Κεφάλαιο 7
Λογικός Προγραμματισμός: Η Γλώσσα Prolog

Π. Ροντογιάννης

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Βασικά Στοιχεία Γλώσσας Prolog

- **Ορισμοί (statements)**: Επιτελούν το ρόλο εντολών στις κλασσικές γλώσσες προγραμματισμού
  - Γεγονότα
  - Κανόνες
  - Ερωτήσεις
- **Όροι (terms)**: Μοναδική δομή δεδομένων Prolog

Γεγονός

- Είναι το απλούστερο είδος ορισμού, που υποστηρίζει η Prolog.
- Παρέχει τρόπο έκφρασης της **σχέσης** που ισχύει ανάμεσα σε οντότητες.

Παράδειγμα Γεγονότος

Γεγονός

```
father (john, mary).
```

Εκφράζει ότι ο john *είναι πατέρας της* mary *ή ότι η σχέση* father *ισχύει ανάμεσα στα άτομα* john *και* mary.

Ορισμοί

- Σχέσεις όπως το `father` ονομάζονται **κατηγορήματα**.
- Οι οντότητες `mary` και `john` ονομάζονται **ορίσματα** του κατηγορήματος.
- Ένα κατηγόρημα μαζί με τα ορίσματα που περιλαμβάνει ονομάζεται **ατομικός τύπος**.

Σχέση Plus

Σύνολο γεγονότων

```
plus(0,0,0).
plus(0,1,1).
plus(1,0,1).
plus(2,0,2).
plus(0,2,2).
```

Σχέση Plus

- Ορίζει την πρόσθεση φυσικών αριθμών
- Δεν μπορεί να οριστεί αποκλειστικά με τη χρήση γεγονότων
  - Απαιτείται άπειρος αριθμός γεγονότων, (όπως παραπάνω)

Μεταβλητές

- Η Prolog επιτρέπει τη χρήση **μεταβλητών** σε
  - Γεγονότα, κανόνες και ερωτήσεις
- Οι μεταβλητές αρχίζουν πάντα με κεφαλαίο γράμμα
- Η χρήση μεταβλητών επιτρέπει την έκφραση **συνόλου γεγονότων**

Παράδειγμα

```
plus(0,X,X).
```

Εκφράζει την ιδιότητα του ουδέτερου στοιχείου της πρόσθεσης.

Ερωτήσεις

- Παρέχουν τη δυνατότητα εξαγωγής πληροφορίας από ένα λογικό πρόγραμμα.
- Διερευνούν την ισχύ μιας σχέσης ανάμεσα σε ορισμένες οντότητες.

Παράδειγμα

Ερώτηση

```
?-father(john, mary).
```

Διερευνά αν η σχέση father *ισχύει ανάμεσα στα άτομα* john *και* mary.

Παράδειγμα

Ερώτηση

Αν έχει δοθεί στην Prolog το γεγονός:

```
father(john, mary).
```

Τότε η απάντηση στην ερώτηση `?-father(john, mary).`
θα είναι YES.

Μεταβλητές και Ερωτήσεις (1)

Ερώτηση

```
?-father(X,mary).
```

Διαβάζεται ως: «Υπάρχει κάποιο πρόσωπο `X` *το οποίο είναι πατέρας της* `mary`*;»*

Μεταβλητές και Ερωτήσεις (2)

- Μία ερώτηση μπορεί να επιστρέψει περισσότερες από μία απαντήσεις.

Παράδειγμα

Αν έχουμε δώσει στην Prolog δύο γεγονότα

```
father(john, mary).
father(john, ann).
```

Τότε η ερώτηση: `?-father(john, X).`
Θα επιστρέψει δύο λύσεις `X=mary` *και* `X=ann`

Σύνθετες Ερωτήσεις

Ερώτηση

```
?-father(X,mary), father(X,ann)
```

Αναζητά το πρόσωπο που είναι πατέρας της mary *και της* ann.

- Η σύνθετη ερώτηση αποτελείται από υποερωτήσεις (κλήσεις). Οι κλήσεις σε μία σύνθετη ερώτηση χωρίζονται μεταξύ τους με κόμμα, που έχει την έννοια λογικού «και».
- Για να αληθεύει μία σύνθετη ερώτηση πρέπει να αληθεύουν όλες οι απλούστερες ερωτήσεις, που την αποτελούν.

Κανόνες

- Επιτρέπουν τον ορισμό νέων σχέσεων ως συνάρτηση άλλων σχέσεων.

Παράδειγμα

```
son(X,Y):-father(Y,X), male(X).
```

Ορίζει τη σχέση son *ως συνάρτηση των σχέσεων* father *και* male.
Διαβάζεται ως: «Ο X *είναι γιός του* Y *αν ο* Y *είναι πατέρας του* X *και ο* X *είναι γένους αρσενικού».*

- Το σύμβολο «:-» διαβάζεται σαν «εάν», ενώ το «,» ως «και».

Αναδρομικοί Κανόνες

Παράδειγμα

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

Η σχέση ancestor *(πρόγονος) μπορεί να ορισθεί με χρήση δύο κανόνων, εκ των οποίων ο ένας είναι αναδρομικός.*

- Γενικά, ένας κανόνας έχει τη μορφή

$$\mathtt{H}\texttt{:-}\mathtt{B}_1\texttt{,}\ldots\texttt{,}\mathtt{B}_n\texttt{.}$$

όπου το H λέγεται **κεφαλή** του κανόνα και οι ατομικοί τύποι $\mathtt{B}_1,\ldots,\mathtt{B}_n$, αποτελούν το **σώμα** του κανόνα.

Όροι

- Βασική δομή στον λογικό προγραμματισμό
- Σταθερά (όπως `john`), μεταβλητή (όπως `X`) ή σύνθετος όρος
  - Ο σύνθετος όρος αποτελείται από ένα συναρτησιακό σύμβολο (functional symbol), που εφαρμόζεται σε ακολουθία από όρους.

Παράδειγμα

`list(a,nil)`

όπου `list` *το συναρτησιακό σύμβολο και* `a,nil` *σταθερές*

Όροι

- Χρήση για κωδικοποίηση σύνθετων μορφών πληροφορίας

Παράδειγμα

Ορισμός κατηγορήματος book, *που λαμβάνει σαν παραμέτρους όρους για περιγραφή των χαρακτηριστικών της έννοιας βιβλίο. Με τον τρόπο αυτό δημιουργείται μία απλή βάση δεδομένων:*

```
book(author(surname(stoll), name(robert)),
     subject(logic)).
book(author(surname(kleene), name(steven)),
     subject(mathematics)).
```

Όροι

Διαφορά ανάμεσα στα

- book (όνομα κατηγορήματος)
- author, surname, name, subject (συναρτησιακά σύμβολα)

Παράδειγμα

Ερώτηση

Αναζήτηση ονοματεπωνύμων, που έχουν γράψει βιβλία σε λογική

```
?-book(author(surname(X), name(Y)),
                subject(logic)).
```

Ερώτηση

Αν δεν ενδιαφέρουν τα μικρά ονομάτων συγγραφέων, η ερώτηση μπορεί να τεθεί:

```
?-book(author(surname(X), _), subject(logic)).
```

Όπου με «_» συμβολίζονται οι όροι των οποίων η τιμή δεν μας ενδιαφέρει.

Όροι

- Γραφή προγραμμάτων όχι μόνο με χρήση κανόνων και γεγονότων

Παράδειγμα

Ορισμός της σχέσης `nat(X)`*, η οποία αληθεύει όταν* X *φυσικός αριθμός.*

- *Ένας τρόπος είναι να συμφωνηθεί η αναπαράσταση των φυσικών αριθμών αντί για* $0, 1, 2, \ldots$ *με* $0, \mathtt{s}(0), \mathtt{s}(\mathtt{s}(0)), \ldots$
- *Το συναρτησιακό σύμβολο* `s` *εκφράζει την έννοια του «επόμενου αριθμού».*

Παράδειγμα

Το πρόγραμμα για το nat γράφεται:

Πρόγραμμα

```
nat(0).
nat(s(X)):-nat(X).
```

Παρόμοια ορίζονται και άλλες σχέσεις πάνω σε φυσικούς αριθμούς.

Παράδειγμα

Το `plus(X,Y,Z)` είναι αληθές αν το `Z` είναι το **άθροισμα** των `X` και `Y`.

Πρόγραμμα

```
plus(0,X,X).
plus(s(X),Y,s(Z)):-plus(X,Y,Z).
```

Θέτοντας την ερώτηση: `?-plus(s(0), s(0),K).`
Παίρνουμε την απάντηση `K = s(s(0))`.

Ορίσματα

- Στην Prolog δεν γίνεται καθορισμός ποια από τα ορίσματα ενός κατηγορήματος είναι είσοδοι και ποια έξοδοι.
- Πολλά προγράμματα έχουν διαφορετικές χρήσεις
- Παράδειγμα: Αν τεθεί η ερώτηση:
  `?-plus(X,s(0), s(s(s(0))))`.
  Δίνεται η απάντηση `X=s(s(0))`, που αντιστοιχεί στον αριθμό, που λαμβάνεται αν αφαιρέσουμε από το τρίτο όρισμα το δεύτερο.

Εκτέλεση Προγραμμάτων Prolog

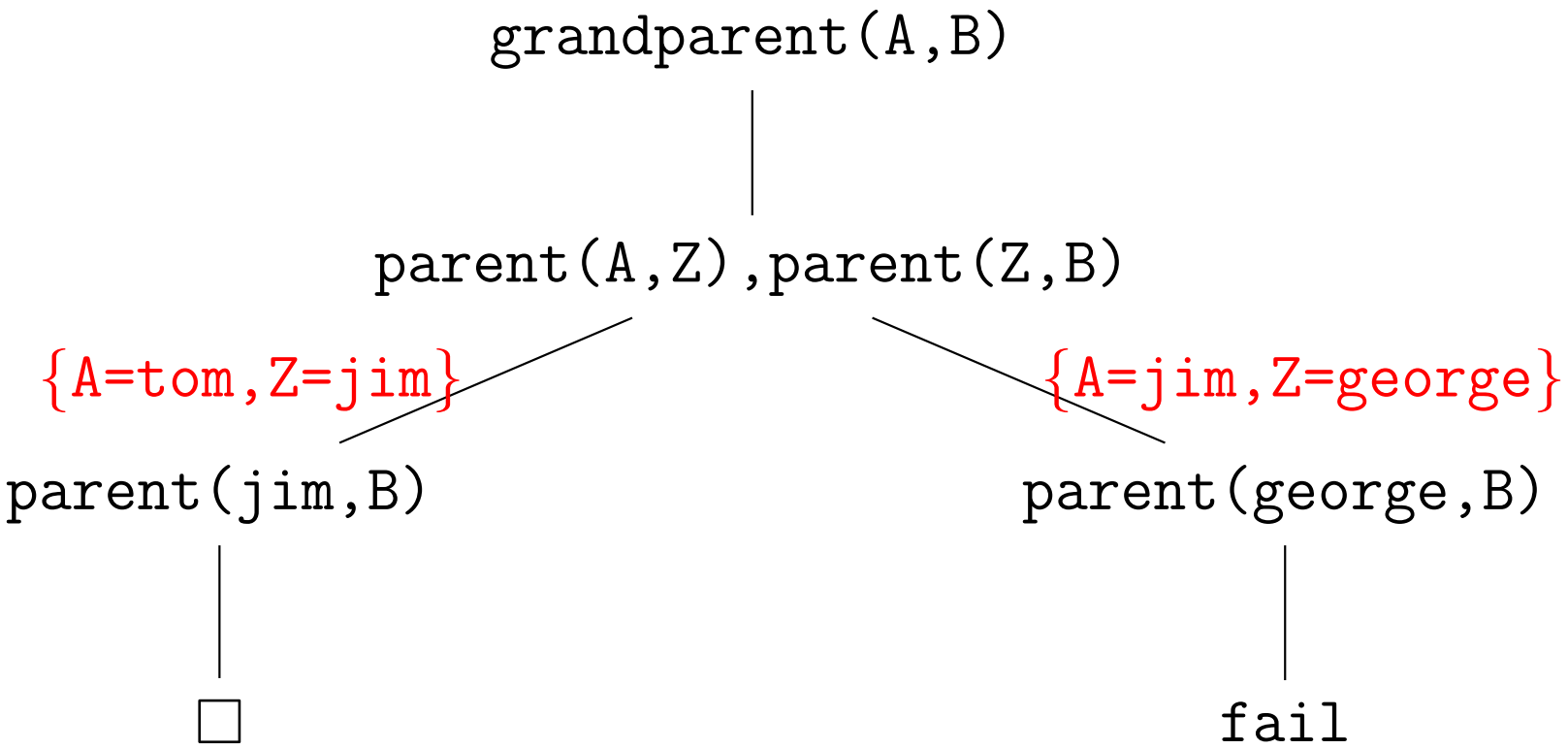
Έστω το πρόγραμμα

Πρόγραμμα

```
grandparent(X,Y):-parent(X,Z), parent(Z,Y).
parent(tom,jim).
parent(jim,george).
```

Όταν τεθεί η ερώτηση:`?-grandparent(A,B).` ο μηχανισμός εκτέλεσης της Prolog εκτελεί αναζήτηση λύσεων της μορφής του δέντρου, που ακολουθεί.

Δέντρο εκτέλεσης της ερώτησης



Εκτέλεση Προγραμμάτων Prolog

- Αρχικά ο μηχανισμός της Prolog αναζητά στο πρόγραμμα κατηγόρημα με το όνομα `grandparent` ώστε να ταιριάζει με τη δεδομένη ερώτηση.
- Μόλις βρεθεί πρόταση, που αφορά αυτό το κατηγόρημα, αντικαθιστά την δεδομένη ερώτηση με το σώμα της πρότασης και δημιουργείται νέα (σύνθετη) ερώτηση `parent(A,Z), parent(Z,B).`

Μηχανισμός Εκτέλεσης Prolog

- Έστω ότι έχουμε ένα πρόγραμμα και μία ερώτηση της μορφής $? - A_1, \ldots, A_n$. Τότε:
- Η Prolog εξετάζει εάν ικανοποιούνται όλοι οι στόχοι $A_1, \ldots, A_n$ ξεκινώντας από το A_1 και πηγαίνοντας προς το A_n.
- Για να ικανοποιήσει ένα από τα A_i η Prolog,
  - διαλέγει την πρώτη πρόταση στο πρόγραμμα της οποίας η κεφαλή ταιριάζει με το A_i,
  - και αντικαθιστά το A_i με το σώμα της πρότασης αυτής (αφού πρώτα το τροποποιήσει κατάλληλα).
- Η διαδικασία συνεχίζεται μέχρι να μην υπάρχει πλέον άλλη κλήση, που να πρέπει να ικανοποιηθεί.

Απλά Αναδρομικά Προγράμματα σε Prolog

Με τη χρήση όρων γράφονται απλά και εκφραστικά προγράμματα, όπως το

Πρόγραμμα

```
plus(0,X,X).
plus(s(X),Y,s(Z)):-plus(X,Y,Z).
```

Όμοια ορίζεται για τον **πολλαπλασιασμό** φυσικών αριθμών

Πρόγραμμα

```
times(0,X,0).
times(s(X),Y,Z):-times(X,Y,W), plus(W,Y,Z).
```

Σχόλια

- Η παραπάνω σχέση χρησιμοποιεί δύο βασικά αξιώματα του πολλαπλασιασμού:
  - $0 \cdot X = 0$
  - $(X + 1) \cdot Y = X \cdot Y + Y$
- Ο ορισμός του πολλαπλασιασμού χρησιμοποιεί αυτόν της πρόσθεσης

Μαθηματικές Σχέσεις Φυσικών Αριθμών

Με χρήση του ορισμού του πολλαπλασιασμού δίνεται η σχέση για το **παραγοντικό** ενός φυσικού αριθμού

Πρόγραμμα

```
factorial(0,s(0)).
factorial(s(N),F):-factorial(N,F1),times(s(N),F1,F).
```

Μαθηματικές Σχέσεις Φυσικών Αριθμών

Σχέση `between(X,Y,Z)`: αληθεύει όταν `X` είναι ένας φυσικός αριθμός μικρότερος ή ίσος από τον `Y` και ο `Y` είναι μικρότερος ή ίσος από τον `Z`.

Πρόγραμμα

```
between(0,0,Z).
between(0,s(Y),s(Z)):-between(0,Y,Z).
between(s(X),s(Y),s(Z)):-between(X,Y,Z).
```

Σχόλια

- Οι φυσικοί αριθμοί αναπαρίστανται με πολύ πιο βολικό τρόπο στην Prolog (όπως και στις υπόλοιπες γλώσσες προγραμματισμού).
- Η αναπαράσταση, που χρησιμοποιήθηκε στα παραπάνω προγράμματα δεν είναι γενικά βολική και αποτελεσματική. Υιοθετήθηκε για να δειχθούν οι αρχικές δυνατότητες της Prolog.

Αναδρομικός Προγραμματισμός με Λίστες

Λίστα

- Πολύ χρήσιμη δομή δεδομένων για προγραμματισμό σε Prolog
- **Κενή** - συμβολίζεται με `[]`
- Περιέχει στοιχεία, π.χ. η `[a,b,c]` είναι **μη κενή** λίστα
  - Το πρώτο στοιχείο λίστας ονομάζεται **κεφαλή (head)**
  - Η λίστα που προκύπτει αν αφαιρέσουμε την κεφαλή, ονομάζεται **ουρά (tail)**
  - Ο τελεστής `|` αποτελεί ειδική αναπαράσταση, που δείχνει άμεσα την κεφαλή και την ουρά μιας λίστας. Η λίστα `[a,b,c]` γράφεται ως `[a|[b,c]]` και η `[a]` ως `[a|[]]`.

Παράδειγμα 7.1

- Οι λίστες Prolog μπορούν να περιέχουν σα στοιχεία τους και άλλες λίστες, αλλά και πολύπλοκους όρους, όπως
  - `[[]]` επιτρεπτή λίστα
  - `[[1,X], s(s(X))]`
- Οι `[a|[1|[2]]]` και `[X,Y|[a]]` είναι ισοδύναμες με τις λίστες `[a,1,2]` και `[X,Y,a]` αντίστοιχα.

Παράδειγμα 7.2

Η σχέση `member(X,Y)` είναι αληθής, όταν το `X` είναι **στοιχείο** της λίστας `Y`.

Πρόγραμμα

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

Το νόημα του προγράμματος είναι: «Το `X` είναι μέλος μιας λίστας αν είναι η κεφαλή της λίστας ή αν είναι μέλος της ουράς της λίστας».

Παράδειγμα 7.3

Η σχέση append(X,Y,Z) είναι αληθής, όταν το Z είναι η λίστα, που προκύπτει από την **συνένωση** των λιστών X και Y.

Πρόγραμμα

```
append([],Y,Y).
append([X|Xs],Ys,[X|Zs]):-append(Xs,Ys,Zs).
```

Η δομή του παραπάνω προγράμματος είναι παρόμοια με αυτή του προγράμματος για τον ορισμό της σχέσης plus.

Παράδειγμα 7.4

- Συχνά στον προγραμματισμό σε Prolog χρησιμοποιούνται απλές σχέσεις, ήδη ορισμένες για να γραφούν πιο πολύπλοκα προγράμματα.
- Σχέσεις όπως η `member` και η append χρησιμοποιούνται σχεδόν σε οποιοδήποτε μεγάλο πρόγραμμα γραφτεί σε Prolog.

Παράδειγμα 7.4

Σχέση `reverse(X,Y)` ορίζεται σα συνάρτηση της append. Είναι αληθής, όταν η λίστα `Y` είναι η **αντίστροφη** της λίστας `X`:

Πρόγραμμα

```
reverse([],[]).
reverse([X|Xs],Ys):-reverse(Xs,Rs),
                    append(Rs,[X],Ys).
```

Το νόημα του παραπάνω προγράμματος είναι: «Η αντίστροφη της **κενής** λίστας είναι η κενή λίστα.
Η αντίστροφη **μη-κενής λίστας** μπορεί να παραχθεί αντιστρέφοντας την **ουρά** της και προσκολλώντας στο τέλος της αντεστραμμένης ουράς την **κεφαλή** της αρχικής λίστας.»

Παράδειγμα 7.4

Το προηγούμενο παράδειγμα μπορεί να γραφεί και χωρίς τη χρήση της append, αλλά με προσθήκη ενός βοηθητικού κατηγορήματος του reverse1, που διαθέτει ένα επιπλέον όρισμα:

Πρόγραμμα

```
reverse(Xs, Ys):-reverse1(Xs,[],Ys).

reverse1([],Ys, Ys).
reverse1([X|Xs],A,Ys):-reverse1(Xs,[X|A],Ys).
```

Το όρισμα A ονομάζεται **συσσωρευτής (accumulator)** διότι συσσωρεύει βήμα-βήμα το τελικό αποτέλεσμα (δηλαδή την αντίστροφη λίστα).

Δέντρα

- Πολύ χρήσιμος τύπος δεδομένων
- Αναπαρίστανται με χρήση **όρων** (μοναδική δομή δεδομένων Prolog)
  - Χρήση του συναρτησιακού συμβόλου `tree` με τρεις παραμέτρους:
    `tree(Element, Left, Right)`
    Το στοιχείο `Element` είναι η **ρίζα** του συγκεκριμένου δέντρου, το `Left` το **αριστερό** υπόδεντρο και το `Right` το **δεξί** υποδέντρο.
  - Το **κενό** δέντρο αναπαρίσταται με τη σταθερά `void`.

Παράδειγμα

Παράδειγμα

Δέντρο με κορυφή a, *αριστερό παιδί* b *και δεξί παιδί* c *γράφεται ως:*

```
tree(a, tree(b,void,void), tree(c,void, void))
```

Παράδειγμα

Πρόγραμμα για έλεγχο του αν ένας δεδομένος όρος είναι πράγματι **δυαδικό** δέντρο (ανάλογο του κατηγορήματος nat)

Πρόγραμμα

```
binary_tree(void).
binary_tree(tree(E,L,R)):-binary_tree(L),
                          binary_tree(R).
```

Παράδειγμα

Πρόγραμμα για έλεγχο του αν ένα δεδομένο στοιχείο **ανήκει** σε ένα δέντρο ή όχι. Το πρόγραμμα είναι ανάλογο με το *member*, που ορίσθηκε για τις λίστες:

Πρόγραμμα

```
tree_member(X, tree(X,_,_)).
tree_member(X, tree(Y,L,R)):-tree_member(X,L).
tree_member(X, tree(Y,L,R)):-tree_member(X,R).
```

Παράδειγμα

- Το παραπάνω πρόγραμμα ελέγχει αν ένα δεδομένο στοιχείο είναι ταυτόσημο με την κορυφή του δεδομένου δέντρου.
- Αν ναι, το πρόγραμμα σταματά με επιτυχία.
- Αν όχι, το πρόγραμμα συνεχίζει αναδρομικά τη διερεύνηση στο αριστερό και το δεξί υποδέντρο.

Κατηγορήματα

- Ουσιαστικά «διασχίζουν» ένα δεδομένο δέντρο με μία **προκαθορισμένη** σειρά.
- Το αποτέλεσμα της διάσχισης επιστρέφεται σε μία λίστα.

Παράδειγμα

Η preorder διάσχιση ενός δυαδικού δέντρου ορίζεται από το ακόλουθο κατηγόρημα:

Πρόγραμμα

```
preorder(void,[]).
preorder(tree(X,L,R),Xs):-preorder(L,Ls),
                          preorder(R,Rs),
                          append([X|Ls],Rs,Xs).
```

Κατά την preorder διάσχιση ενός δέντρου καταγράφεται αρχικά η ρίζα του δέντρου και κατόπιν επισκεπτόμαστε αναδρομικά πρώτα το αριστερό υποδέντρο και μετά το δεξί.

Παράδειγμα

Με ανάλογο τρόπο γράφεται πρόγραμμα, που υλοποιεί την διάσχιση inorder:

Πρόγραμμα

```
inorder(void,[]).
inorder(tree(X,L,R),Xs):-inorder(L,Ls),
                          inorder(R,Rs),
                          append(Ls,[X|Rs],Xs).
```

Παράδειγμα

Πρόγραμμα για την postorder διάσχιση:

Πρόγραμμα

```
postorder(void,[]).
postorder(tree(X,L,R),Xs):-postorder(L,Ls),
                           postorder(R,Rs),
                           append(Ls,Rs,Ms),
                           append(Ms,[X],Xs).
```

Τα παραπάνω προγράμματα διαφέρουν μόνο ως προς τον τρόπο, που γίνεται η **συνένωση** των επιμέρους αποτελεσμάτων (append).

Τελεστές

- Συναρτησιακά σύμβολα
- Χρήση σε επεξεργασία πιο πολύπλοκων δομών από ότι οι λίστες και τα δέντρα
- Στην έκφραση 1+3, το + είναι **τελεστής**
  - Ισοδύναμη έκφραση +(1,3) και όχι ο αριθμός 4, όπως συμβαίνει σε άλλες γλώσσες προγραμματισμού
  - Η πρόσθεση θα πραγματοποιούταν αν ο όρος 1+3 ήταν όρισμα στο ειδικό κατηγόρημα `is` της Prolog για υπολογισμό αριθμητικών εκφράσεων

Τελεστές (2)

- Οι τελεστές +, −, *, /, . . . δεν λαμβάνουν συγκεκριμένο νόημα από την Prolog
- Χρησιμοποιούνται για να γραφούν εφαρμογές με κομψό και συνοπτικό τρόπο
- Στο επόμενο παράδειγμα οι τελεστές χρησιμοποιούνται, για τη διατύπωση της διαδικασίας παραγώγισης συναρτήσεων

Παραγώγιση

Πρόγραμμα

```
derivative(N,X,0):-nat(N).
derivative(X,X,s(0)).
derivative(X^s(N),X,s(N)*(X^N)).
derivative(sin(X),X,cos(X)).
derivative(cos(X),X,-sin(X)).
derivative(e^X,X,e^X).
derivative(log(X),X,1/X).
```

Παραγώγιση

Πρόγραμμα

```
derivative(F+G, X, DF+DG):-
    derivative(F,X,DF),derivative(G,X,DG).

derivative(F-G, X, DF-DG):-
    derivative(F,X,DF),derivative(G,X,DG).

derivative(F*G, X, F*DG+G*DF):-
    derivative(F,X,DF),derivative(G,X,DG).

derivative(1/F, X, -DF/(F*F)):-
derivative(F,X,DF).

derivative(F/G, X, G*DF-F*DG/(G*G)):-
    derivative(F,X,DF), derivative(G,X,DG).
```

Παραγώγιση

- Το πρόγραμμα υπολογίζει την **παράγωγο** μίας μεγάλης κατηγορίας συναρτήσεων.
- Οι τελεστές, που χρησιμοποιούνται εκφράζουν πράξεις μεταξύ συναρτήσεων.
- Την έκφραση `F*G`, ο προγραμματιστής την καταλαβαίνει σαν πολλαπλασιασμό συναρτήσεων και αυτό είναι δυνατό καθώς η Prolog δεν δίνει προκαθορισμένο νόημα σε σύμβολα, όπως το *.

Παραγώγιση

Αν στο παραπάνω πρόγραμμα δοθεί η ερώτηση

```
?-derivative(cos(x)*(x^s(s(0))),x,D).
```

Η οποία αναζητά την παράγωγο της συνάρτησης $\cos x \cdot x^2$

Θα λάβουμε την απάντηση

```
D=cos(x)*(s(s(0))*x^s(0))+x^s(s(0))*(-sin(x))
```

Η οποία αντιστοιχεί στη συνάρτηση $\cos x \cdot 2x + x^2 \cdot (-\sin x)$.

Αριθμητικές πράξεις και Prolog

- Κάθε γλώσσα προγραμματισμού παρέχει ιδιαίτερες ευκολίες στη χρήση και επεξεργασία αριθμών.
- Η Prolog διαθέτει
  - Ειδικά κατηγορήματα, γνωστά ως **κατηγορήματα συστήματος (system predicates)**.

Κατηγορήματα Prolog

- Βασικό κατηγόρημα το `is`, που χρησιμοποιείται στη μορφή `V is E`.
- Η Prolog υπολογίζει την τιμή του `E` και αν αυτή η τιμή συμφωνεί με το `V`, τότε επιτυγχάνει.
- Αν `V` μεταβλητή, τότε λαμβάνει την τιμή του `E`.
- Όταν `E` περιέχει μεταβλητές με άγνωστη τιμή τη στιγμή της εκτέλεσης, ο διερμηνέας της Prolog δίνει μήνυμα λάθους.

Παράδειγμα 7.5

- Η ερώτηση

  `?-8 is 5+3` **επιτυγχάνει**

- Η ερώτηση

  `?-5+3 is 8` **αποτυγχάνει**

  διότι η Prolog βλέπει το αριστερό μέλος της σαν τον όρο `+(5,3)` και δεν υπολογίζει την τιμή του.

Παράδειγμα

- Η ερώτηση

  ```
  ?-X is 5+3
  ```

  θα δώσει την απάντηση `Q = 8`
- Η ερώτηση

  ```
  ?-X is 2+Y
  ```

  θα δώσει μήνυμα λάθους,
  διότι η αριθμητική τιμή του δεξιού μέλους δεν μπορεί να υπολογισθεί λόγω της μεταβλητής `Y`.

Κατηγορήματα της Prolog για αριθμητικές πράξεις

`X =:= Y`	Οι αριθμητικές τιμές των `X` και `Y` είναι ίδιες
`X == Y`	Οι αριθμητικές. τιμές των `X` και `Y` είναι διαφορετικές
`X < Y`	Η αριθμητική τιμή του `X` είναι μικρότερη από αυτή του `Y`
`X =< Y`	Η αριθμητική τιμή του `X` είναι μικρότερη ή ίση από αυτή του `Y`
`X > Y`	Η αριθμητική τιμή του `X` είναι μεγαλύτερη από αυτή του `Y`
`X >= Y`	Η αριθμητική τιμή του `X` είναι μεγαλύτερη ή ίση από αυτή του `Y`

Κατηγορήματα

- Με χρήση των προηγούμενων κατηγορημάτων μπορούν να ξαναγραφούν κατηγορήματα που ορίσθηκαν με χρήση των όρων

$$(\pi.\chi.\ 0, s(0), s(s(0)) \ldots)$$

Κατηγορήματα

Το κατηγόρημα `plus` μπορεί να ορισθεί ως:

Πρόγραμμα

```
plus(X,Y,Z):-Z is X+Y
```

Μειονέκτημα: Χάνεται η αντιστρεψιμότητα κάποιων προγραμμάτων

Κατηγορήματα

- Η ερώτηση

  ```
  ?-plus(A,B,8)
  ```

  θα δώσει μήνυμα λάθους (και όχι όλα τα ζεύγη φυσικών με άθροισμα 8).
- Αυτό συμβαίνει γιατί η παραπάνω ερώτηση ανάγεται από το μηχανισμό της Prolog στην ερώτηση

  ```
  ?-8 is A+B
  ```

Κατηγορήματα

Με χρήση κατηγορημάτων συστήματος μπορούν να γραφούν προγράμματα, που εκτελούν αριθμητικές πράξεις με το συνηθισμένο τρόπο:

Πρόγραμμα

```
sumlist([ ],0).
sumlist([I|L],Sum):-sumlist(L,S), Sum is S+I.
```

Αποκοπή

- Προγράμματα Prolog
  - Αναποτελεσματικά, γιατί το δέντρο αναζήτησης είναι αρκετά μεγάλο
- Ενδιαφέρον
  - Ύπαρξη λύσης προβλήματος
  - Όχι για την ποσότητα ή είδος λύσεων

Αποκοπή

- Δεν είναι απαραίτητο ο μηχανισμός εκτέλεσης της Prolog να διασχίσει ολόκληρο το δέντρο αναζήτησης, αλλά ένα μέρος του.
- Ο προγραμματιστής καθοδηγεί το μηχανισμό αναζήτησης προς αποφυγή περιττών αναζητήσεων με τη βοήθεια της αποκοπής (cut) (σύμβολο «!»).

Παράδειγμα 7.6

Έστω ερώτηση της μορφής

```
?- A, B, C.
```

Και το ακόλουθο τμήμα προγράμματος

Πρόγραμμα

$B : -A_1, \cdots, A_k, !, A_{k+1}, \cdots, A_n.$

$B : -D_1, \cdots, D_m.$

$B : -F_1, \cdots, F_k.$

Ρόλος Αποκοπής

- Όταν ο μηχανισμός εκτέλεσης της Prolog περάσει από την αποκοπή, τότε
  - τα $A_1, \cdots, A_k$ δεν πρόκειται να ξαναεξεταστούν για επιπλέον λύσεις
  - οι εναλλακτικές προτάσεις για το B, που υπάρχουν στο πρόγραμμα μετά την πρόταση, που περιέχει την αποκοπή, δεν πρόκειται να εξεταστούν
- Κάποιες λύσεις αποκόπτονται από τη διαδικασία

Παράδειγμα 7.7

Έστω το πρόγραμμα:

Πρόγραμμα

```
father(tom,jim).
male(jim).
son(X,Y):-father(Y,X), male(X).
son(george, jim).
son(john, nick).
```

Και η ερώτηση

```
?-son(S,F).
```

Παράδειγμα 7.7

Ο μηχανισμός της Prolog θα δώσει τρεις λύσεις:

```
{ S=jim, F=tom }

{ S=george, F=jim }

{ S=john, F=nick }
```

Παράδειγμα 7.7

Αν το πρόγραμμα αντικατασταθεί με το:

Πρόγραμμα

```
father(tom,jim).
male(jim).
son(X,Y):-father(Y,X), !, male(X).
son(george, jim).
son(john, nick).
```

Παράδειγμα 7.7

Ο μηχανισμός εκτέλεσης της Prolog δίνει τη λύση

$$\{\texttt{S=jim, F=tom}\}.$$

Οι άλλες δύο λύσεις αποκόπτονται λόγω του συμβόλου !, που δεν επιτρέπει την εξέταση των εναλλακτικών προτάσεων για το `son(S,F)`.

Σχόλια

- Η αποκοπή χρησιμοποιείται συνήθως όταν γνωρίζουμε ότι το πρόβλημα έχει μία και μοναδική λύση και επιθυμούμε τον περιορισμό άσκοπου ψαξίματος. Τα προγράμματα είναι έτσι πιο αποτελεσματικά.
- Η αποκοπή δεν ανήκει στο λογικό τμήμα της Prolog και γι΄ αυτό ενδέχεται να δημιουργήσει δυσνόητα προγράμματα. Λανθασμένη χρήση της περικόπτει λύσεις χρήσιμες και επιθυμητές.