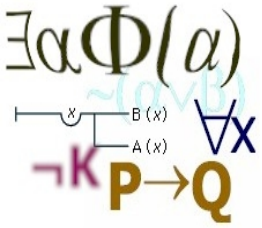
Πανεπιστήμιο Μακεδονίας
Τμ. Εφαρμοσμένης Πληροφορικής

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

Συλλογή Λύσεων

Κατηγορήματα Συλλογής Λύσεων

- Κατά την οπισθοδρόμηση η Prolog παράγει όλες τις λύσεις ενός κατηγορήματος.
- Ένα κατηγόρημα να συγκεντρώνει όλες τις λύσεις.
- Υπάρχουν τρία διαθέσιμα Prolog κατηγορήματα:
  - *findall/3*
  - *setof/3*
  - *bagof/3*

findall/3

- *findall(Var, Predicate, Solutions)*
  - Ενοποιεί τη λίστα *Solutions* με όλες τις λύσεις για την μεταβλητή *Var* που ικανοποιούν το κατηγόρημα *Predicate*.
  - Το Var μπορεί να είναι και όρος που περιέχει μεταβλητές (Term)
  - Το Predicate μπορεί να είναι και σύνθετος στόχος.
- Δεν αποτυγχάνει **ποτέ** – απλώς επιστρέφει την κενή λίστα.

Παράδειγμα

- *findall(X,male(X),List).*

  List = [petros,nick,adam]
- *findall(N,follows(nick,N),L).*

  L = [petros]
- *findall(X,follows(adam,X),L)*

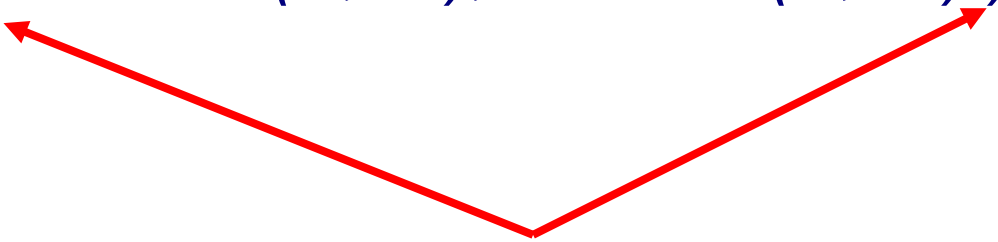
  L = []

```
male(petros).
male(nick).
male(adam).

follows(nick,petros).
follows(petros,nick).

friends(X,Y):-
  follows(X,Y),
  follows(Y,X).
```

Παράδειγμα

- Τομή δύο λιστών

```
intersect4(L1,L2,L3):-
    findall(X, ( member(X,L1),member(X,L2) ),L3).
```



Σύνθετοι στόχοι μέσα σε παρενθέσεις

setof/3

- *setof(Term, Predicate, Solutions)*
- Ενοποιεί τη λίστα *Solutions* με τιμές του *Term* που ικανοποιούν το στόχο *Predicate*.
- Εάν δεν υπάρχει λύση τότε το κατηγόρημα **αποτυγχάνει**.
- Η λίστα Solutions δεν περιέχει διπλές εμφανίσεις στοιχείων.

Παράδειγμα

- Συγκέντρωση όλων των μοναδικών στοιχείων μιας λίστας.

```
remove_duplicates( List, Result):-
    setof( X, member( X, List), Result).
```

Ύπαρξη “ελεύθερων” Μεταβλητών

- Αν υπάρχουν “ελεύθερες” μεταβλητές τότε οι λύσεις επιστρέφονται για κάθε εναλλακτική τιμή της ελεύθερης μεταβλητής.

?- setof(SH, shape(SH, C), Sols).

C = green Sols = [circle] ;

C = red Sols = [circle, square, triangle] ;

C = yellow Sols = [triangle];

Program

```
shape(triangle, red).
shape(triangle, yellow).
shape(circle, red).
shape(circle, green).
shape(square, red).
```

Ύπαρξη “ελεύθερων” Μεταβλητών

- Η συμπεριφορά μπορεί να διαφοροποιηθεί από τον τελεστή ^.

?- setof(SH, C^shape(SH, C), Sols).

Sols = [circle, square, triangle] ;

Program

shape(triangle, red).
shape(triangle, yellow).
shape(circle, red).
shape(circle, green).
shape(square, red).

- Ενώ το findall/3 θα έβρισκε

?- findall(SH, shape(SH, C), Sols).

Sols = [triangle, triangle, circle, circle, square]

Yes (0.00s cpu)

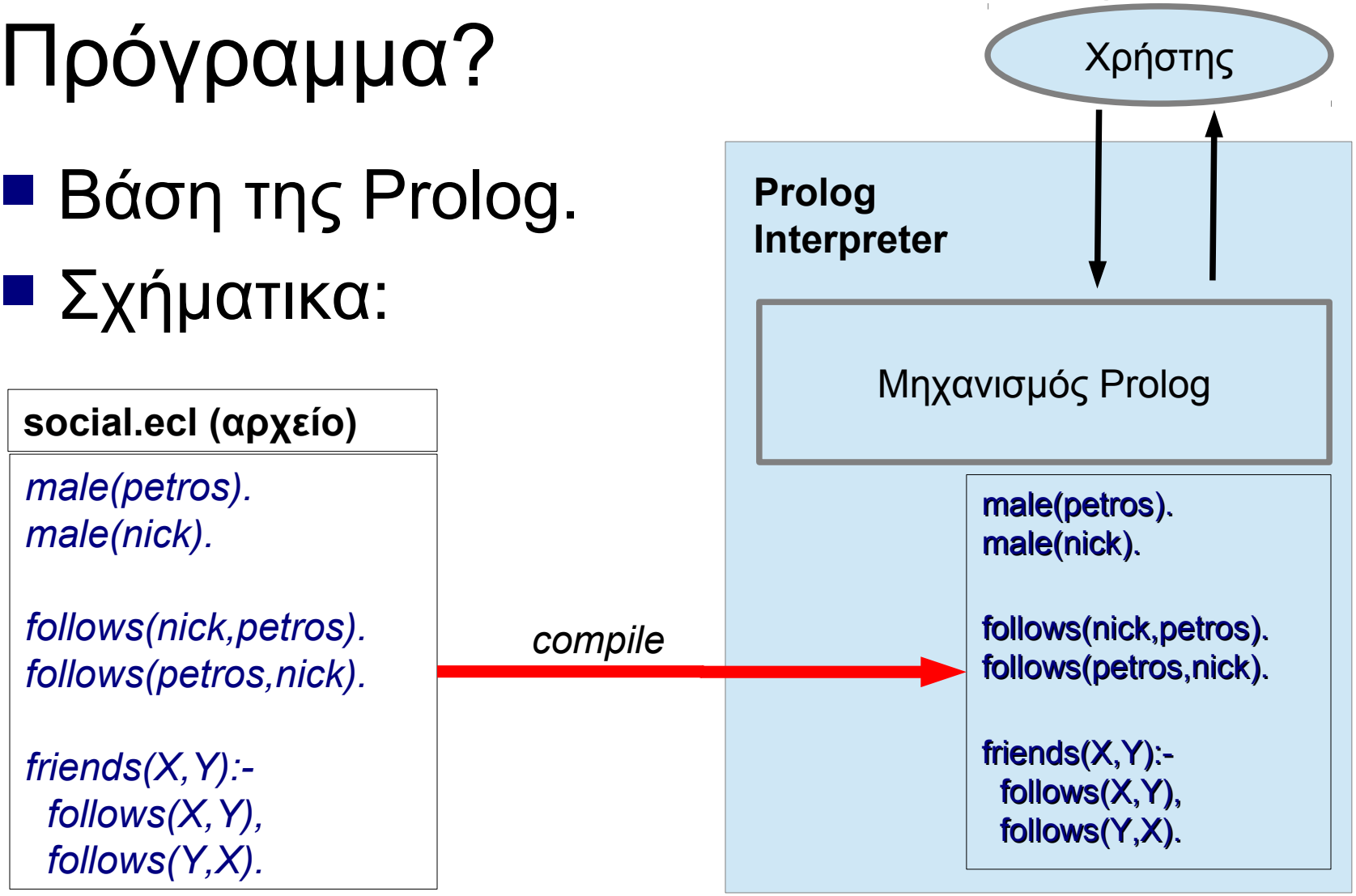
bagof/3

- *bagof(Term, Predicate, Solutions)*
- Ενοποιεί τη λίστα *Solutions* με τιμές του *Term* που ικανοποιούν το στόχο *Predicate*.
- Εάν δεν υπάρχει λύση τότε το κατηγόρημα **αποτυγχάνει**.
- Η λίστα Solutions **μπορεί** περιέχει διπλές εμφανίσεις στοιχείων.

Δυναμικά Κατηγορήματα

Που αποθηκεύεται ένα Prolog Πρόγραμμα?

- Βάση της Prolog.
- Σχήματικα:



Στατικά κατηγορήματα

- Κατηγορήματα τα οποία δεν αλλάζουν **κατά την εκτέλεση** του προγράμματος.
  - "Φορτώνονται" από το αρχείο.
    - Όποια αλλαγή πρέπει αν γίνει απαιτεί επαναφόρτωση του αρχείου.
  - Οι προτάσεις τους πρέπει να είναι "μαζί".
  - … ότι κατηγόρημα έχουμε γράψει μέχρι στιγμής...

Δυναμικά Κατηγορήματα

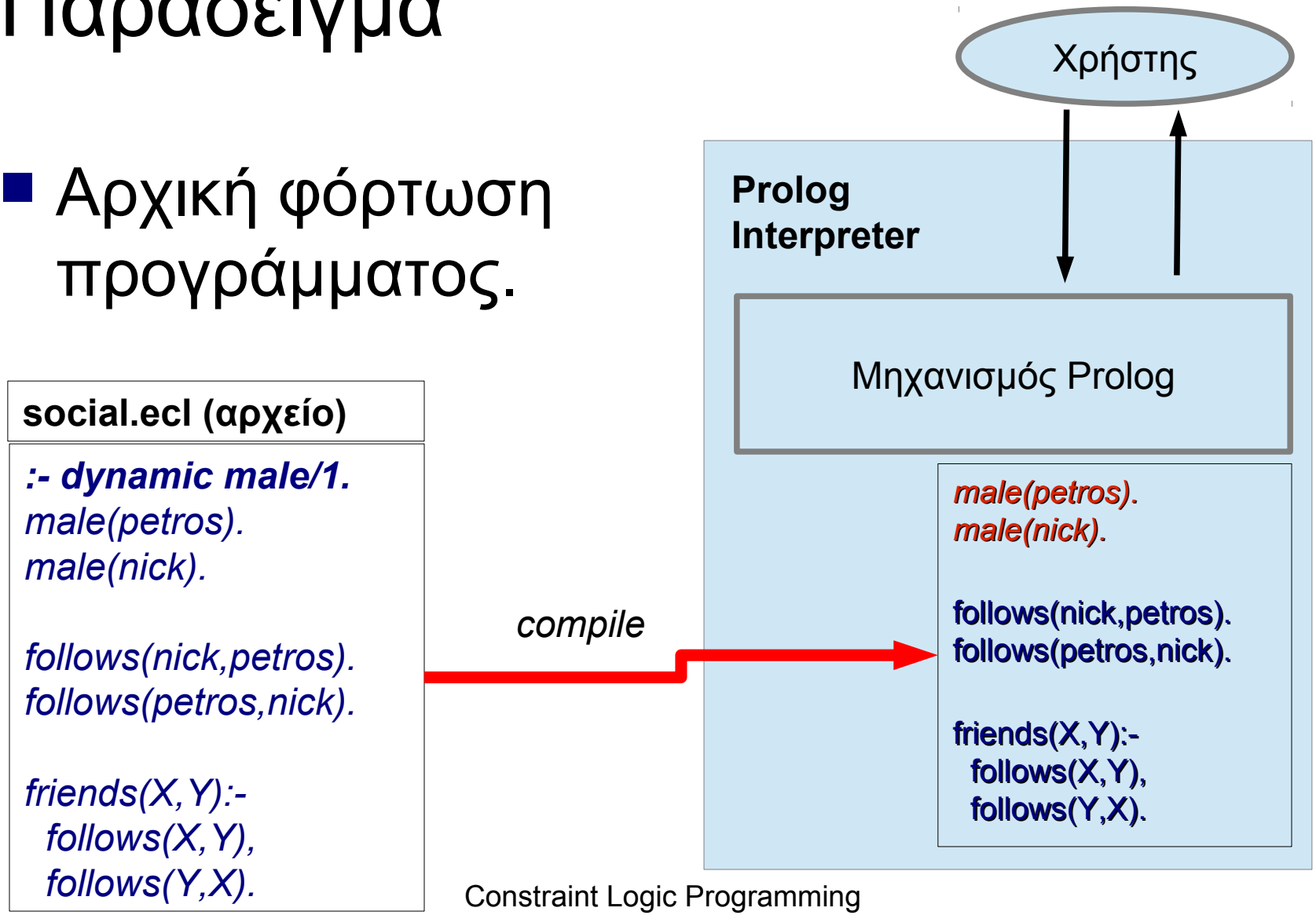
- Τα δυναμικά κατηγορήματα στην ECLiPSe, έχουν την δυνατότητα **να μεταβληθούν κατά την εκτέλεση** (at run-time).
- Δηλώνονται με *:-dynamic pred/arity.*
- Αλλαγή της συμπεριφοράς του προγράμματος.
- Πρέπει να χρησιμοποιούνται με μεγάλη προσοχή.
  - Μπορεί να δώσουν μην αναμενόμενα αποτελέσματα.

Διαχείριση της Βάσης των Κατηγορήματων

- *asserta(Clause) – assertz(Clause)*
  - Προσθέτει την πρόταση Clause στην αρχή ή στο τέλος του κατηγορήματος.
  - Απαιτεί δήλωση *dynamic(Pred/Arity).*
- *retract(Clause)*
  - Διαγράφει ένα κατηγόρημα από τη βάση.
- *reatractall(Head)*
  - Διαγράφει ένα κατηγόρημα (all clauses).

Παράδειγμα

- Αρχική φόρτωση προγράμματος.

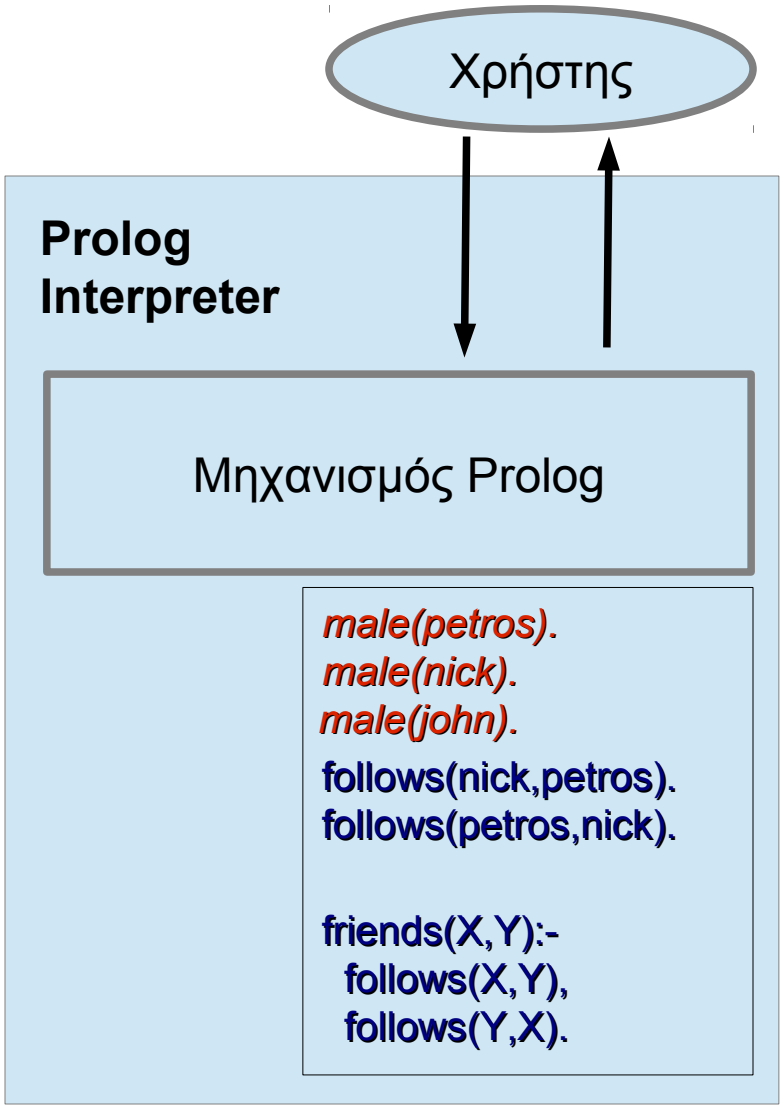


Παράδειγμα

- *?- male(petros)*

  yes
- *?- assertz(male(john)).*
- *?- male(john).*

  yes

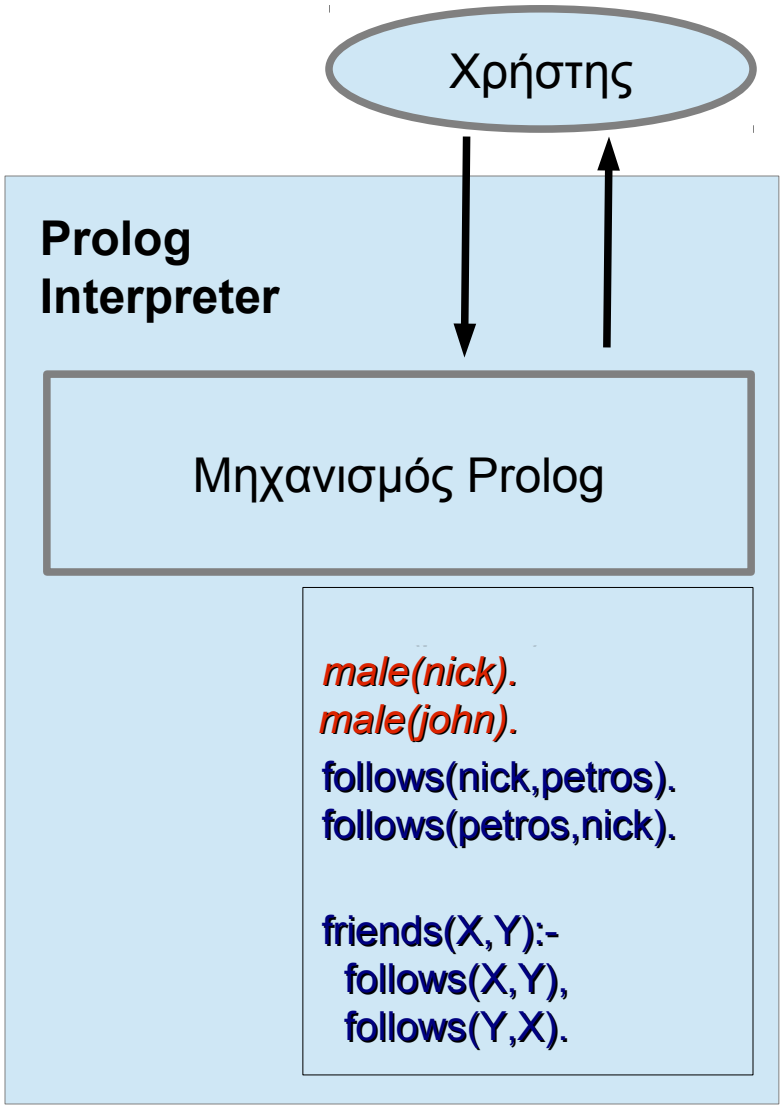


Παράδειγμα

- *?- retract(male(petros)).*

  yes
- *?- male(petros).*

  no



Clause/2

- Το κατηγόρημα *clause(H,B)* πετυχαίνει αν υπάρχει στη βάση ένα κατηγόρημα με κεφαλή Η και σώμα Β.
- Παράδειγμα: Να ορίσετε ένα κατηγόρημα το οποίο “συλλέγει” όλα τα σώματα των διαφορετικών προτάσεων ενός κατηγορήματος.

```
collect_bodies( Predicate, Bodies):-
    findall( Body,clause( Predicate, Body), Bodies).
```

Χρήση

- Υλοποίηση προγραμμάτων που μαθαίνουν.
- Υλοποίηση προσαρμοζόμενων προγραμμάτων.
- Υλοποίηση “καθολικών” μεταβλητών.
- και πολλά άλλα...

Υλοποιώντας Λήμματα

- Μαθαίνοντας με την μορφή γεγονότων υπο-στόχους που έχουν αποδειχθεί.
- Βελτίωση της απόδοσης του προγράμματος.

```
lemma(Goal):-
    call(Goal),
    assert(Goal).
```

Fibonacci Numbers

```
fibonacci(0,1).
fibonacci(1,1).
fibonacci(N,F):-
    N>1,
    N1 is N -1,
    N2 is N -2,
    fibonacci(N1,F1),fibonacci(N2,F2),
    F is F1+F2.
```

Αρχική έκδοση

Fibonacci Numbers

```
fibonacci(0,1).
fibonacci(1,1).
fibonacci(N,F):-
    N>1,
    N1 is N -1,
    N2 is N -2,
    lemma(fibonacci(N1,F1)),
    fibonacci(N2,F2),
    F is F1+F2.
```

Έκδοση που μαθαίνει

Ένα καλύτερο Lemma

```
lemma_once(Goal):-
    call(Goal),
    (not( clause(Goal,!) )->
            asserta((Goal:-!));true).
```