ΕΡΓΑΣΤΗΡΙΟ VII

ΑΠΟΚΟΠΗ

ΜΕΡΟΣ Ι

Η Έννοια της Αποκοπής

Η οπισθοδρόμηση είναι βασικό στοιχείο της Prolog και γίνεται αυτόματα από τη γλώσσα. Σε ορισμένες περιπτώσεις όμως η οπισθοδρόμηση συνεπάγεται σπατάλη χρόνου. Η ***αποκοπή*** «κλαδεύει» το δέντρο υπολογισμού εμποδίζοντας το μηχανισμό οπισθοδρόμησης να ακολουθήσει μονοπάτια που δεν επιθυμούμε.

Τρόπος Λειτουργίας

Ας υποθέσουμε ότι εκτελείται μια πρόταση ενός προγράμματος (πχ *sibling(X,Y):-father(Z,X),father(Z,Y),!,X\=Y*) μιας διαδικασίας (*sibling/2*). Καθώς αξιολογούνται οι επιμέρους προτάσεις όταν φτάνει η Prolog στο σημείο της αποκοπής, υπάρχουν δύο περιπτώσεις:

1. Οι προτάσεις που προηγήθηκαν του συμβόλου της αποκοπής δίνουν λύσεις (είναι συνολικά αληθείς). Σε αυτήν την περίπτωση, η αποκοπή προκαλεί την εκτέλεση των υπόλοιπων υπο-προτάσεων μέχρι το τέλος της γραμμής (τελεία) και η εκτέλεση σταματάει, χωρίς να εξεταστούν τυχόν άλλες προτάσεις του κανόνα.

sibling(X,Y):-father(Z,X),father(Z,Y),!,X\=Y.
~~sibling(X,Y):-father(Z,X),mother(Z,Y),!,X\=Y.~~
~~sibling(X,Y):-mother(Z,X),mother(Z,Y),!,X\=Y.~~
~~sibling(X,Y):-mother(Z,X),father(Z,Y),!,X\=Y.~~

2. Οι κανόνες που βρίσκονται πριν το σύμβολο της αποκοπής στην ίδια πρόταση δεν δίνουν εναλλακτικές λύσεις. Σε αυτήν την περίπτωση η εκτέλεση της πρότασης σταματάει στο σημείο της αποκοπής, αγνοεί ότι έπεται του συμβόλου της αποκοπής, και το πρόγραμμα προχωράει στις επόμενες προτάσεις του κανόνα εάν αυτές υπάρχουν.

sibling(X,Y):- ~~*father(Z,X),father(Z,Y),*~~ *!,X\=Y*

Ιδιαιτερότητες της Αποκοπής

- Όταν ο έλεγχος φτάσει στην αποκοπή τότε ***η αποκοπή ικανοποιείται πάντα.***
- Κάποιες λύσεις είναι πιθανό να αποκόπτονται και να μην εμφανίζονται.

  Π.χ. αν ψάχνουμε μια και μόνη λύση στο πρόβλημα που θέλουμε να λύσουμε, με την αποκοπή γλιτώνουμε πολλές άσκοπες αναζητήσεις, αλλά αν ψάχνουμε όλες τις δυνατές λύσεις είναι πιθανό μερικές να

αποκόπτονται επειδή οι κανόνες πριν την αποκοπή ή οι επόμενες προτάσεις της ίδιας διαδικασίας δεν δίνουν εναλλακτικές λύσεις.
Κατά συνέπεια, ο προγραμματιστής μπορεί να καθοδηγεί το μηχανισμό αναζήτησης προς αποφυγή περιττών αναζητήσεων με τη βοήθεια της αποκοπής. Δηλαδή με τη χρήση της αποκοπής δεν είναι απαραίτητο ο μηχανισμός εκτέλεσης της Prolog να διασχίσει ολόκληρο το δέντρο αναζήτησης, αλλά ένα μέρος του.

- Η αποκοπή δεν ανήκει στο λογικό τμήμα της Prolog και η προσθήκη ή αφαίρεσή της από ένα πρόγραμμα μπορεί να προκαλέσει προβλήματα. Διακρίνουμε δυο περιπτώσεις:

<u>Πράσινη αποκοπή (Green cut)</u>
Δεν αλλάζει τη δηλωτική ερμηνεία του προγράμματος και αν παραληφθεί δεν θα πάρουμε λάθος αποτελέσματα.

Π.χ. έστω το πρόγραμμα *parent/2* που είδαμε στο Εργαστήριο Ι:

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

Στο παραπάνω πρόγραμμα η προσθήκη μιας πράσινης αποκοπής θα μπορούσε να έχει γίνει ως εξής:

```
parent1(X,Y) :- father(X,Y),!.
parent1(X,Y) :- mother(X,Y).
```

Αν ο έλεγχος φτάσει μέχρι την αποκοπή σημαίνει ότι βρήκαμε ότι ο Χ είναι γονέας του Υ και μάλιστα πατέρας του, οπότε δεν χρειάζεται να εξετάσουμε την δεύτερη πρόταση για πιθανές λύσεις. Αν παραληφθεί η αποκοπή τότε το πρόγραμμα *parent1 /2* εξακολουθεί να δίνει σωστά αποτελέσματα.

<u>Κόκκινη αποκοπή (Red cut)</u>
Αλλάζει τη δηλωτική ερμηνεία του προγράμματος και αν παραληφθεί μπορεί και να πάρουμε λάθος αποτελέσματα.

Π.χ. έστω το πρόγραμμα *megisto/3* που βρίσκει το μέγιστο δυο αριθμών:

```
megisto(X,Y,X) :- X>Y.
megisto(X,Y,Y) :- X=<Y.
```

Στο παραπάνω πρόγραμμα η προσθήκη μιας κόκκινης αποκοπής θα μπορούσε να έχει γίνει ως εξής:

```
megisto1(X,Y,X) :- X>Y,!.
megisto1(X,Y,Y).
```

σκεπτόμενοι ότι αν ο έλεγχος φτάσει μέχρι την αποκοπή αυτό σημαίνει ότι *X=<Y* και ο έλεγχος στην δεύτερη πρόταση είναι περιττός. Όμως αν παραληφθεί η αποκοπή, το πρόγραμμα *megisto1 /3* δεν δίνει πλέον σωστά αποτελέσματα

Χρήση της Αποκοπής

Οι συνηθέστερες χρήσεις της αποκοπής είναι οι παρακάτω:

1. ***Σε περιπτώσεις που θέλουμε να δηλώσουμε στην Prolog ότι έχει βρει τη μοναδική σωστή πρόταση για την ικανοποίηση μιας κλήσης.*** Εμποδίζονται έτσι να εξεταστούν για λύση κατά την οπισθοδρόμηση οι υπόλοιπες προτάσεις της διαδικασίας. Με τον τρόπο αυτό επιταχύνεται σημαντικά η εκτέλεση του προγράμματος.

2. ***Για τερματισμό παραγωγής εναλλακτικών λύσεων.*** Στην περίπτωση αυτή η αποκοπή δηλώνει στο σύστημα ότι δεν υπάρχει άλλη ή δεν χρειαζόμαστε άλλη εναλλακτική λύση.

3. ***Για να εξαναγκάσουμε το σύστημα σε αποτυχία. Συνδυασμός αποκοπής και αποτυχίας (fail).*** Στην περίπτωση αυτή δηλώνουμε στο σύστημα ότι αφού έχει φθάσει μέχρι το ! είναι πια γνωστό ότι η κλήση που προκάλεσε αυτή τη διαδρομή δεν αληθεύει και δεν χρειάζεται να προσπαθεί άλλο να την αποδείξει. Εδώ χρησιμοποιούμε μαζί με την αποκοπή το κατηγόρημα αποτυγχάνει ή fail το οποίο αποτυγχάνει πάντα.

4. ***Για να δημιουργήσουμε δομές δεδομένων που υπάρχουν σε κλασικές γλώσσες προγραμματισμού, όπως η γνωστή if-then-else.***

Ορίζουμε τη δομή if_then_else(C,S1,S2): ← αν ισχύει η συνθήκη C τότε κάλεσε την S1, διαφορετικά κάλεσε την S2.

```
if_then_else(C,S1,S2):- C, !, S1.
if_then_else(C,S1,S2):- S2.
```

Τα προγράμματα του εργαστηρίου βρίσκονται στο αρχείο exercise7.pl που θα βρείτε έτοιμο στην διαδρομή: http://iiwm.teikav.edu.gr/digital_lessons/ στο μάθημα Λογική και Λογικός Προγραμματισμός – Ενότητα 7 και θα χρησιμοποιηθούν για να απαντήστε στα παρακάτω ερωτήματα:

ΜΕΡΟΣ ΙΙ

ΓΕΝΙΚΗ ΧΡΗΣΗ ΤΗΣ ΑΠΟΚΟΠΗΣ

Δίνεται το παρακάτω πρόγραμμα :

```
man(jim).
man(tsali).
woman(elsa).
woman(nicki).
woman(sofi).
human(X) :- man(X).
```

human(X) :- woman(X).

Το κατηγόρημα human(X) εκφράζει ότι ο Χ είναι άνθρωπος όταν είναι άνδρας ή ο Χ είναι άνθρωπος όταν είναι γυναίκα.

ΑΣΚΗΣΕΙΣ

1. Διατυπώστε μια ερώτηση στην Prolog έτσι ώστε να σας εμφανίσει ποιοι είναι άνθρωποι (ο κανόνας και τα σχετικά γεγονότα δίνονται στο αρχείο exercise7.pl.
2. Με την παρακάτω προσθήκη στο πρόγραμμα, δίνεται μόνο μία απάντηση στην παραπάνω ερώτηση.

   Η πρόταση *human(X) :- man(X).* Γίνεται *human(X) :- man(X),!.*

   Έτσι η ερώτηση *?-human(X).* θα τυπώσει μόνο *X = jim*, αφού οι εναλλακτικές λύσεις που αφορούν άντρες αποκόπτονται από την Prolog γιατί ο κανόνας *man(X)* βρίσκεται πριν από την αποκοπή, ενώ οι εναλλακτικές λύσεις που αφορούν γυναίκες αποκόπτονται γιατί η πρόταση *human(X) :- woman(X).* βρίσκεται μετά την πρόταση που περιέχει την αποκοπή.

3. Κάντε τις απαραίτητες μετατροπές στο πρόγραμμα ώστε να σας δώσει μόνο μία γυναίκα ως απάντηση στην παραπάνω ερώτηση.

_____

_____

_____

_____

_____

ΜΕΡΟΣ ΙΙΙ

ΑΠΟΚΟΠΗ ΩΣ ΑΡΝΗΣΗ

Η χρήση της αποκοπής σε συνδυασμό με το *αποτυγχάνει (fail)* αποτελεί τη βάση για την υλοποίηση μιας μορφής άρνησης στην Prolog που ονομάζεται «άρνηση σαν αποτυχία» - υπάρχει στη βιβλιοθήκη της Prolog το not(X).

```
oxi(X):- X, !,fail.
oxi(_).
```

ΑΣΚΗΣΗ

Για να ελέγξετε τη λειτουργία του *oxi(X),* εκτελέστε το ερώτημα *oxi(member(1,[m,hellas,3,1])).* στην Prolog και ελέγξτε την αναζήτηση που κάνει για να απαντήσει στην ερώτηση, χρησιμοποιώντας την εντολή trace. Παρατηρείτε κάτι περίεργο και διαφορετικό από ότι περιμένατε μελετώντας τον κανόνα της *oxi(X)* που ορίζεται παραπάνω;

ΜΕΡΟΣ IV

ΑΠΟΚΟΠΗ ΓΙΑ ΠΑΡΑΛΕΙΨΗ ΕΛΕΓΧΩΝ

Η αποκοπή μπορεί να χρησιμοποιηθεί σε ένα πρόγραμμα για να παραλειφθούν κάποιοι έλεγχοι που γίνονται για την επιλογή της κατάλληλης πρότασης. Έστω ότι θέλουμε να ορίσουμε τη σχέση megisto(X,Y,Z) την οποία είδαμε στο Μέρος I και η οποία επιστρέφει στο Z το μεγαλύτερο αριθμό από τους X, Y.

```
megisto(X,Y,X) :- X>Y.
megisto(X,Y,Y) :- X=<Y.
```

Όπως είδαμε, μπορούμε να ορίσουμε την σχέση με την βοήθεια της αποκοπής ως εξής:

```
megisto1(X,Y,X) :- X>Y,!.
megisto1(X,Y,Y).
```

Η χρήση της αποκοπής στον πρώτο κανόνα μας επιτρέπει να παραλείψουμε τον έλεγχο X<=Y. Αυτό γιατί η δεύτερη πρόταση εξετάζεται μόνο αν δεν ισχύει ο έλεγχος του πρώτου κανόνα, διαφορετικά η αποκοπή εμποδίζει τον έλεγχο να φτάσει στη δεύτερη πρόταση. Στη χρήση της αποκοπής σε αυτές τις περιπτώσεις (*Κόκκινες Αποκοπές*) πρέπει να προσέχεται γιατί αν στη σχέση megisto1 παραλειφθεί η αποκοπή τότε αλλάζει το νόημα του προγράμματος.

Έστω ότι έχουμε την παρακάτω συνάρτηση:

$$f(x) = \begin{cases} 0 & \text{αν } x<3 \\ 2 & \text{αν } 3\le x<6 \\ 4 & \text{αν } x\ge 6 \end{cases}$$

Η συνάρτηση αυτή στην Prolog υλοποιείται χρησιμοποιώντας το ακόλουθο

πρόγραμμα:

f(X,0) :- X<3.
f(X,2) :- 3=<X, X<6.
f(X,4) :- X>=6.

Για να μειωθούν οι έλεγχοι που κάνει η Prolog χρησιμοποιούμε την (πράσινη) αποκοπή (green cut) και ξαναορίζουμε την παραπάνω συνάρτηση ως f1 (για να ξεχωρίζει από την προηγούμενη υλοποίηση της).

f1(X,0) :- X<3,!.
f1(X,2) :- 3=<X, X<6,!.
f1(X,4) :- X>=6.

Απλοποιώντας ακόμη περισσότερο την υλοποίηση της συνάρτησης χρησιμοποιούμε κόκκινη αποκοπή (red cut) :

f2(X,0) :- X<3,!.
f2(X,2) :- X<6,!.
f2(X,4).

ΑΣΚΗΣΕΙΣ

1. Τι παρατηρείτε στην αναζήτηση που κάνει η Prolog για να απαντήσει στις ερωτήσεις f(3,Y), f1(3,Y), f2(3,Y), ως προς τον αριθμό των εναλλακτικών λύσεων που εξετάζονται (δένδρο αναζήτησης); (χρησιμοποιήστε την εντολή trace)

2. Δίνεται το πρόγραμμα:

p(a).
p(b) :- !.
p(g).

Εξετάστε ποιες τιμές θα πάρουμε για τις μεταβλητές X και Y στις ερωτήσεις που ακολουθούν :

A) ? p(X).

X = a ;
X = b ;

B) ? p(X), p(Y).

X = a
Y = a ;

X = a
Y = b ;

X = b
Y = a ;

X = b
Y = b ;

Γ) ? p(X), !, p(Y).

ι) Γιατί οι μεταβλητές X και Y δεν παίρνουν ποτέ την τιμή g; (Εξετάστε το ρόλο της αποκοπής στην πρόταση *p(b) :- !.* , λαμβάνοντας υπόψη και τον τρόπο λειτουργίας της αποκοπής.)

ιι) Γιατί στην ερώτηση Γ παίρνουμε μόνο δύο ζευγάρια λύσεων;