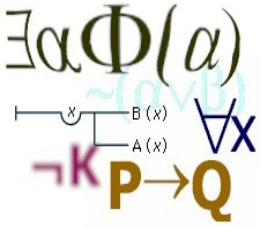
Πανεπιστήμιο Μακεδονίας
Τμ. Εφαρμοσμένης Πληροφορικής

Αποκοπή και Κατηγορήματα Ανώτερης Τάξης
(Cut and Higher Order Predicates)



Ηλίας Σακελλαρίου

Δομή

- Άρνηση ως Αποτυχία
- Έλεγχος Οπισθοδρόμησης
  - Cut (αποκοπή)
  - Ενσωματωμένα κατηγορήματα Ελέγχου
- Κατηγορήματα Ανώτερης Τάξης
  - Μεταβλητή Κλήση-Variable call (*call/1)*
  - Άρνηση-Negation (υλοποίηση)
  - Εκτέλεση υπο συνθήκη-Conditional
  - *Συλλογή Λύσεων-Solution Gathering*
  - *Δημιουργία όρων-Term Creation*
  - *Διαχείριση Βάσης-Database Manipulation*

Άρνηση

- Απόδειξη των αρνητικών προτάσεων.
  - "Υπόθεση του κλειστού κόσμου" (closed world assumption)

"ότι δεν είναι δυνατό να αποδειχθεί αληθές, θεωρείται ψευδές"

- Για παράδειγμα:

male(petros).

male(ilias).

male(demos).

male(nikos).

?-male(alex)
no
?-not(male(alex))
yes

Παρατηρήσεις

- Οποιαδηπότε κλήση:

?- not(member(15,[10,20,30])).

yes.

- Προσοχή στην ύπαρξη μεταβλητών:

?- not(X = 10)

no

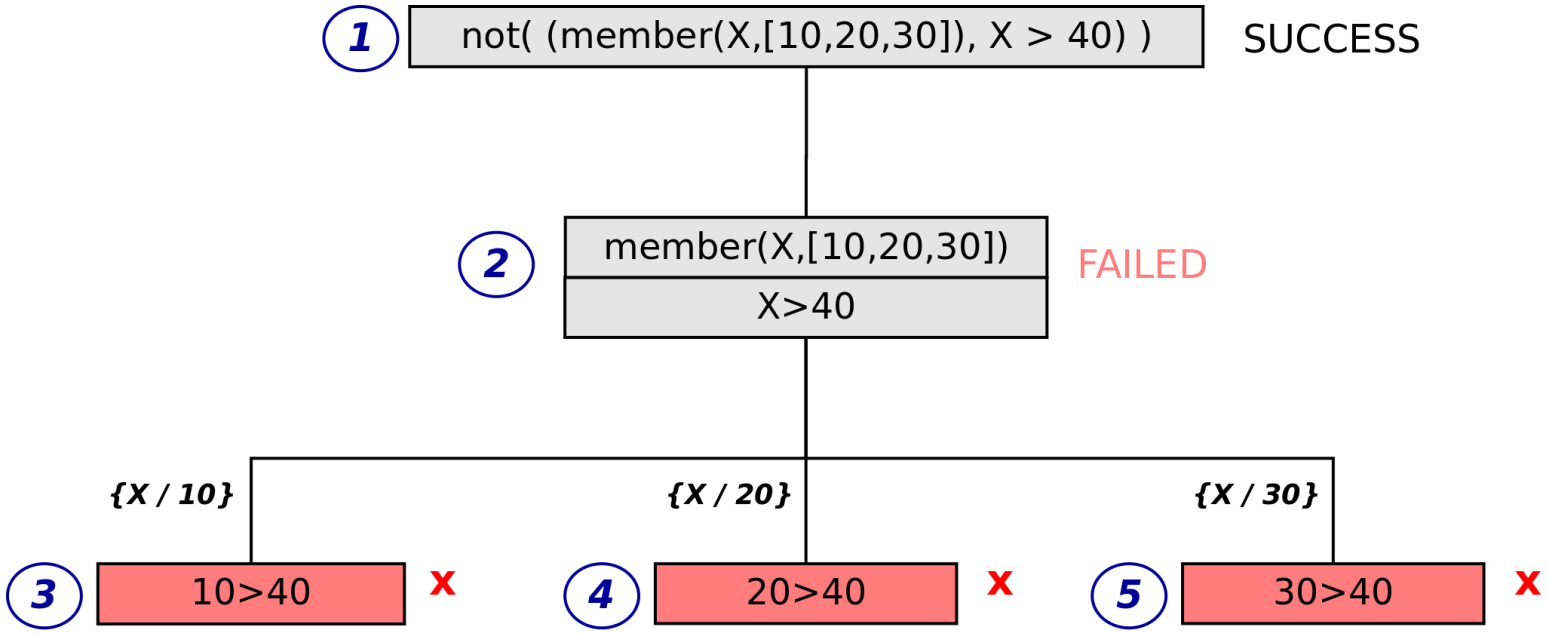
?- not(male(X))

no

Σύνθετοι Στόχοι

?- not((member(X,[10,20,30]), X > 40)).

yes



Έλεγχος Οπισθοδρόμησης

Έλεγχος Οπισθοδρόμησης

- **Backtracking**
  - Παράγει όλες τις δυνατές λύσεις για μια ερώτηση
  - Παράδειγμα:
    - X=a,member(X,[1,2,a,3,4,a,5,6,a]).
  - Πετυχαίνει (ίσως) περισσότερες φορές από ότι απαιτείται.
- Κατηγόρημα ! (cut) ελέγχει την οπισθοδρόμηση

The Cut (!) Predicate

- Αναπαρίσταται ως θαυμαστικό “**!**”.
  - Arity 0.
- Τοποθετείται σαν κανονικός υποστόχος (subgoal) στο σώμα μιας πρότασης.
- Δεν έχει δηλωτική έννοια!
- Διαφοροποιεί τη συμπεριφορά (εκτέλεση) του κατηγορήματος.
- Αυξάνει την εκφραστική ικανότητα της Prolog.

Τα αποτελέσματα του Cut

- Τα αποτελέσματα του cut είναι τα ακόλουθα:
  - Επιτυγχάνει πάντα.
  - Περιορίζει (αποκλείει) την οπισθοδρόμηση πριν από το σημείο που δηλώνεται μέσα στον ορισμό (κατηγόρημα) που εμφανίζεται.
- Με απλά λόγια:
  - Όταν το cut επιτυγχάνει τότε οι όποιες εναλλακτικές απαντήσεις οι οποίες προέρχονται από το κατηγόρημα που περιέχει το cut αποκλείονται.

Απλό Παράδειγμα

- Πρόγραμμα:

  p(1). *p(2).*

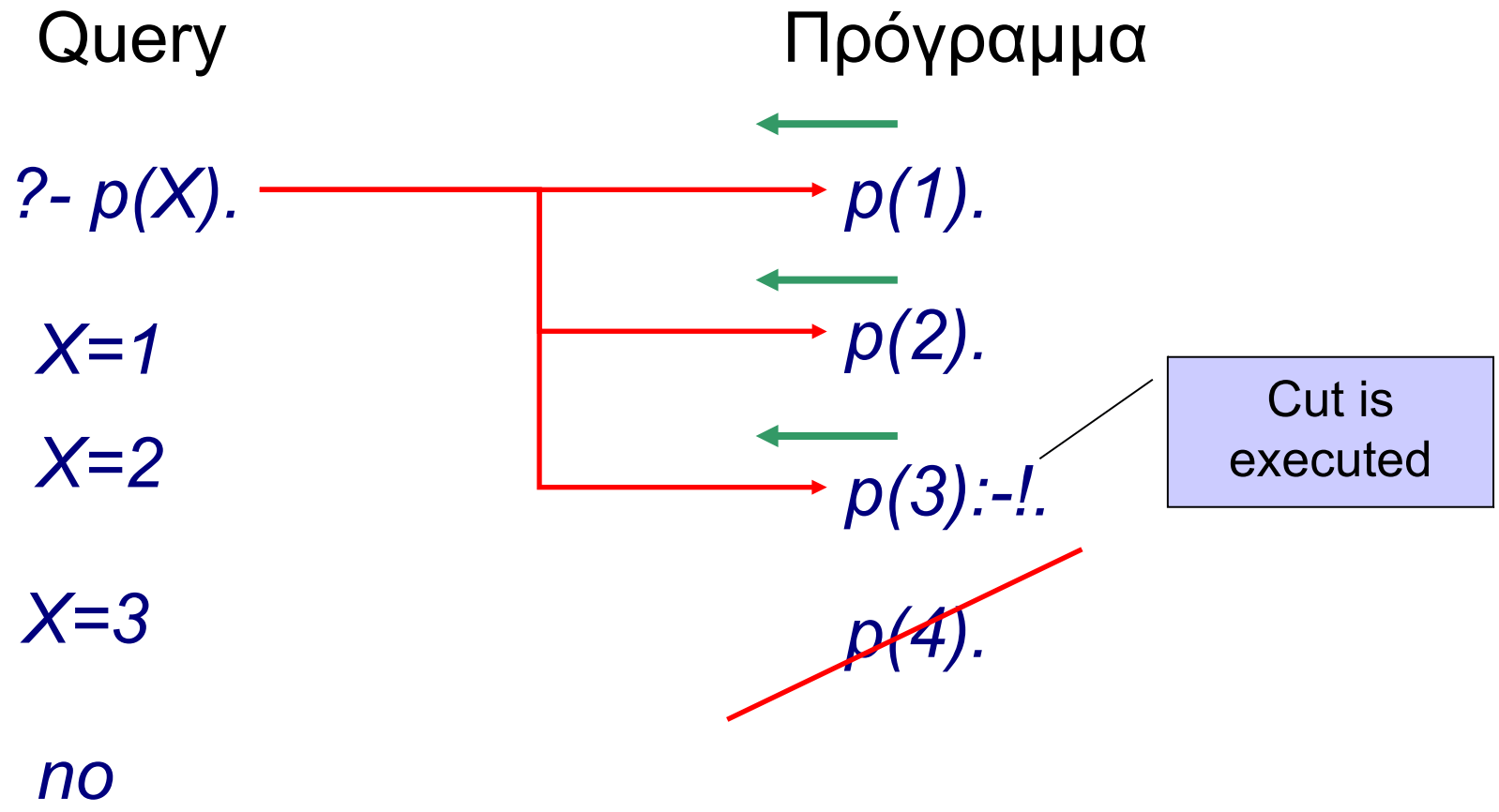
  p(3):-!. *p(4).*

- Ερωτήσεις

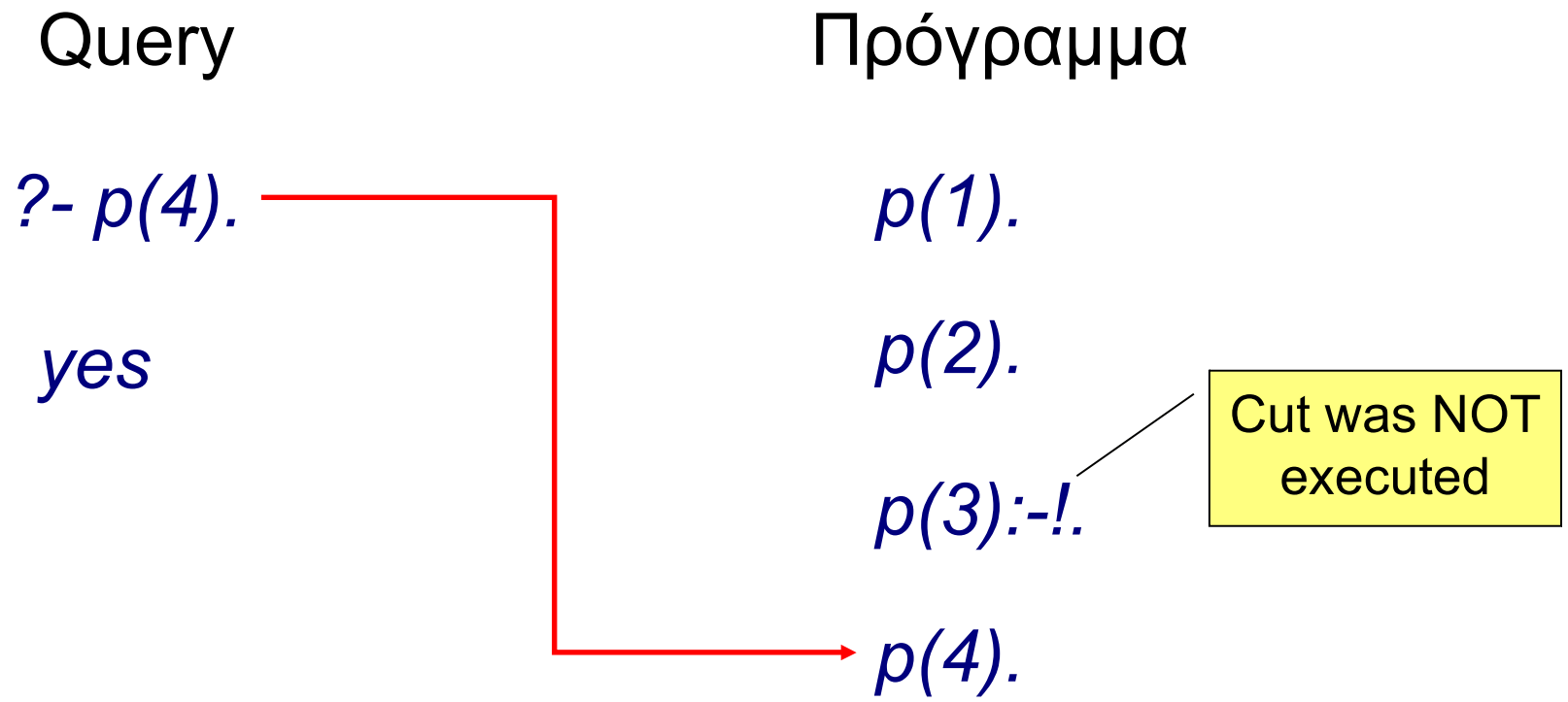
  ?-p(X).

  ?-p(4).

Prolog Tree



Prolog Tree



Ένα λίγο πιο πολύπλοκο παράδειγμα (Α)

a(1).
a(2).
b(X):-a(X).
b(3).

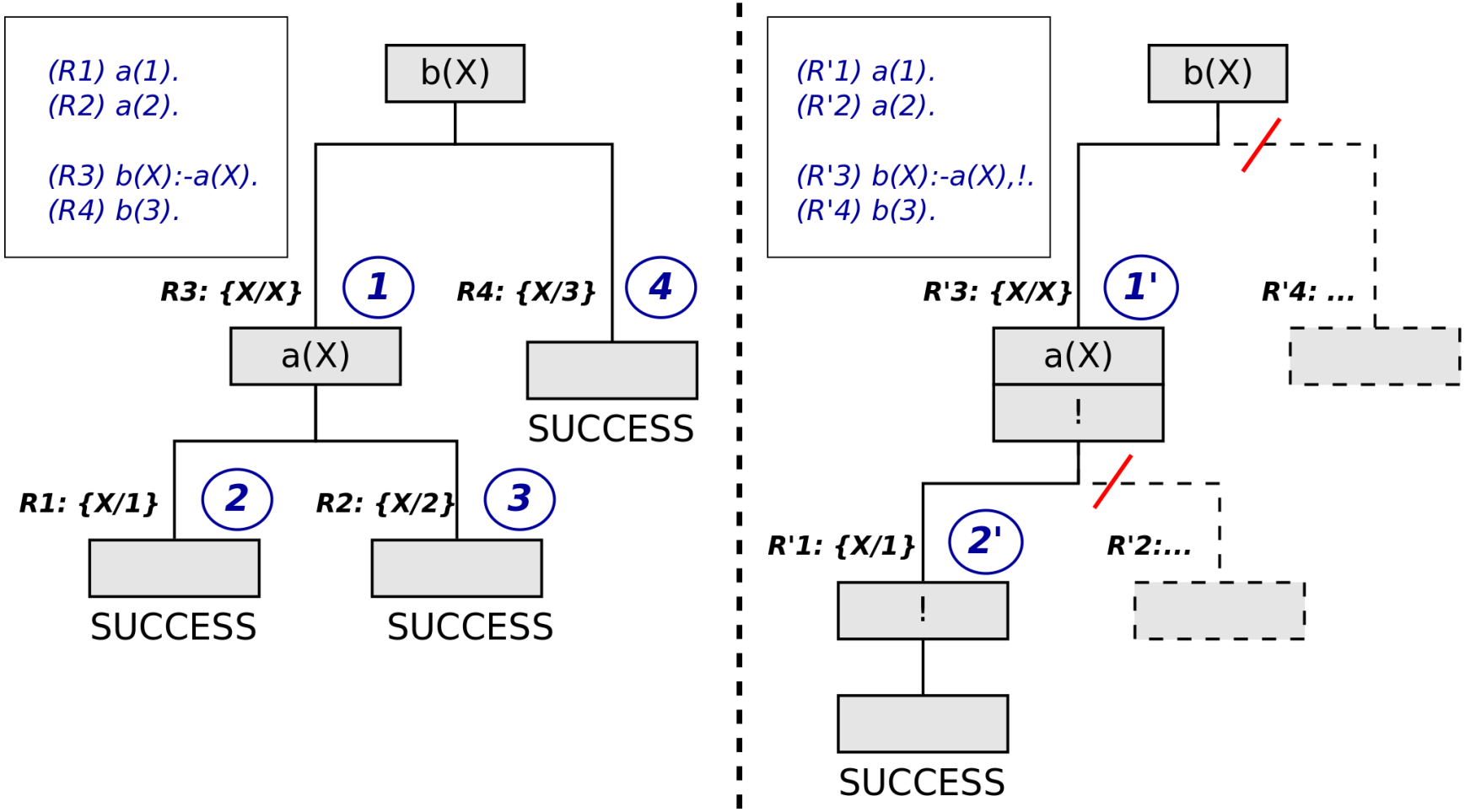
- Ερώτηση b(X) :

?- b(X).
X = 1 ;
X = 2 ;
X = 3.

a(1).
a(2).
b(X):-a(X),!.
b(3).

?- b(X).
X = 1;

Δένδρο Αναζήτησης Α



Ένα λίγο πιο πολύπλοκο παράδειγμα (Β)

a(1).
a(2).
b(X):-a(X).
b(3).

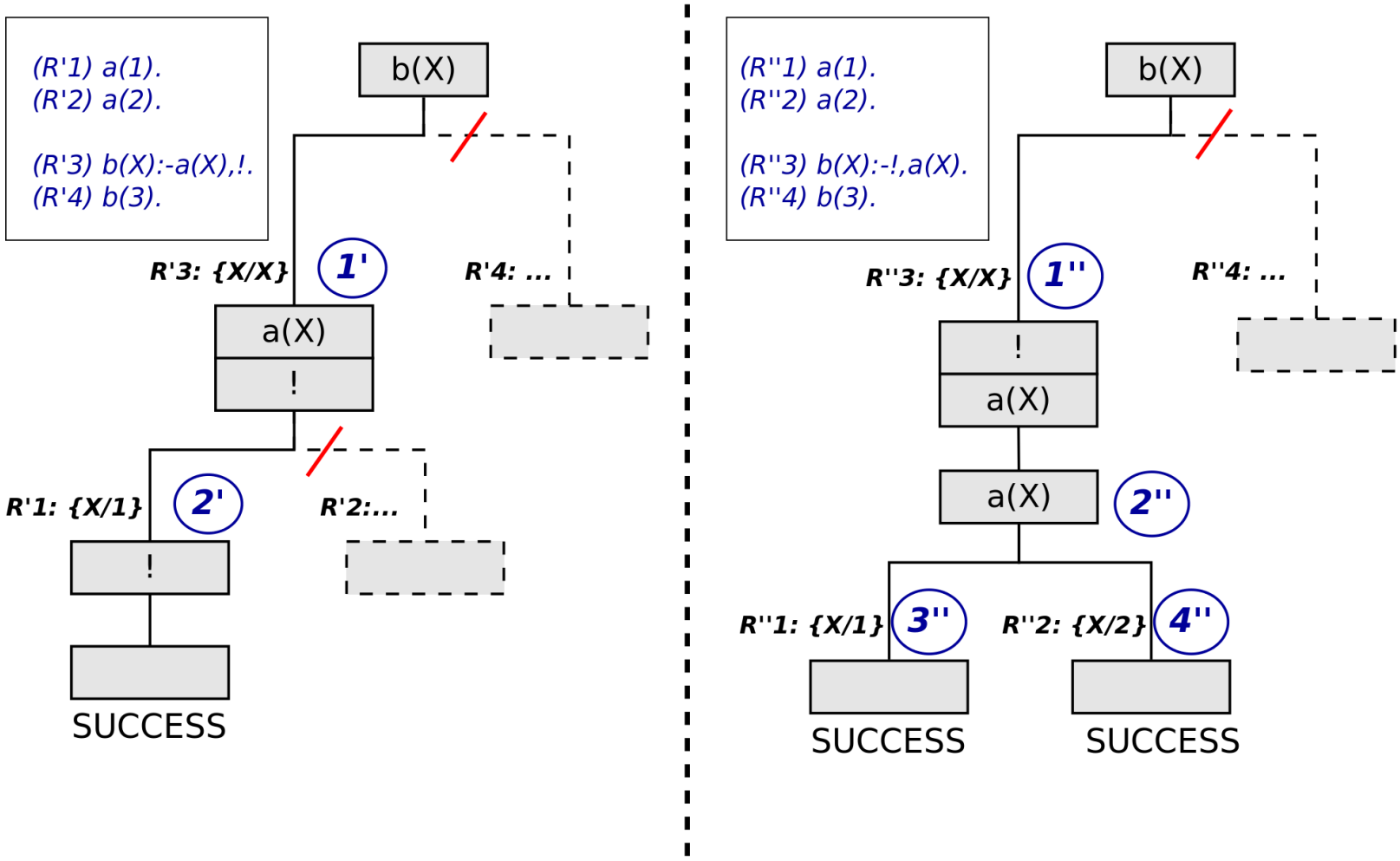
- Ερώτηση b(X) :

?- b(X).
X = 1 ;
X = 2 ;
X = 3.

a(1).
a(2).
b(X):-!,a(X).
b(3).

?- b(X).
X = 1;
X = 2;
no

Δένδρο Αναζήτησης Β



… και η τελευταία πρόταση?

- Είναι άχρηστη;
  - *?-b(3)*
  - *yes*

```
a(1).
a(2).
b(X):-a(X),!.
b(3).
```

Ένα περισσότερο σύνθετο παράδειγμα

```
q(X,Y,Z):-a(X),b(Y),!,c(Z).
q(X,Y,Z):-!,d(X,Y,Z),e(Y).
q(10,11,12).

a(1).
a(2).

b(first).
b(second).

c(4).
c(5).

d(6,7,8).
d(6,8,8).

e(8).
e(7).
```

?- q(X,Y,Z).

?- q(6,Y,Z).

?-q(10,X,Y).

Χρήσεις της Αποκοπής

- Απόδοση
  - Αποκλεισμός της οπισθοδρόμησης όταν δεν είναι απαραίτητη.
  - Παραδείγματα:
    - member_chk/2
    - delete_one/3
- ΑΠΑΙΤΕΙΤΑΙ ιδιαίτερη προσοχή όταν γράφονται ορισμοί με cut!

Κριτική

- Δεν έχει δηλωτική σημασία παρά μόνο διαδικαστική.
  - Extra-logical κατηγόρημα της Prolog.
- Αλλάζει τη σημασία του κατηγορήματος.
- Άρα η Prolog ΔΕΝ είναι “καθαρή” γλώσσα Λογικού Προγραμματισμού.

Τομή δύο Λιστών

- Να ορίσετε ένα κατηγόρημα intesect/3 το οποίο βρίσκει την τομή δύο λιστών.
- Με χρήση **not/1**.

?- intersect([1, 2], [1, 2, 3, 4], Res).
Res = [1, 2]
Yes

?- intersect([1, 2, 5, 6], [1, 2, 3, 4], Res).
Res = [1, 2]
Yes

Τομή δύο Λιστών

- Λύση:

```
intersect([],_,[]).
intersect([X|Rest],List,[X|IntList]):-
    member(X,List), intersect(Rest,List,IntList).
intersect([X|Rest],List,IntList):-
    not(member(X,List)),  intersect(Rest,List,IntList).
```

Τομή δύο Λιστών ver 2

- Να ορίσετε ένα κατηγόρημα intesect/3 το οποίο βρίσκει την τομή δύο λιστών.
- Με χρήση cut.

```
intersect([],_,[]).
intersect([X|Rest],List,[X|IntList]):-
    member(X,List), !, intersect(Rest,List,IntList).
intersect([X|Rest],List,IntList):-
    intersect(Rest,List,IntList).
```

Ενσωματωμένα κατηγορήματα ελέγχου

Ενσωματωμένα Κατηγορήματα Ελέγχου (Control built-in Predicates)

- *true/0*
  - Πάντα επιτυγχάνει (μια φορά)
- *fail/0*
  - Πάντα αποτυγχάνει.
- *repeat/0*
  - Επιτυγχάνει πολλές φορές κατά την οπισθοδρόμηση.

Παράδειγμα – Εντολή write/1

- Να ορίσετε ένα κατηγόρημα το οποίο τυπώνει στην κονσόλα όλα τα στοιχεία μιας λίστας.

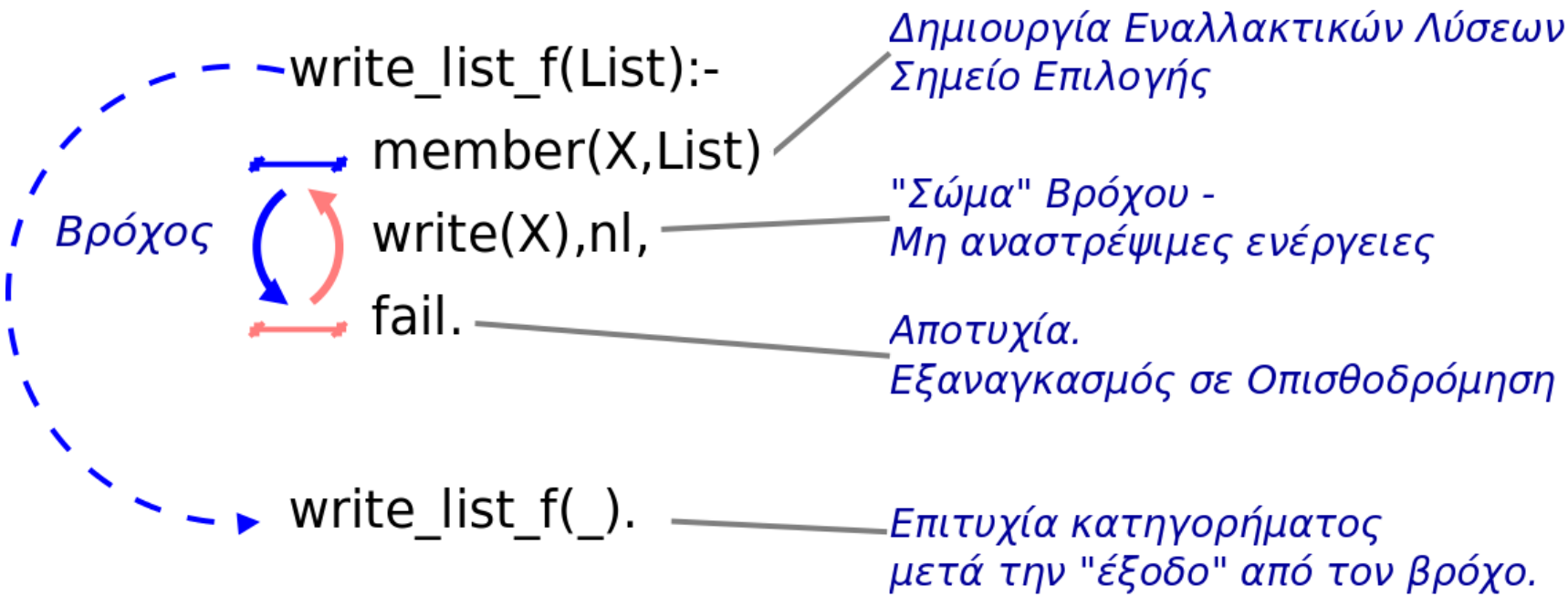
```
write_nr([]).
write_nr([X|List]):-
        write(X),
        write_nr(List).
```

Παράδειγμα

- Να ορίσετε ένα κατηγόρημα το οποίο τυπώνει στην κονσόλα όλα τα στοιχεία μιας λίστας **χωρίς τη χρήση αναδρομής!**

```
write_lst(List):-
        member(X,List),
        write(X),nl,
        fail.
write_lst(_).
```

Failure Driven Loop



Κατηγορήματα Ανώτερης Τάξης

Κατηγορήματα Ανώτερης Τάξης

- Η Prolog **ΔΕΝ** αποτιμά τα ορίσματα των κατηγορημάτων.
- Υπάρχει ορισμένα κατηγορήματα τα οποία διαφεύγουν του παραπάνω κανόνα.
- Λόγοι για τους οποίους προστέθηκαν αυτά τα κατηγορήματα:
  - Αύξηση της εκφραστικής ικανότητας της γλώσσας.
  - Επιτρέπουν την ανάπτυξη περισσότερων πολύπλοκων προβλημάτων.

Μεταβλητή Κλήση - Variable Call

- Το κατηγόρημα ***call/1*** επιτυγχάνει όταν το όρισμά του είναι ένα κατηγόρημα που επιτυγχάνει.

  ?- call(member(1,[1,2,3])).

  yes
- Χρήση:
  - Δημιουργία κατηγορημάτων τα οποία αποτιμούν τα ορίσματά τους
  - Δημιουργία meta-interpreters.

Άρνηση ως Αποτυχία

- Negation as failure
- Να ορίσετε ένα κατηγόρημα *not_pred/1* το οποίο επιτυγχάνει όταν το όρισμά του (άλλο κατηγόρημα αποτυγχάνει.

```
not_pred(X):-
        call(X), !, fail.
not_pred(_).
```

Cut-Fail Combination

- Οι περισσότερες υλοποιήσεις της Prolog προσφέρουν το *not/1*.

Εκτέλεση υπό συνθήκη

- Να ορίσετε ένα κατηγόρημα ***if_then_else/3*** που έχει τη συνήθη σημασιολογία.

```
?- X = a, if_then_else(member(X, [a, b]), Y = 1, Y = 2).
X = a
Y = 1
Yes

?- X = c, if_then_else(member(X, [a, b]), Y = 1, Y = 2).
X = c
Y = 2
Yes
```

Εκτέλεση υπό συνθήκη

- Κατηγόρημα ***if_then_else/3*** :

```
if_then_else( Cond, ThenPart, ElsePart):-
    call( Cond ), !, call(ThenPart).

if_then_else( Cond, ThenPart, ElsePart):-
    call(ElsePart).
```

Eclipse Prolog Conditionals

- **Condition -> Then ; Else**
  - Πετυχαίνει όταν
    - Condition επιτυγχάνει και το Then επιτυγχάνει για την **πρώτη λύση** του Condition
    - Condition αποτυχαίνει και το Else επιτυγχάνει.
- **Condition *-> Then ; Else**
  - Όπως παραπάνω, αλλά και για κάθε λύση του Condition, όσον αφορά την πρώτη περίπτωση.

Τομή δύο Λιστών ver 3

- Να ορίσετε ένα κατηγόρημα intesect/3 το οποίο βρίσκει την τομή δύο λιστών.
- Με χρήση if-then-else

```
intersect([],_,[]).
intersect([X|Rest],List,Result):-
    intersect( Rest , List, IntRest),
    if_then_else(member(X,List),
                 Result=[X|IntRest],
                 Result = IntRest).
```

Σύνθεση/Αποδόμηση όρων

- Term Composition/Decomposition
- Κατηγόρημα Univ/2 αναπαρίσταται ως **=..**
- Term =..[Functor,Arg1,Arg2,..Argn]
- Παράδειγματα:

?- father(nick, john)=..[father, nick, john].

yes

?- member(X,[1,2,3])=..List.

List = [member,X,[1,2,3]]

Συνδυάζοντας Μεταβλητή Κλήση & Univ

- Να ορίσετε ένα κατηγόρημα το οποίο θα επιστρέφει την τομή ή την ένωση (append) δύο λιστών ανάλογα με το πρώτο όρισμά του. Παράδειγμα:

  ?- list_operation(append, [1, 2], [1, 2, 3, 4], L).
  L = [1, 2, 1, 2, 3, 4]
  Yes

  ?- list_operation(intersect, [1, 2], [1, 2, 3, 4], L).
  L = [1, 2]
  Yes

Συνδυάζοντας Μεταβλητή Κλήση & Univ

- Λύση:

```
list_operation(Oper,List1,List2,Result):-
        Predicate =..[Oper,List1,List2,Result],
        call( Predicate ).
```