

Τεχνητή Νοημοσύνη

Λογικός Προγραμματισμός - Prolog

Δρ. Δημήτριος Κουτσομητρόπουλος

ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (1)

- ▶ Εξαγωγή αποτελεσμάτων-συμπερασμάτων από ένα πεπερασμένο σύνολο λογικών εκφράσεων, που ονομάζεται **λογικό πρόγραμμα** (logic program)
- ▶ Πηγάζει από το πεδίο της αυτοματοποιημένης απόδειξης θεωρημάτων (automated theorem proving) με απαιτήσεις υψηλότερης αποδοτικότητας.
- ▶ Γι' αυτό χρησιμοποιεί μια περιορισμένη μορφή λογικής, που οδηγεί σε αυτό που ονομάζεται **οριστικό λογικό πρόγραμμα** (definite logic program)



ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (2)

- ▶ Η λογική αυτή χρησιμοποιεί λογικές προτάσεις της μορφής:
- ▶ $\forall x_1 \forall x_2 \dots \forall x_m (A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow A_o) (n \geq 0)$
- ▶ όπου $A_o, A_1, A_2, \dots, A_n$ είναι (θετικές) ατομικές εκφράσεις
- ▶ Η CNF-προτασιακή μορφή αυτής είναι:
- ▶ $A_o \vee \neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n (n \geq 0)$
- ▶ ή $\{A_o, \neg A_1, \neg A_2, \dots, \neg A_n\} (n \geq 0)$
- ▶ Δηλ. αντιστοιχεί σε προτάσεις με ένα θετικό στοιχείο (το A_o).
- ▶ Αυτές οι προτάσεις λέγονται **οριστικές προτάσεις** (definite clauses) ή **τύπου Horn** και παριστάνουν θετική γνώση.

ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (3)

- ▶ Μια τέτοια πρόταση στον λογικό προγραμματισμό συμβολικά γράφεται ως εξής:

$$\begin{array}{c} A_o \leftarrow A_1, A_2, \dots, A_n \\ \underbrace{\hspace{1cm}} \quad \underbrace{\hspace{1cm}} \\ \text{κεφαλή (head)} \quad \text{σώμα (body)} \end{array}$$

και είναι τύπου: **κανόνας** (rule)

- Αν $n=0$, τότε δεν υπάρχει σώμα, παραλείπεται το $\leftarrow$ και η πρόταση γίνεται τύπου: **γεγονός** (fact):
$$A_o$$
- Αν δεν υπάρχει κεφαλή τότε η πρόταση παίρνει την μορφή:
$$\leftarrow A_1, A_2, \dots, A_n$$

και λέγεται τύπου: **στόχος** (goal). Τα $A_1, A_2, \dots, A_n$ αποτελούν **υποστόχους** (subgoals).

ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (4)

Για να δούμε τη λογική πίσω από μια πρόταση-στόχο, ας θεωρήσουμε την ισοδύναμη μορφή σε ΚΛΠΤ:

$$\forall x_1 \forall x_2 \dots \forall x_m \neg (A_1 \wedge A_2 \wedge \dots \wedge A_n)$$

που μπορεί να γραφεί

$$\neg \exists x_1 \exists x_2 \dots \exists x_m (A_1 \wedge A_2 \wedge \dots \wedge A_n)$$

που αποτελεί την άρνηση της

$$\exists x_1 \exists x_2 \dots \exists x_m (A_1 \wedge A_2 \wedge \dots \wedge A_n)$$

που αποτελεί ένα ερώτημα σε ΚΛΠΤ (δηλ. μια προς απόδειξη πρόταση υπαρξιακού τύπου)



ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (5)

Η απόδειξη ενός στόχου γίνεται με τη χρήση του κανόνα **SLD-Επίλυση** (SLD-Resolution principle):

$$\frac{(\leftarrow A_1, \dots, A_{i-1}, A_i, A_{i+1}, \dots, A_n) \quad (B_o \leftarrow B_1, B_2, \dots, B_m)}{(\leftarrow A_1, \dots, A_{i-1}, B_1, B_2, \dots, B_m, A_{i+1}, \dots, A_n)\theta}$$

όπου $\theta = \gamma.ε.(A_i, B_o)$

Η SLD-Επίλυση είναι *refutation complete* μόνο για Horn clauses!



ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (6)

Ας υποθέσουμε ότι ένα σύνολο αποτελείται από τις παρακάτω προτάσεις ΚΛΠΤ:

$\forall x_o \forall y_o \forall z (father(x_o, y_o) \wedge parent(y_o, z_o)) \Rightarrow grandfath(x_o, z_o)$

$\forall x_1 \forall y_1 father(x_1, y_1) \Rightarrow parent(x_1, y_1)$

$\forall x_2 \forall y_2 mother(x_2, y_2) \Rightarrow parent(x_2, y_2)$

$father(a, b)$

$moth(b, c)$

Ερώτηση: Who has a as grandfather?

ΚΛΠΤ: $\exists x grandfath(a, x)$

Το ισοδύναμο περιορισμένο λογικό πρόγραμμα θα είναι:

(1) $grandfath(x_o, z_o) \leftarrow father(x_o, y_o), parent(y_o, z_o)$

(2) $parent(x_1, y_1) \leftarrow father(x_1, y_1)$

(3) $parent(x_2, y_2) \leftarrow mother(x_2, y_2)$

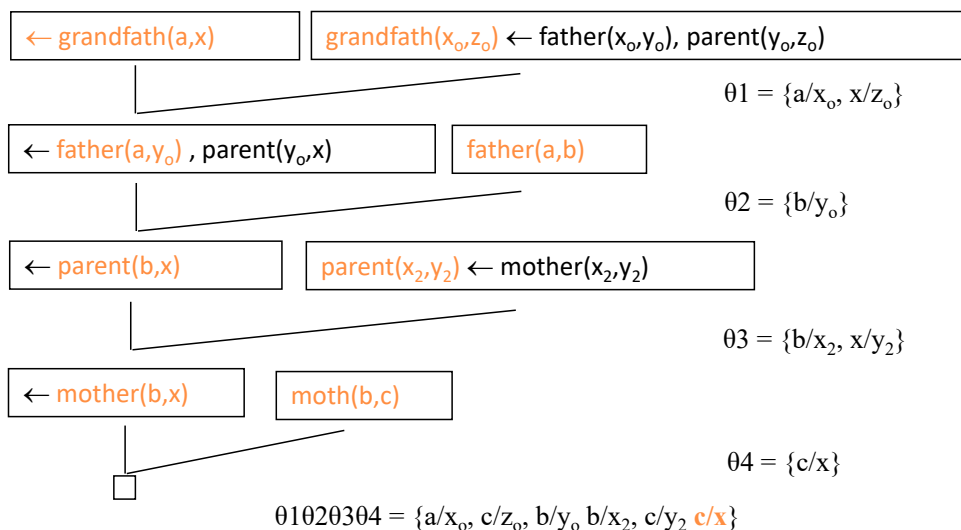
(4) $father(a, b)$

(5) $moth(b, c)$

ΛΠ: $\leftarrow grandfath(a, x)$

ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (7)

Μια απόδειξη (εξαγωγή) του στόχου με χρήση SLD-Επίλυση είναι:



ΛΟΓΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ (8)

- Τα οριστικά προγράμματα εκφράζουν μόνο θετική γνώση (πώς σχετίζονται μεταξύ τους οντότητες του πεδίου του προβλήματος)
- Δεν εκφράζουν αρνητική γνώση (τι δεν ισχύει)
- Για να υπερβούμε το εμπόδιο αυτό χρησιμοποιείται η λεγόμενη **«υπόθεση κλειστού κόσμου» (closed world assumption-CWA)**
- $KB^+ \rightarrow \{ KB \cup \neg f \mid KB \not\models f \}$
- Η cwa είναι μια αρχή που αναφέρει ότι αν ένα θετικό στοιχείο A χωρίς μεταβλητές δεν μπορεί να αποδειχθεί από ένα οριστικό πρόγραμμα τότε συμπεραίνουμε το $\neg A$.
- Η KB μπορεί να γίνει εύκολα **ασυνεπής** για μη Horn προτάσεις, π.χ.
{ Γλυκό(φρούτο) $\vee$ Ξινό (φρούτο), $\neg$ Γλυκό(φρούτο), $\neg$ Ξινό (φρούτο) }
- Πώς όμως γνωρίζουμε αν $KB \not\models f$;
- **«άρνηση ως αποτυχία» (negation as failure-NAF) ← Prolog**
 - Πιο περιορισμένη εκδοχή της CWA
 - δεν μπορεί να ανιχνεύσει καταστάσεις που έχουμε άπειρους κλάδους (infinite branches) στο SLD-δέντρο, παρά μόνο κλάδους που είναι αδιέξοδοι.



PROLOG

- ▶ PROLOG ← PROgramming in LOGic
- ▶ Πρώτη υλοποίηση: Alain Colmerauer, Μασσαλία
 - ▶ Αρχή Επίλυσης, Εργασία R. Kowalski
- ▶ Δεύτερη υλοποίηση: D. Warren, Εδιμβούργο
- ▶ Δηλωτική γλώσσα: διάκριση μεταξύ 'λογικής' και 'ελέγχου' σ' ένα πρόγραμμα
 - ▶ Kowalski: πρόγραμμα = λογική + έλεγχος



ΒΑΣΙΚΑ

- ▶ Ένα πρόγραμμα Prolog αφορά οντότητες του προβλήματος με το οποίο ασχολείται, τις ιδιότητές τους και τις σχέσεις τους.
 - ▶ Ένα πρόγραμμα Prolog μπορεί να περιλαμβάνει
 - ▶ **Γεγονότα (Facts):** παριστάνουν συγκεκριμένες ιδιότητες ή σχέσεις μεταξύ συγκεκριμένων οντοτήτων του προβλήματος-τα δεδομένα του προβλήματος.
 - ▶ **Κανόνες (Rules):** παριστάνουν γενικευμένες σχέσεις μεταξύ οντοτήτων του προβλήματος-οι συλλογισμοί για την επίλυση του προβλήματος.
 - ▶ **Ερωτήματα (Questions/Queries):** παριστάνουν τα ζητούμενα ενός προβλήματος-ο τρόπος αλληλεπίδρασης με τον χρήστη.
 - ▶ Σχόλια: ό,τι βρίσκεται ανάμεσα σε /* ... */
-



ΑΤΟΜΙΚΕΣ ΕΚΦΡΑΣΕΙΣ (1)

- ▶ Μια ατομική έκφραση έχει την εξής μορφή:
 - ▶ <κατηγόρημα>(<όρος1>, <όρος2>, ..., <όροςn>).
 - ▶ Το **κατηγόρημα (predicate)** εκφράζει μια ιδιότητα μιας οντότητας ή μια σχέση μεταξύ δύο ή περισσότερων οντοτήτων. Ο αριθμός των σχετιζόμενων οντοτήτων ονομάζεται **τάξη (arity)** του κατηγορήματος. Ένα κατηγόρημα είναι ένα αλφαριθμητικό (συμβολοσειρά) που ξεκινά με μικρό γράμμα, επιτρέπεται η χρήση του '_' (π.χ. loves, mother, man, make_tree).
-



ΑΤΟΜΙΚΕΣ ΕΚΦΡΑΣΕΙΣ (2)

- ▶ Ένας **όρος (term)** μπορεί να είναι:
 - ▶ Μια **σταθερά (constant)**, δηλ. ένας αριθμός (π.χ. 2, 3.4, 8, -10) ή ένα αλφαριθμητικό (συμβολοσειρά) που ξεκινά με μικρό γράμμα (επιτρέπεται και το '_') ή βρίσκεται σε απλά εισαγωγικά, π.χ. class1, x_1, giannis, p2, 'Petros'. Μια σταθερά παριστάνει μια συγκεκριμένη οντότητα.
 - ▶ Μια **μεταβλητή (variable)**, δηλ. ένα αλφαριθμητικό (συμβολοσειρά), που ξεκινά με κεφαλαίο γράμμα ή το '_' (π.χ. X, X2, _Y, Obj3). Μια μεταβλητή αντιπροσωπεύει ένα σύνολο ομοειδών οντοτήτων. Υπάρχει η **ανώνυμη μεταβλητή (_)**, που χρησιμοποιείται όπως μια μεταβλητή, αλλά όταν δεν θέλουμε να χρησιμοποιήσουμε την τιμή δέσμησης της μεταβλητής.
 - ▶ Μια **δομή (structure)**, δηλ. ένας σύνθετος όρος της μορφής: <όνομα-δομής>(<όρος-1>, <όρος-2>, ..., <όρος-n>). Αντιπροσωπεύει μια οντότητα (όνομα-δομής) με συγκεκριμένα χαρακτηριστικά (ορος-1, ορος-2 ... ορος-n).



ΓΕΓΟΝΟΤΑ (FACTS)

- ▶ Γεγονός = ατομική έκφραση χωρίς μεταβλητές.

- ▶ Παραδείγματα

man(giannis). (man/1)

father(giannis, kostas). (father/2)

gives(giannis, maria, book1). (gives/3)

gives(giannis, maria, book(black_horse, ellis)). (gives/3)



ΚΑΝΟΝΕΣ (1)

- ▶ Ένας κανόνας έχει την μορφή:
 - ▶ <έκφραση-κεφαλή> :- <έκφραση-σώμα>.
- ▶ Η <έκφραση-κεφαλή> ενός κανόνα είναι πάντα μια ατομική έκφραση και ονομάζεται **κεφαλή (head)** του κανόνα.
- ▶ Η <έκφραση-σώμα> είναι μια σύνθετη έκφραση και ονομάζεται **σώμα (body)** του κανόνα:
 - ▶ <έκφραση-σώμα> = <ατομ. έκφρ.-1><λογικός-τελεστής-1> ... <λογικός-τελεστής-n><ατομ.-έκφρ.-n>.
- ▶ Οι **λογικοί τελεστές** είναι: το ' , ' (AND), το ' ; ' (OR) και το 'not' (NOT).



ΚΑΝΟΝΕΣ(2)

- ▶ Παραδείγματα
 - ▶ likes(giannis, X) :- likes(X, wine).
 - ▶ parent(X, Y) :- father(X, Y).
 - ▶ sibling(X, Y) :- father(Z, X), father(Z, Y).
 - ▶ sibling(X, Y) :- brother(X, Y); sister(X, Y).
- ▶ Διπλή ανάγνωση ενός κανόνα
 - ▶ Δηλωτική: Εάν <σώμα> τότε <κεφαλή>
 - ▶ Διαδικαστική: Για να αποδειχθεί η <κεφαλή> θα πρέπει να αποδειχθεί το <σώμα>



ΕΡΩΤΗΣΕΙΣ

▶ ?- <έκφραση>

▶ Π.χ.

?- likes (giannis, maria). (απάντηση: yes-no)

?- likes (giannis, X). (απάντηση: μεταβλητή-τιμή)

?- likes (_, maria). (απάντηση: yes-no)

▶

▶

ΕΞΟΔΟΣ ΣΤΗΝ ΟΘΟΝΗ

▶ write(X) → εκτύπωση της τιμής της X

▶ write(hello) → hello

▶ write('Hello') → Hello

▶ nl → αλλαγή γραμμής

▶ ?- write(one), write(two) → onetwo

▶ ?- write('one '), write(two) → one two

▶ ?- write(one), nl, write(two) →

one

two

▶

ΕΙΣΟΔΟΣ ΑΠΟ ΠΛΗΚΤΡΟΛΟΓΙΟ

- ▶ `read(X).` (εισαγωγή όρων με δέσμευση της X)
 `|: george.` (!!!προσοχή στην τελεία)
 `X = george`
 `Yes`



ΤΕΛΕΣΤΕΣ ΣΥΓΚΡΙΣΗΣ

- ▶ Τελεστές σύγκρισης: `=`, `\=`, `<`, `>`, `=<`, `>=`
- ▶ Εκφράσεις : `2 > 4`, `X =< 5`, `X = Y`

```
king-time (george, 867, 920).  
king-time (john, 920, 960).  
king-time (paul, 961, 990).  
king (X, Y) :- king-time (X, A,  
B),  
                                    Y >= A,  
                                    Y =< B.
```

```
?- king (george,  
900).  
Yes  
?- king (john, 900).  
No  
?- king (X, 930).  
X=john  
false  
?- king (X, 920).  
X=george;  
X=john;  
false
```



ΠΡΑΞΕΙΣ

- ▶ Αριθμητικοί τελεστές: +, -, *, /, mod
 - ▶ Αριθμητικές εκφράσεις: $3*4+5$, $X*Y+2$, $-2*X$
 - ▶ Τελεστής υπολογισμών: is

```
tr-dim (abc, 10, 15).  
tr-dim (abd, 8, 12).  
tr-dim (ade, 6, 10).  
tr-area (X, A) :- tr-dim (X, Y1,  
Y2),  
A is  
Y1*Y2/2.
```

```
?- tr-area (abd,  
X).  
X=48  
Yes  
?- tr-area (adg,  
X).  
No
```

ΤΟ ΚΑΤΗΓΟΡΗΜΑ ΙΣΟΤΗΤΑΣ (1)

- ▶ Χρησιμοποιείται ως **ενδοθεματικός** τελεστής: $X=Y$
- ▶ Τα X , Y είναι εν γένει δύο όροι που επιτρέπεται να περιέχουν αδέσμευτες μεταβλητές.
- ▶ Ο έκφραση-στόχος « $X=Y$ » (X ίσον Y) έχει σαν αποτέλεσμα τον έλεγχο ταιριάσματος (matching) των X , Y , δηλ. έλεγχο του αν είναι ταυτόσημα ή μπορούν να γίνουν ταυτόσημα. Αν ταιριάζουν, τότε ο στόχος επιτυγχάνει (true), αλλιώς αποτυγχάνει (false).
- ▶ Αντίθετα ενεργεί το κατηγορήμα ανισότητας: $X \neq Y$

ΤΟ ΚΑΤΗΓΟΡΗΜΑ ΙΣΟΤΗΤΑΣ (2)

- ▶ Κανόνες ελέγχου ισότητας/ταιριάσματος της έκφρασης $X=Y$
 - ▶ Αν X είναι αδέσμευτη και Y είναι δεσμευμένη (ή μια σταθερά), τότε X, Y είναι ίσες (ταιριάζουν). Επί πλέον, η X αναγκάζεται να δεσμευτεί από τον ίδιο όρο που δεσμεύεται και η Y (ή τη σταθερά).
 - ▶ $\text{plays}(\text{postman}, \text{football}) = X$. (Επιτυγχάνει, και η X δεσμεύεται από τη δομή « $\text{plays}(\text{postman}, \text{football})$ »)
 - ▶ Οι ακέραιοι αριθμοί και τα άτομα είναι ίσα (ταιριάζουν) με τον εαυτό τους. ($126 = 126$, $\text{ball} = \text{ball}$)
 - ▶ Δύο δομές είναι ίσες αν έχουν το ίδιο όνομα και αριθμό όρων και όλοι οι αντίστοιχοι όροι είναι ίσοι (ταιριάζουν).
 - ▶ $\text{plays}(\text{postman}, \text{football}) = \text{plays}(\text{postman}, X)$ (Επιτυγχάνει, και η X δεσμεύεται από τον όρο 'football').
 - ▶ Στην περίπτωση που έχουμε δύο αδέσμευτες μεταβλητές, τότε ταιριάζουν και υποχρεώνονται να είναι «κοινής χρήσης», δηλ. οποτεδήποτε η μία δεσμεύεται από ένα όρο, δεσμεύεται και η άλλη από τον ίδιο όρο.

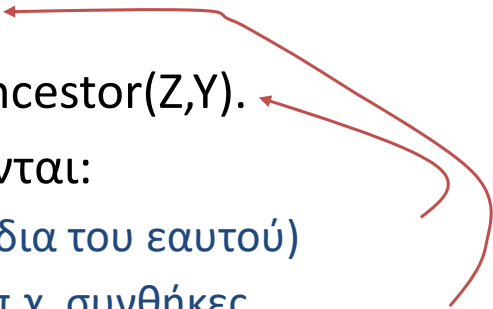


ΑΝΑΔΡΟΜΙΚΟΤΗΤΑ (1)

- ▶ Η αναδρομή (recursion) αναφέρεται σε περιπτώσεις όπου ο ορισμός ενός κατηγορήματος (κανόνα) γίνεται μέσω του εαυτού του.
- ▶ Π.χ. Ας υποθέσουμε ότι θέλουμε να ορίσουμε το κατηγορήμα 'πρόγονος'. Ως γνωστό: «Πρόγονος κάποιου είναι ο γονέας του. Επίσης, ο γονέας του γονέα του, ο γονέας του γονέα του γονέα του ...». Στην PROLOG θα έπρεπε να γράψουμε :
 - ▶ $\text{ancestor}(X,Y) : - \text{parent}(X,Y).$
 - ▶ $\text{ancestor}(X,Y) : - \text{parent}(X,Z), \text{parent}(Z,Y).$
 - ▶ $\text{ancestor}(X,Y) : - \text{parent}(X,Z1), \text{parent}(Z1,Z2), \text{parent}(Z2,Y).$
 - ▶ Κ.Ο.Κ.

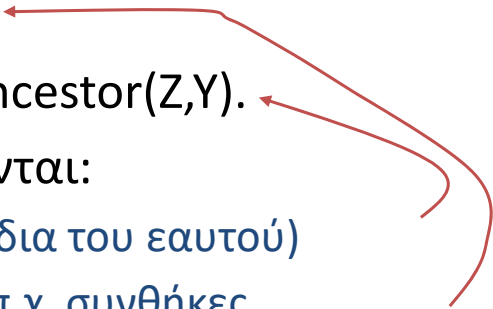


ΑΝΑΔΡΟΜΙΚΟΤΗΤΑ (2)

- ▶ Η λύση είναι ένας αναδρομικός ορισμός του 'ancestor':
 $\text{ancestor}(X,Y) : - \text{parent}(X,Y).$
 $\text{ancestor}(X,Y) : - \text{parent}(X,Z), \text{ancestor}(Z,Y).$
 - ▶ Σε περίπτωση αναδρομής απαιτούνται:
 - ▶ Ένας αναδρομικός ορισμός (ορισμός δια του εαυτού)
 - ▶ Προσδιορισμός οριακών συνθηκών (π.χ. συνθήκες τερματισμού)
- 



ΛΙΣΤΕΣ (1)

- ▶ Ένας άλλος τύπος όρου ή δομής στην Prolog.
 - ▶ Είναι μια διατεταγμένη ακολουθία στοιχείων (χωρίς καθορισμένο μήκος).
 - ▶ Διακρίνουμε την **κεφαλή (head)** και την **ουρά (tail)**.
 - ▶ Μια λίστα μπορεί να περιέχει σαν στοιχεία οποιουσδήποτε όρους (σταθερές, μεταβλητές, δομές, άλλες λίστες).
 - ▶ Η **κενή λίστα** είναι μια λίστα χωρίς στοιχεία.
- 



ΛΙΣΤΕΣ (2)

- ▶ Παραδείγματα:

[the, ship, [has, good, fish]]

[a, X, 2, [Y, Z], [b]]

[]

- ▶ Υπάρχει ειδικός συμβολισμός για την εξαγωγή (δέσμευση σε μεταβλητές) της κεφαλής και της ουράς μιας λίστας: **[X | Y]** .

p([a, b, c]).

p([the, ship, [has, good, fish]]).

?- p([X | Y]).

X = a Y = [b, c] ;

X = the Y = [ship, [has, good, fish]]

?- p([_, _, [_ | X]]).

X = [good, fish]



ΑΝΑΔΡΟΜΙΚΗ ΑΝΑΖΗΤΗΣΗ (1)

- ▶ Η αναζήτηση ενός στοιχείου σε μια δομή που περιέχει άλλες ίδιες δομές (π.χ. λίστα που περιέχει λίστες) ή μπορεί να θεωρηθεί ως συνιστώμενη από όμοιες υπό-δομές απαιτεί αναδρομή.



ΑΝΑΔΡΟΜΙΚΗ ΑΝΑΖΗΤΗΣΗ (2)

- ▶ Παράδειγμα: Αναζήτηση στοιχείου σε λίστα:

`member(X, L)`

- ▶ Ορισμός: Το X είναι μέλος της λίστας L αν

(α) το X είναι η κεφαλή της L είτε

(β) το X είναι μέλος της ουράς της L

- ▶ Τα παραπάνω μεταφράζονται ως εξής:

`member(X, [X|_]).`

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

(αναδρομικός ορισμός: ορισμός του `member` δια του εαυτού του)



ΧΕΙΡΙΣΜΟΣ ΛΙΣΤΩΝ

- ▶ `member(X, L)` (επιτυχία, αν η τιμή της μεταβλητής X είναι μέλος της λίστας L, αλλιώς αποτυχία). Π.χ.

- ▶ `?- member(a, [b, a, c]) → Yes`

- ▶ `append(L1, L2, X)` (Ενοποίηση των L1, L2 και το αποτέλεσμα στη μεταβλητή X). Π.χ.

- ▶ `?- append([a, b], [1, 2, 3], X) → X=[a, b, 1, 2, 3]`

- ▶ `?- append(X, [c, d], [a, b, c, d]) → X=[a, b]`

- ▶ `length(L, N)` (Το πλήθος των στοιχείων της L ανατίθεται στην N). Π.χ.

- ▶ `?- length([a, b, c, d], N) → N=4`



Συλλογισμός στην Prolog

- ▶ Η SLD-Resolution καθορίζει το **πώς** θα σχηματιστεί το δέντρο των αποδείξεων
 - ▶ Δεν καθορίζει πώς θα γίνει η διαπέρασή του
- ▶ Η Prolog χρησιμοποιεί **depth-first backward chaining**
 - ▶ Οι προτάσεις δοκιμάζονται με τη σειρά που είναι γραμμένες στην KB
 - ▶ Ο συλλογισμός επομένως είναι **μη πλήρης**, *ακόμα και σε Datalog*
 - ▶ *Infinite paths*
 - ▶ Επίσης, *redundant paths* λόγω DFS
 - ▶ Συμβιβασμός μεταξύ αποδοτικότητας και εκφραστικότητας
- ▶ Εξω-λογικά χαρακτηριστικά
 - ▶ Αριθμητικές συναρτήσεις: εκτελείται κώδικας και όχι συμπερασμός
 - ▶ Π.χ. "X is 4+3"
 - ▶ Build-in κατηγορήματα: CUT, ASSERT, RETRACT
 - ▶ Δεν αντιστοιχούν στη Λογική και μπορούν να προκαλέσουν συνέπειες ή αλλαγές στην KB

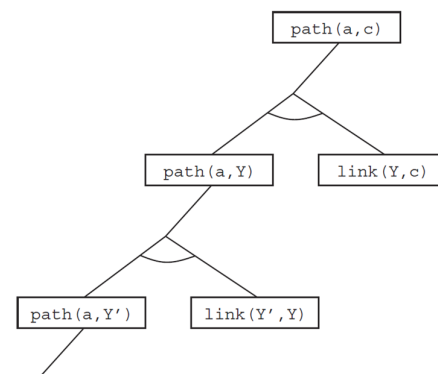
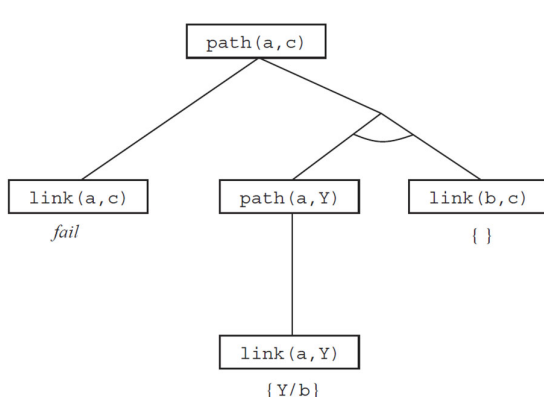


Συλλογισμός στην Prolog – Μη πληρότητα

Έστω οι παρακάτω κανόνες για τον έλεγχο ύπαρξης μονοπατιού ανάμεσα σε 2 κόμβους

Και τα γεγονότα $\text{link}(a,b)$ και $\text{link}(b,c)$, δηλ. $a \rightarrow b \rightarrow c$

$\text{path}(X,Z) \text{ :- } \text{link}(X,Z).$ $\text{path}(X,Z) \text{ :- } \text{path}(X,Y), \text{link}(Y,Z).$	$\text{path}(X,Z) \text{ :- } \text{path}(X,Y), \text{link}(Y,Z).$ $\text{path}(X,Z) \text{ :- } \text{link}(X,Z).$
--	--

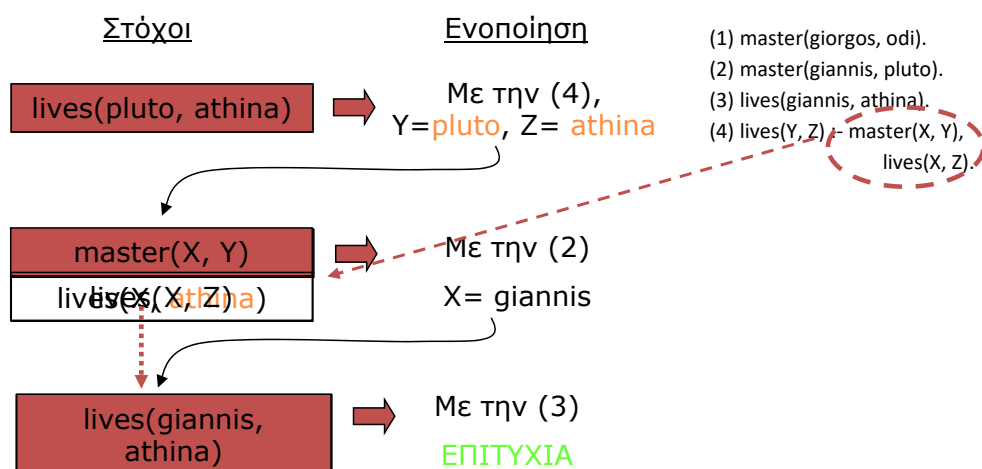


Η μη πληρότητα της SLD-επίλυσης επομένως οφείλεται στον **τρόπο που γίνεται η αναζήτηση στο SLD-δέντρο (DFS backward-chaining)**

ΔΙΑΔΙΚΑΣΙΑ ΑΠΟΔΕΙΞΗΣ (1)

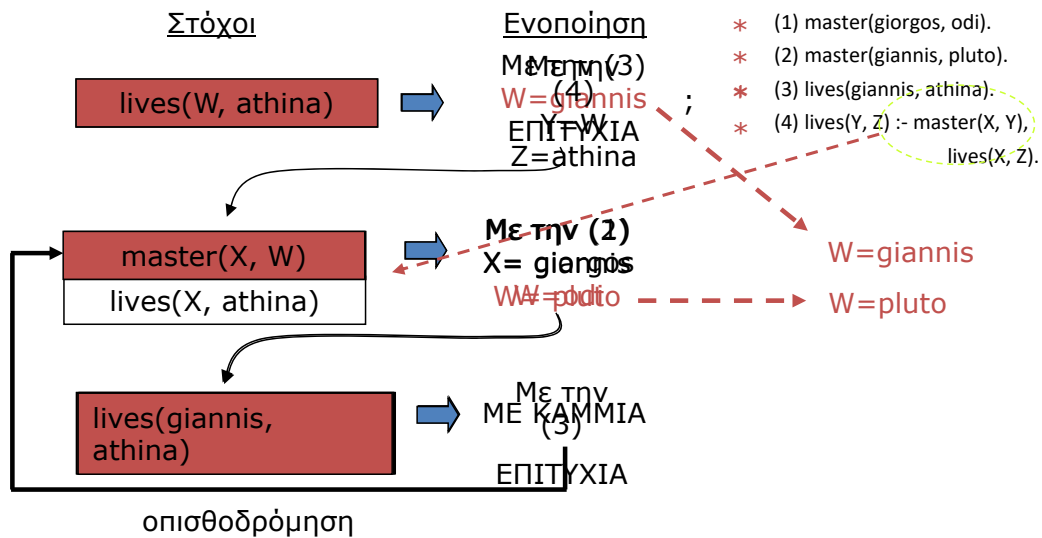
- ▶ Έχουμε τις προτάσεις:
 - (1) `master(giorgos, odi)`.
 - (2) `master(giannis, pluto)`.
 - (3) `lives(giannis, athina)`.
 - (4) `lives(Y, Z) :- master(X, Y), lives(X, Z)`.
- ▶ Θέλουμε να αποδείξουμε:
 - ?- `lives(pluto, athina)`.

ΔΙΑΔΙΚΑΣΙΑ ΑΠΟΔΕΙΞΗΣ (2)



ΟΠΙΣΘΟΔΡΟΜΗΣΗ (BACKTRACKING) (1)

Θέλουμε να αποδείξουμε: $?- \text{lives}(W, \text{athina})$



CUT (!)

- ▶ Τίθεται σαν στόχος (στοιχείο) σε κανόνες. Όταν το συναντήσει η διαδικασία απόδειξης, ικανοποιείται άμεσα.
- ▶ Σαν αποτέλεσμα έχει την καλύτερη απόδοση (και χωρικά και χρονικά).
- ▶ Περιπτώσεις χρήσης:
 - ▶ Για να σταματήσει η διαδικασία σε συγκεκριμένο κανόνα
 - ▶ Για να σταματήσει την προσπάθεια ικανοποίησης συγκεκριμένου στόχου
 - ▶ Για να αποτρέψει την οπισθοδρόμηση, δηλ. την εύρεση εναλλακτικών λύσεων

CUT-ΠΕΡΙΠΤΩΣΗ 1

► Σταμάτημα σε Κανόνα

► Π.χ. αναδρομικοί ορισμοί, όπως το N!:

$n_parag(1,1) :- !.$

$n_parag(N, P) :- NEXT \text{ is } N-1,$

$n_parag(NEXT, REST),$

$P \text{ is } REST * N.$



CUT-ΠΕΡΙΠΤΩΣΗ 2

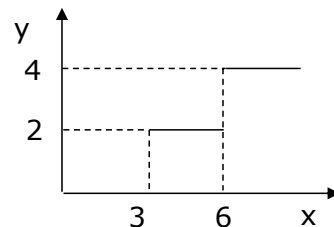
► Σταμάτημα Οπισθοδρόμησης

► Π.χ. Αναπαράσταση βηματικής συνάρτησης

$f(X,0) :- X < 3, !.$

$f(X, 2) :- 3 \leq X, X < 6, !.$

$f(X,4) :- 6 \leq X.$



ΔΥΝΑΜΙΚΗ ΔΙΑΧΕΙΡΙΣΗ ΒΑΣΗΣ ΓΝΩΣΗΣ

- ▶ Η Prolog μπορεί να κάνει δυναμική διαχείριση γεγονότων και κανόνων, μέσω δύο ειδικών κατηγορημάτων (assert και retract).
- ▶ Με το **assert** εισάγει κάποια γεγονότα ή κάποιους κανόνες στη μνήμη, ενώ με το **retract** αφαιρεί.

assert(<πρόταση Prolog>).

retract(<πρόταση Prolog>).

:- dynamic p/1.

```
?- assertz(p(a)).
```

```
?- p(X).
```

```
X = a.
```

```
?- retractall(p(_)).
```

```
?- p(X).
```

```
false.
```

