4. ΛΙΣΤΕΣ

Η **λίστα** (***list***) είναι μία πολύ απλή δομή δεδομένων η οποία χρησιμοποιείται ευρήτατα για τον προγραμματισμό μη-αριθμητικών εφαρμογών. Με τη βοήθεια της λίστας στην PROLOG μπορούν να αναπαρασταθούν άλλες οποιασδήποτε πολυπλοκότητας δομές δεδομένων.

Λίστα είναι μία ακολουθία

$$\alpha_1, \alpha_2, \ldots, \alpha_n$$

από μηδέν ή περισσότερα στοιχεία (οποιουδήποτε αριθμού). Για την PROLOG η λίστα είναι μία ειδική περίπτωση συναρτησιακού όρου για τον οποίο παρέχεται ειδικός συμβολισμός: τα στοιχεία της λίστας χωρίζονται μεταξύ τους με κόμματα και περικλείονται σε τετράγωνες παρενθέσεις:

$$[\alpha_1, \alpha_2, \ldots, \alpha_n]$$

Μερικά παραδείγματα λιστών δίνονται στη συνέχεια:

```
[1, 2 , 3, 4, 5]
[john, mary, nick]
[son,mon,tues,wen,thi,fri,sat]
[1, a, b, 3, f(1,2)]
[1, [4, 6], 3, [7, [8, 9], 5] ]
```

Μερικά σχόλια είναι απαραίτητα σε σχέση με τα παραπάνω παραδείγματα. Στην PROLOG τα στοιχεία μιας λίστας δεν είναι υποχρεωτικό να είναι του ίδιου τύπου. (Οπως αυτό συμβαίνει για παράδειγμα στην PASCAL.) Ενα στοιχείο μιάς λίστας μπορεί να είναι γενικά οποιοσδήποτε όρος (σταθέρα, μεταβλητή, ή σύνθετος όρος). Κατ' επέκταση τα στοιχεία μιας λίστας μπορούν να είναι με τη σειρά τους λίστες.

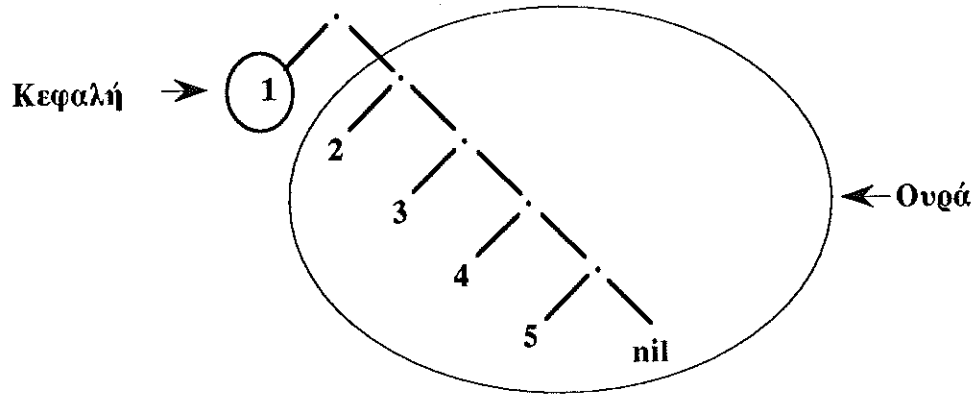
Ο ειδικός συμβολικός τρόπος γραφής της λίστας που χρησιμοποιείται στα παραπάνω παραδείγματα, δίνει μόνο την εξωτερική εμφάνιση της λίστας. Στην ουσία όμως η λίστα για την PROLOG είναι μια αναδρομική δομή δεδομένων. Αυτό μπορούμε να το κατανοήσουμε καλύτερα αν θεωρήσουμε τον εναλλακτικό τρόπο παράστασης της λίστας που αναφέραμε στην παράγραφο 2.1.3. Σύμφωνα με αυτόν η λίστα

[1, 2, 3, 4, 5]

μπορεί να παρασταθεί με τη βοήθεια του παρακάτω αναδρομικού όρου: (Η αναπαράσταση του με τη βοήθεια δέντρου δίνεται στο σχήμα 4.1.)

.(1, .(2, .(3, .(4, .(5,nil)))))

Ο όρος αυτός έχει σαν συναρτησιακό σύμβολο την "." (τελεία) και δύο ορίσματα. Το πρώτο όρισμα είναι ένα στοιχείο της λίστας και το δεύτερο όρισμα είναι με τη σειρά του (εδώ φαίνεται η έννοια του αναδρομικού όρου) μία λίστα. Στη σταθερά **nil** προσδίδουμε την ειδική σημασία, να παριστάνει την κενή λίστα, μία λίστα δηλαδή με κανένα στοιχείο. Ονομάζουμε το πρώτο όρισμα **κεφαλή *(head)*** της λίστας και το δεύτερο όρισμα **ουρά *(tail)*** της λίστας.



Σχήμα 4.1: Σχηματική αναπαράσταση λίστας με τη βοήθεια δέντρου

Με βάση τον ειδικό συμβολισμό που χρησιμοποιεί η PROLOG η κενή λίστα (**nil**) παριστάνεται με το σύμβολο "[ ]". Ο όρος **.(X,Y)**, όπου η μεταβλητή **X** αντιστοιχεί στην κεφαλή της λίστας και η μεταβλητή **Y** στην ουρά της λίστας, παριστάνεται με **[X|Y]**. Πρέπει να τονίσουμε εδώ ότι ο ειδικός αυτός συμβολισμός που χρησιμοποιείται για την αναπαράσταση της λίστας είναι απλά "συντακτική ζάχαρη" (*syntactic sugar*) του όρου **.(X,Y)** και δεν προσθέτει κανέναν επιπλέον μηχανισμό στην PROLOG.

Οι παρακάτω τρόποι γραφής μπορούν να χρησιμοποιηθούν εναλλακτικά για την αναπαράσταση της λίστας **[1,2,3,4,5]** και είναι ισοδύναμοι:

[1 | [2, 3, 4, 5]]
[1, 2 | [3, 4, 5]]
[1, 2, 3 | [4, 5]]
[1, 2, 3, 4 | [5]]
[1, 2, 3, 4, 5 | [ ]]

Τα παρακάτω παραδείγματα δείχνουν πώς με τη βοήθεια της διαδικασίας ενοποίησης, τμήματα μιας λίστας μπορούν να τοποθετήσουν μεταβλητές:
(Το σύμβολο '=' αποτελεί τον τελεστή ενοποίησης και παρέχεται σαν ενσωματωμένο κατηγόρημα.)

```
?- [p,r,o,l,o,g] = [X,Y,Z,T,L,M].
   X = p
   Y = r
   Z = o
   T = l
   L = o
   M = g

?- [p,r,o,l,o,g] = [X|Y].
   X = p
   Y = [r,o,l,o,g]
```

```
?- [p,r,o,l,o,g] = [X,Y|Z].
   X = p
   Y = r
   Z = [l,o,g]

?- [prolog] = [X|Y].
   X = prolog
   Y = [ ]

?- [[v,e,r,y],h,a,r,d] = [H|T].
   H = [v,e,r,y]
   T = [h,a,r,d]

?- [[v,e,r,y],h,a,r,d] = [[V|W],X|Y].
   V = v
   W = [e,r,y]
   X = h
   Y = [a,r,d]

?- [[my,X]|Y] = [[Z,cat],[is,here]].
   X = cat
   Y = [[is,here]] {λίστα με στοιχείο λίστα}
   Z = my
```

Επίσης έχουμε:

```
?- [1,2,3 | []] = [1,2,3].
   yes

?- [1,2,3,[]] = [1,2,3].
   no

?- [X|Y] = [ ].
```

no

?- [X,Y|Z,W] = [1,2,3,4].
{συντακτικό λάθος - η εκτέλεση δεν μπορεί να γίνει**}**

Ιδιαίτερη προσοχή πρέπει να δίνουμε (κατά κανόνα να το αποφεύγουμε) όταν μετά το σύμβολο '|' τοποθετούμε μία σταθερά, όπως φαίνεται από το παρακάτω παραδείγμα ενοποίησης:

?- [cat|dog] = [X|Y].
X = cat
Y = dog

ενώ αντίθετα:

?- [cat,dog] = [X|Y].
X = cat
Y = [dog]

Ο όρος **[cat|dog]** αναπαριστά μία λίστα με κεφαλή **cat** και που η ουρά της **dog** είναι σταθερά και όχι λίστα !!! (Εναι προφανές ότι **[cat|dog]** =/= **[cat,dog]**.)

Ενας επιπλέον ειδικός μηχανισμός, παρέχεται εναλλακτικά, από την PROLOG, για την αναπαράσταση μιάς λίστας ακεραίων, όταν αυτοί αντιστοιχούν σε ASCII κωδικούς χαρακτήρων. Λίστες αυτής της μορφής είναι στην ουσία συμβολοσειρές (*strings*). Για παράδειγμα η σειμβολοσειρά:

"PROLOG"

αποτελεί εναλλακτική αναπαράσταση της λίστας:

[80, 82, 79, 76, 79, 71]

4.1 Βασικά Κατηγορήματα για το Χειρισμό Λιστών

Ενας μεγάλος αριθμός από κατηγορήματα μπορεί να οριστεί για τον χειρισμό της λίστας. Στη συνέχεια περιγράφουμε μερικά από αυτά, που είναι ιδιαίτερα χρήσιμα σ' ένα μεγάλο πλήθος προγραμμάτων εφαρμογών της PROLOG.

4.1.1. Μέλος μιας Λίστας.

Ο έλεγχος, εάν ένα στοιχείο **Element**, αποτελεί μέλος μιας λίστας **List**, μπορεί να γίνει με τη βοήθεια του κατηγορήματος:

```
% member(Element,List)
```

Ο στόχος **member(Element,List)** αληθεύει όταν το στοιχείο **Element** εμφανίζεται στη λίστα **List**. Η επιθυμητή συμπεριφορά του κατηγορήματος αυτού φαίνεται από τα παρακάτω παραδείγματα εκτέλεσης:

Παράδειγματα εκτέλεσης:

```
?-member(3,[1,4,3,2]).
yes

?-member(5,[1,4,3,2]).
no
```

Η υλοποίηση του κατηγορήματος αυτού μπορεί να βασιστεί στην εξής διπλή παρατήρηση:

Το **X** είναι μέλος της λίστας **L**, εάν
(α) Το **X** είναι η κεφαλή της λίστας **L** ή
(β) Το **X** είναι μέλος της ουράς της λίστας **L**.

Τα παραπάνω μπορούν να γραφούν στην PROLOG με τη βοήθεια δύο φράσεων, ενός γεγονότος και ενός αναδρομικού κανόνα, ως εξής:

(M1) **member(X,[X|Tail])** .
(M2) **member(X,[Head|Tail])** :- **member(X,Tail).**

4.1.2. Συνένωση δύο Λιστών.

Το πρόβλημα εδώ είναι να οριστεί ένα κατηγόρημα

% append(List1,List2,List),

το οποίο να δημιουργεί τη λίστα **List** σαν τη συνένωση των δύο λιστών **List1** και **List2.**

Παράδειγμα εκτέλεσης:

?-append([1,2,3],[6,5],X).
X = [1,2,3,6,5]

Το κατηγόρημα **append** υλοποιείται με τη βοήθεια των εξής δύο φράσεων:

(A1) **append([ ],List,List).**
(A2) **append([H|Tail],List,[H|NTail])** **:-append(Tail,List,NTail).**

Σύμφωνα με τη φράση **(A1)**: Εάν το πρώτο όρισμα είναι η κενή λίστα και το δεύτερο όρισμα είναι μια οποιαδήποτε λίστα, τότε η λίστα που προκύπτει από την συνένωση (τρίτο ορισμα), πρέπει να είναι ακριβώς ίδια με αυτήν του δευτέρου ορίσματος.

Σύμφωνα με τη φράση **(A2)**: Εάν το πρώτο όρισμα δεν είναι η κενή λίστα τότε μπορεί να γραφεί σαν

[H|Tail]

και η λίστα που προκύπτει από τη συνένωσή της με τη **List** έχει τη μορφή:

[H|NTail]

έχει δηλαδή την ίδια κεφαλή **H** και η ουρά της προκύπτει αναδρομικά από τη συνένωση της ουράς **Tail**, της αρχικής λίστας και της **List**.

Στη συνέχεια δίνουμε μερικά επιπλέον παραδείγματα χρήσης του κατηγορήματος **append** :

```
?- append([1,2,3], [a,b,c], L).
L = [1,2,3,a,b,c]

?- append([1,[2,3,4],5],[6,[7],[ ],8], L).
L = [1,[2,3,4],5,[6,[7],[ ],8]
```

Οι απαντήσεις που δίνονται στις παρακάτω ερωτήσεις δείχνουν πώς οι δύο απλές φράσεις που υλοποιούν το κατηγόρημα **append**, μπορούν να χρησιμοποιηθούν, για να υπολογίσουν οποιοδήποτε όρισμα εάν δίνονται τα άλλα δύο:

```
?- append([1,2],[3,4],X).
X = [1,2,3,4]

?- append(X,[3,4],[1,2,3,4]).
X = [1,2]

?- append([1,2],X,[1,2,3,4]).
X = [3,4]

?- append(X,[4,3],[1,2,3,4]).
no
```

Το κατηγόρημα **append** χρησιμοποιείται με επιτυχία και όταν τα δύο πρώτα του ορίσματα είναι άγνωστα, παράγει μάλιστα όλες τις δυνατές λύσεις:

```
?- append(X, Y, [1,2,3]).
X = [ ]          Y = [1,2,3] -> ;
X = [1]          Y = [2,3] -> ;
X = [1,2]        Y = [3] -> ;
X = [1,2,3]      Y = [] -> ;
no
```

Μπορεί επίσης να χρησιμοποιηθεί με ακόμη πολυπλοκότερο τρόπο, όπως:

```
?- append(Before, [tue|After], [sun,mon,tue,wed,thi,fri,sat]).
Before = [sun,mon]
After = [wed,thi,fri,sat]
```

όπου αναζητούμε τις μέρες της εβδομάδας που προηγούνται και αυτές που ακολουθούν την ημέρα **tue** (Τρίτη) ή:

```
?- append(_, [P, tue, S | _], [sun,mon,tue,wed,thi,fri,sat]).
P = mon
S = wed
```

όπου ρωτάμε ποιά μέρα προηγείται της **tue** και ποιά ακολουθεί.

4.1.3. Διαγραφή ενός στοιχείου μιάς Λιστας.

Το πρόβλημα εδώ είναι να οριστεί ένα κατηγόρημα

```
% delete(X,List1,List2),
```

το οποίο να διαγράφει το στοιχείο **X** από τη λίστα **List1** και να δημιουργεί τη λίστα **List2**, η οποία δεν περιέχει το **X**.

Παράδειγμα εκτέλεσης:

```
?-delete(a,[b,a,s],X).
X = [b,s]
```

Το κατηγόρημα **delete** υλοποιείται με τη βοήθεια των εξής δύο φράσεων:

```
(D1)  delete(X,[X|Y],Y).
(D2)  delete(X,[Y|Z],[Y|T]) :- delete(X,Z,T).
```

Η φράση (**D1**) αναφέρεται στην περίπτωση που το στοιχείο **X** που ζητάμε να διαγραφεί βρίσκεται στην κεφαλή της λίστας. Τότε το αποτέλεσμα που προκύπτει μετά τη διαγραφή, είναι η ουρά **Y** της αρχικής λίστας.

Η φράση (**D2**) αναφέρεται στην περίπτωση που το στοιχείο **X** δεν βρίσκεται στην κεφαλή της λίστας, αλλά στην ουρά της. Τότε η τελική λίστα που θα προκύψει μετά τη διαγραφή θα έχει σαν κεφαλή **Y** την ίδια με αυτήν της αρχικής λίστας και σαν ουρά **T** αυτήν που θα προκύψει αναδρομικά από τη διαγραφή του **X** από την ουρά **Z** της αρχικής λίστας.

Το κατηγόρημα **delete** αποτυγχάνει, όταν το στοιχείο που ζητάμε να διαγραφεί δεν υπάρχει στην αρχική λίστα, όπως για παράδειγμα:

```
?- delete(1, [2,4,3], L).
no
```

Εάν το στοιχείο **X** που ζητάμε να διαγραφεί εμφανίζεται περισσότερες από μια φορές στην λίστα, τότε η σχέση **delete** που ορίστηκε έχει τη δυνατότητα να διαγράψει οποιαδήποτε εμφάνιση του **X**, κατά τη διαδικασία της οπισθοδρόμησης, άν ζητήσουμε διαφορετική λύση, όπως για παράδειγμα:

```
?- delete(1, [1, 2, 1, 3, 1], L).
L = [2, 1, 3, 1] -> ;
L = [1, 2, 3, 1] -> ;
L = [1, 2, 1, 3] -> ;
no
```

Παρατηρούμε βέβαια ότι σε κάθε εναλλακτική εκτέλεση, διαγράφεται μόνο μία εμφάνιση του στοιχείου, ενώ οι άλλες μένουν ανέπαφες. Το κατηγόρημα **total_del** που δίνουμε στη συνέχεια φροντίζει, ώστε η τελική λίστα που προκύπτει να μην περιέχει καθόλου το στοιχείο που ζητάμε να διαγραφεί.

```
(T1)   total_del(X,[],[]).
(T2)   total_del(X,[X|L],L1) :- total_del(X,L,L1).
(T3)   total_del(X,[Y|L],[Y|L1]) :- total_del(X,L,L1).
```

Για παράδειγμα:

```
?- total_del(1, [1, 2, 1, 3, 1], L).
L = [2, 3]
```

Ενδιαφέρον παρουσιάζει η χρησιμοποίηση του κατηγορήματος **delete**, όταν το πρώτο και το τρίτο του όρισμα είναι τοποθετημένα και το δεύτερο μή τοποθετημένο:

```
?- delete(4, L, [1, 2, 3]).
L = [4, 1, 2, 3] ;
L = [1, 4, 2, 3] ;
L = [1, 2, 4, 3] ;
L = [1, 2, 3, 4] ;
no
```

Στην περίπτωση αυτή χρησιμοποιούμε ουσιαστικά το κατηγόρημα **delete** για την **εισαγωγή** ενός νέου στοιχείου σε μιά λίστα. Θα μπορούσαμε να ορίσουμε την πράξη εισαγωγής **insert** ως εξής:

```
insert(X, L, L1) :- delete(X, L1, L).
```

4.1.4. Αντιστροφή των Στοιχείων μιας Λίστας.

Το πρόβλημα εδώ είναι να οριστεί το κατηγόρημα:

% reverse(List,Rlist)

το οποίο να κατασκευάζει τη λίστα **Rlist**, η οποία να περιέχει τα στοιχεία της λίστας **List** με αντίστροφη σειρά.

Παράδειγμα εκτέλεσης:

```
?-reverse([m,a,r,i,a],X).
X = [a,i,r,a,m]
```

Το κατηγόρημα **reverse** υλοποιείται με τη βοήθεια των εξής δύο φράσεων:

```
(R1)  reverse([],[]).
(R2)  reverse([H|T],L) :- reverse(T,R), append(R,[H],L).
```

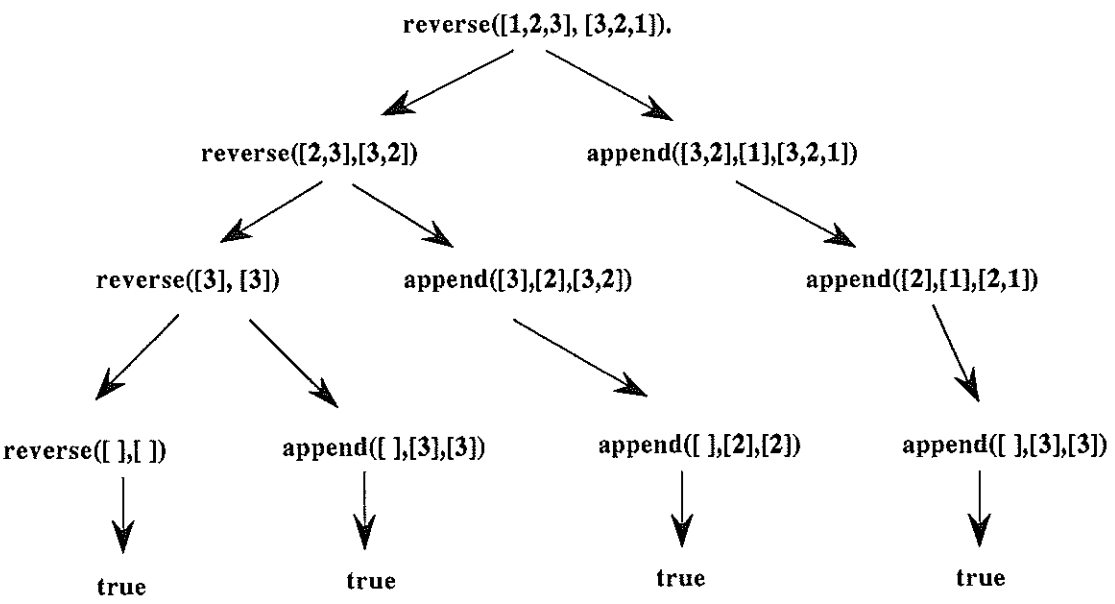
Η σημασία της φράσης (**R1**) είναι προφανής. Σύμφωνα με τη φράση (**R2**) αντιστρέφεται αναδρομικά η ουρά **T** της λίστας [**H**|**T**], για να δώσει τη λίστα **R**. Στη συνέχεια προστίθεται, με τη βοήθεια της **append** στο τέλος της **R** η κεφαλή **H** της αρχικής λίστας, για να προκύψει τελικά η **L**.

Στο σχήμα 4.2 δίνεται το δέντρο εκτέλεσης (μόνο οι AND κόμβοι) της ερώτησης:

```
?- reverse([1,2,3], [3,2,1]).
```

Ενας εναλλακτικός τρόπος υλοποίησης της αντιστροφής δίνεται με τη βοήθεια του κατηγορήματος **ireverse** που ορίζουμε παρακάτω:

```
(I1)  ireverse(X,Y) :- ireverse(X,[],Y).
(I2)  ireverse([H|T],Sofar,Y) :- ireverse(T,[H|Sofar],Y).
(I3)  ireverse([],X,X).
```



Σχήμα 4.2: Δέντρο εκτέλεσης της **reverse**

Με την **ireverse/2** η αντιστροφή γίνεται δυνατή όχι με την άμεση κλήση της append, αλλά έμεσα με τη βοήθεια της σχέσης **ireverse/3**. Η φράση **(I1)** συσχετίζει την **ireverse/2** με την **ireverse/3**. Η σχέση

% ireverse(X,Y,Z)

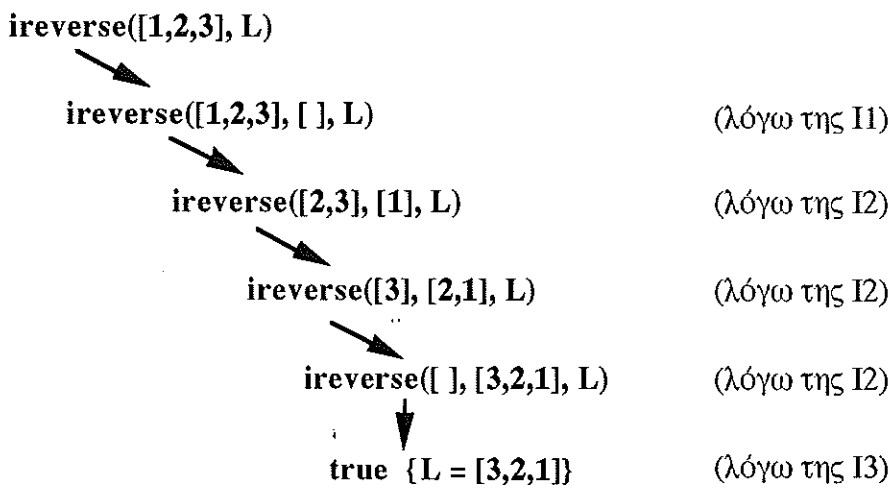
αληθεύει όταν **Z** είναι η λίστα που προκύπτει από τη συνένωση των στοιχείων της λίστας **Y**, πίσω από τα στοιχεία της λίστας που προκύπτει από την αντιστροφή της **X**.

Στη φράση **(I2)** το δεύτερο όρισμα που προστέθηκε στην **ireverse/3** παίζει το ρόλο μιας **παραμέτρου συσσώρευσης** (***accumulating parameter***), όπου συσσωρεύεται, κατά τη διάρκεια της εκτέλεσης, η "μέχρι στιγμής" αντεστραμμένη λίστα.

Ας θεωρήσουμε σαν παράδειγμα τη διαδικασία εκτέλεσηςτης ερώτησης:

?- ireverse([1,2,3],L).

που δίνεται στο σχήμα 4.3. Παρατηρούμε ότι το τρίτο όρισμα της **ireverse/3** παραμένει μή τοποθετημένο κατά τη διάρκεια της εκτέλεσης. Τοποθετείται τελικά (με τη βοήθεια της φράσης (**I3**)) με την αντεστραμένη λίστα (που αποτελεί και την απάντηση), όταν το πρώτο όρισμα γίνει η κενή λίστα.



Σχήμα 4.3 : Δέτρο εκτέλεσης της **ireverse**

Η **ireverse** είναι πιό αποτελεσματική από την **reverse**, γιατί όπως φαίνεται και από τις αντίστοιχες διαδικασίες εκτέλεσης, απαιτεί λιγότερο χώρο και χρόνο.

```
B1:     < sort([2,1],S),
        (sort([X|Y],Z) :- sort(Y,T), insert(X,T,Z)),
        { X = 2, Y = [1], S = Z } >

B2:     < (sort([1],T), insert(2,T,Z)),
        (sort([X1|Y1],Z1) :- sort(Y1,T1), insert(X1,T1,Z1))
        { X1 = 1, Y1 = [], T = Z1 }

B3:     < (sort([],T1), insert(1,T1,Z1), insert(2,Z1,Z)),
        sort([],[]),
        { T1 = [] } >

B4:     < (insert(1,[],Z1), insert(2,Z1,Z)),
        insert(X2,[],[X2]),
        { Z1 = [X2], X2 = 1 } >

B5:     <  insert(2,[1],Z),
        (insert(X3,[Y3|Z3],[Y3|T3]) :- X3>Y3, insert(X3,Z3,T3)),
        { X3 = 2, Y3 = 1, Z3 = [], Z = [1|T3] } >

B6:     < (2>1, insert(2,[],T3)),
        2>1,
        {} >

B7:     < insert(2,[],T3),
        insert(X4,[],X4),
        { X4 = 2, T3 = [2] } >

B8:     < [],[],{} >
```

Σχήμα 4.4 : Τα βήματα εκτέλεσης του στόχου ?- sort([2,1],S).

4.2 ΑΣΚΗΣΕΙΣ

4.2.1. Να γραφεί πρόγραμμα που να υπολογίζει το άθροισμα των στοιχείων μιάς λίστας ακεραίων αριθμών. (Να οριστεί το κατηγόρημα **sum(L,S)** που να επιστρέφει στο **S** το άθροισμα των στοιχείων της λίστας **L**).

4.2.2. Να γραφεί πρόγραμμα που να υπολογίζει το μήκος μιάς λίστας. (Να οριστεί το κατηγόρημα **length(L,M)** που να επιστρέφει στο **M** το μήκος της λίστας **L**).

4.2.3. Να γραφεί πρόγραμμα που να βρίσκει το μέγιστο στοιχείο μιάς λίστας ακεραίων αριθμών. (Να οριστεί το κατηγόρημα **max(L, M)** που να επιστρέφει στο **M** το μέγιστο στοιχείο της λίστας **L**).

4.2.4. Να οριστούν τα κατηγορήματα:

- **last_of(L, X)** το οποίο να επιστρέφει στο **X** το τελευταίο στοιχείο της λίστας **L**.
- **third_of_list(L, X)** το οποίο να επιστρέφει στο **X** το τρίτο στοιχείο της λίστας **L**.
- **nth_of_list(L, N, S)** το οποίο να επιστρέφει στο **S** το **N**-στό στοιχείο της λίστας **L**. Μπορεί το κατηγόρημα αυτό να χρησιμοποιηθεί αντίστροφα για την εύρεση της Θέσης **N** ενός στοιχείου **S** στη λίστα **L**;

4.2.5. Να γραφεί πρόγραμμα που να δημιουργεί όλες τις μεταθέσεις των στοιχείων μιας λίστας. (Να υλοποιθεί το κατηγόρημα **permutate(L,P)**, το οποίο δοθείσης μιας λίστας **L**, να δημιουργεί την λίστα **P** σαν μετάθεση της **L**.)

Παράδειγμα εκτέλεσης:

```
?- permutate([a,b,c],P).
P = [a,b,c]  ->;
P = [a,c.b]  ->;
```

P = [b,a,c] ->;

. . .

4.2.6. Να γραφεί πρόγραμμα το οποίο να ελέγχει εάν σε μία λίστα υπάρχουν δύο διαδοχικά στοιχεία με την ίδια τιμή.

4.2.7. Θεωρήστε ότι ένα σύνολο μπορεί να αναπαρασταθεί με τη βοήθεια μιας λίστας, στην οποία κάθε στοιχείο εμφανίζεται μία και μόνον μία φορά.

(α) Ορίστε το κατηγόρημα **union(X,Y,Z)** το οποίο δοθέντων δύο συνόλων **X** και **Y** δημιουργεί την ένωσή τους και την επιστρέφει στο **Z**.

Παράδειγμα εκτέλεσης: **?- union([1,4,2], [2,3,4], A).**
A = [1,4,2,3]

(β) Ορίστε το κατηγόρημα **tomi(X,Y,Z)** το οποίο δοθέντων δύο συνόλων **X** και **Y** δημιουργεί την τομή τους και την επιστρέφει στο **Z**.

Παράδειγμα εκτέλεσης: **?- tomi([1,4,2], [2,3,4], A).**
A = [2,4]

4.2.8. Δίνεται μία λίστα ακεραίων αριθμών Στη λίστα αυτή ένας αριθμός **X** μπορεί να επαναλαμβάνεται περισσότερες από μία φορές. Να γραφεί ένα κατηγόρημα **change(X, Y, L)** το οποίο να αντικαθιστά όλες τις εμφανίσεις του αριθμού **X** στη λίστα **L** με τον **Y**.

4.2.9. Ορίστε το κατηγόρημα **length(L, N)** το οποίο να επιστρέφει στο **N** το μήκος της λίστας **L**. Χρησιμοποιώντας τα κατηγορήματα **length** και **append** (αλλά μόνον αυτά) ορίστε το κατηγόρημα **sublist(L1, N, L2)**, το οποίο να επιστρέφει τη λίστα **L2** που να αποτελείται από **N** συνεχόμενα στοιχεία της λίστας **L1**.

παράδειγμα εκτέλεσης: **?- sublist([a, b, c, d], 2, L).**
L = [a, b] --> ;

L = [b, c] --> ;
L = [c, d]

Επίσης ορίστε το κατηγόρημα **last_items(L1, N, L2)**, το οποίο να επιστρέφει στη λίστα **L2** τα **N** τελευταία στοιχεία της λίστας **L1**.

4.2.10. Σε μία λίστα, κάποια στοιχεία πιθανά να εμφανίζονται δύο ή περισσότερες φορές. Γράψτε ένα κατηγόρημα **dups_out(DL,L)** το οποίο δέχεται τη λίστα **DL** και δημιουργεί την **L** η οποία περιέχει όλα τα στοιχεία της **DL** από μία μόνο φορά.

Παράδειγμα εκτέλεσης: **?- dups_out([4,1,2,1,3,2,1],L).**
L = [4,1,2,3]

4.2.11. Ορίστε το κατηγόρημα **buble_sort(L1, L2)** το οποίο να ταξινομεί τη λίστα **L1** σύμφωνα με τον αλγόριθμο των φυσαλίδων και να επιστρέφει την ταξινομημένη λίστα στη **L2**.

4.2.12. Μία λίστα μπορεί να περιέχει σαν στοιχεία της άλλες λίστες. Ας ονομάσουμε μία τέτοια λίστα "λίστα πολλών επιπέδων". Να οριστεί το κατηγόρημα **flatten(List,Flist)** το οποίο να μετατρέπει τη λίστα πολλών επιπέδων **List** σε μία λίστα **Flist** η οποία να είναι "ενός επιπέδου" .

Παραδείγματα εκτέλεσης: **?- flatten([1,[3,2],[ ],8], X).**
X = [1,3,2,8]
?- flatten([1,[3,[2,4,5],[9,8]],[3,2]],X).
X = [1,3,2,4,5,9,8,3,2]