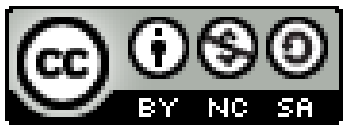
ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

Υπολογιστική Λογική και Λογικός Προγραμματισμός

Ενότητα 5: Λογικός Προγραμματισμός: Σύνθετοι Όροι, Αναδρομικοί Όροι, Λίστες

Νίκος Βασιλειάδης, Αναπλ. Καθηγητής
Τμήμα Πληροφορικής



ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

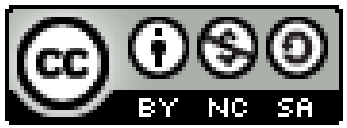
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

Λογικός Προγραμματισμός: Σύνθετοι Όροι, Αναδρομικοί Όροι, Λίστες.

Σύνθετοι Όροι

- Οι σύνθετοι όροι χρησιμεύουν στην αναπαράσταση σύνθετων οντοτήτων ενός προβλήματος

triangle(point(0,0),point(1,4),point(5,12))

  - Αναπαράσταση τριγώνου
- Από τη μορφή των σύνθετων όρων είναι φανερό πως μοιάζουν με τις σύνθετες δομές δεδομένων των συμβατικών γλωσσών προγραμματισμού
  - Π.χ. *εγγραφές* (*records*) στην Pascal ή *δομές* (*structs*) στην C

Αναδρομικοί Σύνθετοι Όροι

- Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι αναδρομικές δομές δεδομένων, δηλαδή οι σύνθετοι όροι ο οποίοι ορίζονται αναδρομικά εγκλείοντας όρους της ίδιας μορφής με τον εαυτό τους.
- Οι αναδρομικοί σύνθετοι όροι είναι ικανοί να αναπαραστήσουν
  - μη-πεπερασμένα πεδία (Π.χ. φυσικοί αριθμοί)
  - συγκεντρώσεις αντικειμένων αγνώστου (εκ των προτέρων) αριθμού στοιχείων (Π.χ. Λίστες, Σύνολα, Δένδρα).

Φυσικοί Αριθμοί ως Σύνθετοι Όροι

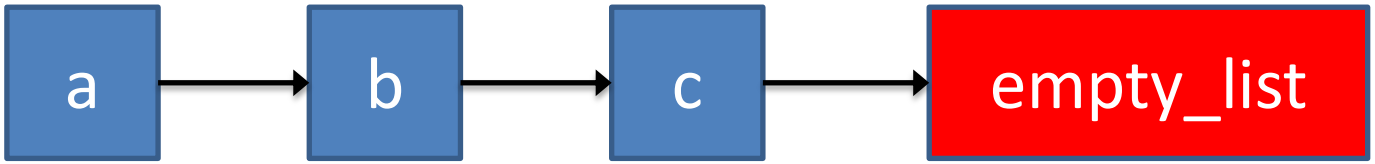
- Ένας φυσικός αριθμός μπορεί να αναπαρασταθεί ως σύνθετος όρος ως εξής:
  - Το μηδέν (0) αναπαρίσταται ως απλός όρος: **0**
  - Το ένα (1) αναπαρίσταται ως ο σύνθετος όρος: **s(0)**
    - Σημασία: *Το 1 είναι ο επόμενος αριθμός από το μηδέν*
  - Το δύο (2) αναπαρίσταται ως ο σύνθετος όρος: **s(s(0))**
  - **…**
- Να γραφεί κατηγόρημα **natural_number/1** το οποίο να πετυχαίνει όταν το όρισμά του είναι φυσικός αριθμός.

```
natural_number(0).
natural_number(s(X)) :-
    natural_number(X).
```

Άλλοι αναδρομικοί σύνθετοι όροι

Λίστες

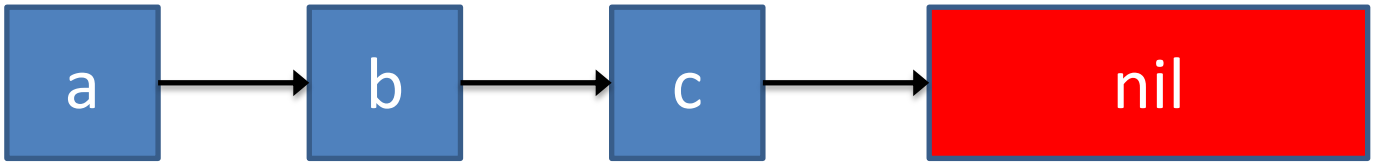
- list(first(a),
  rest(list(first(b),
  rest(list(first(c),
  rest(empty_list))))))



Άλλοι αναδρομικοί σύνθετοι όροι

Λίστες

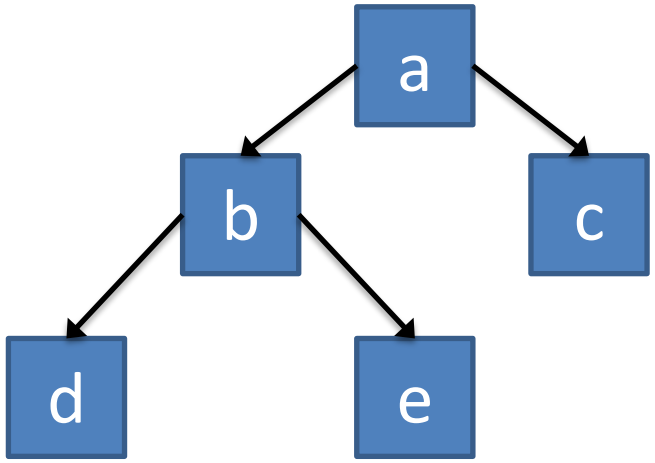
- list(first(a),
  rest(list(first(b),
  rest(list(first(c),
  rest(nil))))))



Άλλοι αναδρομικοί σύνθετοι όροι

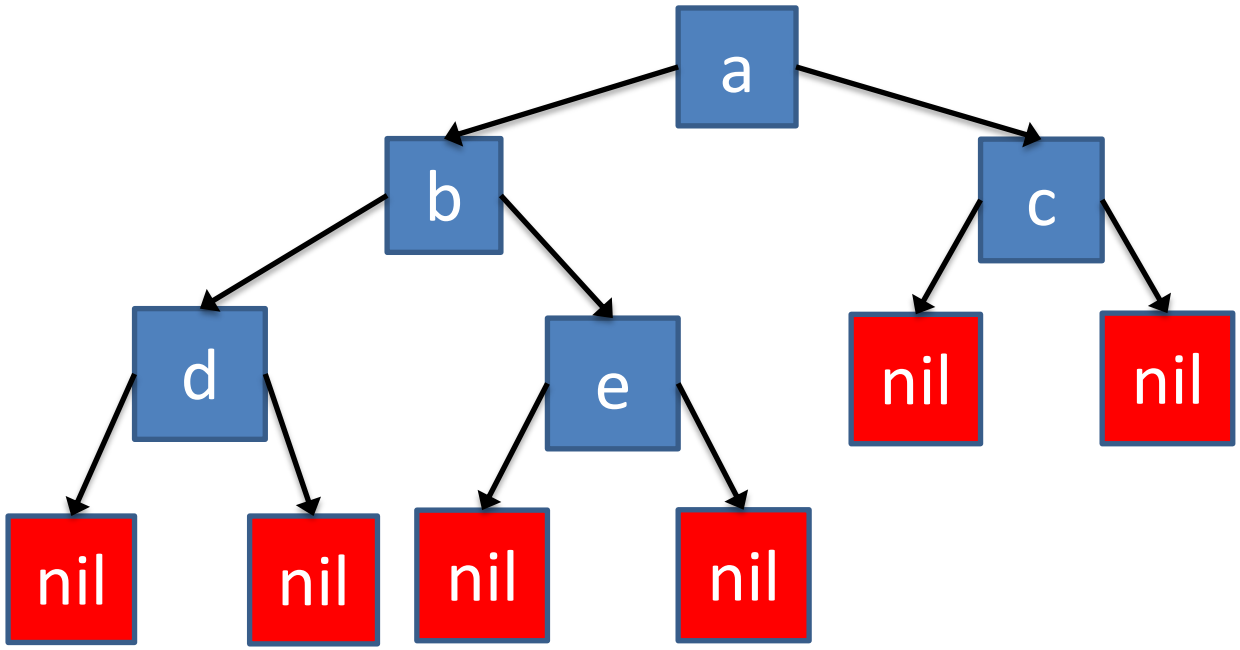
Δέντρα

- tree(data(a),
  left(tree(data(b),
  left(tree(data(d),
  left(nil),
  right(nil))),
  right(tree(data(e),
  left(nil),
  right(nil)))),
  right(tree(data(c),
  left(nil),
  right(nil))))



Άλλοι αναδρομικοί σύνθετοι όροι

Δέντρα

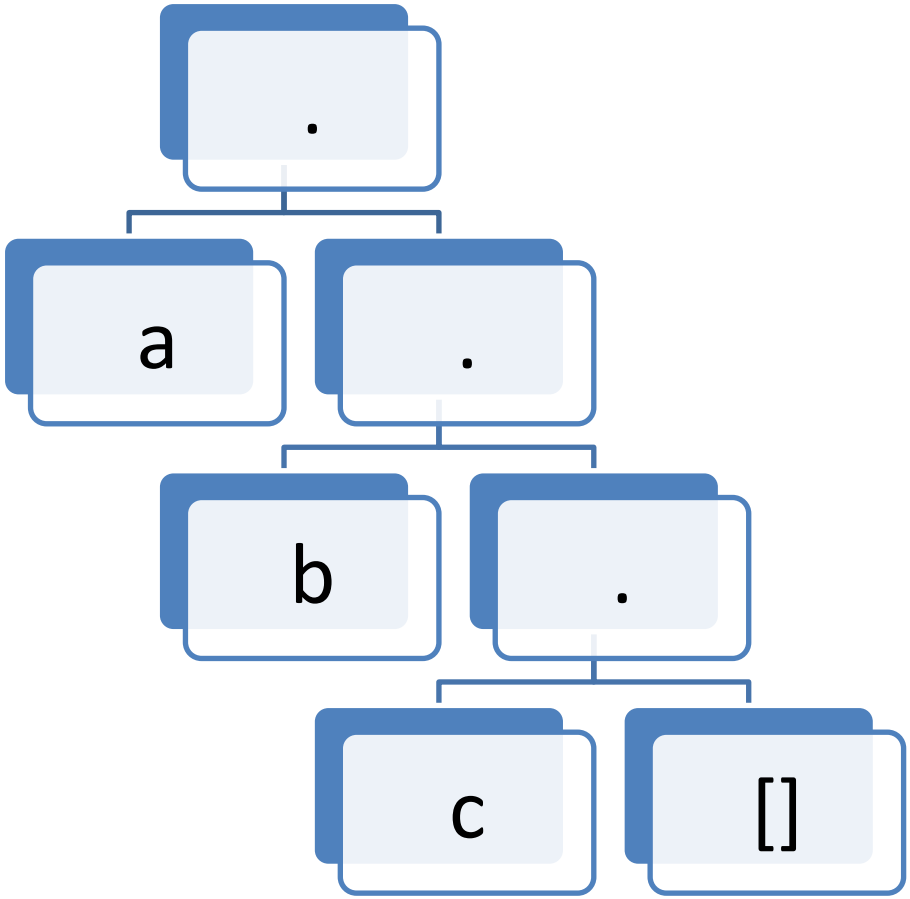


Λίστες

- Μια ειδική (πολύ σημαντική) κατηγορία αναδρομικών σύνθετων όρων είναι η *λίστα* (*list*).
- Πρόκειται για μια σύνθετη δομή που έχει συναρτησιακό σύμβολο την τελεία "." και έχει 2 ορίσματα: την κεφαλή και την ουρά
  - Η κεφαλή (head) είναι το πρώτο (first) στοιχείο της λίστας και μπορεί να είναι οποιοσδήποτε όρος, ακόμα και άλλη λίστα.
  - Η ουρά (tail) περιέχει όλα τα υπόλοιπα (rest) στοιχεία της λίστας, εκτός της κεφαλής, επομένως είναι πάντα λίστα.
- Μια λίστα μπορεί να μην έχει στοιχεία, οπότε ονομάζεται κενή λίστα (empty_list) και συμβολίζεται με [] (nil)

Παράδειγμα Λίστας

- Λίστα με 3 στοιχεία **.(a,.(b,.(c,[])))**
- Εναλλακτική αναπαράσταση **[a, b, c]**



Σύνταξη Λίστας

- Οι λίστες είναι απλή, αλλά ιδιαίτερα σημαντική δομή δεδομένων της Prolog, με την οποία μπορούν να αναπαρασταθούν συλλογές δεδομένων ως μία οντότητα.
- Μια λίστα στην Prolog αναπαρίσταται με μια ακολουθία από οποιονδήποτε αριθμό στοιχείων.
- Τα στοιχεία της λίστας τοποθετούνται μέσα σε αγκύλες "[", "]" και χωρίζονται μεταξύ τους με κόμμα ",".

Κεφαλή και Ουρά

- Η λίστα που έχει κεφαλή Η και ουρά Τ παριστάνεται ως **[H|T]**
  - Αντιστοιχεί στο σύνθετο όρο **.(H,T)**
  - Στη λίστα **[a, b, c]** κεφαλή είναι το στοιχείο **a** και ουρά η λίστα **[b,c]**.
- Η ουρά μιας λίστας είναι **πάντα** λίστα ενώ η κεφαλή μπορεί να είναι οποιοσδήποτε όρος, απλός ή σύνθετος, ακόμα και λίστα
  - Η **[[a,b],c]** έχει κεφαλή τη λίστα **[a,b]** και ουρά τη λίστα **[c]**

Παραδείγματα

Λίστα	Κεφαλή	Ουρά
[a, b, c]	a	[b, c]
[X,Y]	X	[Y]
[X|Y]	X	Y
[1]	1	[]
[]	*Δεν υπάρχει*	*Δεν υπάρχει*
[[a, b] , c]	[a,b]	[c]
[1,2|X]	1	[2 | X]
[f(a,b),[a]]	f(a,b)	[[a]]

Ταυτοποίηση λιστών

- Δύο λίστες ταυτοποιούνται, εφόσον έχουν τον ίδιο αριθμό στοιχείων και εφόσον τα αντίστοιχα στοιχεία τους μπορούν να ταυτοποιηθούν.

Λίστα 1	Λίστα 2	Τιμές μεταβλητών
[a, b, c]	[a, b, c, d]	*αποτυγχάνει*
[a, b, c]	[X, Y]	*αποτυγχάνει*
[a, b, c]	[X| Y]	{X=a, Y=[b,c]}
[a, b]	[a, X]	{X=b}
[a, b]	[a | X]	{X=[b]}
[f(1), k, [1]]	[f(X), Y, Z]	{X=1,Y=k, Z=[1]}
[f(1), k, [1]]	[f(x), Y | Z]	{X=1, Y=k, Z=[[1]] }
[[a]]	[X | Y]	{X=[a],Y=[]}

Χειρισμός Λιστών

- Ο χειρισμός των λιστών γίνεται συνήθως με αναδρομικούς κανόνες
- Ο ***αναδρομικός κανόνας :***
  - επιτελεί κάποια λειτουργία στην ***κεφαλή*** της λίστας και
  - στη συνέχεια καλεί αναδρομικά τον εαυτό του ώστε να επιτελέσει την ίδια λειτουργία και στην ***ουρά*** της λίστας.
- Συνήθως υπάρχει και ένας ***τερματικός κανόνας*** ο οποίος ασχολείται με την περίπτωση της ***κενής*** λίστας.
- Καθώς στη λίστα προσθέτουμε ή αφαιρούμε στοιχεία μόνο από την κεφαλή (LIFO), θυμίζει τη ***στοίβα*** (***stack***)

Διαπίστωση αν ένας Όρος είναι Λίστα

- Το κατηγόρημα **is_list** ελέγχει αν το όρισμα είναι λίστα
  - Ορισμός: "*Ένας όρος είναι λίστα αν είναι η κενή λίστα, ή αν βρίσκεται στη μορφή [Κεφαλή|Ουρά], όπου η Ουρά πρέπει να είναι και αυτή με τη σειρά της λίστα*".

```
is_list([]).
is_list([Head|Tail]) :- is_list(Tail).
```

Η ουρά δεν είναι λίστα

```
?- is_list([]).
yes
?- is_list([a,b]).
yes
```

```
?- is_list(b).
no
?- is_list([[a],b]).
yes
```

```
?- is_list([a|[b]]).
yes
?- is_list([a|b]).
no
```

Εύρεση του Τελευταίου Στοιχείου μιας Λίστας

- Το κατηγόρημα **last** δέχεται δύο ορίσματα.
  - Το πρώτο όρισμα είναι η λίστα.
  - Το δεύτερο όρισμα είναι είτε ένας όρος, είτε μια μεταβλητή.
- Ανάλογα με τον τρόπο που θα κληθεί το κατηγόρημα, μπορεί να χρησιμοποιηθεί
  - είτε για να ελέγξει αν ένα στοιχείο είναι το τελευταίο στοιχείο μιας δοσμένης λίστας,
  - είτε για να επιστρέψει το τελευταίο στοιχείο της λίστας.

Κατηγόρημα last/2

Παραδείγματα Χρήσης

?- last([a,b,c], c).

yes

?- last([a,b,c], b).

no

?- last([a,b,c], X).
X=c

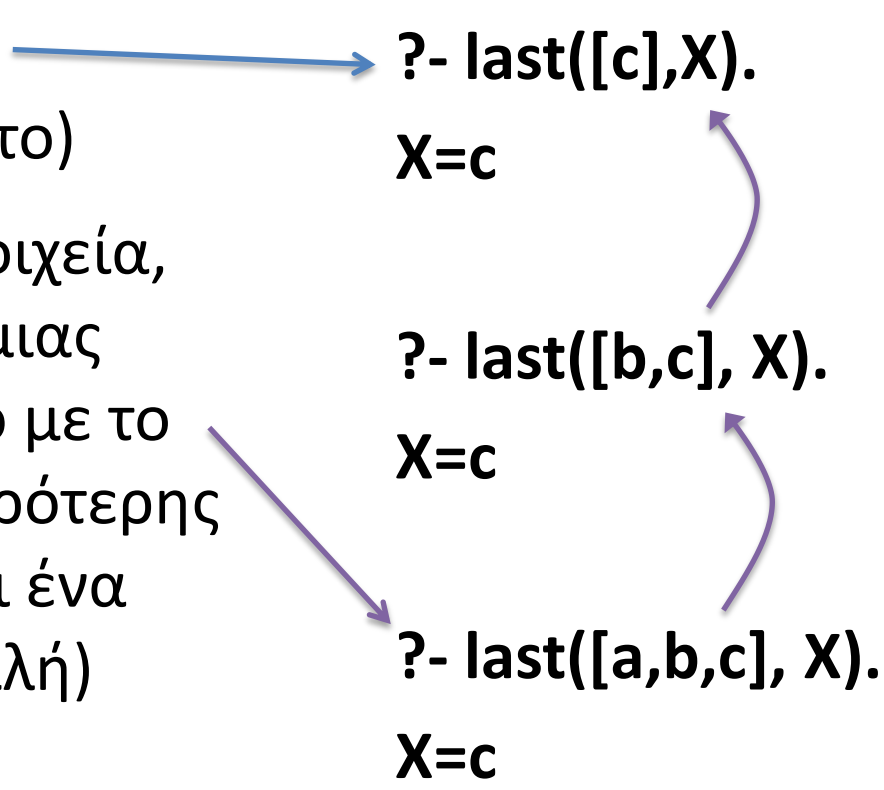
?- last([b,c], X).
X=c

?- last([c], X).
X=c

?- last([], X).
no

Παρατηρήσεις

- Όταν η λίστα έχει ένα μόνο στοιχείο, αυτό είναι και το τελευταίο (αλλά και το πρώτο)
- Όταν η λίστα έχει πολλά στοιχεία, τότε το τελευταίο στοιχείο μιας μεγάλης λίστας είναι το ίδιο με το τελευταίο στοιχείο μιας μικρότερης λίστας (της ουράς), που έχει ένα στοιχείο λιγότερο (την κεφαλή)



Κατηγόρημα last/2

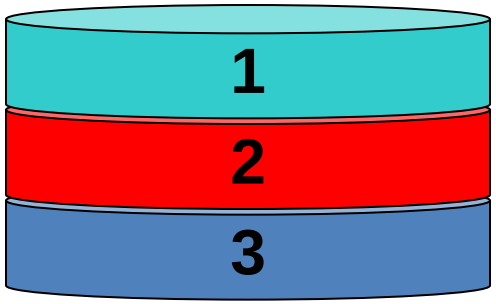
```
last( [X], X).
last( [Head|Tail], X) :- last(Tail,X).
```

- ***Δηλωτικός ορισμός***: Για να είναι ένα στοιχείο το τελευταίο μιας λίστας θα πρέπει:
  - είτε να είναι το μοναδικό στοιχείο της λίστας,
  - είτε να είναι το τελευταίο στοιχείο της ουράς της.
- ***Διαδικαστικός ορισμός***: Για να βρω το τελευταίο στοιχείο μιας λίστας:
  - Αν η λίστα έχει 1 μόνο στοιχείο, τότε είναι και το τελευταίο,
  - Αλλιώς, επέστρεψε το τελευταίο στοιχείο της ουράς

Παρομοίωση με Στοίβα (LIFO)

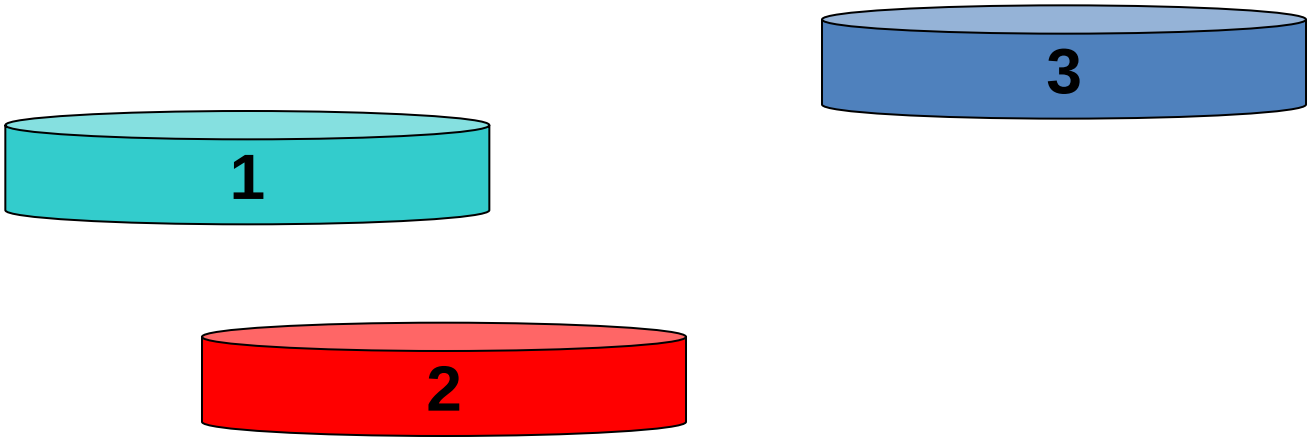
- Έχουμε 1 στοίβα από αντικείμενα (π.χ. πιάτα)
- Πρέπει να βρούμε το τελευταίο στοιχείο της στοίβας – το τελευταίο πιάτο!
- Κάθε φορά μπορούμε να μετακινούμε μόνο 1 πιάτο από την κορυφή της στοίβας
- Στο τραπέζι έχουμε όσο χώρο θέλουμε για να αφήνουμε τα πιάτα

Αρχική Κατάσταση



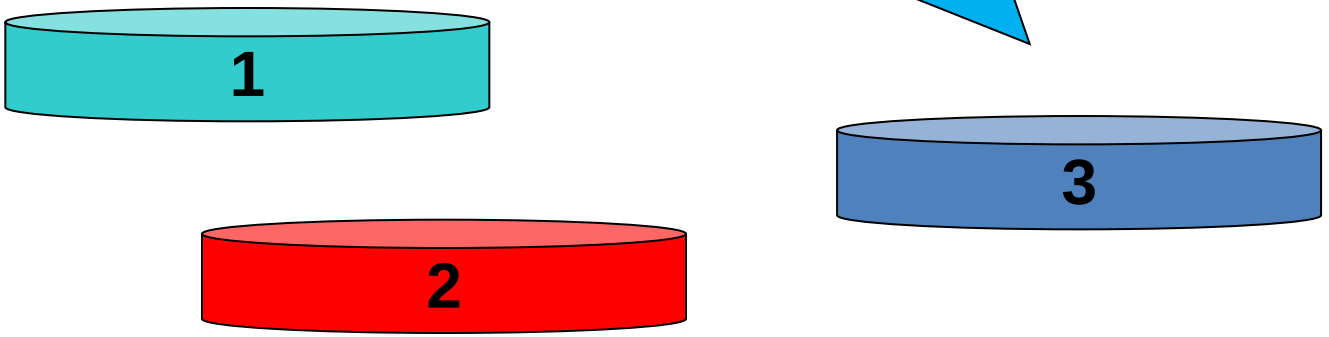
1ο Βήμα

2ο Βήμα



Επίλυση

Έμεινε μόνο 1 πιάτο στη στοίβα.
Αυτό είναι και το τελευταίο στοιχείο της στοίβας!

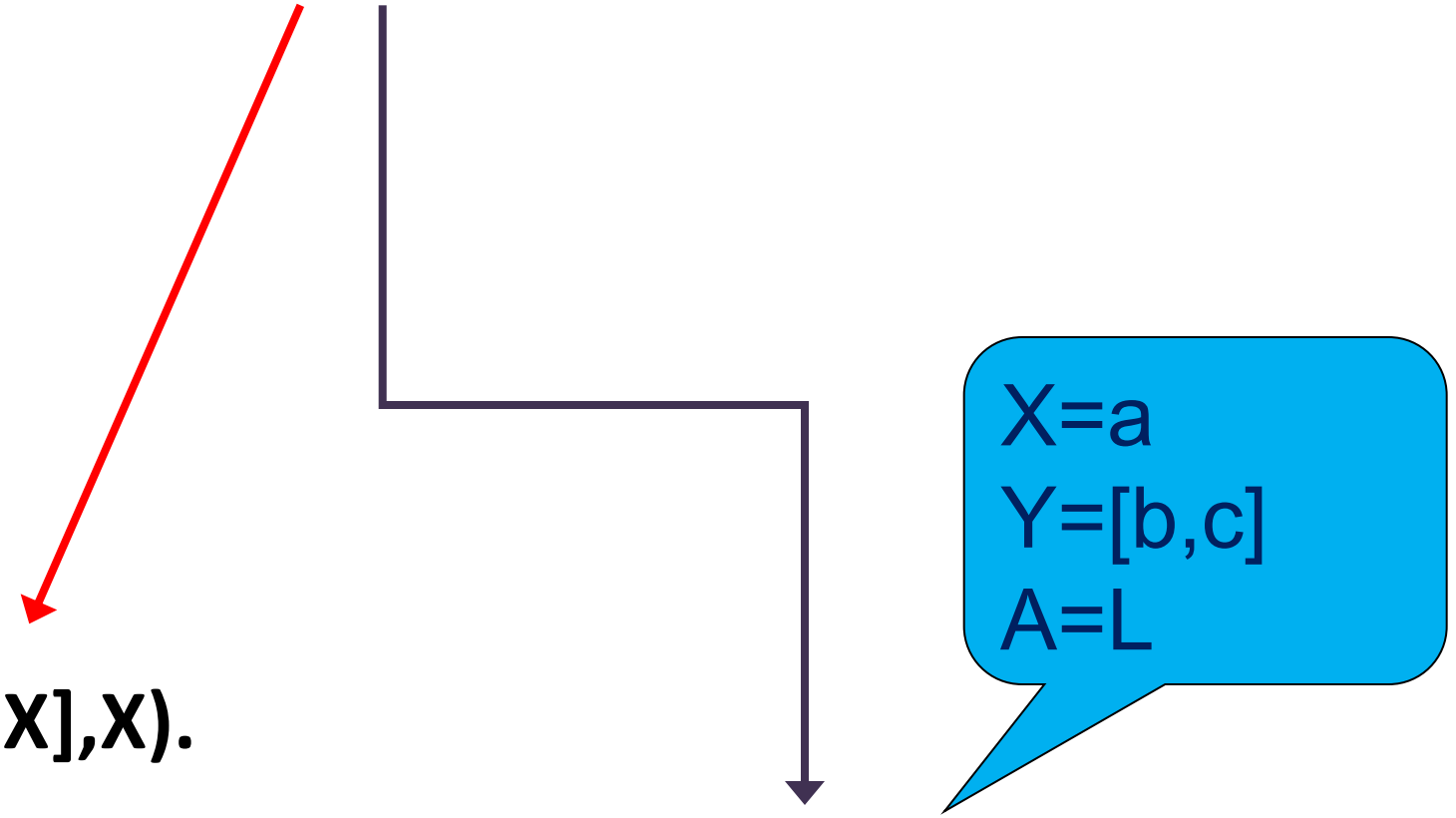


“Αλγόριθμος” last/2

- Αν η λίστα έχει μόνο ένα στοιχείο, τότε αυτό είναι και το τελευταίο
  - Επέστρεψέ το!
  - Τερματική συνθήκη
- Αν η λίστα έχει περισσότερα από ένα στοιχεία, τότε βγάλε το πρώτο στοιχείο από τη λίστα (κεφαλή), και βρες το τελευταίο στοιχείο της λίστας που απομένει (ουρά)
  - Αναδρομή

Επίλυση – Βήμα 1 – Κλήση

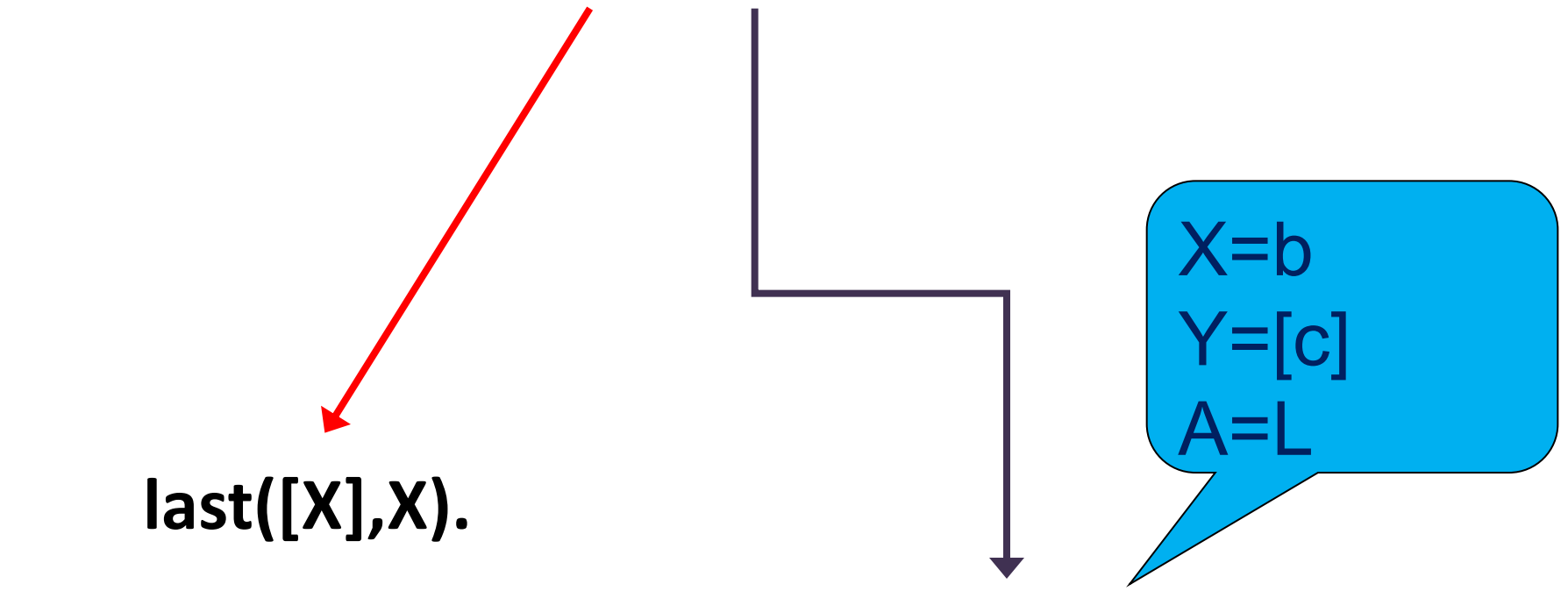
?- last([a,b,c],A).



Επίλυση – Βήμα 2 – Κλήση

?- last([a,b,c],A).

?- last([b,c],A).



last([X],X).

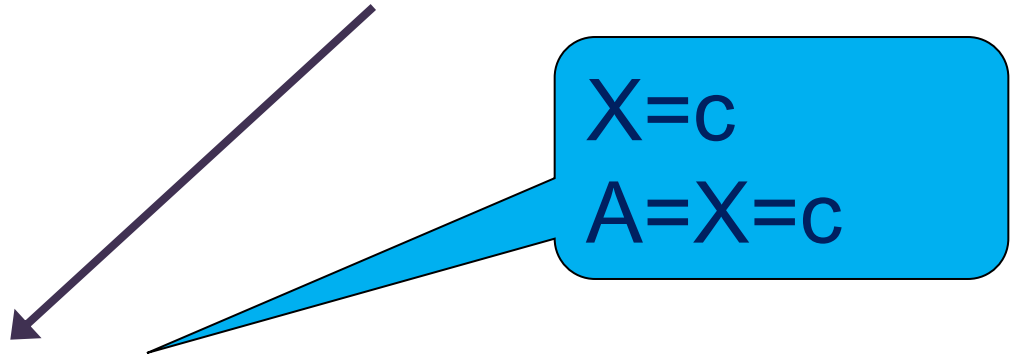
last([X|Y],L) :- last(Y,L).

Επίλυση – Βήμα 3 – Κλήση Επιστροφή

?- last([a,b,c],A).

?- last([b,c],A).

?- last([c],A).



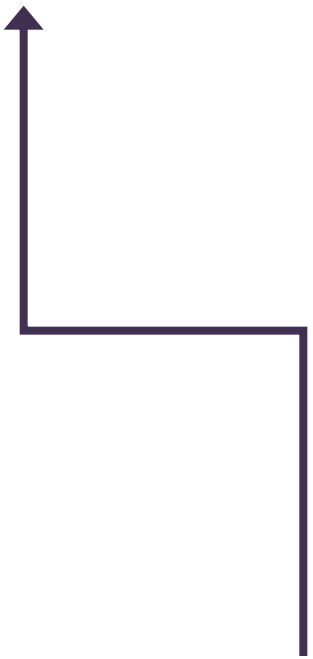
last([X],X).

last([X|Y],L) :- last(Y,L).

Επίλυση – Βήμα 2 - Επιστροφή

?- last([a,b,c],A).

?- last([b,c],A).



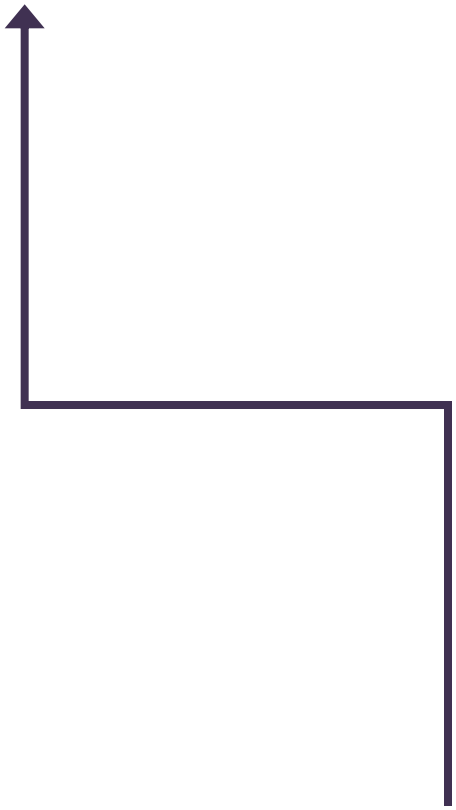
X=b
Y=[c]
A=L=c

last([X],X).

last([X|Y],L) :- last(Y,L).

Επίλυση – Βήμα 1 – Επιστροφή

```
?- last([a,b,c],A).
```



X=a
Y=[b,c]
A=L=c

```
last([X],X).

last([X|Y],L) :- last(Y,L).
```

Επίλυση – Απάντηση

?- last([a,b,c],A).

A=c

last([X],X).

last([X|Y],L) :- last(Y,L).

Παρατήρηση για τον Τερματικό Κανόνα

- Ο τερματικός κανόνας του κατηγορήματος **last** αφορά τις λίστες που έχουν 1 στοιχείο και όχι τις κενές λίστες.
- Στην περίπτωση της κενής λίστας δεν έχει νόημα η εύρεση του τελευταίου στοιχείου
  - Η επιθυμητή απάντηση θα ήταν η αποτυχία.
  - Η αποτυχία «επιτυγχάνεται» γιατί και οι δύο κανόνες δεν μπορούν να ενοποιηθούν με την κενή λίστα.
- Δεν ακολουθείται πάντα η γενική αρχή, πως ο τερματικός κανόνας αφορά την κενή λίστα.
  - Σε κάθε κατηγόρημα εξετάζεται το συγκεκριμένο πρόβλημα που καλείται να επιλύσει.
  - Έτσι προκύπτει το είδος και το πλήθος των κανόνων.

Έλεγχος Συμπερίληψης Στοιχείου σε Λίστα

- Το κατηγόρημα **member** δέχεται δύο ορίσματα
  - Το πρώτο είναι είτε ένας όρος, είτε μια μεταβλητή.
  - Το δεύτερο στοιχείο είναι η λίστα.
- Ανάλογα με τον τρόπο που θα κληθεί το κατηγόρημα, μπορεί να χρησιμοποιηθεί
  - είτε για να ελέγξει αν ένα στοιχείο ανήκει σε μία λίστα,
  - είτε για να επιστρέψει όλα τα στοιχεία της λίστας.

Κατηγόρημα member/2
Παραδείγματα Χρήσης

```
?- member(b,[b,c]).
yes
?- member(b,[a,b,c]).
yes
?- member(b,[c]).
no
?- member(b,[]).
no
```

```
?- member(X,[a,b,c]).
X=a;
X=b;
X=c;
no
?-member(X,[]).
no
```

Παρατήρηση

- Όταν το στοιχείο που ψάχνω να βρω είναι το πρώτο στοιχείο της λίστας (κεφαλή), τότε το βρήκα!
- Αλλιώς, μπορώ να το βρω στην ουρά της λίστας, δηλαδή στην λίστα που προκύπτει αν φύγει η κεφαλή.

?- member(b,[b,c]).

yes

?- member(b,[a,b,c]).

yes

Κατηγόρημα member/2

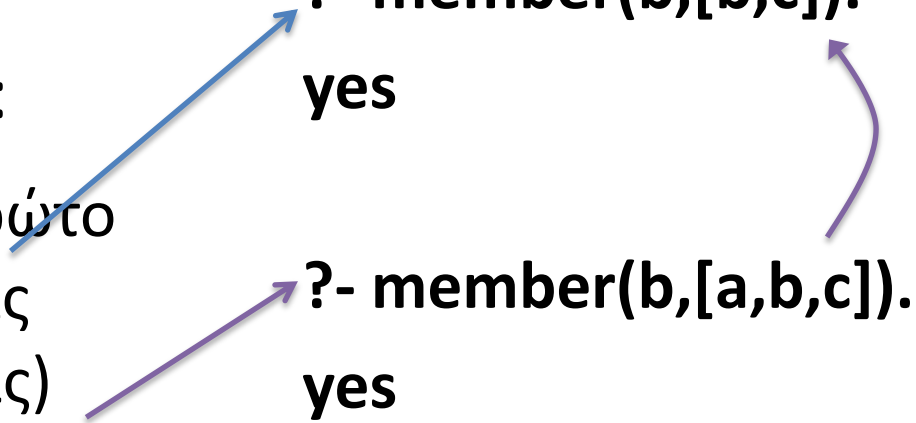
member(X,[X|Y]).

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

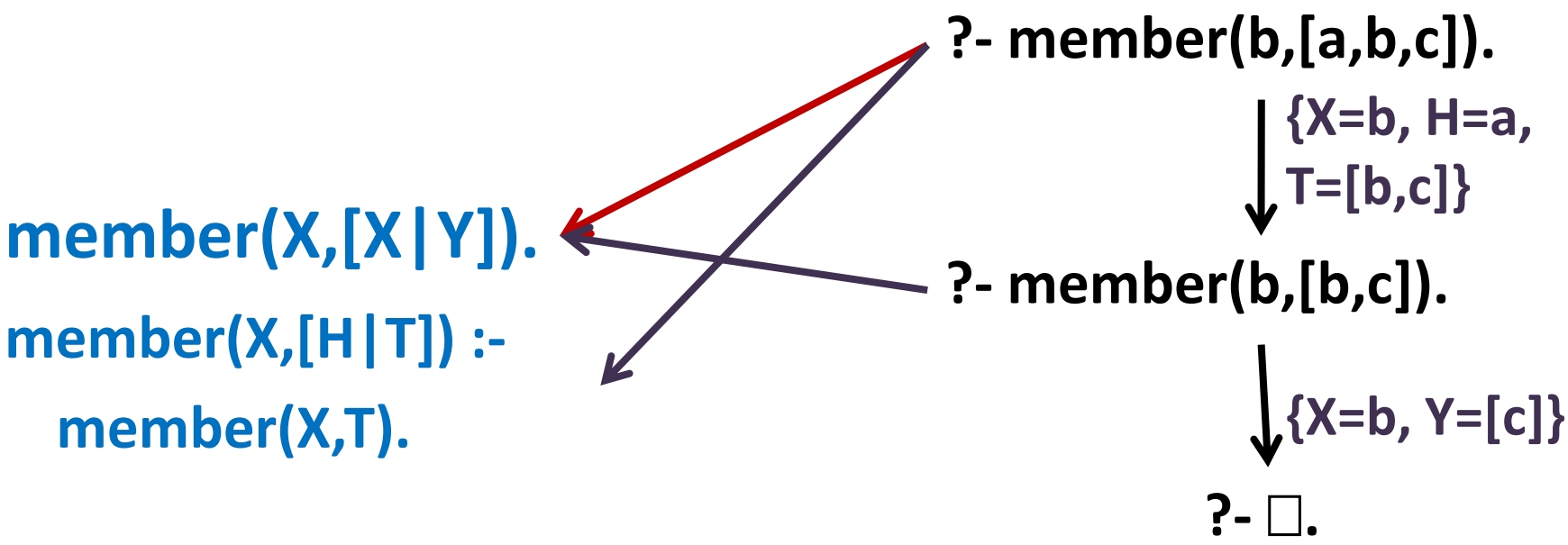
- ***Δηλωτικός ορισμός***: Ένα στοιχείο είναι μέλος μιας λίστας:
  - είτε αν είναι το πρώτο στοιχείο της λίστας (κεφαλή της λίστας)
  - είτε αν είναι μέλος της ουράς της, δηλαδή της λίστας που προκύπτει αν φύγει η κεφαλή.
- ***Διαδικαστικός ορισμός***: Για να βρεις αν ένα στοιχείο είναι μέλος μιας λίστας:
  - Αν ταιριάζει με την κεφαλή της λίστας, τότε το βρήκες.
  - Αλλιώς, ψάξτο στην ουράς της λίστας.

Παρατήρηση

- ένα στοιχείο είναι μέλος μιας λίστας:
  - είτε αν είναι το πρώτο στοιχείο της λίστας (κεφαλή της λίστας)
  - είτε αν είναι μέλος της ουράς της, δηλαδή της λίστας που προκύπτει αν φύγει η κεφαλή.



Παράδειγμα εκτέλεσης member



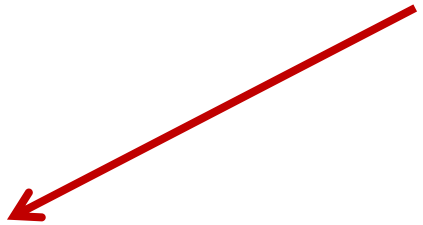
Παράδειγμα εκτέλεσης member

?- member(c,[a,b]).

```
member(X,[X|Y]).
member(X,[H|T]) :-
   member(X,T).
```

Παράδειγμα εκτέλεσης member

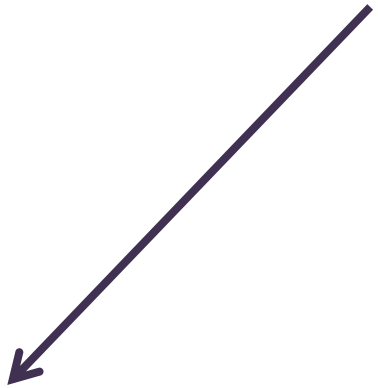
?- member(c,[a,b]).



```
member(X,[X|Y]).
member(X,[H|T]) :-
   member(X,T).
```

Παράδειγμα εκτέλεσης member

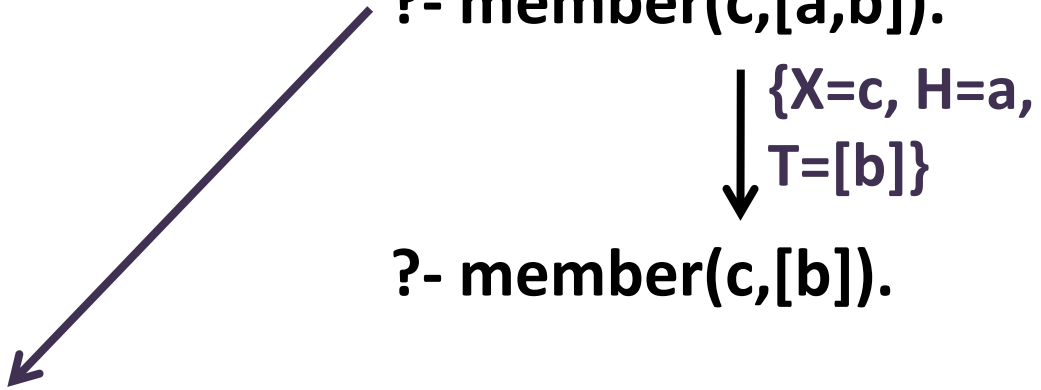
?- member(c,[a,b]).



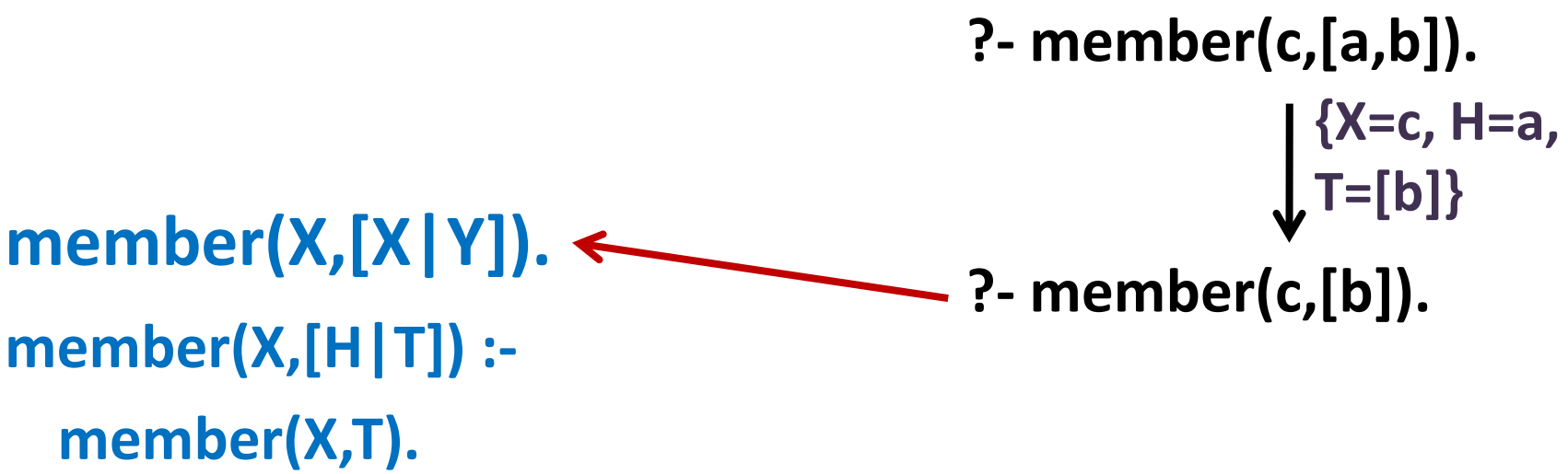
```
member(X,[X|Y]).
member(X,[H|T]) :-
    member(X,T).
```

Παράδειγμα εκτέλεσης member

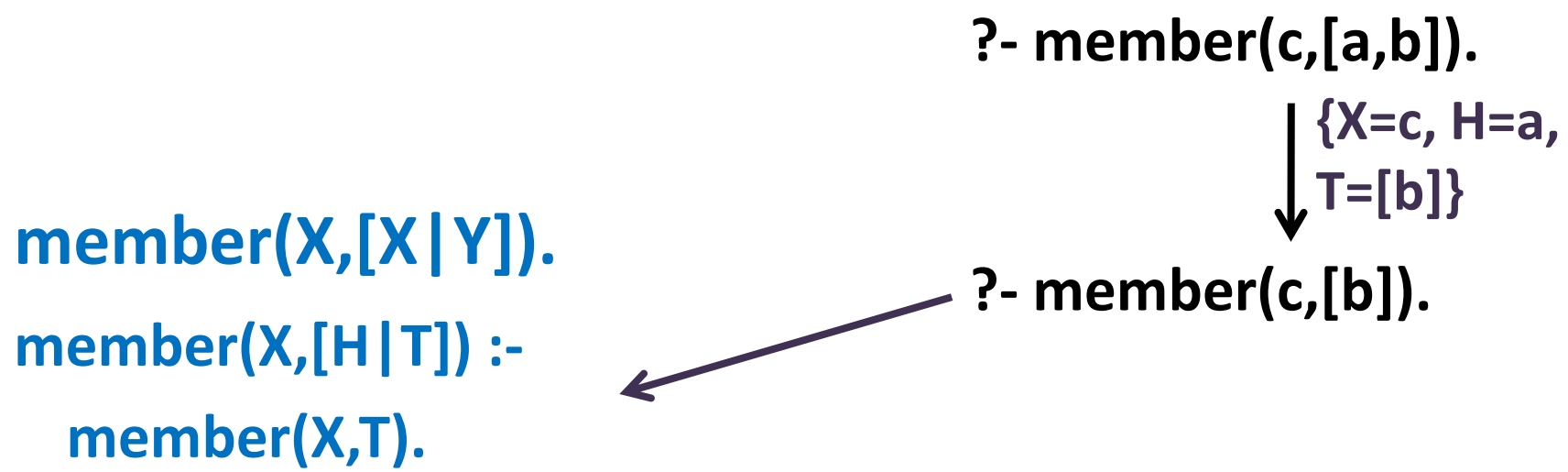
```
member(X,[X|Y]).
member(X,[H|T]) :-
    member(X,T).
```



Παράδειγμα εκτέλεσης member



Παράδειγμα εκτέλεσης member



Παράδειγμα εκτέλεσης member

```
member(X,[X|Y]).
member(X,[H|T]) :-
    member(X,T).
```

?- member(c,[a,b]).

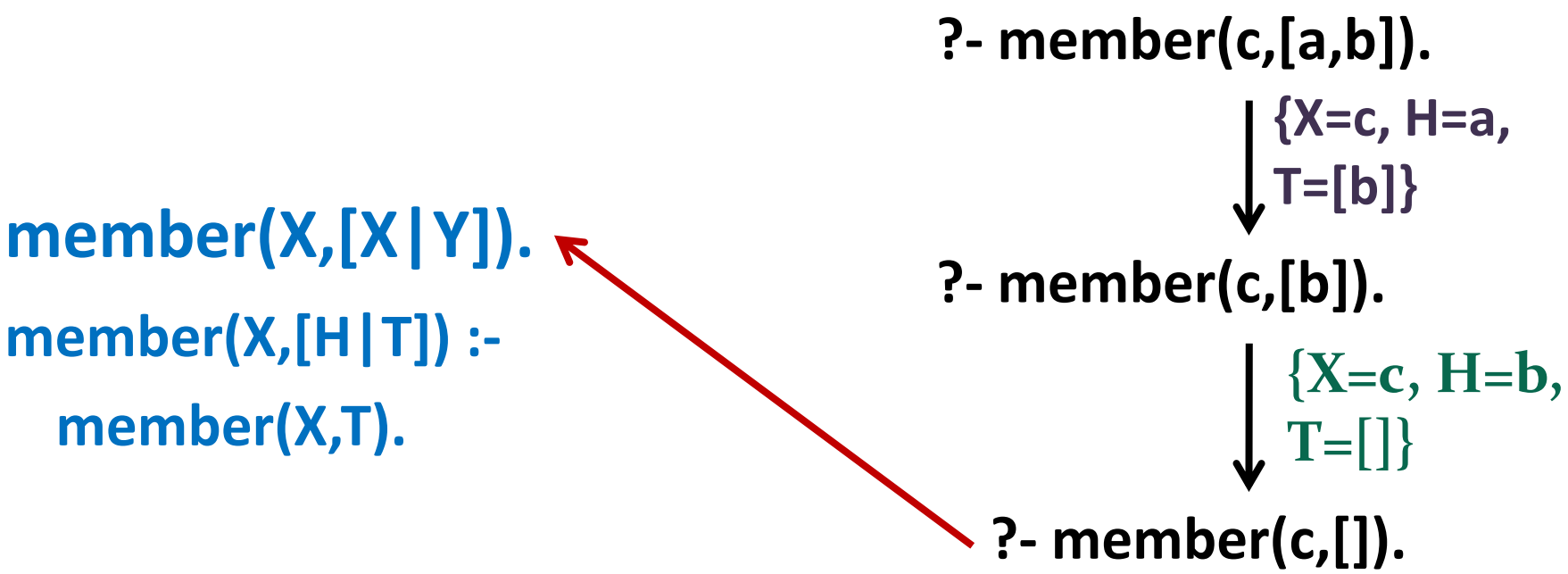
↓ {X=c, H=a, T=[b]}

?- member(c,[b]).

↓ {X=c, H=b, T=[]}

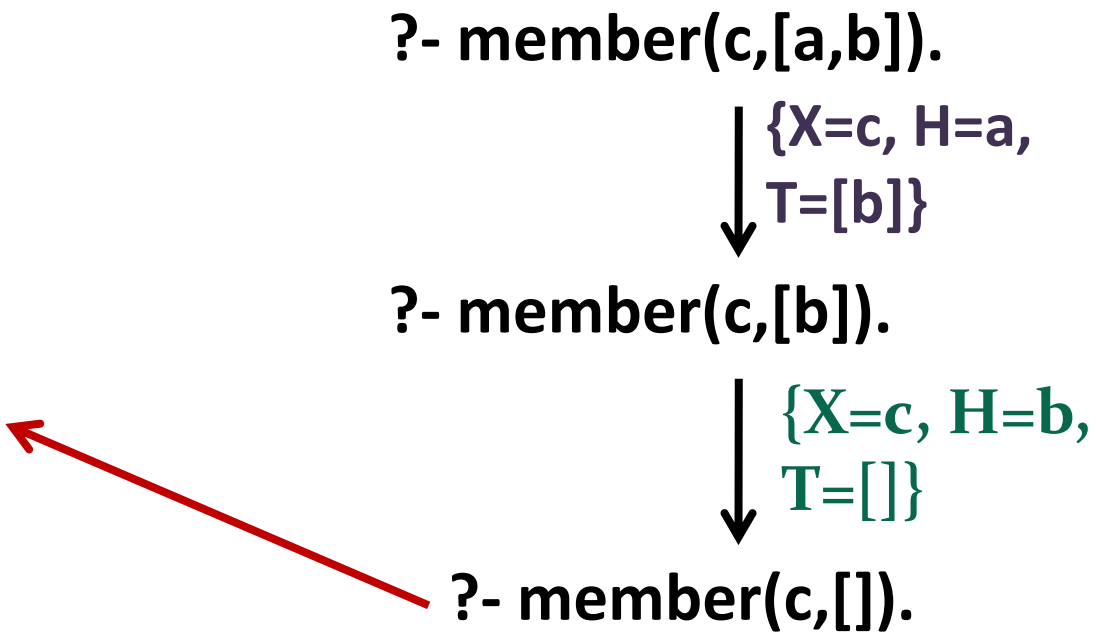
?- member(c,[]).

Παράδειγμα εκτέλεσης member



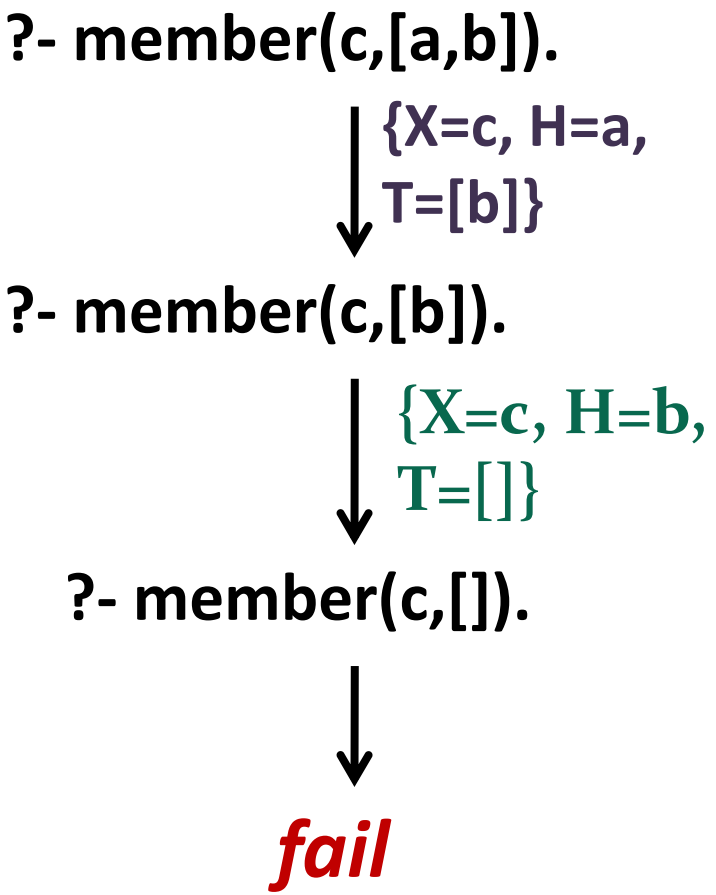
Παράδειγμα εκτέλεσης member

```
member(X,[X|Y]).
member(X,[H|T]) :-
    member(X,T).
```



Παράδειγμα εκτέλεσης member

```
member(X,[X|Y]).
member(X,[H|T]) :-
   member(X,T).
```



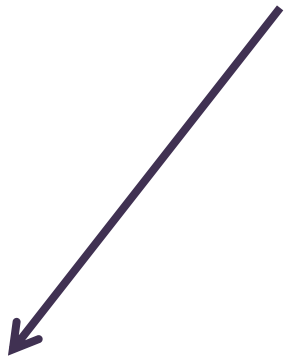
Παράδειγμα εκτέλεσης member

?- member(E,[a,b]).

```
member(X,[X|Y]).
member(X,[H|T]) :-
   member(X,T).
```

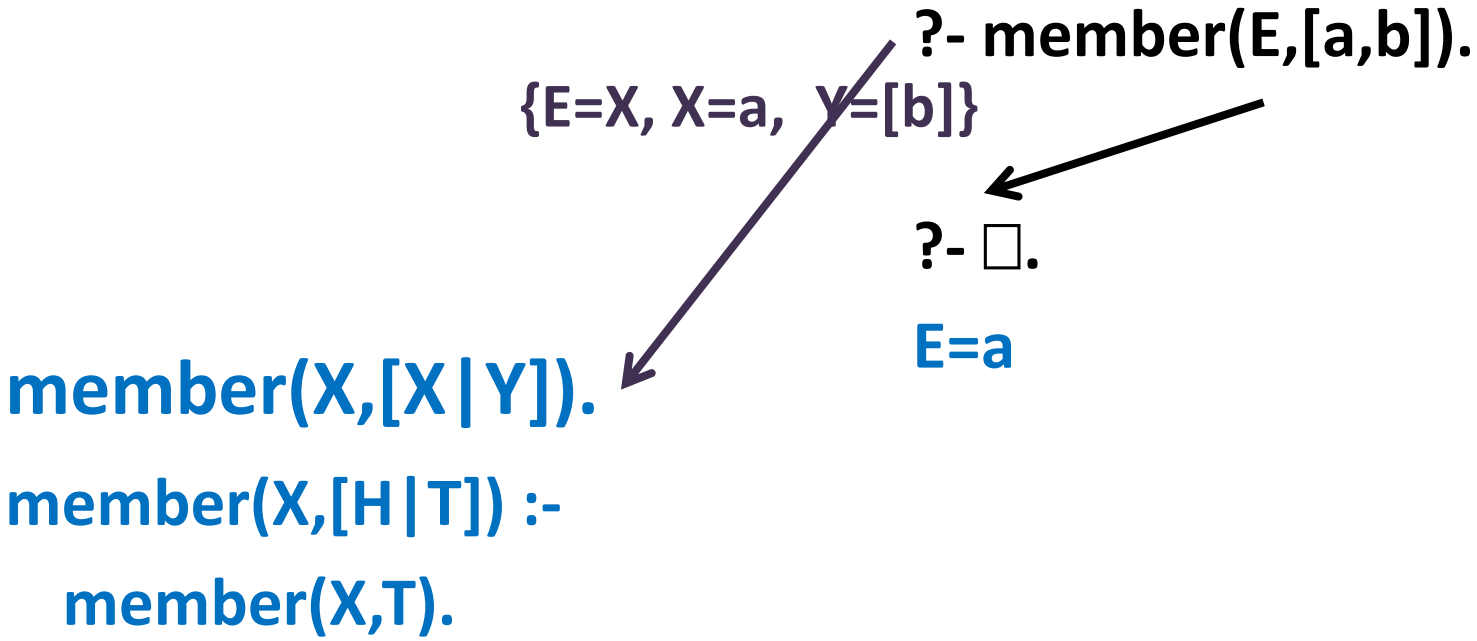
Παράδειγμα εκτέλεσης member

?- member(E,[a,b]).

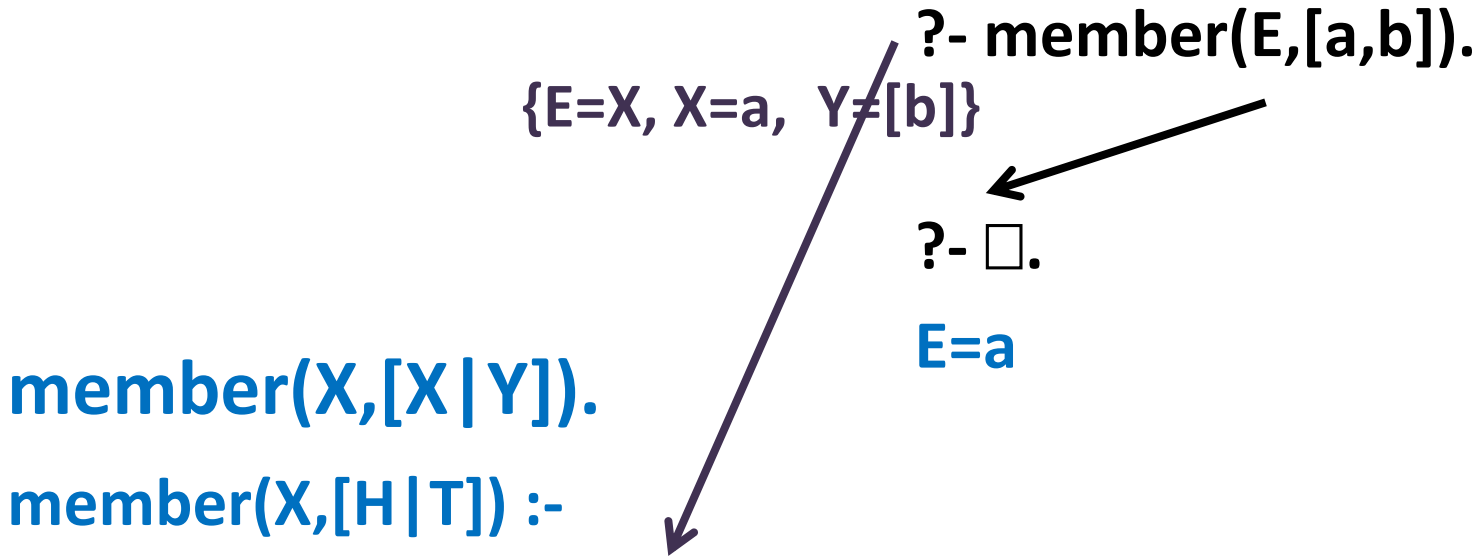


```
member(X,[X|Y]).
member(X,[H|T]) :-
   member(X,T).
```

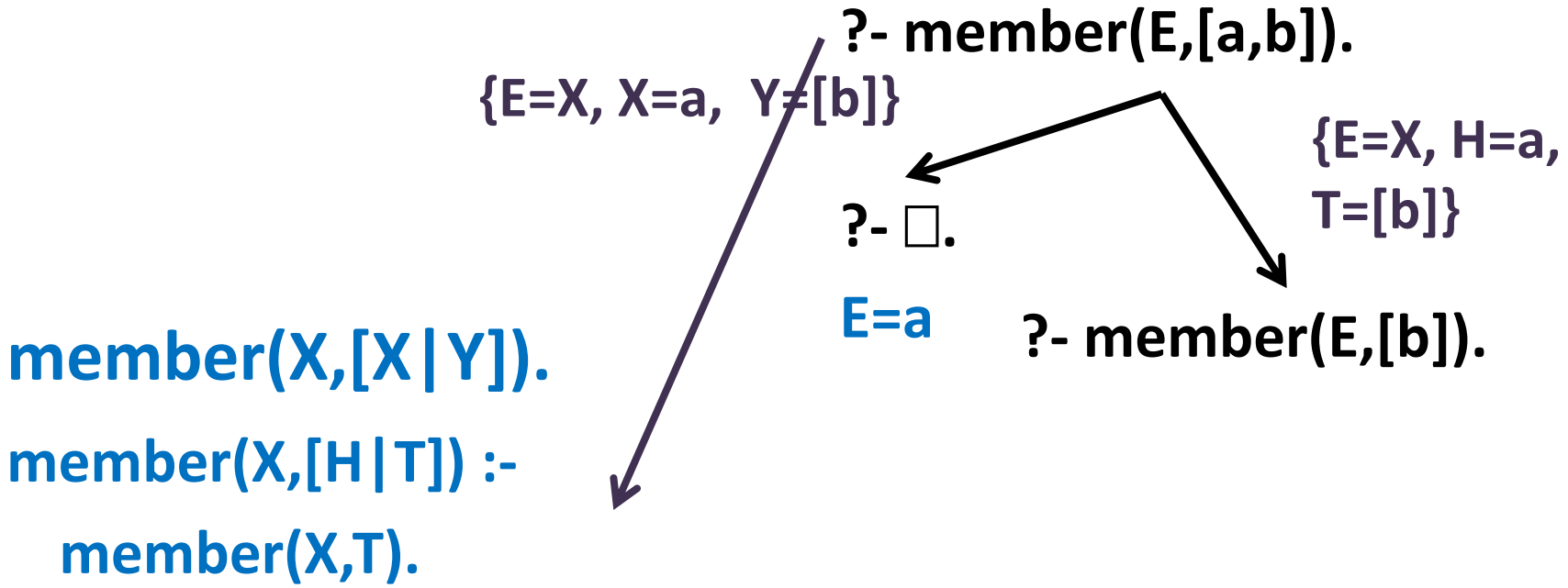
Παράδειγμα εκτέλεσης member



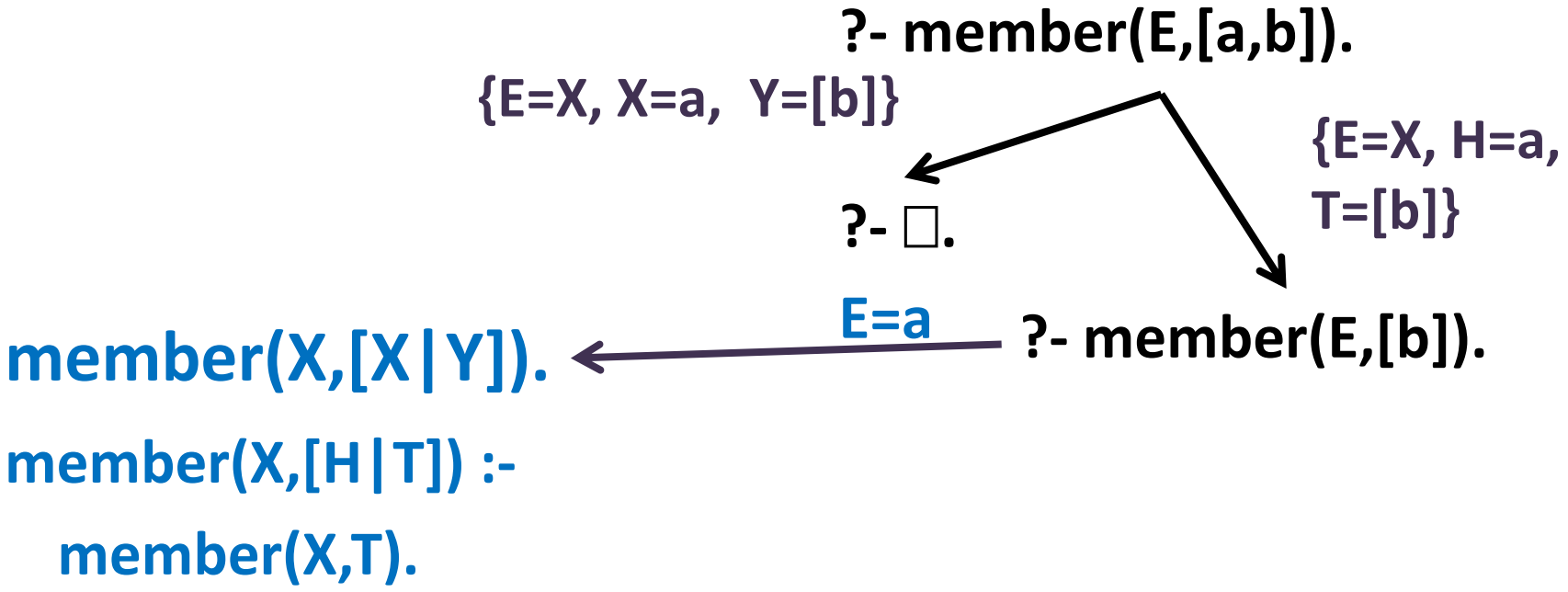
Παράδειγμα εκτέλεσης member



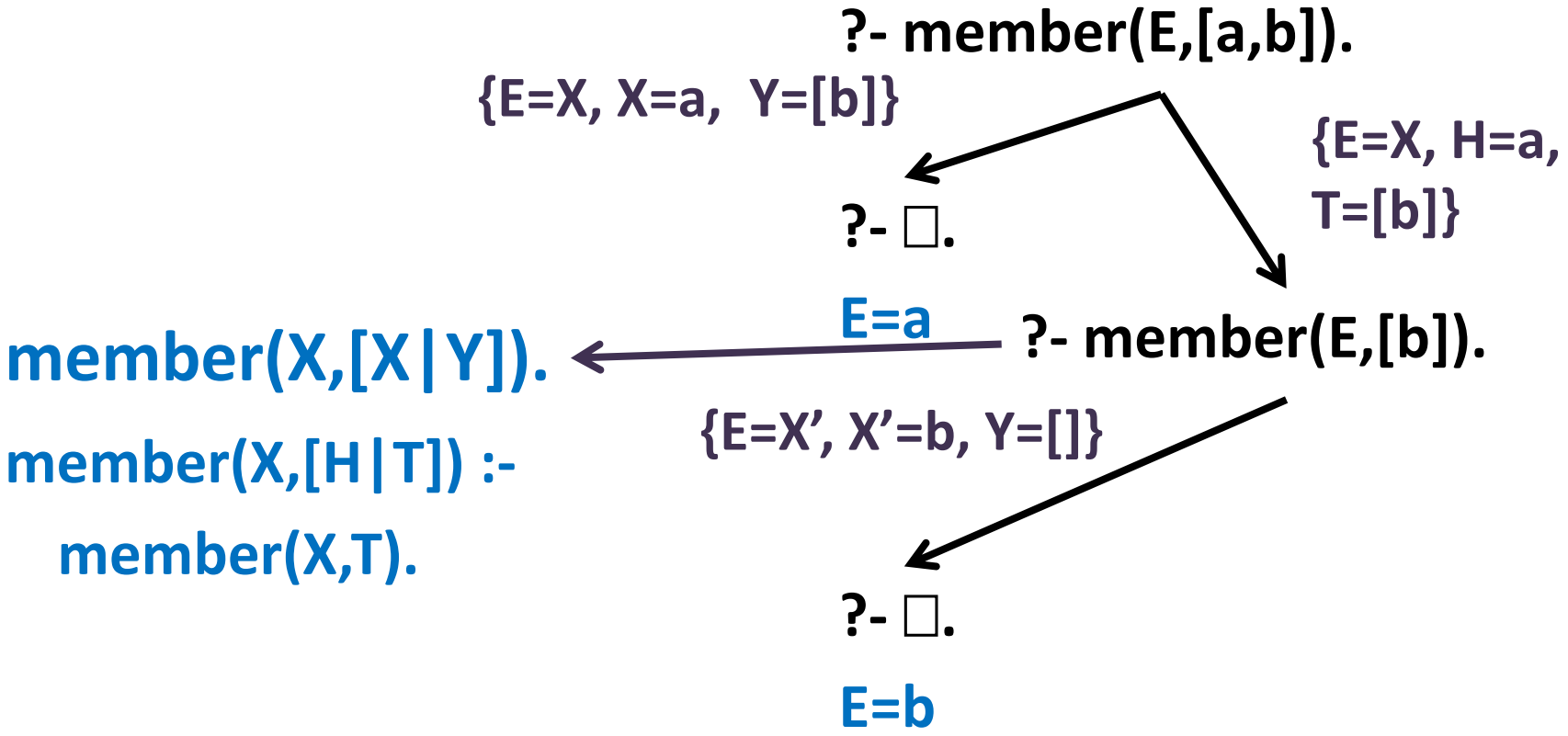
Παράδειγμα εκτέλεσης member



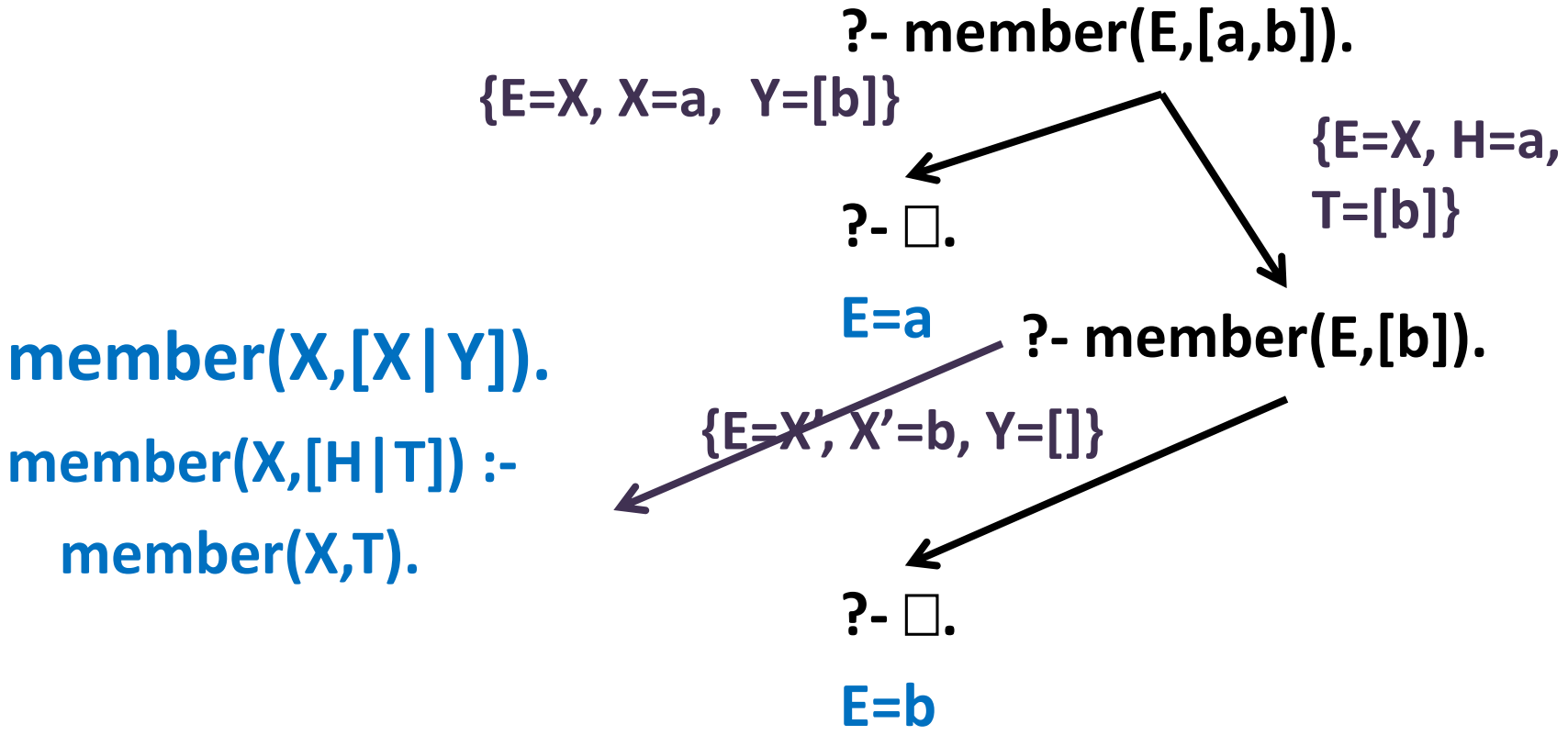
Παράδειγμα εκτέλεσης member



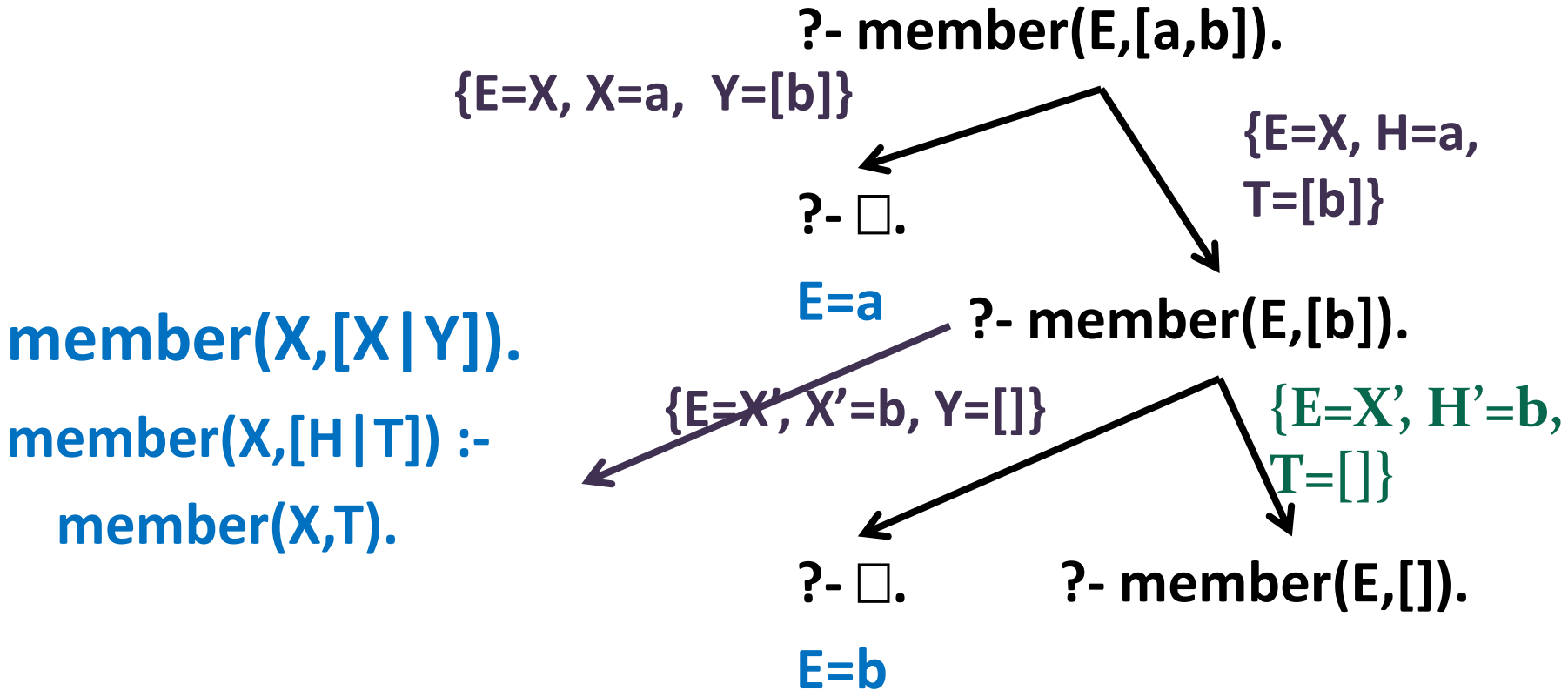
Παράδειγμα εκτέλεσης member



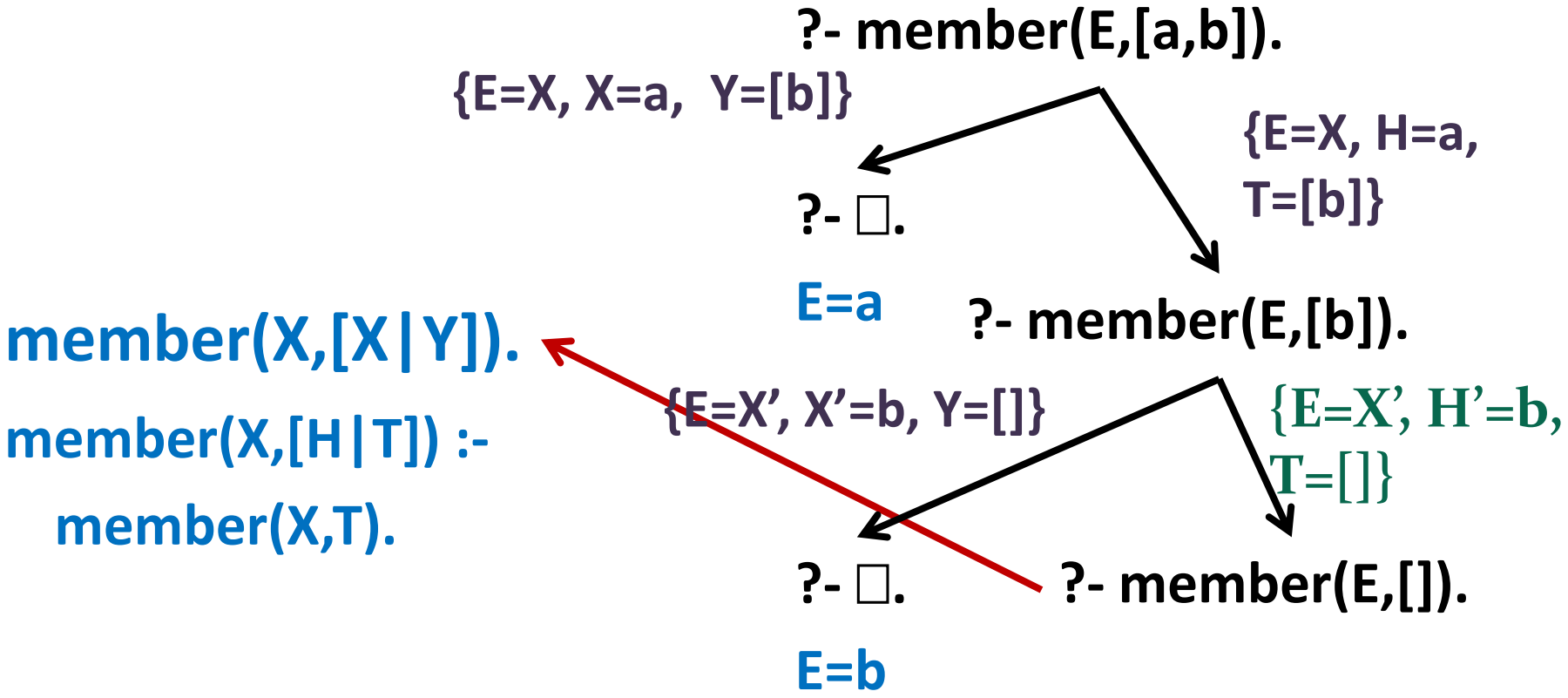
Παράδειγμα εκτέλεσης member



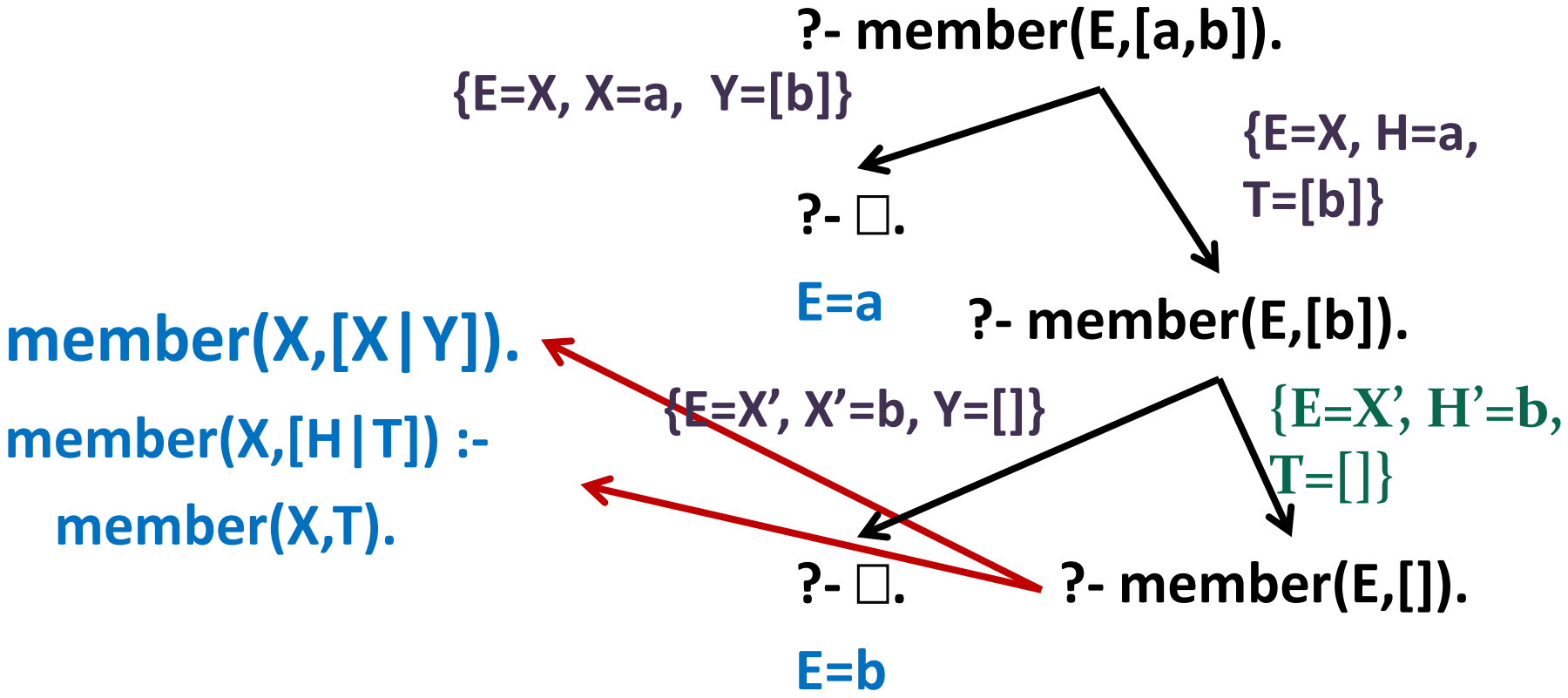
Παράδειγμα εκτέλεσης member



Παράδειγμα εκτέλεσης member



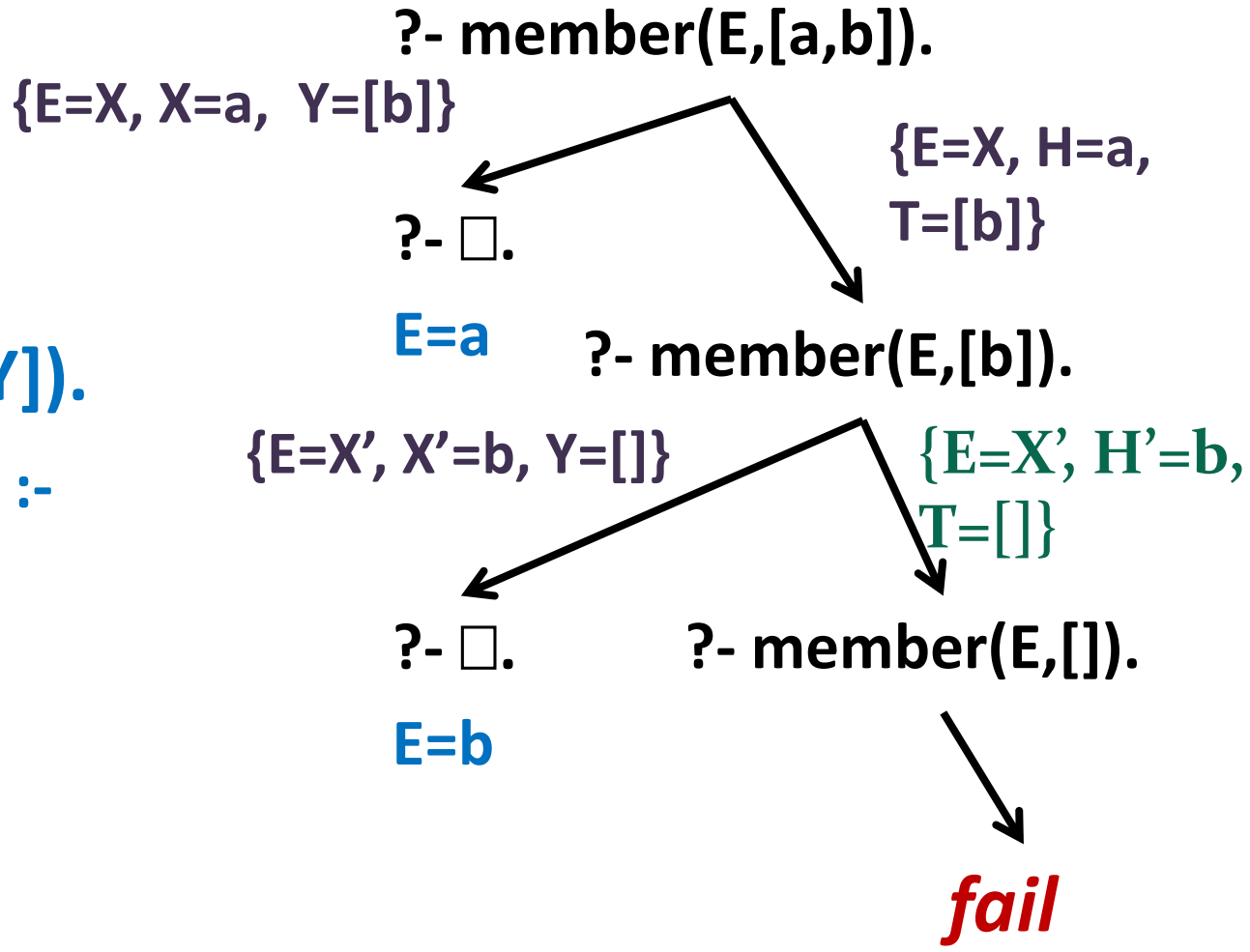
Παράδειγμα εκτέλεσης member



Παράδειγμα εκτέλεσης member

member(X,[X|Y]).

member(X,[H|T]) :-
  member(X,T).



Παρατηρήσεις

- Στην περίπτωση του κατηγορήματος **member** ο τερματικός κανόνας επίσης δεν αφορά την κενή λίστα.
  - Ουσιαστικά η **member** τερματίζει την πρώτη φορά που το προς αναζήτηση στοιχείο ενοποιείται με την κεφαλή της λίστας, καθώς την διατρέχουμε.
- Η κενή λίστα δεν έχει στοιχεία συνεπώς η προσπάθεια αναζήτησης στοιχείων στην κενή λίστα πρέπει πάντα να αποτυγχάνει.
  - Η αποτυχία «επιτυγχάνεται» γιατί και οι δύο κανόνες δεν μπορούν να ενοποιήσουν το 2ο όρισμα της κεφαλής τους με την κενή λίστα, η οποία δεν μπορεί να "χωριστεί" σε κεφαλή και ουρά.

Ασκήσεις – first/2

- Να γραφεί κατηγόρημα που επιστρέφει το πρώτο στοιχείο μιας λίστας, ή απλά κάνει έλεγχο.

first([X | Y] , X).

?- first([a,b,c,d],X). *% Ποιο είναι το 1ο στοιχείο της λίστας;*

X=a

?- first([a,b,c],a). *% Είναι το 'a' πρώτο στοιχείο της λίστας;*

yes

?- first([F,b,c,d],a). *% Βάλε το 'a' ως πρώτο στοιχείο της λίστας*

F=a

?- first(L,a).

Φτιάξε μια λίστα (άγνωστος αριθμός στοιχείων), με πρώτο στοιχείο το 'a'

L= [a | _791323]

Μεταβλητή (θέση μνήμης) της οποίας το όνομα είναι απροσδιόριστο

Ασκήσεις – third/2

- Να γραφεί κατηγόρημα που να επιστρέφει (ελέγχει) το τρίτο στοιχείο μιας λίστας.

?- third([a,b,c,d], A).

A=c

?- third([a,b], A).

no

- Λύση

third([A, B, C | D], C).

ή

third ([_ , _ , X | _], X).

Άσκηση μετατροπής μιας λίστας σε άλλη (1/2)

- Να οριστεί το κατηγόρημα **translate(List1,List2),** το οποίο να μεταφράζει μία λίστα με αριθμούς από 0 μέχρι 9 στις αντίστοιχες λέξεις.
  - Αν κάποιο στοιχείο της λίστας δεν είναι αριθμός, να μένει ίδιο

?- translate([3,5,1,hello,3],A).

A = [three,five,one,hello,three]

Άσκηση μετατροπής μιας λίστας σε άλλη (2/2)

```
?- translate([3,5,1,hello,3],A).
A =
   [three,five,one,hello,three]
```

```
translate([],[]).
translate([H|T],[H1|T1]) :-
   translate_one(H,H1),
   translate(T,T1).
```

```
translate_one(0,zero).
translate_one(1,one).
translate_one(2,two).
translate_one(3,three).
translate_one(4,four).
translate_one(5,five).

translate_one(6,six).
translate_one(7,seven).
translate_one(8,eight).
translate_one(9,nine).
translate_one(X,X).
```

Άσκηση – Διαγραφή ενός στοιχείου από λίστα (1/2)

```
?- delone(a,[a,b,c],L).
L=[b,c]
?- delone(b,[a,b,c],L).
L=[a,c]
?- delone(a,[a,b,a,c],L).
L=[b,a,c]
?- delone(d,[a,b,c],L).
no
```

```
delone(X, [X|T], T).
delone(X, [Y|T], [Y|T1]) :-
    delone(X, T, T1).
```

Άσκηση – Διαγραφή ενός στοιχείου από λίστα (1/2)

- Αν θέλω να επιστρέφεται η αρχική λίστα, όταν δεν υπάρχει το προς-διαγραφή-στοιχείο στη λίστα

```
?- delone(d,[a,b,c],L).

L=[a,b,c]
```

```
delone(X,[],[]).
delone(X, [X|T], T).
delone(X, [Y|T], [Y|T1]) :-
   delone(X, T, T1).
```

Άσκηση – Διαγραφή πολλών στοιχείων από λίστα

- Να ορισθεί το κατηγόρημα **delall/3** που να διαγράφει όλες τις εμφανίσεις ενός στοιχείου από μια λίστα.

```
?- delall(a,[a,b,a,c],L).
L=[b, c]
?- delall(b,[a,b,c],L).
L=[a,c]
?- delall(d,[a,b,c],L).
L=[a,b,c]
?- delall(a,[a,a,a],L).
L=[]
```

```
delall(X,[],[]).
delall(X,[X|T],T1) :-
   delall(X,T,T1).
delall(X,[H|T],[H|T1]) :-
   delall(X,T,T1).
```

Ασκήσεις - Ισομήκεις Υπολίστες

- Να ορισθεί η σχέση **divide_list(L,L1,L2)** έτσι ώστε η λίστα **L** να χωρίζεται σε 2 υπολίστες με περίπου ίδιο μήκος (τα στοιχεία εναλλάξ).

?- divide_list([a,b,c,d,e],A,B).

A=[a,c,e], B=[b,d]

- Λύση:

2 συνθήκες τερματισμού

```
divide_list([],[],[]).
divide_list([X],[X],[]).
divide_list([X,Y|L],[X|L1],[Y|L2]) :-
    divide_list(L,L1,L2).
```

Για τις λίστες με άρτιο αριθμό στοιχείων

Για τις λίστες με περιττό αριθμό στοιχείων

Σε κάθε «κύκλο» παίρνω 2 στοιχεία από την «αρχή» της πρώτης λίστας και τοποθετώ από ένα στην κεφαλή των δύο άλλων λιστών

Άσκηση – not_member/2

- Να κατασκευαστεί κατηγόρημα **not_member(X,L)**, το οποίο να αληθεύει όταν το στοιχείο **X** δεν περιέχεται στη λίστα **L**.

?- not_member(a,[b,a,c]).

no

?- not_member(d,[b,a,c]).

yes

- Η **not_member** είναι αντίθετη της **member**

?- member(a,[b,a,c]).

yes

?- member(d,[b,a,c]).

no

Παρατηρήσεις – not_member/2

```
?- not_member(b,[b,a,c]).
no
```

- Αν το στοιχείο είναι ίδιο με την κεφαλή της λίστας, τότε αποτυχία

```
?- not_member(a,[b,a,c]).
no
?- not_member(d,[b,a,c]).
yes
```

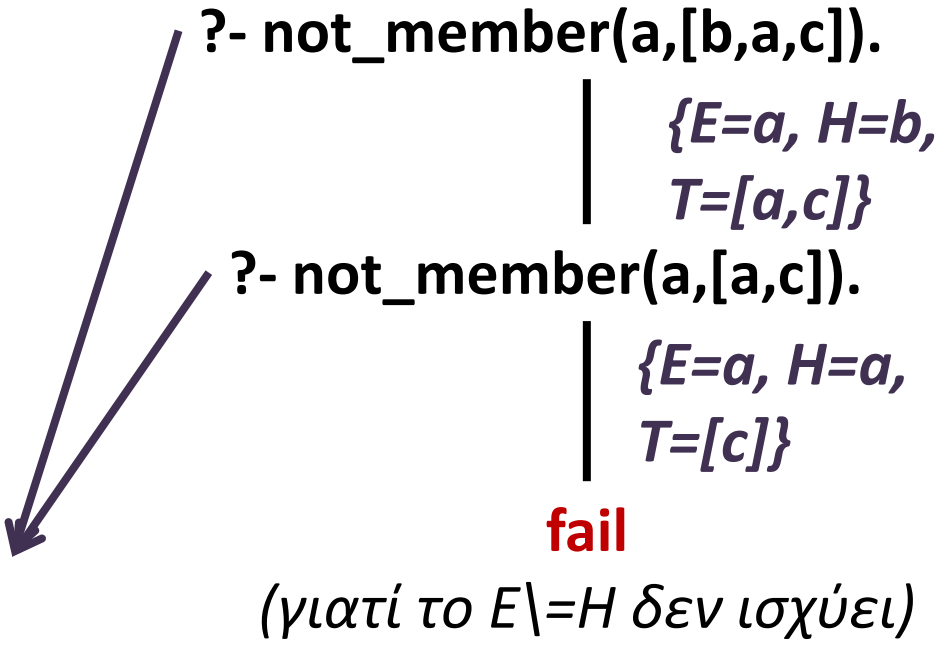
- Αν το στοιχείο δεν είναι ίδιο με την κεφαλή της λίστας, τότε υπάρχει πιθανότητα να μην είναι και μέλος.
- Η διαδικασία πρέπει να ελέγξει όλα τα στοιχεία της λίστας, άρα πρέπει να συνεχιστεί (αναδρομικά).

```
?- not_member(d,[]).
yes
```

- Αν η λίστα μείνει κενή, τότε όντως το στοιχείο δεν είναι μέλος της, γιατί η κενή λίστα δεν έχει κανένα μέλος!

Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```



Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

Κατηγόρημα not_member/2

?- not_member(d,[b,a,c]).

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

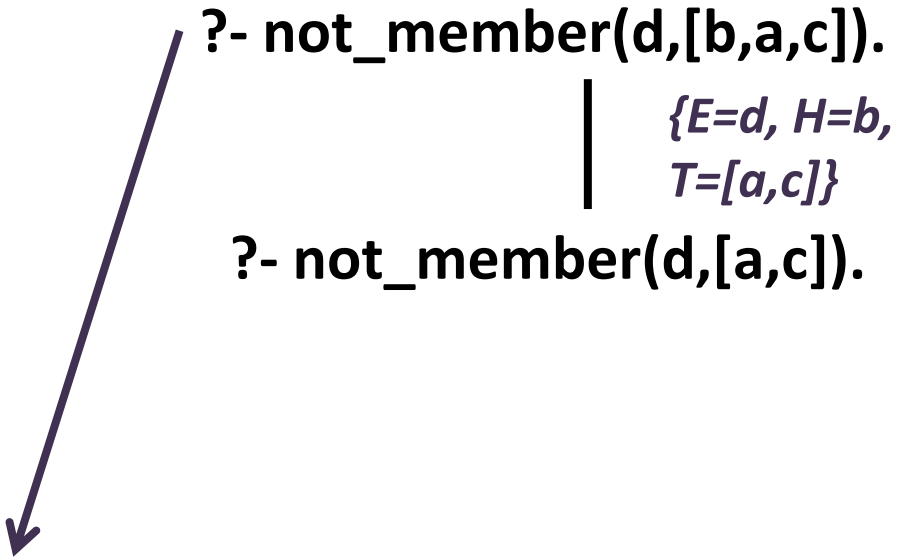
Κατηγόρημα not_member/2

?- not_member(d,[b,a,c]).

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```



Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

?- not_member(d,[b,a,c]).

| *{E=d, H=b, T=[a,c]}*

?- not_member(d,[a,c]).

Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

?- not_member(d,[b,a,c]).

{E=d, H=b, T=[a,c]}

?- not_member(d,[a,c]).

{E=d, H=a, T=[c]}

?- not_member(d,[c]).

Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

?- not_member(d,[b,a,c]).

| {E=d, H=b, T=[a,c]}

?- not_member(d,[a,c]).

| {E=d, H=a, T=[c]}

?- not_member(d,[c]).

Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```

?- not_member(d,[b,a,c]).

| {E=d, H=b, T=[a,c]}

?- not_member(d,[a,c]).

| {E=d, H=a, T=[c]}

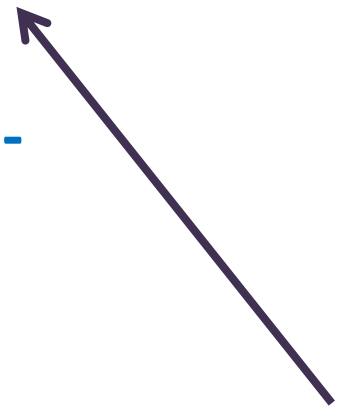
?- not_member(d,[c]).

| {E=d, H=c, T=[]}

?- not_member(d,[]).

Κατηγόρημα not_member/2

```
not_member(_,[]).
not_member(E,[H|T]) :-
    E \= H,
    not_member(E,T).
```



?- not_member(d,[b,a,c]).
{E=d, H=b, T=[a,c]}
?- not_member(d,[a,c]).
{E=d, H=a, T=[c]}
?- not_member(d,[c]).
{E=d, H=c, T=[]}
?- not_member(d,[]).
?- □.
yes

Λίστες - Σύνολα

- Οι λίστες μπορούν να χρησιμοποιηθούν για αναπαράσταση συνόλων.
- Στα σύνολα δεν παίζει ρόλο η σειρά των στοιχείων,
  - Συνεπώς χρησιμοποιούμε το κατηγόρημα **member/2**, το οποίο ελέγχει αν υπάρχει ένα στοιχείο σε μία λίστα χωρίς να μας ενδιαφέρει η σειρά.
- Στα σύνολα δεν υπάρχουν "διπλά" στοιχεία.
  - Κάθε στοιχείο εμφανίζεται μια μόνο φορά.
  - Μπορούμε να μετατρέψουμε μία λίστα σε σύνολο αν εξαλείψουμε τα διπλά στοιχεία.

Κατηγόρημα remove_duplicates/2

- Να κατασκευαστεί κατηγόρημα **remove_duplicates(L1,L2)**, το οποίο να "απομακρύνει" τα διπλά στοιχεία από μία λίστα **L1**, και να επιστρέφει μία καινούρια λίστα **L2**.

?- remove_duplicates([a,b,d,a,b,c],A).

A = [d,a,b,c]

?- remove_duplicates([a,b,c,d],X).

A = [a,b,c,d]

Παρατηρήσεις

?- remove_duplicates([a,d,a,b,c],A).

A = [d,a,b,c]

?- remove_duplicates([a,b,d,a,b,c],A).

A = [d,a,b,c]

?- remove_duplicates([a,b,c,b,d],X).

A = [a,c,b,d]

- Αν η κεφαλή της λίστας, υπάρχει στην ουρά (είναι μέλος της), τότε υπάρχει τουλάχιστον δύο φορές στη λίστα και δεν περιλαμβάνεται στην τελική λίστα.
- Είτε υπάρχει, είτε δεν υπάρχει, η ουρά της λίστας μπορεί να έχει και άλλα διπλά στοιχεία, συνεπώς η διαδικασία πρέπει να επαναληφθεί έως ότου προκύψει η κενή λίστα.

Κατηγόρημα remove_duplicates/2

```
remove_duplicates([],[]).
remove_duplicates([H|T],Result) :-
    member(H,T),
    remove_duplicates(T,Result).
remove_duplicates([H|T],[H|Result])
    :-
    not_member(H,T),
    remove_duplicates(T,Result).
```

Ένας τερματικός κανόνας για την κενή λίστα

Δύο αναδρομικοί κανόνες

Όταν η κεφαλή υπάρχει πιο «μέσα» στη λίστα

Όταν η κεφαλή ΔΕΝ υπάρχει πιο «μέσα» στη λίστα

Συνένωση Λιστών

- Το κατηγόρημα **append** δέχεται τρία ορίσματα
  - Τα δυο πρώτα, L1 και L2, είναι οι λίστες που συνενώνονται.
  - Το τρίτο όρισμα, L3, είναι το αποτέλεσμα της συνένωσης.
- Ανάλογα με τον τρόπο που θα κληθεί το κατηγόρημα, μπορεί να χρησιμοποιηθεί
  - είτε για να συνενώσει δύο δεδομένες λίστες σε μία,
  - είτε για να χωρίσει μία δεδομένη λίστα σε δύο.

Κατηγόρημα append/3

Παραδείγματα Χρήσης – Συνένωση Λιστών

?- append([a,b,c],[d,e,f],A).

A = [a,b,c,d,e,f]

?- append([b,c],[d,e,f],A).

A = [b,c,d,e,f]

?- append([c],[d,e,f],A).

A = [c,d,e,f]

?- append([],[d,e,f],A).

A = [d,e,f]

Κατηγόρημα append/3
Παραδείγματα Χρήσης – Διαχωρισμός σε 2 Λίστες

```
?- append(X,Y,[a,b,c,d]).
X = [] , Y = [a,b,c,d] ;
X = [a] , Y = [b,c,d] ;
X = [a,b] , Y = [c,d] ;
X = [a,b,c] , Y = [d] ;
X = [a,b,c,d] , Y = [] ;
no
```

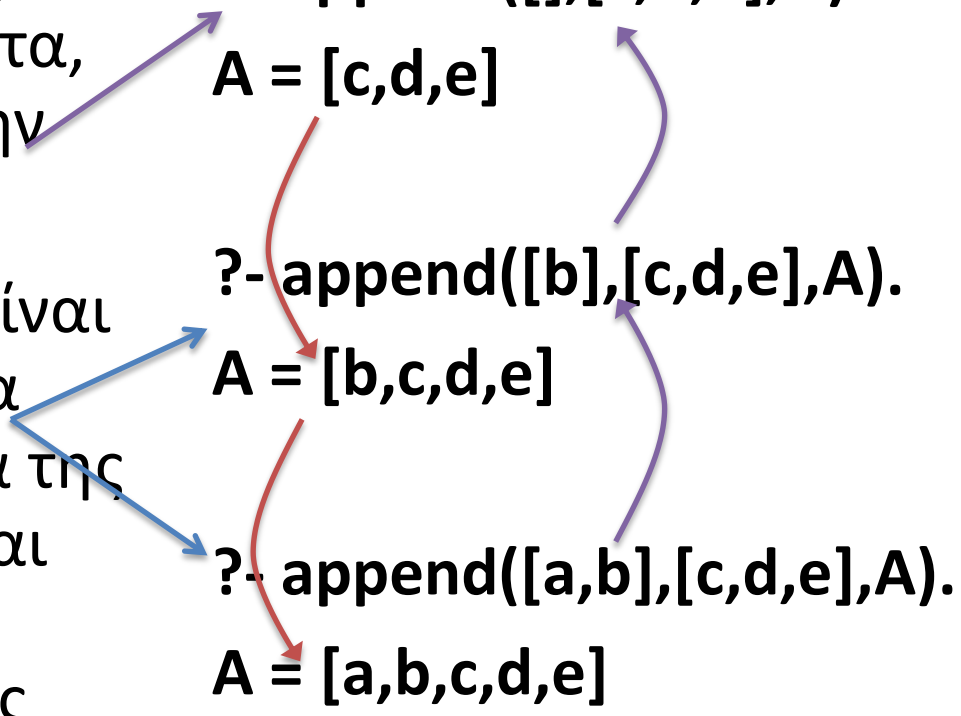
Κατηγόρημα append/3

Παραδείγματα Χρήσης – Εύρεση Αρχικού/Τελικού Τμήματος Λίστας

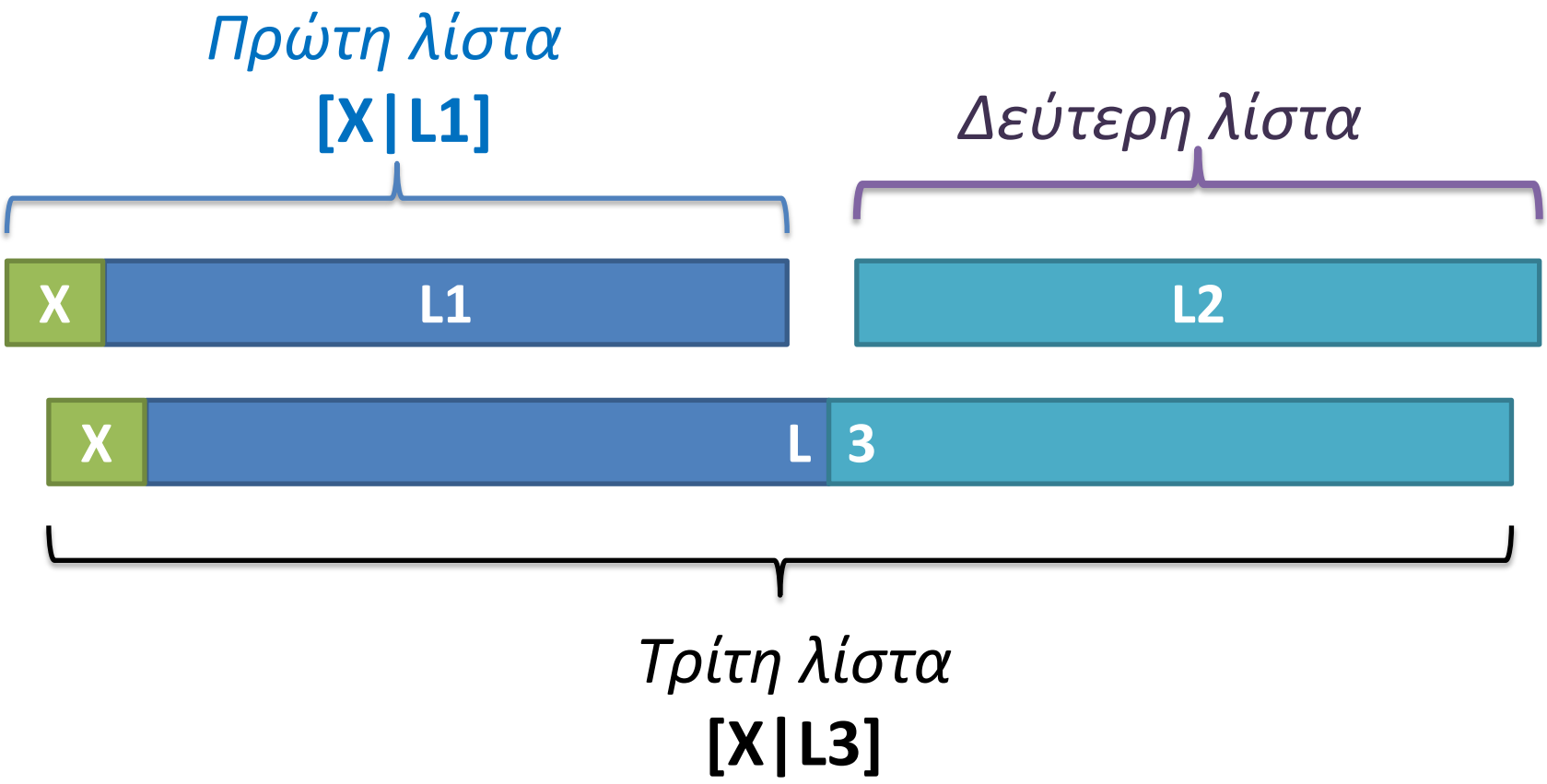
```
?- append(X,[c,d],[a,b,c,d]).
X = [a,b]
?- append([a,b],Y,[a,b,c,d]).
Y = [c,d]
```

Παρατηρήσεις

- Η συνένωση μιας κενής λίστας, με μια άλλη λίστα, δίνει ως αποτέλεσμα την άλλη λίστα.
- Αν η πρώτη λίστα δεν είναι κενή, τότε μπορούμε να συνενώσουμε την ουρά της με την δεύτερη λίστα και στο αποτέλεσμα να προστεθεί η κεφαλή της πρώτης λίστας.



Σχηματική αναπαράσταση συνένωσης



Κατηγόρημα append/3

append([],L,L).

append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

- Για να συνενώσεις 2 λίστες:
  - Αν η πρώτη λίστα είναι κενή, τότε το αποτέλεσμα ισούται με την δεύτερη λίστα (**L**).
  - Αλλιώς, συνένωσε αναδρομικά την ουρά της πρώτης λίστας (**L1**) με την δεύτερη λίστα (**L2**) και στο αποτέλεσμα που θα προκύψει (**L3**), πρόσθεσε την κεφαλή της πρώτης λίστας (**X**) – έτσι προκύπτει το τελικό αποτέλεσμα **[X|L3].**

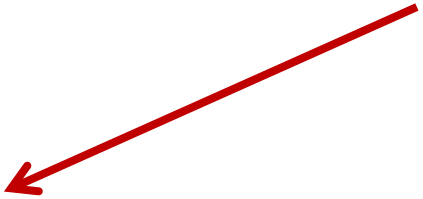
Εκτέλεση append/3

?- append([a,b],[c,d],A).

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

Εκτέλεση append/3

?- append([a,b],[c,d],A).

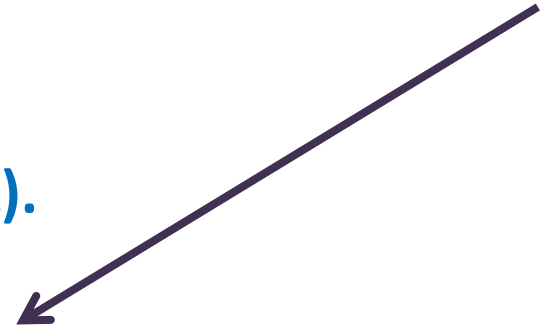


append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

Εκτέλεση append/3

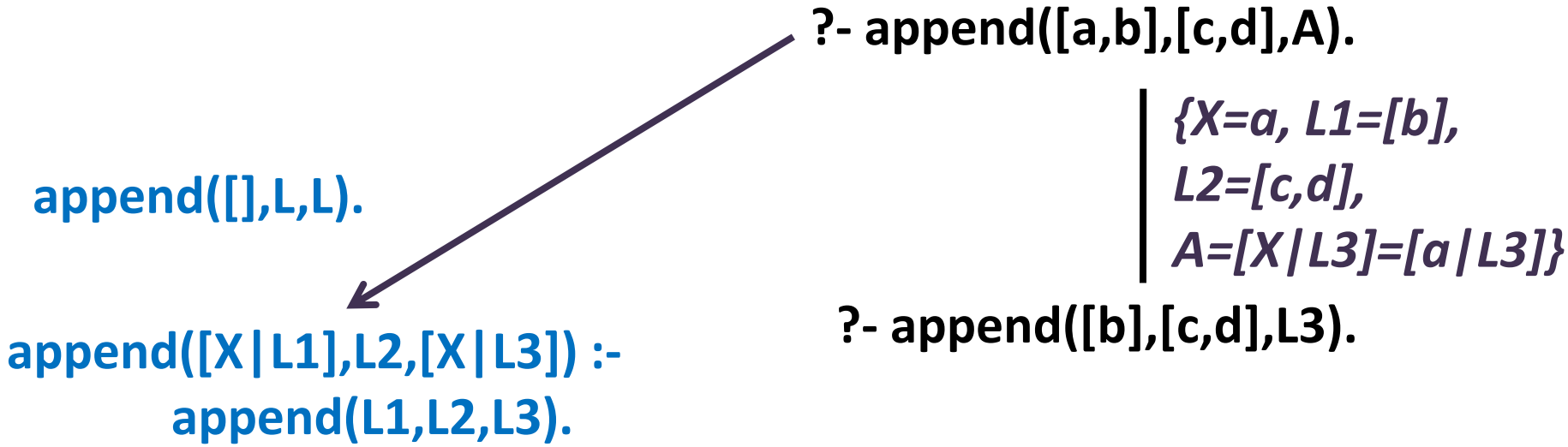
?- append([a,b],[c,d],A).



append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

Εκτέλεση append/3



Εκτέλεση append/3

?- append([a,b],[c,d],A).

{X=a, L1=[b],
L2=[c,d],
A=[X|L3]=[a|L3]}

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

?- append([b],[c,d],L3).

Εκτέλεση append/3

?- append([a,b],[c,d],A).

{X=a, L1=[b],
L2=[c,d],
A=[X|L3]=[a|L3]}

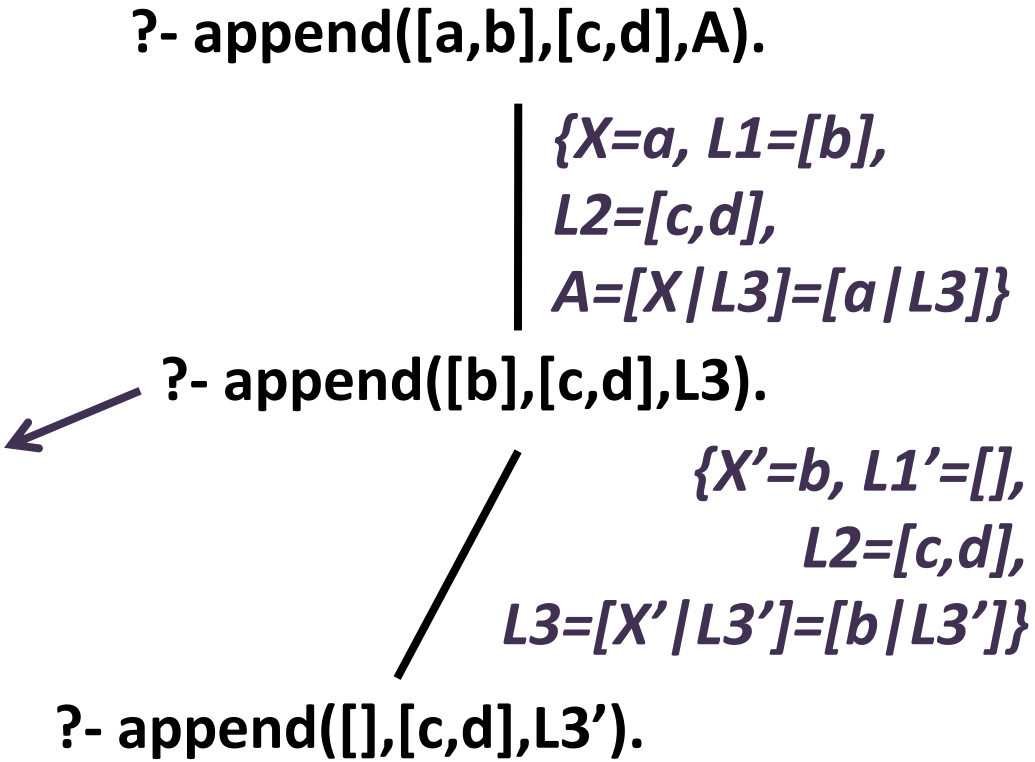
append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

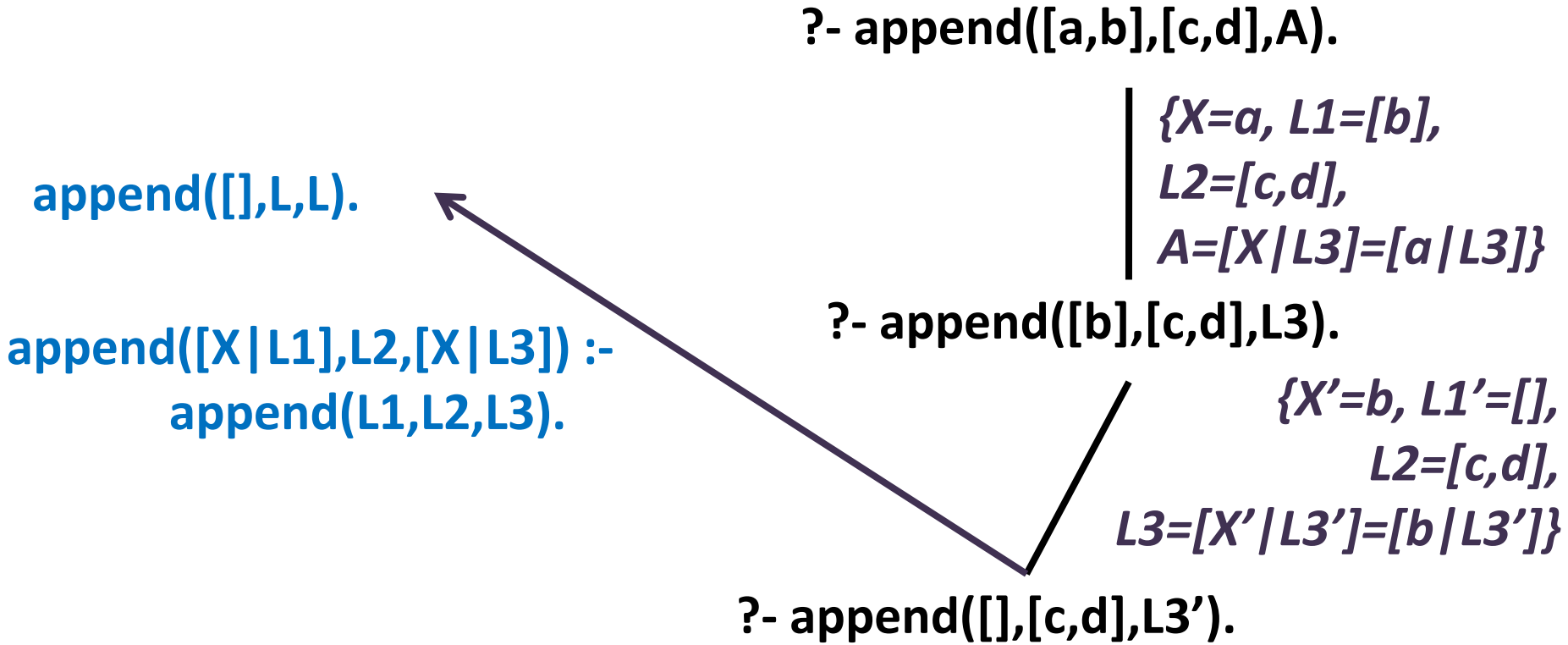
?- append([b],[c,d],L3).

Εκτέλεση append/3

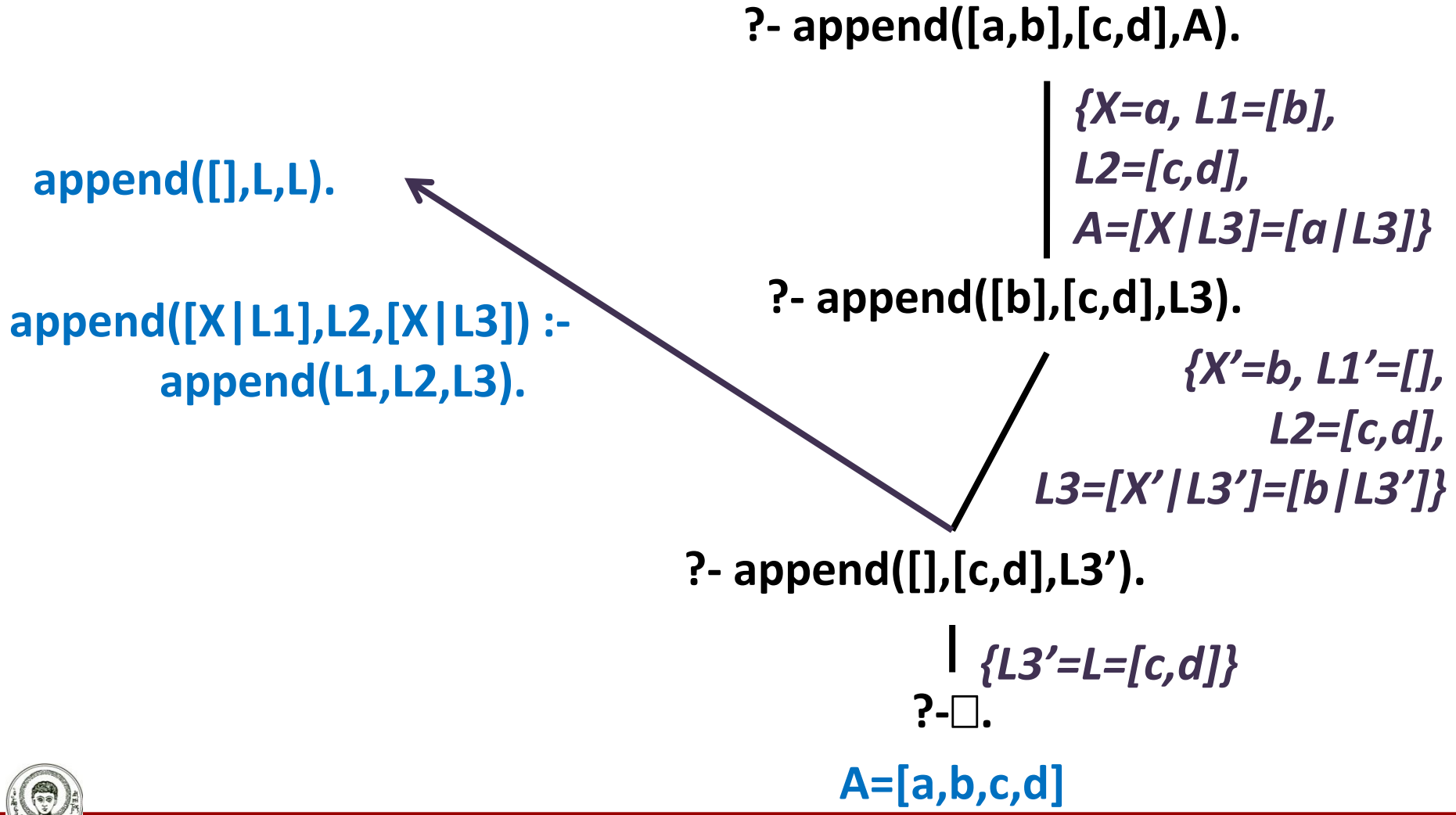
```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```



Εκτέλεση append/3



Εκτέλεση append/3



Εκτέλεση append/3

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

A=[a|L3]=
=[a|[b|L3']]=
=[a|[b|[c,d]]]=
=[a|[b,c,d]]=
=[a,b,c,d]

?- append([a,b],[c,d],A).

{X=a, L1=[b],
L2=[c,d],
A=[X|L3]=[a|L3]}

?- append([b],[c,d],L3).

{X'=b, L1'=[],
L2=[c,d],
L3=[X'|L3']=[b|L3']}

?- append([],[c,d],L3').

{L3'=L=[c,d]}

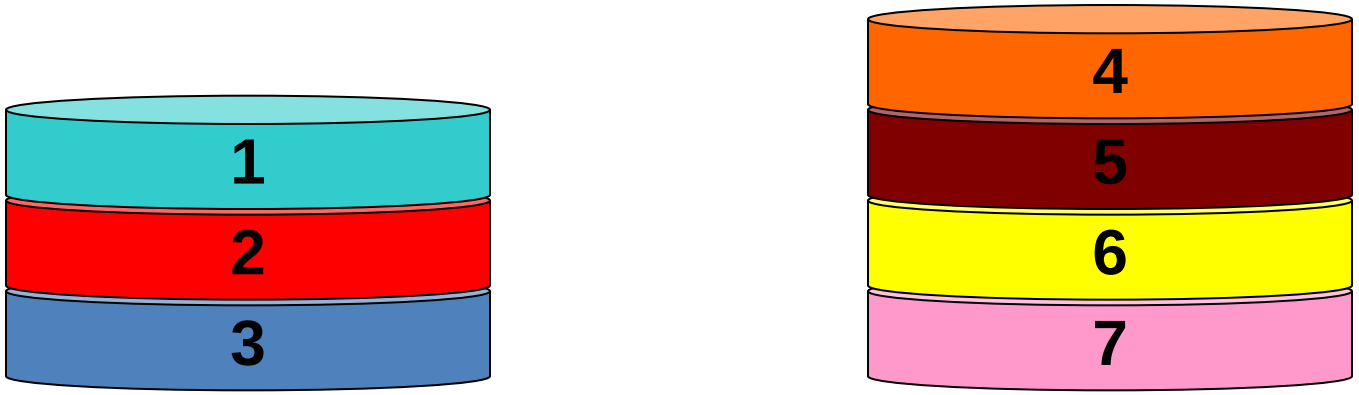
?-□.

A=[a,b,c,d]

Παρομοίωση με Στοίβες (LIFO)

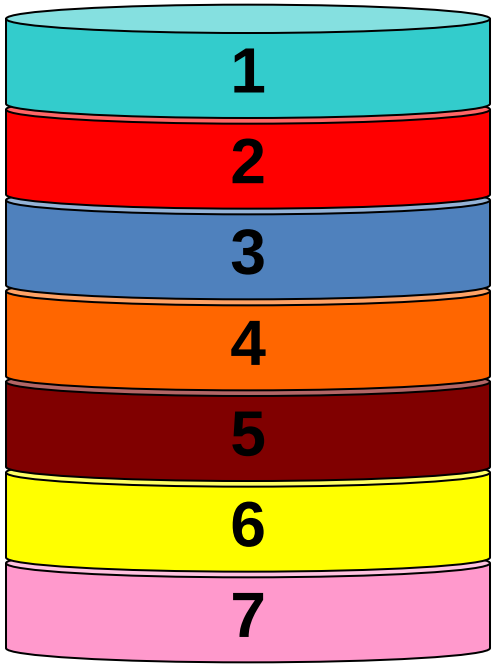
- Έχουμε 2 στοίβες από αντικείμενα (π.χ. πιάτα).
- Πρέπει να ενώσουμε τις 2 στοίβες σε 1.
- Κάθε φορά μπορούμε να μετακινούμε μόνο 1 πιάτο από την κορυφή της στοίβας.
- Στο τραπέζι έχουμε όσο χώρο θέλουμε για να αφήνουμε τα πιάτα.

Αρχική Κατάσταση

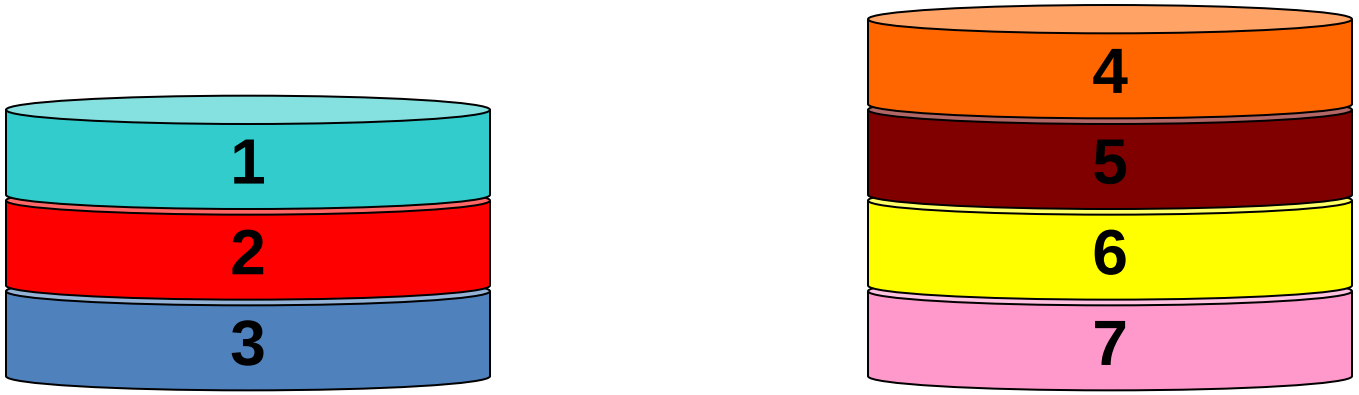


Τελική Κατάσταση

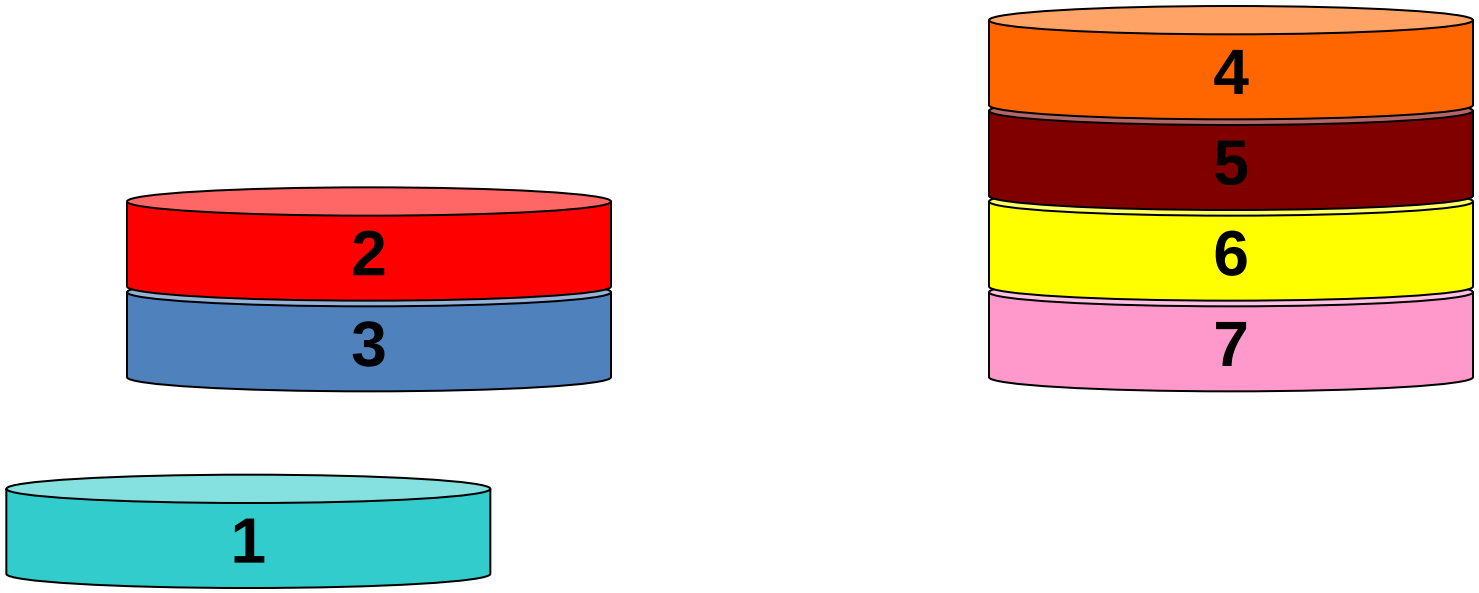
Άδεια στοίβα



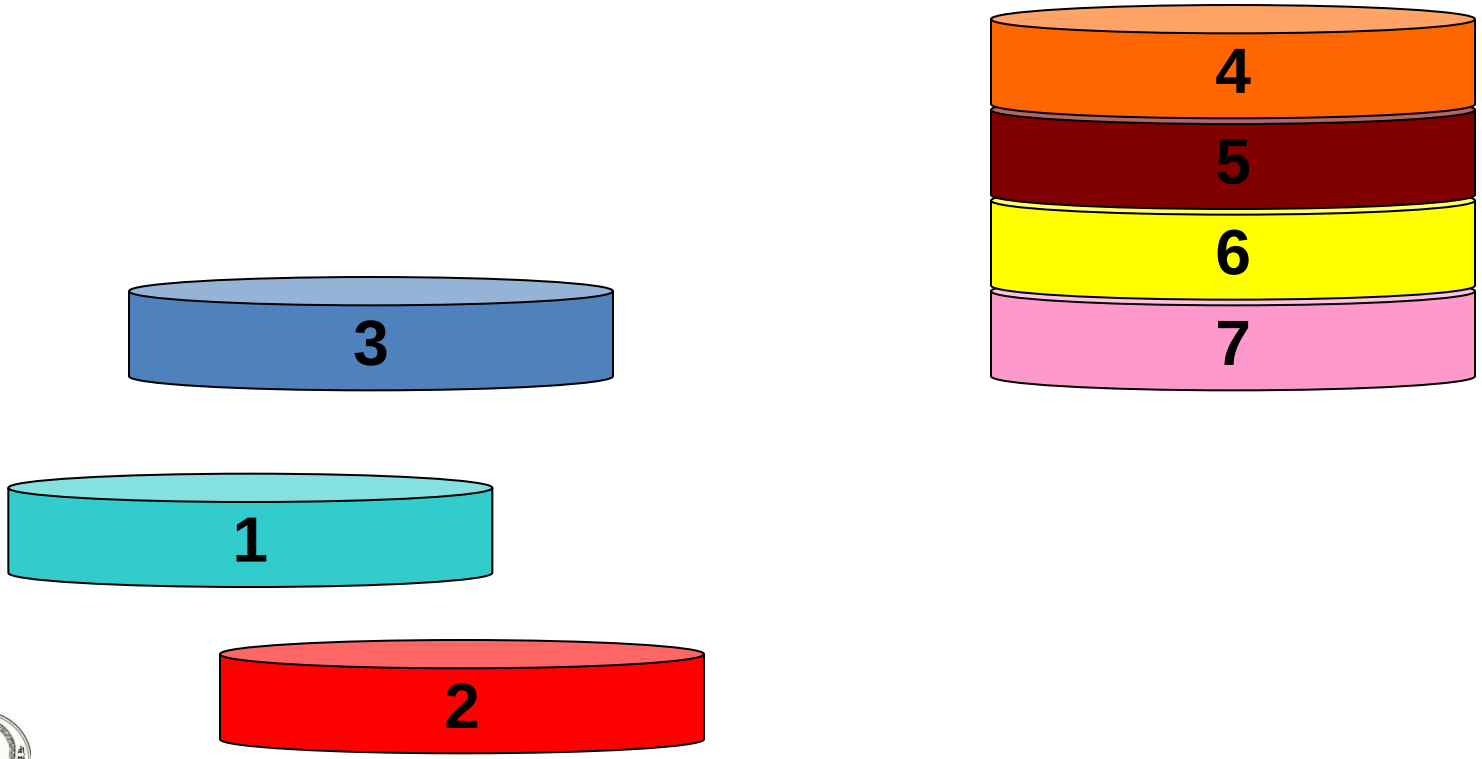
Επίλυση – Βήμα 1



Επίλυση – Βήμα 2

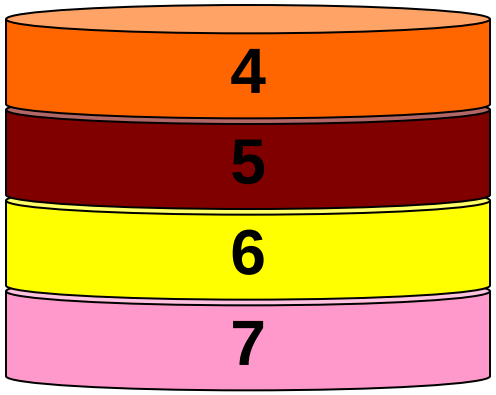


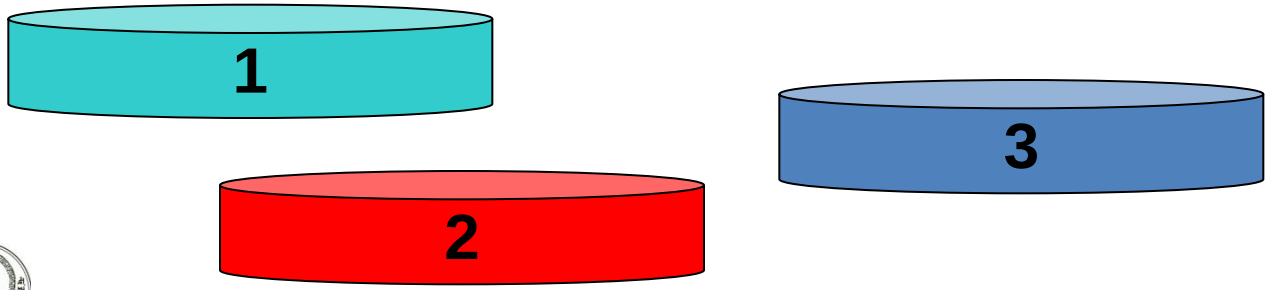
Επίλυση – Βήμα 3



Επίλυση – Βήμα 4

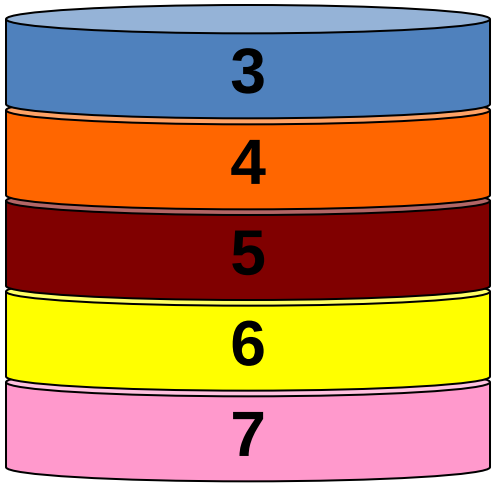
Άδεια στοίβα

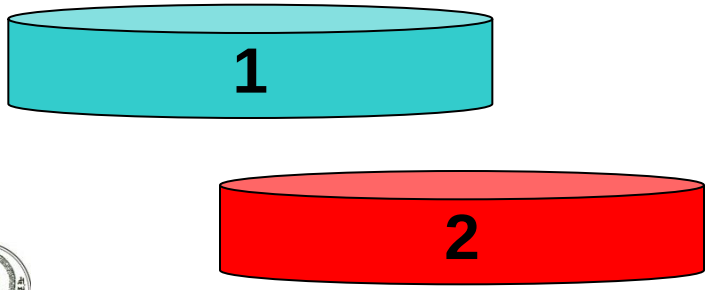




Επίλυση – Βήμα 5

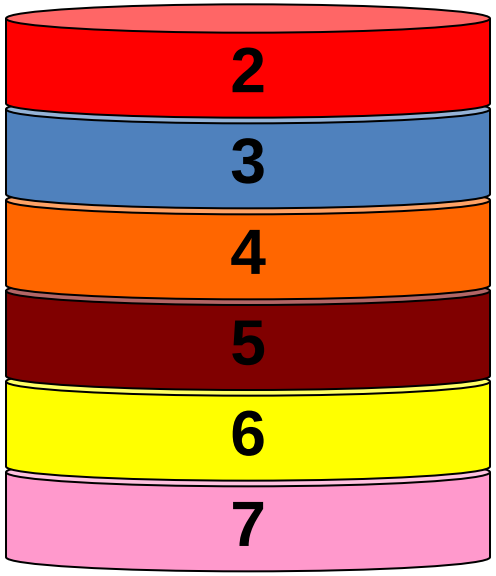
Άδεια στοίβα

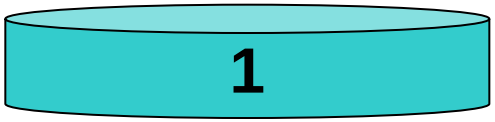




Επίλυση – Βήμα 6

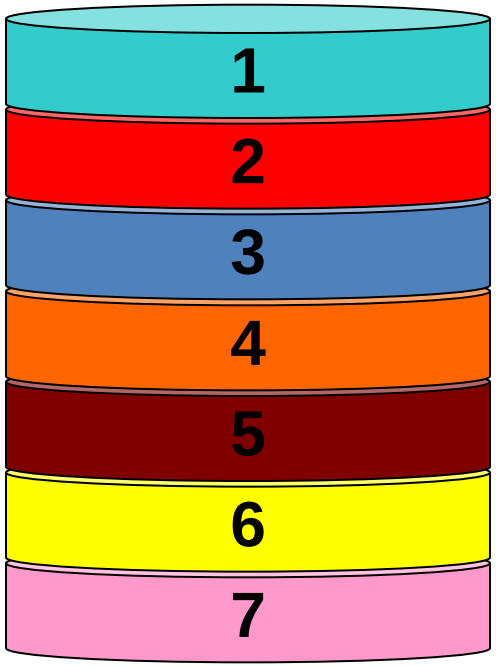
Άδεια στοίβα





Επίλυση – Βήμα 7

Άδεια στοίβα



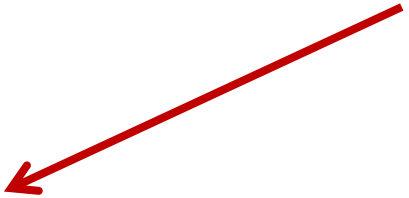
Εκτέλεση append/3

?- append([a,b],A,[a,b,c,d]).

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

Εκτέλεση append/3

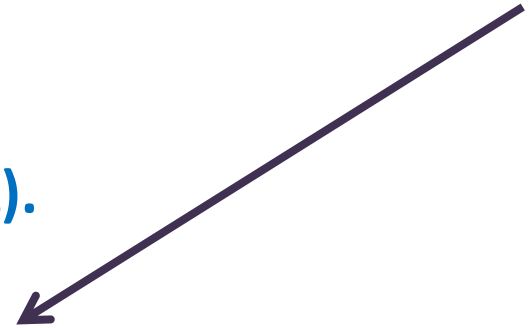
?- append([a,b],A,[a,b,c,d]).



```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

Εκτέλεση append/3

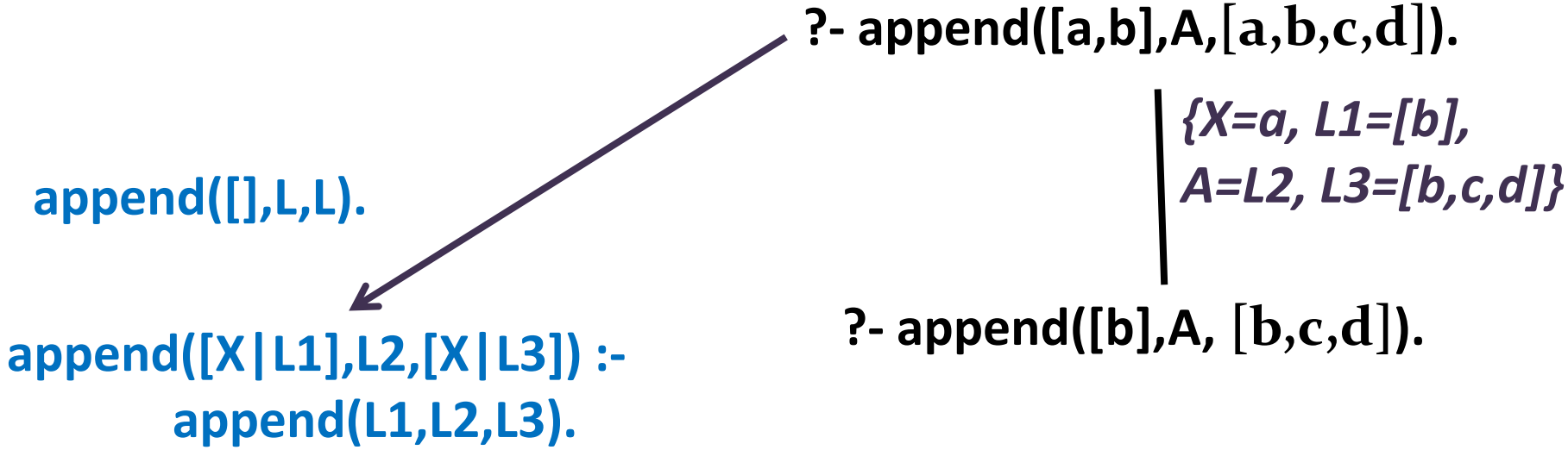
?- append([a,b],A,[a,b,c,d]).



append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

Εκτέλεση append/3



Εκτέλεση append/3

?- append([a,b],A,[a,b,c,d]).

{X=a, L1=[b],
A=L2, L3=[b,c,d]}

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

?- append([b],A, [b,c,d]).

Εκτέλεση append/3

?- append([a,b],A,[a,b,c,d]).

{X=a, L1=[b],
A=L2, L3=[b,c,d]}

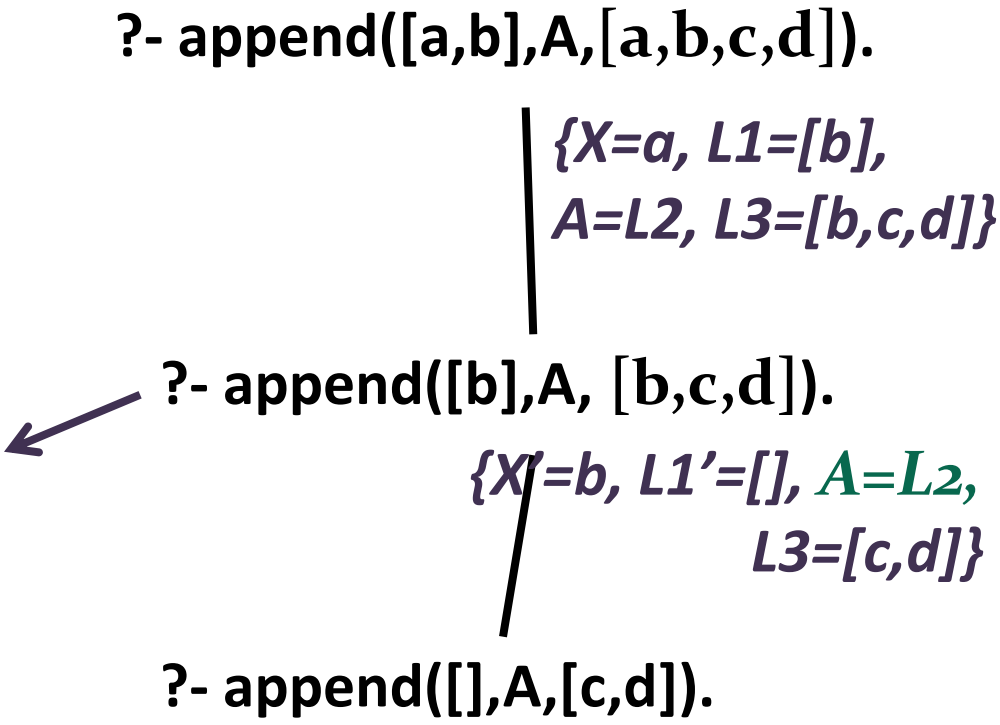
append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

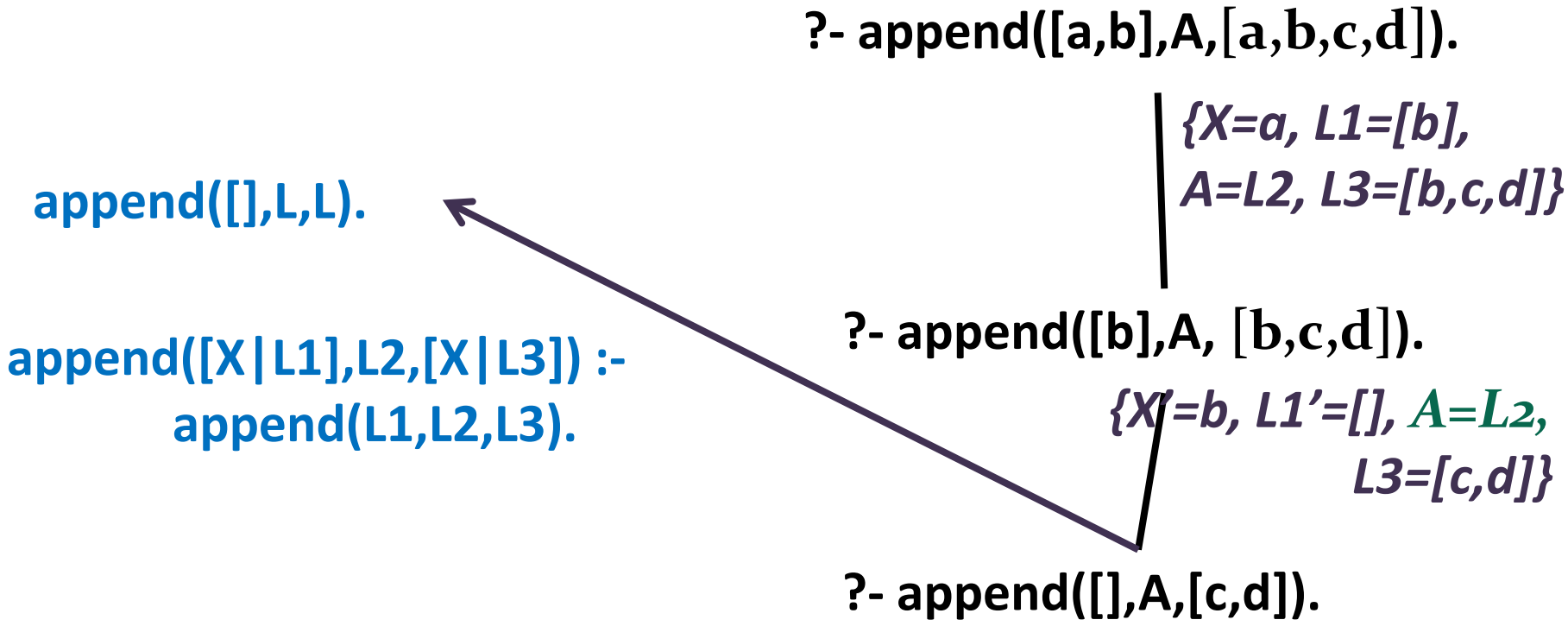
?- append([b],A, [b,c,d]).

Εκτέλεση append/3

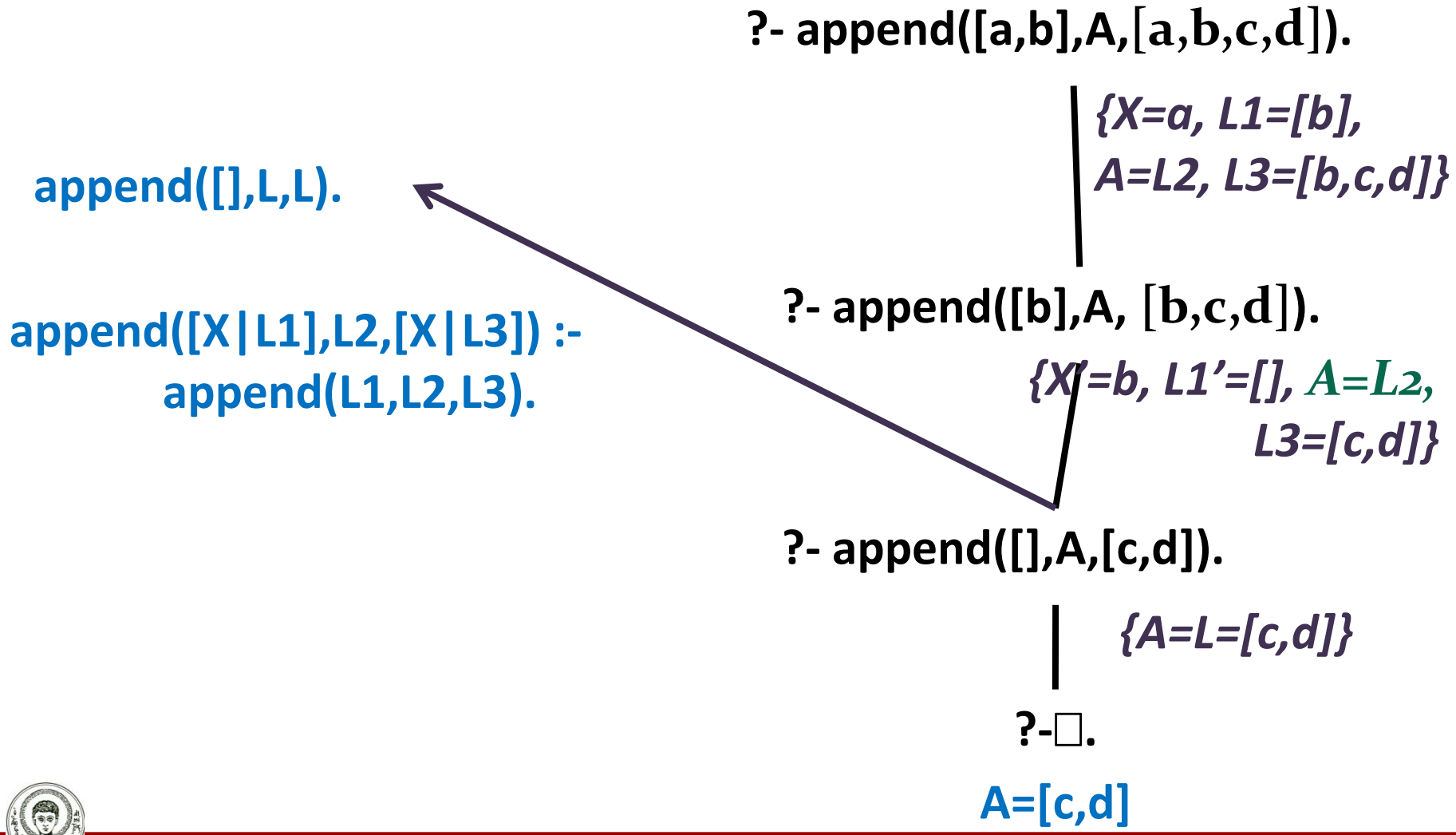
```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```



Εκτέλεση append/3



Εκτέλεση append/3



Εκτέλεση append/3

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

?- append([a,b],A,[a,b,c,d]).

{X=a, L1=[b],
A=L2, L3=[b,c,d]}

?- append([b],A, [b,c,d]).

{X=b, L1'=[], A=L2,
L3=[c,d]}

?- append([],A,[c,d]).

{A=L=[c,d]}

?-□.

A=[c,d]

Εκτέλεση append/3

?- append(A,[c,d], [a,b,c,d]).

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

Εκτέλεση append/3

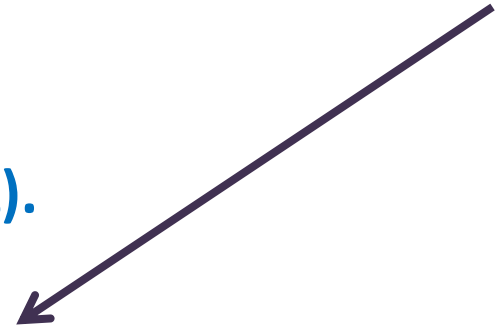
?- append(A,[c,d], [a,b,c,d]).

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

Εκτέλεση append/3

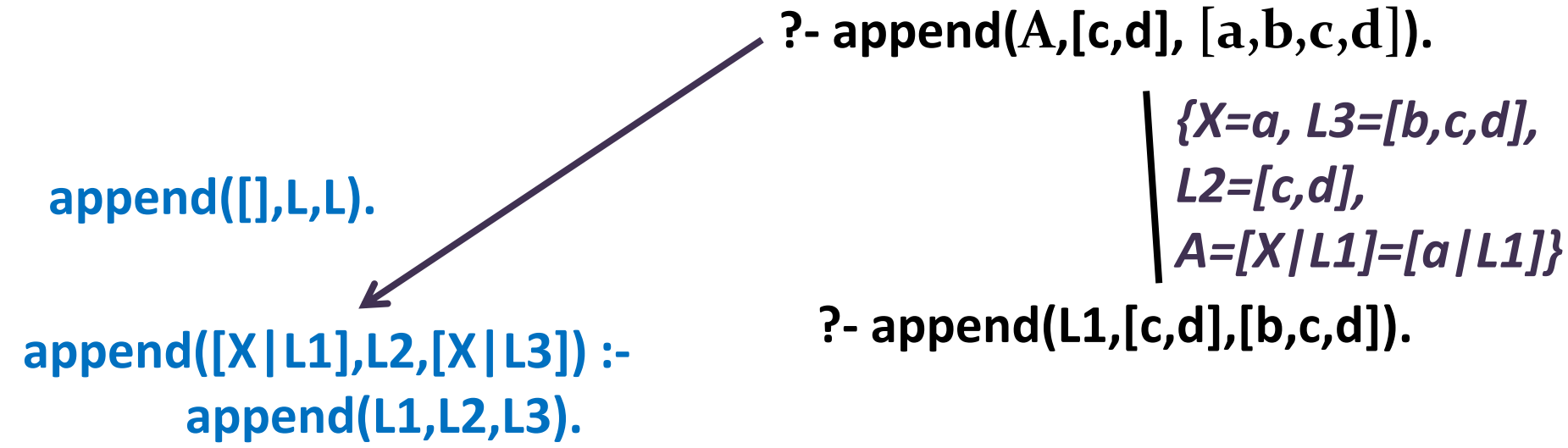
?- append(A,[c,d], [a,b,c,d]).



append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

Εκτέλεση append/3



Εκτέλεση append/3

?- append(A,[c,d], [a,b,c,d]).

{X=a, L3=[b,c,d],
L2=[c,d],
A=[X|L1]=[a|L1]}

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

?- append(L1,[c,d],[b,c,d]).

Εκτέλεση append/3

?- append(A,[c,d], [a,b,c,d]).

{X=a, L3=[b,c,d],
L2=[c,d],
A=[X|L1]=[a|L1]}

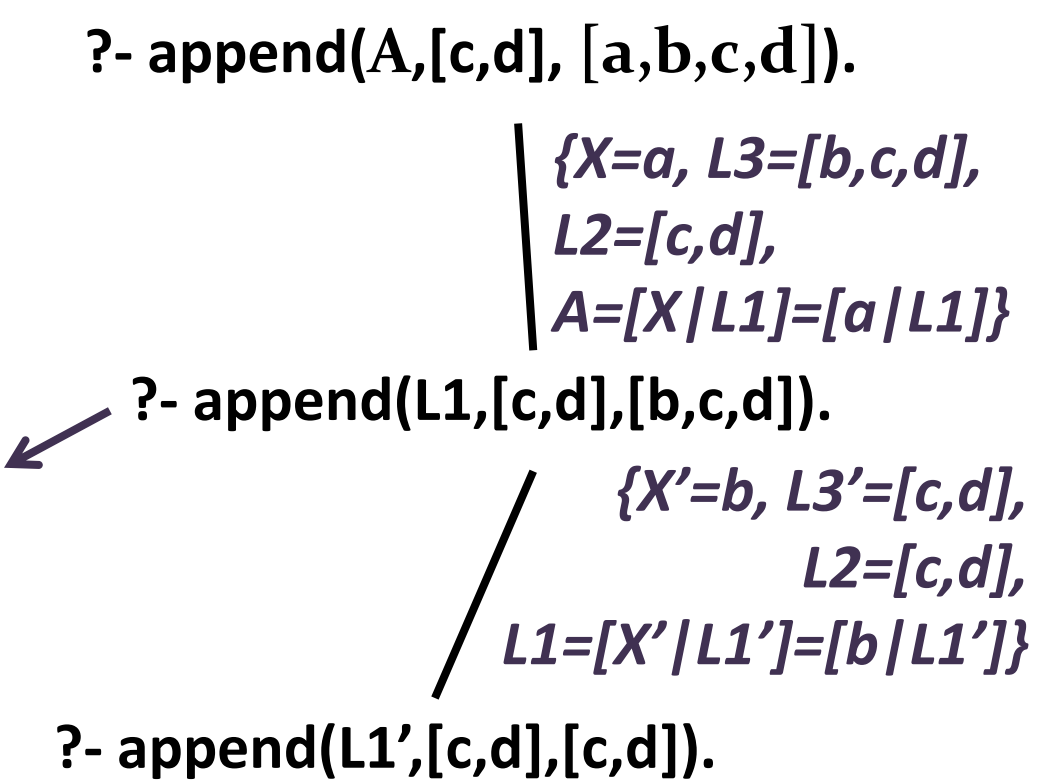
append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

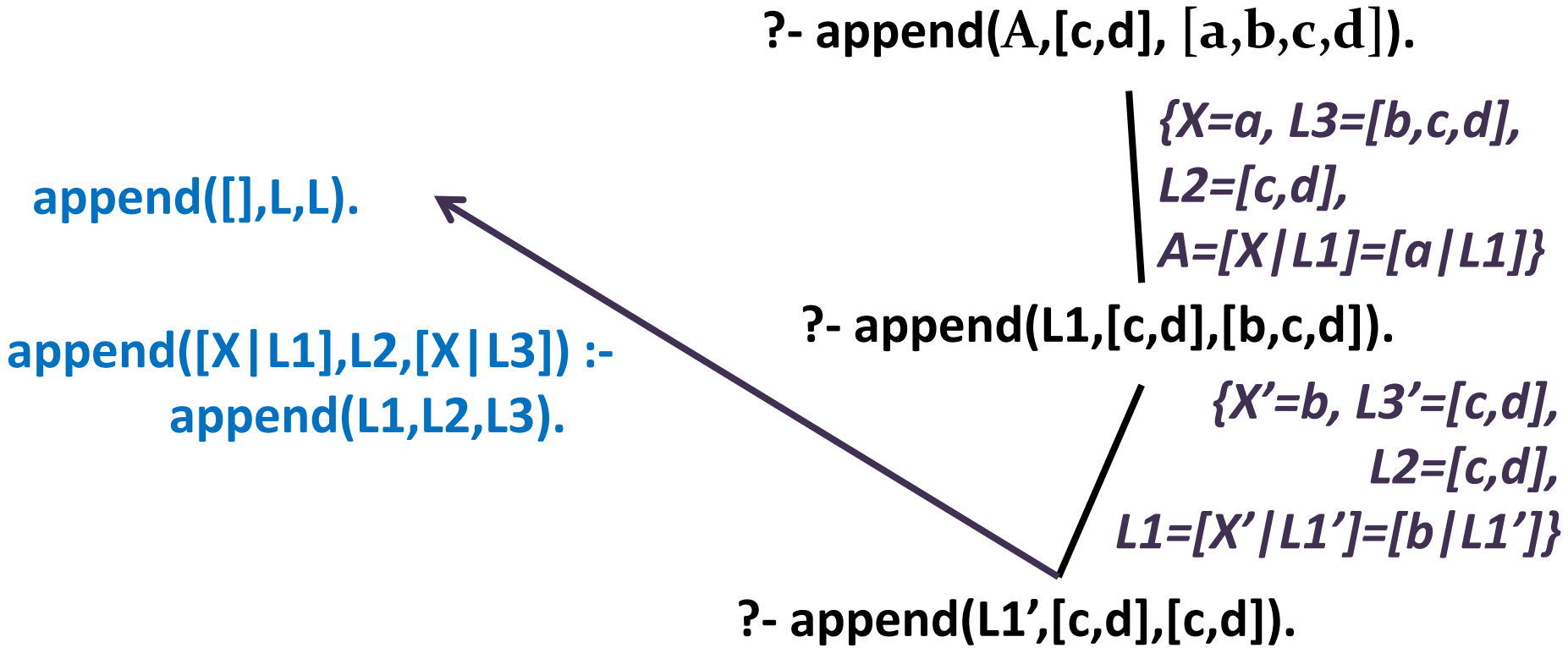
?- append(L1,[c,d],[b,c,d]).

Εκτέλεση append/3

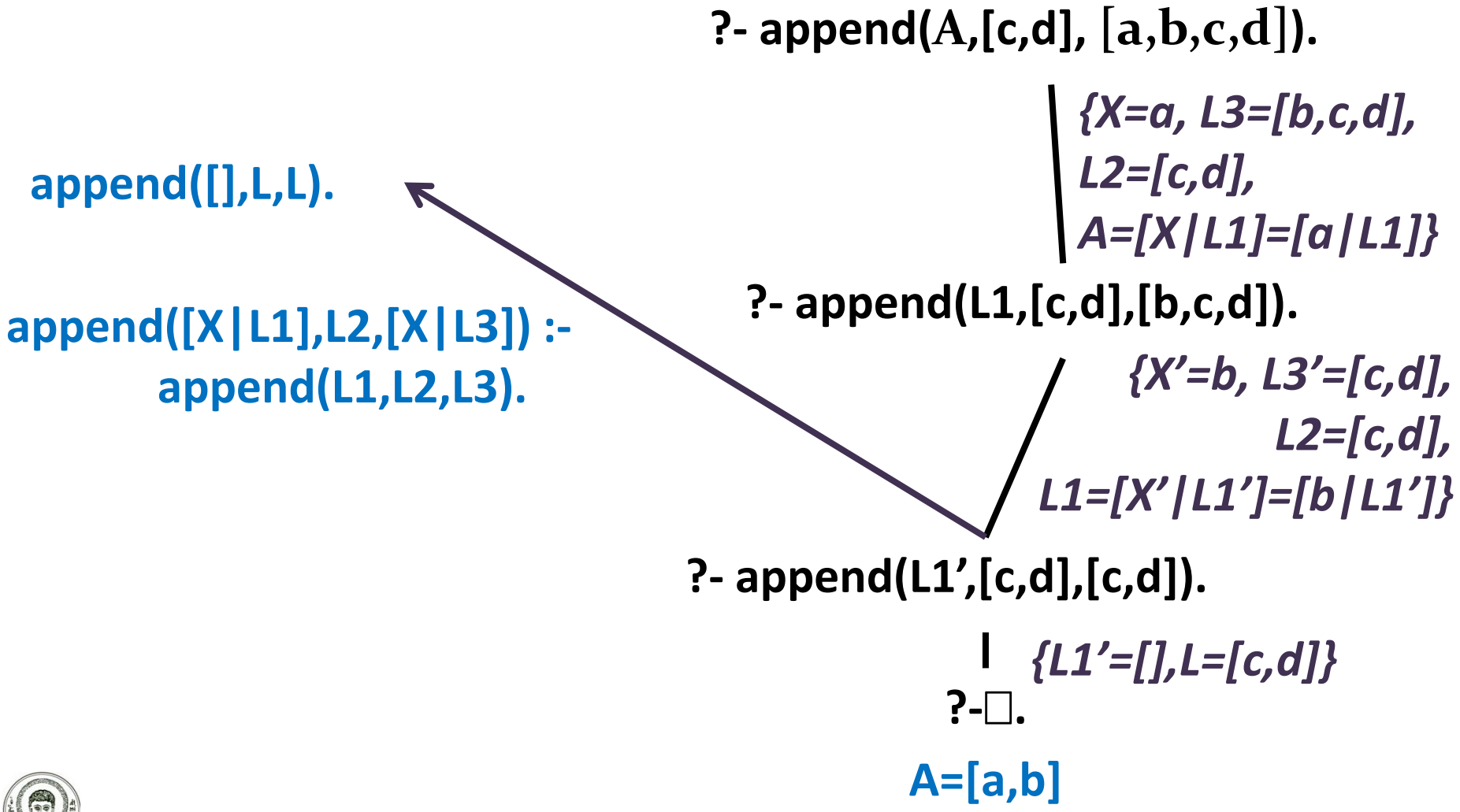
```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```



Εκτέλεση append/3



Εκτέλεση append/3



Εκτέλεση append/3

append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

?- append(A,[c,d], [a,b,c,d]).

{X=a, L3=[b,c,d],
L2=[c,d],
A=[X|L1]=[a|L1]}

?- append(L1,[c,d],[b,c,d]).

{X'=b, L3'=[c,d],
L2=[c,d],
L1=[X'|L1']=[b|L1']}

?- append(L1',[c,d],[c,d]).

{L1'=[],L=[c,d]}

?-□.

A=[a,b]

A=[a|L1]=
=[a|[b|L1']]=
=[a|[b|[]]]=
=[a|[b]]=
=[a,b]

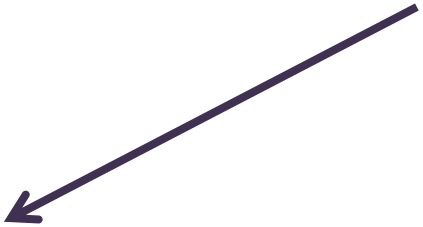
Εκτέλεση append/3

?- append(A,B,[a,b,c,d]).

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

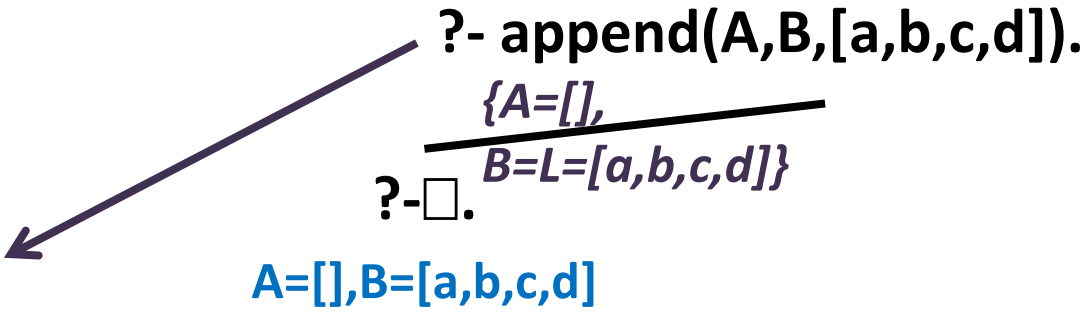
Εκτέλεση append/3

?- append(A,B,[a,b,c,d]).

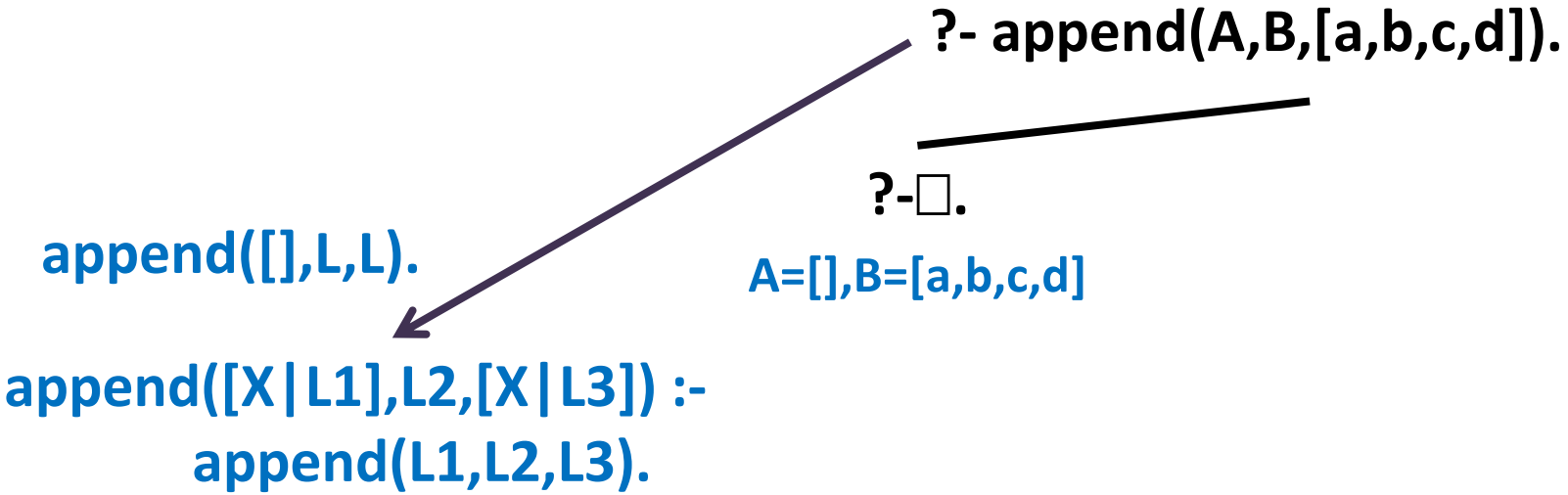


```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

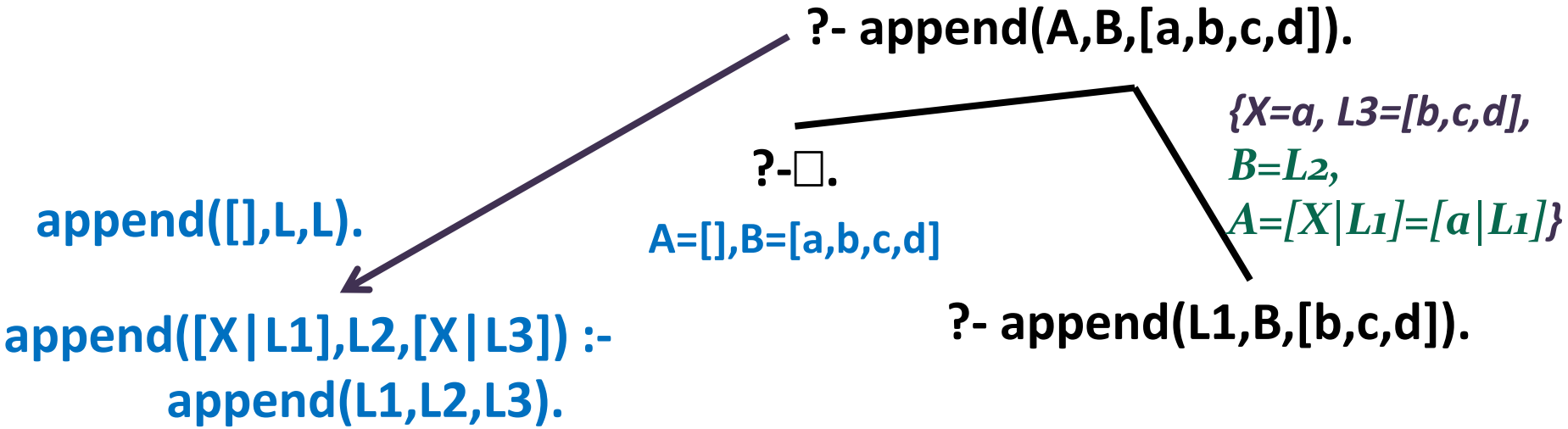
Εκτέλεση append/3



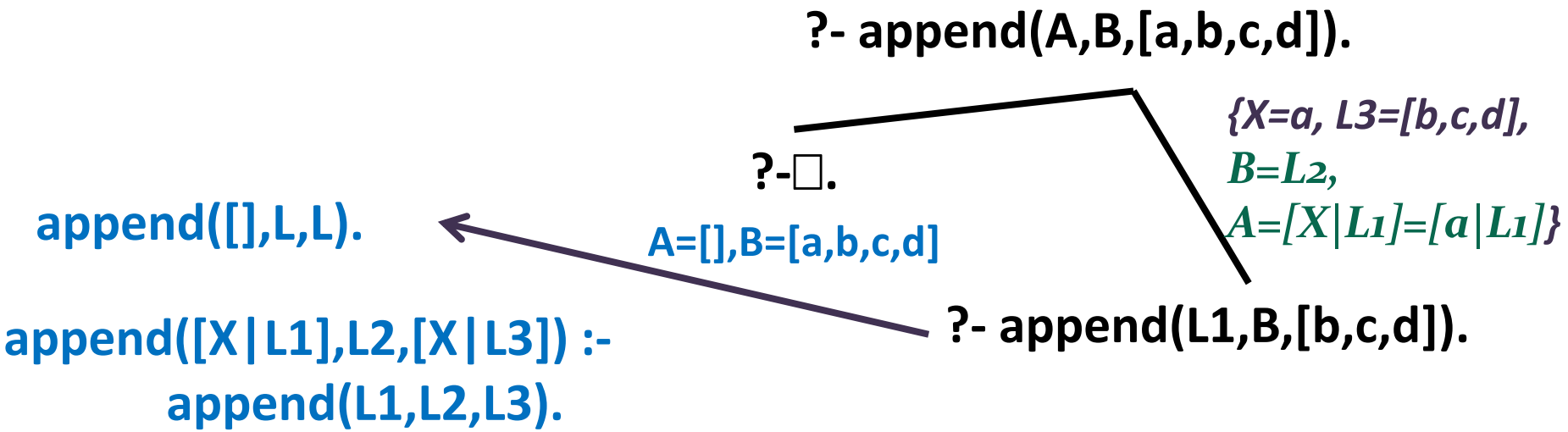
Εκτέλεση append/3



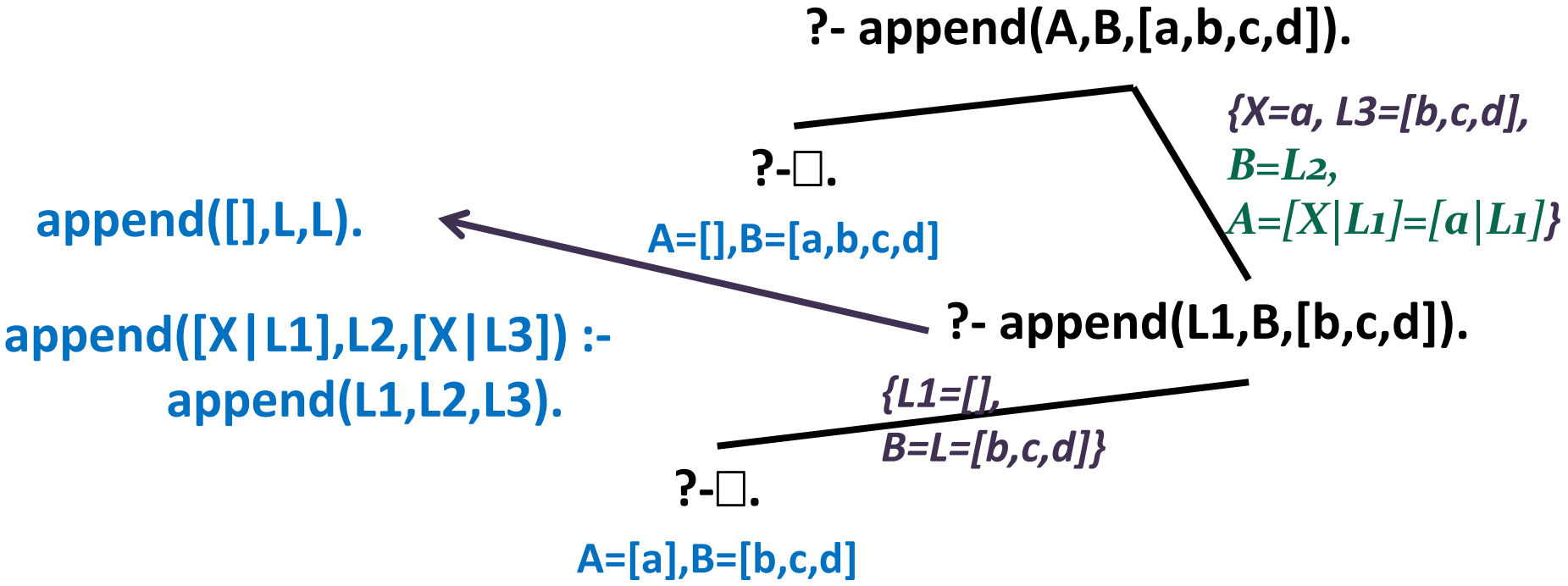
Εκτέλεση append/3



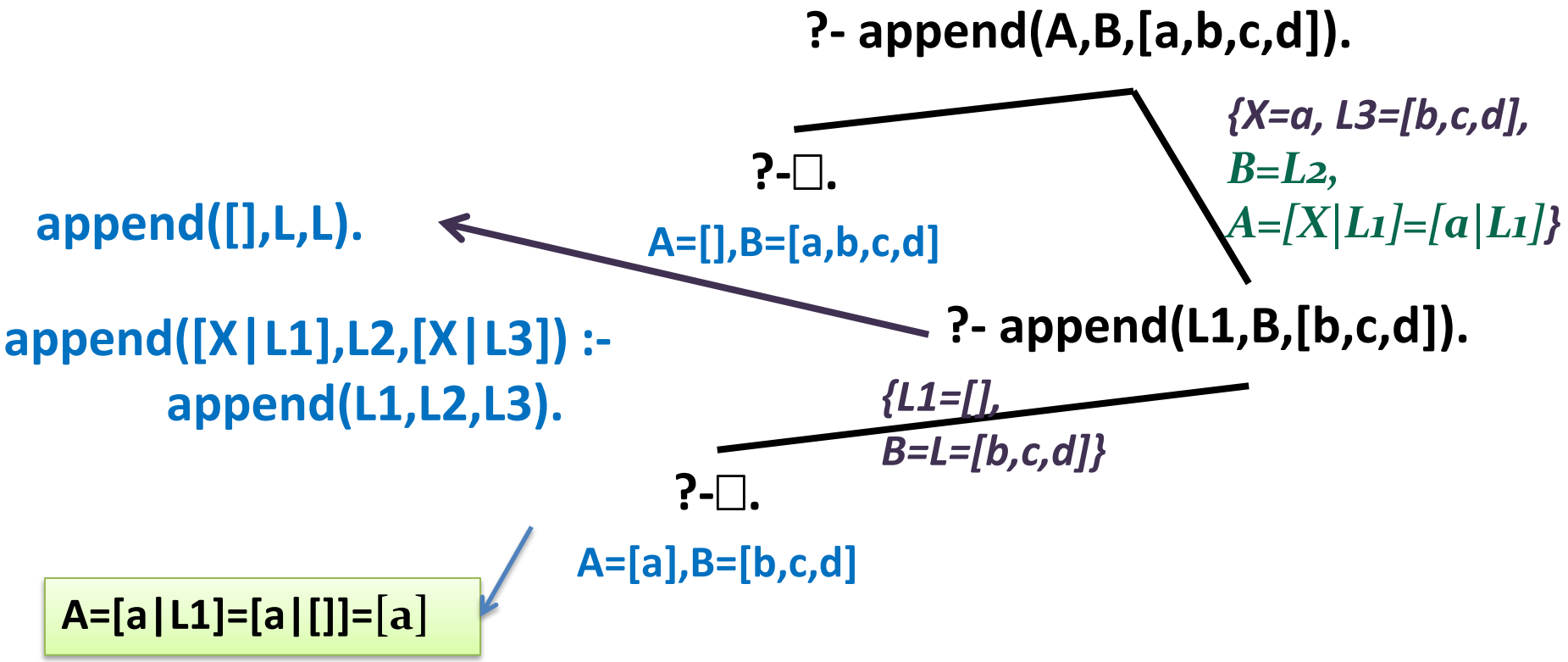
Εκτέλεση append/3



Εκτέλεση append/3



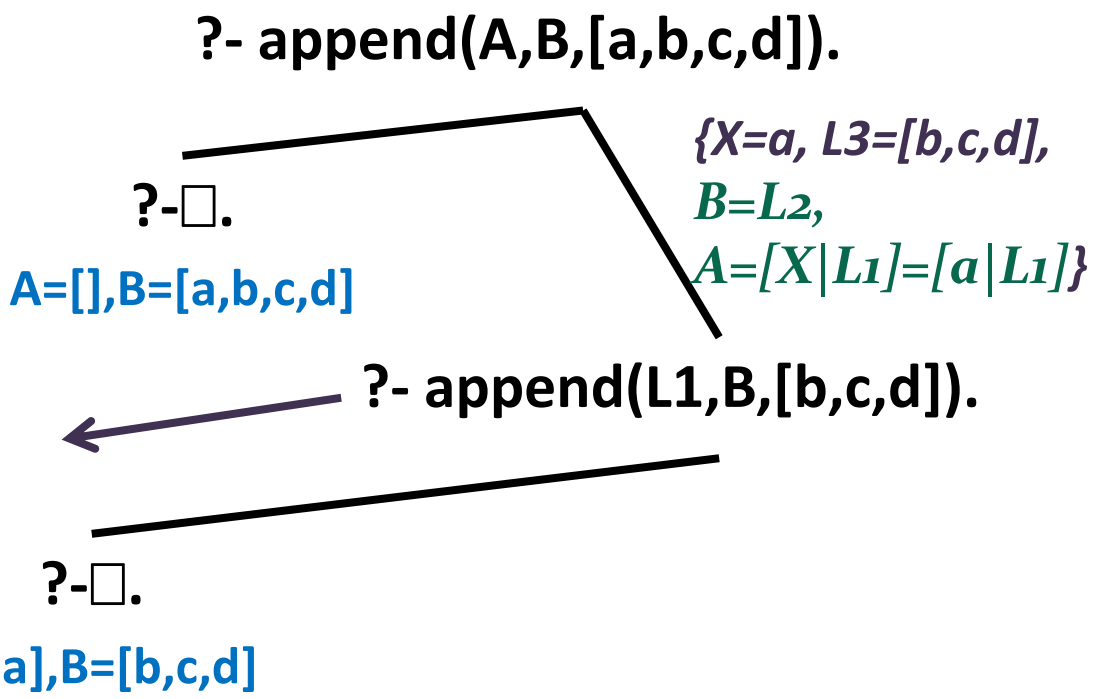
Εκτέλεση append/3



Εκτέλεση append/3

append([],L,L).

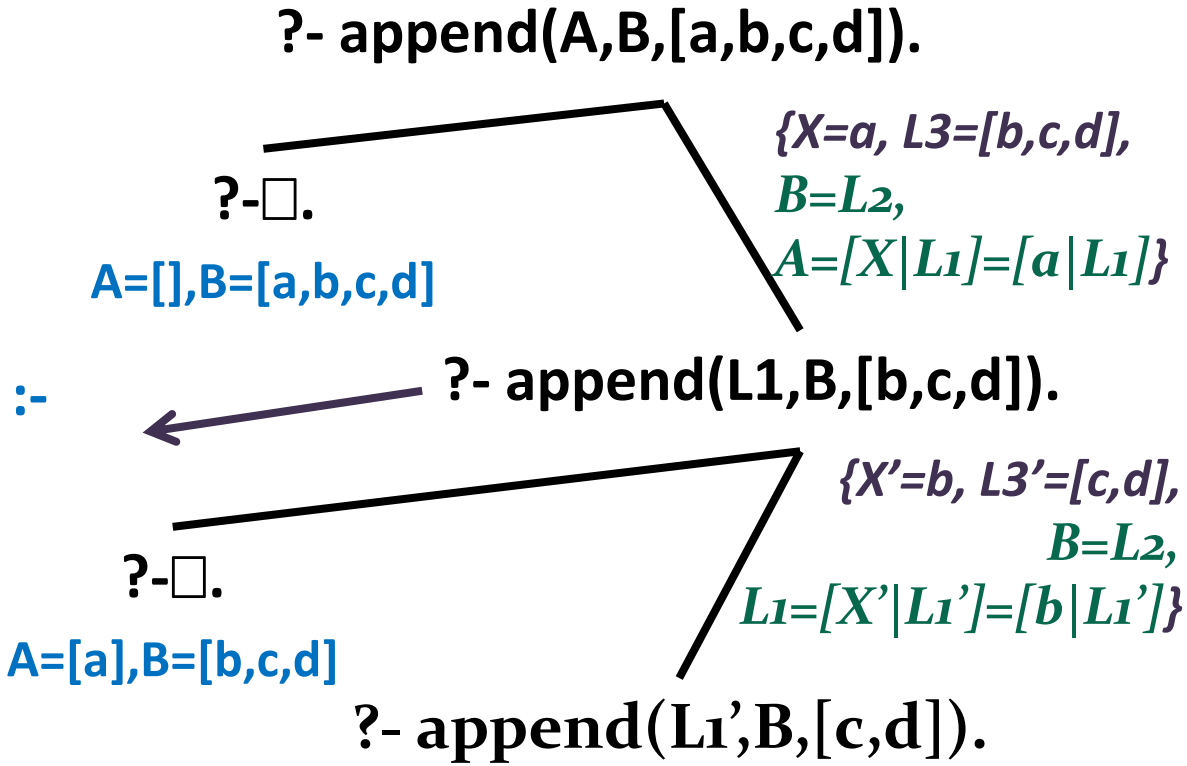
append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).



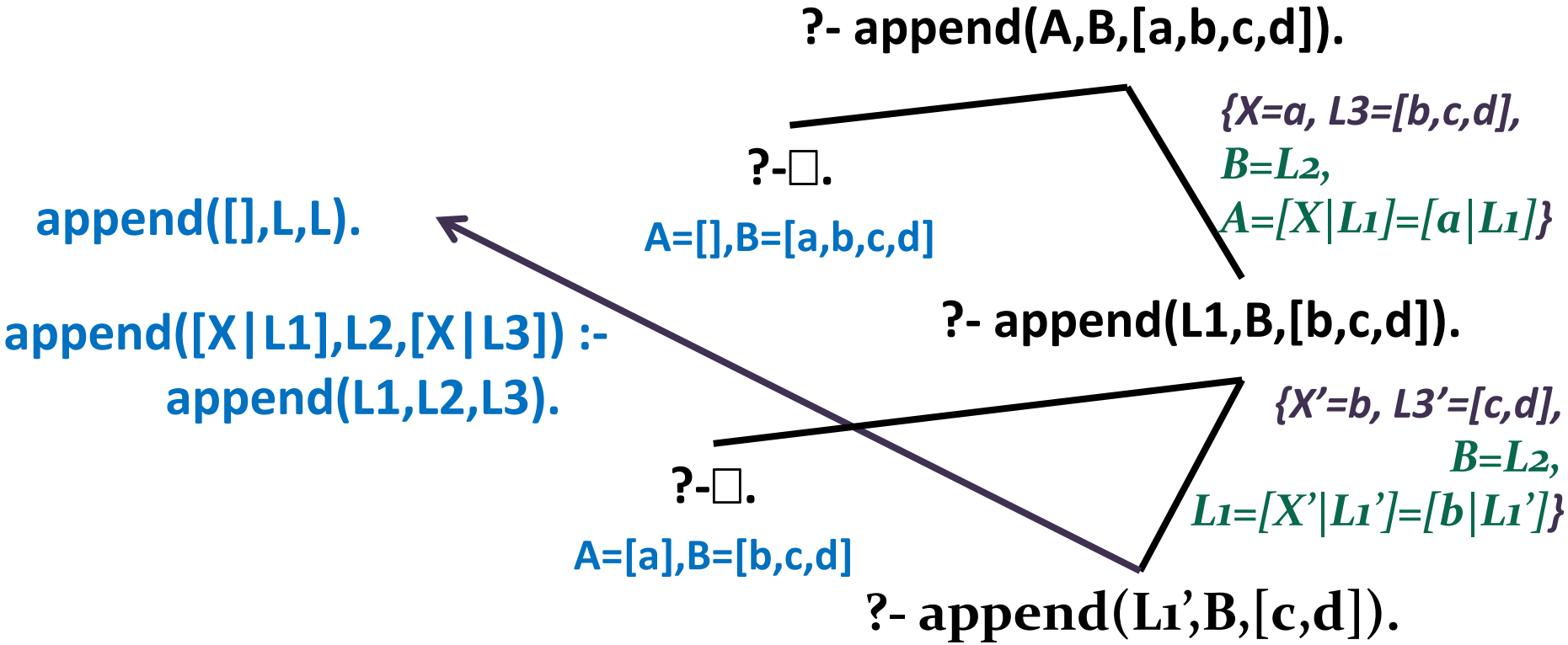
Εκτέλεση append/3

append([],L,L).

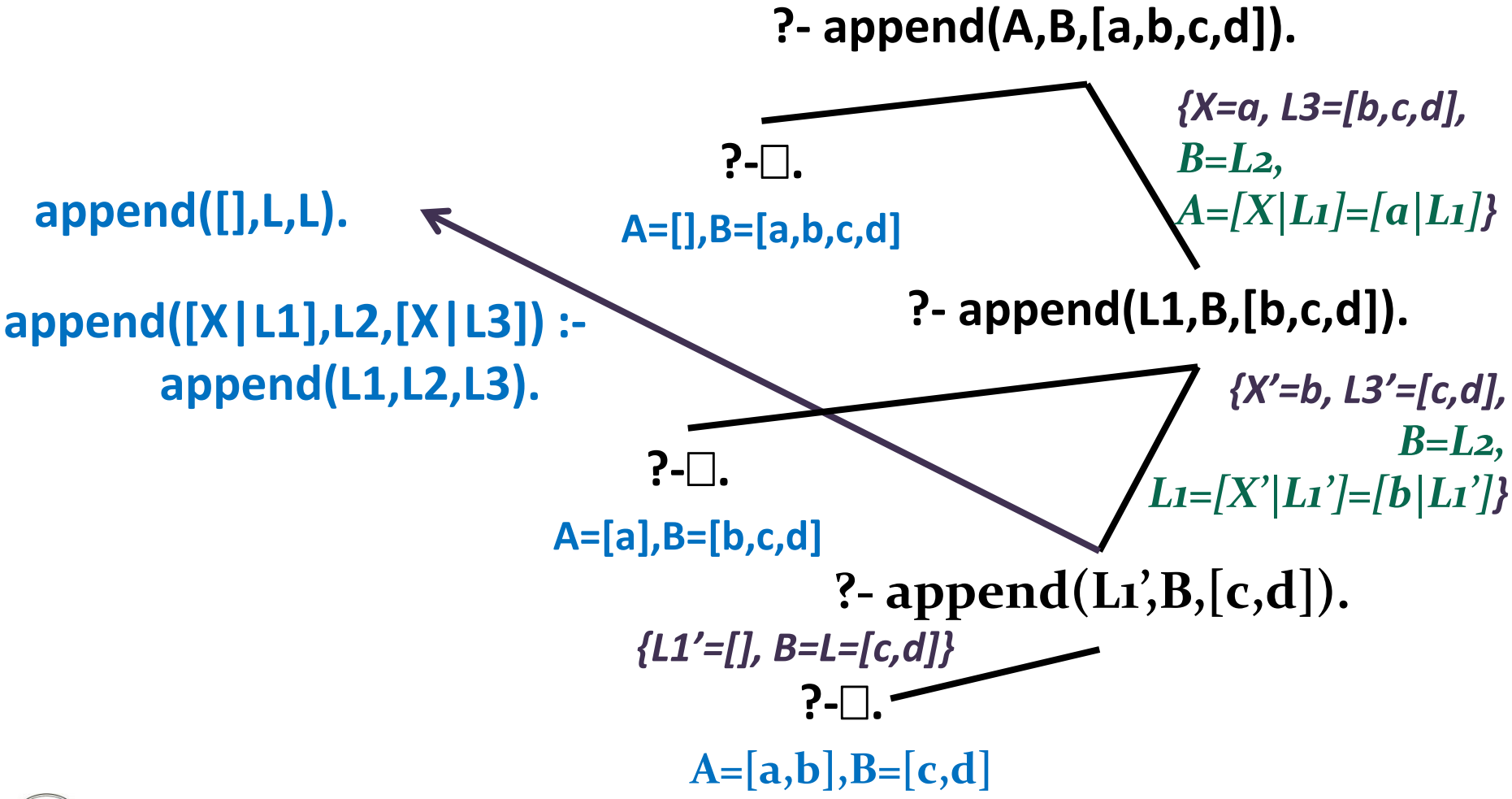
append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).



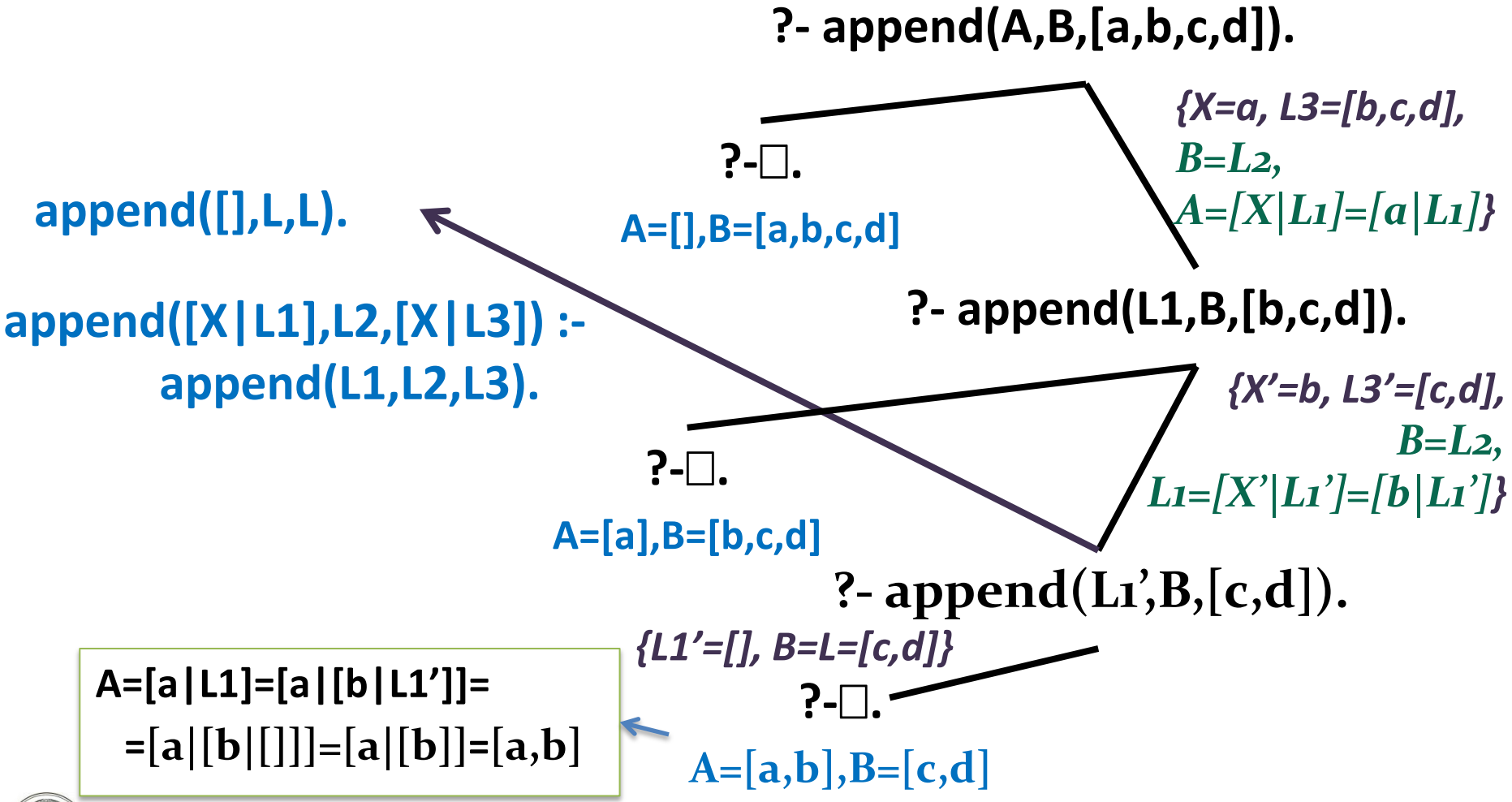
Εκτέλεση append/3



Εκτέλεση append/3



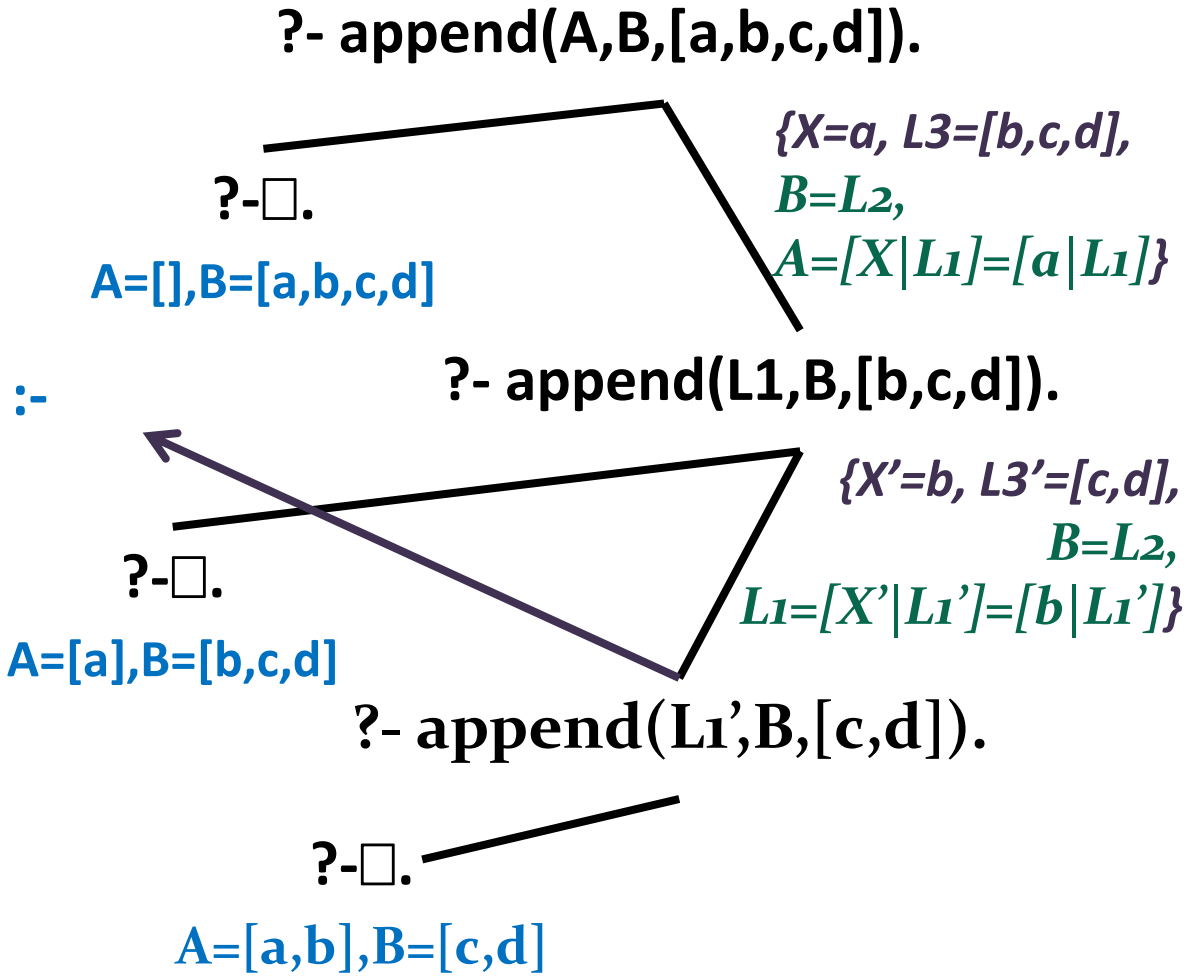
Εκτέλεση append/3



Εκτέλεση append/3

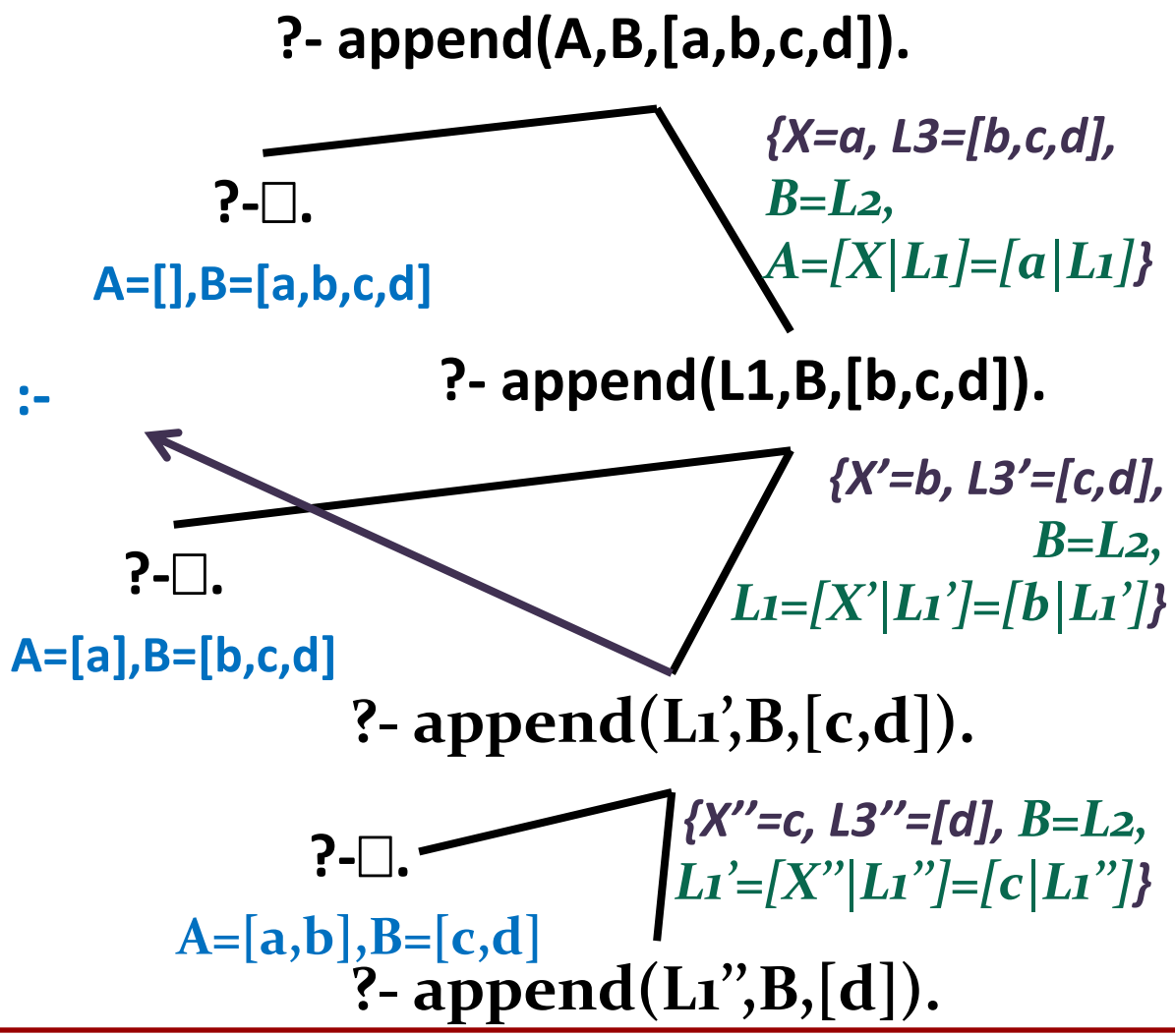
append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).



Εκτέλεση append/3

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```



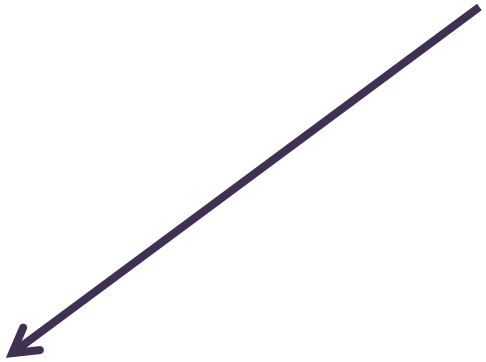
Εκτέλεση append/3

?- append(L1”,B,[d]).

```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

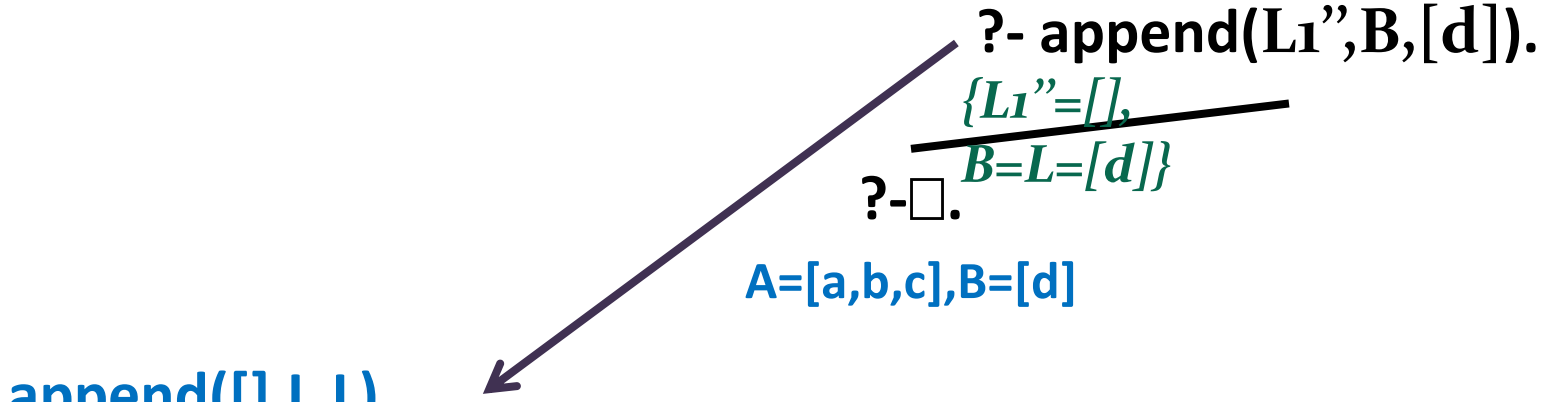
Εκτέλεση append/3

?- append(L1",B,[d]).



```
append([],L,L).

append([X|L1],L2,[X|L3]) :-
        append(L1,L2,L3).
```

Εκτέλεση append/3



Εκτέλεση append/3

A=[a|L1]=[a|[b|L1']]=
=[a|[b|[c|L1'']]]=
=[a|[b|[c|[]]]]=
=[a|[b,c]]=[a,b,c]

?- append(L1'',B,[d]).

{L1''=[],
B=L=[d]}

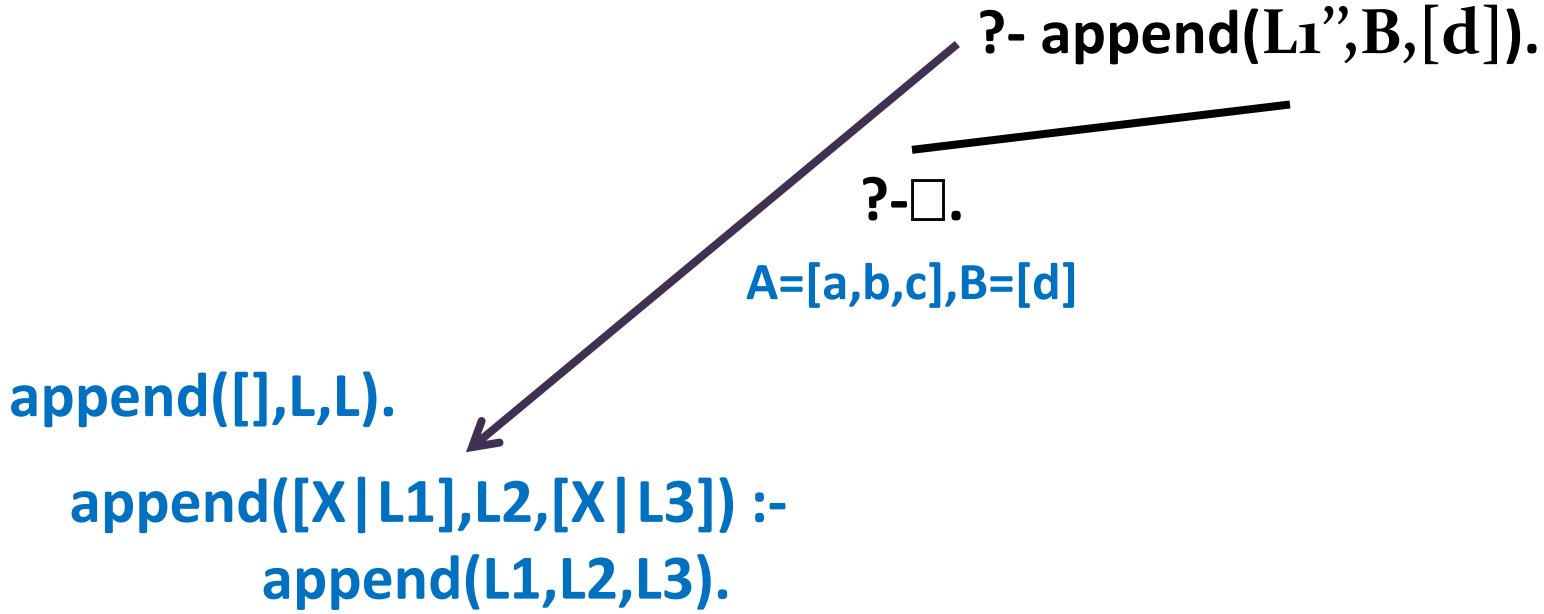
?-□.

A=[a,b,c],B=[d]

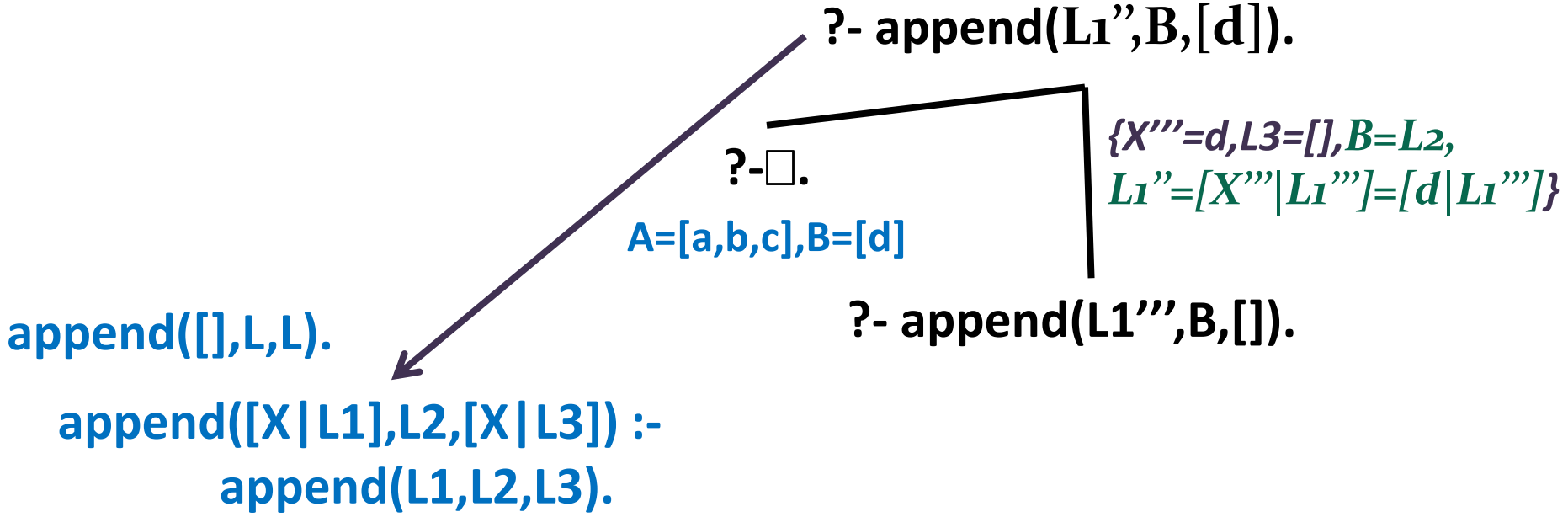
append([],L,L).

append([X|L1],L2,[X|L3]) :-
append(L1,L2,L3).

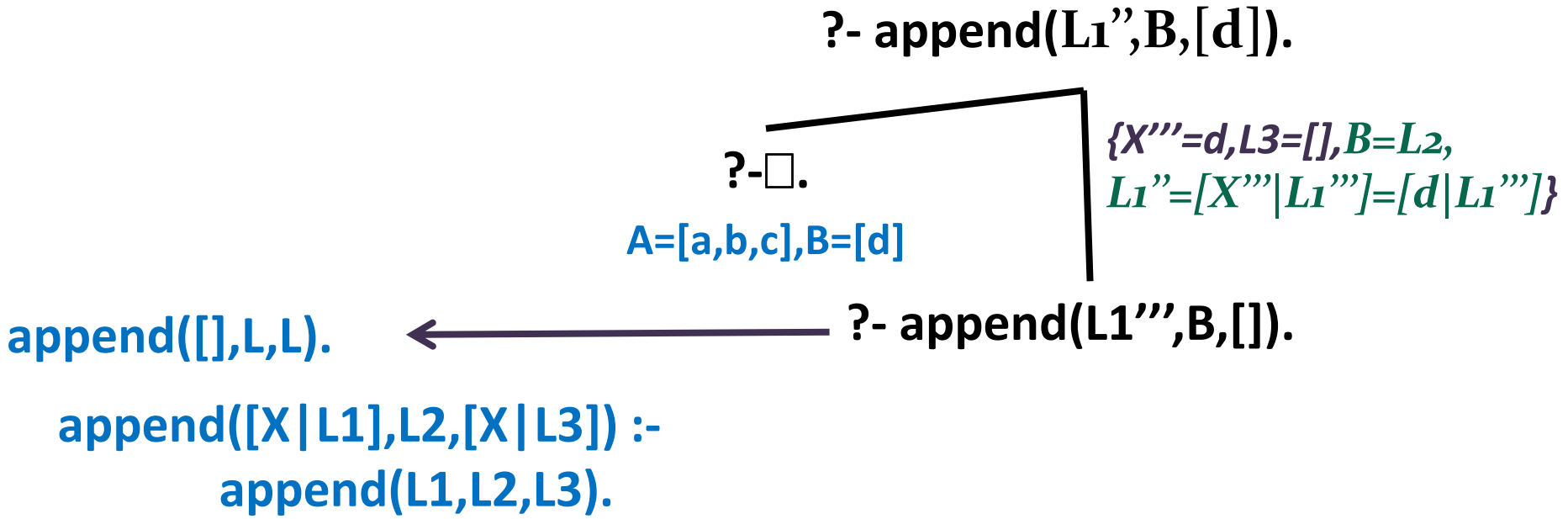
Εκτέλεση append/3



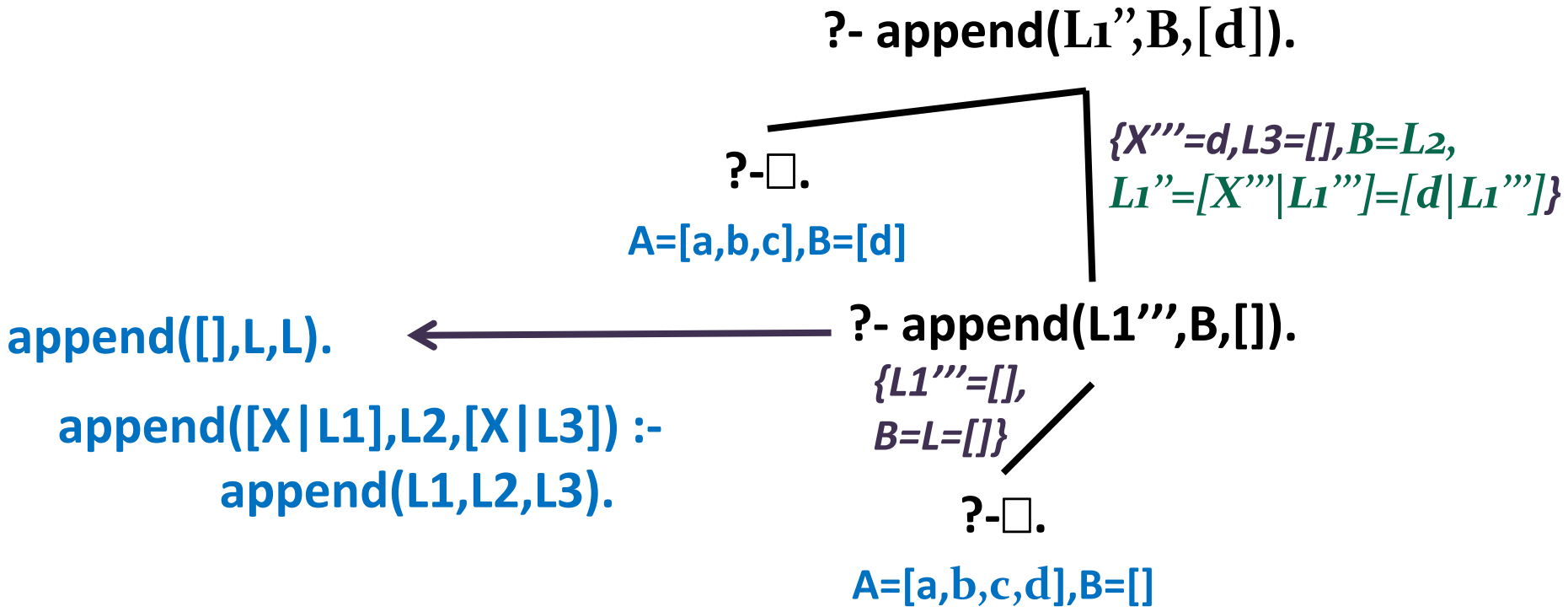
Εκτέλεση append/3



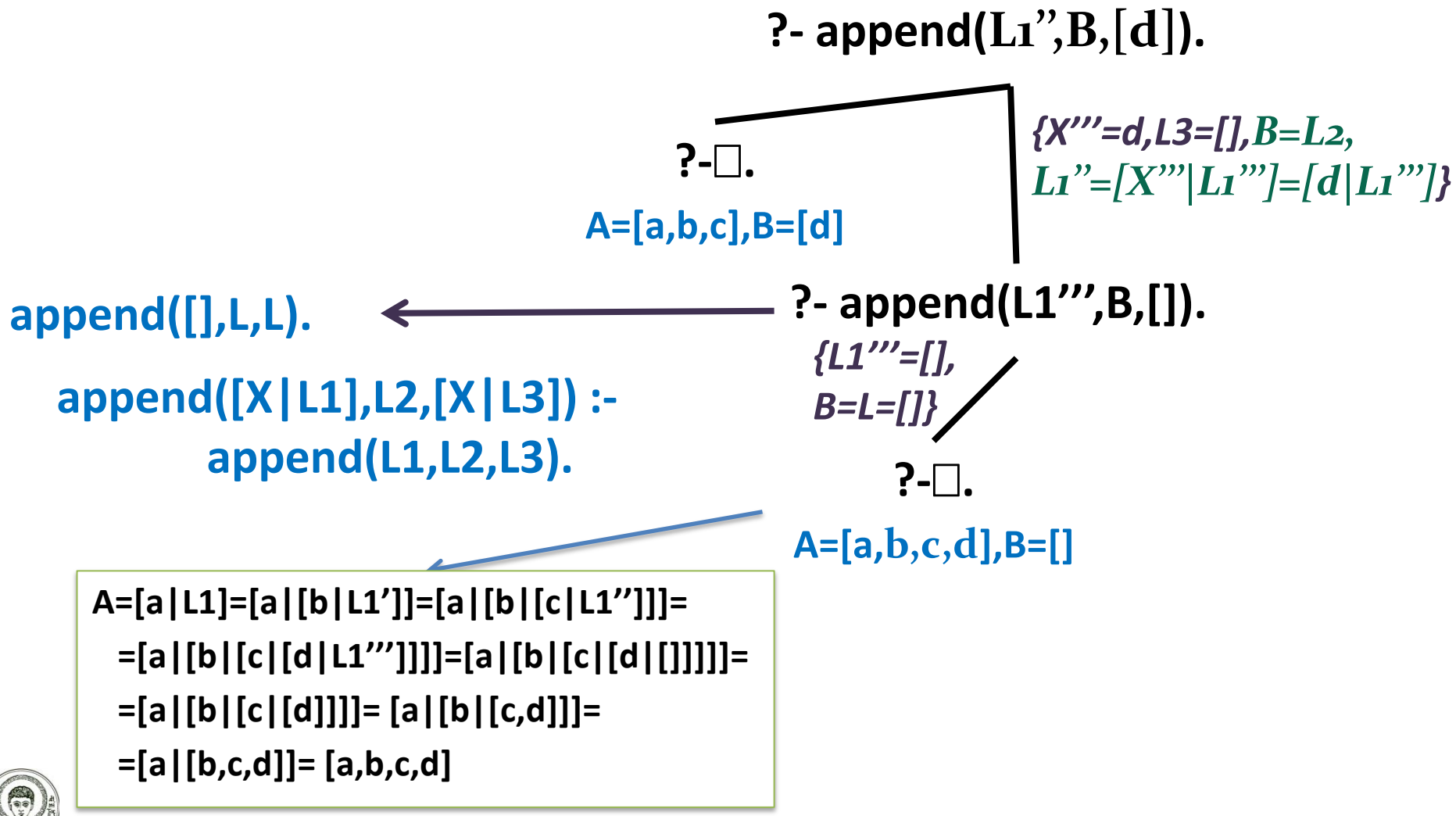
Εκτέλεση append/3



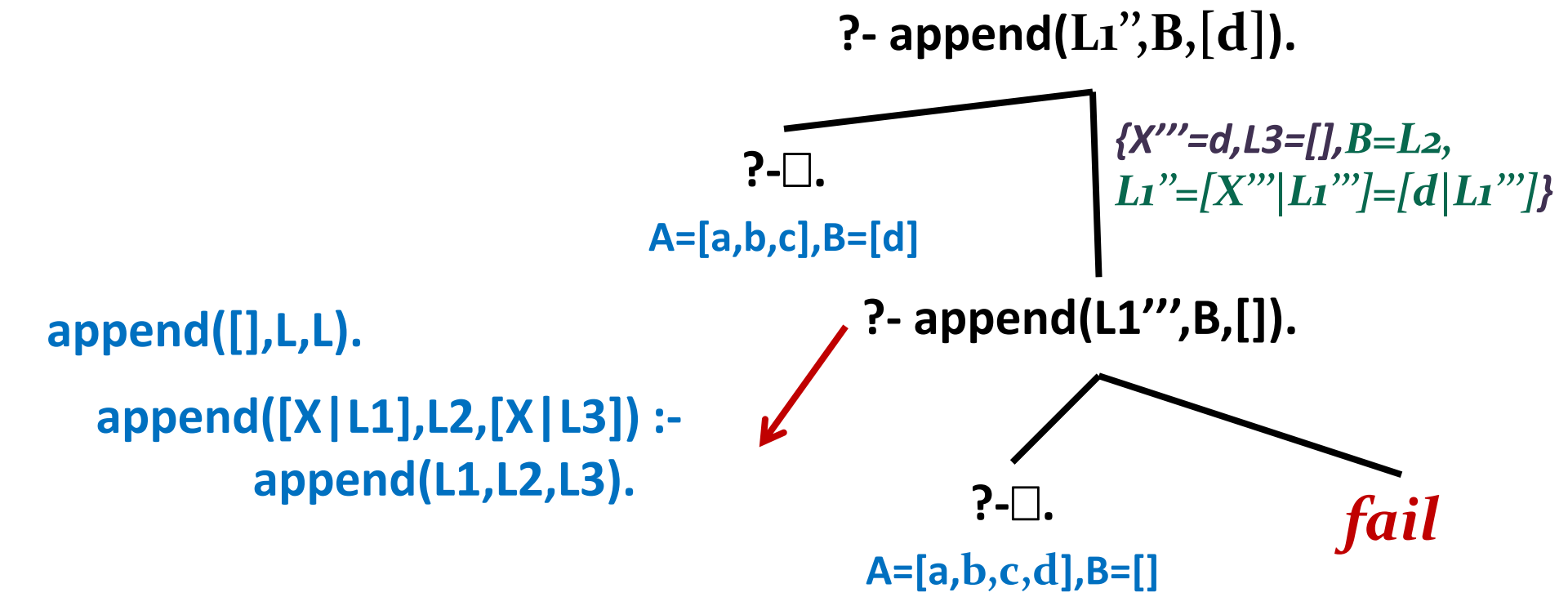
Εκτέλεση append/3



Εκτέλεση append/3



Εκτέλεση append/3



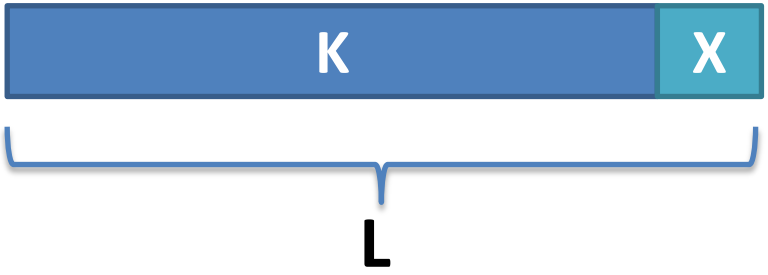
Άσκηση με append

- Να γραφεί κατηγόρημα, το οποίο να ορίζει τη σχέση **last(L,X),** με τη βοήθεια της **append/3**
- Λύση:

last(L,X) : - append(K,[X],L).

?- last([1,2,3],A).

A = 3



  - Παρατήρηση**:** Αν αντί για **[X]** βάλουμε **X** τότε, η απάντηση δε θα είναι το τελευταίο στοιχείο, αλλά οι τελευταίες υπολίστες.

Ασκήσεις με append

- Να γραφούν κατηγορήματα τα οποία χρησιμοποιώντας το **append**
  - α) Να σβήνουν τα τελευταία 3 στοιχεία από μια λίστα **L** παράγοντας μια άλλη λίστα **L1.**
  - β) Να σβήνουν τα πρώτα 3 στοιχεία από μια λίστα **L** παράγοντας μια άλλη λίστα **L1.**
  - γ) Να σβήνουν τα 3 πρώτα και τα 3 τελευταία στοιχεία μιας λίστας **L** παράγοντας μια λίστα **L1.**
- Λύση

```
del_first_3(L, L1) :- append([_,_,_], L1, L).

del_last_3(L, L1) :- append(L1, [_,_,_], L).

del_3_and_3(L, L1) :- append([_,_,_| L1], [_,_,_], L).

?- del_first_3( [1,2,3,4,5], L).

L = [4,5]
```

Διαδοχικά στοιχεία μιας λίστας

- Θέλουμε να ορισθεί μια σχέση **succ(S1,S2,L)** η οποία είναι αληθής όταν τα στοιχεία **S1** και **S2** είναι διαδοχικά στη λίστα **L.**

?- succ(2,3,[2,3,4]).

Yes

?- succ(2,3,[1,2,3,4]).

yes

- Παρατηρούμε :
  - Τα στοιχεία **S1** και **S2** είναι διαδοχικά σε μια λίστα αν το **S1** είναι το πρώτο και το **S2** το δεύτερο στοιχείο.
  - Αν δεν συμβαίνει αυτό, τότε αφαιρούμε ένα στοιχείο από την κεφαλή της λίστας, και ξαναελέγχουμε για τα υπόλοιπα.

Διαδοχικά στοιχεία μιας λίστας

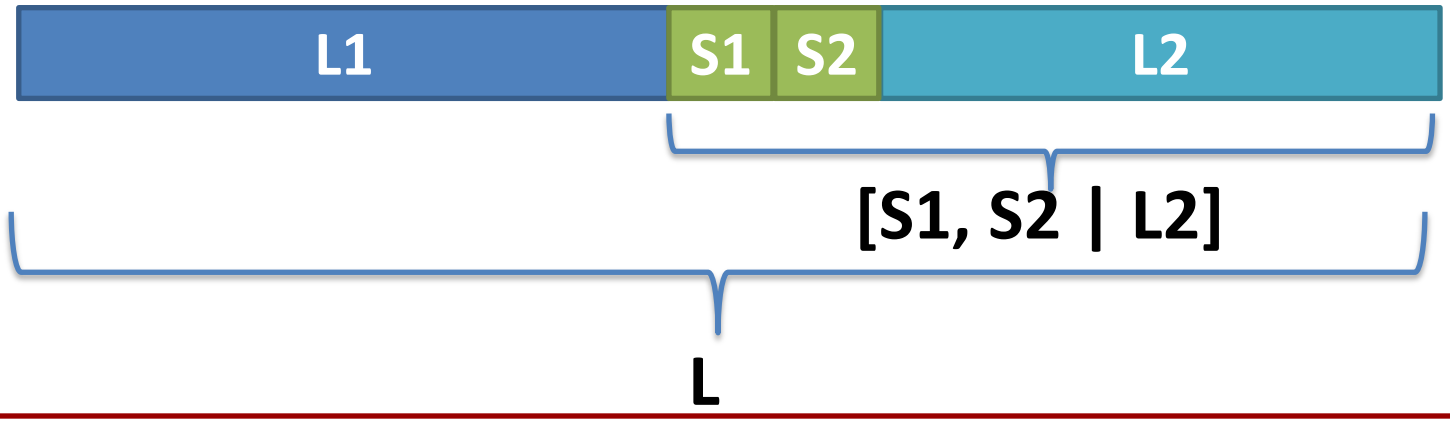
- Λύση 1:

succ(S1, S2, [S1, S2 | L]).

succ(S1, S2, [S | L]) :- succ(S1, S2, L).

- Λύση 2:

succ(S1, S2, L) :- append(L1, [S1, S2 | L2], L).



Ασκήσεις - Υπολίστα

- Να γραφεί πρόγραμμα το οποίο να ορίζει τη σχέση **sublist(S,L),** έτσι ώστε η λίστα **S** να είναι υπολίστα της **L.**

?- sublist([3,4,5,6], [1,2,3,4,5,6,7,8]).

yes

?- sublist([3,4,6], [1,2,3,4,5,6,7]).

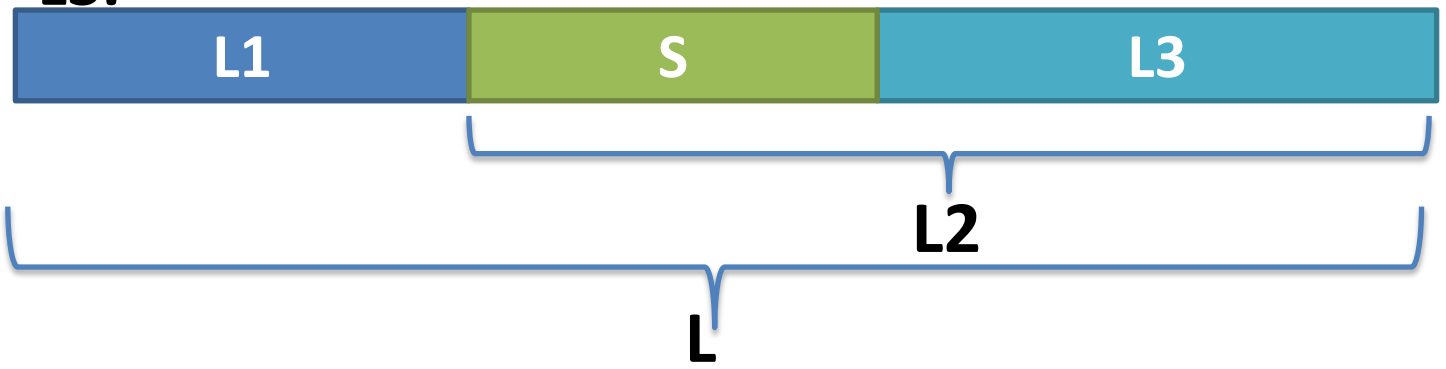
no

- Παρατήρηση: Η **S** είναι κάποιο «ατόφιο» κομμάτι της **L**, κάπου μέσα της.
  - Η **L** πρέπει να «σπάσει» σε 3 κομμάτια και το **S** να είναι το μεσαίο.

Υπολίστα

```
sublist(S,L) :-
    append(L1, L2, L),      % Χωρίζω την L σε L1 και L2.
    append(S, L3, L2).      % Χωρίζω την L2 σε S και L3.
```

- Δηλαδή η **S** είναι υπολίστα της **L** αν:
  - η **L** μπορεί να χωριστεί σε 2 υπολίστες **L1** και **L2**, και
  - η **L2** μπορεί να χωριστεί σε 2 υπολίστες **S** και κάποια **L3**.



Προσθήκη στοιχείων σε λίστα

- Το πρόβλημα της προσθήκης στοιχείου σε μία λίστα μπορεί να εμφανιστεί με δύο διαφορετικές μορφές:
  - την προσθήκη του στοιχείου στην αρχή της λίστας
  - την προσθήκη του στοιχείου σε οποιαδήποτε θέση της λίστας.
- Προσθήκη στοιχείου στην αρχή:

add(List,Element,[Element|List]).

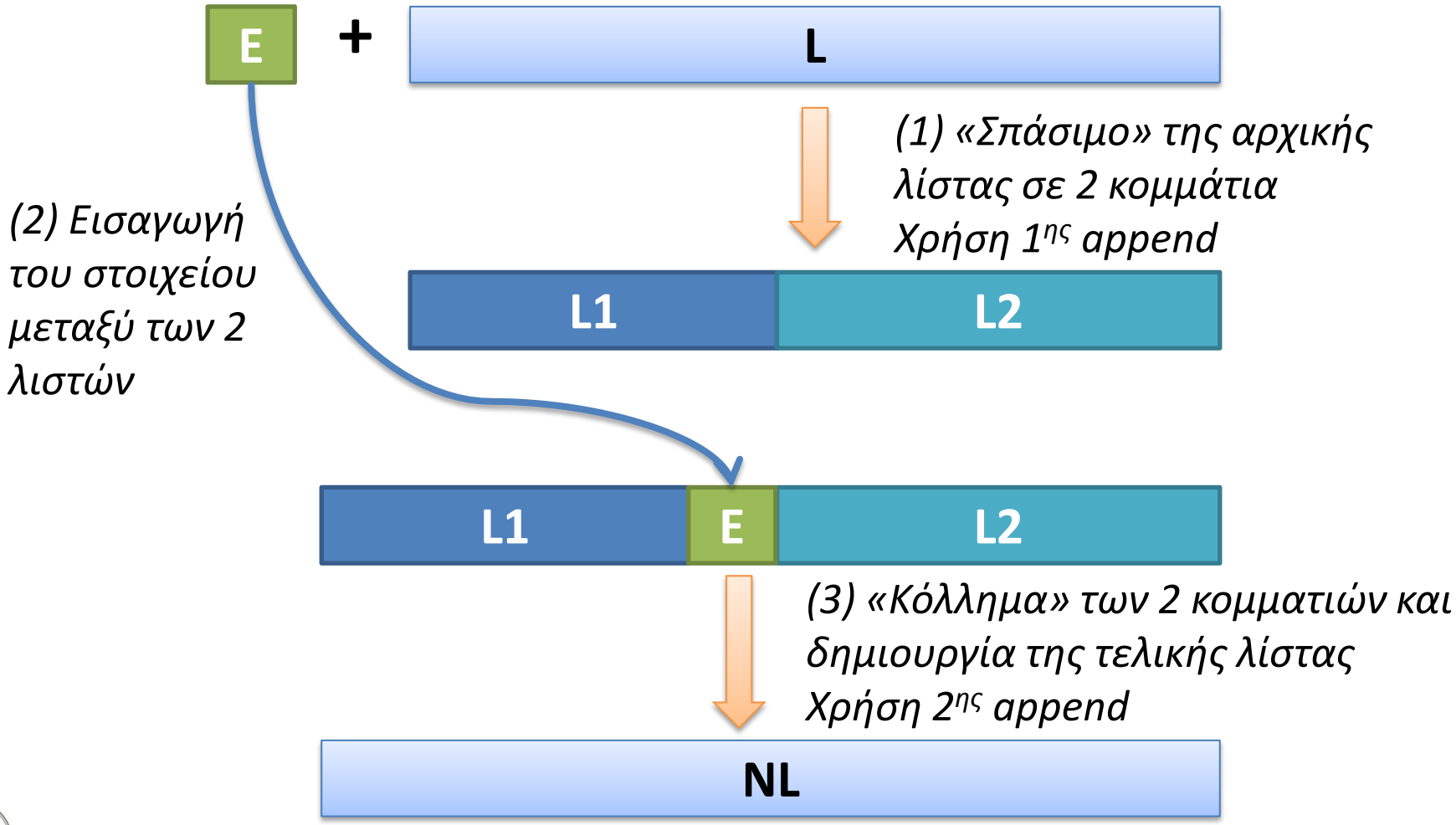
?-add([a,b,c,d],q,L).

L=[q,a,b,c,d]

Προσθήκη στοιχείων σε λίστα

- Αν, όμως, θέλουμε να προσθέσουμε το στοιχείο σε μια οποιαδήποτε (όχι συγκεκριμένη) θέση μέσα στη λίστα, τότε πρέπει να χρησιμοποιηθεί η **append** για
  - Να «σπάσει» τη λίστα σε δυο κομμάτια,
  - Να προσθέσει το νέο στοιχείο στην μέση των δύο λιστών (στην αρχή της δεύτερης),
  - Να «ξανα-κολλήσει» τα δύο κομμάτια σε ένα.

Αλγόριθμος


E
+
L
(1) «Σπάσιμο» της αρχικής λίστας σε 2 κομμάτια
Χρήση 1ης append
(2) Εισαγωγή του στοιχείου μεταξύ των 2 λιστών
L1
L2
L1
E
L2
(3) «Κόλλημα» των 2 κομματιών και δημιουργία της τελικής λίστας
Χρήση 2ης append
NL

Κώδικας και παραδείγματα

```
add(L,E,NL):-
        append(L1,L2,L),
        append(L1,[E|L2],NL).
```

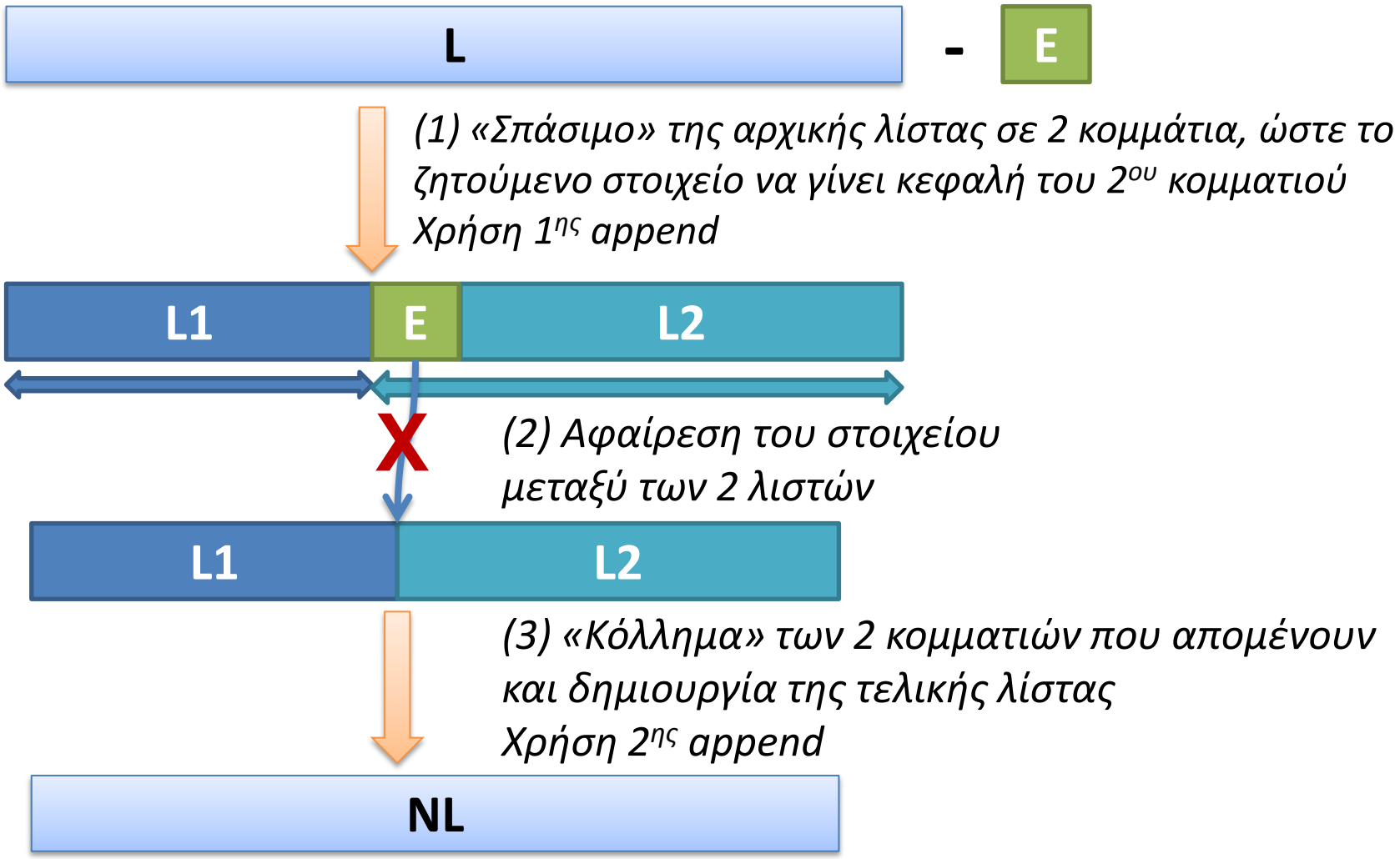
```
?-add([1,2,3],7,[1,2,7,3]).
yes
?-add([1,2,3],6,[1,2,3,4]).
no
```

```
?-add([a,b,c],1,L).
L=[1,a,b,c];
L=[a,1,b,c];
L=[a,b,1,c];
L=[a,b,c, 1] ;
no
```

Αφαίρεση στοιχείου από λίστα

- Με την αντιστροφή της προηγούμενης διαδικασίας μπορούμε να αφαιρέσουμε ένα στοιχείο από μία λίστα.
  - «Σπάμε» τη λίστα σε δυο κομμάτια, ώστε το ζητούμενο στοιχείο να «απομονωθεί», δηλαδή να γίνει κεφαλή του $2^{ου}$ κομματιού.
  - Αφαιρούμε το στοιχείο.
  - «Ξανα-κολλάμε» τα δύο κομμάτια που απομένουν σε ένα.
- Έχουμε ήδη δει τα κατηγορήματα **delone/3** και **delall/3**, τα οποία χρησιμοποιούν αναδρομή.
  - Εδώ θα δούμε πώς μπορούμε να χρησιμοποιήσουμε το **append/3** για την ίδια λειτουργία.

Αλγόριθμος


L
-
E
(1) «Σπάσιμο» της αρχικής λίστας σε 2 κομμάτια, ώστε το ζητούμενο στοιχείο να γίνει κεφαλή του 2ου κομματιού
Χρήση 1ης append
L1
E
L2
X
(2) Αφαίρεση του στοιχείου μεταξύ των 2 λιστών
L1
L2
(3) «Κόλλημα» των 2 κομματιών που απομένουν και δημιουργία της τελικής λίστας
Χρήση 2ης append
NL

Κώδικας και παραδείγματα

```
delete(E,L,NL):-
        append(L1,[E|L2],L),
        append(L1,L2,NL).
```

?-delete(a,[a,b,c,d],L).

L= [b, c, d]

?-delete(b,[a,b,c], L) .

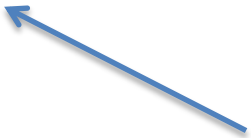
L=[a,c]

?-delete (1, [2,1,3,4,1,1], L) .

L=[2,3,4,1,1] ;

L=[2,1,3,4,1] ;

L=[2,1,3,4,1]



Διαγράφει το στοιχείο μόνο μια φορά, ακόμη και αν αυτό εμφανίζεται περισσότερες από μία φορές μέσα στη λίστα

Αναστροφή λιστών

- Η αναστροφή μιας λίστας, δηλαδή η δημιουργία μιας νέας λίστας που θα έχει τα στοιχεία της αρχικής λίστας με ανάστροφη φορά, μπορεί να επιτευχθεί με δύο διαφορετικούς τρόπους.

?-reverse([1,2,3,4], [4,3,1,2]).

no

?-reverse([1,2,3,4], [4,3,2,1]).

yes

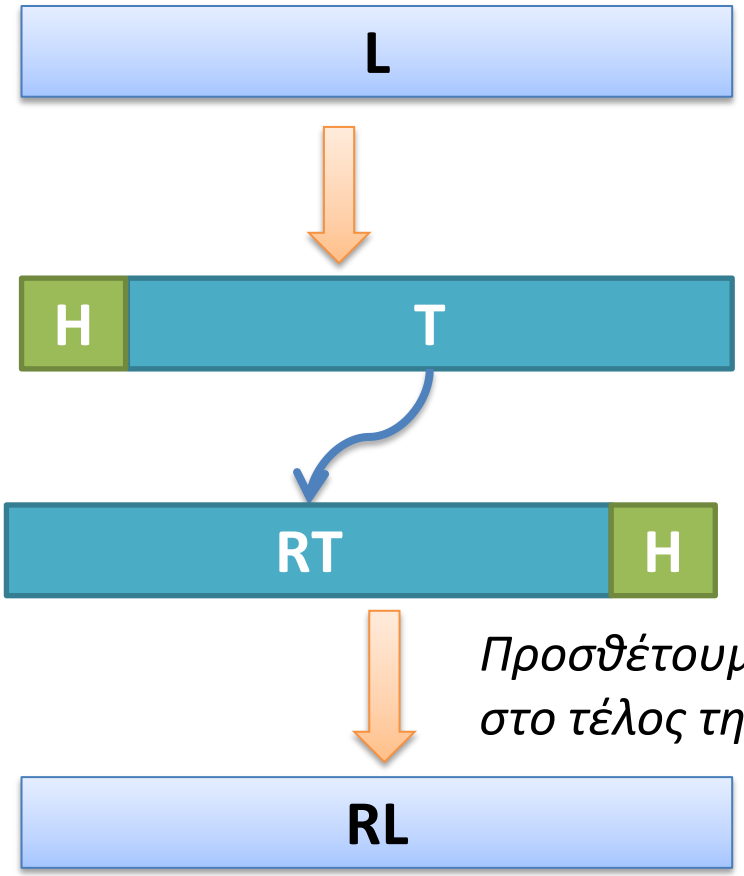
?-reverse([a,b,c],L).

L= [c, b, a]

Α' τρόπος αναστροφής

- Βήμα 1°: Εάν η λίστα είναι κενή, η ανεστραμμένη της είναι επίσης κενή.
- Βήμα 2°: Αλλιώς, χωρίζουμε την αρχική λίστα σε κεφαλή και ουρά.
- Βήμα 3°: Αντιστρέφουμε την ουρά.
  - Με αναδρομική κλήση της reverse
- Βήμα 4°: Προσθέτουμε την κεφαλή της αρχικής λίστας στο τέλος της αντεστραμμένης ουράς με την **append/3**
  - Γιατί δεν μπορεί να μπει στοιχείο στο τέλος μιας λίστας πιο «εύκολα»

Αλγόριθμος



Χωρίζουμε την αρχική λίστα σε κεφαλή και ουρά

Αντιστρέφουμε την ουρά

Προσθέτουμε την κεφαλή της αρχικής λίστας στο τέλος της αντεστραμμένης ουράς με την append

Κώδικας και Παρατηρήσεις

```
reverse( [ ], [ ]) .
reverse([H|T],RL):-
    reverse(T,RT),
    append(RT,[H],RL).
```

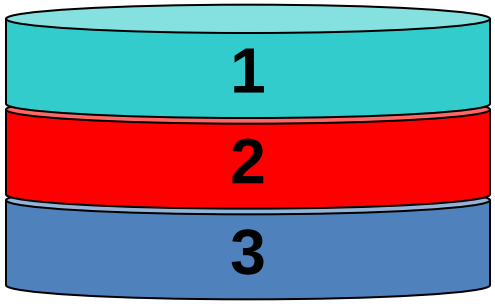
reverse *append*

$$(N+1)+\sum_{i=1}^{N} i =$$

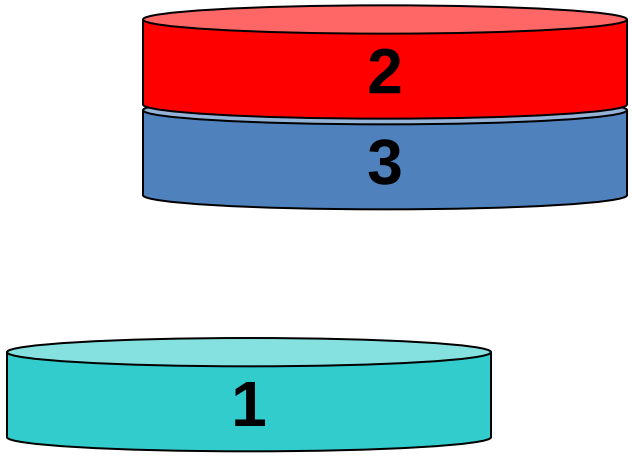
$$(N+1)+\frac{N*(N+1)}{2}$$

- Παρατήρηση: Ο παραπάνω ορισμός της **reverse** είναι πολύ αναποτελεσματικός και πολύ αργός καθώς γίνονται πολλές κλήσεις της **reverse** και της **append**.
  - Κάθε κλήση της **append** κάνει N αναδρομές, όπου N ο αριθμός στοιχείων της πρώτης λίστας
  - Η λύση αυτή λέγεται «αφελής» (naïve reverse)

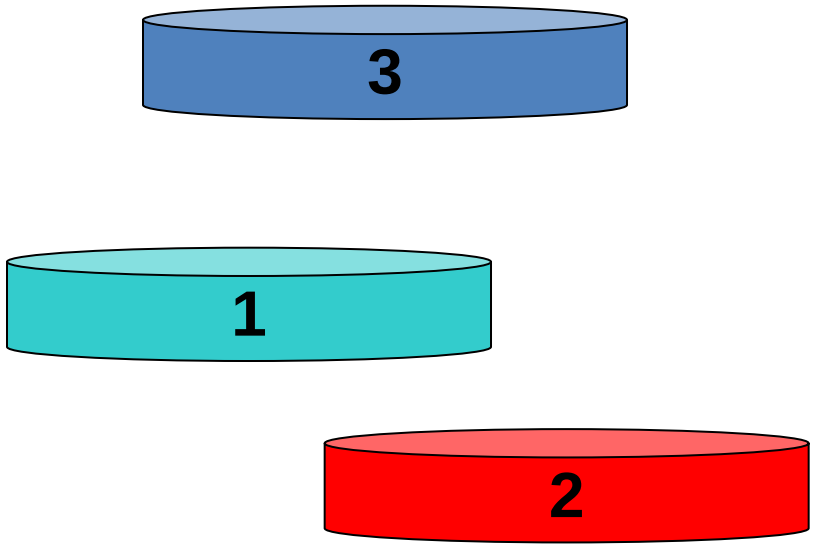
Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 1



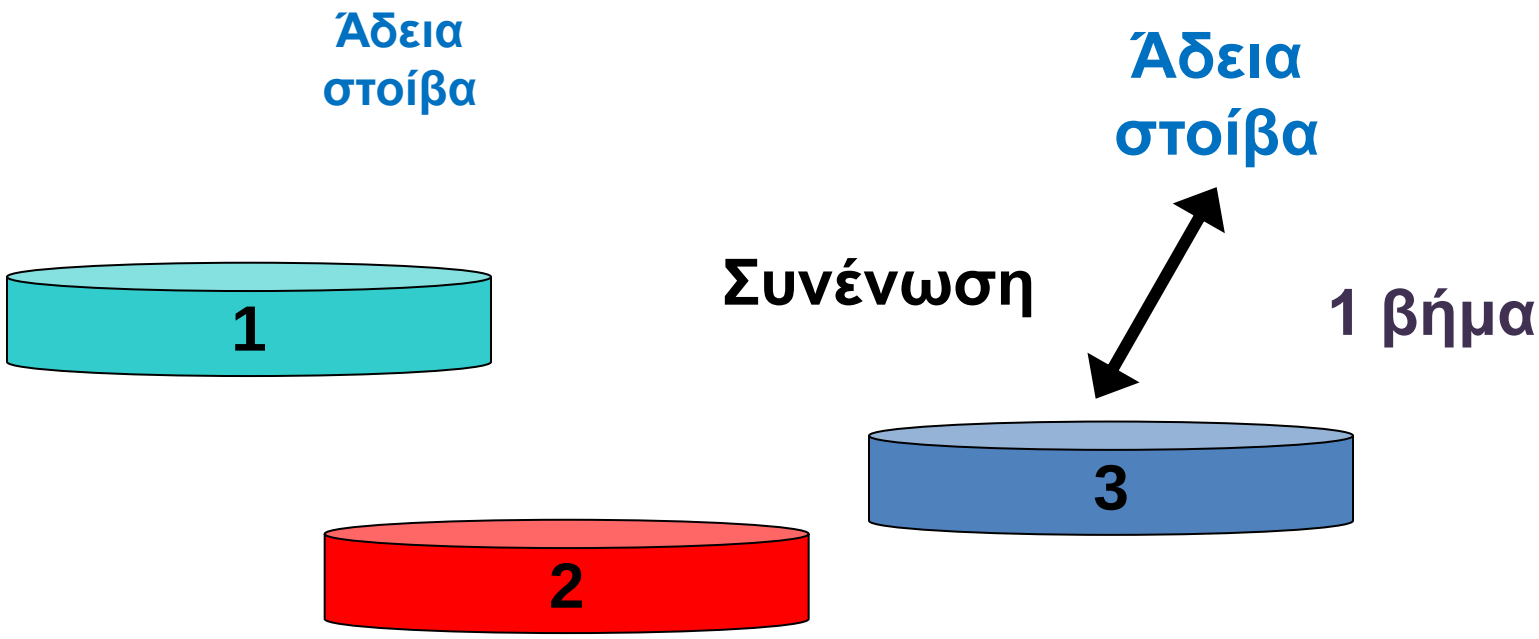
Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 2



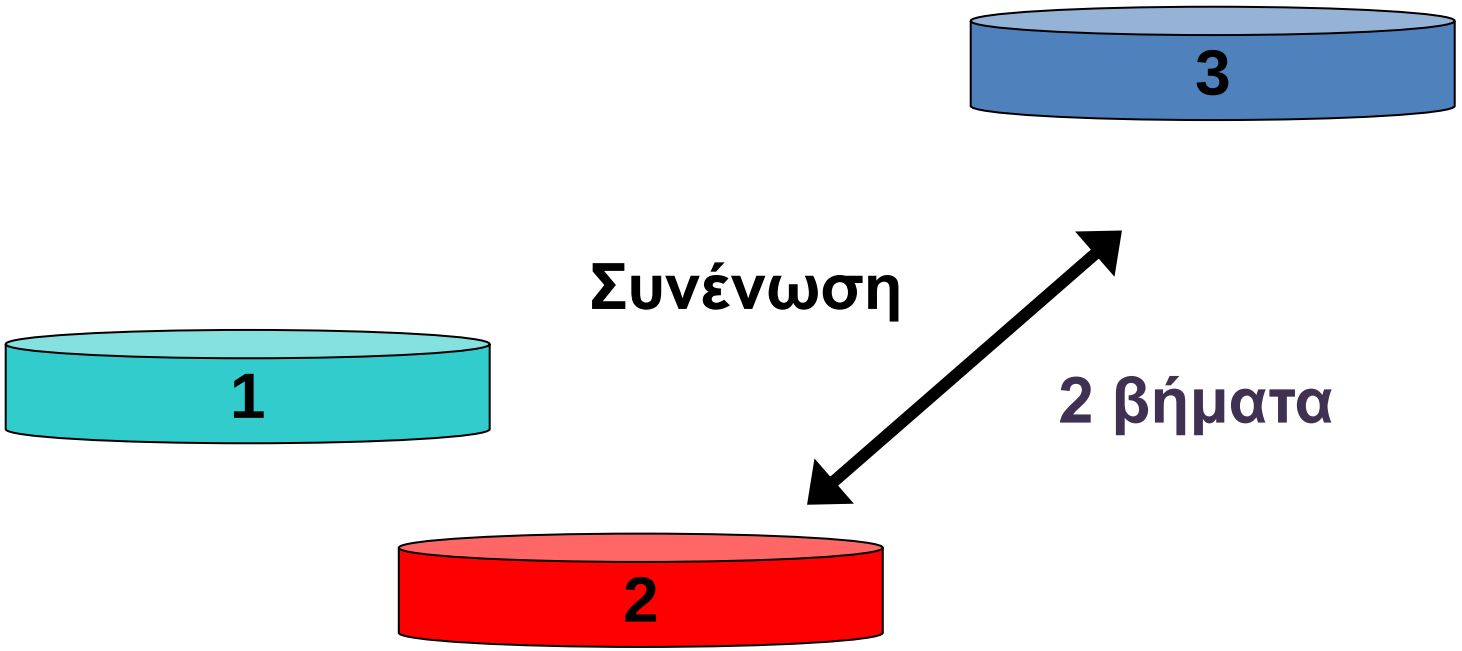
Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 3



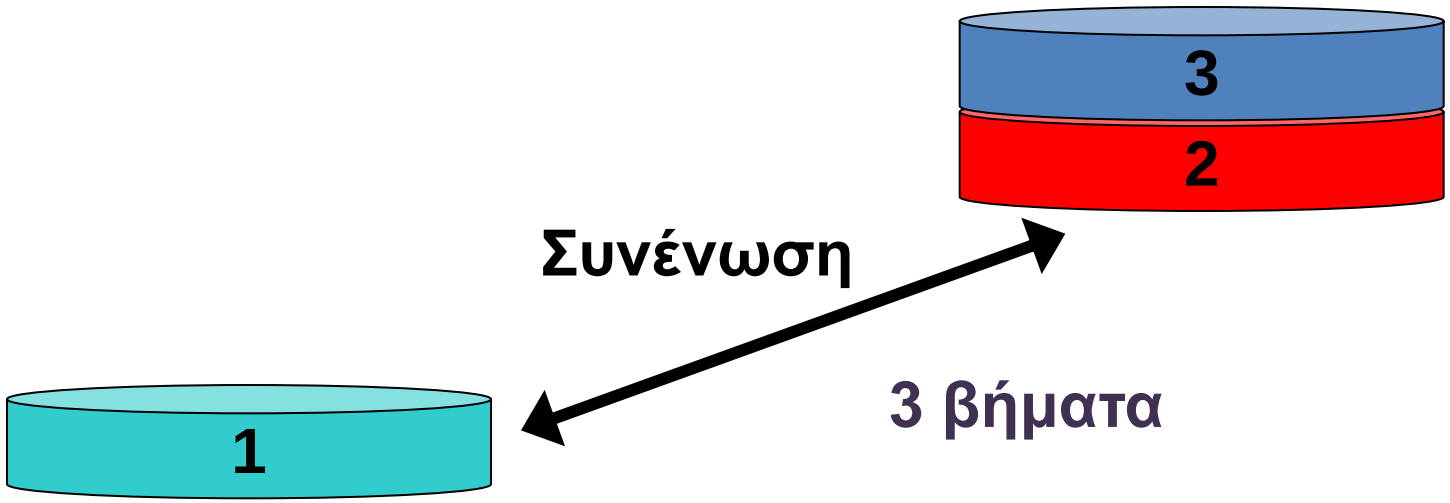
Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 4



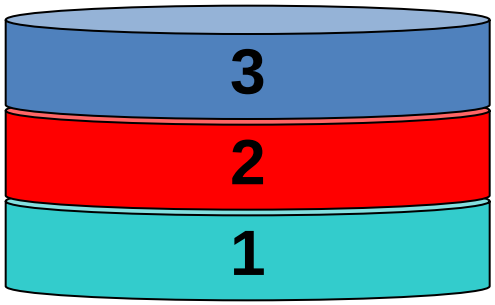
Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 5



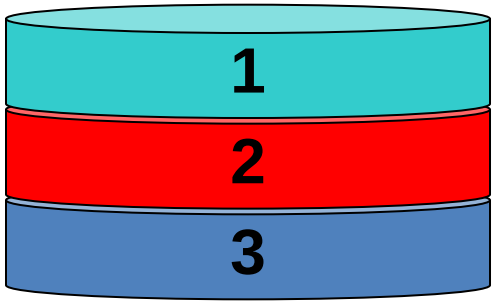
Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 7



Αντιστροφή Λιστών «Αφελής Λύση» - Βήμα 10



Αντιστροφή Λιστών
«Έξυπνη λύση» Βήμα 1

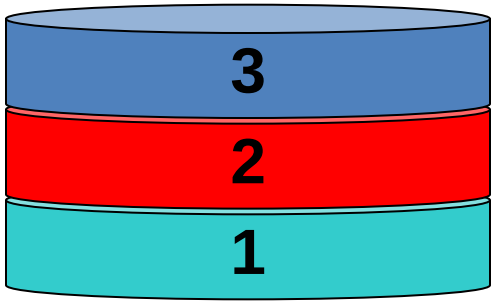


Αντιστροφή Λιστών
«Έξυπνη λύση» Βήμα 2

Αντιστροφή Λιστών
«Έξυπνη λύση» Βήμα 3

Αντιστροφή Λιστών «Έξυπνη λύση» Βήμα 4

Άδεια στοίβα



Β' τρόπος αναστροφής

- Στην περίπτωση αυτή, χρησιμοποιείται μια βοηθητική λίστα.

1. Η κεφαλή της αρχικής λίστας τοποθετείται στην κεφαλή της βοηθητικής λίστας.
2. Η διαδικασία επαναλαμβάνεται για την ουρά της αρχικής λίστας.
3. Όταν η αρχική λίστα αδειάσει, η βοηθητική λίστα περιέχει τα στοιχεία της αρχικής σε ανάστροφη σειρά.
   - Άρα, η βοηθητική λίστα είναι το τελικό αποτέλεσμα.

Κώδικας

Αρχική τιμή της βοηθητικής λίστας

L=[1, 2, 3]

```
reverse(L, RL):-
    rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
    rev(T, RL, [H|Temp]).
```

[H|T]	Temp
[1, 2, 3]	[]
[2, 3]	[1]
[3]	[2, 1]
[]	[3, 2, 1]

RL=[3, 2, 1]

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

A = [3,2,1]

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

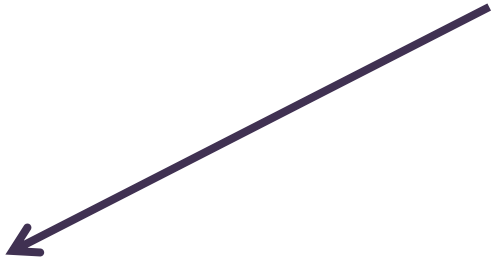
Εκτέλεση reverse

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

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).

rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

Εκτέλεση reverse

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



```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).

rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).

rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

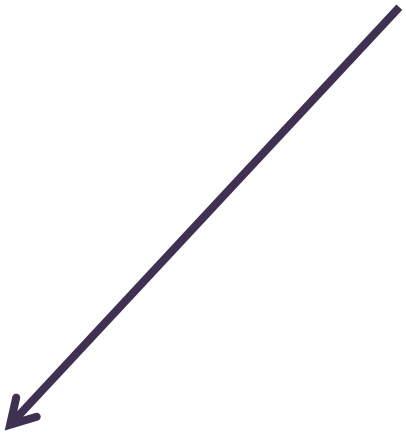
```
?- reverse([1,2,3],A).
```

{L=[1,2,3], A=RL}

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

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).

rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```



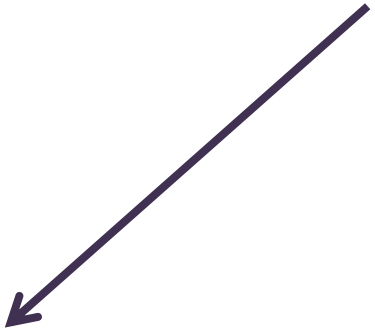
?- reverse([1,2,3],A).

{L=[1,2,3], A=RL}

?- rev([1,2,3],A,[]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).

rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```



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

{L=[1,2,3], A=RL}

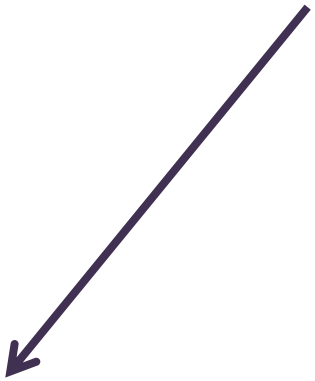
?- rev([1,2,3],A,[]).

{H=1,T=[2,3], Temp=[], A=RL}

?- rev([2,3],A,[1]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).

rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```



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

{L=[1,2,3], A=RL}

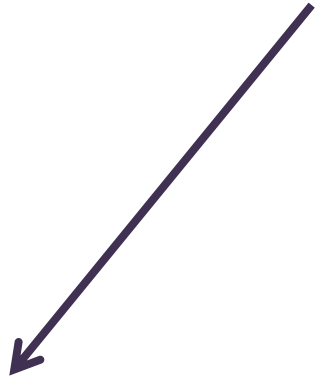
?- rev([1,2,3],A,[]).

{H=1,T=[2,3], Temp=[], A=RL}

?- rev([2,3],A,[1]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```



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

{L=[1,2,3], A=RL}

?- rev([1,2,3],A,[]).

{H=1,T=[2,3], Temp=[], A=RL}

?- rev([2,3],A,[1]).

{H=2,T=[3], Temp=[1], A=RL}

?- rev([3],A,[2,1]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

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

{L=[1,2,3], A=RL}

?- rev([1,2,3],A,[]).

{H=1,T=[2,3], Temp=[], A=RL}

?- rev([2,3],A,[1]).

{H=2,T=[3], Temp=[1], A=RL}

?- rev([3],A,[2,1]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

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

{L=[1,2,3], A=RL}

?- rev([1,2,3],A,[]).

{H=1,T=[2,3], Temp=[], A=RL}

?- rev([2,3],A,[1]).

{H=2,T=[3], Temp=[1], A=RL}

?- rev([3],A,[2,1]).

{H=3,T=[], A=RL, Temp=[2,1]}

?- rev([],A,[3,2,1]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```

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

{L=[1,2,3], A=RL}

?- rev([1,2,3],A,[]).

{H=1,T=[2,3], Temp=[], A=RL}

?- rev([2,3],A,[1]).

{H=2,T=[3], Temp=[1], A=RL}

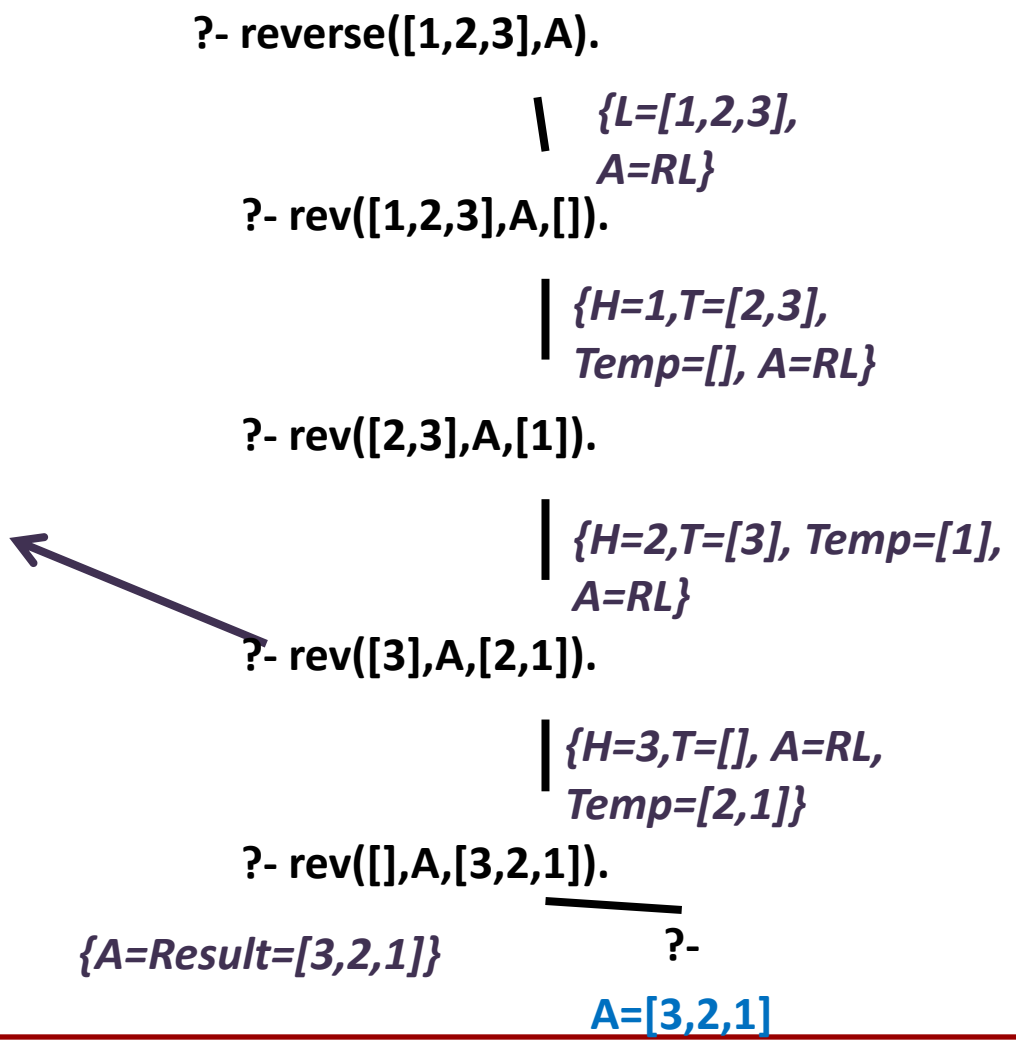
?- rev([3],A,[2,1]).

{H=3,T=[], A=RL, Temp=[2,1]}

?- rev([],A,[3,2,1]).

Εκτέλεση reverse

```
reverse(L, RL):-
   rev(L, RL, []).

rev([], Result, Result).
rev([H|T], RL, Temp) :-
   rev(T, RL, [H|Temp]).
```



Ασκήσεις - Συμμετρική Λίστα

- Να ορίσετε τη σχέση **palindrome(L),** η οποία αληθεύει όταν η λίστα L είναι συμμετρική ή παλινδρομική,
  - Δηλαδή είναι η ίδια αν τη διατρέξουμε ορθά ή αντίστροφα.
- Α' λύση:

```
palindrome(L) :- reverse(L,L).
```

- Β' λύση:

```
palindrome([ ]).
palindrome([_]).
palindrome(L) :-
    append([X|Y],[X],L),
    palindrome(Y).
```

Τερματικές συνθήκες

```
?- palindrome([a,b,c,b,a]).
yes
?- palindrome([a,b,b,a]).
yes
?- palindrome([a,b,c,w,a]).
no
```

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Τέλος ενότητας

Επεξεργασία: Εμμανουήλ Ρήγας

Θεσσαλονίκη, Εαρινό Εξάμηνο 2013-2014

Σημειώματα

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.