχή της λίστας διαχωρίζοντάς τα απλά με κόμμα, όπως π.χ. [X1,X2,X3|T]. Η μορφή αυτή είναι η απλοποιημένη της: [X1|[X2|[X3|T]]]. Και στις δύο περιπτώσεις η λίστα που αναπαριστάνεται έχει τουλάχιστον 3 στοιχεία.

Αναδρομή

Μία βασική τεχνική που χρησιμοποιείται στον προγραμματισμό με Prolog είναι η αναδρομή, δηλαδή η δυνατότητα ενός στόχου να έχει ως υποστόχο τον εαυτό του με άλλα ορίσματα. Προφανώς αν ένας στόχος έχει ως υποστόχο τον εαυτό του με τα ίδια ακριβώς ορίσματα τότε θα έχουμε άπειρη αναδρομή. Έτσι π.χ. ο κανόνας

```
son(X,Y) :-
      son(X,Y).
```

και όλοι οι παρόμοιοι κανόνες δεν πρέπει να χρησιμοποιούνται αφού οδηγούν σε *"stack overflow"* και κόλλημα του συστήματος.

Βασική τεχνική της αναδρομής σε λίστες

Έστω ότι έχω να λύσω ένα πρόβλημα για μια λίστα L. Σε πάρα πολλά προβλήματα χωρίζω τη λίστα αυτή σε κεφαλή και ουρά L=[H|T] και θεωρώ ότι έχω λύσει το πρόβλημα για την ουρά T. Ζητάω να βρω τη σχέση μεταξύ (α) της λύσης για την ουρά και (β) της λύσης για όλη τη λίστα, δηλαδή της λύσης που προκύπτει αν προσθέσω στη λίστα και την κεφαλή H. Πρέπει φυσικά να προσθέσω και μία ή περισσότερες οριακές περιπτώσεις.

Η αναδρομή είναι ιδιαίτερα χρήσιμη στην επεξεργασία λιστών αφού η πρόσβαση στα στοιχεία μιας λίστας γίνεται μόνο μέσα από τον διαχωρισμό της σε κεφαλή και ουρά.

Παράδειγμα αναδρομικού κατηγορήματος

Το κατηγόρημα `length(X,L)` υπολογίζει το πλήθος `L` των στοιχείων της λίστας `X`.

```
length([],0).
length([H|T],L) :-
      length(T,L1),
      L is L1+1.
```

Το μήκος της πιο απλής περίπτωσης

Θέλω να βρω το μήκος L της λίστας [H|T],

έστω ότι L1 είναι το μήκος της λίστας T...

...τότε L είναι L1+1

ΤΙ ΠΡΕΠΕΙ ΝΑ ΚΑΝΕΤΕ

(α) Γράψτε το κατηγόρημα `sum(L,X)` το οποίο επιστρέφει στο `X` το άθροισμα των στοιχείων της λίστας `L` (η `L` υποτίθεται ότι αποτελείται μόνο από αριθμούς). Για παράδειγμα,

```
?- sum([3,5,11,9], X).          ?- sum([-3, 3.14, 11,9], X).
X=28                                  X=20.14
```

(β) `teleytaio(L,X)`: επιστρέφει στο X το τελευταίο στοιχείο της λίστας L. Για παράδειγμα,

```
?- teleytaio([4,a,d,2], X).          ?- teleytaio([red, blue, green], X).
X=2                                  X=green
```

(γ) `melos(X,L)`: επιστρέφει Yes αν το στοιχείο X είναι μέλος της λίστας L και No αν όχι. Για παράδειγμα,

```
?- melos(a, [1,2,3,a,b]).            ?- melos(yellow, [red, blue, green]).
Yes (True)                                 No (False)
```

(δ) Γράψτε το κατηγόρημα `element(L,N,X)` το οποίο επιστρέφει στο `X` το `N`-οστό στοιχείο της λίστας `L`. Για παράδειγμα,

```
?- element([a,b,c,d], 3, X).         ?- element([a, b, c], 1, X).
X = c                                X = a

?- element([a, [c,e], d], 2, X).     ?- element([[a,b], [c], d], 4, X).
X = [c,e]
                                     No
```

(ε) Γράψτε το κατηγόρημα `max(L,X)` το οποίο επιστρέφει στο `X` τον μέγιστο αριθμό της λίστας `L` (η `L` υποτίθεται ότι αποτελείται μόνο από αριθμούς). Για παράδειγμα,

```
?- max([13,52,12,29], X).            ?- max([-13, 14, -32, 9], X).
X=52                                 X=14
```

ΣΥΜΠΛΗΡΩΜΑΤΙΚΕΣ ΑΣΚΗΣΕΙΣ

(1) Γράψτε το κατηγόρημα `del(X,L,NL)`: Διαγράφει το στοιχείο X από τη λίστα L και επιστρέφει τη λίστα NL που δεν το περιλαμβάνει. Για παράδειγμα,

```
?- del(a,[1,2,3,a,b],L).          ?- del(blue, [red, blue, green, blue],L).
L=  [1,2,3,b]                     L = [red, green, blue] ->;

                                  L = [red, blue, green]
```

(2) Γράψτε το κατηγόρημα `delall(X,L,NL)`: Διαγράφει όλες τις εμφανίσεις του στοιχείου X από τη λίστα L και επιστρέφει τη λίστα NL που δεν περιλαμβάνει καθόλου το X. Για παράδειγμα,

```
?- delall(a,[1,α,2,3,a,b],L).     ?- delall(blue, [red, blue, green, blue],L).
L= [1,2,3,b]                      L = [red, green]
```