7. ΕΛΕΓΧΟΣ ΤΗΣ ΔΙΑΔΙΚΑΣΙΑΣ ΕΚΤΕΛΕΣΗΣ - ΑΠΟΚΟΠΗ

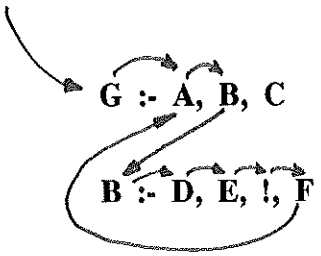
Οπως έχουμε αναφέρει στην §3, κατά τη διάρκεια της εκτέλεσης μιάς ερώτησης, εάν κάποιος υποστόχος της αποτύχει, η PROLOG οπισθοδρομεί αυτόματα στον προηγούμενο υποστόχο και προσπαθεί να βρεί εναλλακτικό τρόπο ικανοποίησης του. Είναι δυνατόν να παρακάμψουμε την αυτόματη αυτή διαδικασία της οπισθοδρόμησης, εάν χρησιμοποιήσουμε ένα ειδικό κατηγόρημα που ονομάζεται αποκοπή.

Η **αποκοπή** (***cut***) αποτελεί το πιό αμφισβητήσιμο ενσωματωμένο κατηγόρημα της PROLOG πού χρησιμοποιείται για τον έλεγχο της διαδικασίας εκτέλεσης του προγράμματος. Η αποκοπή παριστάνεται με το σύμβολο του θαυμαστικού "**!**" (κατηγόρημα με πληθικό αριθμό 0) και μπορεί να εισαχθεί στο σώμα κάποιας φράσης όπως ένα οποιοδήποτε κατηγόρημα:

$$\mathbf{A} \text{ :- } \mathbf{B_1}, \ldots, \mathbf{B_{i-1}}, !, \mathbf{B_{i+1}}, \ldots, \mathbf{B_n}$$

Στη διάρκεια της εκτέλεσης όταν συναντάται σαν στόχος για πρώτη φορά πετυχαίνει. Οταν όμως η εκτέλεση επιστρέψει λόγω οπισθοδρόμησης (αποτυχία του στόχου $\mathbf{B_{i+1}}$) στην αποκοπή, το αποτέλεσμα είναι να αποτύχει ο στόχος που είχε ενεργοποιήσει τη φράση που περιείχε την αποκοπή (Ο στόχος που ενοποιήθηκε με το **A** στη συγκεκριμένη περίπτωση, ο οποίος ονομάζεται και "**στόχος-πατέρας**"). Αυτό σημαίνει ότι δεν αναζητείται εναλλακτικός τρόπος ικανοποίησης του στόχου αυτού. Στο σχήμα 7.1 δίνεται μιά σχηματική αναπαράσταση της επίπτωσης που έχει η αποκοπή σε περίπτωση αποτυχίας του στόχου **F**:

Στην ουσία η αποκοπή περιορίζει το δέντρο εκτέλεσης ενός PROLOG προγράμματος (ή καλύτερα αποκόπτει ένα τμήμα του). Γιά το λόγο αυτό χρησιμοποιείται για να κάνει τη διαδικασία εκτέλεσης πιό αποτελεσματική (από την άποψη του χρόνου και του χώρου).



Σχήμα 7.1: Η επίπτωση της αποκοπής

Παράδειγμα 7.1 :

Στο σχήμα 7.3 φαίνεται πως η τοποθέτηση της αποκοπής σ' ένα πρόγραμμα, (βλέπε σχήμα 7.2), επιδρά στον περιορισμό του δέντρου εκτέλεσης της ερώτησης:

```
?- a(Z).
```

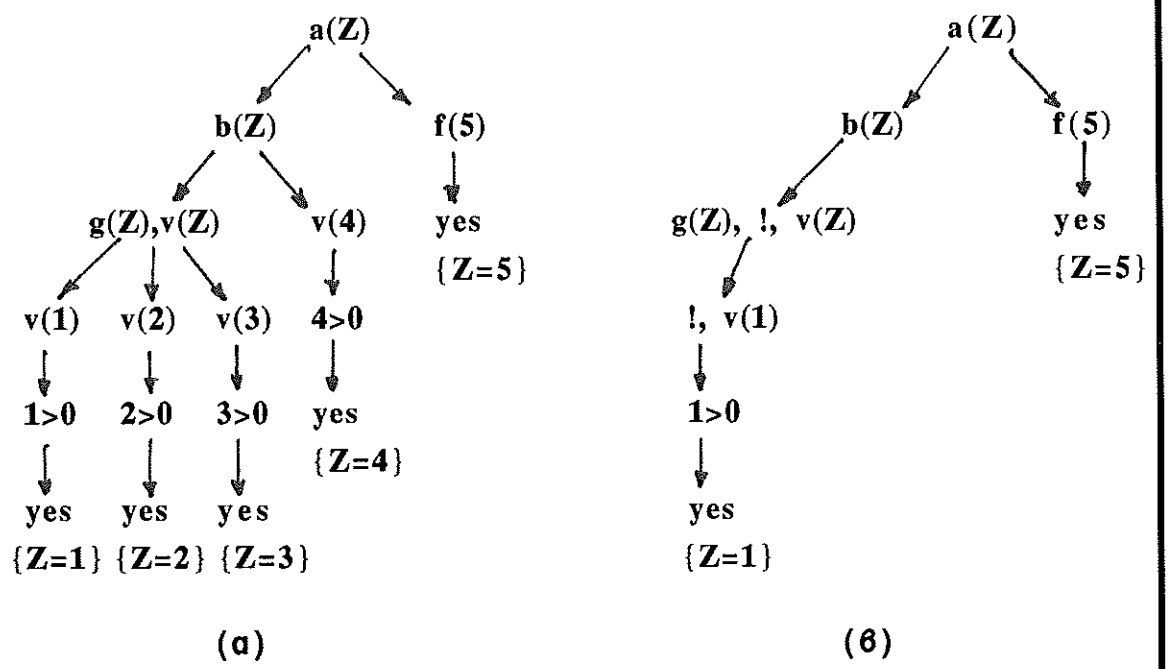
	(α)	(β)
(A1)	a(X) :- b(X).	a(X) :- b(X).
(A2)	a(X) :- f(X).	a(X) :- f(X).
(B1)	b(X) :- g(X), v(X).	b(X) :- g(X), !, v(X).
(B2)	b(X) :- X=4, v(X).	b(X) :- X=4, v(X).
(G1)	g(1).	g(1).
(G2)	g(2).	g(2).
(G3)	g(3).	g(3).
(V1)	v(X) :- X > 0.	v(X) :- X > 0.
(F1)	f(5).	f(5).

Σχήμα 7.2 : Τοποθέτηση της αποκοπής σ' ένα πρόγραμμα

Στην περίπτωση (φράσεις του σχήματος 7.2.α) η ερώτηση επιδέχεται πέντε απαντήσεις:

```
?- a(Z).
```

Z = 1 ;
Z = 2 ;
Z = 3 ;
Z = 4 ;
Z = 4



Σχήμα 7.3 : Τα δέντρα εκτέλεσης για τις φράσεις του σχήματος 7.2

Η εκτέλεση ξεκινά με την εξής διαδοχή στόχων:

?- a(Z).	{αρχική ερώτηση}
?- b(Z).	{λόγω της (A1)}
?- g(Z), v(Z).	{λόγω της (B1)}

Ο υποστόχος g(Z) λόγω των (G1), (G2), (G3) παράγει τις τιμές **1**, **2**, **3** για το **Z**, κάθε μία από τις οποίες αποτελεί τελικά λύση για την αρχική ερώτηση a(Z). Μία τέταρτη λύση (η **Z=4**) παράγεται, μετά από οπισθοδρόμηση στο στόχο b(Z) και την εναλλακτική του ικανοποίηση, μέσω της φράσης (B2). Η

τελευταία λύση (**Z=5**) παράγεται, μετά από οπισθοδρόμηση στον αρχικό στόχο **a(Z)** και την εναλλακτική του υλοποίηση, μέσω της φράσης (A2).

Στη δεύτερη περίπτωση, ας θεωρήσουμε τις φράσεις του σχήματος 7.2.β, όπου το κατηγόρημα της αποκοπής έχει τοποθετηθεί στην πρώτη φράση που ορίζει το **b**. Η αποκοπή εδώ, υποχρεώνει τη διαδικασία εκτέλεσης μετά από την εύρεση της πρώτης λύσης (**Z=1**) να οπισθοδρομήσει κατ' ευθείαν στον αρχικό στόχο **a(Z)**. Η ερώτηση **a(Z)** επιδέχεται τώρα μόνο δύο απαντήσεις:

```
?- a(Z).
Z = 1 ;
Z = 5
```

Η αποκοπή χρησιμοποιείται γενικά όταν ο προγραμματιστής γνωρίζει κάτι που δεν ξέρει η PROLOG. Ετσι το σύστημα της PROLOG δεν σπαταλά χρόνο για να ψάχνει κάποια λύση, που είτε δεν υπάρχει, είτε υπάρχει αλλά δεν ενδιαφέρει τον προγραμματιστή.

Μερικές από τις πιό συνηθισμένες προγραμματιστικές εφαρμογές της αποκοπής είναι οι εξής:

(α) Οταν θέλουμε να δηλώσουμε ότι μια συγκεκριμμένη φράση είναι η μόνη, που μπορεί να εφαρμοστεί σε ένα συγκεκριμένο πρόβλημα. Με αυτή τη χρήση της αποκοπής είναι δυνατόν να ορίσουμε δομές τύπου "**if-then-else**" (βλέπε παράδειγμα 7.2).

(β) Οταν θέλουμε να δηλώσουμε ότι η σχέση που προσπαθούμε να υπολογίσουμε είναι στην ουσία ντετερμινιστική - δηλαδή υπάρχει μόνο μία απάντηση γι' αυτή και συνεπώς δεν υπάρχει λόγος αναζήτησης άλλης εναλλακτικής λύσης.

(γ) Οταν θέλουμε να δηλώσουμε ότι κάτω από ορισμένες προϋποθέσεις κάποιος στόχος πρέπει να αποτύχει (βλέπε παράδειγμα 7.4).

Παράδειγμα 7.2 :

```
type_of(X,negative) :- X < 0, !.
type_of(X,positive) :- X > 0, !.
type_of(X,zero) :- X = 0, !. /* Το cut εδώ δεν είναι απαραίτητο */
```

Το κατηγόρημα **type_of(X,Y)** επιστρέφει στο **Y** την τιμή **negative**, **positive** και **zero**, εάν αντίστοιχα, ο **X** είναι ένας αριθμός θετικός, αρνητικός ή μηδέν. Στην περίπτωση αυτή, η τοποθέτηση της αποκοπής μετά από τη συνθήκη στο σώμα του κάθε κανόνα διασφαλίζει ότι οι τρείς φράσεις είναι αμοιβαία αποκλειόμενες. Λόγω του σειριακού τρόπου αναζήτησης που χρησιμοποιεί ο μηχανισμός εκτέλεσης της PROLOG, η συνθήκη και η αποκοπή στον τρίτο κανόνα δεν είναι απαραίτητες και αυτός μπορεί να αντικατασταθεί απλά από το γεγονός **type_of(X,zero)**.

Παράδειγμα 7.3 :

```
write_name(1) :- !, write('one').
write_name(2) :- !, write('two').
. . .
write_name(10) :- !, write('ten').
```

Στο παράδειγμα αυτό το κατηγόρημα **write_name(X)** χρησιμοποιείται για να τυπώσει το λεκτικό ισοδύναμο του **X**, στην περίπτωση που αυτό είναι ένας αριθμός από το **1** εώς το **10**. Στην περίπτωση αυτή η αποκοπή τοποθετείται σαν πρώτη συνθήκη σε κάθε κανόνα, γιατί είναι σαφές ότι, έαν η ενοποίηση ενός στόχου με την κεφαλή κάποιου κανόνα πετύχει, ο κανόνας αυτός είναι και ο μόνος κατάλληλος.

Παράδειγμα 7.4 :

```
likes(mary,X) :- snake(X), !, fail.
likes(mary,X) :- animal(X).
```

Στο παράδειγμα αυτό το κατηγόρημα **likes** χρησιμοποιείται για να δηλώσει ότι η Μαίρη αγαπάει όλα τα ζώα εκτός από τα φίδια.

7.1 Αρνηση με Αποτυχία (*Negation by failure*)

Η PROLOG απαντά όχι σε μία ερώτηση όταν αποτύχει να την ικανοποιήσει.

"Εφόσον δεν μπορώ να το αποδείξω, πρέπει να είναι ψευδές"

Αυτή η μορφή άρνησης ονομάζεται **άρνηση με αποτυχία (*negation by failure*)**. Ο συνδυασμός της αποκοπής με το κατηγόρημα **fail**, μπορεί να χρησιμοποιηθεί για τον ορισμό της άρνησης αυτού του είδους:

```
not(G) :- G, !, fail.
not(G).
```

Το κατηγόρημα **not** δέχεται σαν όρισμά του ένα στόχο **G** (ή σύζευξη στόχων). Εξ' αιτίας της πρώτης φράσης, εάν ο **G** πετύχει, τότε το **not(G)** αποτυχαίνει, λόγω της αποτυχίας του **fail** και του ότι η αποκοπή δεν επιτρέπει εναλλακτικό τρόπο ικανοποίησης για το **not**. Εάν ο στόχος **G** αποτύχει, τότε το **not(G)** πετυχαίνει εξ' αιτίας της δεύτερης φράσης.