

Εισαγωγή στη Γλώσσα Prolog



Tamara de Lempicka, *Jeune Fille Vert*, 1929

Κωστής Σαγώνας <kostis@cs.ntua.gr>

Περιεχόμενα

- Όροι (terms)
- Χρήση ενός συστήματος Prolog
- Κανόνες (rules)
- Οι δύο όψεις της Prolog
- Τελεστές (operators)
- Λίστες (lists)
- Άρνηση ως αποτυχία (negation as failure)
- Για τι στο καλό είναι καλή η Prolog;

Εισαγωγή στη γλώσσα Prolog

2

Όροι (Terms)

- Κάθε τι στην Prolog κατασκευάζεται από όρους:
 - Τα προγράμματα σε Prolog
 - Τα δεδομένα που χειρίζονται τα προγράμματα σε Prolog
- Τρεις τύποι όρων:
 - Μεταβλητές
 - Σταθερές: ακέραιοι (integers), πραγματικοί αριθμοί (reals), και άτομα (atoms)
 - Σύνθετοι όροι: λίστες (lists) και δομημένα δεδομένα (structures)

Εισαγωγή στη γλώσσα Prolog

3

Άτομα (Atoms)

- Ένα άτομο είναι μια σταθερά που αποτελείται από αλφαριθμητικούς χαρακτήρες:
 - `an, atom, has, many_characters`
- Εάν ο αρχικός χαρακτήρας δεν είναι ένα μικρό γράμμα ή υπάρχουν «περίεργοι» χαρακτήρες, το άτομο πρέπει να βρίσκεται μέσα σε αποστρόφους
 - `'Kostis', '40 παλικάρια', '$^% (@ (&$#*&$'`
- Κάθε άτομο έχει μια τιμή (τους χαρακτήρες του) και είναι ίσο μόνο με άλλα άτομα που έχουν ακριβώς τους ίδιους χαρακτήρες ως τιμή
- Σκεφτείτε τα άτομα κάτι σαν strings σε άλλες γλώσσες

Εισαγωγή στη γλώσσα Prolog

4

Μεταβλητές

- Μεταβλητή είναι κάθε τι που αρχίζει από ένα κεφαλαίο γράμμα ή underscore, και ακολουθείται από γράμματα, αριθμούς ή underscores:
 - `_ , X, Variable, Kostis, _123`
- Οι μεταβλητές έχουν δύο καταστάσεις:
 - Μπορεί να μην έχουν ακόμα τιμή (unbound)
 - Μπορεί να έχουν κάποια τιμή, δηλαδή να είναι δεμένες (bound)
- Όταν όμως μια μεταβλητή είναι σε κατάσταση «με τιμή», η τιμή αυτή δεν αλλάζει

Εισαγωγή στη γλώσσα Prolog

5

Σύνθετοι Όροι (Compound Terms)

- Αρχίζουν με ένα άτομο και ακολουθούνται από μια ακολουθία όρων που διαχωρίζονται με κόμματα και περικλείονται από παρενθέσεις:
 - `university('Ε.Μ.Π.')`,
 - `'+'(2,3.14)`,
 - `couple(adam,eve)`,
 - `tree(V,tree(42,Y,Z),R)`
- Οι λίστες είναι και αυτές σύνθετοι όροι αλλά όπως θα δούμε έχουν ειδική σύνταξη
- Σκεφτείτε τους σύνθετους όρους σαν δομές δεδομένων

Εισαγωγή στη γλώσσα Prolog

6

Σύνταξη των Όρων

```
<term> ::= <constant> | <variable> | <compound-term>
<constant> ::= <integer> | <real number> | <atom>
<compound-term> ::= <atom> ( <termlist> )
<termlist> ::= <term> | <term> , <termlist>
```

- Όλα τα προγράμματα και τα δεδομένα της Prolog κατασκευάζονται από τέτοιους όρους
- Σε λίγο θα δούμε ότι, για παράδειγμα, ο σύνθετος όρος '+' (1,2) συνήθως γράφεται ως 1+2
- Αλλά αυτή είναι απλά μια συντομογραφία του ίδιου όρου

Εισαγωγή στη γλώσσα Prolog

7

Ενοποίηση (Unification)

- Ταίριασμα προτύπων σε όρους της Prolog
- Δύο όροι *ενοποιούνται* εάν υπάρχει κάποιος τρόπος να βρεθούν τιμές για τις μεταβλητές τους που να τους κάνουν ακριβώς τους ίδιους
- Για παράδειγμα, οι όροι `couple(adam, eve)` και `couple(Who, WithWhom)` ενοποιούνται αν η μεταβλητή `Who` πάρει την τιμή `adam` και η μεταβλητή `WithWhom` πάρει την τιμή `eve`
- (Περισσότερες λεπτομέρειες στο επόμενο μάθημα...)

Εισαγωγή στη γλώσσα Prolog

8

Η Βάση Δεδομένων της Prolog

- Η υλοποίηση της Prolog διατηρεί (για αναπαράσταση) των προγραμμάτων που έχουν φορτωθεί στο σύστημα
- Ο κώδικας συμπεριφέρεται σαν μια βάση δεδομένων
- Ένα πρόγραμμα Prolog είναι απλώς ένα σύνολο από δεδομένα για αυτή τη βάση
- Το απλούστερο από τα δεδομένα αυτής της βάσης είναι ένα *γεγονός (fact)*: συντακτικά είναι ένας όρος που ακολουθείται από μια τελεία

Εισαγωγή στη γλώσσα Prolog

9

Παράδειγμα κατηγορήματος με γεγονότα

```
parent(kim, holly).
parent(margaret, kim).
parent(margaret, kent).
parent(ester, margaret).
parent(herbert, margaret).
parent(herbert, jean).
```

- Ένα πρόγραμμα Prolog με έξι γεγονότα
- Το οποίο ορίζει ένα *κατηγορημα (predicate)* `parent/2`
- Αποτελούμενο από γεγονότα που έχουν την «προφανή» ερμηνεία: ο `kim` είναι ο γονιός της `holly`, η `margaret` είναι ο γονιός του `kim`, `kok`.

Εισαγωγή στη γλώσσα Prolog

10

Ο Διερμηνέας της Prolog

```
% swipl
Welcome to SWI-Prolog (Multi-threaded, Version 5.2.13)
Copyright (c) 1990-2003 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. ...
Please visit http://www.swi-prolog.org for details

For help, use ?- help(Topic). or ?- apropos(Word).
?-
```

- Το `?-` είναι η προτροπή της Prolog για μια ερώτηση
- Ο διερμηνέας είναι διαδραστικός: δέχεται μια ερώτηση, τυπώνει τα αποτελέσματά της, και επαναλαμβάνει αυτή τη διαδικασία

Εισαγωγή στη γλώσσα Prolog

11

Το κατηγορημα `consult/1`

```
?- consult(relations).
% relations compiled 0.00 sec, XX bytes
Yes
?-
```

- Κατηγορημα του συστήματος με το οποίο φορτώνουμε ένα πρόγραμμα από ένα αρχείο στη βάση της Prolog
- Το αρχείο `relations` (ή `relations.pl`) περιλαμβάνει τα γεγονότα του κατηγορήματος `parent/2`

```
?- [relations].
% relations compiled 0.00 sec, XX bytes
Yes
```

Εισαγωγή στη γλώσσα Prolog

12

Απλές ερωτήσεις

```
?- parent(margaret, kent) .
Yes
?- parent(fred, pebbles) .
No
?-
```

- Μια ερώτηση προκαλεί την Prolog να αποδείξει κάτι
- Η απάντηση των απλών ερωτήσεων είναι ένα **Yes** ή **No**

(Κάποιες ερωτήσεις, όπως η ερώτηση στο κατηγορημα `consult/1`, εκτελούνται μόνο για τις παρενέργειές τους)

Εισαγωγή στη γλώσσα Prolog

13

Η τελεία

```
?- parent(margaret, kent)
|
|
Yes
?-
```

- Οι ερωτήσεις μπορούν να εκτείνονται σε πολλές γραμμές
- Εάν ξεχάσετε τη τελεία (το χαρακτήρα τέλους της ερώτησης), ο διερμηνέας της Prolog θα σας ζητήσει περισσότερη είσοδο με το χαρακτήρα `|`

Εισαγωγή στη γλώσσα Prolog

14

Ερωτήσεις με μεταβλητές

- Οι ερωτήσεις μπορεί να είναι οποιοσδήποτε όρος Prolog, συμπεριλαμβανομένων όρων με μεταβλητές
- Ο διερμηνέας της Prolog επιστρέφει ως απάντηση τις τιμές των μεταβλητών για τις οποίες η ερώτηση μπορεί να απαντηθεί

```
?- parent(P, jean) .
P = herbert
Yes
?- parent(P, esther) .
No
```

*Στο σημείο αυτό, ο διερμηνέας περιμένει κάποια είσοδο. Δίνουμε **enter** για να σταματήσει η εκτέλεση της ερώτησης.*

Εισαγωγή στη γλώσσα Prolog

15

Ευελιξία στις ερωτήσεις

- Οι μεταβλητές μπορούν φυσικά να βρίσκονται σε οποιαδήποτε θέση κάποιας ερώτησης:
 - `parent(Parent, jean)`
 - `parent(esther, Child)`
 - `parent(Parent, Child)`
 - `parent(Person, Person)`

Εισαγωγή στη γλώσσα Prolog

16

Συζεύξεις (Conjunctions)

```
?- parent(margaret, X) , parent(X, holly) .
X = kim
Yes
```

- Μια σύζευξη ερωτήσεων είναι μια ακολουθία από όρους που διαχωρίζονται από κόμματα
- Ο διερμηνέας της Prolog προσπαθεί να τις αποδείξει όλες (χρησιμοποιώντας το ίδιο σύνολο «δεσιμάτων» για τις μεταβλητές της ερώτησης)

Εισαγωγή στη γλώσσα Prolog

17

Πολλαπλές λύσεις

```
?- parent(margaret, Child) .
Child = kim ;
Child = kent ;
No
```

- Μπορεί να υπάρχουν περισσότεροι από ένας τρόποι για να απαντηθεί μια ερώτηση
- Με το να πληκτρολογήσουμε `;` αντί για `enter`, προτρέπουμε το σύστημα Prolog να βρει και να μας δώσει περισσότερες απαντήσεις

Εισαγωγή στη γλώσσα Prolog

18

```

?- parent (Parent, kim) , parent (Grandparent, Parent) .

Parent = margaret
Grandparent = esther ;

Parent = margaret
Grandparent = herbert ;

No
?- parent (esther, Child) ,
| parent (Child, Grandchild) ,
| parent (Grandchild, GreatGrandchild) .

Child = margaret
Grandchild = kim
GreatGrandchild = holly

Yes
    
```

Εισαγωγή στη γλώσσα Prolog 19

Ανάγκη για κανόνες

- Στο προηγούμενο παράδειγμα χρησιμοποιήσαμε μια μακροσκελή ερώτηση για τα δισέγγονα της *esther*
- Θα ήταν καλύτερο εάν μπορούσαμε να γράψουμε κατ' ευθείαν την ερώτηση:


```
greatgrandparent (esther, GGC)
```
- Αλλά δε θέλουμε να εισαγάγουμε ξεχωριστά γεγονότα αυτού του κατηγορήματος στη βάση των δεδομένων διότι η σχέση *greatgrandparent/2* προκύπτει (και πρέπει να προκύπτει) από την ήδη ορισμένη σχέση *parent/2*

Εισαγωγή στη γλώσσα Prolog 20

Ένα παράδειγμα κανόνα

```

      κεφαλή
      ↓
greatgrandparent (GGP, GGC) :-
  parent (GGP, GP) ,
  parent (GP, P) ,
  parent (P, GGC) .
    
```

← συνθήκες

- Ένας κανόνας καθορίζει πώς αποδεικνύουμε κάτι: για να αποδείξουμε την κεφαλή του κανόνα πρέπει να αποδείξουμε τις συνθήκες του
- Για να αποδείξουμε *greatgrandparent (GGP, GGC)*, πρέπει να βρούμε τιμές *GP* και *P* για τις οποίες μπορούμε να αποδείξουμε ότι *parent (GGP, GP)*, μετά ότι *parent (GP, P)* και τελικά ότι *parent (P, GGC)*

Εισαγωγή στη γλώσσα Prolog 21

Πρόγραμμα με χρήση του κανόνα

```

parent (kim, holly) .
parent (margaret, kim) .
parent (margaret, kent) .
parent (esther, margaret) .
parent (herbert, margaret) .
parent (herbert, jean) .

greatgrandparent (GGP, GGC) :-
  parent (GGP, GP) ,
  parent (GP, P) , parent (P, GGC) .
    
```

- Ένα πρόγραμμα αποτελείται από μια ακολουθία από *προτάσεις (clauses)*
- Μια πρόταση είναι είτε ένα γεγονός ή ένας κανόνας

Εισαγωγή στη γλώσσα Prolog 22

Παράδειγμα

```

?- greatgrandparent (esther, GreatGrandchild) .

GreatGrandchild = holly ;

No
    
```

- Για την απάντηση της παραπάνω ερώτησης, εσωτερικά εξετάζονται διάφοροι ενδιάμεσοι *στόχοι (goals)*:
 - Ο πρώτος (υπο)στόχος είναι η αρχική ερώτηση
 - Ο επόμενος είναι ότι απομένει προς απόδειξη μετά την απόδειξη του πρώτου (υπο)στόχου με χρήση κάποιας πρότασης του προγράμματος (στη συγκεκριμένη περίπτωση, με χρήση του κανόνα για το κατηγορήμα *greatgrandparent/2*)
 - Κοκ., μέχρι να μη μείνει τίποτα προς απόδειξη

Εισαγωγή στη γλώσσα Prolog 23

1. *parent (kim, holly) .*
2. *parent (margaret, kim) .*
3. *parent (margaret, kent) .*
4. *parent (esther, margaret) .*
5. *parent (herbert, margaret) .*
6. *parent (herbert, jean) .*
7. *greatgrandparent (GGP, GGC) :- parent (GGP, GP) , parent (GP, P) , parent (P, GGC) .*

```

greatgrandparent (esther, GreatGrandchild)
  ↓ Clause 7, binding GGP to esther and GGC to GreatGrandChild
parent (esther, GP) , parent (GP, P) , parent (P, GreatGrandchild)
  ↓ Clause 4, binding GP to margaret
parent (margaret, P) , parent (P, GreatGrandchild)
  ↓ Clause 2, binding P to kim
parent (kim, GreatGrandchild)
  ↓ Clause 1, binding GreatGrandchild to holly
    
```

Εισαγωγή στη γλώσσα Prolog 24

Κανόνες με χρήση άλλων κανόνων

```
grandparent(GP, GC) :-
    parent(GP, P), parent(P, GC).

greatgrandparent(GGP, GGC) :-
    grandparent(GGP, P), parent(P, GGC).
```

- Η ίδια σχέση με την προηγούμενη, ορισμένη σε στρώσεις
- Παρατηρήστε ότι και οι δύο κανόνες χρησιμοποιούν μεταβλητές με όνομα P
- Η εμβέλεια κάποιας μεταβλητής είναι μόνο ο κανόνας που την περιέχει
- Οι δύο μεταβλητές με όνομα P είναι ανεξάρτητες

Εισαγωγή στη γλώσσα Prolog

25

Αναδρομικοί κανόνες

```
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :-
    parent(Z, Y),
    ancestor(X, Z).
```

- Ένας X είναι πρόγονος κάποιου Y:
 - Εάν ο X είναι γονιός του Y, ή
 - Εάν υπάρχει κάποιος Z τέτοιος ώστε ο Z να είναι ο γονιός του Y, και ο X να είναι πρόγονος του Z
- Η Prolog δοκιμάζει τους κανόνες με τη σειρά που αυτοί εμφανίζονται, οπότε είναι καλό (και πολλές φορές αναγκαίο) τα γεγονότα και οι μη αναδρομικοί κανόνες να εμφανίζονται πριν από τους αναδρομικούς

Εισαγωγή στη γλώσσα Prolog

26

```
?- ancestor(jean, jean).
```

No

```
?- ancestor(kim, holly).
```

Yes

```
?- ancestor(A, holly).
```

A = kim ;

A = margaret ;

A = esther ;

A = herbert ;

No

Εισαγωγή στη γλώσσα Prolog

27

Ο πυρήνας της σύνταξης της Prolog

- Ολόκληρη η σύνταξη της Prolog:

```
<clause> ::= <fact> | <rule>
<fact> ::= <term> .
<rule> ::= <term> :- <termlist> .
<termlist> ::= <term> | <term> , <termlist>
```

- Εάν εξαιρεθεί η δυνατότητα ορισμού νέων τελεστών από το χρήστη, συντακτικά η Prolog είναι πολύ απλή γλώσσα
- Αυτό ήταν όλο!

Εισαγωγή στη γλώσσα Prolog

28

Η διαδικαστική όψη της Prolog

```
greatgrandparent(GGP, GGC) :-
    parent(GGP, GP),
    parent(GP, P),
    parent(P, GGC).
```

- Ένας κανόνας λέει πως μπορούμε να αποδείξουμε κάτι:
 - Για να αποδείξουμε `greatgrandparent(GGP, GGC)`, πρέπει να βρούμε τιμές GP και P για τις οποίες μπορούμε να αποδείξουμε ότι `parent(GGP, GP)`, μετά ότι `parent(GP, P)` και τελικά ότι `parent(P, GGC)`
- Ένα πρόγραμμα Prolog προσδιορίζει διαδικασίες εύρεσης απαντήσεων σε ερωτήσεις

Εισαγωγή στη γλώσσα Prolog

29

Η δηλωτική όψη της Prolog

- Ένας κανόνας είναι μια λογική δήλωση:
 - Για όλες τις τιμές των GGP, GP, P, και GGC, αν ισχύει ότι `parent(GGP, GP)` και `parent(GP, P)` και `parent(P, GGC)`, τότε `greatgrandparent(GGP, GGC)`
- Είναι απλώς μια φόρμουλα - δε λέει πώς να γίνει κάτι αλλά απλώς ορίζει τις προϋποθέσεις κάτω από τις οποίες μια πρόταση είναι αληθής:

$$\forall GGP, GP, P, GGC. \text{parent}(GGP, GP) \wedge \text{parent}(GP, P) \wedge \text{parent}(P, GGC) \Rightarrow \text{greatgrandparent}(GGP, GGC)$$

Εισαγωγή στη γλώσσα Prolog

30

Δηλωτικές Γλώσσες Προγραμματισμού

- Κάθε κομμάτι του προγράμματος αντιστοιχεί σε μια απλή μαθηματική έννοια:
 - Προτάσεις της Prolog - σε φόρμουλες τις λογικής πρώτης τάξης (first-order logic)
 - Οι ορισμοί συναρτήσεων στην ML - σε μαθηματικές συναρτήσεις
- Πολλές φορές ο όρος *δηλωτικές (declarative) γλώσσες προγραμματισμού* χρησιμοποιείται για να δείξει την αντιδιαστολή των γλωσσών αυτών σε σχέση με τις *προστακτικές (imperative) γλώσσες*
- Ως δηλωτικές γλώσσες συνήθως θεωρούμε τις λογικές και τις συναρτησιακές

Εισαγωγή στη γλώσσα Prolog

31

Πλεονεκτήματα των Δηλωτικών Γλωσσών

- Τα προγράμματα σε προστακτικές γλώσσες πολλές φορές έχουν προβλήματα λόγω παρενεργειών και αλληλεξαρτήσεων
- Επειδή οι δηλωτικές γλώσσες έχουν σχετικά απλή και «καθαρή» σημασιολογία, αποφεύγονται κάποιοι είδους λάθη και η ανάπτυξη και η συντήρηση προγραμμάτων γίνεται πιο εύκολη
- Ο προγραμματισμός είναι ανώτερου επιπέδου και πιο αυτόματος: ο προγραμματιστής απλώς περιγράφει την προδιαγραφή του προβλήματος και κάποια κομμάτια της εκτέλεσής του παρέχονται από το σύστημα υλοποίησης

Εισαγωγή στη γλώσσα Prolog

32

Η Prolog έχει και τα δύο χαρακτηριστικά

- Έχει δηλωτική σημασιολογία
 - Ένα πρόγραμμα Prolog έχει ένα λογικό περιεχόμενο
- Έχει διαδικαστική σημασιολογία
 - Ένα πρόγραμμα Prolog έχει και καθαρά διαδικαστικά χαρακτηριστικά: τη σειρά με την οποία εμφανίζονται οι κανόνες, τη σειρά εμφάνισης των υποστόχων σ' ένα κανόνα, κατηγορήματα με παρενέργειες (π.χ. !/0), κ.λπ.
- Όμως είναι σημαντικό ο προγραμματιστής να είναι ενήμερος και να παίρνει υπόψη του κατά τον προγραμματισμό και τις δύο όψεις της γλώσσας

Εισαγωγή στη γλώσσα Prolog

33

Τελεστές της Prolog

- Η Prolog έχει κάποιους προκαθορισμένους τελεστές (και επιτρέπει τον ορισμό νέων τελεστών από το χρήστη)
- Ένας τελεστής είναι απλώς ένα κατηγορήμα για το οποίο είναι δυνατή η χρήση κάποιας ειδικής συντακτικής συντομογραφίας στις χρήσεις του

Εισαγωγή στη γλώσσα Prolog

34

Το κατηγορήμα =/2

- Ο στόχος (X, Y) επιτυγχάνει αν και μόνον αν οι όροι X και Y μπορούν να ενοποιηθούν:

```
?- =(parent(adam, seth), parent(adam, X)).
X = seth ;
No
```

- Επειδή το άτομο '=' είναι ορισμένο ως δυαδικός τελεστής, συνήθως γράφουμε το παραπάνω ως:

```
?- parent(adam, seth) = parent(adam, X).
X = seth ;
No
```

Εισαγωγή στη γλώσσα Prolog

35

Αριθμητικοί Τελεστές

- Τα άτομα '+', '-', '*', και '/' είναι ορισμένα ως τελεστές, με τη συνήθη προτεραιότητα και προσεταιριστικότητα

```
?- X = +(1, *(2, 3)).
X = 1+2*3 ;
No
?- X = 1+2*3.
X = 1+2*3 ;
No
```

Η Prolog μας επιτρέπει να χρησιμοποιήσουμε σύνταξη τελεστών στην είσοδο, και χρησιμοποιεί αυτή τη μορφή στην έξοδο των όρων, αλλά ο υποκείμενος όρος είναι ο $+(1, *(2, 3))$

Εισαγωγή στη γλώσσα Prolog

36

Μη αποτιμημένοι όροι

```
?- +(X,Y) = 1+2*3.
X = 1
Y = 2*3 ;
No
?- 7 = 1+2*3.
No
```

- Μετά την ενοποίηση, ο όρος παραμένει + (1 , * (2 , 3))
- Με άλλα λόγια, ο όρος είναι μη αποτιμημένος

(Σε επόμενη διάλεξη θα δούμε ότι υπάρχει τρόπος στην Prolog να αποτιμήσουμε τέτοιους όρους)

Λίστες στην Prolog

- Συντακτικά είναι περίπου σα τις λίστες στην ML
- Το άτομο [] αναπαριστά την κενή λίστα
- Το κατηγορημα ' . ' /2 αντιστοιχεί στον τελεστή :: της ML

Έκφραση στην ML	Όρος στην Prolog
[]	[]
1 :: []	. (1, [])
1 :: 2 :: 3 :: []	. (1, . (2, . (3, [])))
Δεν υπάρχει αντίστοιχο.	. (1, . (parent(X,Y), []))

Οι αναπαραστές των λιστών στην Prolog

Λίστα	Όρος στον οποίο αντιστοιχεί
[]	[]
[1]	. (1, [])
[1, 2, 3]	. (1, . (2, . (3, [])))
[1, parent(X, Y)]	. (1, . (parent(X, Y), []))

- Η μορφή που συνήθως χρησιμοποιείται είναι η αριστερή
- Αλλά είναι απλώς μια συντομογραφία της μορφής που φαίνεται δεξιά στον παραπάνω πίνακα
- Η Prolog τυπώνει τις λίστες με χρήση της μορφής στο αριστερό μέρος του πίνακα

Παράδειγμα

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

Αναπαραστές λιστών με ουρά

Λίστα	Όρος στον οποίο αντιστοιχεί
[1 X]	. (1, X)
[1, 2 X]	. (1, . (2, X))
[1, 2 [3, 4]]	Το ίδιο με [1, 2, 3, 4]

- Πολλές φορές το τέλος της λίστας είναι το σύμβολο | ακολουθούμενο από την ουρά της λίστας
- Χρησιμοποιείται σε πρότυπα: ο όρος [1, 2|X] ενοποιείται με κάθε λίστα που αρχίζει με 1, 2 και έχει κάποια λίστα X ως ουρά

```
?- [1, 2|X] = [1, 2, 3, 4, 5] .
X = [3, 4, 5] ;
No
```

Το κατηγορημα append/3

- Το προκαθορισμένο κατηγορημα `append(X, Y, Z)` επιτυγχάνει αν και μόνο αν η λίστα Z είναι το αποτέλεσμα της συνένωσης της λίστας Y στο τέλος της λίστας X

```
?- append([1, 2], [3, 4], Z) .
Z = [1, 2, 3, 4] ;
No
```

Όχι απλώς μια συνάρτηση

```
?- append(X, [3,4], [1,2,3,4]).
X = [1, 2] ;
No
```

- Το κατηγορήμα `append` μπορεί να χρησιμοποιηθεί με κάθε όρο της Prolog (με άλλα λόγια, με μεταβλητές σε κάθε όρισμα)

Εισαγωγή στη γλώσσα Prolog

43

Όχι απλώς μια συνάρτηση

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

Εισαγωγή στη γλώσσα Prolog

44

Η υλοποίηση του κατηγορήματος `append/3`

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

Εισαγωγή στη γλώσσα Prolog

45

Άλλα κατηγορήματα για λίστες

Κατηγορήμα	Περιγραφή
<code>member(X, Y)</code>	Provable if the list <code>Y</code> contains the element <code>X</code> .
<code>select(X, Y, Z)</code>	Provable if the list <code>Y</code> contains the element <code>X</code> , and <code>Z</code> is the same as <code>Y</code> but with one instance of <code>X</code> removed.
<code>nth0(X, Y, Z)</code>	Provable if <code>X</code> is an integer, <code>Y</code> is a list, and <code>Z</code> is the <code>X</code> th element of <code>Y</code> , counting from 0.
<code>length(X, Y)</code>	Provable if <code>X</code> is a list of length <code>Y</code> .

- Όλα ευέλικτα, σαν το `append/3`
- Οι ερωτήσεις μπορούν να περιέχουν μεταβλητές σε οποιαδήποτε όρισμα

Εισαγωγή στη γλώσσα Prolog

46

Χρήση του κατηγορήματος `select/3`

```
?- select(2, [1,2,3], Z).
Z = [1, 3] ;
No
?- select(2, Y, [1,3]).
Y = [2, 1, 3] ;
Y = [1, 2, 3] ;
Y = [1, 3, 2] ;
No
```

Εισαγωγή στη γλώσσα Prolog

47

Το κατηγορήμα `reverse/2`

- `reverse(X, Y)` είναι αληθές εάν ο όρος `Y` ενοποιείται με την αναστροφή της λίστας `X`

```
?- reverse([1,2,3,4], Y).
Y = [4, 3, 2, 1] ;
No
```

Εισαγωγή στη γλώσσα Prolog

48

Η υλοποίηση του κατηγορήματος `reverse/2`

```
reverse([], []).
reverse([Head|Tail], Reversed) :-
    reverse(Tail, RevTail),
    append(RevTail, [Head], Reversed).
```

- Δεν είναι ένας αποδοτικός τρόπος να αντιστρέψουμε μια λίστα
- Υπάρχουν πιο αποδοτικοί τρόποι αντιστροφής λιστών

Εισαγωγή στη γλώσσα Prolog

49

Όταν δεν πάνε όλα καλά...

```
?- reverse(L, [1,2,3,4]).
L = [4, 3, 2, 1] ;
Action (h for help) ? a
% Execution Aborted
?-
```

- Ζητήσαμε μια ακόμα λύση και βρεθήκαμε σ' έναν ατέρμονο βρόχο
- Παύουμε την εκτέλεση με `control-C`, και στη συνέχεια πατάμε `a` για το οριστικό σταμάτημά της
- Το `reverse/2` δεν είναι τόσο ευέλικτο όσο το `append/3`

Εισαγωγή στη γλώσσα Prolog

50

Αντιστρέψιμα και μη αντιστρέψιμα κατηγορήματα

- Σ' έναν ιδεατό κόσμο, τα κατηγορήματα είναι όλα τόσο ευέλικτα όσο το `append/3`
- Είναι πιο δηλωτικά και με λιγότερους περιορισμούς στη χρήση τους
- Αλλά πολλές φορές μη ευέλικτες υλοποιήσεις είναι προτιμότερες, για λόγους απόδοσης ή/και απλότητας
- Ένα παράδειγμα του τι σημαίνει η παραπάνω φράση μας δείχνει το κατηγορήμα `sort/2`...

Εισαγωγή στη γλώσσα Prolog

51

Το κατηγορήμα `sort/2`

```
?- sort([2,3,1,4], X).
X = [1, 2, 3, 4] ;
No
?- sort(X, [1,2,3,4]).
ERROR: Arguments are not sufficiently instantiated
```

- Ένα αντιστρέψιμο κατηγορήμα `sort/2` θα έπρεπε να μπορεί να «αντι-ταξινομήσει» μια ταξινομημένη λίστα – με άλλα λόγια να επιστρέφει όλες τις αναδιατάξεις των στοιχείων της
- Θέλουμε κάτι σαν το παραπάνω;

Εισαγωγή στη γλώσσα Prolog

52

Ανώνυμες μεταβλητές

- Η μεταβλητή `_` είναι μια μεταβλητή χωρίς όνομα
- Κάθε εμφάνισή της είναι ανεξάρτητη από όλες τις υπόλοιπες
- Με άλλα λόγια, συμπεριφέρεται σαν τις μεταβλητές `_` της ML: ενοποιείται με κάθε όρο

Εισαγωγή στη γλώσσα Prolog

53

Το κατηγορήμα `member/2`

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

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

Εισαγωγή στη γλώσσα Prolog

54

Προσοχή στις προειδοποιήσεις!

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

Warning: Singleton variables [L1,L1]

- Μην αγνοείτε τα προειδοποιητικά μηνύματα ότι μεταβλητές εμφανίζονται μόνο μια φορά
- Όπως και στην ML, είναι κακό στυλ προγραμματισμού να εισάγουμε μεταβλητές που δεν χρησιμοποιούμε - ειδικά επειδή είναι το μόνο μήνυμα που το σύστημα θα πάρουμε

Εισαγωγή στη γλώσσα Prolog

55

Το κατηγορημα \+/1

```
?- member(1, [1,2,3]).
Yes
?- \+ member(4, [1,2,3]).
Yes
```

- Σε απλές εφαρμογές, συνήθως λειτουργεί σαν την άρνηση της λογικής
- Αλλά έχει και μια σημαντική διαδικαστική πλευρά...

Εισαγωγή στη γλώσσα Prolog

56

Άρνηση ως αποτυχία (negation as failure)

- Για να αποδείξει το \+ X, η Prolog προσπαθεί να αποδείξει το X
- \+ X επιτυγχάνει εάν ο στόχος X αποτυγχάνει
- Οι δύο όψεις ξανά:
 - Δηλωτική: \+ X = ~X
 - Διαδικαστική: \+ X επιτυγχάνει εάν το X αποτυγχάνει, και αποτυγχάνει εάν το X επιτυγχάνει, και δεν τερματίζει εάν το X δεν τερματίζει

Εισαγωγή στη γλώσσα Prolog

57

Παράδειγμα

```
sibling(X, Y) :-
    \+ (X = Y),
    parent(P, X),
    parent(P, Y).
```

```
?- sibling(kim, kent).
Yes
?- sibling(kim, kim).
No
?- sibling(X, Y).
No
```

```
sibling(X, Y) :-
    parent(P, X),
    parent(P, Y),
    \+ (X = Y).
```

```
?- sibling(X, Y).
X = kim
Y = kent ;
X = kent
Y = kim ;
X = margaret
Y = jean ;
X = jean
Y = margaret ;
No
```

Εισαγωγή στη γλώσσα Prolog

58

Ένα μεγαλύτερο παράδειγμα σε Prolog



Εισαγωγή στη γλώσσα Prolog

59

Μια κλασική σπαζοκεφαλιά

- Ένας άνθρωπος (man) ταξιδεύει μαζί μ' ένα λύκο (wolf), μια κατσίκα (goat) και ένα λάχανο (cabbage)
- Θέλει να διασχίσει ένα ποτάμι από τη δυτική όχθη στην ανατολική
- Στην αρχική όχθη υπάρχει μια βάρκα, η οποία όμως χωράει τον άνθρωπο και μόνο ένα από τα υπάρχοντά του
- Ο λύκος τρώει την κατσίκα εάν βρεθεί μόνος μαζί της
- Η κατσίκα τρώει το λάχανο εάν βρεθεί μόνη μαζί του
- Με ποιο τρόπο μπορεί ο άνθρωπος να περάσει στην απέναντι όχθη με όλα τα υπάρχοντά του;

Εισαγωγή στη γλώσσα Prolog

60

Δομές δεδομένων της λύσης

- Θα αναπαραστήσουμε τους σχηματισμούς του συστήματος με μια λίστα που δείχνει την όχθη στην οποία είναι το κάθε τι με την εξής σειρά:
man, wolf, goat, cabbage
- Ο αρχικός σχηματισμός: [w, w, w, w]
- Εάν για παράδειγμα ο άνθρωπος με το λύκο μπουν στη βάρκα, ο νέος σχηματισμός είναι ο [e, e, w, w] - αλλά τότε η κατσίκα θα φάει το λάχανο, οπότε ο παραπάνω σχηματισμός δεν είναι επιτρεπτός ως ενδιάμεσος
- Ο επιθυμητός τελικός σχηματισμός: [e, e, e, e]

Εισαγωγή στη γλώσσα Prolog

61

Κινήσεις

- Σε κάθε κίνηση, ο άνθρωπος διασχίζει το ποτάμι με το πολύ ένα από τα υπάρχοντά του
- Θα αναπαραστήσουμε τις τέσσερις επιτρεπτές κινήσεις με τέσσερα άτομα: **wolf, goat, cabbage, nothing**
- (Όπου, **nothing** σημαίνει ότι ο άνθρωπος διασχίζει το ποτάμι μόνος του μέσα στη βάρκα)

Εισαγωγή στη γλώσσα Prolog

62

Οι κινήσεις τροποποιούν τον σχηματισμό

- Κάθε κίνηση μεταμορφώνει ένα σχηματισμό σ' έναν άλλο
- Στην Prolog, θα προγραμματίσουμε τη λύση με ένα κατηγορήμα: **move(Config, Move, NextConfig)**
 - **Config** είναι ένας σχηματισμός (π.χ. [w, w, w, w])
 - **Move** είναι μια κίνηση (π.χ. wolf)
 - **NextConfig** είναι ο σχηματισμός-αποτέλεσμα (στο συγκεκριμένο παράδειγμα ο [e, e, w, w])

Εισαγωγή στη γλώσσα Prolog

63

Το κατηγορήμα move/2

```
change(e, w).
change(w, e).

move([X,X,Goat,Cabbage], wolf, [Y,Y,Goat,Cabbage]) :-
    change(X, Y).
move([X,Wolf,X,Cabbage], goat, [Y,Wolf,Y,Cabbage]) :-
    change(X, Y).
move([X,Wolf,Goat,X], cabbage, [Y,Wolf,Goat,Y]) :-
    change(X, Y).
move([X,Wolf,Goat,C], nothing, [Y,Wolf,Goat,C]) :-
    change(X, Y).
```

Εισαγωγή στη γλώσσα Prolog

64

Ασφαλείς σχηματισμοί

- Ένας σχηματισμός είναι ασφαλής εάν
 - Τουλάχιστον ένας από τους λύκο και κατσίκα είναι στην ίδια όχθη με τον άνθρωπο, και
 - Τουλάχιστον ένας από τους κατσίκα και λάχανο είναι στην ίδια όχθη με τον άνθρωπο

```
oneEq(X, X, _).
oneEq(X, _, X).

safe([Man, Wolf, Goat, Cabbage]) :-
    oneEq(Man, Goat, Wolf),
    oneEq(Man, Goat, Cabbage).
```

Εισαγωγή στη γλώσσα Prolog

65

Λύσεις

- Μια λύση αρχίζει από τον αρχικό σχηματισμό και δημιουργεί μια λίστα από κινήσεις που οδηγούν στο σχηματισμό [e, e, e, e], έτσι ώστε όλοι οι ενδιάμεσοι σχηματισμοί να είναι ασφαλείς

```
solution([e,e,e,e], []).
solution(Config, [Move|Rest]) :-
    move(Config, Move, NextConfig),
    safe(NextConfig),
    solution(NextConfig, Rest).
```

Εισαγωγή στη γλώσσα Prolog

66

Το κατηγορημα `length/2`

```
?- length(L, N) .
X = []
Y = 0 ;

X = [_]
Y = 1 ;

X = [_, _]
Y = 2 ;

X = [_, _, _]
Y = 3 ;

X = [_, _, _, _]
Y = 4
Yes
```

Εισαγωγή στη γλώσσα Prolog

67

Η Prolog βρίσκει τη λύση

```
?- length(L, N), solution([w,w,w], L) .
L = [goat, nothing, wolf, goat, cabbage, nothing, goat]
Yes
```

Σημείωση: χωρίς τη χρήση του `length/2`, η Prolog δε θα έβρισκε κάποια λύση (με το συγκεκριμένο πρόγραμμα). Θα μπερδευόταν ψάχνοντας για λύσεις της μορφής:
[goat, goat, goat, goat, goat...]

Εισαγωγή στη γλώσσα Prolog

68

Κάποια καλά χαρακτηριστικά της Prolog

- Το πρόγραμμά μας απλώς κατέγραψε την εκφώνηση του προβλήματος προς επίλυση
- Δεν προσδιορίσαμε κάποιο τρόπο αναζήτησης της λύσης του προβλήματος - η ίδια η Prolog έκανε την αναζήτηση
- Αυτού του είδους τα προβλήματα είναι ακριβώς τα προβλήματα που είναι κατάλληλα προς επίλυση σε γλώσσα Prolog
- Περισσότερα παραδείγματα σε επόμενες διαλέξεις...

Εισαγωγή στη γλώσσα Prolog

69