Η ΧΡΗΣΗ ΤΗΣ ΤΟΜΗΣ ΣΤΗ ΓΛΩΣΣΑ ΛΟΓΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PROLOG

Αθανάσιος Κ. Τσαδήρας
Επίκουρος Καθηγητής

Τμήμα Πληροφορικής
Α.Τ.Ε.Ι. Θεσσαλονίκης

Περίληψη

Η παρούσα εργασία εξετάζει τόσο την έννοια της τομής στη γλώσσα Λογικού προγραμματισμού (PROLOG) όσο και επιμέρους προβλήματα, προοπτικές και χρήση της.

Λέξεις Κλειδιά
Τομή, PROLOG, Εφαρμογές

1. Εισαγωγή

Η τομή cut(!) είναι ένα ενσωματωμένο κατηγόρημα της Γλώσσας Λογικού Προγραμματισμού PROLOG και μάλιστα ένα από τα πιο ενδιαφέροντα & πολυσυζητημένα [1]-[5]. Συμβολίζεται με το θαυμαστικό και δεν έχει μεταβλητές (arguments).

Η τομή μεταβάλλει τη ροή εκτέλεσης του προγράμματος. Την πρώτη φορά που θα τη συναντήσουμε στην πορεία εκτέλεσης, αυτή πετυχαίνει και η ροή εκτέλεσης δεν μεταβάλλεται. Όταν όμως για κάποιο λόγο την ξανασυναντήσουμε (λόγω οπισθοδρόμησης/backtracking) αυτή αποτυγχάνει, με ότι συνέπειες αυτό συνεπάγεται.

Όλες οι εκλογές-επιλογές-ενοποιήσεις που έχουν γίνει μέχρι, την τομή, 'κοκαλώνουν', 'παγώνουν', ενώ οι εναλλακτικές λύσεις/κανόνες δεν λαμβάνονται υπόψη.

Ας δούμε όμως τι σημαίνουν όλα αυτά με ένα παράδειγμα. Έστω ότι θέλουμε να δούμε αν ισχύει ή όχι, ο στόχος μας (goal) G , ο οποίος δίνεται, από τους εξής δύο κανόνες:

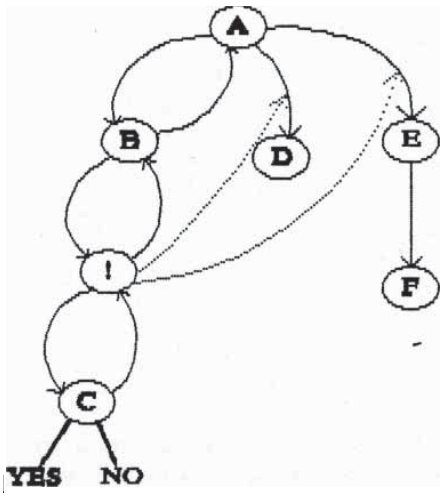
```
G:- A, B, C, !, D, E, F.

G:- K, L, M.
```

Μπορούμε να φανταστούμε την τομή σαν ένα φράχτη. Έτσι, αν περάσουμε κατά την ροή της εκτέλεσης του προγράμματος μας την τομή(φράχτη) από αριστερά προς τα δεξιά, όλες οι ενοποιήσεις που έχουν γίνει, μέχρι, τότε (δηλαδή μέσα στα A,B,C) 'κοκαλώνουν' και δεν μπορούν να μεταβληθούν ούτε μέσω οπισθοδρόμησης.

Για τα D,E,F ισχύει η οπισθοδρόμηση, αν όμως κάποια στιγμή αποτύχει το D, είναι σαν να περνάμε το φράχτη ! από δεξιά προς τα αριστερά. Αυτό έχει σαν συνέπεια να αποτύχει ολόκληρο το G άσχετα αν υπάρχει και άλλος κανόνας για το G.

Γενικά η τομή ελαττώνει το χώρο ψαξίματος, κλαδεύοντας δυναμικά το δέντρο ψαξίματος. Εμποδίζει το ψάξιμο σε κλαδιά στα οποία ο προγραμματιστής ξέρει ότι δεν θα βρει λύσεις (φρούτα).

Σχήμα 1. Συμπεριφορά της τομής κατά την εκτέλεση.



Λύση

Ας δούμε για παράδειγμα το ρόλο της τομής στο πρόγραμμα που ακολουθεί

A:- B, !, C.

A:- D.

A:- E, F.

Θέλοντας να δείξουμε την ισχύ του A, ξεκινάμε από τον πρώτο κανόνα και πάμε να δούμε αν ισχύει το B. Αν δεν ισχύει τότε πάμε στον επόμενο κανόνα, ενώ αν ισχύει, φτάνουμε στην τομή. Τότε στέλνονται 'σήματα' και χάνονται οι σύνδεσμοι του A με τους επόμενους κανόνες (Σχήμα 1). Πλέον δεν υπάρχει περίπτωση να τους χρησιμοποιήσουμε. Ακολουθεί το C που, αν είναι αληθές, βρήκαμε μία λύση ενώ αν δεν είναι, επιστρέφουμε στην τομή. Αποτέλεσμα αυτού είναι να επιστρέψουμε αμέσως στο A και στο σημείο αυτό να αποτύχουμε μια και δεν 'βλέπουμε' άλλους κανόνες.

2. Λόγοι Χρήσης της Τομής

Δύο είναι οι βασικοί λόγοι, χρήσης της τομής:

α) **Οικονομία Χρόνου**. Χρησιμοποιώντας την τομή δεν σπαταλάμε χρόνο προσπαθώντας να βρούμε λύσεις εκεί που ξέρουμε ότι δεν υπάρχουν. Αυτό βέβαια προϋποθέτει γνώσεις από πριν είτε μέσω της λογικής είτε λόγω εμπειρίας. Θα πρέπει, να τονιστεί ότι η χρήση της τομής δεν κάνει μόνο το πρόγραμμα μας πιο γρήγορο αλλά υπάρχουν περιπτώσεις που ένα ! είναι η διαφορά ανάμεσα σε ένα πρόγραμμα που τρέχει, και ένα που δεν τρέχει. Η χρήση της τομής κάνει τον αλγόριθμο μας πια αποτελεσματικό. Για παράδειγμα δεν υπάρχει λόγος οπισθοδρόμησης σε πρόγραμμα που βρίσκει το ελάχιστο μεταξύ δύο αριθμών. Εφόσον βρούμε μία λύση αυτή θα είναι και μοναδική και μία μετέπειτα οπισθοδρόμηση δεν θα τη μεταβάλει. Με την τομή σταματάμε την οπισθοδρόμηση και δεν σπαταλάμε χρόνο άδικα.

β) **Οικονομία Χώρου Μνήμης του Υπολογιστή**. Αυτό γίνεται λόγω του ότι μέσω της τομής δεν χρειάζεται να κρατάμε σημεία οπισθοδρόμησης στα οποία πιθανόν αργότερα θα έπρεπε να επιστρέψουμε και να ελέγξουμε. Έτσι μένει ελεύθερος κάποιος χώρος στη μνήμη που μπορεί να είναι ζωτικής σημασίας για μία ογκώδη εφαρμογή μας.

3. Περιπτώσεις που χρησιμοποιούμε τη Τομή.

Υπάρχουν τρεις ομάδες περιπτώσεων στις οποίες χρησιμοποιούμε τη τομή:

Πρώτη Ομάδα. Περιπτώσεις όπου ξέρουμε ότι, για το συγκεκριμένο στόχο πρέπει να χρησιμοποιήσουμε το συγκεκριμένο κανόνα. Είναι σαν να λέμε στο υπολογιστή: 'Κοίταξε, αν έφτασες ως εδώ, έχεις διαλέξει το σωστό κανόνα και αν υπάρχει λύση(ή λύσεις), αυτή (ή αυτές) θα δίνονται από το συγκεκριμένο κανόνα', μην χάσεις χρόνο ψάχνοντας άλλο κανόνα.

Σε αυτές τις περιπτώσεις το ! μπαίνει είτε στην αρχή του κανόνα είτε ανάμεσα στους διάφορους υποστόχους δηλαδή:

A: — !,A1,..., An .

A:— A1,..., Ai, ! , Ai+1 , ..., An.

Βέβαια το να μπαίνει η τομή στην αρχή ενός κανόνα, μπορεί να προκαλέσει σε κάποιον αμφιβολίες, μια και μπορεί να διερωτηθεί πώς ξέρουμε ότι σίγουρα αυτός είναι ο κανόνας που πρέπει να χρησιμοποιήσουμε. Η απάντηση είναι ότι το ξέρουμε μέσω κάποιων πληροφοριών που βρίσκονται στην κεφαλή του κανόνα. Αυτό φαίνεται καθαρά στο παράδειγμα που ακολουθεί.

```
salary(Job, graduated, Sum):- !,
                    weight(Job,X),
                    Sum is X*1200.

salary(Job,_, Sum):-
                weight(Job,X),
                Sum is X*1000.
```

Με το παραπάνω πρόγραμμα βρίσκουμε το μισθό κάποιου, ξέροντας τη δουλειά που κάνει και το αν είναι απόφοιτος ανώτερης ή ανώτατης σχολής. Αν είναι απόφοιτος τότε το 'βάρος' της εργασίας που κάνει, πολλαπλασιάζεται με το 1200, ενώ στην αντίθετη περίπτωση πολλαπλασιάζεται με το 1000. Η τομή βρίσκεται στην αρχή του κανόνα δηλώνοντας ότι εφόσον φτάσαμε μέχρι εκεί, ο μισθωτός είναι απόφοιτος και σίγουρα πρέπει να χρησιμοποιήσουμε τον πρώτο (και μόνο) κανόνα. Με το να περάσουμε την τομή από αριστερά προς τα δεξιά δεν αφήνουμε σημείο οπισθοδρόμησης για το δεύτερο κανόνα (είναι πλέον σαν να μην υπάρχει), κάτι που επιθυμούμε ώστε σε περίπτωση οπισθοδρόμησης να μην βρούμε και δεύτερο ποσό μισθού μέσω του δεύτερου κανόνα.

Ας δούμε ακόμα ένα παράδειγμα με τη τομή ενδιάμεσα αυτή την φορά.

```
employee(Name):-person(Name,Age),
                    male(Name),
                    !,
                    Age<32.
```

```
employee(Name):-person(Name,Age),
                female(Name),
                !,
                Age<28.
```

Εδώ ελέγχουμε το αν ένα άτομο person(Name,Age) τηρεί τις προϋποθέσεις για να γίνει υπάλληλος κάποιας εταιρείας. Υπάρχουν δύο κανόνες, ένας για τις γυναίκες και ένας για τους άνδρες. Η εταιρία δέχεται υπαλλήλους, άνδρες κάτω των 32 χρονών, ενώ γυναίκες κάτω των 28. Η τομή μπαίνει αμέσως μετά το φύλο μια και αυτό καθορίζει ποιο κανόνα θα χρησιμοποιήσουμε. Ξέροντας το φύλο ξέρουμε ποιο κανόνα (και μόνο) πρέπει να χρησιμοποιήσουμε οπότε μπορούμε να ξεχάσουμε τον άλλο. Πράγματι, αν ο Γιώργος είναι 34 ετών, βάση του πρώτου κανόνα (και αφού έχουμε περάσει την τομή) δεν μπορεί να γίνει υπάλληλος. Μετά την αποτυχία στο σημείο Age<32 επιστρέφουμε στη τομή με αποτέλεσμα να αποτύχει όλος ο στόχος μας χωρίς να εξετασθεί ο δεύτερος κανόνας(Θα εξεταζόταν άδικα).

Ίσως παρατηρήσατε ότι η τομή στο δεύτερο κανόνα ουσιαστικά δεν χρειάζεται (μια και δεν υπάρχει τρίτος κανόνας) αλλά προστίθεται για λόγους ομοιομορφίας με το πρώτο κανόνα και για να μην υπάρξει πρόβλημα στη περίπτωση μελλοντικής αλλαγής της σειράς των δύο κανόνων.

Θα πρέπει να πούμε ότι η τομή μπορεί να αντικατασταθεί από το not. Ας δούμε αυτή τη μετατροπή στο ακόλουθο παράδειγμα:

```
A:— B,!,C          A:-B, C.
A:- D.             A:- not(B),D.
```

Τα δύο αυτά προγράμματα δίνουν το ίδιο αποτέλεσμα με το δεύτερο να είναι πιο ευανάγνωστο και πιο κατανοητό. Έτσι η αντικατάσταση του ! με not είναι μια σωστή προγραμματιστική συνήθεια. Βέβαια η χρήση του not κάνει το πρόγραμμα μας πιο αργό αφού πρέπει στο δεύτερο κανόνα να ελέγξουμε για δεύτερη φορά την ισχύ του B. Αυτό είναι ιδιαίτερα χρονοβόρο σε περιπτώσεις όπου το B είναι πολύπλοκο, ή έχουμε συχνή κλήση της ρουτίνας A.

Δεύτερη Ομάδα. Περιπτώσεις όπου ξέρουμε από πριν ότι ο στόχος μας θα αποτύχει ή θέλουμε να αποτύχει οπότε δεν χρειάζεται να κοιτάξουμε εναλλακτικές λύσεις (άλλους κανόνες). Είναι σαν να λέμε στο υπολογιστή: 'Κοίταξε, αφού έφτασες μέχρι εδώ δεν χρειάζεται να προσπαθήσεις να βρεις λύση, γιατί απλούστατα δεν υπάρχει'. Σε αυτές τις περιπτώσεις προσθέτουμε ένα !,fail στο τέλος του κανόνα ('negation as failure' ή 'cut-fail combination' στη διεθνή ορολογία). Αυτή η μέθοδος είναι κλασσική για να δηλώσουμε εξαιρέσεις. Ας δούμε όμως ένα παράδειγμα.

```
mayor(Man):- foreign(Man),
                !, fail.
mayor(Man):- person(Man, Age),
                Age < 20,
                ! , fail.
mayor(Man):- never_quilt(Man).
```

Εδώ ψάχνουμε να βρούμε αν κάποιος Man μπορεί να δηλώσει υποψηφιότητα για δήμαρχος. Με τους δύο πρώτους κανόνες δηλώνουμε δύο εξαιρέσεις. Αν είναι αλλοδαπός προφανώς δεν έχει το δικαίωμα να γίνει υποψήφιος και δεν υπάρχει λόγος να ψάχνουμε (με οπισθοδρόμηση) τους άλλους κανόνες. Το ίδια συμβαίνει και όταν η ηλικία του είναι μικρότερη των 20 ετών. Για να δούμε όμως πως συμβαίνει η αποτυχία. Τη πρώτη φορά που συναντάμε το ! αυτό πετυχαίνει, όμως ακολουθεί το fail και έτσι επιστρέφουμε στο !. Τώρα όμως, όπως έχουμε πει, αποτυγχάνει όλος ο στόχος μας χωρίς να εξετασθούν οι υπόλοιποι κανόνες.

Πάλι μπορούμε να αντικαταστήσουμε το ! με not κάνοντας το πρόγραμμα λιγότερο δυσνόητο όπως μπορούμε να δούμε.

```
mayor(Man):- person (Man, Age),
                not(foreign (Man)),
                not (Age < 20),
                never_quilty (Man).
```

Τρίτη Ομάδα. Περιπτώσεις όπου δεν θέλουμε να βρούμε άλλες πιθανές λύσεις, είτε γιατί μας αρκεί η μία, είτε γιατί

ξέρουμε ότι υπάρχει μόνο μία λύση (όποτε για ποιο λόγο να ψάχνουμε για άλλη).Είναι σαν να λέμε στον υπολογιστή: 'Κοίταξε, αφού έφτασες μέχρι εδώ έχεις βρει ήδη μία λύση και δεν υπάρχει λόγος να συνεχίσεις για άλλη λύση'.

Τώρα το ! μπαίνει πάντα στο τέλος του κανόνα και έτσι ονομάζεται ουραία τομή (tail cut) δηλαδή είναι της μορφής:

A:—A1,…,An, !.

Είναι φανερό ότι φτάνοντας στο ! έχουν ικανοποιηθεί όλοι οι προηγούμενοι υποστόχοι μας και έτσι σίγουρα έχουμε βρει μία λύση. Η τομή τοποθετείται για να μην εξετάσουμε τους υπόλοιπους κανόνες στην περίπτωση οπισθοδρόμησης. Συχνή χρήση τέτοιων τομών υπάρχει στα προσδιοριστικά προβλήματα.

Παραδείγματα που μας αρκεί μία λύση είναι π.χ. το να ψάχνουμε το αν το στοιχείο a βρίσκεται μέσα στη λίστα L. Δεν μας ενδιαφέρει αν υπάρχει μία ή περισσότερες φορές, μας αρκεί να βρίσκεται έστω μία φορά. Ομοίως στο σαχ στο σκάκι. Έστω ένα πιόνι να απειλεί το βασιλιά έχουμε σαχ (και δεύτερο να υπάρχει δεν μας ενδιαφέρει μια και έχουμε ήδη σαχ) .

Παράδειγμα στο οποίο φαίνεται ότι θέλουμε μόνο μία λύση αφού ξέρουμε ότι υπάρχει μόνο μία, είναι η ρουτίνα που μας δίνει την απόλυτη τιμή ενός αριθμού. Μια άμεση προσέγγιση αυτού του προβλήματος είναι το ακόλουθο πρόγραμμα:

```
abs(X, X):- X >= O.
abs(X, Y):- X < O, Y is -1*X.
```

Στη περίπτωση μη αρνητικού αριθμού χρησιμοποιούμε τον πρώτο κανόνα. Αν όμως ζητήσουμε δεύτερη απόλυτη τιμή (μέσω οπισθοδρόμησης) θα εξετασθεί, ανεπιτυχώς βέβαια, ο δεύτερος κανόνας ξοδεύοντας έτσι άδικα χρόνο. Επομένως μία βελτίωση αυτού του προγράμματος είναι το εξής πρόγραμμα:

```
abs(X, X):- X >= O, !.
abs(X, Y):- X < O, Y is -1*X.
```

Σε αυτό έχει προστεθεί μία τομή στο τέλος του πρώτου κανόνα. Τώρα στη περίπτωση μη αρνητικού αριθμού πάλι θα πετύχει ο πρώτος κανόνας και πετυχαίνοντας, η τομή σβήνει το σημείο οπισθοδρόμησης για το δεύτερο κανόνα που έτσι, τώρα δεν υπάρχει πλέον περίπτωση ούτε καν να εξετασθεί.

4. Υλοποίηση not, if_then_else.

Η τομή μας είναι χρήσιμη και για τον επιπρόσθετο λόγο ότι μέσω αυτής υλοποιούνται διάφορα κατηγορήματα της PROLOG, όπως το not και το if_then_else [3],[4]. Ας δούμε πρώτα το not και τους δύο κανόνες που το υλοποιούν:

```
not(P):- call (P) , !, fail, not(P).
```

Το not ξέρουμε ότι το χρησιμοποιούμε όταν θέλουμε να δείξουμε ότι κάτι δεν ισχύει. Έτσι με τον πρώτο κανόνα, βλέπουμε αν ισχύει το P και αν ισχύει τότε το not αποτυγχάνει αφού ακολουθεί cut-fail combination (δεν ελέγχεται ο δεύτερος κανόνας). Αν το P δεν ισχύει τότε πάμε στον δεύτερο κανόνα και τώρα το not πετυχαίνει (όπως και θα έπρεπε). Θα πρέπει να τονισθεί η έννοια που έχει το not στην PROLOG. Αν ισχύει το not(P) αυτό δεν δηλώνει ότι ισχύει το αντίθετο του P αλλά απλά ότι σύμφωνα με τη βάση γνώσης που έχει την συγκεκριμένη στιγμή η PROLOG το P δεν ισχύει. Έτσι αν είναι αληθές το not(woman(mary)), δεν σημαίνει ότι η mary δεν είναι γυναίκα (δηλαδή είναι άνδρας) αλλά ότι δεν έχουμε δηλώσει ότι η mary είναι γυναίκα (κρίνει σύμφωνα με το κόσμο που του έχουμε ορίσει).

Το if_then_else υλοποιείται και αυτό με δύο κανόνες:

```
if0then0else(P, Q, R):-P, !, Q.
if0then0else(P, Q, R):-R.
```

Με τον πρώτο κανόνα, αν ισχύει το P τότε φτάνουμε στην τομή οπότε πρέπει να ξεχάσουμε το δεύτερο κανόνα, και εκτελούμε το Q (εάν P τότε Q). Εάν δεν ισχύει το P τότε πάμε στο δεύτερο κανόνα και εκτελείται το R (διαφορετικά—else R).

5. Διαχωρισμός σε Πράσινες και Κόκκινες τομές (Green & Red Cuts)

Οι πράσινες τομές (green cuts) είναι 'αθώες' τομές, με την έννοια ότι δεν μεταβάλλουν τον δηλωτικό τρόπο ερμηνείας του προγράμματος [5], [6]. Είναι οι τομές που χρησιμοποιούμε απλά για να κάνουμε το πρόγραμμα μας πιο γρήγορο και δεν αλλάζουν τον αριθμό και το είδος των λύσεων που θα παίρναμε ακόμη και αν δεν τις χρησιμοποιούσαμε. Οι τομές αυτές δηλώνουν ντετερμινισμό-προσδιοριστικότητα και με την απομάκρυνση τους από το πρόγραμμα δεν αλλάζουν την έννοια—νόημα του προγράμματος (δεν παρατηρούμε καμία μεταβολή στην συμπεριφορά του προγράμματος από την άποψη των λύσεων). Μπορούμε να τις αγνοήσουμε όταν πάμε να 'διαβάσουμε' με δηλωτικό τρόπο το πρόγραμμά μας. Έτσι πράσινη είναι η τομή στο πρόγραμμα που ακολουθεί και δίνει τον μικρότερο αριθμό μεταξύ δύο αριθμών:

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

Οποιαδήποτε ερώτηση και αν κάνουμε για να βρούμε τον ελάχιστο δεν μπορούμε να καταλάβουμε αν χρησιμοποιήσαμε το παραπάνω πρόγραμμα ή το αντίστοιχο χωρίς την τομή. Η τομή απλώς κάνει το πρόγραμμα αυτό πιο γρήγορο.

Ας δούμε όμως και το παρακάτω πρόγραμμα:

```
min(X,Y,X):- X =< Y,!. min(X,Y,Y).
```

Φαίνεται πολύ λογικό να μην ελέγξουμε το X > Y αφού για φτάσουμε στο δεύτερο κανόνα σίγουρα δεν ισχύει X =<

Υ. Αυτό όμως δεν σωστό. Σε μία ερώτηση ?-min(5, 9, 9). θα πάρουμε εσφαλμένα την απάντηση yes. Τι έγινε;

Το παραπάνω παράδειγμα είναι παράδειγμα χρήσης κόκκινης τομής (red cut)· θα μπορούσαμε να πούμε ότι κόκκινες τομές είναι οι μη πράσινες. Ένας πιο σαφής ορισμός είναι ότι είναι οι τομές που επηρεάζουν το δηλωτικό τρόπο ερμηνείας του προγράμματος. Στο παραπάνω πρόγραμμα δεν μπορούμε να πάρουμε τον κάθε κανόνα ξεχωριστά και να τον ερμηνεύσουμε με δηλωτικό τρόπο (που είναι και ένα από τα μεγαλύτερα πλεονεκτήματα της PROLOG) και έτσι δεν μπορούμε να αγνοήσουμε αυτές τις τομές όταν διαβάζουμε το πρόγραμμα. Η απομάκρυνση μίας κόκκινης τομής αλλάζει δραματικά το νόημα του προγράμματος και μεταβάλει το σύνολο των στόχων που μπορούν να επαληθευτούν μέσω του προγράμματος. Επίσης οι κόκκινες τομές κάνουν το πρόγραμμα μας δυσανάγνωστο και δυσνόητο.

Θέλοντας να επιμείνουμε σε αυτό τον κρίσιμο διαχωρισμό ας επαναλάβουμε το πρόγραμμα που δίνει την απόλυτη τομή ενός αριθμού.

```
abs(X, X):- X >= 0, !.
abs(X, Υ):- X < 0, Υ is -1*X.
```

Εδώ n τομή είναι πράσινη αφού αν την απομακρύνουμε δεν μεταβάλουμε τις λύσεις που θα μας έδινε. Επίσης οι δύο κανόνες μπορούν να διαβαστούν ανεξάρτητα.

Κάποιος, βλέποντας το επόμενο πρόγραμμα θα μπορούσε να το θεωρήσει ως βελτίωση του προηγούμενου.

```
abs(X, X):- X >= 0, !.
abs(X, Υ):- Υ is -1*X.
```

Τώρα δεν υπάρχει ο έλεγχος X < 0 στο δεύτερο κανόνα αφού για να φτάσουμε στο δεύτερο κανόνα σίγουρα X < 0. Αυτό το λέμε έχοντας στο μυαλό του ερωτήσεις του τύπου ?-abs(-5, X). Σε αυτό το είδος ερωτήσεων το δεύτερο πρόγραμμα δεν παρουσιάζει προβλήματα. Αν όμως θέλουμε να ρωτήσουμε ποιοι αριθμοί έχουν σαν απόλυτη τιμή τον αριθμό π.χ. 7, τότε δεν λειτουργεί σωστά. Η τομή είναι κόκκινη και όπως βλέπουμε περιορίζει τη γενικότητα χρήσης του προγράμματός μας.

6. Η Κατάλληλη Θέση της Τομής

Η δυσκολία που συναντούν οι περισσότεροι χρήστες της PROLOG, στη χρήση της τομής είναι ότι δεν ξέρουν το σημείο που πρέπει να τοποθετήσουν την τομή. Η απάντηση είναι η εξής: 'Τοποθετείται στο σημείο στο οποίο είμαστε απόλυτα βέβαιοι ότι μπορούμε να απορρίψουμε τους άλλους κανόνες, ξέροντας ότι αν υπάρχει λύση αυτή θα δίνεται από τον κανόνα στον οποίο ήδη βρισκόμαστε'.

Δεν πρέπει να τοποθετούμε την τομή πολύ αργά γιατί έτσι αποκλείουμε κάποιες πιθανές λύσεις. Αυτό συμβαίνει γιατί, όπως είδαμε, με την τομή 'παγώνουν' οι μέχρι την τομή ενοποιημένες μεταβλητές, με άμεση συνέπεια να μην μπορούν να αλλάξουν τιμή. Έτσι αποκλείουμε κάποια λύση που θα μπορούσε να υπάρξει αν κάποια 'παγωμένη' μεταβλητή είχε άλλη τιμή.

Ούτε πρέπει όμως να τοποθετούμε την τομή πολύ νωρίς, γιατί έτσι επιβαρύνουμε την Prolog με άσκοπες-άστοχες εξερευνήσεις υποβαθμίζοντας έτσι τη αξία της τομής στο πρόγραμμα μας. Οι εξερευνήσεις αυτές θα γίνονται άσκοπα γιατί μια μεταβλητή που θα έπρεπε να είχε 'παγώσει' δεν πάγωσε και έτσι εξετάζουμε άσκοπα αν υπάρχει λύση για όλες τις τιμές που μπορεί να πάρει η μεταβλητή.

Για το λόγο αυτό, καλά θα ήταν να προσέχουμε ιδιαίτερα τη θέση της τομής σε κάποιο κανόνα. Μια εύκολη λύση είναι να ξεκινήσουμε από την αρχή του κανόνα και 'νοητά' να τοποθετούμε την τομή στη πρώτη θέση που μπορεί να μπει ελέγχοντας τα αποτέλεσμα της χρήσης της. Αν δεν είναι τα επιθυμητά τότε πάμε στη δεύτερη, στη συνέχεια στην τρίτη κ.ο.κ. ώσπου να φτάσουμε στα επιθυμητά. Σε αυτό το σημείο θα μπορούμε να πούμε ότι πρέπει, να χρησιμοποιήσουμε το συγκεκριμένο κανόνα και κανένα άλλο.

Ας δούμε όμως ένα παράδειγμα, χρησιμοποιώντας ένα πρόγραμμα που ήδη αναφέραμε.

```
employee(Name):- person(Name,Age),
                 male(Name),
                 Age<32.
```

```
employee(Name):- person(Name,Age),
                 female(Name),
                 Age<28.
```

Εδώ ψάχνουμε να βρούμε αν κάποιος (ή το σύνολο των ατόμων, αν το Name δοθεί σαν μεταβλητή) μπορεί να γίνει υπάλληλος. Πού πρέπει να τοποθετήσουμε την τομή;

Θα ακολουθήσουμε την τεχνική που αναφέραμε. Πάμε στον πρώτο κανόνα και η πρώτη θέση της τομής είναι πριν το person(Name,Age). Δεν είναι η κατάλληλη θέση γιατί με το να φτάσουμε στην τομή δεν είμαστε σίγουροι ότι πρέπει να χρησιμοποιήσουμε το συγκεκριμένο κανόνα. Αν το τοποθετήσουμε μετά το person(Name,Age), πάλι δεν ξέρουμε ότι βρισκόμαστε στον κατάλληλο κανόνα. Η επόμενη θέση είναι μετά το male(Name). Τώρα είμαστε βέβαιοι ότι εφόσον φτάσαμε μέχρι εδώ το άτομο είναι άνδρας και σίγουρα αν είναι άνδρας τότε πρέπει να χρησιμοποιήσουμε τον πρώτο κανόνα. Έτσι, καταλήγουμε ότι αυτή είναι η ενδεδειγμένη θέση της τομής. Αν συνεχίσαμε και βάζαμε την τομή στην επόμενη θέση, τότε αυτό θα ήταν ουραία τομή, με αποτέλεσμα, όπως έχουμε πει, να τερματίσουμε στην πρώτη λύση.

7. Τοπικότητα της Τομής

Έστω έχουμε το εξής πρόγραμμα:

```
A:- B, C, D.
C:- K, L, !, M, Q, R.
C:- N, O.
```

Θέλοντας να δούμε αν ισχύει το A, θα πρέπει να δείξουμε αν ισχύουν τα B,C,D. Έστω ότι το B ισχύει. Συνεχίζοντας ελέγχουμε την ισχύ του C. Θα πρέπει να χρησιμοποιήσουμε το δεύτερο και τρίτο κανόνα. Αν στο δεύτερο κανόνα ισχύον τα K,L τότε φτάνουμε στην τομή η οποία μας φέρνει τα εξής αποτελέσματα:

i) Τα K,L 'παγώνουν, κοκαλώνουν',
ii) Ο τρίτος κανόνας χάνεται (δεν υπάρχει περίπτωση να τον χρησιμοποιήσουμε).

iii) Οπισθοδρόμηση επιτρέπεται πια, μόνο στο τμήμα M,Q,R.

Αν αποτύχουμε στο τμήμα K,L τότε μπορούμε να πάμε και να ελέγξουμε τον τρίτο κανόνα.

Το σημαντικό στοιχείο σε όλα τα παραπάνω είναι ότι η τομή δεν είναι φανερή στο A αλλά κυρίως στο B. Ότι, και να συμβεί στο C, το B δεν παγώνει, και έτσι μπορούμε να έχουμε οπισθοδρόμηση σε αυτό. Συμπεραίνουμε ότι, η τομή έχει τοπικό χαρακτήρα.

Εδώ μπορούμε να δώσουμε μιαν απάντηση σε κάποιον που για ορισμένους λόγους συμπτύσσει το παραπάνω πρόγραμμα στο επόμενο (αντικαθιστά το C με τους κανόνες του):

```
A:- B, (K, L, !, M, Q, R), D.
A:- B, N, O, D.
```

Δεν φαίνεται, να διαφέρει λογικά από το προηγούμενο. Ας προσέξουμε όμως την τομή. Όταν φτάσουμε σε αυτήν θα κοκαλώσουν όχι μόνο τα K,L αλλά και το B. Επομένως τα δύο προγράμματα δεν είναι ισοδύναμα και αν θέλουμε να μην παγώνει το B (κάτι που συμβαίνει πολύ συχνά) τότε πρέπει να χρησιμοποιήσουμε το πρώτο πρόγραμμα.

8. Λόγοι ύπαρξης δύο τομών στο ίδιο κανόνα

Υπάρχει η λαθεμένη εντύπωση στους προγραμματιστές της PROLOG ότι δεν υπάρχει λόγος ύπαρξης δύο (ή περισσότερων) τομών στον ίδιο κανόνα π.χ. A:- B, !, C, D,!. θεωρώντας ότι κατά κάποιο τρόπο η δεύτερη τομή αναιρεί την πρώτη. Αυτό είναι λάθος. Υπάρχουν τέτοιες περιπτώσεις ανάγκης χρήσης δύο τομών, όμως πρέπει να ομολογήσουμε ότι είναι αρκετά εξεζητημένες. Θα μπορούσαμε να πούμε απλώς ότι π.χ. στο παράδειγμα μας, οι δύο τομές λειτουργούν σαν συνδυασμός του ότι επιλέξαμε τον κατάλληλο κανόνα και ότι ζητούμε (ή ότι υπάρχει) μόνο μία λύση.

9. Πλεονεκτήματα και Μειονεκτήματα της χρήσης της Τομής

Τα πλεονεκτήματα της χρήσης της τομής, όπως ήδη είδαμε, είναι αρκετά. Με την χρήση της έχουμε οικονομία τόσο σε χρόνο όσο και σε χώρο μνήμης. Επίσης υλοποιούμε κατηγορήματα όπως το not και το if _then__else αυξάνοντας έτσι τη ισχύ της γλώσσας.

Έχουμε και σημαντικά μειονεκτήματα όπως το ότι έχουμε δύσκολη αποσφαλμάτωση (debugging) και συχνά πολλές παρενέργειες (side effects).

Γενικά με τη τομή αυξάνουμε την αποτελεσματικότητα του αλγορίθμου με τίμημα την ευελιξία. Χάνουμε τις πολλαπλές χρήσεις που συνήθως μας δίνει ένα πρόγραμμα σε PROLOG, χάνοντας έτσι την ευελιξία που θα μας έδινε το δίχως τομές (cut-free) ισοδύναμο πρόγραμμα.

Επίσης χάνουμε το πλεονέκτημα του δηλωτικού τρόπου ερμηνείας των κανόνων. Εξάλλου ξέρουμε ότι σε ένα πρόγραμμα χωρίς τομές, η σειρά των κανόνων δεν επηρεάζει το σύνολο των λύσεων του προβλήματος. Αν όμως χρησιμοποιήσουμε τομές τότε η σειρά των κανόνων παίζει βασικό ρόλο και μπορεί να επιφέρει αλλαγή στο σύνολο των λύσεων.

Ένα χαρακτηριστικό στη χρήση της τομής είναι ότι δεν υπάρχουν σίγουρα σωστά αποτελέσματα όταν κάνουμε ερωτήσεις (βάζουμε στόχους) διαφορετικού τύπου από αυτές για τις οποίες φτιάξαμε το πρόγραμμα.

Αυτό γίνεται φανερό στο πρόγραμμα που ακολουθεί:

```
shoes(adidas, 100):- !.
shoes(nike, 120):- !.
shoes(0, 80).
```

Αυτό το πρόγραμμα το φτιάξαμε για να μας δίνει την τιμή πώλησης κάποιων αθλητικών παπουτσιών. Τα adidas πωλούνται 100€, τα nike 120€ ενώ τα υπόλοιπα 80€. Έτσι αν ρωτήσουμε πόσο κοστίζουν τα nike δηλαδή ?-shoes(nike,X). η απάντηση που θα πάρουμε είναι X=120 ενώ αν ζητήσουμε άλλη λύση θα πάρουμε no δηλαδή ότι δεν υπάρχει άλλη λύση. Τι θα γίνει όμως αν κάνουμε ερώτηση άλλου τύπου και ρωτήσουμε αν τα nike κοστίζουν 80 (?-shoes(nike,80).); Η απάντηση είναι Yes!!!.

Πράγματι αυτό ισχύει λόγω του τελευταίου κανόνα και δεν είναι αυτό που περιμέναμε. Ο λόγος που έγινε αυτό είναι ότι με τις τομές αλλοιώθηκε η γενικότητα του προγράμματος μας και τα αποτελέσματα δεν είναι τα σωστά για όλες τις χρήσεις του προγράμματος. Εδώ απλώς θα αναφέρουμε ότι μια λύση για το παραπάνω πρόβλημα θα ήταν να μετατρέψουμε τους κανόνες στην μορφή shoes(nike,Price):-!,Price = 120.

Θα μπορούσαμε να προσθέσουμε ότι οι τομές σταματούν τη μη προσδιοριστικότητα. Όμως όπως ξέρουμε η μη προσδιοριστικότητα είναι ένα από τα βασικά χαρακτηριστικά της PROLOG και χάνοντας την, κάνει την PROLOG να μοιάζει με όλες τις άλλες γλώσσες. Για το λόγο αυτό η τομή υιοθετήθηκε με σκεπτικισμό από τους ιδρυτές της γλώσσας. Δεν πρέπει όμως να ξεχνούμε ότι βοηθά σε προβλήματα που είναι από την φύση τους προσδιοριστικά-ντετερμινιστικά.

10. Πιθανά Λάθη - Συμβουλές

Τα πιο πιθανά λάθη των προγραμματιστών στην χρήση της τομής είναι:

Πρώτον: να υποθέσουν (νομίζουν) ότι με τη τομή κόβονται κλαδιά τα οποία δεν κόβονται, με άμεση συνέπεια να ψάχνουμε σε αυτά τα κλαδιά άσκοπα και να χάνουμε χωρίς λόγο χρόνο.

Δεύτερο: να υποθέσουν (πιστέψουν) ότι δεν κόβουν κάποια κλαδιά ενώ στην πραγματικότητα τα κόβουν, με άμεση συνέπεια να χάνονται κάποιες λύσεις τις οποίες αναζητούν.

Γενικά, οι προγραμματιστές με την χρήση της τομής γίνονται, επιρρεπείς στα λάθη. Γι αυτό η χρήση της θα πρέπει, να αποφεύγεται όσο γίνεται, ιδίως από τους αρχάριους. Σαν εναλλακτική λύση προτείνεται, η χρήση του not που μπορεί να κάνει το πρόγραμμα πιο αργά, αλλά είναι, όπως είπαμε, πιο ευανάγνωστο (κατανοητό). Αυτά συνιστάται ιδίως σε περιπτώσεις όπου έχουμε not σε απλές σχέσεις π.χ. not(nationality(greek)) μια και η μείωση της αποτελεσματικότητας είναι πολύ μικρή σε σχέση με αυτό που κερδίζουμε στην ανάγνωση-κατανόηση του αλγορίθμου.

Αναφορές / Βιβλιογραφία

[1] W.F. Clocksin & C.S. Melish, 'Programming in Prolog', Spinger-Verlag.
[2] L. Sterlinh & Eh. Shapiro, 'The Art of Prolog', MIT Press, 1988.
[3] Ivan Bratko, 'Prolog- Programming for Artificial Intelligence', Addison Wesley.
[4] H. Cohlho, J. Cotta, L. Periera, 'How to solve it with Prolog', 3rd edition, 1982.
[5] Σπ. Ξανθάκης, 'Prolog – Τεχνικές Προγραμματισμού', Εκδόσεις Νέων Τεχνολογιών, 1990.
[6] J. Loyds, 'Foundation of Logic Programming' Spinger-Verlag. 1984.
[7] R.A. O'Keefe, 'Programming Meta_logical Operations In Prolog', DAI, Univ. of Edinburgh, 1983.