Εργαστήριο Λογικού Προγραμματισμού

Μανόλης Μαρακάκης, Καθηγητής
mmarak@cs.hmu.gr

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Σχολή Μηχανικών
Ελληνικό Μεσογειακό Πανεπιστήμιο

Ενότητα 4: Μάθημα 7

Δέντρο Αναζήτησης Οπισθοδρόμησης και Άρνηση

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

- 4.1. Δέντρο Αναζήτησης
- 4.2. Οπισθοδρόμηση και Αποκοπή (!)
  - 4.2.1. Παραδείγματα με Αποκοπή
- 4.3. Άρνηση σε Prolog
- 4.4. Το κατηγόρημα fail

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση
4.1. Δέντρο Αναζήτησης

- **Ορισμός 4.1:** Έστω **P** ένα λογικό πρόγραμμα, **R** ένας κανόνας υπολογισμών και **G0** ένας στόχος. Μια ***SLD-εξαγωγή (SLD-derivation)*** του **G0**, χρησιμοποιώντας το **P** και τον **R**, είναι μια πεπερασμένη ή μη πεπερασμένη ακολουθία στόχων **G0, G1,…,Gi, Gi+1** όπου το **Gi+1** εξάγεται από το **Gi** και μια μετονομασμένη πρόταση **Ci** του **P** χρησιμοποιώντας τον **R**.
- Η ονομασία ***SLD-εξαγωγή*** προέρχεται από το όνομα του συμπερασματικού κανόνα που χρησιμοποιείται για εξαγωγή του **G0** ο οποίος ονομάζεται ***SLD-επίλυση***. Η ονομασία ***SLD-επίλυση (SLD-resolution)*** βγαίνει από τα αρχικά των λέξεων ***Selection Linear Definite*** που υπάρχουν στο πλήρες όνομα του συμπερασματικού κανόνα
  - ***Linear resolution for Definite clauses with Selection function.***

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση
4.1. Δέντρο Αναζήτησης

- **Ορισμός 4.2: Μια *SLD-απόρριψη (SLD-refutation)*** είναι **μια πεπερασμένη SLD-εξαγωγή G0,…,Gn** όπου το **Gn** είναι η **άδεια πρόταση (□).**
- **Ορισμός 4.3:** Έστω **P** ένα σύνολο από προτάσεις, **R** ένας κανόνας υπολογισμών και **G** μια πρόταση στόχος. Όλες οι δυνατές *SLD-εξαγωγές* μπορούν να επιδειχθούν σε ένα ***SLD-δέντρο (SLD-tree)*** ή ***δέντρο Αναζήτησης (search tree).*** Το ***SLD-δέντρο (SLD-tree)*** ή ***δέντρο αναζήτησης (search tree)*** του **G** (χρησιμοποιώντας το **P** και τον **R**) ορίζεται ως εξής:
  - **α**. Η ετικέτα στην **ρίζα του δέντρου** είναι ο αρχικός στόχος **G**.
  - **β. Εάν** το δέντρο περιέχει ένα κόμβο με ετικέτα τον στόχο G_i **και εάν** υπάρχει μια μετονομασμένη πρόταση **πi∈P** τέτοια ώστε ο G_{i+1} εξάγεται από τον G_i (**κεντρική πρόταση**) και την **πi** (**πλευρική πρόταση**) μέσω του **R** τότε ο κόμβος G_i έχει παιδί τον κόμβο G_{i+1}.

    Η ακμή που τους συνδέει έχει σαν ετικέτα την πρόταση π_i.

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση
4.1. Δέντρο Αναζήτησης

- Το κατηγόρημα **gonios(X, Y)** είναι αληθές εάν ο **X** είναι γονιός του **Y**.
- Το κατηγόρημα **progonos(X, Y)** είναι αληθές εάν ο **X** είναι πρόγονος του **Y**.

π1: gonios(yannis, kostas).

π2: gonios(yannis, elene).

π3: gonios(kostas, manos).

π4: gonios(kostas, anna).

π5: gonios(manos, nikos).

π6: progonos(X, Y) :- gonios(X, Y).

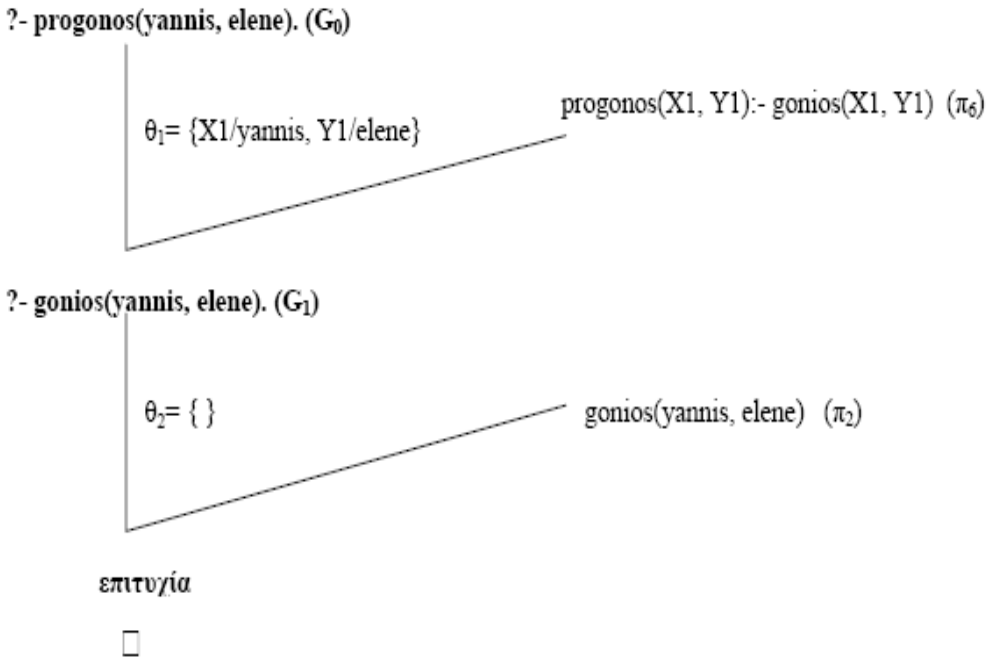
π7: progonos(X, Y) :- gonios(X, Z), progonos(Z, Y).

Πρόγραμμα 4.1: *Η σχέση progonos/2*

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

4.1. Δέντρο Αναζήτησης

- Έστω ότι θέλουμε να κάνουμε την εξής ερώτηση, ″**είναι ο Γιάννης πρόγονος της Ελένης;**″. Αυτή εκφράζεται σε Prolog από τον εξής στόχο **G0**, **?- progonos(yannis, elene).**



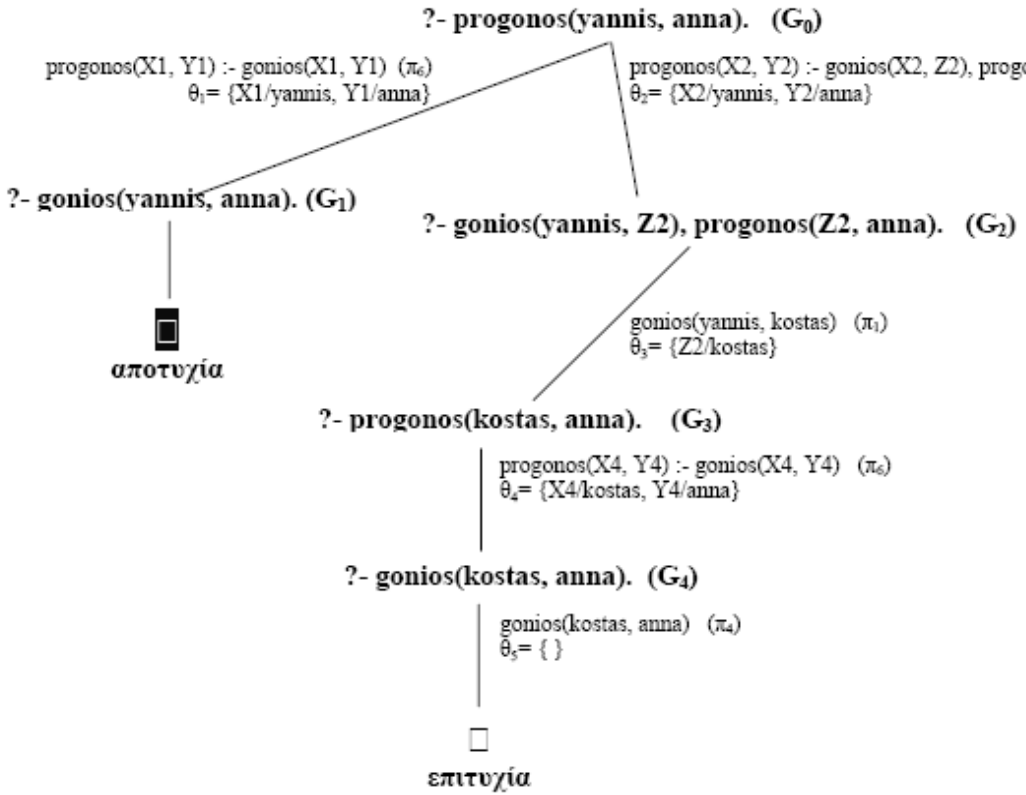
π1: gonios(yannis, kostas).
π2: gonios(yannis, elene).
π3: gonios(kostas, manos).
π4: gonios(kostas, anna).
π5: gonios(manos, nikos).
π6: progonos(X,Y) :-
gonios(X,Y).
π7: progonos(X,Y) :-
gonios(X,Z), progonos(Z,Y).

Σχήμα 4.1:SLD-απόρριψη για τον στόχο progonos(yannis, elene)

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

4.1. Δέντρο Αναζήτησης

- Η ερώτηση την οποία θέλουμε να κάνουμε είναι η εξής, **"είναι ο Γιάννης πρόγονος της Άννας;"**. Αυτή εκφράζεται σε Prolog από τον στόχο **G0, ?- progonos(yannis, anna).**



π1: gonios(yannis, kostas).

π2: gonios(yannis, elene).

π3: gonios(kostas, manos).

π4: gonios(kostas, anna).

π5: gonios(manos, nikos).

π6: progonos(X,Y) :-
gonios(X,Y).

π7: progonos(X,Y) :-
gonios(X,Z), progonos(Z,Y).

Σχήμα 4.2: Δέντρο έρευνας για "?- progonos(yannis, anna)."

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

4.1. Δέντρο Αναζήτησης

- Για να γίνει δυνατή η επίδειξη ενός πιο μεγάλου δέντρου αναζήτησης πρέπει το πρόγραμμα **Πρόγραμμα 4.1** να απλοποιηθεί στο **Πρόγραμμα 4.2**. (˝ Κάθε όνομα κατηγορήματος και σταθεράς του προγράμματος **Πρόγραμμα 4.1** θα αντικατασταθεί από το πρώτο του γράμμα.)

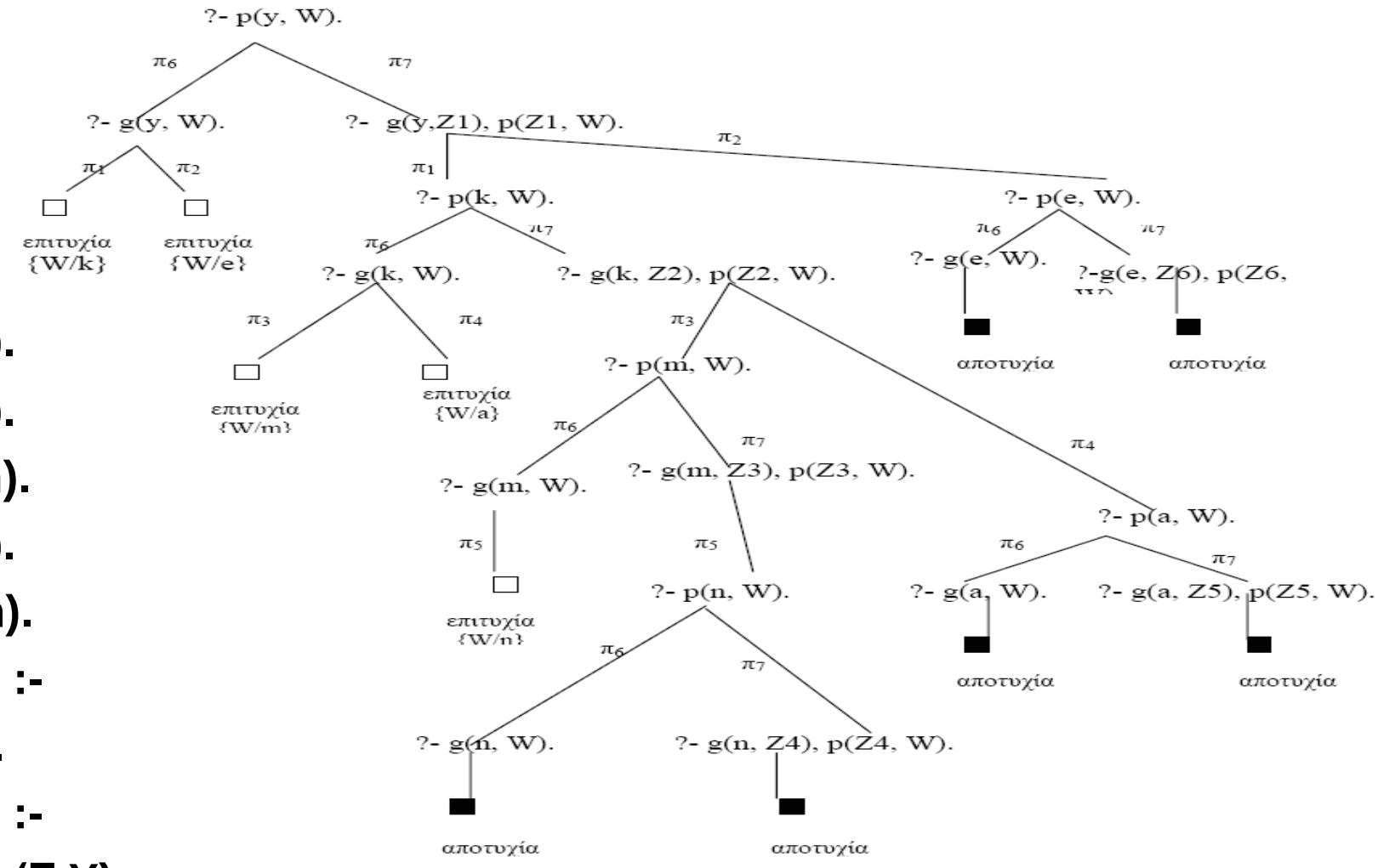
```
π1: g(y, k).
π2: g(y, e).
π3: g(k, m).
π4: g(k, a).
π5: g(m, n).
π6: p(X, Y) :- g(X, Y).
π7: p(X, Y) :- g(X, Z), p(Z, Y).
```

Πρόγραμμα 4.2: *Prolog πρόγραμμα για το κατηγόρημα p/2.*

- Θα θεωρήσουμε την ερώτηση ˝ **ποιοι είναι οι απόγονοι του Γιάννη;**˝ ή «**Τίνων / ποιων πρόγονος είναι ο Γιάννης;»,** η οποία εκφράζεται σε Prolog ως εξής, **?- p(y, W).** Το δέντρο αναζήτησης της ερώτησης φαίνεται στο **Σχήμα 4.3**.

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

4.1. Δέντρο Αναζήτησης



π1: g(y, k).

π2: g(y, e).

π3: g(k, m).

π4: g(k, a).

π5: g(m, n).

π6: p(X,Y) :-
g(X,Y).

π7: p(X,Y) :-
g(X,Z), p(Z,Y).

Σχήμα 4.3: Δέντρο έρευνας για τον στόχο ?- p(y, W).

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

4.2. Οπισθοδρόμηση και Αποκοπή (!)

- Η Prolog για να ικανοποιήσει ένα στόχο **G** δημιουργεί ένα δέντρο αναζήτησης **επιλέγοντας κάθε φορά τον πλέον αριστερό στοιχειώδη τύπο από τον στόχο**.
  Οπισθοδρόμηση έχουμε στις εξής δύο περιπτώσεις:
  1. **Όταν** ο επιλεγέντας πλέον αριστερός στοιχειώδης τύπος **αποτύχει τότε** η **Prolog οπισθοδρομεί αυτόματα στον προηγούμενο κόμβο του δέντρου αναζήτησης.**
     Προσπαθεί να βρει μια άλλη λύση για τον στόχο αυτού του κόμβου **ενοποιώντας τον πλέον αριστερό στοιχειώδη τύπο αυτού του στόχου με την κεφαλή της επόμενης πρότασης του προγράμματος,** και ούτω καθεξής.
  2. **Όταν** βρει μια λύση για τον αρχικό στόχο και ο χρήστης του προγράμματος θέλει και άλλη λύση, ο χρήστης δίνει την εντολή **';' (or)**, **οπότε η Prolog αρχίζει οπισθοδρόμηση με απαίτηση του χρήστη.**

4.2. Οπισθοδρόμηση και Αποκοπή (!)

- Σε κάποια προγράμματα ο προγραμματιστής θα επιθυμούσε να **ελέγχει** το τμήμα του SLD-δέντρου που θα ερευνηθεί.
  - **Όταν** ο προγραμματιστής γνωρίζει ότι **δεν υπάρχουν επιτυχείς κόμβοι σε κάποιο τμήμα του δέντρου αναζήτησης τότε** θα επιθυμούσε **να μην ερευνηθεί αυτό το τμήμα του SLD-δέντρου.**
- Η Prolog διαθέτει το κατηγόρημα ***αποκοπή ή κλάδεμα (cut) !.*** Η συνέπεια από την εκτέλεση του ! είναι ότι
  - **επιτυγχάνει μόλις κληθεί** και
  - **αποκόπτει ένα τμήμα του δέντρου αναζήτησης**.

4.2. Οπισθοδρόμηση και Αποκοπή (!)

- Έστω το **Πρόγραμμα 4.3** στο οποίο **p, q** είναι ατομικοί τύποι και **I1, I2, I3, I4, I5, I6, I7**, είναι **στοιχειώδεις τύποι**. Θεωρούμε τον στόχο **«?- p»**.

π1: p :- I1, I2, !, I3, I4.

π2: p :- I5, I6.

π3: q :- I7.

Πρόγραμμα 4.3: *Πρόγραμμα με !.*

- Εάν η πρόταση **π1** **αποτύχει πριν την ικανοποίηση του !** λόγω αποτυχίας κάποιου από τους στόχους **I1** ή **I2** τότε η Prolog προσπαθεί να ικανοποιήσει τον στόχο **"?- p."** με την επόμενη πρόταση **π2**.

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση
4.2. Οπισθοδρόμηση και Αποκοπή (!)

- **Πρόγραμμα 4.3**: στόχος **«?- p»**.

  π1: p :- l1, l2, !, l3, l4.

  π2: p :- l5, l6.

  π3: q :- l7.

- **Εάν** οι στοιχειώδεις τύποι **l1** και **l2** **επιτύχουν**, **η αποκοπή (!) επιτυγχάνει αυτόματα**. Οι συνέπειες με την ικανοποίηση της αποκοπής (!) είναι οι εξής:

  1. **Οπισθοδρόμηση δεν μπορεί να γίνει στους στόχους που βρίσκονται αριστερά της αποκοπής (!) στην πρόταση π1**, δηλαδή στο **l1** και **l2**. **Η ικανοποίηση του ! δεσμεύει το σύστημα σε όλες τις επιλογές τις οποίες έχει κάνει μέχρι την ικανοποίηση του !**. ***Σημείωση:*** Πριν την ικανοποίηση του ! οπισθοδρόμηση μπορεί να γίνει στους στόχους αριστερά του ! .
  2. **Οπισθοδρόμηση μπορεί να γίνει στους στόχους που βρίσκονται δεξιά της αποκοπής**, στο **l3** και **l4**. Η αποκοπή (!) δεν επηρεάζει τους στόχους που βρίσκονται δεξιά του στην πρόταση **π1**.
  3. Η ύπαρξη του ! στην πρόταση **π1** **δεν επιτρέπει την ενοποίηση του στόχου "?- p." με άλλη πρόταση με ίδια κεφαλή**. Δηλαδή, **εάν** η πρόταση **π1** αποτύχει **τότε** η Prolog αγνοεί την πρόταση **π2** λόγω ικανοποίησης του **!** στην **π1**.

4.2. Οπισθοδρόμηση και Αποκοπή (!)

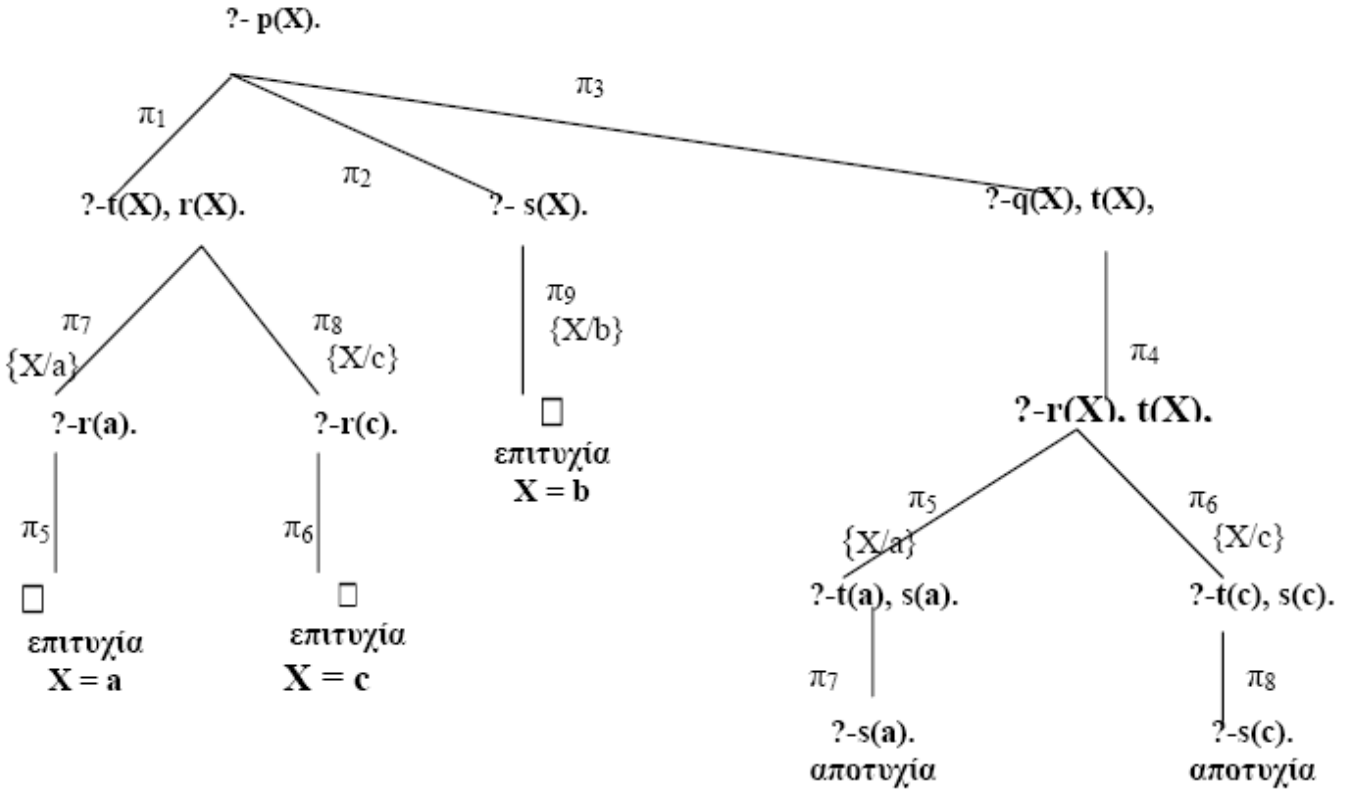
- ***Παράδειγμα:*** Έστω το **Πρόγραμμα 4.4**. Το δέντρο αναζήτησης αυτού του προγράμματος για τον στόχο "**?- p(X).**" φαίνεται στο **Σχήμα 4.4**. Στη συνέχεια θα μελετήσουμε διαφορετικές εκδόσεις του ίδιου προγράμματος με αποκοπή (!) για να δούμε σε κάθε περίπτωση ποιο μέρος του δέντρου αναζήτησης ερευνάται και ποιο αποκόπτεται

```
π1: p(X)  :- t(X), r(X).
π2: p(X)  :- s(X).
π3: p(X)  :- q(X), t(X), s(X).
π4: q(X)  :- r(X).
π5: r(a).
π6: r(c).
π7: t(a).
π8: t(c).
π9: s(b).
```

Πρόγραμμα 4.4:*Πρόγραμμα χωρίς !*

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση
4.2. Οπισθοδρόμηση και Αποκοπή (!)

π1: p(X) :-
t(X), r(X).
π2: p(X) :- s(X).
π3: p(X) :-
q(X), t(X), s(X).
π4: q(X) :- r(X).
π5: r(a).
π6: r(c).
π7: t(a).
π8: t(c).
π9: s(b).



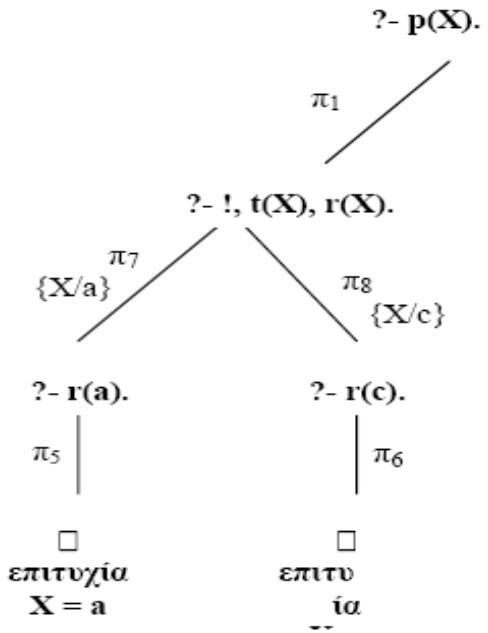
Σχήμα 4.4: *Δέντρο έρευνας του προγράμματος* **Πρόγραμμα 4.4** *για τον στόχο ?- p(X).*

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση

4.2. Οπισθοδρόμηση και Αποκοπή (!)

- ***Περίπτωση 1:***

π1: p(X) :- !, t(X), r(X).
π2: p(X) :- s(X).
π3: p(X) :- q(X), t(X), s(X).
π4: q(X) :- r(X).
π5: r(a). π6: r(c). π7: t(a). π8: t(c). π9: s(b).



Σχήμα 4.5: *Δέντρο έρευνας του προγράμματος* **Πρόγραμμα 4.5** *για στόχο ?- p(X).*

4. Δέντρο Αναζήτησης, Οπισθοδρόμηση και Άρνηση
4.2. Οπισθοδρόμηση και Αποκοπή (!)

- ***Περίπτωση 2:***

π1: p(X) :- t(X), !, r(X).

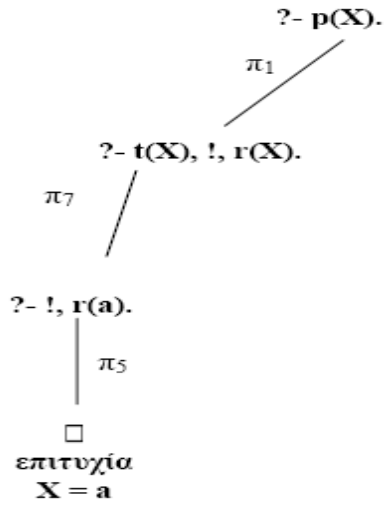
π2: p(X) :- s(X).

π3: p(X) :- q(X), t(X), s(X).

π4: q(X) :- r(X).

π5: r(a). π6: r(c). π7: t(a). π8: t(c). π9: s(b).

Πρόγραμμα 4.6:*Πρόγραμμα με ! (Περίπτωση 2)*



Σχήμα 4.6:*Δέντρο έρευνας του προγράμματος* Πρόγραμμα 4.6 *για στόχο ?- p(X).*

4.2. Οπισθοδρόμηση και Αποκοπή (!)

- ***Περίπτωση 3:***

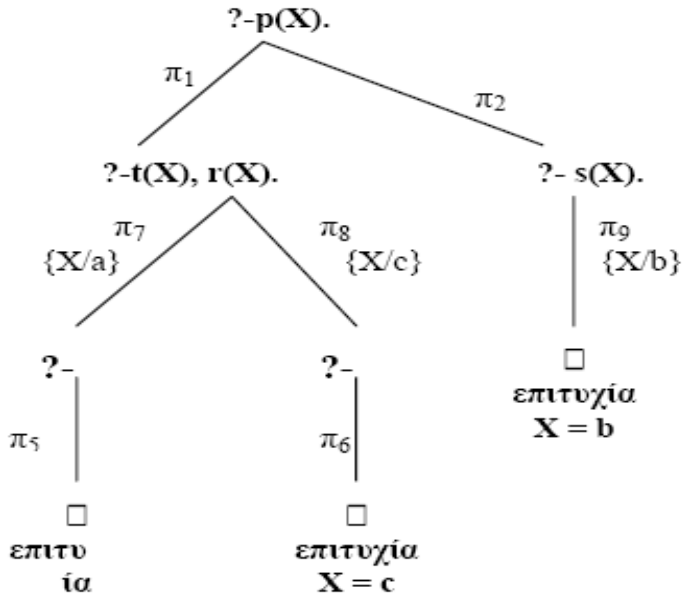
π1: p(X) :- t(X), r(X).

π2: p(X) :- s(X), !.

π3: p(X) :- q(X), t(X), s(X).

π4: q(X) :- r(X).

π5: r(a). π6: r(c). π7:t(a). π8: t(c). π9: s(b).



Σχήμα 4.7:*Δέντρο έρευνας του προγράμματος* Πρόγραμμα 4.7 *για στόχο ?- p(X).*

4.2. Οπισθοδρόμηση και Αποκοπή (!)

- ***Παράδειγμα 2***: Το κατηγόρημα **enoseTaxinList(List1, List2, List3)** είναι αληθές εάν η λίστα **List3** είναι μια ταξινομημένη λίστα η οποία προκύπτει από την ένωση των ταξινομημένων λιστών **List1** και **List2**.

```
enoseTaxinList( [Kef1 | List1], [Kef2 | List2], [Kef1 | List3] ) :-
        Kef1 < Kef2, !,
        enoseTaxinList( List1, [Kef2 | List2], List3 ).
enoseTaxinList([Kef1|List1], [Kef2|List2], [Kef1,Kef2|List3]) :-
        Kef1 = Kef2, !,
        enoseTaxinList( List1, List2, List3 ).
enoseTaxinList( [Kef1 | List1], [Kef2 | List2], [Kef2 | List3] ) :-
        Kef1 > Kef2,
        enoseTaxinList( [Kef1 | List1], List2, List3 ).
enoseTaxinList( List1, [ ], List1 ).
enoseTaxinList( [ ], List2, List2 ).
```

Πρόγραμμα 4.10: Ένωση *Δυο Ταξινομημένων Λιστών.*

Τέλος Διάλεξης

Ευχαριστώ!

Ερωτήσεις;