Λογικός Προγραμματισμός

Σημειώσεις Διδασκαλίας

(Εκδοση 2.1)

Θεμιστοκλής Ν. Παναγιωτόπουλος

Επίκουρος Καθηγητής
Τμήμα Πληροφορικής
Πανεπιστήμιο Πειραιά

Πειραιάς 2001

Πρόλογος

Οι σημειώσεις αυτές γράφτηκαν για να βοηθήσουν από πλευράς ύλης το μάθημα του Λογικού Προγραμματισμού, μάθημα του 5ου εξαμήνου του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς.

Οι σημειώσεις είναι αρκετά εκτεταμένες ούτε και πλήρεις όσον αφορά τα τελευταία κυρίως κεφάλαια, και βρίσκονται στην δεύτερή τους έκδοση. Εγινε μιά προσπάθεια να καλυφθούν θέματα θεωρητικά, κι αυτό θεωρώ ότι έγινε με πληθώρα παραδειγμάτων, που θα λύσουν τις απορίες των φοιτητών και θα τους εντάξουν στο χώρο του Λογικού Προγραμματισμού με σχετική ευκολία.

Τα πρώτα κεφάλαια έχουν ακόμη αρκετές ελλείψεις που θα καλυφθούν την επόμενη χρονιά. Λείπουν βασικά θέματα όπως το θεώρημα του Herbrand και η απόδειξή του, αποδείξεις των θεωρημάτων ορθότητας και πληρότητας όσον αφορά στη Προτασιακή και Κατηγορηματική Λογική α' τάξης, πολλές άλλες στρατηγικές της αρχής της απόφασης, ενδιαφέροντα θέματα που αφορούν την τεχνολογία της Prolog, κ.λ.π.

Τα τελευταία κεφάλαια αυτών των σημειώσεων έχουν αφιερωθεί στην Prolog. Σε σχέση με την πρώτη έκδοση έχουν προστεθεί αρκετά κεφάλαια στα οποία έχουμε αρκετά παραδείγματα, σχήματα, μεθόδους προγραμματισμού, μικρές και μεγάλες εφαρμογές, κ.λ.π.

Τόσο μέσα από το μάθημα όσο και μέσα από τις εργασίες που ανατέθηκαν στους φοιτητές, πιστεύω να πήραν αυτό που ήθελα από την αρχή : να κατανοήσουν τις δύσκολες έννοιες του λογικού προγραμματισμού, να εκτιμήσουν την Prolog σαν γλώσσα προγραμματισμού, και να έρθουν σε επαφή με σύγχρονα ερευνητικά θέματα.

Νομίζω ότι είχαν την δυνατότητα να κινηθούν (έστω γιά λίγο) στα σύνορα των σύγχρονων ερευνητικών δραστηριοτήτων, να καταλάβουν κάποια βαθύτερα θέματα και να τους μείνει τελικά, και μετά από χρόνια η απαραίτητη εκείνη επιστημονική αντίληψη για το τί σημαίνει λογικός προγραμματισμός και η γλώσσα Prolog.

Με τιμή

Θ.Ν.Π.

1

Τεχνητή Νοημοσύνη και Λογικός Προγραμματισμός

1.1 Τι είναι Τεχνητή Νοημοσύνη

Η Τεχνητή Νοημοσύνη (Artificial Intelligence, AI, ΤΝ) είναι ο κλάδος της Πληροφορικής, ο οποίος μελετά την νοήμονα συμπεριφορά και προσπαθεί να αναπτύξει μοντέλα και συστήματα που επιδεικνύουν νοήμονα συμπεριφορά.

Γιά το λόγο αυτό η ΤΝ αποτελεί μιά σύνθεση μοντέλων, μεθοδολογιών, εργαλείων, και συστημάτων που τα δανείζεται, αλλά ταυτόχρονα τα ενοποιεί και τα εξειδικεύει, από τέσσερις διαφορετικές επιστημονικές περιοχές :

- τις Γνωστικές Επιστήμες (Cognitive Sciences), όπως Φιλοσοφία, Λογική, Ψυχολογία, Κυβερνητική,
- την Μαθηματική επιστήμη και ειδικότερα την Μαθηματική Λογική, την Θεωρία Μοντέλων και Αποδείξεων, Επιχειρησιακή Ερευνα,
- την ευρύτερη επιστημονική περιοχή της Πληροφορικής καθώς το κυριότερο προϊόν της ΤΝ είναι τα έξυπνα πληροφοριακά συστήματα, τα έμπειρα συστήματα, οι αποδείκτες θεωρημάτων, κ.λ.π.
- την Αυτοματική και τη Θεωρία Ελέγχου σε συνεργασία με την βιοτεχνολογία, που προτείνουν την ανάπτυξη ευφυιών συστημάτων σε επίπεδο υλικού (hardware, νευρωνικά δίκτυα).

1950	Νευρωνικά δίκτυα, Rosenblatt,Wiener, McCulloch,PERCEPTRON
1960	Ευρετική ανίχνευση, Newell,Simon,Shannon, Turing, GPS
1970	Παράσταση της γνώσης, Sortliffe,Minsky, McCarthy, MYCIN
1980	Σύστημα μάθησης, Lenat,Samuel, Holland, EURISCO

Σχήμα 1.1. Ιστορικά Βήματα της Τεχνητής Νοημοσύνης ανά δεκαετία

Ιστορικά, μπορούμε να διακρίνουμε κάποια βήματα στην εξέλιξη της ΤΝ (Σχ.1). Τα βήματα αυτά δεν είναι αντιπροσωπευτικά, με την έννοια ότι σε καμμιά περίπτωση δεν δείχνουν την εξέλιξη της ΤΝ σε όλη της την έκταση, αλλά είναι απλώς ενδεικτικά κάποιων ερευνητικών τάσεων ανά δεκαετία.

Στην συνέχεια θα ασχοληθούμε πολύ περιληπτικά με τα θέματα που εξετάζει και ερευνά η ΤΝ, τα μοντέλα που προτείνει, τις λύσεις που προσφέρει στα σύγχρονα προβλήματα και τα πεδία στα οποία εφαρμόζεται.

1.2 Η φύση της γνώσης και της νοημοσύνης

Η ΤΝ εξετάζει την φύση της ***γνώσης*** και της ***νοημοσύνης*** και ασχολείται τόσο γενικά όσο και ειδικά με φαινόμενα στα οποία η γνώση και η νοημοσύνη αποτελούν κυρίαρχο παράγοντα.

Ερωτήματα που έχουν εν μέρει απαντηθεί ή περιμένουν ακόμη απάντηση έχουν την μορφή : Είναι η γνώση απλή ή σύνθετη, δομημένη ή αδόμητη, δηλωτική ή διαδικαστική, σαφής ή ασαφής, βέβαιη ή αβέβαιη, ακριβής ή ανακριβής (Σχ.2,3). Φαίνεται ότι η γνώση την οποία κατέχει ένας άνθρωπος διαθέτει όλα αυτά τα χαρακτηριστικά, αλλά τα ανθρώπινα όντα διαθέτουν επιπλέον μιά χαρακτηριστική ευκολία να συνδυάζουν τέτοια ανομοιογενή στοιχεία γνώσης χωρίς να ξέρουν τον τρόπο που το κάνουν. Η ίδια η έννοια της νοημοσύνης (Σχ.4,5), που είναι η ικανότητα του "σκέπτεσθαι", φαίνεται να είναι αρκετά πολύπλοκη και η εξέταση της είναι μέχρι τώρα πολύ επίπονη εργασία.

απλή	-	σύνθετη,
δομημένη	-	αδόμητη,
δηλωτική	-	διαδικαστική,
ρηχή	-	βαθειά,
σαφής	-	ασαφής,
βέβαιη	-	αβέβαιη,
ακριβής	-	ανακριβής.
πλήρης	-	ελλιπής,
συνολικά συμβατή	-	τοπικά ασύμβατη

Σχήμα 1.2. Χαρακτηριστικά της γνώσης

Ετσι λοιπόν η ΤΝ εξετάζει πως ένα σύστημα (π.χ. ο άνθρωπος) αποκτάει γνώσεις, προσθέτει νέες γνώσεις στις ήδη υπάρχουσες, αλλά και πως γίνεται η σύλληψη, κατηγοριοποίηση και δόμηση των γνώσεων αυτών. Εξετάζει ακόμα τους τρόπους της σκέψης, τους τρόπους δηλαδή εκείνους που χρησιμοποιεί ένα νοήμον σύστημα γιά να συνδυάσει και να συσχετίσει την υπάρχουσα γνώση και τους τρόπους με βάση τους οποίους ένα νοήμον σύστημα παράγει νέα γνώση.

Η προηγούμενη εξέταση άπτεται κυρίως των Γνωστικών Επιστημών. Ομως η ΤΝ προχωράει στη συνέχεια στην ανάπτυξη μαθηματικών θεωριών και μοντέλων που είναι δυνατόν να περιγράψουν με την απαραίτητη μαθηματική αυστηρότητα την ***φύση της γνώσης***, τόσο *γενικά : τα πραγματικά, τα νοητά, τα συγκεκριμένα, τα αφηρημένα, και τα διακεκριμένα αντικείμενα*, όσο και *ειδικά : οι επιμέρους ιδιότητες των αντικειμένων αυτών και οι ιδιαίτερες συνθήκες και αλληλοσυσχετίσεις τους σε πολύ συγκεκριμένα περιβάλλοντα*, με στόχο όχι να παράγει μιά νέα μαθηματική θεώρηση, αλλά να αναπτύξει ένα περιβάλλον το οποίο θα επιτρέψει στη συνέχεια να αποτυπωθούν και να υποστούν επεξεργασία όλα αυτά σε ένα υπολογιστικό σύστημα.

Ενα τέτοιο περιβάλλον, που μπορεί να περιγράψει με την απαραίτητη Μαθηματική αυστηρότητα την φύση της γνώσης, είναι μιά ***τυπική γλώσσα***. Γιά τον ορισμό μιάς τυπικής

γλώσσας προυποτίθεται ένα αυστηρό ***συντακτικό*** : αλφάβητο, λεξικό, καθώς και συντακτικοί κανόνες γιά τον σχηματισμό προτάσεων. Προυποτίθεται επίσης μιά αυστηρά καθορισμένη ***σημασιολογία*** : το πως δηλαδή ερμηνεύονται τα σύμβολα της γλώσσας και το τι αναπαριστούν όσον αφορά τα αντικείμενα του κόσμου. Τέλος είναι τελείως απαραίτητη και μια ***θεωρία απόδειξης*** η οποία επιτρέπει την παραγωγή νέων προτάσεων με βάση τις υπάρχουσες προτάσεις. Η σχεδίαση και ανάπτυξη μιάς τέτοιας γλώσσας επιτρέπει στην ΤΝ να εκφράσει και να αναπαραστήσει φαινόμενα και προβλήματα του κόσμου και των επιστημών.

Οντότητες (αντικείμενα) : φυσικές/τεχνητές, συγκεκριμένες/διακεκριμένες, συγκεκριμένες/αφηρημένες, πραγματικές/φανταστικές.
Εννοιες - (συσχετίσεις - συναρτήσεις) : Εννοια - Ιδιότητα - Τιμή, Δίκτυα εννοιών, Κλάσεις εννοιών, Στιγμιότυπα εννοιών, Σύνθετες - απλές - πρωταρχικές έννοιες
Περιεχόμενο της γνώσης : Στατική γνώση γιά την ύπαρξη οντοτήτων, εννοιών, συσχετίσεων. Δυναμική γνώση (μέσω συλλογισμού) γιά την ύπαρξη. Μεταγνώση : Γνώση του γνωστικού αντικειμένου του ίδιου του συστήματος, Γνώση του γνωστικού περιεχομένου ενός άλλου συστήματος

Σχήμα 1.3. Τι αναπαριστούμε σαν γνώση

Μονότονη - Μη μονότονη
Ακριβής - Ανακριβής
Βέβαιη - Αβέβαιη
Σαφής - Ασαφής
Αναλογική (με μεταφορές)
Βασιζόμενη σε προηγούμενη εμπειρία κ.λ.π.

Σχήμα 1.4. Τύποι συμπερασματολογίας

Η διαφοροποίηση από την καθαρή μαθηματική επιστήμη είναι εδώ η εξής : Η ΤΝ δεν ενδιαφέρεται τόσο στην ανάπτυξη θεωριών γιά αυθαίρετα ή πραγματικά αντικείμενα και τις ιδιότητές τους, όσο γιά την ανάπτυξη μοντέλων και θεωριών που περιγράφουν με ακρίβεια γνωστά αντικείμενα (νοητά ή πραγματικά) και τις ιδιότητές τους και μπορούν να προβλέπουν και να ερμηνεύουν ***αυτόματα*** την συμπεριφορά αυτών των αντικειμένων.

Ετσι λοιπόν εάν κάποιος ερευνητής της ΤΝ ασχοληθεί με ένα μοντέλο που περιγράφει ένα φυσικό φαινόμενο ή μιά έννοια όπως π.χ. ο χρόνος, δεν ενδιαφέρεται να αναπτύξει μιά νέα φυσική θεωρία, αλλά ενδιαφέρεται να αναπτύξει ένα σύστημα το οποίο μπορεί να προβλέπει, να ερμηνεύει, να εξηγεί το φαινόμενο ακριβώς όπως το προβλέπει το ερμηνεύει και το εξηγεί ένας φυσικός. Η μπορεί να θέλει να μοντελοποιήσει με ένα παραστατικό τρόπο την συμπεριφορά κάποιων οντοτήτων μέσα στον χρόνο, έτσι ώστε το σύστημα που αναπτύσσει να προβλέπει αυτόματα τις αναμενόμενες εξελίξεις με βάση την μοντελοποιημένη συμπεριφορά.

Κανόνες Συμπερασμού :

- ***Modus Ponens*** : Από Α, Α-->Β συμπεραίνω το Β
- ***Γενίκευση*** : Οταν μιά ιδιότητα αληθεύει γιά τους αντιπροσώπους μιάς κλάσης, αληθεύει γιά όλη την κλάση.
- ***Ειδίκευση*** : Οταν μιά ιδιότητα αληθεύει γιά μιά κλάση αληθεύει γιά κάθε αντιπρόσωπο της κλάσης.
- ***Κληρονομικότητα*** : Μιά ιδιότητα (ή/και η τιμή της) κληρονομείται από μιά υπερκλάση σε υποκλάσεις της.
- ***Αναγνώριση*** : Μιά οντότητα αναγνωρίζεται ως αντιπρόσωπος μιάς κλάσης.
- ***Συγγένεια εννοιών*** : Δύο οντότητες αναγνωρίζονται να ικανοποιούν κάποιες συσχετίσεις.
- ***Αναμενόμενη Τιμή*** : Εάν λείπει η σχετική πληροφορία δώσε σε μιά ιδιότητα μιά τυπική τιμή της κλάσης.

Συλλογισμοί :

- ***Απόδειξη*** : Παραγωγή προτάσεων χρησιμοποιώντας την υπάρχουσα γνώση και κανόνες συμπερασμού.
- ***Απαντήσεις σε ερωτήσεις*** : Παραγωγή της πρότασης εκείνης που ενοποιείται με την ερώτηση.
- ***Στόχοι, υποστόχοι, δεδομένα, στρατηγικές απόδειξης*** κατευθυνόμενη από τους στόχους ή από τα δεδομένα, κατά βάθος ή κατά πλάτος κ.λ.π.
- ***Μηχανισμός ενεργοποιημένης διάχυσης ψαξίματος διασταυρούμενων δρόμων*** *(spreading activation intersection search)*.

Σχήμα 1.5. Η νοημοσύνη στη φύση της (Σκέψη-Λογική-Συλλογισμοί)

Ετσι εμφανίζεται στην ΤΝ μιά τάση να προτείνονται συστήματα τα οποία περιγράφουν υπάρχουσες θεωρίες και φαινόμενα που έχουν ήδη κατανοηθεί. Στα συστήματα αυτά πρέπει να υπάρχει τρόπος να αναπαρασταθεί η γνώση των αντικειμένων του κόσμου, των νόμων στους οποίους υπακούουν, των διαδικασιών συμπερασματολογίας, κ.λ.π.

Από την Μαθηματική επιστήμη η ΤΝ δανείζεται και εξειδικεύει την ***Μαθηματική Λογική***, την ***Θεωρία Μοντέλων και Αποδείξεων***. Οι τυπικές γλώσσες που προτείνονται χαρακτηρίζονται βέβαια από τις έννοιες της σημασιολογίας και της παραγωγής προτάσεων, της ικανοποιησιμότητας, των τιμών αλήθειας ή ψεύδους, των κανόνων παραγωγής, της ορθότητας και πληρότητας.

Στις ***μη μονότονες λογικές***, υλοποιούνται και κανόνες παραγωγής προτάσεων που δεν διατηρούν την ορθότητα της σημασιολογίας της κλασσικής λογικής (όπως η υπόθεση του κλειστού κόσμου, οι κανόνες της αναμενόμενης αλήθειας, η ολοκλήρωση των κατηγορημάτων, και πολλά άλλα).

Καθώς όμως μιλάμε γιά γλώσσες και μοντέλα που πρέπει να υλοποιηθούν και να τρέξουν σε ένα υπολογιστικό σύστημα, με τους περιορισμούς σε χρόνο εκτέλεσης, κεντρική μνήμη, ταχύτητες, κ.λ.π. εξετάζονται και αναπτύσσονται μοντέλα και συστήματα που λαμβάνουν υπόψη τους τέτοιους περιορισμούς. Τέτοια μοντέλα αναφέρονται ως ***στρατηγικές συμπερασματολογίας***, και παραβλέπουν (κατά την θεωρία και την υλοποίηση) την πληρότητα των κλασσικών συστημάτων λογικής.

Προτείνονται ακόμη λογικές υψηλότερης τάξης (2ης, 3ης κ.λ.π), λογικές της κατάστασης, αυτοαναφερόμενες λογικές, λογική της γνώσης και της άποψης, χρονικές λογικές, χωρικές λογικές κ.λ.π.

Λογικές : Προτασιακή Λογική, Κατηγορηματική Λογική α' τάξης, Λογικές ανωτέρων τάξεων, Λογική της κατάστασης, Χρονική Λογική, Χωρική Λογική, Αυτο-επιστεμική Λογική,Λογική της γνώσης και της πεποίθησης, Λογική αναμενόμενης τιμής, Λογική των τύπων κ.λ.π.
Σημασιολογικά δίκτυα : Πέντε επίπεδα αναπαράστασης : επίπεδο υλοποίησης, λογικό επίπεδο, επιστημολογικό επίπεδο, εννοιολογικό επίπεδο, γλωσσικό επίπεδο.
Πλαίσια : Δομημένοι κόμβοι σε ιεραρχικά εννοιολογικά δίκτυα.
Σενάρια : Πλαίσια με χρονική συνιστώσα.
Κανόνες Παραγωγής : Συνδυασμός δηλωτικής με διαδικαστική γνώση.

Σχήμα 1.6. Μοντέλα Αναπαράστασης και επεξεργασίας της γνώσης

Ενα άλλο στοιχείο που λαμβάνει σοβαρά υπόψη της η ΤΝ είναι ότι γιά να περιγράψει την νοήμονα συμπεριφορά πολύπλοκων συστημάτων όπως ο άνθρωπος, είναι απαραίτητο να μοντελοποιήσει και στοιχεία γνώσης που δεν εμφανίζονται συνήθως σε μαθηματικά συστήματα : ***αβέβαιη***, ***ανακριβή***, ***ασαφή*** ακόμα και γνώση που μπορεί να περιέχει αντικρουόμενα στοιχεία (***τοπικές ασυμβατότητες***). Η ΤΝ πρέπει να δώσει λύσεις σέ όλα αυτά τα θέματα με τα οποία η Μαθηματική Επιστήμη δεν ασχολείται. Λύσεις σε αυτά δίδονται από τα μοντέλα ανακρίβειας και αβεβαιότητας, την ασαφή λογική, κ.λ.π.

Αλλα χαρακτηριστικά της γνώσης που με μεγάλη δυσκολία μοντελοποιούνται είναι η δυνατότητα που έχει ο άνθρωπος να καταλήγει σε συμπεράσματα από ***ελλειπή γνώση***, η δυνατότητα που έχει να γενικεύει από ελλειπή στοιχεία, να αναθεωρεί τις απόψεις του, να έχει άποψη γιά την άποψη των άλλων, να γνωρίζει τι γνωρίζει, κ.λ.π.

Η λογική είναι ένα περιορισμένο πλαίσιο αναπαράστασης και επεξεργασίας της γνώσης. Εχει βέβαια ένα έντονο ***δηλωτικό χαρακτήρα (declarative knowledge)***, δεν αναπαριστά όμως την γνώση δομημένα, δεν την κατατάσει ούτε την ομαδοποιεί με κάποιο τρόπο. Το ισχυρότερο προτέρημα της λογικής ως μοντέλο αναπαράστασης και επεξεργασίας της γνώσης είναι η αυστηρότητά της ως προς την σημασιολογία.

Εκτός από την λογική, έχουν προταθεί πολλά άλλα μοντέλα (Σχ.6) γιά τον ίδιο σκοπό όπως τα ***σημασιολογικά δίκτυα, τα πλαίσια, τα σενάρια, οι κανόνες παραγωγής***, κ.λ.π. Ολα αυτά προτείνουν με τη σειρά τους δικές τους μεθοδολογίες, δομές και μηχανισμούς αναπαράστασης της γνώσης και υλοποιούν τρόπους επεξεργασίας της γνώσης που αναφέρονται τόσο στις γνωσιακές δομές αυτές καθ'εαυτές όσο και στις λειτουργίες της ανθρώπινης σκέψης.

Στα ***σημασιολογικά δίκτυα*** (semantic networks, ΣΔ) γίνεται η παραδοχή ότι η γνώση είναι δομημένη και μάλιστα με τέτοιο τρόπο που οι έννοιες αντιστοιχούν στους κόμβους ενός γράφου και οι μεταξύ τους σχέσεις αναπαρίστανται από τους δεσμούς του γράφου. Τα ΣΔ λοιπόν αποτελούν ένα είδος γράφου εφοδιασμένου με σημασιολογία. Τα ΣΔ αναπαριστούν σχέσεις κλάσης-υποκλάσης, έννοια - υπερέννοια - υποέννοια, έννοια - ιδιώματα - τιμές, γενικές έννοιες - στιγμιότυπα εννοιών, ταξινομικές ιεραρχίες, περιορισμούς ιδιοτήτων, σημασιολογικές περιπτώσεις (semantic cases), καταστάσεις κ.λ.π. Ορίζονται ακόμη πρωταρχικοί σύνδεσμοι (primitive links) και πρωταρχικές έννοιες. Η επεξεργασία της γνώσης (η παραγωγή συμπερασμάτων) γίνεται με ένα ***μηχανισμό***

ενεργοποιημένης διάχυσης ψαξίματος διασταυρούμενων δρόμων (spreading activation intersection search). Ορίζονται όμως και άλλοι μηχανισμοί συμπερασματολογίας όπως ***κληρονομικότητα, ειδίκευση, γενίκευση***, κ.λ.π.

Τα ***πλαίσια*** (frames) είναι και αυτά δίκτυα που έχουν όμως το χαρακτηριστικό ότι κάθε κόμβος περιέχει από μόνος του μιά σύνθετη δομή καθώς αποτελείται από σχισμές (slots) που περιγράφουν κάποια ιδιώματα της έννοιας, και τιμές γιά κάθε σχισμή. Είναι δυνατόν η τιμή μιάς σχισμής να είναι ένα άλλο πλαίσιο, να περιορίζεται, και σε περιπτωση απουσίας πληροφορίας να παίρνει αυτόματα μιά ***αναμενόμενη τιμή*** (default value). Εχουμε και εδώ μηχανισμούς κληρονομικότητας, γενίκευσης και ειδίκευσης. Αν προσθέσουμε και την διάσταση του χρόνου, παίρνουμε τα ***σενάρια*** (scripts) τα οποία περιγραφουν φαινόμενα που εξελίσσονται μέσα στον χρόνο.

Τα σημασιολογικά δίκτυα, τα πλαίσια και τα σενάρια, διαθέτουν μερικές φορές και μηχανισμούς ***διαδικαστικής γνώσης*** (procedural knowledge). Στην διαδικαστική γνώση δεν δηλώνουμε τι γνωρίζουμε, αλλά παρέχουμε μιά γρήγορη διαδικασία με βάση την οποία υπολογίζεται αλγοριθμικά αυτό που θέλουμε να αναπαραστήσουμε. Η διαδικαστική αναπαράσταση και επεξεργασία της γνώσης δεν είναι "κομψή" γιατί δεν διαθέτει σημασιολογία και δεν μπορεί να αναλυθεί σε επιμέρους στοιχεία γνώσης, αλλά είναι γρήγορη και αποδοτική. Συνήθως χρησιμοποιείται σαν μοντέλο αναπαράστασης των πιό στοιχειωδών γνωσιακών δομών ή και αυτών των ίδιων μηχανισμών συμπερασματολογίας.

Ενα χαρακτηριστικό παράδειγμα στο οποίο χρησιμοποιείται σε μεγάλη έκταση διαδικαστική γνώση είναι οι ***κανόνες παραγωγής*** (production rules). Αυτοί είναι κανόνες της μορφής "εάν Α τότε Β" που ενεργοποιούνται όταν ικανοποιούνται οι συνθήκες του κανόνα.

1.3 Προβλήματα, περιοχές εφαρμογής, εργαλεία και ανάπτυξη συστημάτων της ΤΝ.

Πολύ περιληπτικά μερικά ενδεικτικά προβλήματα που ασχολείται και λύνει η ΤΝ είναι :

- Παίξιμο παιχνιδιών
- Απόδειξη θεωρημάτων
- Κατανόηση φυσικής γλώσσας
- Αντίληψη (όραση, ομιλία)
- Επίλυση εξειδικευμένων προβλημάτων
- Συμβολικά μαθηματικά
- Ιατρική διαγνωστική
- Διαγνωστική γενικώτερα στις επιστήμες
- Διάγνωση βλαβών συστημάτων - Διαγνωστική Τεχνολογία)
- Πρόβλεψη (π.χ. μετεωρολογία, χρονοπρογραμματισμός, Χρηματιστήριο, κ.λ.π.)
- Χημική Ανάλυση
- Σχεδιασμός δράσης robot και κυττάρων βιομηχανικής κατασκευής
- Σχεδίαση γενικά και ειδικά Αρχιτεκτονική, Πολιτικών Μηχανικών, κυκλωμάτων, VLSI, κ.λ.π.)

Ενδεικτικά και πάλι μερικές γενικές περιοχές εφαρμογής της ΤΝ είναι :

- ***Μαθηματικά*** **:** Μαθηματική συμπερασματολογία, και αποδείξεις θεωρημάτων στην περιοχή των μαθηματικών

- ***Φυσική*** **:** Μοντελοποίηση συστημάτων, Ποιοτική συμπερασματολογία, αφελής θεωρία φυσικής
- ***Βιομηχανία*** **:** Αυτοματοποίηση βιομηχανικής δραστηριότητας
- ***Οικονομία*** **:** Μικρο- και Μακρο- οικονομία
- ***Τεχνολογία*** **:** Μηχανολογία, Ναυπηγική, Αεροναυπηγική, Διαστημική Τεχνολογία Ηλεκτρολογία,Ηλεκτρονική,Μικροηλεκτρονική,Χημεία,Φυσική
- ***Πυρηνική φυσική*** **:** Ελεγχος καλής λειτουργίας πυρηνικών αντιδραστήρων
- ***Ιατρική*** **:** Διάγνωση, Θεραπευτική
- ***Νομικά*** **:** Νόμοι - Δικαστικές υποθέσεις
- ***Εμπόριο, Τραπεζικές εφαρμογές*** **:** Ποικίλες εφαρμογές
- ***Στρατιωτικές εφαρμογές*** **:** Ποικίλες εφαρμογές
- ***Μουσική και Ζωγραφική*** **:** Αυτόματη παραγωγή μουσικής/εικόνας

Σαν εργαλεία για την ανάπτυξη εφαρμογών στην ΤΝ έχουν χρησιμοποιηθεί :

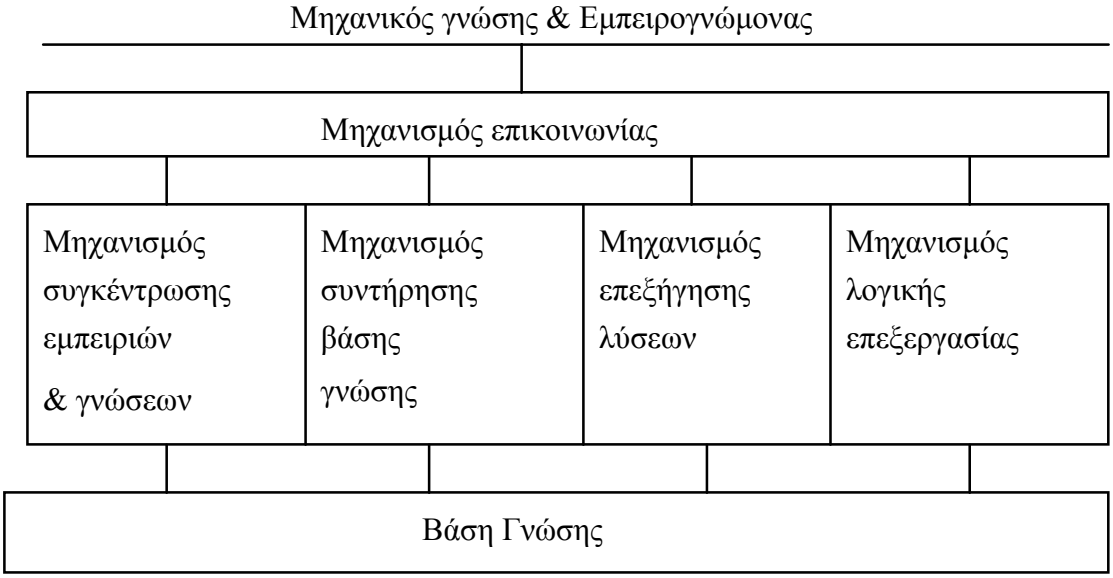
- Συμβατικές γλώσσες προγραμματισμού.
- Γλώσσες επεξεργασίας συμβόλων (π.χ. LISP).
- Προχωρημένες γλώσσες που διαθέτουν μηχανισμό συμπερασματολογίας (π.χ. Prolog).
- Γλώσσες Τεχνολογίας γνώσης.
- Ολοκληρωμένα περιβάλλοντα ανάπτυξης έμπειρων συστημάτων.

Τα ***Εμπειρα συστήματα*** ή ***Συστήματα εμπειρογνώμονες*** (expert systems) αποτελούν ένα ιδιαίτερο κλάδο γιατί σχετίζονται με συγκεκριμένες πάντα εφαρμογές. Είναι ο συνδετικός κρίκος της ΤΝ με τις άλλες επιστήμες. Μπορούμε να ορίσουμε ένα έμπειρο σύστημα σαν ένα σύστημα που "*αξιοποιεί τις ειδικές εμπειρίες και γνώσεις των εμπειρογνωμόνων με σκοπό να επιλύσει ή να συμβουλεύσει τον χρήστη στην επίλυση πολύπλοκων προβλημάτων μιάς καθορισμένης περιοχής ενδιαφέροντος, που θα ήταν δύσκολο αν όχι αδύνατο να επιλυθούν με συμβατικά συστήματα λογισμικού*". Ετσι, ένα έμπειρο σύστημα πρέπει να είναι ικανό :

- να αποφασίζει γιά την σχετικότητα του προβλήματος στην περιοχή ειδίκευσης.
- να μπορεί να αξιοποιεί περιορισμένη, ανακριβή ή ασαφή γνώση γιά την επίλυση προβλημάτων.
- να μπορεί να επεξηγεί με κατανοητό τρόπο πως οδηγήθηκε στα διάφορα συμπεράσματα.
- να μπορεί να αποκτά νέα γνώση με πιθανή αναπροσαρμογή και αναδιοργάνωση της βάσης γνώσης.
- να διαχειρίζεται εξαιρέσεις των κανόνων που χαρακτηρίζουν ορισμένες καταστάσεις.
- να προτείνει λύσεις σχετικά ικανοποιητικές σε προβλήματα που άπτονται οριακά στην περιοχή ενδιαφέροντός του.
- να έχει "επίγνωση" των περιορισμών του και των δυνατοτήτων του.

Η **Τεχνολογία γνώσης** (Knowledge Engineering) απασχολείται με τον μηχανισμό ανάπτυξης έμπειρων συστημάτων. Η ανάπτυξη έμπειρων συστημάτων προαπαιτεί την ύπαρξη ενός εμπειρογνώμονα, ενός μηχανικού γνώσης και ενός προηγμένου συστήματος ανάπτυξης έμπειρων συστημάτων. Ο μηχανικός γνώσης ακολουθεί μιά διαδικασία απόκτησης της γνώσης από τον εμπειρογνώμονα με σκοπό να την αναπαραστήσει σε κάποιο από τα μοντέλα αναπαράστασης της γνώσης (Σχ.6). Στην συνέχεια

χρησιμοποιώντας ένα πυρήνα έμπειρων συστημάτων (Σχ.7) οικοδομεί την βάση γνώσης. Φροντίζει δε, να την ενημερώνει και να την αναπροσαρμόζει συνεχώς με κάθε νέα πληροφορία η οποία καταφθάνει.

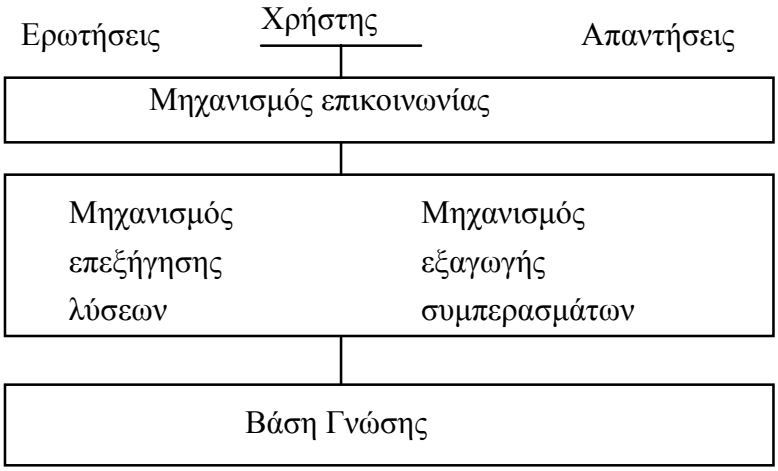


Σχήμα 1.7. Περιβάλλον Ανάπτυξης έμπειρου συστήματος

Το έμπειρο σύστημα που τελικά αναπτύσσεται διαθέτει την βάση γνώσης, ένα μηχανισμό συμπερασματολογίας, ένα μηχανισμό επεξήγησης συμπερασμάτων, και ένα ακόμη πιό σημαντικό μηχανισμό επικοινωνίας ανθρώπου-μηχανής (Σχ.8).

Παραθέτουμε στην συνέχεια ένα μικρό αριθμό από δημοφιλή έμπειρα συστήματα :

- ***Dendral*** : Ενας "εξυπνος" βοηθός γιά Χημικούς ο οποίος βοηθάει στην αποσαφήνιση, επιβεβαίωση, επαλήθευση, της ταυτότητας χημικών ουσιών στην Οργανική Χημεία.
- ***Meta-Dendral*** : Σχηματισμός κανόνων με εκμάθηση από δεδομένα φασματογράφου μάζας.
- ***Mycin*** : Πρόγραμμα ιατρικής διάγνωσης που διαθέτει κανόνες της μορφής : Εάν <συμπτώματα> τότε <πιθανή διάγνωση> και κάθε κανόνας συνοδεύεται και από ένα συντελεστή βεβαιότητας γιά το συμπέρασμα δεδομένων των συνθηκών.
- ***Prospector*** : Είναι ένα έμπειρο σύστημα γεωλογικής αναγνώρισης.
- ***Hearsay-II*** : Ενα έμπειρο σύστημα αναγνώρισης φωνής.
- ***Teiresias*** : Ενα διαλογικό έμπειρο σύστημα που βοηθάει τον εμπειρογνώμονα να "οικοδομήσει" βάσεις δεδομένων γιά διάφορα γνωστικά αντικείμενα.



Σχήμα 1.8. Περιβάλλον Λειτουργίας έμπειρου συστήματος

1.4 Περί Λογικού Προγραμματισμού

Ο Λογικός Προγραμματισμός (logic programming) αποτελεί ένα σημαντικό κλάδο της ΤΝ που βασίζεται σε μεγάλο βαθμό στη Μαθηματική Λογική, αλλά την προεκτείνει εισάγοντας νέες μεθόδους αυτοματοποιημένης συμπερασματολογίας. Η μελέτη του Λογικού Προγραμματισμού είναι να εξηγηθεί η *λογική βάση* του. Χωρίς αυτήν την μαθηματική εξήγηση δεν θα γίνει αντιληπτή η σπουδαιότητα του λογικού προγραμματισμού. Η βάση αυτή θα φανεί χρήσιμη σε διάφορα θέματα όπως :

- στη περιγραφή ενός συστήματος (δηλαδή στην κωδικοποίηση της γνώσης) με τη μορφή ενός συνόλου λογικών προτάσεων (δηλαδή με τη μορφή ενός λογικού προγράμματος)
- με ποιό τρόπο κινούμαστε από την Μαθηματική Λογική στο λογικό προγραμματισμό και στη συνέχεια στην γλώσσα προγραμματισμού Prolog.

Θα μας εφοδιάσει λ.χ. με το τι μπορούν ή δεν μπορούν να εκφράσουν ή υπολογίσουν τα λογικά προγράμματα, εν συντομία, τι *σημαίνουν*. Επιπλέον, μελετώντας σημασιολογικά και θεωρητικά ζητήματα θα σχολιασθεί, όπου κρίνεται απαραίτητο, και η *πρακτική* σπουδαιότητα των υπό μελέτη θεμάτων.

Θα γνωρίσουμε τη γλώσσα της Μαθηματικής Λογικής - Προτασιακής και Κατηγορηματικής Λογικής α' τάξης - ως μιας γλώσσας που μπορεί να χρησιμοποιηθεί γιά την επίλυση προβλημάτων. Στη συνέχεια θα δούμε πως η λογική και κατ' επέκταση ο λογικός προγραμματισμός δεν είναι απλά μια ακόμη γλώσσα για την επίλυση συνηθισμένων υπολογιστικών προβλημάτων.

Το κύριο σημείο εδώ είναι ότι, η λογική μπορεί να χρησιμοποιηθεί για την αναπαράσταση γνώσης που είναι συναφής με διάφορους τομείς, και μπορεί να διακινηθεί ή να αυτοματοποιηθεί με πολλούς τρόπους. Γιά το σκοπό αυτό θα δούμε διάφορες πρακτικές εφαρμογές που θα βασίζονται στην χρήση του λογικού προγραμματισμού και ιδιαίτερα στη χρήση της γλώσσας Prolog.

Ο λογικός προγραμματισμός έχει τη δυνατότητα να εξυπηρετήσει κάθε υπολογιστικό πλαίσιο που δίνει περιθώρια ***για αυτοματοποιημένη συμπερασματολογία*** (mechanized reasoning).

Οι λόγοι γιά τους οποίους θα ασχοληθούμε με την κλασσική λογική είναι :

- η κλασική λογική είναι πιο απλή και κατανοητή από τα άλλα είδη λογικής.
- μόνο αυτή έχει διερευνηθεί επαρκώς και με πληρότητα

- ως τώρα, μόνο η κλασική λογική έχει ανταποκριθεί σε υπολογιστικούς σκοπούς με μεγάλο βαθμό επιτυχίας, και αυτό βέβαια λόγω της γλώσσας Prolog.

Βέβαια, η κλασσική λογική, και κατ'επέκταση ο λογικός προγραμματισμός, δεν καλύπτει τις ανάγκες αναπαράστασης της γνώσης έτσι όπως αναπτύχθηκαν σε προηγούμενες παραγράφους. Ομως αποτελεί ίσως την πιό κατάλληλη 'θύρα' γιά την ξενάγηση του αναγνώστη στον τεράστιο από πλευράς θεωριών και εφαρμογών κόσμο της Τεχνητής Νοημοσύνης.

Υπάρχουν δύο κύριοι τρόποι προσέγγισης του μαθηματικού περιεχομένου της λογικής : η ***θεωρία μοντέλου*** (model theory) και η ***θεωρία απόδειξης*** (proof theory).

Η θεωρία μοντέλου εξετάζει την σχέση μεταξύ λογικών προτάσεων αφού έχουν ερμηνευθεί, με την βοήθεια εξωτερικών στοιχείων, με την βοήθεια δηλαδή της απόδοσης τιμών αλήθειας. Το λεξιλόγιο της βασικής θεωρίας μοντέλου περιλαμβάνει όρους όπως: αλήθεια (true), ψεύδος (false), ερμηνεία (interpretation), ικανοποίηση (satisfaction), μοντέλο (model), λογικό συμπέρασμα (logical consequence) ή σημασιολογική συνέπεια (semantic consequence).

Η θεωρία απόδειξης μελετά τις σχέσεις μεταξύ προτάσεων, όσον αφορά την δυνατότητα παραγωγής τους από άλλες προτάσεις, μέσω κανόνων που ενεργούν μόνο πάνω στη σύνταξη των προτάσεων αυτών. Το λεξιλόγιο της θεωρίας απόδειξης χρησιμοποιεί λέξεις όπως: αξίωμα (axiom) , κανόνας συμπερασμού (inference rule), θεώρημα (theorem) , απόδειξη (proof), συνέπεια (consistency) και λογικό συμπέρασμα ή συντακτική συνέπεια (syntactic consequence).

Και οι δύο προσεγγίσεις είναι σημαντικές για την κατανόηση του λογικού προγραμματισμού, και θα δούμε ότι στην κλασική λογική υπάρχουν απλές αλλά ουσιώδεις σχέσεις μεταξύ τους. Η πιο ισχυρή σχέση είναι ότι τα στοιχεία που θέλουμε να αληθεύουν θα πρέπει να συμπίπτουν με εκείνα που μπορούμε να αποδείξουμε, δηλαδή, οι απαντήσεις που υπονοούνται από ένα πρόγραμμα να συμπίπτουν με τις απαντήσεις που υπολογίζονται απ' αυτό.

1.5 Συμπληρωματικά συγγράμματα και έντυπο υλικό

Εχουν εκδοθεί πολλά βιβλία για τον λογικό προγραμματισμό, την Prolog και για διάφορα θέματα που συνδέονται μαζί τους. Τα περισσότερα είναι διδακτικοί οδηγοί στον προγραμματισμό με Prolog στο πλαίσιο των εφαρμογών Τεχνητής Νοημοσύνης όπως τα έμπειρα συστήματα. Ποικίλουν τόσο στο βαθμό της πολυπλοκότητάς τους όσο και στον βαθμό που αναγνωρίζουν και επεξεργάζονται βαθύτερες αρχές της λογικής.

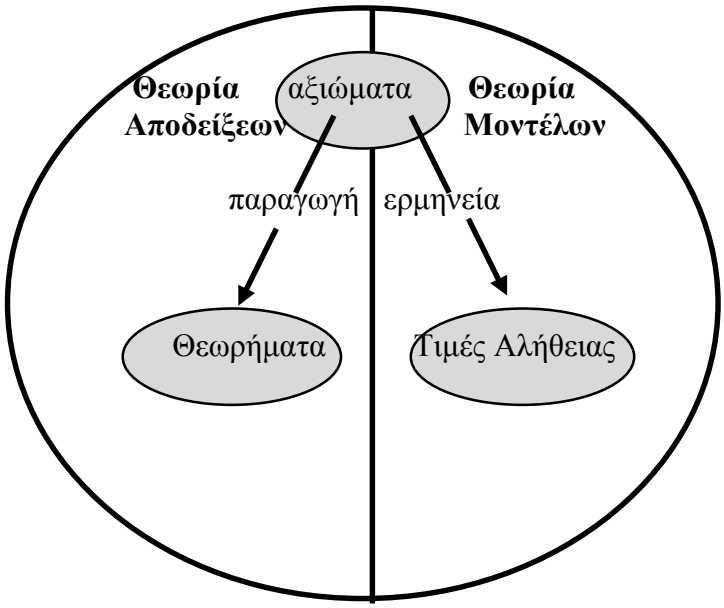
Παρακάτω δίνονται μερικά ξενόγλωσσα βιβλία που μπορούν να χρησιμεύσουν ως συμπλήρωμα του μαθήματος. Κανένα δεν είναι αναγκαστικά απαραίτητο για την ολοκλήρωση του υλικού που παρουσιάζεται σ'αυτές τις σημειώσεις, αλλά το καθένα μπορεί με τον δικό του ιδιαίτερο τρόπο να συμβάλει στην αφομοίωση του υλικού που θα παρουσιαστεί. Σε επόμενες εκδόσεις αυτών των σημειώσεων θα συμπεριληφθούν και Ελληνικά βιβλία.

- "Logic for Problem Solving", του Robert A. Kowalski [1], *ISBN*: 0-444-00365-7

Αυτό είναι το κλασικό σύγγραμμα για την χρήση της υπολογιστικής λογικής για την επίλυση προβλημάτων. Δεν έχει γραφτεί ειδικά για τον λογικό προγραμματισμό, ασχολείται περισσότερο με τη χρήση της λογικής για την αναπαράσταση γνώσης και την επεξεργασία της γενικά, αλλά είναι προσηλωμένο στην προσέγγιση των θεμάτων με τον λογικό προγραμματισμό. Σχεδιασμένο για αρχάριους στον τομέα αυτό, αποτελεί

σημαντικό βοήθημα στην επεξήγηση της προτασιασής λογικής και της ανάλυσης αποδείξεων θεωρημάτων (resolution theorem-proving), που αποτελούν βασικά στοιχεία των περισσότερων μορφών του κλασικού λογικού προγραμματισμού. Μέσα σ' αυτό το πλαίσιο καλύπτονται ποικίλα θέματα. Το βιβλίο περιλαμβάνει μεγάλο αριθμό ασκήσεων και δίνει αρκετή βιβλιογραφία, αν και κάποια απ' αυτή έχει αντικατασταθεί από νέα έρευνα.

Μαθηματικό περιεχόμενο της Λογικής



Σχήμα 1.9. Οι δύο όψεις της λογικής.

- "Essentials of Logic Programming" του Christopher J. Hogger [2],*ISBN*: 0-19-853820-0

Είναι ένα πολύ ενδιαφέρον βιβλίο που επιμένει πολύ στην ίδια την λογική και τον λογικό προγραμματισμό. Ξεκινάει από πολύ απλά θέματα και επεκτείνεται σε βαθειά μαθηματικά προβλήματα, αλλά με ένα βατό και κατανοητό τρόπο. Είναι χρήσιμο τόσο σαν εισαγωγή στις *υπολογιστικές* αρχές του λογικού προγραμματισμού, όσο και σαν αναφορά στην έρευνα του λογικού προγραμματισμού.

- "Foundations of Logic Programming" του John W. Lloyd [3], *ISBN*: 3-540-18199-7

Είναι το πιό πλήρες βιβλίο που αφορά στη θεωρία του λογικού προγραμματισμού. Παρουσιάζει, μεταξύ των άλλων, την θεωρία μοντέλων και αποδείξεων του λογικού προγραμματισμού με συστηματικό τρόπο, αλλά απαιτεί από τον αναγνώστη ένα αρκετά υψηλό επίπεδο Μαθηματικού υποβάθρου. Είναι ίσως το πλέον ουσιαστικό βιβλίο αναφοράς γιά ερευνητές του λογικού προγραμματισμού.

- "The Art of Prolog : Advanced Programming Techniques" των Leon Sterling και Ehud Y. Shapiro [4], *ISBN*: 0-262-19250-0 και

- "Prolog: Programming for Artificial Intelligence" του Ivan Bratko [5] ,*ISBN*: 0-201-41606-9
  Ισως τα δύο καλύτερα ξενόγλωσσα βιβλία για τον προγραμματισμό σε Prolog σήμερα. Περιλαμβάνουν αξιόλογο υλικό παρουσιασμένο με σαφήνεια και πολλά παραδείγματα. Ξεκινούν με τις βασικές αρχές που χρειάζεται να ξέρουν οι αρχάριοι στην Prolog, και συνεχίζει σε πιο πολύπλοκα επίπεδα που ταιριάζουν σε πιο προχωρημένους. Δεν αναφέρουν πολλά ούτε για τις λογικές βάσεις της Prolog ούτε για τις ευρύτερες δυνατότητες της υπολογιστικής λογικής. Μελετούν όμως εκτενώς σημαντικές περιπτώσεις με στόχο την καλύτερη χρησιμοποίηση της Prolog.

Οι παρακάτω πηγές υλικού για τον λογικό προγραμματισμό δεν είναι πλήρεις, αλλά αποτελούν μια καλή εισαγωγή και οδηγό για άλλες εργασίες.

- *Περιοδικά που καλύπτουν τον Λογικό Προγραμματισμό*
  Journal of Logic Programming
  New Generation Computing
  Logic and Computation
  Artificial Intelligence
  Machine Intelligence
  Knowledge Engineering Review
  Journal of Automated Reasoning
  Journal of the ACM
  Communications of the ACM
  Journal of Theoritical Computer Science
  Journal of Symbolic Logic
  Journal of Logic and Computation

- *Συνέδρια που έχουν καλύψει τον Λογικό Προγραμματισμό*
  International. Workshop on Logic Programming
  International. Conference on Logic Programming
  IEEE Symposium on Logic Programming, Atlantic City.
  North American Conference(s) on Logic Programming (**NACLP**)
  Italian Conference(s) on Logic Programming (**GULP**)
  Fifth Generation Computer Systems (**FGCS**)
  Int. Joint Conference(s) on Artificial Intalligence (**IJCAI**)

- *Μερικές χρήσιμες τεχνικές αναφορές και εγκύκλιοι*
  Technical Reports of the Institute for New Generation Computer Technology, (**ICOT**), Tokyo.
  Technical Reports of the European Computer-Industry Research Centre, (**ECRC**), Munich.
  Logic Programming Newsletters of the Association of Logic Programming (**ALP**).

- *Επιλεγμένες Εργασίες και Ανακοινώσεις*
  Οι εργασίες [6]-[13] αξίζει να αναγνωσθούν για την έξοχή τους παρουσίαση και την βασική ιστορική τους σπουδαιότητα.

- *Εργασίες επισκόπησης*
  Οι εργασίες [14]-[19], που είναι εργασίες επισκόπησης έχουν επιλεχθεί για την εκτεταμένη κάλυψη του θέματος, το διδακτικό ύφος και την τεχνική προσέγγιση.

1.6 Τι είναι ο λογικός προγραμματισμός

Κατά μια πρώτη προσέγγιση, ο λογικός προγραμματισμός είναι μια υπολογιστική κωδικοποίηση που συνδυάζει δύο κύριες αρχές:

- χρησιμοποιεί *λογική* για να αναπαραστήσει τη γνώση,
- χρησιμοποιεί το μηχανισμό *των αποδείξεων* και των *συμπερασμάτων* για να επεξεργαστεί γνώση.

Στο πλαίσιο επίλυσης ενός προβλήματος, η πρώτη αρχή αφορά την αναπαράσταση υποθέσεων και συμπερασμάτων, ενώ η δεύτερη, αφορά τον ορισμό και την παραγωγή των λογικών σχέσεων μεταξύ υποθέσεων και συμπερασμάτων. Ο γενικός σκοπός σε κάθε τέτοιο πλαίσιο είναι να ***συνάγεται*** το επιθυμητό αποτέλεσμα από τις δεδομένες υποθέσεις, με έναν τρόπο που είναι υπολογιστικά βιώσιμος.

Συγκεκριμένα, η καθιερωμένη κωδικοποίηση χρησιμοποιεί ένα ορισμένο υποσύνολο - την *προτασιακή λογική* ή την *κατηγορηματική λογικής α' τάξης* - ως τη γλώσσα αναπαράστασης της γνώσης, και χρησιμοποιεί ένα ορισμένο σύστημα συμπερασματολογίας - *αρχή της απόφασης (resolution principle)* - ως τον μηχανισμό για επεξεργασία της γνώσης. Η Prolog προσθέτει στον πυρήνα του λογικού συστήματος

κλασική λογική + αρχή της απόφασης

ένα συγκεκριμένο είδος ***στρατηγικής ελέγχου*** (control strategy) για την εύρεση αποδοτικής υλοποίησης. Ο συνδυασμός αυτών των χαρακτηριστικών συνήθως καλείται ***καθαρή Prolog*** (pure Prolog) και χαρακτηρίζεται από το γεγονός ότι η λογική ανάλυση των προγραμμάτων που γράφονται σε αυτή δεν λαμβάνουν υπόψη ζητήματα συμπεριφοράς.

Η πρόσθεση στην καθαρή γλώσσα του πυρήνα των ***μη-λογικών*** προτύπων (non-logical primitives), δίνει μια νέα κωδικοποίηση που καλείται ***μη καθαρή Prolog*** (impure Prolog). Πολλά από αυτά τα πρότυπα εξασφαλίζουν (δήθεν θετικές) επιδράσεις της συμπεριφοράς αλλά αλλοιώνοντας την λογική βάση της κωδικοποίησης. Δηλαδή, τα συμπεράσματα στα οποία καταλήγει ένα πρόγραμμα της μη καθαρής Prolog δεν θα προκύπτουν οπωσδήποτε λογικά από αυτό το πρόγραμμα.

Αυτή η διάκριση σε καθαρή και μη καθαρή Prolog προκαλεί ένα ουσιαστικό και άλυτο δίλημμα για τον κλάδο του λογικού προγραμματισμού. Η καθαρότητα εξασφαλίζει μια μεγάλη ποικιλία επιθυμητών αρχών που έχουν σαν στόχο να βελτιώσουν την ποιότητα της μεθοδολογίας του προγραμματισμού πολύ περισσότερο απ' ότι χρησιμοποιώντας τις παραδοσιακές κωδικοποιήσεις.

Ως τώρα όμως, η χρήση του λογικού προγραμματισμού για εφαρμογές του "πραγματικού κόσμου" βασίζεται στον τύπο των μη καθαρών χαρακτηριστικών που υπάρχουν στην Prolog. Το δίλημμα αυτό οδηγεί τελικά στο ερώτημα, αν η ελεγχόμενη κλασική λογική συμπερασματολογία επαρκεί ως μηχανισμός πρακτικού υπολογισμού. Αυτό είναι ένα σημαντικό ζήτημα για έρευνα, και διερευνάται με διάφορους τρόπους, που συμπεριλαμβάνουν τον μετα-προγραμματισμό, τις γλώσσες ελέγχου, τις αρχιτεκτονικές μηχανών και τις μη-κλασικές λογικές. Το αποτέλεσμα αυτών των προσπαθειών είναι ακόμη άγνωστο.

Πως μοιάζει ένα λογικό πρόγραμμα; - Απλά, σαν ένα σύνολο προτάσεων σε μορφή clause που περιγράφει σχέσεις, όπως φαίνεται στο παρακάτω παράδειγμα:

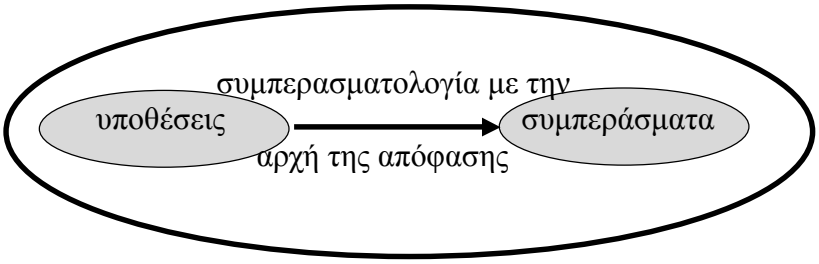
likes(chris, Anyone) if reads(Anyone, these_notes)

```
reads(Anyone, these_notes) if sensible(Anyone)
sensible(you)
```

Ισως μπορείτε να συμπεράνετε από αυτό το πρόγραμμα ότι ο 'chris likes you'. Στα επόμενα κεφάλαια θα δούμε πως γίνεται αυτό και γιατί έχει τόση σημασία.

Ο λογικός προγραμματισμός αντιπροσωπεύει ένα σημείο σύγκλισης των κλάδων της λογικής, της μηχανικής απόδειξης θεωρημάτων και της πληροφορικής. Με την σύγκλιση αυτή οι αντιλήψεις για την *παραγωγή* και την *ερμηνεία* των προτάσεων μπόρεσαν να διασαφηνιστούν. Αυτό το οφείλουμε σε δύο διάσημους ερευνητές της λογικής: τον Gottlob Frege, του οποίου η εργασία οδήγησε στην σημερινή καθιερωμένη κωδικοποίηση της λογικής πρώτης τάξης, και τον Alfred Tarski, ο οποίος διασαφήνισε τις παλιές σημασιολογικές συγχύσεις μεταξύ 'αλήθειας' και 'απόδειξης'.

Λογικός Προγραμματισμός και προτάσεις μορφής clause



Σχήμα 1.10 Το βασικό χαρακτηριστικό του λογικού προγραμματισμού.

Η θεωρία της λογικής με την μορφή προτάσεων του λογικού προγραμματισμού, και ειδικότερα το κύριο θεώρημα της, το οφείλουμε σε έναν άλλο ερευνητή της λογικής τον Jacques Herbrand. Η ανακάλυψη της αρχής της απόφασης (resolution) - ένα τεράστιο βήμα στη μηχανοποίηση απόδειξης θεωρημάτων της προτασιακής λογικής οφείλεται στον J. Alan Robinson. Η προσαρμογή αυτών των εξελίξεων στην υπηρεσία του προγραμματισμού με ηλεκτρονικό υπολογιστή και γενικότερα στην πληροφορική και την τεχνητή νοημοσύνη, ξεκίνησε από πολλούς ξεχωριστά αλλά κυρίως από τους Carl Hewitt, Alain Colmerauer και Robert Kowalski.

Παρακάτω σκιαγραφούνται γεγονότα-κλειδιά των τελευταίων 25 ετών κατά χρονολογική σειρά, τα οποία έχουν επηρεάσει την εξέλιξη του λογικού προγραμματισμού.

1965: Ο J. Alan Robinson δημοσιεύει την αρχή της απόφασης για την απόδειξη θεωρημάτων στην κατηγορηματική λογική.

1971: Η αναφορά Lighthill ευγενικά συμβουλεύει την κυβέρνηση της Μ. Βρεττανίας να μειώσει το επίπεδο της επιδότησης για έρευνα στην τεχνητή νοημοσύνη.

1972: Ο Robert A. Kowalski τυποποιεί την πολύ σημαντική γλώσσα προγραμματισμού για την ερμηνεία της προτασιακής λογικής.

1973: Ο Alain Colmerauer, ο Philippe Roussel και άλλοι στο Πανεπιστήμιο του Aix-Marseille υλοποιούν το πρώτο σύστημα της Prolog.

1974: Ο Robert A. Kowalski παρουσιάζει τη δημιουργική του δημοσίευση στο συνέδριο IFIP-74.

1976: Το πρώτο διεθνές συνέδριο στο Λογικό Προγραμματισμό λαμβάνει χώρα στο Imperial College του Λονδίνου και γίνεται ο πρόγονος μιας διαδοχικής σειράς Διεθνών Συνεδρίων στον Λογικό Προγραμματισμό.

1977: Ο Keith L. Clark δημοσιεύει τα βασικά σημεία που συνδέουν την λογική άρνηση με την πεπερασμένη αποτυχία, δημιουργώντας έτσι ένα θεωρητικό υπόβαθρο για την υποστήριξη της αυτόματης αιτιολόγησης στην Prolog.

1981: Οι William F. Clocksin και Christopher S. Mellish εκδίδουν το παγκοσμίως πρώτο και καλύτερο σε πωλήσεις βιβλίο για την Prolog ("Programming in Prolog", Springer Verlag, Berlin, 1981, *ISBN*: 3-540-11046-1).

1981: Εκπληξη! - Το Ιαπωνικό Υπουργείο Εμπορίου και Βιομηχανίας ανακοινώνει τα εγκαίνια του προγράμματος για τα συστήματα ηλεκτρονικών υπολογιστών πέμπτης γενιάς, αναγνωρίζοντας ότι ο λογικός προγραμματισμός είναι μια τεχνολογία-κλειδί.

1983: Η Κυβέρνηση της Μ.Βρεττανίας δεσμεύεται να χρηματοδοτήσει το Πρόγραμμα Alvey για έρευνα στην τεχνολογία της πληροφορίας, που περιλαμβάνει και μια μικρή εισαγωγή στον λογικό προγραμματισμό.

1984: Εγκαινιάσεται το Journal of Logic Programming με εκδότη αρχικά τον J. Alan Robinson και κατόπιν τον Jean-Louis Lassez.

1984: Ο John W. Lloyd εκδίδει το πρώτο και πολύ σημαντικό βιβλίο για τις θεωρητικές βάσεις του λογικού προγραμματισμού.

1986: Ιδρυση του Association of Logic Programming, του οποίου το έργο περιλαμβάνει την οργάνωση των International και North American Conferences on Logic Programming, καθώς και την έκδοση τακτικών εγκυκλίων (Newsletters).

1987: Από εκεί και πέρα ο λογικός προγραμματισμός αποκτάει μιά ιδιαίτερη ερευνητική αίγλη και χρησιμοποιείται σε μεγάλο πλήθος εφαρμογών, χωρίς όμως ακόμα να έχει περάσει στους εμπορικούς χώρους - εταιρείες κ.λ.π.

1.7 Πλεονεκτήματα του λογικού προγραμματισμού

Πολλά επιχειρήματα έχουν αναπτυχθεί υπέρ του λογικού προγραμματισμού. Το κύριο επιχείρημα είναι ότι δίνει την δυνατότητα σε κάποιον να εκφράσει τη γνώση αποκλειστικά με έναν τρόπο ανεξάρτητο της μηχανής, φτιάχνοντας προγράμματα πιο συμπαγή, ευέλικτα και κατανοητά.

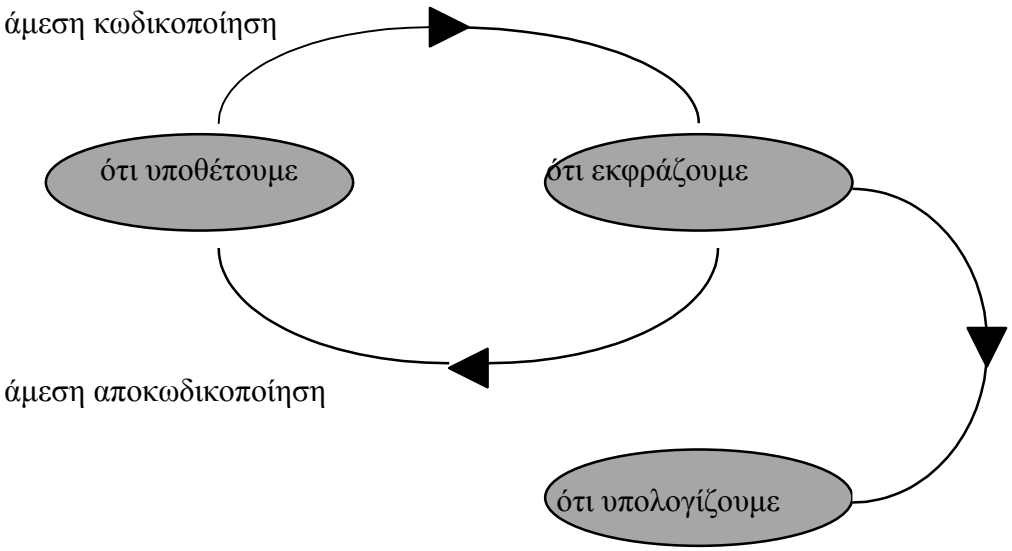
Η αντίληψη ότι η αναπαράσταση και η παραγωγή γνώσης μπορεί να γίνει με παραδοσιακούς τρόπους διαδικαστικού προγραμματισμού, δεν υποστηρίζεται πλέον από την πλειοψηφία των προγραμματιστών, και ακόμη και εκείνοι που την υποστηρίζουν δεν αντιμετωπίζουν τον λογικό προγραμματισμό ως ένα φυσικό τρόπο για να το αντιληφθούν. Οπως και να 'χει, τα κύρια σημεία αυτής της αντίληψης είναι τα εξής:

1. Ο λογικός προγραμματισμός μπορεί να θεωρηθεί ως ο ακρογωνιαίος λίθος του προγραμματισμού με βάση τη γνώση (knowledge-based programming). Η εκφραστικότητα της λογικής δίνει την δυνατότητα κωδικοποίησης υποθέσεων στο πλαίσιο του προβλήματος με έναν άμεσο τρόπο ανεξάρτητο της υλοποίησης. Αντίστροφα, μπορεί κανείς να αποκωδικοποιήσει μια τέτοια κωδικοποίηση με σκοπό να επανακτήσει τις υπονοούμενες υποθέσεις. Επιπλέον, τα συμπεράσματα που υπολογίζονται από την κωδικοποίηση είναι επακριβώς οι λογικές συνέπειες, και επομένως σχετίζονται άμεσα με τις υποθέσεις.

2. *Ευνοεί την μαθηματική υπευθυνότητα* - δίνοντας ακριβείς και απλούς μαθηματικούς χαρακτηρισμούς των σχέσεων μεταξύ

- προγραμμάτων και αποτελεσμάτων που υπολογίζονται από αυτά

- προγραμμάτων και προδιαγραφών
- προγραμμάτων και άλλων προγραμμάτων



Σχήμα 1.11 Επιθυμητά στοιχεία του προγραμματισμού με βάση την γνώση.

3. *Διαχωρίζει τη γνώση από τη χρήση* - κάποιος μπορεί να αλλάξει λεπτομέρειες της υλοποίησης χωρίς να επηρρεάσει την λογική συνέχεια του προγράμματος: ειδικότερα, κάποιος μπορεί να αλλάξει είτε τις στρατηγικές ελέγχου είτε τις αρχιτεκτονικές της μηχανής χωρίς να χρειάζεται οπωσδήποτε να μεταβάλει τα προγράμματα ή την υπονοούμενη γλώσσα.

4. *Προσφέρει ένα ομοιόμορφο παράδειγμα για τεχνολογία λογισμικού* - δηλαδή, μια κωδικοποίηση που χρησιμεύει για την κατασκευή και διαχείριση προγραμμάτων, προδιαγραφών και σχετικών εργαλείων λογισμικού.

5. *Μπορεί να διαμορφωθεί ή να επεκταθεί με φυσικούς τρόπους* για να υποστηρίξει ειδικούς τύπους γνώσης, όπως γνώση μετα-επιπέδου ή γνώση υψηλής τάξης, ή για την επανακατασκευή φαινομενικά μη-λογικών κωδικοποιήσεων.

6. Οι κύριες αρχές του σκεπτικού της μπορούν να διδαχθούν και να κατανοηθούν χωρίς την αναφορά στην τεχνολογία του ηλεκτρονικού υπολογιστή - *μπορεί να συμβάλει άμεσα σε άλλους μη υπολογιστικούς επιστημονικούς κλάδους* βασιζόμενη σε ακριβή μέσα έκφρασης και αιτιολόγησης.

7. *Είναι αποτελεσματική*! - δηλαδή, με βάση αυτήν έχουν φτιαχθεί βιώσιμες τεχνολογίες προγραμματισμού.

1.8 Μειονεκτήματα του λογικού προγραμματισμού

Πριν δεκαπέντε χρόνια, ο λογικός προγραμματισμός φαινόταν να παραμένει μακριά από το υπολογιστικό πεδίο. Αλλά κατά τα επόμενα δέκα χρόνια, φάνηκε ότι ήταν μεγάλη η ανάγκη αυτή, και πολλοί από τους ερευνητές του χώρου ένιωσαν ότι υπήρχε μεγάλη

πιθανότητα να μπορέσει ουσιαστικά να αντικαταστήσει παλαιότερες και σχετικά λιγότερο ικανοποιητικές κωδικοποιήσεις.

Οι Ιάπωνες είχαν επενδύσει πολλά στο λογικό προγραμματισμό για τους σκοπούς του εθνικού τους έργου για τα συστήματα ηλεκτρονικών υπολογιστών πέμπτης γενιάς, και το πλήθος των τομέων όπου χρησιμοποιήθηκε η Prolog ανέρχονταν σε δεκάδες - ή και μερικές εκατοντάδες χιλιάδες. Υπήρξε τεράστιο ενδιαφέρον για τα έμπειρα συστήματα, πολλά από τα οποία γράφθηκαν σε Prolog. Οι βιομηχανίες στη Μ.Βρεττανία και αλλού στην Ευρώπη συμμετείχαν στα διάφορα ακαδημαϊκά προγράμματα που χρηματοδοτούνταν για έρευνα και ανάπτυξη, και πολυάριθμες εταιρείες ιδρύθηκαν με σκοπό να προμηθεύουν τεχνολογία λογικού προγραμματισμού.

Παρά τον ενθουσιασμό, ο λογικός προγραμματισμός δεν είχε τόσο μεγάλο αντίκτυπο όπως αναμενόταν. Ενας λόγος ίσως να είναι ότι, αρχικά, ήταν ανεπαρκής η επένδυση για την προσφορά δυναμικών περιβαλλόντων, εργαλείων και γλωσσικών χαρακτηριστικών, και έτσι πολλοί υποψήφιοι χρήστες δεν εξυπηρετήθηκαν επαρκώς. Τα τελευταία χρόνια, η έλλειψη ευκολιών για αριθμητικές, ευμετάβλητες δομές, τύπους, είσοδο-έξοδο, γραφικά, διαχείριση αρχείων και πολλά άλλα είχαν ανασταλτική επίδραση σε μεγάλα τμήματα της προγραμματιστικής κοινότητας. Οταν τελικά δόθηκαν αυτές οι ευκολίες, συχνά ήταν σε βάρος της καθορισμένης σημασιολογίας της κωδικοποίησης, οδηγώντας έτσι σε περισσότερες δυσχέρειες. Η Prolog θεωρήθηκε - και χρησιμοποιήθηκε - από πολλούς ως μια φανερά μη-δηλωτική γλώσσα της οποίας το ιδιαίτερο χάρισμα ήταν λίγο-πολύ η ευκολία στην ενσωματωμένη ομοιομορφία και αναζήτηση.

Η υπερβολική συσχέτιση με τον κλάδο της τεχνητής νοημοσύνης είχε επίσης αρνητικές συνέπειες. Γνήσια προβλήματα της τεχνητής νοημοσύνης έτειναν να αντιμετωπίζονται σαν χαρακτηριστικά προβλήματα του λογικού προγραμματισμού. Για παράδειγμα, μια συχνή κατηγορία ήταν ότι η Prolog είχε ένα ‘πρόβλημα με τη λογική άρνηση’. Στην πραγματικότητα, η ευκολία της λογικής άρνησης μέσω αποτυχίας (negation-by-failure) που παρεχόταν ήταν ένα πλεονέκτημα για τους χρήστες που απαιτούσαν έναν ενσωματωμένο μηχανισμό αυτόματου συμπεράσματος, η σημασιολογία του οποίου ήταν θέμα συζητήσεων και διαμαχών από πολύ πριν εφευρεθεί ο λογικός προγραμματισμός. Εναν τέτοιο μηχανισμό προσπάθησαν να τον υποστηρίξουν και άλλες γλώσσες προγραμματισμού.

Πέρα από τέτοιου είδους άσχετα ζητήματα, υπήρχαν, δικαιολογημένα, έμφυτες αδυναμίες στην ίδια τη γλώσσα. Δεν έχει βρεθεί ένας απόλυτα ικανοποιητικός τρόπος για την αναπαράσταση εκείνων των υπολογιστικών σκεπτικών τα οποία, σε μια συμβατική γλώσσα, υποστηρίζονται άμεσα από ενσωματωμένους μηχανισμούς μεταβλητών κατάστασης και επιτακτικών ενεργειών.

Για παράδειγμα, αν κάποιος θέλει να ανανεώσει μια βάση δεδομένων τότε ο μόνος τρόπος να περιγράψει *πλήρως* μια τέτοια πράξη με καθαρό συμβολισμό προτάσεων τύπου Horn (ειδικού τύπου λογικές προτάσεις), είναι να αναπαραστήσει την βάση δεδομένων με έναν σύνθετο όρο - η άμεση υλοποίηση του οποίου είναι πρακτικά αδύνατη.

Παίρνοντας ένα άλλο παράδειγμα, κάποιος ίσως να θέλει να μεταβάλει επανειλλημένα μια μεγάλη μήτρα μέχρις ότου φθάσει σε μια ποθητή κατάσταση. Για να το επιτύχει με προτάσεις τύπου Horn με τρόπο που να χρησιμοποιεί οικονομικά την μνήμη, πρέπει να αγωνιστεί να βρει την κατάλληλη αναδρομική διαδικασία για την επανάληψη, και μετά να προσθέσει με πολύπλοκες και όχι πολύ πειστικές ιστορίες για αφανείς διαδικασίες ‘αναδρομής ουράς’ (tail-recursion) και ‘συλλογής σκουπιδιών’ (garbage-collection).

Τα λογικά προγράμματα *υπόκεινται* σε μια λειτουργική μεταγλώττιση η οποία συμβάλει στην υποστήριξη σκεπτικών των ενεργειών που προσανατολίζονται στην κατάσταση. Πολλοί ιδεαλιστές του λογικού προγραμματισμού έχουν υποστηρίξει την

διπλή ανάπτυξη των δηλωτικών και λειτουργικών μεταγλωττίσεων έτσι ώστε ο προγραμματιστής να μπορέσει να ολοκληρώσει αρμονικά τις λογικές και υπολογιστικές του εμπνεύσεις κατά την κατασκευή του προγράμματος. Δυστυχώς, αυτή η φιλοσοφία κινδυνεύει από το γεγονός ότι η συνηθισμένη λειτουργική μεταγλώττιση αντανακλά μόνο την *απλή, γενικού σκοπού* διαδικασία απόδειξης που χρησιμοποιείται για την εκτέλεση του προγράμματος.

Σχετικά με αυτό το θέμα, ο Dov Gabbay έδωσε μια αναλογία [2]. Ενας μαραγκός φτιάχνοντας ένα αντικείμενο χρησιμοποιεί *πολλά, ειδικού σκοπού* εργαλεία για να το διαμορφώσει αποτελεσματικά. Εχοντας απλώσει τα κατάλληλα εργαλεία τα παίρνει και τα χρησιμοποιεί όπως και όποτε χρειάζεται, ανάλογα με το φυσικά του ένστικτα. Εστω τώρα ότι κατάσχωνται όλα αυτά τα εργαλεία και αντικαθιστώνται από ένα απλό, σταθερό περιστρεφόμενο πριόνι στημένο στο ταβάνι. Κατόπιν για να φτιάξει το ίδιο αντικείμενο πρέπει να επινοήσει παράλογους τρόπους για να τι κρατήσει μπροστά από το πριόνι έτσι ώστε να *προσομοιώσει* τις διάφορες ενέργειες που έγιναν προηγουμένως χρησιμοποιώντας τα εργαλεία όπως έχει συνηθίσει. Το ανάλογο του περιστρεφόμενου πριονιού στον λογικό προγραμματισμό είναι η σταθερή διαδικασία απόδειξης. Η δυσκολία στην χρήση του πριονιού είναι η δυσκολία στο να φτιάξουμε την απλή διαδικασία απόδειξης για την προσομοίωση μιας ποικιλίας υπολογιστικών ενεργειών. Και τα συμπτώματα αυτής της δυσκολίας αντανακλώνται στις δυσνόητες λογικές δομές που χρειάζονται σε ένα πρόγραμμα για να επιτευχθούν οι προσομοιώσεις. Αυτή η αναλογία ίσως να μην είναι επαρκής, αλλά εξασφαλίζει μια ορισμένη δυνατότητα που οι υποστηρικτές του λογικού προγραμματισμού θα πρέπει να μελετήσουν καλά.

Ο δηλωτικός προγραμματισμός δεν αρέσει σε όλους. Υπάρχουν πολλοί που απλά προτιμούν την φανερή λειτουργική μορφή των προγραμμάτων που προσανατολίζονται στη μηχανή - γι' αυτούς, υπάρχει μεγαλύτερη ψυχολογική ικανοποίηση με την άσκηση ενεργού ελέγχου πάνω στα 'κινούμενα τμήματα' ενός υπολογισμού απ' ότι στην περισσότερο παθητική δουλειά της δήλωσης του τι πρέπει να επιτύχει ο υπολογισμός. Αυτή η προτίμηση ίσως διατηρηθεί παρά τους οποιουσδήποτε ισχυρισμούς που γίνονται ή αποδεικνύονται σχετικά με τα πλεονεκτήματα του (σημερινού) λογικού προγραμματισμού.

Ελλείψεις διαφόρων τύπων διακρίνονται ακόμα και από τους πρωταθλητές του προγραμματισμού με βάση τη λογική. Τα παράπονά τους στοχεύουν κυρίως την τρέχουσα προσήλωση του λογικού προγραμματισμού στη λογική *πρώτης τάξης*. Επιθυμούν να έχουν εκτελέσιμες λογικές στις οποίες μπορεί κανείς να εκφράσει άμεσα, για παράδειγμα, μη μονοτονική αιτιολόγηση και ιδέες όπως, ιδιοκτησία και υποχρέωση. Κατά την άποψη τους, η αναπαράσταση αυτών των ιδεών με την χρήση μεταλογικών χαρακτηριστικών του λογικού προγραμματισμού απλά ξεφεύγει από το θέμα, και εξαρτάται από τις τεχνητές προσομοιώσεις του τι πραγματικά θέλει να εκφράσει ο προγραμματιστής. Θα χρειαστεί αρκετή συγκριτική μελέτη πριν αυτό το θέμα επιλυθεί ικανοποιητικά.

1.9 Συμπεράσματα

Στην εισαγωγή που προηγήθηκε κάναμε μιά πολύ περιορισμένη ξενάγηση στη ΤΝ, και στις εφαρμογές της. Εγινε επίσης μιά εισαγωγική αναφορά στο Λογικό Προγραμματισμό με σύντομη ιστορική αναφορά, τις έντυπες πηγές, τα πλεονεκτήματα και τα μειονεκτήματα του. Είναι όμως τόσο περιορισμένη η ξενάγηση αυτή που μπορεί να παραπλανήσει τον αναγνώστη ως προς την έκταση και το βάθος τόσο της ΤΝ και των εφαρμογών της, όσο και του Λογικού Προγραμματισμού. Στην επόμενη παράγραφο παρατίθεται κάποια περιορισμένη, πολύ γενική και εισαγωγική βιβλιογραφία γιά τον αναγνώστη που θα ήθελε

να αποκτήσει μιά καλύτερη αντίληψη τόσο γιά τη ΤΝ όσο και για το Λογικό Προγραμματισμό.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] R. A. Kowalski, "Logic for Problem Solving", Elsevier North Holland, 1979

[2] C. J. Hogger, "Essentials of Logic Programming", Oxford University Press, 1990

[3] J. W. Lloyd, " Foundations of Logic Programming ", Springer Verlag, 1987

[4] L.Sterling, E.Shapiro, "The Art of Prolog : Advanced Programming Techniques", MIT Press, 1986.

[5] Ivan Bratko, "Prolog : Programming for Artificial Intelligence", Addison Wesley, 1990

[6] J. A. Robinson, "A machine-oriented logic based on the resolution principle", Journal of the ACM **12**, 1965, pp.23-41.

[7] P. J. Hayes, "Computation and deduction" Proc. of the 2nd Symposium on the Mathematical Foundations of Computer Science, Czechoslovak Academy of Sciences, 1973, pp.105-118.

[8] R. A. Kowalski, "Predicate logic as a programming language" Proc. of IFIP-74, North Holland Publishing Company, Amsterdam, 1974, pp.569-574.

[9] R. A. Kowalski and M. H. van Emden, "The semantics of predicate logic as a programming language", Journal of the ACM **23**, 1976, pp.733-742.

[10] M. H. van Emden, "Programming in resolution logic" Machine Intelligence **8**, Ellis Horwood Ltd. Chichester, 1977, pp.266-299.

[11] K. L. Clark and S.-A. Tarnlund, "A first order theory of data and programs", Proc. of IFIP-77, North Holland Publishing Company, Amsterdam, 1977, pp.939-944.

[12] K. L. Clark, "Negation as failure", Logic and Data Bases (*Eds*. H. Gallaire and J. Minker), Plenum, New York, 1978, pp.293-322.

[13] R. A. Kowalski, "Algorithm = logic + control", Comm. of the ACM **22**, 1979, pp.424-431.

[14] R. A. Kowalski, "Logic programming" Invited Paper in Proc. of IFIP-83, North Holland Publishing Company, Amsterdam, 1983, pp.133-145.

[15] R. A. Sammut and C. A. Sammut, "Prolog: a tutorial introduction" Australian Computer Journal **15**, 1983, pp.42-51.

[16] M. R. Genesereth and M. L. Ginsberg, “Logic programming” Comm. of the ACM **28**, 1985, pp.933-941.

[17] L. Wos *et al*., “An overview of automated reasoning and related fields”, Journal of Automated Reasoning **1**, 1985, pp.5-48.

[18] C. J. Hogger and R. A. Kowalski, “Logic programming”, Encyclopedia of Artificial Intelligence (Ed. S. C. Shapiro), John Wiley & Sons, New York, 1987, pp.544-558.

[19] R. A. Kowalski, “The early years of logic programming”, Comm. of the ACM **31**, 1988, pp.38-43.

[20] N.J.Nilsson, "Principles of Artificial Intelligence", Springer-Verlag, 1982

[21] A.Barr & E.A.Feigenbaum, "The Handbook of Artificial Intelligence", Vol.2., Pitman Books LTd., 1983.

[22] P.R.Cohen & E.A.Feigenbaum, "The Handbook of Artificial Intelligence", Vol.3., Pitman Books LTd., 1984.

[23] E.Rich, "Artificial Intelligence", McGraw-Hill, 1983.

[24] R.Forsyth, "Expert Systems, Principles and Case Studies", Chapman and Hall Computing, 1984.

[25] P.H.Winston, "Artificial Intelligence", Addison-Wesley publishing Company, 1984

[26] E.Charniak, D.McDermott, "Introduction to Artificial Intelligence", Addison-Wesley, 1985.

[27] R.Frost, "Introduction to knowledge Base Systems", Collins, 1986.

[28] M.R.Genesereth, N.J.Nilsson, "Logical Foundations of Artificial Intelligence", Morgan Kaufmann Publishers Inc., 1988.

2

Προτασιακή Λογική

2.1 Εισαγωγή

Οπως έχουμε αναφέρει στο προηγούμενο κεφάλαιο, ένα από τα μέσα γιά την αναπαράσταση της γνώσης είναι η συμβολική λογική. Το πιό απλό σύστημα συμβολική λογικής είναι η Προτασιακή Λογική (ΠΛ, propositional logic) ή Λογική μηδενικής τάξης όπως μερικές φορές αναφέρεται. Η ΠΛ αποτελεί βέβαια ένα πολύ στοιχειώδες μέσο αναπαράστασης της γνώσης, αλλά βασίζεται σε αυστηρούς ορισμούς (είναι καλά ορισμένος) και αποτελεί το πρώτο βήμα γιά την Κατηγορηματική Λογική α' τάξης (ΚΛ, first order logic).

Ετσι σε ό,τι αναφέρουμε στη ΠΛ θα πρέπει να ξέρουμε ότι έχει μια αντίστοιχη επέκταση στη ΚΛ. Σε ένα όποιοδήποτε σύστημα φορμαλιστικής (τυπικής) γλώσσας, άρα και σε ένα σύστημα λογικής, ορίζουμε με αυστηρό τρόπο το συντακτικό, την σημασιολογία και κάποιο σύστημα απόδειξης.

Το συντακτικό αναφέρεται στον ορισμό της τυπικής γλώσσας, δηλαδή σε ένα σύνολο ορισμών που αναφέρονται στο αποδεκτό αλφάβητο, τα λογικά σύμβολα, και τις αποδεκτές προτάσεις της γλώσσας.

Η σημασιολογία αναφέρεται στην ανάθεση κάποιας ερμηνείας στα σύμβολα της γλώσσας και εξετάζει τον τρόπο με τον οποίο οι προτάσεις της γλώσσας σχετίζονται με το πραγματικό τους περιεχόμενο.

Το σύστημα απόδειξης αποτελείται από ένα σύνολο κανόνων συμπερασμού οι οποίο μπορούν να εφαρμοστούν σε ένα σύνολο προτάσεων ώστε να προκύπτουν νέες προτάσεις.

Το συντακτικό της ΠΛ θα το ορίσουμε μοναδικά. Οσον αφορά στη σημασιολογία θα βρούμε πολλούς ισοδύναμους τρόπους με τους οποίους θα μπορούμε να υπολογίσουμε την τιμή αλήθειας μιάς πρότασης. Τα συστήματα απόδειξης που θα μελετήσουμε θα αποτελέσουν και τη βάση γιά αυτοματοποιημένη συμπερασματολογία που είναι και το κατ'εξοχήν ενδιαφέρον τμήμα από πλευράς ΤΝ.

2.2 Συντακτικό : η γλώσσα της ΠΛ

Η Προτασιακή Λογική ασχολείται με προτάσεις που αποφαίνονται γιά κάτι, δηλαδή κάνουν κάποιες διαπιστώσεις. Οι πιό απλές προτάσεις ονομάζονται ***ατομικές προτάσεις*** ή ***άτομα*** (atomic propositions, atoms). Οι προτάσεις αυτές συνδεόμενες με τους ***λογικούς συνδέσμους*** (logical connectives) συνθέτουν τις ***σύνθετες προτάσεις*** (compound propositions). Οι λογικοί σύνδεσμοι είναι : $\neg$ (not, άρνηση), $\wedge$ (and, και, σύζευξη), $\vee$ (or, ή, διάζευξη), $\leftarrow$, $\rightarrow$ (implies, συνεπαγωγή), $\leftrightarrow$ (equivalent, ισοδυναμία).

Ορισμός 2.1 Η γλώσσα της ΠΛ αποτελείται από

- τα σύμβολα των ατόμων : $p_1, p_2, ...$
- τα σύμβολα των συνδέσμων : $\neg \ \wedge \ \vee \ \leftarrow \ \rightarrow \ \leftrightarrow$
- τις παρενθέσεις : ( )

Οι προτάσεις της ΠΛ κατασκευάζονται από τα παραπάνω σύμβολα με ορισμένους κανόνες μετασχηματισμού :

Ορισμός 2.2 Οι προτάσεις της ΠΛ

- Τα άτομα είναι προτάσεις
- Εάν φ,ψ είναι προτάσεις τότε και οι εκφράσεις $\neg\varphi$, $\varphi\wedge\psi$, $\varphi\vee\psi$, $\varphi\leftarrow\psi$, $\varphi\rightarrow\psi$, $\varphi\leftrightarrow\psi$, (φ) είναι επίσης προτάσεις.

Παράδειγμα 2.1 Τα παρακάτω είναι προτάσεις της ΠΛ

$\neg(p3\leftarrow ((p1\vee p2) \wedge p3))$ $p1\leftarrow (p3\wedge p4)$ $p4$

Τα παρακάτω δεν είναι προτάσεις της ΠΛ

$\neg p3\leftarrow$ $\leftarrow (p3\wedge p4)$ $p4\vee$

Γιά να αποφύγουμε την υπερβολική χρήση παρενθέσεων, ορίζουμε τις παρακάτω προτεραιότητες στους λογικούς συνδέσμους : $\neg \ \wedge \ \vee \ \rightarrow \ \leftrightarrow$. Κατ'αυτό τον τρόπο οι προτάσεις απλοποιούνται : η πρόταση $(p \rightarrow (q \wedge (\neg r)))$ γίνεται $p\rightarrow q\wedge\neg r$.

2.3 Αναπαράσταση της γνώσης στη ΠΛ

Η Πληροφορική είναι η επιστήμη της πληροφορίας. Η πληροφορία προκύπτει από την επεξεργασία των δεδομένων, τα οποία με τη σειρά τους είναι τα στοιχεία εκείνα που συνθέτουν ένα γεγονός, ένα φαινόμενο, κ.λ.π. Σε ένα μετεωρολογικό φαινόμενο, γιά παράδειγμα, έχουμε πολλά δεδομένα : υγρασία, ατμοσφαιρική πίεση, κατευθύνσεις και εντάσεις ανέμων, κ.λ.π. Η κωδικοποίηση αυτών των δεδομένων σε μιά επεξεργάσιμη μορφή και η αποθήκευσή τους σε κάποιο μαγνητικό μέσο αποτελεί αυτό που αποκαλούμε πληροφορία.

Ομως, οι φυσικοί νόμοι που προκαλούν τα φαινόμενα δεν ανήκουν ούτε στα δεδομένα ούτε βέβαια στις πληροφορίες. Η φύση τους και τα αποτελέσματά τους μελετώνται από τις διάφορες επιστήμες και γιά την αναπαράστασή τους προκύπτουν προσεγγιστικοί κανόνες/τύποι που μας βοηθούν να κατανοήσουμε, σε κάποιο βαθμό, τον τρόπο αλληλοδιαδοχής των δεδομένων, την αιτιότητα και τα αποτελέσματά της, κ.λ.π.

Η Πληροφορική προσπάθησε να αποτυπώσει τέτοιους νόμους, προσπαθώντας να εκφράσει την αλληλοδιαδοχή των τύπων σαν αλληλοδιαδοχή προβλέψιμων στιγμιοτύπων της πληροφορίας ενός αλγορίθμου. Με αυτό τον τρόπο αναπτύχθηκαν προγράμματα τα οποία βοηθούν σήμερα τις επιστήμες στην πρόβλεψη φαινομένων, την αυτόματη επεξεργασία μεγάλου αριθμού δεδομένων, κ.λ.π.

Οταν εκτελείται ένας τέτοιος αλγόριθμος, ουσιαστικά πραγματοποιείται η αναπαράσταση της γνώσης γιά ένα φαινόμενο μέσω της εκτέλεσης στον Η/Υ μιάς

διαδικασίας, συνάρτησης κ.λ.π. Δηλαδή, αναπαρίσταται μια αφηρημένη οντότητα, η γνώση, μέσω μιας διαδικασίας. Αυτό στη ΤΝ το ονομάζουμε **διαδικαστική αναπαράσταση της γνώσης** (procedural knowledge representation). Εκτός αυτού του τύπου αναπαράστασης στην ΤΝ μπορούμε να αναπαριστούμε τη γνώση που αφορά ένα φαινόμενο με ένα πιό προφανή τρόπο, οπότε μιλούμε γιά **δηλωτική αναπαράσταση της γνώσης** (declarative knowledge representation).

Βλέπουμε λοιπόν πως η έννοια της γνώσης διαφοροποιείται από την έννοια της πληροφορίας. Εννοιολογικά, η γνώση σε μιά τέτοια περίπτωση, δεν είναι ο φυσικός νόμος, αλλά η αντίληψη αυτού του νόμου εκ μέρους ενός νοήμονος όντος. Αυτές βέβαια οι παρατηρήσεις μπορούν έυκολα να επεκταθούν σε οποιοδήποτε αντικείμενο (συγκεκριμένο ή αφηρημένο) αποτελεί περιεχόμενο ενδιαφέροντος και απασχόλησης ενός νοήμονος όντος, εφόσον αυτό το αντικείμενο μας προσφέρει δυνατότητες συμβολικής αναπαράστασης και εμπίπτει στις διαδικασίες επεξεργασίας κάποιου τύπου λογικής. Στη περίπτωση λ.χ. των Μαθηματικών έχουμε αφηρημένα αντικείμενα που αναπαρίστανται συμβολικά στη γλώσσα των Μαθηματικών, και υπόκεινται λογική επεξεργασία των αποδείξεων.

Ορισμός 2.3 Ορισμός της γνώσης

Γνώση είναι το σύνολο των διαφόρων τύπων συσχετίσεων μεταξύ συγκεκριμένων ή αφηρημένων αντικειμένων που εμπίπτουν στην αντιληπτική ικανότητα, το ενδιαφέρον και την απασχόληση ενός νοήμονος όντος, προσφέρουν δυνατότητες συμβολικής αναπαράστασης και μπορούν να υποστούν τη επεξεργασία κάποιας λογικής διεργασίας.

Στη ΠΛ είναι δυνατό να πραγματοποιήσουμε αναπαράσταση της γνώσης. Η γνώση την οποία έχει ένα ανθρώπινο ον γιά τον κόσμο, αποθηκεύεται με κάποιο τρόπο, που δεν έχει επαρκώς διερευνηθεί, πιθανότατα, και ίσως όχι μόνο, σε κάποιο τμήμα του εγκεφάλου του. Πιθανότατα να αποθηκεύεται γνώση και στα ίδια τα κύτταρα, όπως μας πληροφορεί η Βιολογία, μέσω των αλυσίδων DNA.

Η πλέον χειροπιαστή και προσπελάσιμη για μας γνώση, είναι συνήθως αυτή που εκφράζεται με προτάσεις φυσικής γλώσσας. Οπως βέβαια είδαμε στον προηγούμενο ορισμό, η γνώση αυτή καθ'εαυτή δεν είναι η έκφρασή της μέσα από προτάσεις φυσικής γλώσσας, αλλά η αντίληψη του κόσμου, των φαινομένων κ.λ.π. από ένα άνθρωπο που μπορεί με κάποιο τρόπο να αποτυπωθεί σαν ένα σύνολο μαθηματικών σχέσεων. Ομως, επειδή δεν έχουμε τα μέσα να αποκτήσουμε άμεση αντίληψη αυτών των σχέσεων, πρέπει αναγκαστικά να στηριχθούμε στα μέσα έκφρασης ενός ανθρωπίνου όντος όπως η φυσική γλώσσα.

Βέβαια, η φυσική γλώσσα δεν είναι το μοναδικό μέσο έκφρασης ενός νοήμονος όντος. Αλλα μέσα είναι η κίνηση, οι δραστηριότητές του στην Επιστήμη και στην Τέχνη, ο τρόπος που αντιδρά, σχεδιάζει κ.λ.π. Επίσης υπάρχουν μέσα έκφρασης που είναι συλλογικά και όχι ατομικά, δηλαδή μέσα έκφρασης μιάς κοινωνίας νοημόνων όντων, όπως ο Πολιτισμός, τα ήθη, τα έθιμα, οι συλλογικές νοοτροπίες, κ.λ.π. Ομως, παρόλο που τέτοιες εκφράσεις κρύβουν τεράστιες δεξαμενές ατομικής και συλλογικής γνώσης, είναι ακόμη πολύ δύσκολο να μελετηθούν και να αποτυπωθούν με τον τρόπο που απαιτεί η ΤΝ. Η φυσική γλώσσα όμως μας προσφέρει ένα πλούσιο και πρώτης τάξεως υλικό γιά να ξεκινήσουμε τους πειραματισμούς μας με την αναπαράσταση και επεξεργασία της γνώσης.

Γιά το σκοπό αυτό θα χρησιμοποιήσουμε προτάσεις της φυσικής γλώσσας σαν πηγές έκφρασης της γνώσης. Υποθέστε την πρόταση : "Εάν έχει πολύ υγρασία και μεγάλη θερμοκρασία, δεν αισθάνομαι άνετα". Μπορούμε να ορίσουμε τα p1="έχει πολύ υγρασία",

p2="έχει μεγάλη θερμοκρασία", p3="αισθάνομαι άνετα", οπότε η αρχική πρόταση γίνεται ($\neg$p3) $\leftarrow$ (p1$\wedge$p2)

Για να μετατρέψουμε μιά πρόταση φυσικής γλώσσας σε πρόταση της ΠΛ πρέπει να βρούμε στην πρόταση της φυσικής γλώσσας τις στοιχειώδεις υποπροτάσεις που θα αναπαρασταθούν σαν ατομικές προτάσεις και που συνδεονται μεταξύ τους με συνδέσμους όπως ... ή ..., ... και ... , εάν ... τότε ..., όχι, δεν, κ.λ.π. Η μετατροπή αυτή ούτε είναι πάντοτε εύκολη, ούτε είναι πάντοτε δυνατή. Αυτό θα γίνει πιό προφανές στο επόμενο κεφάλαιο που αναφέρεται στην Κατηγορηματική Λογική.

Παράδειγμα 2.2 Ας υποθέσουμε ότι διαθέτουμε τις παρακάτω προτάσεις, και θέλουμε να τις μετατρέψουμε σε προτάσεις της ΠΛ.

1. Θα παρακολουθήσω το έργο, εάν έχω έγχρωμη τηλεόραση και δεν είμαι απασχολημένος.
2. Θα γράψω το έργο, εάν έχω video και είμαι απασχολημένος.
3. Θα παρακολουθήσω το έργο εάν το έχω γράψει.
4. Εχω έγχρωμη τηλεόραση.
5. Εχω video.

Κατ'αρχήν ορίζουμε τις ατομικές προτάσεις :

p="Θα παρακολουθήσω το έργο",
q="έχω έγχρωμη τηλεόραση",
r="είμαι απασχολημένος",
s="θα γράψω το έργο",
t="έχω video"

Στη συνέχεια με βάση τις προτάσεις της φυσικής γλώσσας και του ορισμούς των ατομικών προτάσεων, προσπαθούμε να συνθέσουμε τις προτάσεις της προτασιακής λογικής. Ετσι, η πρόταση

'Θα παρακολουθήσω το έργο, εάν έχω έγχρωμη τηλεόραση και δεν είμαι απασχολημένος',

υπονοεί την πρόταση

'*εάν* (έχω έγχρωμη τηλεόραση)
και (δεν είμαι απασχολημένος)
τότε (θα παρακολουθήσω το έργο).

που τελικά γίνεται η πρόταση :

$q \wedge \neg r \rightarrow p$

Με βάση αυτή την ανάθεση οι παραπάνω προτάσεις γίνονται :

$\Sigma = \{ \ q \wedge \neg r \rightarrow p, \ \ t \wedge r \rightarrow s \ , \ \ s \rightarrow p, \ \ q, \ \ t \}$.

Μέσα στα πλαίσια των περιορισμών της ΠΛ θα ασχοληθούμε στη συνέχεια με τη σημασιολογία.

2.4 Σημασιολογία : Η ερμηνεία των προτάσεων

Η αναπαράσταση της γνώσης στην ΠΛ είναι πολύ στοιχειώδης : Βασίζεται στην ανάθεση κάποιων προτάσεων της φυσικής (ή άλλης) γλώσσας σε ατομικές προτάσεις, και η χρησιμοποίηση συνδέσμων γιά την σύνθεση προτάσεων που αφορούν πιό σύνθετα φαινόμενα. Η αναπαράσταση της γνώσης βέβαια στην ΠΛ συναρτάται άμεσα με την ***σημασιολογία*** ή σημαντική (semantics), η οποία με την σειρά της συναρτάται με την ***ερμηνεία*** (interpretation) των προτάσεων της ΠΛ.

Η έννοια της ερμηνείας των συμβόλων μιάς τυπικής γλώσσας σχετίζεται με τη νοητική κατασκευή που συνθέτει ένα νοήμον ον όταν πέφτουν στην αντίληψή του προτάσεις αυτής της γλώσσας.

Ποιά είναι η ερμηνεία των προτάσεων μιάς άγνωστης σε μας φυσικής γλώσσας; Αν για παράδειγμα δεν γνωρίζετε ιερογλυφικά πως θα ερμηνεύσετε τα σύμβολα της γλώσσας αυτής όταν πέσει στα χέρια σας ένας αρχαίος αιγυπτιακός πάπυρος; Η ερμηνεία λοιπόν δεν είναι κάτι απλό και εξαρτάται από την πολυπλοκότητα και την εκφραστική ικανότητα της προς μελέτη γλώσσας.

Στη ΠΛ τα πράγματα είναι πολύ απλά. Η μόνη δυνατή ερμηνεία μιάς πρότασης της ΠΛ είναι η ***εκτίμησή*** της (valuation), η απονομή δηλαδή μιάς ***τιμής αλήθειας*** (truth value) στην πρόταση. Ας υποθέσουμε, γιά λόγους απλότητας, ότι η τιμή αλήθειας μιάς πρότασης δεν έχει καμιά σχέση με την υποκειμενική ή αντικειμενική αλήθεια που ορίζονται στην φιλοσοφία.

Η εκτίμηση μιάς πρότασης στη ΠΛ ορίζεται με βάση τις εκτιμήσεις των ατομικών προτάσεων λαμβάνοντας υπόψη τους ***πίνακες αλήθειας*** (truth tables) των λογικών συνδέσμων. Οι πίνακες αλήθειας των λογικών συνδέσμων ορίζονται παρακάτω. Με βάση αυτούς τους πίνακες μπορούμε να βρούμε συγκεκριμένους τύπους συναρτήσεων γιά τις διάφορες περιπτώσεις συνδέσμων.

I(p)	I(q)	I(¬p)	I(p∧q)	I(p∨q)	I(p→q)	I(p↔q)
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Από τα προηγούμενα φαίνεται ότι ουσιαστικά μιά εκτίμηση ορίζεται από την ανάθεση τιμών αλήθειας στις ατομικές προτάσεις. Ετσι αν η πρόταση περιέχει n διακεκριμένα άτομα τότε υπάρχουν 2^n εκτιμήσεις της πρότασης.

Στη ΠΛ υπάρχουν δύο μόνο τιμές αλήθειας : αληθές (true) και ψευδές (false). Οι τιμές αυτές αναπαρίστανται και σαν 1 ή t, και 0 ή f αντίστοιχα. Υπάρχουν και άλλοι συμβολισμοί που χρησιμοποιούνται σε διάφορα εγχειρίδια Λογικής, στους οποίους όμως δεν θα επεκταθούμε.

Ορισμός 2.4 Η ερμηνεία μιάς πρότασης στη ΠΛ
Δεδομένης μιάς πρότασης φ, έστω $p_1,p_2,..,p_n$, τα άτομα της πρότασης. Μιά ερμηνεία I της φ είναι μιά ανάθεση τιμών αλήθειας σε κάθε άτομο p_i, i=1,...,n.

Η ανάθεση μιάς τιμής αλήθειας ενός ατόμου p στην I συμβολίζεται με I(p)=1 (ανάθεση αληθούς τιμής), I(p)=0 (ανάθεση ψευδούς τιμής).

Ορισμός 2.5 Η αποτίμηση της τιμής αλήθειας μιας πρότασης στη ΠΛ
Η πρόταση φ είναι αληθής κάτω από μιά ερμηνεία I εάν η φ αποτιμάται ως αληθής στην I. Αλλιώς η φ είναι ψευδής στην ερμηνεία I.

Η αποτίμηση μιάς τιμής αλήθειας μιάς πρότασης φ στην I συμβολίζεται με I(φ)=1 (ανάθεση αληθούς τιμής), I(φ)=0 (ανάθεση ψευδούς τιμής).

Στην ΠΛ η ερμηνεία I είναι μιά κατ'αρχήν αυθαίρετη ανάθεση τιμών αλήθειας στα άτομα της πρότασης. Ετσι στο παράδειγμα της έγχρωμης τηλεόρασης, μιά πιθανή ερμηνεία γιά τα

άτομα {p,q,r,s,t} θα ήταν η {0,0,1,1,0}. Δεδομένου ότι p="Θα παρακολουθήσω το έργο", q="έχω έγχρωμη τηλεόραση", r="είμαι απασχολημένος", s="θα γράψω το έργο", t="έχω video", μιά τέτοια ερμηνεία σημαίνει ότι δεν θα παρακολουθήσω το έργο, δεν έχω έγχρωμη τηλεόραση, είμαι απασχολημένος, θα γράψω το έργο, δεν έχω video. Η αποτίμηση της τιμής αλήθειας μιάς πρότασης στην ερμηνεία I, βρίσκεται από την σύνθεση των τιμών αλήθειας των ατόμων με βάση τους πίνακες αλήθειας των λογικών συνδέσμων.

Παράδειγμα 2.3 Ας θεωρήσουμε την πρόταση $(p\wedge q)\rightarrow(r\leftrightarrow\neg s)$. Η πρόταση διαθέτει 4 άτομα, οπότε υπάρχουν 16 ερμηνείες. Γιά κάθε ερμηνεία η πρόταση έχει τιμή αλήθεια ή ψέμα. Με βάση τον πίνακα αλήθειας μπορούμε να βρούμε ποιές ερμηνείες ικανοποιούν, δίνουν εκτίμηση 1, στην πρόταση.

I(p)	I(q)	I(r)	I(s)	$I(\neg s)$	$I(p\wedge q)$	$I(r\leftrightarrow\neg s)$	$I((p\wedge q)\rightarrow(r\leftrightarrow\neg s))$
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	1	0	1	1
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	1
0	1	1	1	0	0	0	1
1	0	0	0	1	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	1	0	1	1
1	0	1	1	0	0	0	1
1	1	0	0	1	1	0	0
1	1	0	1	0	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	0	1	0	0

Ορισμός 2.6-ταυτολογία, αντιλογία, ικανοποιησιμότητα
Μιά πρόταση που είναι αληθής κάτω από όλες τις ερμηνείες λέγεται ***ταυτολογία*** (valid, tautology). Μιά πρόταση που είναι ψευδής κάτω από όλες τις ερμηνείες λέγεται ***αντίφαση*** ή ***αντιλογία*** ή ***ασυνεπής*** (inconsistent, unsatisfiable, contradiction). Μιά πρόταση είναι ***ικανοποιήσιμη*** (satisfiable) άν υπάρχει τουλάχιστον μιά ερμηνεία κάτω από την οποία είναι αληθής.

Με βάση τον ορισμό 2.5 μπορούμε να πούμε 'Η πρόταση φ είναι αληθής κάτω από την ερμηνεία I'. Ενας άλλος τρόπος γιά να το πούμε αυτό είναι : 'Η πρόταση φ ικανοποιείται στην ερμηνεία I'. Η έννοια της ***ικανοποιησιμότητας*** (satisfaction) μιάς πρότασης συναντάται σε όλα τα εγχειρίδια της Μαθηματικής Λογικής.

Ορισμός 2.7 (Ικανοποίηση μιάς πρότασης και συμβολισμοί)
Εστω φ μιά πρόταση και I μιά ερμηνεία. Αν I(φ)=1 τότε λέμε ότι η I ικανοποιεί ή επαληθεύει την φ, και χρησιμοποιούμε τον συμβολισμό $I \models \varphi$, ή τον συμβολισμό $\models_I \varphi$. Αν I(φ)=0 τότε λέμε ότι η I δεν ικανοποιεί ή δεν επαληθεύει την φ, και χρησιμοποιούμε τον συμβολισμό $I \not\models \varphi$, ή τον συμβολισμό $\not\models_I \varphi$. Η ταυτολογία συμβολίζεται και $\models\varphi$, γιατί

ισχύσει ότι $\models_I \varphi$ γιά κάθε I. Η αντίφαση συμβολίζεται και $\not\models \varphi$ γιατί δεν υπάρχει I τέτοιο ώστε $\models_I \varphi$.

Ενας άλλος τρόπος γιά να βρούμε την τιμή αλήθειας μιάς πρότασης, χωρίς την χρήση των πινάκων αλήθειας, είναι η χρήση των προτάσεων ικανοποιησιμότητας. Ο συμβολισμός 'ανν' σημαίνει 'αν και μόνο αν'. Οι προτάσεις ικανοποιησιμότητας θα βρουν μεγαλύτερη πρακτική εφαρμογή στη Κατηγορηματική Λογική. Εκεί θα τις χρησιμοποιήσουμε εκτενέστερα. Είναι όμως καλό να εξοικιωθεί κανείς με το σχετικά δύστροπο συμβολισμό, γιατί αυτός χρησιμοποιείται στην θεμελίωση των λογικών συστημάτων.

Ορισμός 2.8 Προτάσεις ικανοποιησιμότητας (satisfaction formulae)

$\models_I \varphi$, όπου φ είναι άτομο, ανν $I(\varphi)=1$

$\models_I \neg\varphi$ ανν $\not\models_I \varphi$

$\models_I (\varphi\wedge\psi)$ ανν $\models_I \varphi$ και $\models_I \psi$

$\models_I (\varphi\vee\psi)$ ανν $\models_I \varphi$ ή $\models_I \psi$

$\models_I (\varphi\rightarrow\psi)$, ανν $\not\models_I \varphi$ ή $\models_I \psi$

$\models_I (\varphi\leftarrow\psi)$, ανν $\models_I (\psi\rightarrow\varphi)$

$\models_I (\varphi\leftrightarrow\psi)$, ανν $\models_I (\varphi\rightarrow\psi)$ και $\models_I (\psi\rightarrow\varphi)$

Παράδειγμα 2.4 Αν πάρουμε κάποια γραμμή από τον πίνακα αλήθειας του προηγουμένου παραδείγματος :

I(p)	I(q)	I(r)	I(s)	$I(\neg s)$	$I(p\wedge q)$	$I(r\leftrightarrow\neg s)$	$I((p\wedge q)\rightarrow(r\leftrightarrow\neg s))$
0	1	0	1	0	0	1	1

Σε αυτή τη γραμμή ορίζεται η ερμηνεία I={0,1,0,1} γιά το σύνολο ατόμων {p,q,r,s}. Χρησιμοποιώντας τις προτάσεις ικανοποιησιμότητας μπορούμε να βρούμε την τιμή αλήθειας της πρότασης $(p\wedge q)\rightarrow(r\leftrightarrow\neg s)$ στην ερμηνεία I ως εξής :

$\models_I (p\wedge q)\rightarrow(r\leftrightarrow\neg s)$

ανν $\not\models_I (p\wedge q)$ ή $\models_I (r\leftrightarrow\neg s)$

ανν οχι ($\models_I (p\wedge q)$) ή ($\models_I (r \rightarrow \neg s)$ και $\models_I (\neg s \rightarrow r)$)

ανν οχι ($\models_I (p\wedge q)$) ή (($\not\models_I r$ ή $\models_I \neg s$) και ($\not\models_I \neg s$ ή $\models_I r$))

ανν οχι ($\models_I p$ και $\models_I q$) ή (($\not\models_I r$ ή $\not\models_I s$) και ($\models_I s$ ή $\models_I r$))

ανν οχι (I(p)=1 και I(q)=1) ή ((I(r)=0 ή I(s)=0) και (I(s)=1 ή I(r)=1))

ανν I(p)=0 ή I(q)=0 ή ((I(r)=0 ή I(s)=0) και (I(s)=1 ή I(r)=1))

ανν αληθές ή ψευδές ή κ.λ.π.
που είναι αληθές

Αρα $\models_I (p\wedge q)\rightarrow(r\leftrightarrow\neg s)$ είναι αληθές, δηλαδή $I((p\wedge q)\rightarrow(r\leftrightarrow\neg s))=1$

Ενας ακόμη τρόπος γιά την εύρεση της τιμής αλήθειας μιάς πρότασης είναι η χρήση των συναρτήσεων αλήθειας, οι οποίες εύκολα προκύπτουν από τους πίνακες αλήθειας.

Ορισμός 2.9 Συναρτήσεις αλήθειας

$I(\neg\varphi) = 1 - I(\varphi)$

$I(\varphi\wedge\psi) = I(\varphi) * I(\psi)$

$I(\varphi\vee\psi) = I(\varphi) + I(\psi) - I(\varphi)*I(\psi)$

$I(\varphi\rightarrow\psi) = I(\neg\varphi\vee\psi) = 1 - I(\varphi) + I(\varphi)*I(\psi)$

$I(\varphi\leftrightarrow\psi) = I((\varphi\rightarrow\psi) \wedge (\psi\rightarrow\varphi)) = 1 - I(\varphi) - I(\psi) + 2*I(\varphi)*I(\psi)$

Παράδειγμα 2.5 Αν πάρουμε τη γραμμή από τον πίνακα αλήθειας του προηγουμένου παραδείγματος :

I(p)	I(q)	I(r)	I(s)	I(¬s)	I(p∧q)	I(r↔¬s)	I((p∧q)→(r↔¬s))
0	1	0	1	0	0	1	1

Με βάση την ερμηνεία I={0,1,0,1} γιά το σύνολο ατόμων {p,q,r,s} και χρησιμοποιώντας τις συναρτήσεις αλήθειας μπορούμε να βρούμε την τιμή αλήθειας της πρότασης (p∧q)→(r ↔¬s) στην ερμηνεία I ως εξής :
I((p∧q)→(r↔¬s)) = 1 - I(p∧q) + I(p∧q) * I(r↔¬s). Αλλά I(p∧q) = I(p)*I(q) = 0*1 = 0. Οπότε I((p∧q)→(r↔¬s)) = 1 - 0 + 0 * I(r↔¬s) = 1.

Πρέπει να τονιστεί ότι οι συναρτήσεις αλήθειας δεν ορίζονται μονοσήμαντα. Θα μπορούσαμε να ορίσουμε κάποιους άλλους τύπους συμβατούς με τους πίνακες αλήθειας. Ετσι, γιά παράδειγμα, μπορούμε να ορίσουμε I(φ∨ψ) = max(I(φ),I(ψ)).

Ορισμός 2.10 Ικανοποίηση συνόλου προτάσεων
Μιά ερμηνεία I ικανοποιεί ένα σύνολο προτάσεων Σ άν η I ικανοποιεί κάθε πρόταση του Σ. Δηλαδή : $\models_{\mathbf{I}} \Sigma$ **ανν** $(\forall \varphi \in \Sigma) \models_{\mathbf{I}} \varphi$. Στην περίπτωση αυτή η ερμηνεία I λέγεται ***μοντέλο*** του Σ.

Στη πραγματικότητα, πίσω από κάθε ερμηνεία κρύβεται μιά δυνατή κατάσταση του κόσμου. Ετσι όλες οι ερμηνείες μαζύ καλύπτουν όλες τις δυνατές καταστάσεις. Οσες όμως από τις ερμηνείες ικανοποιούν όλες τις προτάσεις ενός συνόλου, αυτές οι ερμηνείες είναι υποψήφιες γιά να προσδιορίζουν την κατάσταση του κόσμου.

Ισως στο σημείο αυτό πρέπει να συζητήσουμε λίγο εκτενέστερα την έννοια της ερμηνείας και σε τι χρειάζεται. Οπως είπαμε στην αρχή του κεφαλαίου αυτού η ερμηνεία χρειάζεται γιά να προσδιορίσει το περιεχόμενο των συμβόλων μιάς γλώσσας. Στη συγκεκριμένη περίπτωση έχουμε την γλώσσα της προτασιακής λογικής. Εχοντας μπροστά μας ένα σύνολο προτάσεων της προτασιακής λογικής που περιγράφει μιά συγκεκριμένη συγκυρία γεγονότων (π.χ. ένα φαινόμενο), και εφόσον δεν έχουμε κοντά μας τον συγγραφέα του συνόλου των προτάσεων, πρέπει να επινοήσουμε σε ποιά έννοια αντιστοιχεί κάθε άτομο του συνόλου των προτάσεων. Θεωρώντας σαν δεδομένο ότι το σύνολο των προτάσεων περιγράφει με ορθότητα το εν λόγω φαινόμενο, γράφουμε όλες τις δυνατές ερμηνείες και κρατάμε μόνο εκείνες που ικανοποιούν όλες τις προτάσεις του συνόλου. Αυτές τις ονομάζουμε μοντέλα, γιατί 'προβλέπουν' με επιτυχία την τιμή αλήθειας όλων των προτάσεων του συνόλου.

Η έννοια του μοντέλου υπάρχει και σε άλλες επιστήμες (π.χ. Ιατρική). Γενικά, σε τέτοιες επιστήμες, μοντέλο είναι κάτι που περιγράφει με επιτυχία μιά κατάσταση, μπορεί να προβλέπει μελλοντικές καταστάσεις, να δίνει εκτιμήσεις της πραγματικότητας, κ.λ.π. Η έννοια του μοντέλου στη Λογική είναι παρόμοια, μόνο που αναφέρεται σε ερμηνείες ενός συνόλου προτάσεων.

Για να κατανοήσουμε λίγο παραπάνω την έννοια του μοντέλου θα δούμε το ανάλογο ενός ανθρώπου που προσπαθεί να μάθει μιά καινούρια γλώσσα χωρίς τη χρήση λεξικού (γιατί βρίσκεται ξαφνικά σε μιά ξένη γλώσσα και πρέπει να επικοινωνήσει). Για να συνειδητοποιήσει τι σημαίνουν οι άγνωστες λέξεις ακούει γύρω του, κρατάει στη μνήμη του αυτές τις λέξεις και τις πιθανές τους ερμηνείες. Κάθε φορά που ακούει την άγνωστη λέξη προσπαθεί να βρεί ποιές από τις ερμηνείες που έχει στη μνήμη του μπορούν να επαληθεύουν την λέξη. Εκείνες που προφανώς δεν την επαληθεύουν τις ακυρώνει. Συνεχίζει να θυμάται όμως τις υπόλοιπες πιθανές ερμηνείες - τις ερμηνείες που είναι

μοντέλα. Οσο περισσότερες προτάσεις ακούει, τόσο περισσότερο αντιλαμβάνεται το νόημα - τη σημασία - των λέξεων, έως ότου να γνωρίσει το σωστό νόημα κάθε λέξης.
Στη ΠΛ όσο μεγαλύτερος είναι ο αριθμός των προτάσεων - γιά τον ίδιο αριθμό ατόμων - τόσο λιγότερα είναι τα υποψήφια μοντέλα, τόσο δηλαδή λιγότερες από όλες τις δυνατές ερμηνείες ικανοποιούν το σύνολο των προτάσεων. Τελικά, μετά από ικανή αύξηση του αριθμού των προτάσεων του αρχικού συνόλου, φτάνουμε στο να υπάρχει ένα μόνο μοντέλο - μιά μόνο ερμηνεία που ικανοποιεί όλες τις προτάσεις - και η οποία είναι η από τον συγγραφέα των προτάσεων υπονοούμενη ερμηνεία. Είναι δηλαδή η ερμηνεία που είχε στο νού του ο συγγραφέας όταν έγραφε τις προτάσεις.

Ομοια στην αποκρυπτογράφηση του κειμένου μιάς πλάκας με γραμμική γραφή, όσο περισσότερες είναι οι προτάσεις της, τόσο λιγώτερες οι δυνατές ερμηνείες που επεξηγούν - μεταφράζουν - με επιτυχία την πλάκα (δηλ. τα μοντέλα).

2.5 Λογικό συμπέρασμα : Η αλήθεια σε ένα σύνολο προτάσεων

Ορισμός 2.11 Λογικό Συμπέρασμα (|=) (logical consequence)
Εστω μιά πρόταση φ και ένα σύνολο προτάσεων Σ. Η φ λέγεται λογικό συμπέρασμα του Σ, και συμβολίζεται Σ|=φ, αν κάθε μοντέλο του Σ είναι και μοντέλο της φ. Δηλαδή :

$$\Sigma \models \varphi \text{ ανν } (\forall i)\ (\models_i \Sigma \Rightarrow \models_i \varphi).$$

Η ουσία του λογικού συμπεράσματος είναι ότι μιά πρόταση είναι αληθής με βάση ένα σύνολο προτάσεων, αν αυτή ικανοποιείται σε όλα τα μοντέλα του συνόλου. Αυτός ο έλεγχος είναι αναγκαίος, γιατί συνήθως δεν γνωρίζουμε ποιό από τα μοντέλα ήταν το υπονοούμενο όταν γράφτηκαν οι προτάσεις. Ετσι η νέα πρόταση πρέπει να επαληθεύεται σε όλα τα μοντέλα. Εαν εξάλλου γνωρίζαμε την υποκριπτόμενη ερμηνεία του συνόλου των προτάσεων, θ μπορούσαμε απ'ευθείας να ελέγξουμε την τιμή αλήθειας της πρότασης με βάση τους πίνακες αλήθειας, ή τις προτάσεις ικανοποιησιμότητας, ή τις συναρτήσεις αλήθειας, γιά μία μόνο ερμηνεία : εκείνη που υπονοούσε ο συγγραφέας του συνόλου των προτάσεων.

Ο έλεγχος ως προς το αν μιά πρόταση είναι λογικό συμπέρασμα ενός συνόλου προτάσεων γίνεται με πολλούς τρόπους, αλλά βασίζεται κυρίως στους πίνακες αλήθειας. Θα περιγράψουμε 6 μεθόδους με τις οποίες επιτυγχάνεται αυτός ο έλεγχος. Οι μέθοδοι αυτές είναι κατά βάση ισοδύναμες και στηρίζονται όλες στο γεγονός ότι με δεδομένο ένα σύνολο προτάσεων Σ, δεν γνωρίζουμε εκ των προτέρων την υπονοούμενη από τον συγγραφέα των προτάσεων ερμηνεία.

Πρόταση 2.1 (Λογικό συμπέρασμα - Μέθοδος 1η)
Δεδομένου ενός συνόλου $\Sigma=\{\varphi_1,\varphi_2,...,\varphi_v\}$ και μιάς πρότασης φ, η σχέση Σ|=φ ισχύει ανν γιά κάθε ερμηνεία κάτω από την οποία είναι αληθής η $\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v$ και η φ είναι αληθής.

Κατασκευάζουμε τον πίνακα αλήθειας των $\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v$ και φ και ελέγχουμε τις γραμμές γιά τις οποίες αληθεύει η $\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v$ (τα μοντέλα του Σ). Γιά τις γραμμές αυτές πρέπει να αληθεύει και η φ. (Δεν μας ενδιαφέρει αν η φ αληθεύει και σε άλλες γραμμές).

Παράδειγμα 2.6 Θεωρείστε το εξής σύνολο προτάσεων : $\Sigma=\{q\rightarrow\neg r,\ \neg p\rightarrow q,\ r\}$ και την πρόταση p. Θα προσπαθήσουμε να αποδείξουμε ότι η πρόταση p είναι λογικό συμπέρασμα του Σ.

I(p)	I(q)	I(r)	I(q→¬r)	I(¬p→q)	I((q→¬r) ∧ (¬p→q) ∧ r)
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	1	0

Πράγματι βλέπουμε ότι η μοναδική εμηνεία στην οποία είναι αληθές το Σ, δηλ. κάθε προταση του Σ, είναι η ερμηνεία I={1,0,1}στην οποία η p είναι αληθής. Αρα Σ|=p.

Πρόταση 2.2 (Λογικό συμπέρασμα - Μέθοδος 2η)
Δεδομένου ενός συνόλου $\Sigma=\{\varphi_1,\varphi_2,...,\varphi_v\}$ και μιάς πρότασης φ, η σχέση Σ|=φ ισχύει ανν η πρόταση $(\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v\rightarrow\varphi)$ είναι ταυτολογία.

Κατασκευάζουμε τον πίνακα αλήθειας της $(\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v\rightarrow\varphi)$ και ελέγχουμε αν εμφανίζεται 1 σε όλες τις γραμμές του πίνακα.

Παράδειγμα 2.7 Συνεχίζοντας το προηγούμενο παράδειγμα, δημιουργούμε άλλη μια στήλη με την συνεπαγωγή ((q→¬r) ∧ (¬p→q) ∧ r)→p. Παρατηρούμε ότι όλες οι γραμμές της τελευταίας στήλης έχουν την τιμή 1. Ενας γρήγορος τρόπος να το βρούμε αυτό είναι να ελέγξουμε τις περιπτώσεις που η συνεπαγωγή φ→ψ έχει τιμή αλήθειας 0 : αυτό συμβαίνει μόνο όταν η τιμή αλήθειας του φ είναι 0 ενώ ταυτόχρονα η τιμή αλήθειας του ψ είναι 1.

I(p)	I(q)	I(r)	I(q→¬r)	I(¬p→q)	I((q→¬r) ∧ (¬p→q) ∧ r)	I(Σ →p)
0	0	0	1	0	0	1
0	0	1	1	0	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	0	1	1	1	1	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1

Πρόταση 2.3 Λογικό συμπέρασμα - Μέθοδος 3η
Δεδομένου ενός συνόλου $\Sigma=\{\varphi_1,\varphi_2,...,\varphi_v\}$ και μιάς πρότασης φ, η σχέση Σ|=φ ισχύει ανν η πρόταση $(\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v\wedge\neg\varphi)$ είναι αντιλογία.

Κατασκευάζουμε τον πίνακα αλήθειας της $(\varphi_1\wedge\varphi_2\wedge...\wedge\varphi_v\wedge\neg\varphi)$ και ελέγχουμε αν εμφανίζεται 0 σε όλες τις γραμμές του πίνακα.

Παράδειγμα 2.8 Με τον ίδιο τρόπο όπως στα δύο τελευταία παραδείγματα, μόνο που στην τελευταία στήλη βρίσκουμε τις τιμές αλήθειας της σύζευξης ((q→¬r) ∧ (¬p→q) ∧ r) ∧ ¬p. Παρατηρούμε ότι όλες οι γραμμές της τελευταίας στήλης έχουν την τιμή αλήθειας 0.

I(p)	I(q)	I(r)	I(q→¬r)	I(¬p→q)	I((q→¬r) ∧ (¬p→q) ∧ r)	I(Σ ∧ ¬p)
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	0	1	1	1	1	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Ορισμός 2.12 Λογική Ισοδυναμία (|=|).
Οι προτάσεις φ,ψ λέγονται λογικά ισοδύναμες, και συμβολίζεται ως φ |=| ψ, αν η κάθε μιά είναι λογικό συμπέρασμα της άλλης (φ|=ψ και ψ|=φ).

Ο έλεγχος της ισοδυναμίας δύο προτάσεων μπορεί να γίνει με τους πίνακες αλήθειας. Εξετάζουμε αν οι δύο προτάσεις παρουσιάζουν τις ίδιες τιμές αλήθειας σε όλες τις γραμμές. Μπορεί όμως να γίνει και με διαδοχικούς μετασχηματισμούς. Χρησιμοποιώντας ένα σύνολο βασικών μετασχηματισμών μετατρέπουμε την κάθε πρόταση σε ισοδύναμή της, έως ότου να βρούμε τρόπο να να παράγουμε την τελική πρόταση από την αρχική. Οι μετασχηματισμοί αυτοί αποτελούν σχέσεις λογικής ισοδυναμίας απλών προτάσεων :

Πίνακας Βασικών Λογικών Ισοδυναμιών προτάσεων της ΠΛ

(2.1) (φ↔χ) |=| (φ→χ)∧(χ→φ)
(2.2) (φ→χ) |=| ¬φ ∨ χ
(2.3) (φ∨χ) |=| (χ∨φ) (φ∧χ) |=| (χ∧φ)
(2.4) (φ∨χ)∨ψ |=| φ∨(χ∨ψ) (φ∧χ)∧ψ |=| φ∧(χ∧ψ)
(2.5) φ∨(χ∧ψ) |=| (φ∨χ)∧(φ∨ψ) φ∧(χ∨ψ) |=| (φ∨χ)∧(φ∨ψ)
(2.6) ¬(¬φ) |=| φ
(2.7) ¬(φ∧χ) |=| ¬φ∨¬χ ¬(φ∨χ) |=| ¬φ∧¬χ
(2.8) φ∧1 |=| φ φ∧0 |=| 0 φ∧φ |=| φ φ∧¬φ |=| 0
(2.9) φ∨1 |=| 1 φ∨0 |=| φ φ∨φ |=| φ φ∨¬φ |=| 1

Λογικό συμπέρασμα - Μέθοδος 4η
Χρησιμοποιώντας του κανόνες μετασχηματισμού ισοδύναμων προτάσεων, μετασχηματίζουμε την πρόταση $(\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_v \wedge \neg\varphi)$ σε μιά ισοδύναμή της που είναι αντίφαση.

Παράδειγμα 2.9 Κινούμενοι πάντα στο ίδιο παράδειγμα που χρησιμοποιήσαμε στις προηγούμενες μεθόδους, προσπαθούμε να μετατρέψουμε την πρόταση Σ ∧ ¬p σε μιά ισοδύναμή της που είναι αντίφαση.

(q→¬r) ∧ (¬p→q) ∧ r ∧ ¬p
|=| (¬q ∨ ¬r) ∧ (¬¬p ∨ q) ∧ r ∧ ¬p
|=| (¬q ∨ ¬r) ∧ (p ∨ q) ∧ r ∧ ¬p
|=| ((¬q ∨ ¬r) ∧ r) ∧ ((p ∨ q) ∧ ¬p)
|=| ((¬q ∧ r) ∨ (¬r ∧ r)) ∧ ((p ∧ ¬p) ∨ (q ∧ ¬p))
|=| ((¬q ∧ r) ∨ 0) ∧ (0 ∨ (q ∧ ¬p))

|=| $(\neg q \wedge r) \wedge (q \wedge \neg p)$
|=| $(\neg q \wedge r \wedge q \wedge \neg p)$
|=| $(\neg q \wedge q \wedge r \wedge \neg p)$
|=| $(0 \wedge r \wedge \neg p)$
|=| 0

Λογικό συμπέρασμα - Μέθοδος 5η
Μπορούμε να χρησιμοποιήσουμε τις προτάσεις ικανοποιησιμότητας γιά να βρούμε αν γιά τα μοντέλα του Σ ικανοποιείται η φ. Δεδομένου ότι $\Sigma=\{\varphi_1,\varphi_2,...,\varphi_v\}$, πρέπει να αποδείξει κανείς ότι γιά όλες τις ερμηνείες I οι οποίες ικανοποιούν τις προτάσεις του Σ, αυτές οι ερμηνείες επίσης επαληθεύουν την φ.

Παράδειγμα 2.10 Εστω $\Sigma=\{q\rightarrow\neg r , \neg p\rightarrow q , r\}$. Θεωρούμε εκείνες τις ερμηνείες I γιά τις οποίες ισχύει : $\models_I (q\rightarrow\neg r)$ και $\models_I (\neg p\rightarrow q)$ και $\models_I r$. Γιά τις ερμηνείες αυτές θα ισχύουν τα παρακάτω :
$\models_I (q\rightarrow\neg r)$ ανν $\models_I (\neg q\vee\neg r)$ ανν $I(\neg q) = 1$ η $I(\neg r) = 1$. Ομως από την $\models_I r$ συμπεραίνουμε ότι $I(r) = 1$. Οπότε $I(\neg r) = 0$ και έτσι πρέπει αναγκαστικά να ισχύει η σχέση $I(\neg q) = 1$ στη πρώτη πρυπόθεση ικανοποιησιμότητας. Αρα πρέπει $I(q) = 0$. Στη συνέχεια από την σχέση $\models_I (\neg p\rightarrow q)$ συμπεραίνουμε ισοδύναμα ότι $\models_I (p\vee q)$ που ισχύει ανν $I(p) = 1$ η $I(q) = 1$. Καθώς όμως $I(q) = 0$, όπως προηγούμενα αποδείξαμε, πρέπει αναγκαστικά να ισχύει ότι $I(p) = 1$, δηλαδή $\models_I p$. Αρα γιά όλα τα I για τα οποία ικανοποιούνται οι προτάσεις του Σ, ικανοποιείται και η p.

Λογικό συμπέρασμα - Μέθοδος 6η
Μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις αλήθειας γιά να βρούμε αν γιά τα μοντέλα του Σ ικανοποιείται η φ. Δεδομένου ότι $\Sigma=\{\varphi_1,\varphi_2,...,\varphi_v\}$, πρέπει να αποδείξει κανείς ότι γιά όλα τις ερμηνείες I οι οποίες ικανοποιούν τις προτάσεις του Σ, αυτές οι ερμηνείες επίσης επαληθεύουν την φ.

Παράδειγμα 2.11 (το γνωστό μας πιά παράδειγμα) Εστω $\Sigma=\{q\rightarrow\neg r , \neg p\rightarrow q , r\}$. Θεωρούμε εκείνες τις ερμηνείες I γιά τις οποίες ισχύει $I(q\rightarrow\neg r)=1$ και $I(\neg p\rightarrow q)=1$ και $I(r)=1$. Γιά τις ερμηνείες αυτές θα ισχύουν τα παρακάτω :
$I(q\rightarrow\neg r) = 1$
ανν $I(\neg q\vee\neg r) = 1$
ανν $I(\neg q) + I(\neg r) - I(\neg q)*I(\neg r)= 1$
Ομως $I(r) = 1$. Οπότε $I(\neg r) = 0$, οπότε
$I(\neg q) + 0 - I(\neg q)*0= 1$
ανν $I(\neg q) = 1$
ανν $I(q) = 0$
Από $I(\neg p\rightarrow q)=1$, έχουμε ότι
$1- I(\neg p)+I(\neg p)*I(q)=1$
ανν $1- I(\neg p)+I(\neg p)*0=1$
ανν $I(\neg p)=0$
οπότε $I(p)=1$, άρα γιά όλα τα I για τα οποία επαληθεύονται οι προτάσεις του Σ, επαληθεύεται και η p.

Οσον αφορά στις τέσσερις πρώτες μεθόδους, η πολυπλοκότητα αυξάνει όσο αυξάνει ο αριθμός των ατόμων του αρχικού συνόλου. Η πέμπτη μέθοδος που χρησιμοποιεί τις

προτάσεις ικανοποιησιμότητας μπορεί να μην έχει πάντοτε προφανή γενική λύση. ίσως, δηλαδή είναι απαραίτητο να καταφύγει κανείς στις επιμέρους ερμηνείες γιά να αποδείξει ότι η προς απόδειξη πρόταση επίσης ικανοποιείται από αυτές. Οσον αφορά στην έκτη μέθοδο, αυτή συνήθως επαφύεται στη λύση ενός μη γραμμικού συστήματος.

Ενα τελευταίο, αλλά σημαντικό θέμα, είναι αυτό της μετατροπής μιάς πρότασης σε κανονική συζευκτική μορφή (conjunctive normal form). Στην μορφή αυτή μετατρέπουμε κάθε πρόταση σε μιά ισοδύναμη πρόταση που αποτελείται από συζεύξεις διαζεύξεων. Στη συνέχεια μπορούμε να θεωρήσουμε κάθε διάζευξη σαν ιδιαίτερη πρόταση του συνόλου, και παίρνουμε έτσι ένα ισοδύναμο σύνολο διαζεύξεων.

Ας σημειωθεί ότι παρόμοια με την κανονική συζευκτική μορφή είναι και η κανονική διαζευκτική μορφή (disjunctive normal form), η οποία όμως αυτή τη στιγμή δεν θα μας απασχολήσει.

Παράδειγμα 2.12
Οι παρακατω προτάσεις είναι σε κανονική συζευκτική μορφή :

$(p \vee \neg r \vee \neg s) \wedge (\neg p \vee r)$	(μιά σύζευξη δύο διαζεύξεων)
$p \vee r \vee \neg s$	(μιά διάζευξη)
$p \wedge r$	(μιά σύζευξη δύο ατόμων)
$\neg s$	(ένα άτομο)

Η παρακατω πρόταση δεν βρίσκεται σε κανονική συζευκτική μορφή :

$((p \vee \neg r) \wedge \neg s) \wedge (\neg p \vee r)$

Πρόταση 2.4 Μετατροπή σε Κανονική Συζευκτική Μορφή.
Κάθε πρόταση μπορεί να μετασχηματιστεί σε ισοδύναμη της που έχει την μορφή σύζευξης διαζεύξεων.

Αλγόριθμος 2.1. Μετασχηματισμός πρότασης σε κανονική συζευκτική μορφή.

Βήμα 1. Χρησιμοποιώντας τον κανόνα $(p \leftrightarrow q) \models (p \rightarrow q) \wedge (q \rightarrow p)$ απαλοίφουμε τις ισοδυναμίες.

Βήμα 2. Χρησιμοποιώντας τον κανόνα $(p \rightarrow q) \models \neg p \vee q$ απαλοίφουμε τις συνεπαγωγές.

Βήμα 3. Εφαρμόζουμε τους κανόνες $\neg(\neg p) \models p$, $\neg(p \wedge q) \models \neg p \vee \neg q$, $\neg(p \vee q) \models \neg p \wedge \neg q$ έως ότου να μην εφαρμόζονται άλλο.

Βήμα 4. Εφαρμόζουμε τους κανόνες $p \wedge (q \vee r) \models (p \vee q) \wedge (p \vee r)$, $p \vee (q \wedge r) \models (p \vee q) \wedge (p \vee r)$ διαδοχικά έως ότου να μετασχηματίσουμε την πρόταση σε κανονική συζευκτική μορφή.

Βήμα 5. Μπορούμε να χρησιμοποιήσουμε και τους κανόνες $(p \wedge 1 \models p)$, $(p \wedge 0 \models 0)$, $(p \wedge p \models p)$, $(p \wedge \neg p \models 0)$, $(p \vee 1 \models 1)$, $(p \vee 0 \models p)$, $(p \vee p \models p)$, $(p \vee \neg p \models 1)$, γιά να απλοποιήσουμε τις προτάσεις.

Παράδειγμα 2.13 Θεωρείστε την πρόταση $(p \wedge q) \rightarrow (r \leftrightarrow \neg s)$. Εφαρμόζουμε τον παραπάνω αλγόριθμο για να τη μετατρέψουμε σε κανονική συζευκτική μορφή.

Βήμα 1 : Απαλοιφή ισοδυναμιών

$(p \wedge q) \rightarrow (r \leftrightarrow \neg s)$

$\models$ $(p \wedge q) \rightarrow ((r \rightarrow \neg s) \wedge (\neg s \rightarrow r))$

Βήμα 2 : Απαλοιφή συνεπαγωγών

$(p \wedge q) \rightarrow ((r \rightarrow \neg s) \wedge (\neg s \rightarrow r))$

$\models$ $\neg(p \wedge q) \vee ((\neg r \vee \neg s) \wedge (\neg\neg s \vee r))$

Βήμα 3 : Μετακίνηση του συμβόλου της άρνησης προς τα μέσα

$\neg(p \wedge q) \vee ((\neg r \vee \neg s) \wedge (\neg\neg s \vee r))$

$|=|$ $(\neg p \vee \neg q) \vee ((\neg r \vee \neg s) \wedge (s \vee r))$

Βήμα 4 : Χρησιμοποίηση της επιμεριστικής ιδιότητας

$(\neg p \vee \neg q) \vee ((\neg r \vee \neg s) \wedge (s \vee r))$

$|=|$ $((\neg p \vee \neg q) \vee (\neg r \vee \neg s)) \wedge ((\neg p \vee \neg q) \vee (s \vee r))$

$|=|$ $(\neg p \vee \neg q \vee \neg r \vee \neg s) \wedge (\neg p \vee \neg q \vee s \vee r)$

Η σημασιολογία όπως ορίστηκε στις προηγούμενες παραγράφους, αναφέρεται στην βιβλιογραφία και σαν ***μοντελοθεωρητική σημασιολογία*** (model theoretic semantics). Το βασικό ερώτημα το οποίο απαντήθηκε με την μοντελοθεωρητική σημασιολογία είναι το άν μιά πρόταση είναι λογικό συμπέρασμα ενός συνόλου προτάσεων. Παρόλο που η μέθοδος των πινάκων αλήθειας είναι αποτελεσματική γιά μικρό αριθμό ατομικών προτάσεων, γίνεται μη αποδοτική γιά μεγάλο αριθμό ατομικό προτάσεων (2^{ν} δυνατές ερμηνείες). Οι άλλες μέθοδοι έχουν και αυτές τα προβλήματά τους.

Ενα τέτοιο αρχικό σύνολο προτάσεων Σ πρέπει να διαθέτει κάποια χαρακτηριστικά αλληλοσυμπλήρωσης και συμβατότητας. Εαν οι προτάσεις του Σ δεν αλληλοσυμπληρώνονται τότε το σύνολο είναι ελλειπές καθώς κάποιο τμήμα του φαινομένου, το οποίο προσπαθούν να ορίσουν οι προτάσεις, δεν περιγράφεται. Εάν κάποιες από τις προτάσεις του Σ είναι μεταξύ τους ασύμβατες, δηλαδή η σύζευξή τους οδηγεί σε αντιλογία, τότε όλο το σύνολο είναι ασύμβατο. Αυτό πρέπει να το προσέξουμε ιδιαίτερα γιατί σε ένα ασύμβατο σύνολο αληθεύουν όλες οι προτάσεις, καθώς η σύζευξη $(\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_{\nu} \wedge \neg\varphi)$ είναι αντιλογία λόγω του ότι το πρώτο μέρος της, δηλαδή το $(\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_{\nu})$, είναι αντιλογία ανεξάρτητα της πρότασης φ. Εάν οι προτάσεις του Σ αλληλοκαλύπτονται, δεν δημιουργείται βέβαια κάποιο προφανές πρόβλημα, όμως η θεωρία την οποία προσπαθούμε να αναπτύξουμε δεν περιγράφεται με σαφήνεια.

Ενα σύνολο προτάσεων Σ χρησιμοποιείται γιά να περιγράψει μιά νέα θεωρία, η οποία προβλέπει με επιτυχία τα αποτελέσματα ενός φαινομένου ή μιάς κατάστασης. Ενα τέτοιο σύνολο θα μπορούσε να περιγράφει τα κριτήρια που χρησιμοποιεί ένας γιατρός στη διάγνωση, ή τους κανόνες στους οποίους στηρίζεται ένα μαθηματικό σύστημα, ή τους λειτουργικούς νόμους στους οποίους στηρίζεται μιά μηχανή. Ενα τέτοιο σύστημα ονομάζεται πολλές φορές και **σύστημα θεμελιωδών αξιωμάτων** (non logical axioms, proper axioms) στις οποίες βασίζεται μιά καινούρια θεωρία. Στην πραγματικότητα κάθε πρόταση φ που ικανοποιείται σε αυτό το σύστημα θεμελιωδών αξιωμάτων ($\Sigma|=\varphi$) ονομάζεται θεώρημα. Ο όρος θεμελιώδη αξιώματα έρχεται σε αντιδιαστολή με τον όρο **λογικά αξιώματα** (logical axioms) που χρησιμοποιείται στην αξιωματοποίηση των λογικών συστημάτων, όπως θα δούμε παρακάτω.

Το πιό σημαντικό πρόβλημα όμως είναι ότι πρέπει, στη γενική περίπτωση, να βρεί κανείς με κάποιο τρόπο τα μοντέλα ενός συνόλου προτάσεων. Το αρχικό σύνολο των προτάσεων έχει συνταχθεί από ένα ειδικό σε ένα θέμα σε συνεργασία με ένα Μηχανικό γνώσης ή ένα Μαθηματικό. Το σύνολο αυτό των προτάσεων, με την βοήθεια του Μηχανικού γνώσης, πρέπει όμως να κωδικοποιηθεί κατάλληλα, με τη μορφή ενός λογικού προγράμματος, να εισαχθεί σε ένα υπολογιστή, και στη συνέχεια να χρησιμοποιηθεί γιά την εξαγωγή συμπερασμάτων, απόδειξη θεωρημάτων, εξαγωγή γνώσης, κ.λ.π.

Ενας τρίτος άνθρωπος, έχοντας κάποια αντίληψη του προβλήματος στο οποίο αναφέρεται το αρχικό σύνολο και βλέποντας το ίδιο το σύνολο μπορεί να κατανοήσει σε αρκετό βαθμό την υπονοούμενη από το συγγραφέα του συνόλου των προτάσεων ερμηνεία. Οταν γιά παράδειγμα δει κάποιος τα θεμελιώδη αξιώματα της Ευκλείδιας Γεωμετρίας κατανοεί τι σημαίνει ευθεία, σημείο, κ.λ.π. Ομοια όταν κάποιος δει ένα σύνολο προτάσεων που περιγράφουν τη λειτουργία μιάς μηχανής μπορεί κατά κάποιο τρόπο να κατανοήσει τι εννοούσε ο συγγραφέας όταν έγραφε το ένα ή το άλλο αξίωμα.

Δεδομένου όμως ότι στη Τεχνητή Νοημοσύνη μας ενδιαφέρει να πραγματοποιήσουμε αυτοματοποιημένη διαδικασία απόδειξης σε ένα υπολογιστή, πως είναι δυνατόν να περιμένουμε από μιά μηχανή να 'κατανοήσει τη σημασιολογία κάποιων προτάσεων; Πρέπει λοιπόν να βρούμε κάποιο τρόπο να οδηγούμαστε στην αλήθεια των προτάσεων χωρίς να εξαρτώμαστε από την ερμηνεία τους.

Βέβαια, πριν τη Τεχνητή Νοημοσύνη, τα Μαθηματικά έχουν ήδη λύσει αυτό το πρόβλημα με την έννοια της απόδειξης. Γιά αυτή την έννοια θα μιλήσουμε εκτενώς στις παρακάτω παραγράφους.

2.6 Αξιωματοποίηση της Προτασιακής Λογικής

Μιά άλλη προσέγγιση στο θέμα της εύρεσης της αλήθειας μιάς πρότασης από ένα αρχικό σύνολο θεμελιωδών αξιωμάτων είναι η ***αποδεικτικοθεωρητική σημασιολογία*** (proof theoretic semantics). Η αποδεικτικοθεωρητική συμπερασματολογία αφορά στην *παραγωγή νέων προτάσεων* από ένα σύνολο αρχικών προτάσεων με την χρήση ***κανόνων παραγωγής*** (κανόνες συμπερασμού, inference rules) και **λογικών αξιωμάτων** (logical axioms). Η παραγωγή αυτή είναι και αυτό που αποκαλούμε **απόδειξη**.

Η προσέγγιση αυτή προυποθέτει την ύπαρξη (α) μιάς τυπικής γλώσσας, (β) κανόνων σχηματισμού των προτάσεων της γλώσσας, (γ) λογικών αξιωμάτων που είναι προτάσεις που αληθεύουν σε όλα τα μοντέλα (ταυτολογίες), (δ) κανόνων παραγωγής που να οδηγούν μηχανικά από ορισμένες προτάσεις σε άλλες, με κριτήριο την μορφή τους και μόνο.

Εαν διαθέτουμε ένα τέτοιο σύστημα παραγωγής, είναι δυνατόν να καταλήγουμε με ένα μηχανικό τρόπο σε προτάσεις που αληθεύουν στο αρχικό σύνολο Σ χωρίς να ελέγχουμε τις ερμηνείες.

2.6.1 Αξιωματοποίηση με τρία αξιωματικά σχήματα και το Modus Ponens

Ενα αξιωματικό σχήμα είναι ένα προτασιακό μορφότυπο που είναι ταυτολογία και περιέχει σύμβολα προτάσεων τα οποία χρησιμοποιούνται σαν μεταβλητές. Εαν δηλαδή αντικαταστήσουμε οποιεσδήποτε προτάσεις στην θέση των μεταβλητών το αξιωματικό σχήμα είναι μιά ταυτολογία της ΠΛ.

Στην πραγματικότητα, δεν αναφέρουμε κάτι καινούριο. Εχουμε ξαναδεί τέτοια αξιωματικά σχήματα και προηγούμενα, απλά δεν τους προσδώσαμε αυτό τον 'ηχηρό' όρο. Οταν μιλούσαμε γιά προτάσεις ικανοποιησιμότητας, συναρτήσεις αλήθειας, καθώς και γιά λογικές ισοδυναμίες υπονοούσαμε την χρήση προτασιακών μεταβλητών. Στη λογική ισοδυναμία $(\varphi\leftrightarrow\chi) \models (\varphi\rightarrow\chi)\wedge(\chi\rightarrow\varphi)$ τα φ,χ,ψ είναι προτασιακές μεταβλητές, καθώς πρέπει να τις αντικαταστήσουμε με ατομικές ή σύνθετες προτάσεις γιά να έχουμε μιά λογική ισοδυναμία. Με την ίδια έννοια έχουμε ένα αγιωματικό σχήμα. Αν αντικαταστήσουμε τις προτασιακές μεταβλητές με ατομικές ή σύνθετες προτάσεις, πέρνουμε μιά ταυτολογία.

Μιά κλασσική προσέγγιση του θέματος αφορά στη χρησιμοποίηση τριών αξιωματικών σχημάτων:

(Π1) $\varphi\rightarrow(\psi\rightarrow\varphi)$

(Π2) $(\varphi\rightarrow(\psi\rightarrow\sigma))\rightarrow((\varphi\rightarrow\psi)\rightarrow(\varphi\rightarrow\sigma))$

(Π3) $(\neg\varphi\rightarrow\neg\psi)\rightarrow((\neg\varphi\rightarrow\psi)\rightarrow\varphi))$

και ενός κανόνα παραγωγής, του Modus Ponens (M.P.) που λέει ότι από τις φ και $\varphi\rightarrow\psi$ παράγεται η ψ.

Ορισμός 2.13. Ορισμός της απόδειξης
Απόδειξη της φ από το Σ στο σύστημα παραγωγής p, (συμβολίζουμε $\Sigma \vdash_p \varphi$), λέγεται μιά πεπερασμένη ακολουθία προτάσεων $\psi_1, \psi_2, ..., \psi_v$, τέτοια ώστε $\psi_v=\varphi$ και κάθε ψ_i είναι ή πρόταση του Σ, ή αξιωματικό σχήμα που έχει υποστεί κάποιες αντικαταστάσεις, ή παράγεται με το Μ.Ρ. από δύο προηγούμενες προτάσεις της ακολουθίας.

Παράδειγμα 2.14 Να αποδειχθεί ότι $\Sigma \vdash_p \rightarrow r$, όπου $\Sigma=\{p\rightarrow q, q\rightarrow r\}$
Απόδειξη :
1. $p\rightarrow q$, που είναι πρόταση του Σ,
2. $q\rightarrow r$, που είναι πρόταση του Σ,
3. $(q\rightarrow r)\rightarrow(p\rightarrow(q\rightarrow r))$, αξίωμα (Π1) με αντικαταστάσεις $\varphi=(q\rightarrow r)$ και $\psi=p$,
4. $p\rightarrow(q\rightarrow r)$, με Μ.Ρ. πάνω στις 2,3,
5. $(p\rightarrow(q\rightarrow r))\rightarrow((p\rightarrow q)\rightarrow(p\rightarrow r))$, αξίωμα (Π1) με αντικαταστάσεις $\varphi=p$, $\psi=q$, $\sigma=r$,
6. $(p\rightarrow q)\rightarrow(p\rightarrow r)$, με Μ.Ρ. πάνω στις 4,5,
7. $(p\rightarrow r)$, με Μ.Ρ. πάνω στις 1,6.

2.6.2. Θεωρήματα ορθότητας - πληρότητας

Υπάρχουν δύο πολύ σημαντικά θεωρήματα, την απόδειξη των οποίων δεν παραθέτουμε εδώ, τα οποία μπορούν να συσχετίσουν ένα σύστημα απόδειξης με την ορισθείσα σημασιολογία. Είναι το θεώρημα Ορθότητας και το θεώρημα Πληρότητας. Γιά κάθε καινούριο σύστημα Λογικής πρέπει να δίδεται ένα σύστημα σημασιολογίας και ένα σύστημα απόδειξης. Πρέπει όμως να χαρακτηρίζεται το σύστημα απόδειξης ως προς την ορθότητα και την πληρότητά του σε σχέση με την σημαιολογία της Λογικής που προτείνεται.

Θεώρημα Ορθότητας (Soundness Theorem) : $\Sigma \vdash_p \varphi \Rightarrow \Sigma \models \varphi$
Το θεώρημα αυτό όταν αποδεικνύεται γιά ένα σύστημα παραγωγής φανερώνει ότι κάθε πρόταση η οποία αποδεικνύεται με βάση το σύστημα παραγωγής είναι πράγματι αληθής στην σημασιολογία. Λέμε τότε ότι το σύστημα απόδειξης (ή σύστημα παραγωγής) είναι ορθό.

Θεώρημα Πληρότητας (Completeness Theorem) : $\Sigma \models \varphi \Rightarrow \Sigma \vdash_p \varphi$
Το θεώρημα αυτό όταν αποδεικνύεται γιά ένα σύστημα παραγωγής φανερώνει ότι κάθε πρόταση η οποία αληθεύει στην σημασιολογία αποδεικνύεται με βάση το σύστημα παραγωγής. Με άλλα λόγια, δεν υπάρχει πρόταση που να αληθεύει στην σημασιολογία και να μη μπορεί να αποδειχθεί από το σύστημα παραγωγής. Λέμε σε αυτή την περίπτωση ότι το σύστημα απόδειξης (ή σύστημα παραγωγής) είναι πλήρες.

2.6.3 Αλλες Αξιωματοποιήσεις της Π.Λ.

Εχουν προταθεί διάφορα συστήματα γιά την αξιωματοποίηση της Π.Λ. Τα παρακάτω είναι μερικά χαρακτηριστικά που χρησιμοποιούν σαν κανόνα παραγωγής το Modus Ponens :

α) Σύστημα Hilbert

Αξιώματα : (1) $\varphi\rightarrow(\psi\rightarrow\varphi)$
(2) $(\varphi\rightarrow(\psi\rightarrow\sigma))\rightarrow((\varphi\rightarrow\psi)\rightarrow(\varphi\rightarrow\sigma))$
(3) $\neg(\neg\varphi)\rightarrow\varphi$

β) Σύστημα Hilbert - Ackermann

Αξιώματα : (1) $\varphi\vee\varphi\rightarrow\varphi$
(2) $\varphi\rightarrow\varphi\vee\varphi$
(3) $\varphi\vee\psi\rightarrow\psi\vee\varphi$
(4) $(\varphi\rightarrow\sigma)\rightarrow(\psi\vee\varphi\rightarrow\psi\vee\sigma)$

γ) Σύστημα Rosser

Αξιώματα : (1) $\varphi\wedge\psi\rightarrow\varphi$
(2) $\varphi\rightarrow\varphi\wedge\varphi$
(3) $(\varphi\rightarrow\psi)\rightarrow(\neg(\psi\vee\sigma)\rightarrow\neg(\sigma\vee\varphi))$

2.6.4. Η αρχή της απόφασης (Resolution Principle)

Το 1965 προτάθηκε από τον Robinson ένας κανόνας συμπερασμού (inference rule) ο οποίος ονομάστηκε αρχή της απόφασης (resolution principle) και ο οποίος μπορεί να χρησιμοποιηθεί γιά την παραγωγή της κενής πρότασης σε μιά αποδεικτική διαδικασία που μοιάζει πολύ με την εις άτοπο απαγωγή. Η αρχή της απόφασης μπορεί να διατυπωθεί ως εξής :

"**Δεδομένων δύο προτάσεων C1 και C2 που είναι σε διαζευκτική μορφή, εάν υπάρχει ένα λεκτικό L1 στην C1 το οποίο είναι αντίθετο με ένα λεκτικό L2 της C2, τότε δημιούργησε μιά νέα πρόταση C που αποτελείται από τις C1, C2 σε διάζευξη αφού έχεις διαγράψει τα L1 και L2**".

Η πρόταση C ονομάζεται αποφαινόμενη πρόταση (resolvent) των C1 και C2. Οι προτάσεις C1 και C2 ονομάζονται γονικές προτάσεις της C.

Παράδειγμα 2.15

Η αποφαινόμενη πρόταση μεταξύ των $\neg p\vee q\vee r$ και $\neg q\vee s$ είναι η $\neg p\vee r\vee s$.

2.7. Απόδειξη με την αρχή της Απόφασης

Η αρχή της απόφασης είναι ένας κανόνας συμπερασμού. Γιά να κάνουμε μηχανιστική και αυτοματοποιημένη απόδειξη με αυτό τον κανόνα, χρειαζόμαστε ένα αλγόριθμο. Ο αλγόριθμος απόδειξης (proof procedure) που παρατίθεται παρακάτω υλοποιεί την "εις άτοπο απαγωγή". Υποθέτει κανείς ότι η προς απόδειξη πρόταση δεν είναι αληθής, οπότε την αντιστρέφει, και την εισάγει στο σύνολο των αξιωμάτων. Εφαρμόζει στη συνέχεια διαδοχικά την αρχή της απόφασης μεταξύ των προτάσεων που διαθέτει και συνεχώς εισάγει τις αποφαινόμενες προτάσεις στο αρχικό σύνολο προτάσεων. Η διαδικασία συνεχίζεται έως ότου φθάσει στη κενή πρόταση. Ετσι έχει φθάσει σε άτοπο.

Αλγόριθμος απόδειξης με την αρχή της απόφασης

Θεωρούμε ένα σύνολο προτάσεων Σ και την πρόταση φ που θέλουμε να αποδείξουμε. Ο αλγόριθμος έχει ως εξής :
Βήμα 1ο. Μετατρέπουμε όλες τις προτάσεις του Σ σε κανονική συζευκτική μορφή και δημιουργούμε έτσι το σύνολο Σ' διαθεσίμων προτάσεων.
Βήμα 2ο. Αντιστρέφουμε την φ (σε $\neg\varphi$), την μετατρέπουμε σε κανονική συζευκτική μορφή και ενώνουμε το σύνολο των παραγομένων προτάσεων Φ' με το Σ' : $\Sigma'' = \Sigma' \cup \Phi'$
Βήμα 3ο. Επαναλαμβάνουμε το παρακάτω βήμα έως ότου βρεθεί κάποια αντίφαση ή έως ότου δεν μπορεί να γίνει καμμία πρόοδος :

> Διαλέγουμε τυχαία δύο προτάσεις από το σύνολο Σ" των διαθέσιμων προτάσεων και εφαρμόζουμε μεταξύ τους την αρχή της απόφασης. Εάν η παραγόμενη πρόταση είναι η άδεια πρόταση, αυτό σημαίνει ότι φτάσαμε σε αντίφαση και σταματάμε, αλλιώς την προσθέτουμε στο σύνολο των διαθέσιμων προτάσεων και συνεχίζουμε.

Ορισμός 2.14 Απόδειξη με την αρχή της απόφασης
Απόδειξη της φ από το Σ στο σύστημα παραγωγής που χρησιμοποιεί την αρχή της απόφασης, (συμβολίζουμε $\Sigma \vdash_r \varphi$), λέγεται μιά πεπερασμένη ακολουθία προτάσεων $\psi_1,\psi_2, ...,\psi_v$, τέτοια ώστε $\psi_v=\square$ και κάθε ψ_i είναι ή πρόταση του $\Sigma'\cup\{\neg\varphi\}'$ (όπου ο τόνος υποδηλώνει ότι τα σύνολα έχουν μετατραπεί σε κανονική συζευκτική μορφή), ή παράγεται με την αρχή της απόφασης από δύο προηγούμενες προτάσεις της ακολουθίας.

Παράδειγμα 2.16 Ας υποθέσουμε ότι διαθέτουμε τις παρακάτω προτάσεις :
1. Θα παρακολουθήσω το έργο, εάν έχω έγχρωμη τηλεόραση και δεν είμαι απασχολημένος.
2. Θα γράψω το έργο, εάν έχω video και είμαι απασχολημένος.
3. Θα παρακολουθήσω το έργο εάν το έχω γράψει.
4. Εχω έγχρωμη τηλεόραση.
5. Εχω video.

Κατ'αρχήν ορίζουμε τις ατομικές προτάσεις : p="Θα παρακολουθήσω το έργο", q="έχω έγχρωμη τηλεόραση", r="είμαι απασχολημένος", s="θα γράψω το έργο", t="έχω video". Με βάση αυτή την ανάθεση οι παραπάνω προτάσεις γίνονται :

$$\Sigma=\{ p \leftarrow q \wedge \neg r,\ \ s \leftarrow t \wedge r,\ \ p \leftarrow s,\ \ q,\ t\}.$$

Στη συνέχεια μετατρέπουμε τις προτάσεις σε κανονική συζευκτική μορφή :

$$\Sigma'=\{ p \vee \neg q \vee r,\ \ s \vee \neg t \vee \neg r,\ \ p \vee \neg s,\ \ q,\ t\}.$$

Η πρόταση που θέλουμε να αποδείξουμε είναι η p. Αρα βρίσκουμε την αντίστροφή της $\neg$p, την μετατρέπουμε σε κανονική συζευκτική μορφή (πάλι $\neg$p), και την εισάγουμε στο Σ' ώστε να πάρουμε το $\Sigma''=\{ p \vee \neg q \vee r,\ \ s \vee \neg t \vee \neg r,\ \ p \vee \neg s,\ \ q,\ t,\ \neg p\}$. Τελικά αρχίζουμε να εφαρμόζουμε διαδοχικά την αρχή της απόφασης :

1. $p \vee \neg q \vee r$, που είναι πρόταση του Σ",
2. $\neg p$, που είναι πρόταση του Σ,
3. $\neg q \vee r$, εφαρμόζοντας την αρχή της απόφασης στις 1,2,
4. $s \vee \neg t \vee \neg r$, που είναι πρόταση του Σ",
5. $p \vee \neg s$, που είναι πρόταση του Σ",

6. $p \vee \neg t \vee \neg r$, εφαρμόζοντας την αρχή της απόφασης στις 5,6,
7. $\neg t \vee \neg r$, εφαρμόζοντας την αρχή της απόφασης στις 2,6,
8. t, που είναι πρόταση του Σ",
9. $\neg r$, εφαρμόζοντας την αρχή της απόφασης στις 7,8,
10. q, που είναι πρόταση του Σ",
11. r, εφαρμόζοντας την αρχή της απόφασης στις 3,10,
12. $\square$, η άδεια πρόταση εφαρμόζοντας την αρχή της απόφασης στις 9,11,

Η αρχή της απόφασης θα μας απασχολήσει κατά ιδιαίτερο τρόπο στα επόμενα κεφάλαια. Θα δούμε πως σ'αυτή στηρίζεται ολόκληρος ο Λογικός Προγραμματισμός και κατ'επέκταση η γλώσσα Prolog. Πρόκειται γιά μιά πολύ απλή αρχή η οποία μοιάζει με το Modus Ponens και είναι στην πραγματικότητα μιά γενίκευση του.

Η απόδειξη με την αρχή της απόφασης είναι η γνωστή μας απαγωγή σε άτοπο και χρησιμοποιείται πολύ εκτεταμένα σε συστήματα απόδειξης που βασίζονται στο Λογικό Προγραμματισμό.

3

Κατηγορηματική Λογική α’ τάξης

3.1 Εισαγωγή

Η προτασιακή λογική είναι πολύ στοιχειώδες μέσον γιά την αναπαράσταση και επεξεργασία της γνώσης, γιατί έχει πολύ περιορισμένες δυνατότητες. Γιά παράδειγμα, από τις προτάσεις "σε κάθε νησί μπορείς να πας με πλοίο", "η Ρόδος είναι νησί", εύκολα συμπεραίνουμε ότι "στη Ρόδο μπορείς να πας με πλοίο". Η προτασιακή λογική όμως δεν έχει τρόπο να αναπαραστήσει την γνώση των απλών αυτών προτάσεων και να εφαρμόσει συμπερασματολογία ακόμα και σε τόσο απλές προτάσεις. Ο λόγος είναι (η εννοούμενη από την φυσική γλώσσα) ύπαρξη καθολικού ποσοδείκτη (κάθε νησί...) που ταυτόχρονα προυποθέτει την ύπαρξη μεταβλητών.

Αντίθετα, στη κατηγορηματική λογική α’ τάξης (First Order Predicate Logic, FOPL), που γιά απλότητα θα ονομάζουμε από εδώ και πέρα κατηγορηματική λογική ή απλά ΚΛ, μπορούμε να αναπαραστήσουμε γνώση που αναφέρεται σε ένα σύνολο αντικειμένων και στις σχέσεις τους, και να εξετάσουμε ως προς την αλήθεια τους προτάσεις γιά αυτά τα αντικείμενα.

3.2 Συντακτικό της ΚΛ

Η Κατηγορηματική Λογική είναι ένα τυπικό σύστημα που μας προμηθεύει με μιά αυστηρά καθορισμένη (τυπική) γλώσσα με την οποία μπορούμε να αναπαραστήσουμε κάποια από τα πράγματα που σκεφτόμαστε. Γιά να την μελετήσουμε λοιπόν πρέπει να ορίσουμε αυτό το συντακτικό.

α) Χαρακτήρες Οι επιτρεπόμενοι χαρακτήρες είναι οι παρακάτω :
A B C D E F G H I J K L M N O P Q R S T V U W X Y Z
a b c d e f g h i j k l m n o p q r s t v u w x y z
Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω
1 2 3 4 5 6 7 8 9 0
. , ( ) { } [ ] + - * / ^
$\in \cup \cap = < > \leq \geq \subset \supset \subseteq \supseteq \neg \wedge \vee \forall \exists \rightarrow \leftarrow \leftrightarrow$

β) Μεταβλητές Γράφουμε τις μεταβλητές με ονόματα που ξεκινούν από κεφαλαία γράμματα ελληνικά ή αγγλικά. Γιά παράδειγμα τα σύμβολα X,Y,Z, Human, Ανθρωπος, είναι μεταβλητές.

γ) Σταθερές Γράφουμε τις σταθερές με ονόματα που περιέχουν μικρά γράμματα ελληνικά ή αγγλικά. Γιά παράδειγμα τα σύμβολα a, b, c, α, β, γ, πετρος, κωστας, ελενη, είναι σταθερές.

δ) Συναρτησιακά σύμβολα Τα γράφουμε όπως ακριβώς τις σταθερές. Οι συναρτήσεις αποτελούνται από ένα συναρτησιακό σύμβολο και ένα αριθμό παραμέτρων.

ε) Σύμβολα σχέσεων (ή σύμβολα κατηγορημάτων, predicate symbols)
Τα γράφουμε είτε με μικρά είτε με κεφαλαία (δεν υπάρχει κίνδυνος σύγχισης). Τα κατηγορήματα αποτελούνται από ένα σύμβολο σχέσης και από ένα αριθμό παραμέτρων.

Μερικά σύμβολα που έχουν καθιερωθεί και μπορούν να χρησιμοποιηθούν σαν συναρτησιακά σύμβολα είναι οι συναρτησιακοί τελεστές : + - * / ^ ∪ ∩ καθώς και τα σύμβολα {}[]. Ο τρόπος που συντάσουμε ένα συναρτησιακό n-θέσεων είναι π(τ1,τ2,...,τn) και ονομάζεται προδιάταξη (prefix) γιατί το σύμβολο του συναρτησιακού προηγείται των ορισμάτων του. Μερικά όμως συναρτησιακά δύο ορισμάτων συντάσσονται κάπως αλλιώς μέσω αυτού που αποκαλούμε ενδοδιάταξη (infix), κατά τέτοιο τρόπο που το σύμβολο μπαίνει ανάμεσα στα δύο ορίσματα. Γιά παράδειγμα αντί γιά +(τ1,τ2) γράφουμε τ1+τ2. Ετσι μπορούμε να γράψουμε :

$$\tau1+\tau2,\ \tau1-\tau2,\ \tau1*\tau2,\ \tau1/\tau2,\ \tau1\hat{}\tau2,\ \tau1 \cup \tau2,\ \tau1 \cap \tau2$$

Πρέπει ακόμα να σημειωθεί εδώ ότι με τον συμβολισμό {τ1,τ2,...,τn} εννοούμε ένα σύνολο, ενώ με τον συμβολισμό [τ1,τ2,...,τn] εννοούμε μιά διατεταγμένη λίστα.

Μερικά άλλα σύμβολα που μπορούν να χρησιμοποιηθούν σαν σύμβολα σχέσεων είναι οι παρακάτω μαθηματικοί τελεστές οι οποίοι επίσης συντάσσονται με ενδοδιάταξη.

$$=,\ <,\ >,\ \leq,\ \geq,\ \in,\ \subset,\ \supset,\ \subseteq,\ \supseteq$$

Με βάση αυτά, τα παρακάτω είναι υποδηλώνουν σχέσεις :

$$\tau1=\tau2+\tau3*\tau4,\ \tau1 \in (\tau1 \cup \tau2),\ \tau1 \supset (\tau2 \cap \tau3)$$

που σε προδιάταξη θα γραφόντουσαν σαν :

$$=(\tau1,+(\tau2,*(\tau3,\tau4))),\ \in(\tau1,\ \cup(\tau1,\tau2),\ \supset(\tau1,\cap(\tau2,\tau3)$$

Ας σημειωθεί ότι θα μπορούσαμε να είχαμε δηλώσει δικές μας σχέσεις και συναρτησιακά, με την ίδια σημασιολογία, και να γράφαμε τα παραπάνω με ένα διαφορετικό τρόπο :

equals(τ1,sum(τ2,product(τ3,τ4))), member(τ1,union(τ1,τ2), superset(τ1,intersection(τ2,τ3)

Η ΚΛ είναι μιά επέκταση του προτασιακού λογισμού. Γιά να γράψουμε προτάσεις, πρέπει να εισάγουμε με αυστηρό τρόπο τις εξής τρείς έννοιες, τους όρους (terms), τα κατηγορηματικά σύμβολα (predicate symbols, predicates), και τους ποσοδείκτες (quantifiers).

Ορισμός 3.1 Ορισμός όρων
Ενας ***όρος*** (term) ορίζεται αναδρομικά ως εξής :
1. Μιά σταθερά είναι ένας όρος.
2. Μιά μεταβλητή είναι ένας όρος.

3. Εαν f είναι ένα συναρτησιακό σύμβολο n-ορισμάτων, και τ1,τ2,...,τn είναι όροι, τότε το f(τ1,τ2,...,τn) είναι επίσης όρος.
4. Ολοι οι όροι που μπορούμε να παράγουμε υπακούουν στους παραπάνω κανόνες.

Παραδείγματα σταθερών, μεταβλητών και συναρτησιακών έχουν δοθεί λίγο προηγούμενα. Πρέπει όμως να δοθεί προσοχή στη διάκριση μεταξύ συναρτησιακών και κατηγορημάτων, επειδή συντακτικά μοιάζουν.

Ορισμός 3.2 Ατομο - Κατηγόρημα
Εάν p είναι ένα κατηγορηματικό σύμβολο n-θέσεων και t1,t2,...,tn είναι όροι τότε το p(t1,t2,...,tn) είναι άτομο ή κατηγόρημα. Καμμιά άλλη έκφραση δεν μπορεί να είναι άτομο.

Παράδειγμα 3.1 Τα παρακάτω είναι κατηγορήματα

ανθρωπος(πετρος)
X≥Y
project(leader(X),programmer(peter),secretary(helen))
i_am_a_list([1,2,3,5,7,8,16])

Η κύρια διαφορά ενός κατηγορήματος από ένα συναρτησιακό είναι ότι το κατηγόρημα, όταν αντικατασταθούν οι μεταβλητές του με κάποιες σταθερές, παίρνει τιμη 'Αλήθεια' ή 'Ψέμα'. Αντίθετα το συναρτησιακό παίρνει τιμή μιά σταθερά.

Ετσι γιά παράδειγμα το σύμβολο ⊃ είναι κατηγόρημα ενώ το σύμβολο ∩ είναι συναρτησιακό, καθώς η τιμή του {1,2,3} ⊃ {2,3,4} είναι 'Ψέμα' ενώ η τιμή του {1,2,3} ∩ {2,3,4}είναι η σταθερά {2,3} (εφόσον βέβαια υποθέσουμε ότι τα σύνολα μπορούν να θεωρηθούν σταθερές).

Η διάκρισή τους όμως είναι θέμα σημασιολογίας. Συντακτικά σχηματίζονται με τους ίδιους κανόνες, αλλά ένα συναρτησιακό δεν μπορεί να είναι μέλος μιάς πρότασης, παρά μόνο αν είναι όρισμα ενός κατηγορήματος. Αυτό φαίνεται στον ορισμό των καλοσχηματισμένων προτάσεων (βλ.παρακάτω).

Ορισμός 3.3 Ποσοδείκτες
Οι ποσοδείκτες είναι δύο σύμβολα, το "∃" (Exists, Υπαρχει) και το "∀" (All, για κάθε) που ονομάζονται αντίστοιχα υπαρξιακός και καθολικός ποσοδείκτης.

Οι ποσοδείκτες χρησιμοποιούνται γιά να δώσουν 'νόημα' στις ελεύθερες μεταβλητές ενός κατηγορήματος. Ετσι αν γράψουμε (∃X) ανθρωπος(X)... εννοούμε ότι *υπάρχει τουλάχιστον ένας* άνθρωπος που ... Γενικά, ο υπαρξιακός ποσοδείκτης (∃X) χρησιμοποιείται γιά εκφράσεις της μορφής '*υπάρχει ένα X*', '*για μερικά X*', '*για τουλάχιστον ένα X*', '*για (τουλάχιστον) ένα συγκεκριμένο X*', κ.λ.π. Αντίθετα, αν γράψουμε (∀X) ανθρωπος(X)... εννοούμε *ότι γιά κάθε άνθρωπο που* ... Ετσι, ο καθολικός ποσοδείκτης (∀X) χρησιμοποιείται γιά εκφράσεις της μορφής '*για όλα τα X*', '*για κάθε X*', κ.λ.π.

Ολα τα παραπάνω τα συνδυάζουμε κατάλληλα ώστε να σχηματίσουμε προτάσεις της ΚΛ. Οι προτάσεις ακολουθούν και αυτές ενα αυστηρό συντακτικό όπως φαίνεται από τον επόμενο ορισμό. Ενα σύνολο προτάσεων του ΚΛ αποτελείται από καλοσχηματισμένες προτάσεις, που είναι βέβαια οι γνωστές μας προτάσεις Λογικής α' τάξης που χρησιμοποιούμε στα μαθηματικά γιά να περιγράψουμε διάφορες θεωρίες. Υπάρχουν και άλλα ήδη προτάσεων όπως θα δούμε παρακάτω, αλλά κατά την συγγραφή ενός συνόλου προτάσεων του ΚΛ ξεκινάμε συνήθως από τις καλοσχηματισμένες προτάσεις.

Ορισμός 3.4 Καλοσχηματισμένες προτάσεις (wffs)
Οι καλοσχηματισμένες προτάσεις (well formed formulas, wffs), στον ΚΛ ορίζονται αναδρομικά ως εξής:

1. Ενα άτομο ή κατηγόρημα είναι μιά wff.
2. Εάν φ,ψ είναι wffs, τότε οι εκφράσεις $\neg\varphi$, $\varphi\vee\psi$, $\varphi\wedge\psi$, $\varphi\rightarrow\psi$, $\varphi\leftrightarrow\psi$, είναι επίσης wffs.
3. Εάν φ είναι μιά wff και X είναι μια ελεύθερη μεταβλητή που εμφανίζεται μέσα στην πρόταση φ, τότε οι εκφράσεις $(\forall X)\varphi$ και $(\exists X)\varphi$ είναι επίσης wffs.
4. Οι προτάσεις δημιουργούνται χρησιμοποιώντας μιά πεπερασμένη ακολουθία των κανόνων 1-3, και μόνον έτσι.

Παράδειγμα 3.2 Οι παρακάτω προτάσεις είναι wffs

$(\forall X)$ (άνθρωπος(X) $\rightarrow$ θνητός(X))
$(\exists X)$ (άνθρωπος(X) $\wedge$ ινδός(X))
$\neg(\exists X)$ (δεινόσαυρος(X) $\wedge$ ζωντανός(X))
$(\forall X)$ μόριο(X) $\wedge$ $(\forall Y)$ άτομο(Y) $\rightarrow$ $(\exists Z)$ ποδήλατο(Z)
$(\forall X)$ (φοιτητής(X) $\rightarrow$ $(\exists Z)$ (πανεπιστήμιο(Z) $\wedge$ πηγαίνει(X,Z)))
$(\forall X)$ $(\forall\Sigma)$ $(\forall T)$ (X $\in$ $\Sigma\cap T$ $\leftarrow$ X $\in$ Σ $\wedge$ X $\in$ T)
$(\forall X)$ άνθρωπος(X)

Μερικές από τις παραπάνω προτάσεις φαίνονται να έχουν κάποιο 'νόημα' και μερικές άλλες φαίνονται σαν ασυναρτησίες. Ομως η σημασιολογία δεν έχει σχέση με την σύνταξη. Για να βρούμε το πραγματικό νόημα μιάς πρότασης της ΚΛ πρέπει να μιλήσουμε γιά την ερμηνεία των προτάσεων.

Οι παρακάτω προτάσεις δεν είναι wffs

$(\forall X)$	είναι μόνο ποσοδείκτης
πέτρος	γιατί είναι μιά σταθερά
φοιτητής(γιώργος) $\rightarrow$ πανεπιστήμιο	γιατί το πανεπιστήμιο δεν είναι κατηγόρημα

Η **εμβέλεια** (scope) ενός ποσοδείκτη είναι το εύρος των προτάσεων στις οποίες έχει ισχύ. Σε μιά wff της μορφής (QX)G, όπου Q $\in$ $\{\forall, \exists\}$, η εμβέλεια (scope) του Q συνίσταται από εκείνα τα μέρη του G που δεν περιέχουν ποσοδείκτες γιά το X. Ολες οι μεταβλητές X που βρίσκονται στην εμβέλεια κάποιου ποσοδείκτη λέγονται **δεσμευμένες** (bound) από τον (QX). **Ελεύθερες** λέγονται οι μεταβλητές που δεν είναι δεσμευμένες. Στη πρόταση

$(\forall X)$ (φοιτητής(X) $\rightarrow$ $(\exists Z)$ (πανεπιστήμιο(Z) $\wedge$ πηγαίνει(X,Z)))

η εμβέλεια του $(\forall X)$ είναι όλη η πρόταση που ακολουθεί τον ποσοδείκτη, ενώ η εμβέλεια του $(\exists Z)$ είναι η υποπρόταση (πανεπιστήμιο(Z) $\wedge$ πηγαίνει(X,Z)). Γενικά, γιά να είμαστε ακριβείς στον ορισμό της εμβέλειας ενός συγκεκριμένου ποσοδείκτη χρησιμοποιούμε παρενθέσεις γιά να την υποδείξουμε. Οταν δεν χρησιμοποιούμε παρενθέσεις υπονοούμε ότι ο ποσοδείκτης εφαρμόζεται μόνο στο άτομο (ή κατηγόρημα) που ακολουθεί. Παράδειγμα :

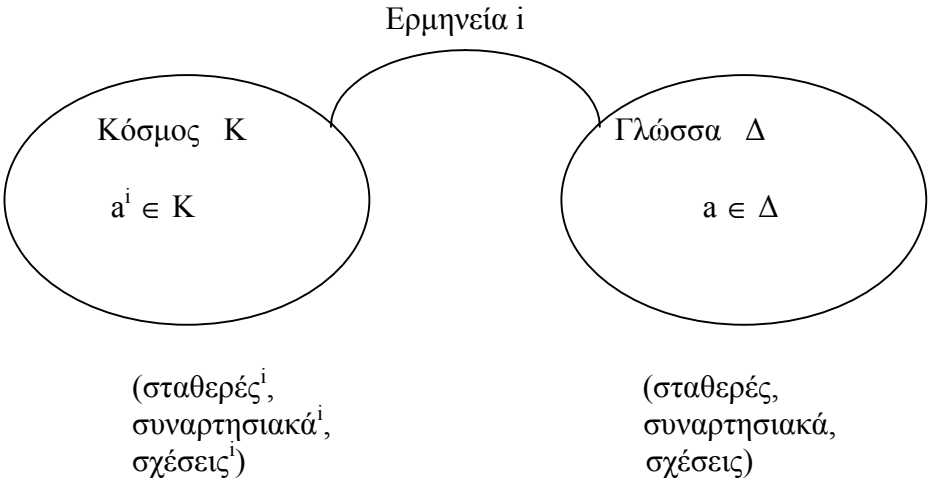
$(\exists X)$ άνθρωπος(X) $\wedge$ $(\exists Y)$ ινδός(Y)

3.3 Εννοιολογική αποτύπωση

Η εννοιολογική αποτύπωση (conceptualization) δεν είναι τίποτα άλλο παρά μιά περιγραφή της εννοιολογικής δομής που σχηματίζεται στον νου μας όταν παρατηρούμε ένα φαινόμενο ή μιά συγκυρία γεγονότων.

Το πρώτο πράγμα που παρατηρούμε όταν αντικρύζουμε ένα φαινόμενο, ή μιά συγκυρία γεγονότων, είναι το σύνολο των αντικειμένων που παίρνουν μέρος στο φαινόμενο. Τα αντικείμενα μπορεί να είναι συγκεκριμένα (αυτές οι σημειώσεις, ο ήλιος, κλπ) ή αφηρημένα (ο αριθμός 2, το σύνολο των ακεραίων, η έννοια της δικαιοσύνης). Μπορεί να είναι απλά (τα στοιχεία ενός κυκλώματος) ή σύνθετα (ένα σύνθετο αντικείμενο όπως ένα μηχάνημα). Μπορεί να είναι πραγματικά ή φανταστικά (Ο Ιούλιος Βερν). Ενα αντικείμενο είναι ο,τιδήποτε στο οποίο θέλουμε να αναφερθούμε. Στη συνέχεια αναρωτιώμαστε το πως αυτά τα αντικείμενα συναρτώνται μεταξύ τους, και τέλος 'καταγράφουμε' τις πιθανές σχέσεις και ιδιότητες που αληθεύουν για αυτά.

Η εννοιολογική αποτύπωση είναι το μαθηματικό εκείνο εργαλείο που θα μας επιτρέψει να αποτυπώσουμε τη μοντελοθεωρητική σημασιολογία της ΚΛ. Είναι μιά κάπως πολύπλοκη έννοια γιατί δεν μιλά απλά γιά συναρτήσεις, σχέσεις κ.λ.π. αλλά γιά συναρτήσεις, σχέσεις αντικείμενα του πραγματικού κόσμου, συναρτήσεις, σχέσεις και σύμβολα της τυπικής γλώσσας και την σχέση μεταξύ τους.



Σχήμα 3.1 Εννοιολογική αποτύπωση : ο κόσμος, η γλώσσα και η ερμηνεία

Τα τρία αυτά στοιχεία, δηλαδή τα αντικείμενα, οι συναρτήσεις τους και οι σχέσεις τους, που αποτυπώνονται στο νου μας αποτελούν αναπαραστάσεις του πραγματικού κόσμου. Το σύνολο των στοιχείων αυτών αποτελεί το **κόσμο K**, όπως φαίνεται στο αριστερό μέρος του σχήματος 3.1. Στην Τ.Ν. ο κόσμος είναι *οποιοδήποτε* κομμάτι του πραγματικού ή φανταστικού κόσμου το οποίο αποτελεί αντικείμενο ενδιαφέροντος ενός νοήμονος όντος. Ετσι ο κόσμος μπορεί να είναι: ο φυσικός κόσμος, ο κοινωνικός κόσμος, ο μαθηματικός κόσμος, ή (συνήθως) κάποια υποσύνολά τους. Χρησιμοποιώντας κάποια αφαίρεση του κόσμου προσπαθούμε να αναπτύξουμε κατ' αρχήν θεωρητικά - μαθηματικά συστήματα - μοντέλα, και στην συνέχεια πληροφοριακά συστήματα.

Η ανάπτυξη όμως ενός πληροφοριακού συστήματος απαιτεί την αναπαράσταση της γνώσης του κόσμου D, με την χρήση μιάς τυπικής γλώσσας ή ενός τυπικού συστήματος. Η

τυπική γλώσσα Δ (formal language), που αποτυπώνεται στα δεξιά του σχήματος 3.1, πρέπει να διαθέτει τα σύμβολα και τις συντακτικές εκείνες δομές που θα κάνουν την αναπαράσταση δυνατή. Πρέπει λοιπόν να διαθέτει σύμβολα γιά την αναπαράσταση αντικειμένων, συναρτήσεων και σχέσεων. Ετσι ένα νοήμον ον χρησιμοποιώντας σύμβολα από μιά τέτοια γλώσσα και αναπτύσσοντας ένα σύνολο αρχικών προτάσεων Σ μπορεί να αποτυπώσει με σύμβολα από τη γλώσσα Δ τον κόσμο Κ.

Η ΚΛ διαθέτει τέτοια συντακτικά μέσα : είναι οι όροι, τα κατηγορήματα και οι καλοσχηματισμένες προτάσεις που χρησιμοποιούμε γιά να συντάξουμε ένα σύνολο αρχικών προτάσεων που κατά την γνώμη μας περιγράφει με επιτυχία ένα φαινόμενο.

Το σύνολο αυτό των αρχικών προτάσεων ονομάζεται μερικές φορές και **σύνολο θεμελιωδών αξιωμάτων**, όπως έχουμε ήδη αναφέρει σε προηγούμενο κεφάλαιο, της θεωρίας που αποτυπώνουμε με την γλώσσα Δ.

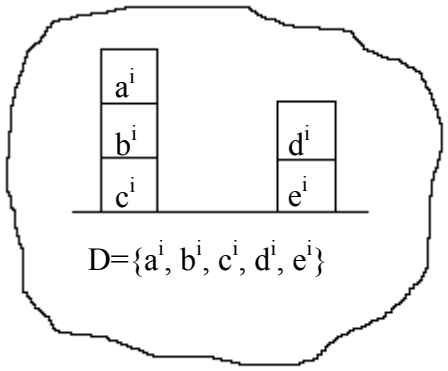
Η **ερμηνεία Ι** στη ΚΛ είναι η απεικόνιση που αντιστοιχεί κάθε σύμβολο της γλώσσας σε κάποιο από τα στοιχεία του κόσμου αναφοράς.

Παράδειγμα 3.3 Ενας κόσμος κύβων

Ας υποθέσουμε γιά παράδειγμα ότι μπροστά μας βρίσκεται ένα τραπέζι που έχει πάνω του πέντε κύβους α,β,γ,δ,ε, τέτοιους ώστε ο α είναι πάνω στον β, ο β πάνω στον γ, και ο γ πάνω στο τραπέζι. Λίγο πιό πέρα βρίσκονται και οι άλλοι δύο κύβοι, ο δ πάνω στον ε (σχήμα 3.2).

Το σύνολο των αντικειμένων που πέρνουν μέρος στην παραπάνω διάταξη είναι το D={α,β,γ,δ,ε} και ονομάζεται **Σύνολο ή Σύμπαν Αναφοράς** (Universe of Discourse). Πρέπει να σημειώσουμε εδώ ότι τα α,β,γ,δ,ε είναι οι πραγματικοί κύβοι που βρίσκονται πάνω στο πραγματικό τραπέζι. Δεν είναι οι κύβοι που απεικονίζονται στο σχήμα. Την απεικόνιση τη χρησιμοποιούμε μόνο γιά παραστατικούς λόγους.

Υπάρχει ένας μεγάλος αριθμός σχέσεων και συναρτήσεων των αντικειμένων. Ομως καθώς δεν μας ενδιαφέρουν όλες, θα αποτυπώσουμε μόνο ένα μέρος από αυτές κάνοντας μιά σχετική ‘αφαίρεση’.



Ενας κόσμος αντικειμένων
Σταθερές C={a, b, c, d, e}
Συναρτησιακά F={hat}
Σχέσεις R={on, above, table, clear}

Σχήμα 3.2 Εννοιολογική αποτύπωση : ένας κόσμος κύβων

Μπορούμε εύκολα να βρούμε ότι τα αντικείμενα συναρτώνται κατ'αρχήν με ένα προφανή τρόπο : Υπάρχει μιά συνάρτηση που αποτυπώνει κάθε αντικείμενο σε αυτό που είναι ακριβώς από πάνω του. Ας ονομάσουμε αυτή την συνάρτηση 'καπέλο'. Ετσι έχουμε καπέλο : D→D, δηλαδή γιά κάποιες τιμές του D={α,β,γ,δ,ε} παίρνει τιμές πάλι μέσα από το D={α,β,γ,δ,ε}. Η συνάρτηση ορίζεται γιά τις δυάδες {<β,α>,<γ,β>,<ε,δ>} (δηλαδή καπέλο(β)=α, καπέλο(γ)=β, καπέλο(ε)=δ).

Παρατηρείστε εδώ ότι γιά μερικά στοιχεία του D, η συνάρτηση καπέλο δεν ορίζεται. Ετσι γιά παράδειγμα δεν ορίζεται το καπέλο(α) γιατί κανένα αντικείμενο δεν βρίσκεται πάνω από το α. Θα μπορούσαμε να συμπληρώσουμε βέβαια άλλο ένα αντικείμενο στο D, τον ουρανό γιά παράδειγμα, και θα είχαμε D={α,β,γ,δ,ε,ουρανός} οπότε θα είχαμε καπέλο(α)=ουρανός, και την επιπρόσθετη δυάδα <α,ουρανός>. Ομως πάλι ποιά θα ήταν η τιμή του καπέλο(ουρανός); Αυτό είναι ένα πρόβλημα που σχετίζεται με το κατά πόσο προσδιορίζονται οι τιμές ενός συναρτησιακού μέσα σε όλο το D.

Παρατηρούμε ακόμη μερικές **σχέσεις** (relations) που αληθεύουν μεταξύ των αντικειμένων. Οι σχέσεις ορίζονται συνήθως σαν μαθηματικές συναρτήσεις της μορφής $D^n \rightarrow$ D, όπου n είναι ο αριθμός των ορισμάτων της σχέσης. Διακρίνουμε τις εξής σχέσεις :

- η σχέση 'ακριβώς-πάνω-από' που αληθεύει μεταξύ δύο αντικειμένων άν το ένα είναι ακριβώς πάνω από το άλλο, δηλ. αληθεύει γιά τις δυάδες {<α,β>,<β,γ>,<δ,ε>}, (δηλαδή η σχέση ακριβώς-πάνω-από(α,β) είναι αληθής γιατί το <α,β> δεν περιέχεται μεταξύ των δυάδων, ενώ η σχέση ακριβώς-πάνω-από(α,ε) είναι ψευδής γιατί το <α,ε> δεν περιέχεται μεταξύ των δυάδων)
- η σχέση 'πάνω-από' που αληθεύει μεταξύ δύο αντικειμένων άν το ένα είναι πάνω από το άλλο, δηλ. αληθεύει γιά τις δυάδες {<α,β>,<β,γ>,<α,γ>,<δ,ε>},
- η σχέση 'κορυφή' που αληθεύει γιά ένα αντικείμενο αν δεν υπάρχει άλλο αντικείμενο από πάνω του, δηλ. αληθεύει γιά το σύνολο {α,δ},
- η σχέση 'τραπέζι' που αληθεύει γιά ένα αντικείμενο αν είναι πάνω στο τραπέζι, δηλ. αληθεύει γιά το σύνολο {γ,ε}.

Για να αναπαραστήσουμε αυτή την απλή διάταξη, χρησιμοποιούμε κάποια σύμβολα της τυπικής γλώσσας, και στη συγκεκριμένη περίπτωση της ΚΛ. Ετσι λοιπόν γιά να αναπαραστήσουμε τους κύβους α,β,γ,δ,ε χρησιμοποιούμε τα σύμβολα a,b,c,d,e. Το σύνολο C={a,b,c,d,e} είναι το σύνολο των σταθερών που θα χρησιμοποιήσουμε. Κάθε σταθερά από το C απεικονίζεται σε ένα από τους κύβους από το D. Η απεικόνιση αυτή ονομάζεται ερμηνεία i.

Υπάρχουν πολλές δυνατές ερμηνείες. Κάποιος γιά παράδειγμα μπορεί να υποστηρίξει ότι η απεικόνιση αυτή έχει ως εξής :

1η ερμηνεία

$a \xrightarrow{i} \alpha$ δηλαδή $\alpha = a^i$
$b \xrightarrow{i} \beta$ δηλαδή $\beta = b^i$
$c \xrightarrow{i} \gamma$ δηλαδή $\gamma = c^i$
$d \xrightarrow{i} \delta$ δηλαδή $\delta = d^i$
$e \xrightarrow{i} \varepsilon$ δηλαδή $\varepsilon = e^i$

Κάποιος άλλος όμως μπορεί να μη συμφωνεί και να υποστηρίξει ότι :

2η ερμηνεία

$a \xrightarrow{i} \alpha$ δηλαδή $\alpha = a^i$
$b \xrightarrow{i} \gamma$ δηλαδή $\gamma = b^i$
$c \xrightarrow{i} \delta$ δηλαδή $\delta = c^i$
$d \xrightarrow{i} \varepsilon$ δηλαδή $\varepsilon = d^i$
$e \xrightarrow{i} \beta$ δηλαδή $\beta = e^i$

κ.λπ.

Οπως ακριβώς και στη Προτασιακή Λογική η ερμηνεία είναι έτσι κι'αλλιώς κάτι αυθαίρετο. Ομως στη ΚΛ η έννοια της ερμηνείας δεν τελειώνει εδώ. Πρέπει να ολοκληρωθεί με τις συναρτήσεις και τις σχέσεις. Γιά το συναρτησιακό 'καπέλο' χρησιμοποιούμε το σύμβολο hat.

Ετσι το hat - που είναι σύμβολο της τυπικής γλώσσας - απεικονίζεται στη συνάρτηση ‘καπέλο’ :

hat $\xrightarrow{i}$ καπέλο δηλαδή καπέλο = hat^i

Εδώ δεν μπορούν να υπάρχουν πολλές ερμηνείες γιατί υπάρχει ένα μόνο στοιχείο από κάθε πλευρά. Γιά τις σχέσεις όμως, γιά τις οποίες πρέπει επίσης να επινοήσουμε κάποια σύμβολα της γλώσσας, υπάρχουν επίσης πολλές ερμηνείες.

1η ερμηνεία

on $\xrightarrow{i}$ ακριβώς-πάνω-από δηλαδή ακριβώς-πάνω-από = on^i
above $\xrightarrow{i}$ πάνω-από δηλαδή πάνω-από = above^i
table $\xrightarrow{i}$ τραπέζι δηλαδή τραπέζι = table^i
clear $\xrightarrow{i}$ κορυφή δηλαδή κορυφή = clear^i

2η ερμηνεία

on $\xrightarrow{i}$ πάνω-από δηλαδή ακριβώς-πάνω-από = on^i
above $\xrightarrow{i}$ ακριβώς-πάνω-από δηλαδή πάνω-από = above^i
table $\xrightarrow{i}$ κορυφή δηλαδή κορυφή = table^i
clear $\xrightarrow{i}$ τραπέζι δηλαδή τραπέζι = clear^i

κ.λπ.

Αν γράψουμε στη γλώσσα μας το συναρτησιακό hat(b) τι θα σημαίνει αυτό; Για να βρούμε τι σημαίνει, πρέπει να βρούμε το $(\text{hat}(b))^i$, δηλαδή το $\text{hat}^i(b^i)$.
Στη πρώτη ερμηνεία έχουμε :

$(\text{hat}(b))^i = \text{hat}^i(b^i) = \text{καπέλο}(\beta) = \alpha$, δηλαδή ο κύβος α.

Ενώ στη δεύτερη ερμηνεία έχουμε :

$(\text{hat}(b))^i = \text{hat}^i(b^i) = \text{καπέλο}(\gamma) = \beta$, δηλαδή ο κύβος β.

Βλέπουμε λοιπόν ότι αλλάζει το νόημα των συμβόλων της γλώσσας ανάλογα με την ερμηνεία. Ομοια βέβαια ισχύει και γιά τις σχέσεις. Το above(b,c) έχει τις παρακάτω σημασίες. Στη πρώτη ερμηνεία έχουμε :
$(\text{above}(b,c))^i = \text{above}^i(b^i, c^i) = \text{πάνω-από}(\beta,\gamma) =$ ‘αληθές’,
διότι $<\beta,\gamma> \in$ πάνω-από

Αντίθετα στη δεύτερη ερμηνεία έχουμε :
$(\text{above}(b,c))^i = \text{above}^i(b^i, c^i) = \text{ακριβώς-πάνω-από}(\gamma,\delta) =$ ‘ψευδές’,
διότι $<\beta,\gamma> \notin$ ακριβώς-πάνω-από

Βλέπουμε λοιπόν ότι αλλάζοντας ερμηνεία, μπορεί να αλλάζουν όλα : Τα νοήματα των συμβόλων, των σχέσεων, οι τιμές αλήθειας, κ.λ.π. Τι συμβαίνει λοιπόν; Στην πραγματικότητα ο συγγραφέας του προγράμματος των προτάσεων είχε στο νου του κάποια συγκεκριμένη ερμηνεία, την πρώτη ερμηνεία σε αυτή την περίπτωση. Και έτσι έχοντας μιά μόνο ερμηνεία στο νού του έγραψε ένα σύνολο από προτάσεις γιά να περιγράψει τη διάταξη των κύβων. Θα μπορούσε γιά παράδειγμα να είχε γράψει τις παρακάτω προτάσεις :

$\Sigma=\{$ $(\forall X)(\forall Y)(\text{on}(X,Y) \rightarrow \text{above}(X,Y))$,
$(\forall X)\,(\forall Y)\,(\forall Z)(\,\text{above}(X,Y) \wedge \text{on}(Y,Z) \rightarrow \text{above}(X,Z))$,
table(c), table(e),
clear(a),clear(d),
on(a,b), on(b,c), on(d,e)$\}$

Εμείς εύκολα συμπεραίνουμε την υπονοούμενη ερμηνεία των συμβόλων, λόγω των αγγλικών όρων. Αυτό όμως δεν συμβαίνει πάντοτε. Εχουμε δηλαδή εδώ το ίδιο πρόβλημα που είχαμε και στη Προτασιακή Λογική. Αν δεν γνωρίζουμε την υπονοούμενη ερμηνεία, πως θα συμπεράνουμε αν μιά νέα πρόταση αληθεύει στο Σ; Πρέπει ακόμη να λάβουμε υπόψη μας ότι στη ΚΛ οι δυνατές ερμηνείες μπορεί να είναι άπειρες. Εχουμε δηλαδή να αντιμετωπίσουμε ένα πρόβλημα ακόμη πιό δύσκολο από αυτό της ΠΛ.

Πριν λύσουμε όμως αυτό το πρόβλημα, θα πρέπει να χρησιμοποιήσουμε αυστηρούς ορισμούς γιά την σημασιολογία.

3.4 Η σημασιολογία στη ΚΛ

Ορισμός 3.6 Η ερμηνεία στη ΚΛ
Η ερμηνεία μιάς πρότασης φ πάνω σε ένα κόσμο K (σύνολο αναφοράς, Universe of Discource), στη ΚΛ, συνίσταται στον ορισμό ενός μη κενού πεδίου D, που αποτελείται από τα αντικείμενα του K, και στην ανάθεση τιμών σε κάθε σταθερά, συναρτησιακή παράσταση και κατηγόρημα της φ ως εξής :

1. $\sigma \in C \Rightarrow \sigma^i \in D$,

δηλαδή, σε κάθε σταθερά σ της γλώσσας αναθέτουμε ένα αντικείμενο του κόσμου D,

2. $\pi \in F \Rightarrow \pi^i : D^n \rightarrow D$,

δηλαδή, σε κάθε συναρτησιακή παράσταση π, n ορισμάτων, αναθέτουμε μιά είναι αντιστοιχία από το D^n στο D.

3. $\rho \in R \Rightarrow \rho^i : D^n \rightarrow \{0, 1\}$ (ή $\rho^i \subseteq D^n$), δηλαδή κάθε κατηγορηματικό σύμβολο n-ορισμάτων αναθέτουμε μιά αντιστοιχία από το D^n στο $\{0,1\}$.

Πολλές φορές ένα κατηγορηματικό σύμβολο n-παραμέτρων ορίζεται σαν μιά αντιστοιχία που το σύνολο τιμών της είναι υποσύνολο του D^n. Σε μιά τέτοια περίπτωση υπονούμε ότι στο κατηγορηματικό σύμβολο αντιστοιχεί ένα σύνολο από n-άδες που αληθεύουν (έχουν την τιμή αλήθειας 1).

Παράδειγμα 3.4 Στο παράδειγμα με τους κύβους έχουμε C={a,b,c,d,e}, F={hat}, R={on, above, table, clear}. Οσον αφορά στις σταθερές έχουμε :

$a \in C \Rightarrow a^i \in D$, και $a^i = \alpha$,
$b \in C \Rightarrow b^i \in D$, και $b^i = \beta$,
$c \in C \Rightarrow c^i \in D$, και $c^i = \gamma$,
$d \in C \Rightarrow d^i \in D$, και $d^i = \delta$,
$e \in C \Rightarrow e^i \in D$, και $e^i = \varepsilon$,

Οσον αφορά στα συναρτησιακά έχουμε :

$hat \in F \Rightarrow hat^i : D^1 \rightarrow D$, και hat^i = καπέλο
επίσης
$hat^i(b^i) = a^i$, δηλαδή καπέλο(β)=α (το α είναι καπέλο του β)
$hat^i(c^i) = b^i$, δηλαδή καπέλο(γ)=β (το β είναι καπέλο του γ)
$hat^i(e^i) = d^i$, δηλαδή καπέλο(ε)=δ (το δ είναι καπέλο του ε)

Οσον αφορά στις σχέσεις έχουμε τα ακόλουθα. Στη σχέση on της γλώσσας ανατίθεται η σχέση 'ακριβώς-πάνω-από' που ισχύει στον κόσμο με τους κύβους και αυτή η ανάθεση μπορεί να ορισθεί με δύο τρόπους :

$on \in R \Rightarrow on^i : D^2 \rightarrow \{0, 1\}$
$on^i(a^i,b^i)=1$, δηλαδή ακριβώς-πάνω-από(α,β)='αληθές'
$on^i(b^i,c^i)=1$, δηλαδή ακριβώς-πάνω-από(β,γ)='αληθές'
$on^i(d^i,e^i)=1$, δηλαδή ακριβώς-πάνω-από(δ,ε)='αληθές'
$on^i(a^i,a^i)=0$, δηλαδή ακριβώς-πάνω-από(α,α)='ψευδές'
$on^i(a^i,c^i)=0$, δηλαδή ακριβώς-πάνω-από(α,γ)='ψευδες',
... (όλες οι άλλες τιμές από το D^2 είναι επίσης ψευδείς.

Ενας άλλος τρόπος να το ορίσουμε αυτό είναι :

$on \in R \Rightarrow on^i : \subseteq D^2$
$on^i=\{<a^i,b^i>,<b^i,c^i>,<d^i,e^i>\}$,

ή ισοδύναμα όπως γράψαμε στο παραδειγμα με τους κύβους, ακριβώς-πάνω-από=$\{<\alpha,\beta>,<\beta,\gamma>,<\delta,\varepsilon>\}$. Ομοια εργαζόμαστε και γιά τις άλλες σχέσεις.

Οπως βλέπουμε ο ορισμός μιάς ερμηνείας είναι αρκετά επίπονη και στρυφνή εργασία. Και φανταστείτε ότι υπάρχει πολύ μεγάλος αριθμός ερμηνειών, σε μερικές περιπτώσεις άπειρος. Ο λόγος που κάναμε τόσο αναλυτικά το παραπάνω παράδειγμα είναι γιά να εξοικειωθούμε με κάθε λεπτομέρεια του θέματος της ανάθεσης μιάς ερμηνείας. Πρακτικά, ποτέ δεν ακολουθούμε μιά τέτοια διαδικασία γιά ένα πραγματικό πρόβλημα. Συνήθως κρατάμε στο νου μας τις λεπτομέρειες της ερμηνείας που διαλέγουμε. Ομως γιά λόγους μαθηματικής πληρότητας πρέπει να είναι δυνατόν να την ακολουθήσουμε βήμα-βήμα και στο χαρτί.

Ο ορισμός 3.6 διατυπώνει με μαθηματική αυστηρότητα όλα αυτά που είπαμε παραπάνω. Ας παρατηρήσουμε ακόμη ότι χρησιμοποιούμε το σύμβολο '0' αντί του 'ψευδές' και το σύμβολο '1' αντί του 'αληθές'. Σε πολλά εγχειρίδια υπάρχουν πολλοί άλλοι συμβολισμοί. Το βασικό μας σύνολο είναι το D, που είναι το σύνολο των αντικειμένων του κόσμου αναφοράς. Στην πραγματικότητα, γιά κάθε σταθερά της πρότασης (ή του συνόλου των προτάσεων) υποθέτουμε ότι υπάρχει και ένα αντίστοιχο αντικείμενο στον κόσμο D. Ενας λοιπόν ορισμός του συνόλου αναφοράς D συνεπής με τον ορισμό 3.6 είναι:

Ορισμός 3.7 Σύνολο Αναφοράς
Το σύνολο αναφοράς D έχει σάν στοιχεία τα αντικείμενα που αντιστοιχούν σε όλους τους χωρίς μεταβλητές όρους που μπορούν να κατασκευαστούν από τις σταθερές και τις συναρτήσεις που εμφανίζονται στο σύνολο των προτάσεων.

Ορισμός 3.8 Ανάθεση μεταβλητών U
Εαν V είναι το σύνολο μεταβλητών της γλώσσας, τότε η ανάθεση μεταβλητών (variable assignment) U είναι μιά απεικόνιση από το V στο D, $X \in V \Rightarrow X^u \in D$

Παράδειγμα 3.5 Στην πρόταση $(\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y))$ εμφανίζονται οι μεταβλητές X και Y. Μιά ανάθεση γιά την X θα ήταν η $X^u = \alpha$, και μιά άλλη ανάθεση για την Y θα ήταν η $Y^u = \gamma$

Η ανάθεση των μεταβλητών συμπληρώνει την ερμηνεία και είναι χρήσιμη στην ανάθεση όρων. Ολα αυτά σε συνδυασμό είναι χρήσιμα στην αποτίμηση της τιμής αλήθειας μιάς πρότασης της Κατηγορηματικής Λογικής. Ακολουθεί ο ορισμός της ανάθεσης όρων. Πρέπει

βέβαια να σημειώσουμε εδώ ότι η ανάθεση μεταβλητών χρησιμοποιείται μόνο στην περίπτωση που υπάρχουν ελεύθερες μεταβλητές μέσα στις προτάσεις, μεταβλητές δηλαδή που δεν υπόκεινται στην εμβέλεια κάποιου ποσοδείκτη. Μπορούμε να αποφύγουμε τέτοιες περιπλοκές αν δεν χρησιμοποιούμε στις προτάσεις τέτοιες μεταβλητές.

Ορισμός 3.9 Ανάθεση όρων T_{IU}
Δεδομένης μιάς ερμηνείας I και μιάς ανάθεσης μεταβλητών U, η ανάθεση όρων (term assignment) T_{IU} είναι μιά απεικόνιση από τους όρους της γλώσσας στα αντικείμενα του κόσμου, και ορίζεται ως ακολούθως :

αν $\tau \in C \Rightarrow T_{IU}(\tau) = \tau^i$
αν $\tau \in V \Rightarrow T_{IU}(\tau) = \tau^u$
αν $\tau \in F$, δηλαδή $\tau = \pi(\tau_1, ..., \tau_n)$, και $\pi^i = g$, $T_{IU}(\tau_j) = x_j$
$\Rightarrow T_{IU}(\tau) = g(x_1, ..., x_n)$

Παράδειγμα 3.6 Θεωρείστε την ερμηνεία και την ανάθεση μεταβλητών των προηγουμένων παραδειγμάτων. Με βάση αυτές στο συναρτησιακό hat(c) πρέπει να ανατεθεί το κουτί β του κόσμου D. Διότι :

hat $\in$ F και hat^i = καπέλο, και c $\in$ C και $T_{IU}(c) = c^i = \gamma$, οπότε
$T_{IU}(hat(c)) = hat^i(c^i) = καπέλο(\gamma) = \beta$

Στο συναρτησιακό hat(hat(Y)) πρέπει να ανατεθεί το κουτί α του κόσμου D, διότι :

$T_{IU}(hat(hat(Y))) = hat^i(hat^i(Y^u)) = καπέλο(καπέλο(\gamma)) = καπέλο(\beta) = \alpha$

Ολοι αυτοί οι πολύπλοκοι ορισμοί είναι απαραίτητοι γιά τον ορισμό της αλήθειας μιάς προτασης της ΚΛ. Η αλήθεια αυτή ορίζεται μέσα από τις προτάσεις ικανοποιησιμότητας (satisfaction formulae) της ΚΛ. Οι προτάσεις αυτές είναι γενίκευση των προτάσεων ικανοποιησιμότητας της Προτασιακής Λογικής (ορισμός 2.8). Στον παρακάτω ορισμό ο συμβολισμός $\models_i \varphi[u]$ συμβολίζει ότι η πρόταση φ ικανοποιείται από μια ερμηνεία i και μια ανάθεση μεταβλητών u.

Ορισμός 3.10 Προτάσεις ικανοποιησιμότητας στη ΚΛ
(1) $\models_i (\sigma=\tau)[u]$ ανν $T_{iu}(\sigma) = T_{iu}(\tau)$
(2) $\models_i \rho(\tau_1, ..., \tau_n)[u]$ ανν $< T_{iu}(\tau_1), ..., T_{iu}(\tau_n) > \in \rho^i$
(3) $\models_i (\neg\varphi)[u]$ ανν $\not\models_i \varphi[u]$
(4) $\models_i (\bigwedge_j \varphi_j)[u]$ ανν $(\forall j) \models_i \varphi_i[u]$
(5) $\models_i (\bigvee_j \varphi_j)[u]$ ανν $(\exists j) \models_i \varphi_i[u]$
(6) $\models_i (\varphi\rightarrow\psi)[u]$ ανν $\not\models_i \varphi[u]$ or $\models_i \psi[u]$
(7) $\models_i (\varphi\leftarrow\psi)[u]$ ανν $\models_i (\psi\rightarrow\varphi)[u]$
(8) $\models_i (\varphi\leftrightarrow\psi)[u]$ ανν $\models_i (\varphi\rightarrow\psi)[u]$ and $\models_i (\psi\rightarrow\varphi)[u]$
(9) $\models_i (\forall x) \varphi[u]$ ανν $\forall d \in D: \models_i \varphi[v]$, where V(x) = d and V(y) = u(y) for $y\neq x$
(10) $\models_i (\exists x) \varphi[u]$ ανν $\exists d \in D: \models_i \varphi[v]$, where V(x) = d and V(y) = u(y) for $y\neq x$

Ο ορισμός αυτός φαίνεται αρκετά ακατανόητος. Ας προσπαθήσουμε να κάνουμε μιά μικρή ανάλυση των συνθηκών ικανοποιησιμότητας :

Η (1) αναφέρεται στην ικανοποιησιμότητα ενός ειδικού κατηγορήματος που είναι η ισότητα, και λέει ότι δύο όροι είναι ίσοι αν τα αντίστοιχα αντικείμενα που ερμηνεύουν είναι ίσα.

Η (2) αναφέρεται στην ικανοποιησιμότητα ενός οποιουδήποτε άλλου κατηγορήματος, και εννοεί ότι ένα κατηγόρημα αληθεύει αν η ν-άδα των αντίστοιχων αντικειμένων στα οποία ερμηνεύονται οι όροι του ανήκουν στη αντίστοιχη σχέση από τον κόσμο D.

Παράδειγμα 3.7 Θεωρείστε το κατηγόρημα above(hat(hat(Y),Y) στην γνωστή, από τα άλλα παραδείγματα, ερμηνεία και στην ανάθεση μεταβλητών U, τέτοια ώστε $Y^u = \gamma$.
Δεδομένων αυτών μπορούμε να πούμε ότι :

$\models_i$ above(hat(hat(Y),Y) [u]
ανν $< T_{IU}(\text{hat(hat(Y)))}, T_{IU}(Y) > \in \text{above}^i$
ανν $< \alpha, \gamma > \in$ πάνω-από
που είναι 'αληθές'

Οι (3),(4),...,(8) αναφέρονται στις συνθήκες ικανοποιησιμότητας των λογικών συνδεσμων, και είναι συνθήκες παρόμοιες με αυτές της Προτασιακής Λογικής.

Οι (9) και (10) αναφέρονται στους ποσοδείκτες, και εννούν ότι γιά την μεταβλητή του ποσοδείκτη πρέπει να χρησιμοποιήσουμε μιά διαφορετική ανάθεση μεταβλητών, κρατώντας την παλιά ανάθεση μεταβλητών ίδια γιά τις υπόλοιπες (ελεύθερες) μεταβλητές. Ας παρατηρήσουμε ότι άν οι προτάσεις μας δεν έχουν ελεύθερες μεταβλητές, τότε η ανάθεση μεταβλητών U των συνθηκών (1)-(10) δεν χρησιμοποιείται.

Με πιό απλά λόγια, χωρίς πολλούς μαθηματικούς συμβολισμούς, και έχοντας υπόψη μας μιά συγκεκριμένη ερμηνεία i, ένας ισοδύναμος αλλά ποιό κατανοητός και απλός ορισμός είναι ο επόμενος :

Ορισμός 3.11 Η τιμή αλήθειας μιάς πρότασης στη ΚΛ
Γιά κάθε ερμηνεία μιάς πρότασης πάνω σε ένα πεδίο D, μπορούμε να υπολογίσουμε την τιμή της μέσα από το σύνολο {0,1}, χρησιμοποιώντας τους παρακάτω κανόνες :

1. Εάν έχουμε ήδη υπολογίσει τις τιμές των προτάσεων φ και ψ, τότε μπορούμε να υπολογίσουμε τις τιμές των προτάσεων $\neg\varphi$, $\varphi\vee\psi$, $\varphi\wedge\psi$, $\varphi\rightarrow\psi$, $\varphi\leftarrow\psi$, $\varphi\leftrightarrow\psi$, χρησιμοποιώντας τους πίνακες αλήθειας των λογικών συνδέσμων.
2. $(\forall X)\varphi$ αποτιμάται 1, εάν η τιμή του φ αποτιμάται 1 γιά κάθε στοιχείο d του συνόλου D (για το οποίο θα κάνουμε την ανάθεση X=d). Αλλιώς αποτιμάται 0.
3. $(\exists X)\varphi$ αποτιμάται 1, εάν υπάρχει τουλάχιστον ένα στοιχείο d του D γιά το οποίο η φ αποτιμάται 1 (για το οποίο θα κάνουμε την ανάθεση X=d). Αλλιώς αποτιμάται 0.

Παράδειγμα 3.8 Θεωρείστε τις προτάσεις $(\forall X)$ p(X) και $(\exists X)\ \neg$p(X), D={α, β}, και την ανάθεση $p^i = \{\alpha\}$. Θεωρούμε ακόμα ότι στη γλώσσα μας πρέπει να υπάρχουν δύο σταθερές a,b τέτοιες ώστε $a^i = \alpha$ και $b^i = \beta$. Τότε :

$\models_i (\forall X)\ p(X)$ [u]
ανν $\models_i p(a)$ και $\models_i p(b)$
ανν $\alpha \in p^i$ και $\beta \in p^i$
ανν αληθές και ψευδές, που είναι ψευδές

Οσον αφορά στη δεύτερη προταση έχουμε

$\models_i (\exists X)\ \neg p(X)$ [u]
ανν $\models_i \neg p(a)$ ή $\models_i \neg p(b)$
ανν $\not\models_i p(a)$ ή $\not\models_i p(b)$
ανν $\alpha \notin p^i$ ή $\beta \notin p^i$
ανν ψευδές ή αληθές, που είναι αληθές

Παράδειγμα 3.9 Θεωρείστε την πρότάση $(\forall X)\ (\exists Y)\ p(X,Y)$, $D=\{\alpha, \beta\}$, και την ανάθεση $p^i = \{<\alpha, \alpha>,.<\beta, \beta>\}$ Θεωρούμε ακόμα ότι στη γλώσσα μας πρέπει να υπάρχουν δύο σταθερές a,b τέτοιες ώστε $a^i = \alpha$ και $b^i = \beta$.Τότε :

$\models_i (\forall X)\ (\exists Y)\ p(X,Y)$ [u]
ανν $\models_i (\exists Y)\ p(a,Y)$ και $\models_i (\exists Y)\ p(b,Y)$
ανν $(\models_i p(a,a)$ ή $\models_i p(a,b))$ και $(\models_i p(b,a)$ ή $\models_i p(b,b))$
ανν $(<a^i, a^i> \in p^i$ ή $<a^i, b^i> \in p^i)$ και
$(<b^i, a^i> \in p^i$ ή $<b^i, b^i> \in p^i)$
ανν $(<\alpha, \alpha> \in p^i$ ή $<\alpha,\beta> \in p^i)$ και
$(<\beta, \alpha> \in p^i$ ή $<\beta, \beta> \in p^i)$
ανν (αληθές ή ψευδές) και (ψευδές ή αληθές), που είναι αληθές

Παράδειγμα 3.10 Θεωρείστε την πρότάση $(\forall X)\ (p(X) \rightarrow q(f(X),a))$, και $D=\{\alpha,\beta\}$. Στην γλώσσα των προτάσεων υπάρχει ήδη μιά σταθερά, η a. Θεωρούμε ακόμα μια σταθερά, την b, και ότι $a^i = \alpha$ και $b^i = \beta$. Εχουμε και την ανάθεση $f^i = \{<\alpha,\beta>,<\beta,\alpha>\}$ για το συναρτησιακό και τις αναθέσεις $p^i = \{\beta\}$ και $q^i = \{<\alpha,\alpha>,<\alpha,\beta>,<\beta,\beta>\}$ γιά τα κατηγορηματικά σύμβολα. Τότε :

$\models_i (\forall X)\ (p(X) \rightarrow q(f(X),a))$
ανν $\models_i (p(a) \rightarrow q(f(a),a))$ και $\models_i (p(b) \rightarrow q(f(b),a))$
ανν $\models_i (\neg p(a) \vee q(f(a),a))$ και $\models_i (\neg p(b) \vee q(f(b),a))$
ανν $(\models_i \neg p(a)$ ή $\models_i q(f(a),a))$ και $(\models_i \neg p(b)$ ή $\models_i q(f(b),a))$
ανν $(\not\models_i p(a)$ ή $\models_i q(f(a),a))$ και $(\not\models_i p(b)$ ή $\models_i q(f(b),a))$
ανν $(a^i \notin p^i$ ή $<f^i(a)^i, a^i> \in q^i)$ και $(b^i \notin p^i$ ή $<f^i(b)^i, a^i> \in q^i)$
ανν $(\alpha \notin p^i$ ή $<\beta, \alpha> \in q^i)$ και $(\beta \notin p^i$ ή $<\alpha, \alpha> \in q^i)$
ανν (αληθές ή ψευδές) και (ψευδές ή αληθές), που είναι αληθές

Παράδειγμα 3.11 Θεωρείστε την πρόταση $(\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y))$ και την γνωστή από το παράδειγμα με τους κύβους υπονοούμενη ερμηνεία. Τότε, δεδομένου ότι δεν έχουμε ελεύθερες μεταβλητές, δεν χρειαζόμαστε το U, και :

$\models_i ((\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y)))$ [u]
ανν $\models_i ((\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y)))$
ανν $\models_i ((\forall Y)(on(a,Y) \rightarrow above(a,Y)))$ και
$\models_i ((\forall Y)(on(b,Y) \rightarrow above(b,Y)))$ και
$\models_i ((\forall Y)(on(c,Y) \rightarrow above(c,Y)))$ και
$\models_i ((\forall Y)(on(d,Y) \rightarrow above(d,Y)))$ και
$\models_i ((\forall Y)(on(e,Y) \rightarrow above(e,Y)))$
ανν $\models_i (on(a,a) \rightarrow above(a,a))$ και ... και $\models_i (on(a,e) \rightarrow above(a,e))$ και
$\models_i (on(b,a) \rightarrow above(b,a))$ και ... και $\models_i (on(b,e) \rightarrow above(b,e))$ και
...
$\models_i (on(e,a) \rightarrow above(e,a))$ και ... και $\models_i (on(e,a) \rightarrow above(e,a))$
ανν $(\not\models_i on(a,a)$ ή $\models_i above(a,a))$ και ... και $(\not\models_i on(a,e)$ ή $\models_i above(a,e))$ και
$(\not\models_i on(b,a)$ ή $\models_i above(b,a))$ και ... και $(\not\models_i on(b,e)$ ή $\models_i above(b,e))$ και
...
$(\not\models_i on(e,a)$ ή $\models_i above(e,a))$ και ... και $(\not\models_i on(e,a)$ ή $\models_i above(e,a))$
ή ισοδύναμα
$(<a^i,a^i> \notin on^i$ ή $<a^i,a^i> \in above^i)$ και ... και $(<a^i,e^i> \notin on^i$ ή $<a^i,e^i> \in above^i)$ και
$(<b^i,a^i> \notin on^i$ ή $<b^i,a^i> \in above^i)$ και ... και $(<b^i,e^i> \notin on^i$ ή $<b^i,e^i> \in above^i)$ και
...

($<e^i,a^i> \notin on^i$ ή $<e^i,a^i> \in above^i$) και ... και ($<e^i,e^i> \notin on^i$ ή $<e^i,e^i> \in above^i$)
Δεδομένου ότι,

$on^i = \{<a^i,b^i>,<b^i,c^i>,<d^i,e^i>\}$, και
$above^i = \{<a^i,b^i>,<b^i,c^i>,<a^i,c^i>,<d^i,e^i>\}$,

η παραπάνω σχέση θα γίνει ισοδύναμα

(1 ή 0) και ... και (1 ή 0)
(1 ή 0) και ... και (1 ή 0)
...
(1 ή 0) και ... και (1 ή 0)

που τελικά παίρνει την τιμή 1 ('αληθές'). Δηλαδή η αρχική προταση είναι αληθής στην επιλεγμένη ερμηνεία.

Θα έχουμε παρατηρήσει ήδη ότι η μορφή στην οποία δίδονται τα στοιχεία της εννοιολογικής αποτύπωσης μοιάζουν με τα κατηγορήματα που δεν έχουν μεταβλητές. Τέτοια κατηγορήματα ονομάζονται **στοιχειώδη κατηγορήματα** (ground predicates). Οταν γιά παράδειγμα γράφουμε

$$on^i = \{<a^i,b^i>,<b^i,c^i>,<d^i,e^i>\}, \text{ και}$$
$$above^i = \{<a^i,b^i>,<b^i,c^i>,<a^i,c^i>,<d^i,e^i>\},$$

και εννοούμε στοιχεία της εννοιολογικής αποτύπωσης γιά το πρόβλημα των κύβων. Εαν από την άλλη μεριά βρούμε τα στοιχειώδη κατηγορήματα τα οποία ικανοποιούνται με βάση την αποτύπωση αυτή θα γράφαμε :

$$\{on(a,b),on(b,c),on(d,e)\}, \text{ και}$$
$$\{above(a,b), above(b,c), above(a,c), above(d,e)\},$$

Αν λάβουμε υπόψη μας αυτή την χρήσιμη παρατήρηση και μπορούσαμε να επινοήσουμε ένα τρόπο να εξάγουμε στοιχειώδη κατηγορήματα από ένα σύνολο προτάσεων, τότε θα μπορούσαμε να βρίσκουμε την υπονοούμενη από τον συγγραφέα των προτάσεων ερμηνεία.

Ορισμός 3.12 Ταυτολογία - Αντίφαση - Μοντέλο
Μιά πρόταση που είναι αληθής κάτω από όλες τις ερμηνείες λέγεται **ταυτολογία** (valid, tautology). Μιά πρόταση που είναι ψευδής κάτω από όλες τις ερμηνείες λέγεται **αντίφαση** ή **αντιλογία** ή **ασυνεπής** (inconsistent, unsatisfiable, contradiction). Μιά πρόταση είναι **ικανοποιήσιμη** (satisfiable) άν υπάρχει τουλάχιστον μιά ερμηνεία κάτω από την οποία είναι αληθής. Κάθε ερμηνεία που ικανοποιεί μια πρόταση φ (ένα σύνολο Σ), γιά όλες τις αναθέσεις μεταβλητών, λέγεται **μοντέλο** της φ (του συνόλου Σ).

3.5 Λογικό συμπέρασμα : Η αλήθεια σε ένα σύνολο προτάσεων

Ορισμός 3.13 Ικανοποίηση συνόλου προτάσεων
Μια ερμηνεία i ικανοποιεί ένα σύνολο Σ, αν ικανοποιεί κάθε πρόταση του Σ.

$$\models_i \Sigma \text{ ανν } (\forall \varphi \in \Sigma) \models_i \varphi$$

Ορισμός 3.14 Λογικό συμπέρασμα
Εστω μιά πρόταση φ και ένα σύνολο προτάσεων Σ. Η φ λέγεται λογικό συμπέρασμα του Σ, και συμβολίζεται σαν $\Sigma \models \varphi$, ανν κάθε μοντέλο του Σ είναι και μοντέλο της φ :

$$\Sigma \models \varphi \text{ iff } (\forall i)\,(\forall u)\,(\models_i \Sigma[u] \Rightarrow \models_i \varphi[u])$$

Ο τύπος αυτός εφαρμόζεται αν έχουμε ελεύθερες μεταβλητές στις προτάσεις του Σ. Εμείς όμως θα φροντίζουμε οι προτάσεις που γράφουμε να μην έχουν ελεύθερες μεταβλητές, αλλά όλες οι μεταβλητές που εμφανίζονται να είναι δεσμευμένες κάτω από ένα ποσοδείκτη. Με αυτό το δεδομένο μπορούμε ποιό απλά να γράψουμε :

$$\Sigma \models \varphi \text{ ανν } (\forall i)\,(\models_i \Sigma \Rightarrow \models_i \varphi)$$

Ορισμός 3.15 Λογική ισοδυναμία
Δύο προτάσεις φ,ψ είναι λογικά ισοδύναμες αν η μιά είναι λογικό συμπέρασμα της άλλης :

$$(\varphi \models\!\models \psi) \text{ ανν } (\varphi \models \psi) \text{ και } (\psi \models \varphi)$$

Συνήθεις λογικές ισοδυναμίες στη ΚΛ

Ισχύουν βέβαια οι λογικές ισοδυναμίες που εξετάστηκαν στην Προτασιακή Λογική. Εκτός όμως από αυτές ισχύουν και οι παρακάτω λογικές ισοδυναμίες που αφορούν τους ποσοδείκτες.

Ας σημειωθεί ότι ο συμβολισμός φ(X) εδώ σημαίνει ότι το κατηγόρημα φ περιέχει την μεταβλητή X, ενώ ο συμβολισμός φ σημαίνει ότι το κατηγόρημα δεν περιέχει σαν μεταβλητή την X.

$(\exists X)\,(\exists Y)\,\varphi(X,Y)$	$\models\!\models$	$(\exists Y)\,(\exists X)\,\varphi(X,Y)$
$(\forall X)\,(\forall Y)\,\varphi(X,Y)$	$\models\!\models$	$(\forall Y)\,(\forall X)\,\varphi(X,Y)$
$(\forall X)\,(\varphi(X) \wedge \psi(X))$	$\models\!\models$	$(\forall X)\,\varphi(X) \wedge (\forall X)\,\psi(X)$
$(\forall X)\,(\varphi(X) \vee \psi)$	$\models\!\models$	$(\forall X)\,\varphi(X) \vee \psi$
$(\exists X)\,(\varphi(X) \vee \psi(X))$	$\models\!\models$	$(\exists X)\,\varphi(X) \vee (\exists X)\,\psi(X)$
$(\exists X)\,(\varphi(X) \wedge \psi)$	$\models\!\models$	$(\exists X)\,\varphi(X) \wedge \psi$
$(\forall X)\,\neg\varphi(X)$	$\models\!\models$	$\neg\,(\exists X)\,\varphi(X)$
$(\exists X)\,\neg\varphi(X)$	$\models\!\models$	$\neg\,(\forall X)\,\varphi(X)$

Οι επόμενες σχέσεις δεν ισχύουν σαν λογικές ισοδυναμίες αλλά μόνο σαν λογικά συμπεράσματα (δηλαδή μόνο κατά τη μια φορά).

$(\exists X)\,(\varphi(X) \wedge \psi(X))$	$\models$	$(\exists X)\,\varphi(X) \wedge (\exists X)\,\psi(X)$
$(\exists X)\,(\forall Y)\,\varphi(X,Y)$	$\models$	$(\forall Y)\,(\exists X)\,\varphi(X,Y)$
$(\forall X)\,\varphi(X)$	$\models$	$(\exists X)\,\varphi(X)$
$\neg(\exists X)\,\varphi(X)$	$\models$	$\neg(\forall X)\,\varphi(X)$
$(\forall X)\,(\varphi(X) \vee \psi(X))$	$=\!\!\models$	$(\forall X)\,\varphi(X) \vee (\forall X)\,\psi(X)$
$(\forall X)\,(\exists Y)\,\varphi(X,Y)$	$=\!\!\models$	$(\exists Y)\,(\forall X)\,\varphi(X,Y)$

Μέθοδοι εύρεσης λογικού συμπεράσματος

Εστω ένα σύνολο Σ που αποτελείται από τις προτάσεις φ_1, ..., φ_n, δηλαδή, $\Sigma = \{\varphi_1, ..., \varphi_n\}$. Εφόσον το σύνολο Σ αποτελείται από αυτές τις προτάσεις είναι λογικά ισοδύναμο με την σύζευξή τους, δηλαδή $\Sigma \models_i (\varphi_1 \wedge ... \wedge \varphi_n)$. Με βάση αυτό μπορούμε να γράψουμε τις παρακάτω ισοδυναμίες :

$\Sigma \models \varphi$ ανν $(\forall i)\ (\models_i (\varphi_1 \wedge ... \wedge \varphi_n) \Rightarrow \models_i \varphi)$
$(\forall i)\ ((\forall j)\ \models_i \varphi_j \Rightarrow \models_i \varphi)$
$(\forall i)\ (\models_i (\varphi_1 \wedge ... \wedge \varphi_n \rightarrow \varphi))$ ταυτολογία
$(\forall i)\ (\not\models_i (\varphi_1 \wedge ... \wedge \varphi_n \wedge \neg\varphi)$ αντιλογία

Από τις ισοδυναμίες αυτές μπορούμε να βρούμε τις μεθόδους που μπορούμε να χρησιμοποιήσουμε γιά να εξετάσουμε αν μια πρόταση είναι λογικό συμπέρασμα ενός συνόλου προτάσεων. Οι μέθοδοι αυτοί είναι σε κάποιο βαθμό παρόμοιες με αυτές που χρησιμοποιήσαμε στη Προτασιακή Λογική.

Παράδειγμα 3.12 Να εξεταστεί αν η πρόταση $\varphi=(\exists X)(d(X) \wedge \neg u(X))$ είναι λογικό συμπέρασμα του συνόλου $\Sigma=\{(\forall X)(u(X)\rightarrow\neg b(X)), (\exists X)(d(X) \wedge b(X))\}$.

Με βάση τον ορισμό, αρκεί να αποδείξουμε ότι γιά όλες τις ερμηνείες γιά τις οποίες ικανοποιείται το Σ, για αυτές ικανοποιείται και η φ. Αρκεί δηλαδή να αποδείξουμε ότι $(\forall i)$ $(\models_i \Sigma \Rightarrow \models_i \varphi)$. Γιά μιά οποιαδήποτε ερμηνεία i, γιά την οποία ισχύει :

$\models_i (\forall X)(u(X)\rightarrow\neg b(X))$ και $\models_i (\exists X)(d(X) \wedge b(X))$
$\Rightarrow$ $\models_i (\forall X)(\neg u(X)\vee\neg b(X))$ και $\models_i (\exists X)(d(X) \wedge b(X))$
$\Rightarrow$ $\models_i (\neg u(c1)\vee\neg b(c1))$ και
$\models_i (\neg u(c2)\vee\neg b(c2))$ και
...
$\models_i (\neg u(cn)\vee\neg b(cn))$ και
$\models_i (d(cj) \wedge b(cj))$

Εφόσον έχουμε συζεύξεις πρέπει κάθε σύζευξη να αληθεύει στην τυχαία ερμηνεία i, άρα και οι συζεύξεις $\models_i (d(cj) \wedge b(cj))$ και $\models_i (\neg u(cj)\vee\neg b(cj))$. Από την πρώτη έπεται ότι τα κατηγορήματα d(cj) και b(cj) ικανοποιούνται και τα δύο από το Σ, οπότε το $\neg b(cj)$ έχει τιμή αλήθειας 0, και έτσι στην δεύτερη αναγκαστικά το $\neg u(cj)$ έχει τιμή αλήθειας 1. Αρα αληθεύει το $\models_i (d(cj) \wedge \neg u(cj))$. Αυτό το τελευταίο είναι ισοδύναμο με το $\models_i (\exists X)(d(X) \wedge \neg u(X))$

Στοιχειώδη κατηγορήματα που αληθεύουν σε ένα σύνολο προτάσεων

Οι συνθήκες ικανοποιησιμότητας αναλύουν μιά πρόταση σε όλο και περισσότερες απλές προτάσεις έως ότου να εμφανιστούν στοιχειώδη κατηγορήματα. Εφαρμόζοντας λοιπόν αυτές τις συνθήκες σε μιά πρόταση φτάνουμε σε ένα σύνολο συζεύξεων και διαζεύξεων που περιέχουν μόνο στοιχειώδη κατηγορήματα. Αυτή η τακτική μπορεί να εφαρμοστεί βέβαια και σε ένα σύνολο προτάσεων.

Παράδειγμα 3.13 Θεωρείστε το σύνολο Σ, που φαίνεται παρακάτω. Βρείτε τα στοιχειώδη κατηγορήματα που ικανοποιούνται από αυτό το σύνολο, δεδομένου ότι C={a,b,c, F={}, R={on, above,table,clear}.

Σ = { $(\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y))$,
$(\forall X)\ (\forall Y)\ (\forall Z)(above(X,Y) \wedge on(Y,Z) \rightarrow above(X,Z))$,
table(c), clear(a), on(a,b), on(b,c) }

Κατ'αρχήν υπάρχουν τα στοιχειώδη κατηγορήματα table(c), clear(a), on(a,b), on(b,c), που ήδη ικανοποιούνται από το Σ, αφού εμπεριέχονται στο Σ. Οσον αφορά στη πρώτη πρόταση, προχωρούμε με τον τρόπο που προχωρήσαμε και σε προηγούμενο παράδειγμα :

$\models_i$ ($(\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y))$) [u]

ανν $\models_i$($(\forall X)(\forall Y)(on(X,Y) \rightarrow above(X,Y))$)

ανν $\models_i$($(\forall Y)(on(a,Y) \rightarrow above(a,Y))$) και

$\models_i$($(\forall Y)(on(b,Y) \rightarrow above(b,Y))$) και

$\models_i$($(\forall Y)(on(c,Y) \rightarrow above(c,Y))$)

ανν

$\models_i(\neg on(a,a) \vee above(a,a)) \wedge \models_i(\neg on(a,b) \vee above(a,b)) \wedge \models_i (\neg on(a,c) \vee above(a,c)) \wedge$
$\models_i(\neg on(b,a) \vee above(b,a)) \wedge \models_i(\neg on(b,b) \vee above(b,b)) \wedge \models_i (\neg on(b,c) \vee above(b,c)) \wedge$
$\models_i(\neg on(c,a) \vee above(c,a)) \wedge \models_i(\neg on(c,b) \vee above(c,b)) \wedge \models_i (\neg on(c,c) \vee above(c,c))$

Από την υπόθεση του προβλήματος γνωρίζουμε ότι αληθεύουν τα :

¬on(a,a), on(a,b), ¬on(a,c),
¬on(b,a), ¬on(b,b), on(b,c),
¬on(c,a), ¬on(c,b), ¬on(c,c)

Οπότε γιά να αληθεύει η προηγούμενη συνθήκη ικανοποιησιμότητας, είναι **απαραίτητο** να αληθεύουν τα : *above(a,b), above(b,c)*. Με ίδιο τρόπο από την δεύτερη πρόταση του παραδείγματος εξάγεται και το στοιχειώδες κατηγόρημα : *above(a,c)*.

Τελικά τα στοιχειώδη θετικά και αρνητικά κατηγορήματα που ικανοποιούνται από το Σ είναι :

¬on(a,a), on(a,b), ¬on(a,c),
¬on(b,a), ¬on(b,b), on(b,c),
¬on(c,a), ¬on(c,b), ¬on(c,c)

¬above(a,a), above(a,b), above(a,c),
¬above(b,a), ¬above(b,b), above(b,c),
¬above(c,a), ¬above(c,b), ¬above(c,c)

¬table(a), ¬table(b), table(c)

clear(a), ¬clear(b), ¬clear(c)

3.6 Αναπαράσταση της γνώσης στη ΚΛ

Ολα αυτά που αναφέραμε ως τώρα δεν θα είχαν καμμία αξία για την Τεχνητή Νοημοσύνη και τον Λογικό Προγραμματισμό, αν δεν μπορούσαμε να πραγματοποιήσουμε αναπαράσταση της γνώσης στη ΚΛ. Βέβαια είδαμε κάποια πολύ απλά παραδείγματα αναπαράστασης της γνώσης μέσω των συνόλων προτάσεων που αναπαριστούσαν το κόσμο των κύβων. Στη συνέχεια θα δούμε μερικά άλλα παραδείγματα αναπαράστασης της γνώσης. Στα παραδείγματα αυτά παρατίθενται προτάσεις σε φυσική γλώσσα και οι αντίστοιχες προτάσεις (wffs) στη ΚΛ.

Παράδειγμα 3.13 Παραδείγματα μικρών προτάσεων

1. "Ο πρόγονος του προγόνου μου είναι πρόγονός μου"

$(\forall X)(\forall Y)(\forall Z) ($ πρόγονος$(X,Y) \wedge$ πρόγονος$(Y,Z) \rightarrow$ πρόγονος$(X,Z))$

2. "Ολοι οι άνθρωποι είναι θνητοί"
$(\forall X)$ (άνθρωπος(X) $\rightarrow$ θνητός(X))
3. "Κάθε μήλο είναι κόκκινο"
$(\forall X)$ [μήλο(X) $\rightarrow$ κόκκινο(X)]
4. "Υπάρχει ένα κόκκινο μήλο"
$(\exists X)$ [μήλο(X) $\wedge$ κόκκινο(X)]
5. "Κάτι είναι είτε μήλο είτε αχλάδι"
$((\exists X)$ μήλο(X)) $\vee$ $((\exists X)$ αχλάδι(X))
6. "Γιά κάθε άνθρωπο υπάρχει κάποιος που τον αγαπάει"
$(\forall X)$ $(\exists Y)$ αγαπάει(X,Y)
7. "Υπάρχει κάποιος που τον αγαπάνε όλοι"
$(\exists X)$ $(\forall Y)$ αγαπάει(Y,X)

Παράδειγμα 3.14 Ποιος δολοφόνησε τον Καίσαρα;
Το πρόβλημα σε φυσική γλώσσα είναι :
1. ο Μάρκος είναι άνθρωπος
2. ο Μάρκος είναι πομπήιος
3. όλοι οι πομπήιοι είναι ρωμαίοι
4. ο Καίσαρας είναι αυτοκράτορας
5. όλοι οι ρωμαίοι ή είναι πιστοί στον Καίσαρα ή τον μισούν
6. ο καθένας είναι πιστός σε κάποιον
7. όσοι προσπαθούν να δολοφονήσουν τον αυτοκράτορα δεν του είναι πιστοί
8. ο Μάρκος προσπάθησε να δολοφονήσει τον Καίσαρα.
9. Θέλουμε να αποδείξουμε ότι ο μάρκος δεν είναι πιστός στον Καίσαρα.

Οι προτάσεις στην γλώσσα της Κατηγορικής Λογικής είναι :
1. άνθρωπος(μάρκος)
2. πομπήιος(μάρκος)
3. $(\forall X)$[πομπήιος(X) $\rightarrow$ ρωμαίος(X)]
4. αυτοκράτορας(καίσαρας)
5. $(\forall X)$(ρωμαίος(X) $\rightarrow$ (πιστός(X,καίσαρας) $\vee$ μισεί(X,καίσαρας)))
6. $(\forall X)(\exists Y)$(πιστός(X,Y))
7. $(\forall X)(\exists Y)$[(άνθρωπος(X) $\wedge$ αυτοκράτορας(Y) $\wedge$ προσπάθειαγιαδολοφονία(X,Y) $\rightarrow$ $\neg$πιστός(X,Y)]
8. προσπάθεια για δολοφονία(μάρκος,καίσαρας)

Η προς απόδειξη πρόταση γίνεται :
9. $\neg$πιστός(μάρκος,καίσαρας).

Παράδειγμα 3.15 : Πρόβλημα συνόλων
1. $(\forall X)(\forall S)(\forall T)[X\in S\wedge X\in T\leftrightarrow X\in S\cap T]$

Υποθέτουμε ότι ανήκει(X,S) συμβολίζει την σχέση $X\in S$, υποσύνολο(S,T) για την σχέση $S\subseteq T$ και το συναρτησιακό τομή(S,T) για την πράξη $S\cap T$. Άρα
$(\forall X)(\forall S)(\forall T)$(ανήκει(X,S) $\wedge$ ανήκει(X,T) $\leftrightarrow$ ανήκει(X,τομή(S,T))).
2. $(\forall S)(\forall T)(((\forall X)(X\in S\rightarrow X\in T))\rightarrow S\subseteq T)$
$(\forall S)(\forall T)(((\forall X)$(ανήκει(X,S) $\rightarrow$ ανήκει(X,T)) $\rightarrow$ υποσύνολο(S,T))

Να αποδειχθεί ότι
3. $(\forall S)(\forall T)(S\cap T\subseteq S)$, δηλαδή
$(\forall S)(\forall T)$υποσύνολο(τομή(S,T),S).

3.7 Αλλες χρήσιμες μορφές προτάσεων στη ΚΛ

Εκτός από τις καλοσχηματισμένες προτάσεις υπάρχουν και άλλα είδη προτάσεων. Μερικά από αυτά είναι πολύ χρήσιμα, ειδικά γιά την περίπτωση της αυτοματοποιημένης συμπερασματολογίας. Θα ασχοληθούμε με τις προτάσεις σε μορφή clause ή φράσεις, οι οποίες θα αποτελέσουν και τις βασικές δομικές μονάδες των αυτοματοποιημένων αποδείξεων. Οι φράσεις είναι μια μορφή λογικών προτάσεων στην οποία έχουν απαλειφθεί όλοι οι ποσοδείκτες καθώς και όλα τα λογικά σύμβολα εκτός της άρνησης και της διάζευξης.

3.7.1 Φράσεις (Clauses)

Ορισμός 3.16 Λεκτικό - Φράση – Σύνολο φράσεων - Μοναδιαία φράση - Κενή φράση

Ενα **λεκτικό** είναι ένα θετικό ή αρνητικό άτομο. Το αρνητικό άτομο είναι ένα άτομο του οποίου προηγείται ο λογικός σύνδεσμος not (¬). Το θετικό άτομο είναι ένα άτομο.

π.χ. ¬a(X,S)

Μιά **φράση** είναι μιά πεπερασμένη ακολουθία από μηδέν ή περισσότερα λεκτικά σε διάζευξη.

π.χ. ¬a(X,S) ∨ ¬a(X,T) ∨ a(X, τ(S,T))

Ένα **φρασεοσύνολο** (σύνολο φράσεων-clause set) είναι ένα σύνολο από φράσεις, που θεωρούμε ότι συνδέονται μεταξύ τους με συζεύξεις.

Μιά φράση που αποτελείται από ένα λεκτικό λέγεται **μοναδιαία φράση** (unit clause).

Μιά φράση που δεν έχει καθόλου λεκτικά λέγεται **άδεια** ή **κενή φράση** (empty clause).

Κάτι που θα συναντήσουμε στο επόμενο κεφάλαιο είναι η αναπαράσταση μιάς φράσης, όχι σαν μιάς διάζευξης λεκτικών αλλά με μορφή συνόλου λεκτικών, μεταξύ των οποίων υπονοείται η διάζευξη.

Παράδειγμα 3.16

¬a(X,S) ∨ ¬a(X,T) ∨ a(X, τ(S,T)) = {¬a(X,S), ¬a(X,T), a(X, τ(S,T))}

Ορισμός 3.17 Φράσεις τύπου Horn (Horn clauses)
Μιά φράση τύπου Horn είναι μιά φράση που έχει ένα το πολύ θετικό λεκτικό.

Παράδειγμα 3.17 Οι παρακάτω είναι φράσεις τύπου Horn
¬a(X,S) ∨ ¬a(X,T) ∨ a(X, τ(S,T))
¬a(f(S,T),T) ∨ u(S,T)}
¬u(τ(a,b),a)}
Η πρόταση a(f(S,T),S) ∨ u(S,T) δεν είναι φράση τύπου Horn γιατί έχει δύο θετικά λεκτικά.

3.7.2 Διαδικασία μετατροπής προτάσεων από καλοσχηματισμένη μορφή σε μορφή φράσεων (wff to clausal form)

Η μετατροπή μιας καλοσχηματισμένης πρότασης σε μορφή φράσεων είναι εφικτή μέσω μιας διαδικασίας που θυμίζει σε μεγάλο βαθμό τη διαδικασία μετατροπής προτάσεων σε κανονική συζευκτική μορφή, όπως την έχουμε ήδη εξετάσει για την περίπτωση του Προτασιακού Λογισμού. Στην κατηγορηματική λογική, όμως, προστίθενται ορισμένοι επιπλέον κανόνες για το χειρισμό των μεταβλητών και των ποσοδεικτών. Η διαδικασία αποτελείται από οκτώ βήματα, τα οποία πρέπει να εκτελεστούν με τη σειρά που παρουσιάζονται στη συνέχεια:

Βήμα 1: Απαλοιφή συνεπαγωγών και ισοδυναμιών.

Στο βήμα αυτό απαλείφουμε όλες τις συνεπαγωγές και τις ισοδυναμίες που εμφανίζονται μέσα σε μια πρόταση, αντικαθιστώντας τις με ισοδύναμες προτάσεις που χρησιμοποιούν μόνο τα σύμβολα $\neg, \vee, \wedge$.

$\varphi_1 \rightarrow \varphi_2 : \neg\varphi_1 \vee \varphi_2$
$\varphi_1 \leftarrow \varphi_2 : \varphi_1 \vee \neg\varphi_2$
$\varphi_1 \leftrightarrow \varphi_2 : (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$

Βήμα 2:Μεταφορά αρνήσεων στο εσωτερικό της πρότασης

Στο βήμα αυτό οι αρνήσεις μεταφέρονται στο εσωτερικό της πρότασης με τη χρήση ισοδύναμων προτάσεων, έως ότου κάθε τελεστής άρνησης εφαρμόζεται σε μια μόνο ατομική πρόταση.

$\neg\neg\varphi : \varphi$
$\neg(\varphi_1 \wedge \varphi_2) : \neg\varphi_1 \vee \neg\varphi_2$
$\neg(\varphi_1 \vee \varphi_2) : \neg\varphi_1 \wedge \neg\varphi_2$
$\neg(\forall X)(\varphi(X)) : (\exists X)(\neg\varphi(X))$
$\neg(\exists X)(\varphi(X)) : (\forall X)(\neg\varphi(X))$

Βήμα 3:Τυποποίηση μεταβλητών

Στο βήμα αυτό μετονομάζουμε τις μεταβλητές έτσι ώστε κάθε ποσοδείκτης να εμπεριέχει μια και μοναδική μεταβλητή (δηλ. η ίδια μεταβλητή δεν βρίσκεται στην εμβέλεια δύο ποσοδεικτών μέσα στην ίδια πρόταση)

$(\forall X)(\varphi(X) \rightarrow (\exists X)(\psi(X))) : (\forall X)(\varphi(X) \rightarrow (\exists Y)(\psi(Y)))$

Βήμα 4:Απαλοιφή υπαρξιακών ποσοδεικτών

Στο βήμα αυτό απαλείφουμε όλους τους υπαρξιακούς ποσοδείκτες με την ακόλουθη μέθοδο:

Εάν ένας υπαρξιακός ποσοδείκτης βρίσκεται εκτός της εμβέλειας καθολικών ποσοδεικτών, τότε διαγράφουμε τον υπαρξιακό ποσοδείκτη και αντικαθιστούμε όλα τα στιγμιότυπα της μεταβλητής που βρισκόταν κάτω από την εμβέλεια του υπαρξιακού ποσοδείκτη με μια σταθερά η οποία δεν εμφανίζεται πουθενά αλλού μέσα στην πρόταση (ή τις προτάσεις) του συνόλου που εξετάζουμε.

$(\exists X)(\varphi(X)):\varphi(\alpha)$ *(με την προϋπόθεση ότι το α δεν εμφανίζεται σε καμία άλλη πρόταση του συνόλου μας)*

Στην περίπτωση που ο υπαρξιακός ποσοδείκτης βρίσκεται κάτω από την εμβέλεια ενός καθολικού ποσοδείκτη, υπάρχει η πιθανότητα εξάρτησης της «υπαρξιακής» μεταβλητής από άλλες μεταβλητές που επηρεάζονται από τους καθολικούς ποσοδείκτες, συνεπώς δεν μπορούμε να εφαρμόσουμε αβίαστα τον προηγούμενο κανόνα. Αντί γι'αυτό, απαλείφουμε τον υπαρξιακό ποσοδείκτη και αντικαθιστούμε τη μεταβλητή με ένα νέο συναρτησιακό που εφαρμόζεται πάνω στις καθολικά εξαρτώμενες μεταβλητές. Όπως και προηγουμένως, το συναρτησιακό σύμβολο δεν πρέπει να έχει χρησιμοποιηθεί προηγουμένως στην ίδια ή σε άλλη πρόταση του συνόλου μας.

$$(\forall X)(p(X)\wedge(\exists Z)(q(X,Y,Z)):(\forall X)(p(X)\wedge q(X,Y,f(X,Y)))$$

Η διαδικασία που περιγράφεται σε αυτό το βήμα είναι γνωστή και ως Σκολεμοποίηση (Skolemization), από το όνομα του Νορβηγού μαθηματικού A.T. Skolem, που εισήγαγε τη μέθοδο το 1920.

Βήμα 5:Απαλοιφή καθολικών ποσοδεικτών

Στο βήμα αυτό απαλείφονται όλοι οι καθολικοί ποσοδείκτες. Καθώς από το προηγούμενο βήμα έχουμε ήδη απαλλαγεί από τους υπαρξιακούς ποσοδείκτες, τώρα είναι δυνατή η διαγραφή των καθολικών ποσοδεικτών χωρίς να δημιουργούνται ασυνέπειες.

$$(\forall X)(p(X)\wedge q(X,Y,f(X,Y))): p(X)\wedge q(X,Y,f(X,Y))$$

Βήμα 6:Μεταφορά των διαζεύξεων στο εσωτερικό της πρότασης

Στο βήμα αυτό μεταφέρουμε όλες τις διαζεύξεις στο εσωτερικό της πρότασης, έως ότου φτάσουμε σε κανονική συζευκτική μορφή, δηλαδή σε μια σύζευξη διαζευγμένων λεκτικών, με τη χρήση των κανόνων επιμερισμού:

$\varphi_1\vee(\varphi_2\wedge\varphi_3):(\varphi_1\vee\varphi_2)\wedge(\varphi_2\vee\varphi_3)$
$(\varphi_1\wedge\varphi_2)\vee \varphi_3:(\varphi_1\vee\varphi_3)\wedge(\varphi_2\vee\varphi_3)$
$(\varphi_1\wedge\varphi_2)\wedge \varphi_3: \varphi_1\wedge(\varphi_2\wedge\varphi_3)$
$(\varphi_1\vee\varphi_2)\vee \varphi_3: \varphi_1\vee(\varphi_2\vee\varphi_3)$

Βήμα 7:Χωρισμός σε φράσεις

Στο βήμα αυτό εκφράζουμε την πρόταση σαν ένα σύνολο φράσεων, διαχωρίζοντας τα συζευγμένα μέρη μιας σύζευξης σε διαφορετικές φράσεις που περιέχουν μόνο διαζεύξεις και αρνήσεις.

$\varphi_1\wedge\varphi_2\wedge\varphi_3:\{\varphi_1\}, \{\varphi_2\}, \{\varphi_3\}$
$\varphi_1\vee\varphi_2\vee\varphi_3:\{\varphi_1\vee\varphi_2\vee\varphi_3\}$
$(\varphi_1\vee\varphi_2)\wedge(\varphi_3\vee\varphi_4):\{\varphi_1\vee\varphi_2\}, \{\varphi_3\vee\varphi_4\}$

Βήμα 8:Μετονομασία μεταβλητών

Στο βήμα αυτό μετονομάζουμε τις μεταβλητές έτσι ώστε καμία μεταβλητή να μην εμφανίζεται σε περισσότερες από μια φράσεις.

p(X):{p(X)}
q(X):{q(Y)}

Παράδειγμα 3.18-Μετατροπη πρότασης από wff σε clause set

	$(\exists Y)(g(Y) \wedge (\forall Z)(r(Z) \rightarrow f(Y,Z)))$
Βήμα 1	$(\exists Y)(g(Y) \wedge (\forall Z)(\neg r(Z) \vee f(Y, Z)))$
Βήμα 2	$(\exists Y)(g(Y) \wedge (\forall Z)(\neg r(Z) \vee f(Y, Z)))$
Βήμα 3	$(\exists Y)(g(Y) \wedge (\forall Z)(\neg r(Z) \vee f(Y, Z)))$
Βήμα 4	$g(a) \wedge (\forall Z)(\neg r(Z) \vee f(a,Z)))$
Βήμα 5	$g(a) \wedge (\neg r(z) \vee f(a,Z)))$
Βήμα 6	$g(a) \wedge (\neg r(z) \vee f(a,Z)))$
Βήμα 7	$\{g(a)\},\{\neg r(Z) \vee f(a, Z)\}$
Βήμα 8	$\{g(a)\},\{\neg r(Z) \vee f(a, Z)\}$

4

Η αρχή της απόφασης

4.1 Εισαγωγή

Στα προηγούμενα κεφάλαια, εξετάσαμε θέματα που σχετίζονται με το συντακτικό, τη σημασιολογία προτάσεων της ΚΛ, καθώς και με την αναπαράσταση γνώσης στη ΚΛ. Χρησιμοποιήσαμε το λογικό συμπέρασμα για να εξετάζουμε τη τιμή αλήθειας μιάς πρότασης με βάση ένα σύνολο προτάσεων. Είδαμε πως ο αριθμός των δυνατών ερμηνειών είναι πολύ μεγάλος στη ΚΛ, με αποτέλεσμα να μην υπάρχουν ικανοποιητικές μέθοδοι που να μπορούν να εξετάσουν όλες τις δυνατές ερμηνείες, να βρουν τα μοντέλα των συνόλων και σ'αυτά να αποδείξουν την αλήθεια των προτάσεων.

Το ίδιο πρόβλημα που είχαμε στη ΠΛ το έχουμε, και μάλιστα επαυξημένο, στη ΚΛ. Εχοντας υποψη αυτά τα προβλήματα, και με σκοπό την εκμάθηση και χρησιμοποίηση ενός συστήματος απόδειξης γιά την ΚΛ, παρουσιάζουμε σε αυτό το κεφάλαιο την αρχή της απόφασης (resolution principle), η ύπαρξη της οποίας οφείλεται στον Robinson (1965). Αυτή η, όπως θα δούμε, απλή αρχή αποτέλεσε το θεμελιακό λιθάρι γιά την ανάπτυξη του Λογικού Προγραμματισμού.

Η αρχή της απόφασης μπορεί να εφαρμοστεί απευθείας σε οποιοδήποτε σύνολο φράσεων για να ελέγξει την μη ικανοποιησιμότητα τους. Η βασική ιδέα είναι να μετατρέπουμε ένα σύνολο προτάσεων Σ σε ένα σύνολο φράσεων Σ', να παίρνουμε την άρνηση της πρότασης φ που θέλουμε να αποδείξουμε, να την μετατρέπουμε και αυτή σε ένα σύνολο φράσεων φ' που ισοδυναμούν με την $\{\neg\varphi\}$, και στη συνέχεια να εφαρμόζουμε την απόδειξη με βάση την αρχή της απόφασης.

Η βασική ιδέα της απόδειξης με την αρχή της απόφασης είναι να ελέγχει αν η απόδειξη καταλήγει κάποια στιγμή στην κενή φράση που συμβολίζουμε με το σύμβολο □. Εάν συμβαίνει αυτό, τότε έχουμε φτάσει σε άτοπο, το σύνολο είναι μη ικανοποιήσιμο. Αυτό σημαίνει ότι $\Sigma' \cup \{\neg\varphi\}$ είναι αντιλογία, οπότε φτάσαμε σε άτοπο. Ετσι η $\neg\varphi$ δεν αληθεύει, άρα αληθεύει η φ. Αν βέβαια δεν παράγουμε την άδεια φράση, τίποτα δεν μπορούμε να συμπεράνουμε. Σε μερικές περιπτώσεις η αποδεικτική διαδικασία δεν εξαντλείται, δηλαδή δεν μπορούμε να πούμε ότι τελειώνει με αποτυχία, γιατί υπάρχουν άπειρες εναλλακτικές περιπτώσεις.

Εχουμε ήδη ασχοληθεί με την αρχή της απόφασης στη Προτασιακή Λογική. Σε αυτό το κεφάλαιο, θα την εξετάσουμε σε μεγαλύτερη έκταση, πρώτα για την προτασιακή λογική και μετά θα την επεκτείνουμε και στην κατηγορική λογική πρώτης τάξεως.

4.2 Η αρχή της απόφασης στη Προτασιακή Λογική

Η αρχή της απόφασης είναι βασικά μία προέκταση του κανόνα ενός λεκτικού των Davis και Putnam. Θεωρείστε τώρα τις παρακάτω φράσεις:

C_1 : p
C_2 : $\neg p \vee q$

Χρησιμοποιώντας τον κανόνα ενός λεκτικού, από τις C_1 και C_2 παίρνουμε τη φράση

C_3 : q

Αυτό που απαιτεί ο κανόνας ενός λεκτικού είναι να εξετάσουμε πρώτα αν υπάρχει το συμπληρωματικό ζεύγος ενός λεκτικού (για παράδειγμα, p) στη C_1 και ενός λεκτικού (για παράδειγμα, $\neg$p) στη C_2, και μετά να αφαιρέσουμε αυτό το ζεύγος από τη C_1 και τη C_2 για να πάρουμε την C_3 η οποία είναι η q.

Επεκτείνοντας τον παραπάνω κανόνα και εφαρμόζοντάς τον σε οποιοδήποτε ζεύγος φράσεων (όχι απαραίτητα μοναδιαίων φράσεων), προκύπτει ο παρακάτω κανόνας, ο οποίος καλείται *η αρχή της απόφασης* :

Ορισμός 4.1 Αρχή της απόφασης

Για οποιεσδήποτε δύο φράσεις C_1 και C_2, αν υπάρχει ένα λεκτικό L_1 στη C_1 το οποίο είναι συμπληρωματικό προς ένα λεκτικό L_2 στη C_2, τότε αφαιρούμε τα L_1 και L_2 από τις C_1 και C_2 αντίστοιχα, και δημιουργούμε τη διάζευξη των φράσεων που απέμειναν. Η παραγόμενη φράση είναι το αποφαινόμενο των C_1 και C_2.

Παράδειγμα 4.1 Θεωρείστε τις παρακάτω προτάσεις:

C_1 : $p \vee r$
C_2 : $\neg p \vee q$

Η φράση C_1 έχει το λεκτικό p, το οποίο είναι συμπληρωματικό του $\neg$p της C_2. Επομένως διαγράφοντας τα p και $\neg$p από τις C_1 και C_2 αντίστοιχα, και κατασκευάζοντας την διάζευξη των φράσεων που έχουν απομείνει, r και q, παίρνουμε το αποφαινόμενο $r \vee q$.

Παράδειγμα 4.2 Θεωρείστε τις παρακάτω προτάσεις:

C_1 : $\neg p \vee q \vee r$
C_2 : $\neg q \vee s$

Το αποφαινόμενο των C_1 και C_2 είναι το $\neg p \vee r \vee s$.

Παράδειγμα 4.3 Θεωρείστε τις παρακάτω προτάσεις:

C_1 : $\neg p \vee q$
C_2 : $\neg p \vee r$

Εφόσον δεν υπάρχει λεκτικό στη C_1 που να είναι συμπληρωματικό προς κάποιο λεκτικό στη C_2, τότε δεν υπάρχει αποφαινόμενο των C_1 και C_2.

Μια σημαντική ιδιότητα ενός αποφαινόμενου είναι ότι οποιοδήποτε αποφαινόμενο δύο φράσεων C_1 και C_2 είναι λογικό συμπέρασμα των C_1 και C_2. Αυτό διατυπώνεται και στο παρακάτω θεώρημα.

Θεώρημα 4.1 Δοθέντων δύο φράσεων C_1 και C_2, το αποφαινόμενο C των C_1 και C_2 είναι λογικό συμπέρασμα των C_1 και C_2.

Αν είχαμε δύο μοναδιαίες φράσεις, τότε το αποφαινόμενό τους, αν υπάρχει αυτό, είναι η κενή φράση □. Αν ένα σύνολο φράσεων S είναι μη ικανοποιήσιμο, μπορούμε να χρησιμοποιήσουμε την αρχή της απόφασης για να παράγουμε την □ από το S. Αυτό το αποτέλεσμα θα παρουσιαστεί σαν θεώρημα στην παράγραφο 4.6 αυτού του κεφαλαίου. Στο μεταξύ, θα ορίσουμε την έννοια της παραγωγής.

Ορισμός 4.2 -Παραγωγή
Δοθέντος ενός συνόλου φράσεων S, η **παραγωγή** του C από το S είναι μια πεπερασμένη ακολουθία $C_1, C_2, \ldots, C_k$ φράσεων τέτοιες ώστε κάθε C_i είτε είναι μια φράση του S είτε ένα αποφαινόμενο φράσεων που προηγούνται του C_i, και $C_k = C$. Η παραγωγή της □ από το S καλείται απαγωγή σε άτοπο ή απόδειξη του S.

Λέμε ότι μια φράση C μπορεί να *παραχθεί* ή να *προέλθει* από το S, αν υπάρχει μια παραγωγή του C από το S. Θα παρουσιάσουμε τώρα αρκετά παραδείγματα για να δείξουμε πως η αρχή της απόφασης μπορεί να χρησιμοποιηθεί για να αποδείξουμε την μη ικανοποιησιμότητα ενός συνόλου φράσεων.

Παράδειγμα 4.4 Θεωρείστε το σύνολο $S=\{\neg p \vee q, \neg q, p\}$. Να αποδειχθεί ότι το S είναι μη ικανοποιήσιμο σύνολο προτάσεων. Εχουμε τις προτάσεις :

(1) $p \vee q$
(2) q
(3) p

Από (1) και (2) παίρνουμε το αποφαινόμενο

(4) $\neg p$

Από (4) και (3) παίρνουμε το αποφαινόμενο

(5) □

Αφού παράχθηκε η □ από το S με την αρχή της απόφασης, σύμφωνα με το Θεώρημα 4.1 η □ είναι μια λογική συνέπεια του S. Ωστόσο, η □ μπορεί να είναι μόνο μια λογική συνέπεια ενός μη ικανοποιήσιμου συνόλου φράσεων. Επομένως, το S είναι μη ικανοποιήσιμο.

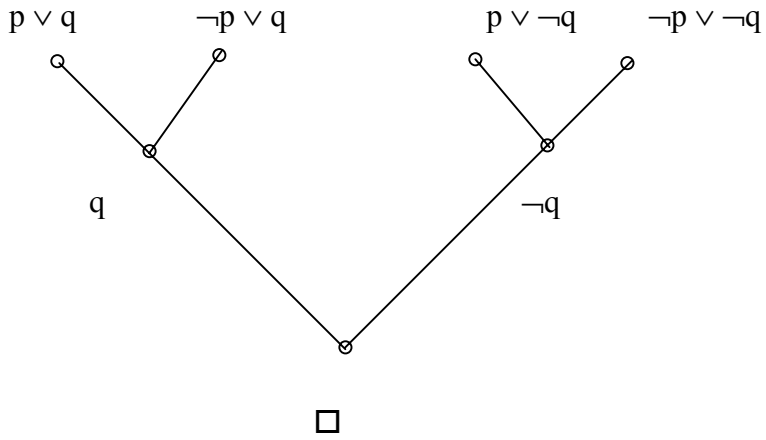
Παράδειγμα 4.5 Για το σύνολο $S=\{p \vee q, \neg p \vee q, p \vee \neg q, \neg p \vee \neg q\}$.
Να αποδειχθεί ότι το S είναι μη ικανοποιήσιμο σύνολο προτάσεων.

(1) $p \vee q$
(2) $\neg p \vee q$
(3) $p \vee \neg q$
(4) $\neg p \vee \neg q$

παράγουμε τα παρακάτω αποφαινόμενα

(5) q από (1) και (2)
(6) $\neg q$ από (3) και (4)
(7) □ από (5) και (6)

Η παραγωγή της κενής πρότασης μπορεί να αναπαρασταθεί και σχηματικά :



Σχήμα 4.1 Παραγωγή της κενής πρότασης με την αρχή της αποφασης

Εφόσον παράχθηκε η □, το S είναι μη ικανοποιήσιμο. Η παραπάνω παραγωγή μπορεί να αναπαραστηθεί από το δένδρο στο Σχήμα 4.1, το οποίο καλείται **δένδρο παραγωγής**.

Η αρχή της απόφασης είναι ένας πολύ ισχυρός συμπερασματικός κανόνας. Στη συνέχεια, θα ορίσουμε αυτή την αρχή στο πλαίσιο της κατηγορικής λογικής πρώτης τάξεως. Η αρχή της απόφασης είναι *πλήρης* στο να αποδεικνύει την μη ικανοποιησιμότητα ενός συνόλου φράσεων. Αυτό σημαίνει, ότι δοθέντος ενός συνόλου φράσεων S, το S είναι μη ικανοποιήσιμο αν και μόνο αν υπάρχει μία παραγωγή, βασισμένη στην αρχή της απόφασης, της κενής φράσης □ από το S. Στην παράγραφο 4.7 θα εφαρμόσουμε την αρχή της απόφασης σε αρκετά παραδείγματα έτσι ώστε ο αναγνώστης να εκτιμήσει την χρησιμότητα αυτού του συμπερασματικού κανόνα.

4.3 Αντικατάσταση και ενοποίηση

Στην προηγούμενη παράγραφο επισημάναμε ότι το πιο σημαντικό τμήμα της εφαρμογής της αρχής της απόφασης είναι η εύρεση ενός λεκτικού σε μια φράση το οποίο να είναι συμπληρωματικό προς ένα λεκτικό μιας άλλης φράσης. Για τις φράσεις που δεν περιλαμβάνουν μεταβλητές, αυτό είναι πολύ απλό. Ωστόσο, για τις φράσεις που

περιλαμβάνουν μεταβλητές, είναι περισσότερο πολύπλοκο. Για παράδειγμα, θεωρείστε τις φράσεις:

C_1 : $p(Y) \vee q(Y)$
C_2 : $\neg p(f(X)) \vee r(X)$

Δεν υπάρχει λεκτικό στη C_1 που να είναι συμπληρωματικό προς ένα λεκτικό στη C_2. Ωστόσο, εάν αντικαταστήσουμε το Y με f(α) στη C_1 και το X με α στη C_2 παίρνουμε

C_1' : $p(f(\alpha)) \vee q(f(\alpha))$
C_2' : $\neg p(f(\alpha)) \vee r(\alpha)$

Γνωρίζουμε ότι τα C_1' και C_2' είναι αρχικοποιημένα στιγμιότυπα των C_1 και C_2, αντίστοιχα, και τα $p(f(\alpha))$ και $\neg p(f(\alpha))$ είναι συμπληρωματικά μεταξύ τους. Επομένως, από τα C_1' και C_2' μπορούμε να πάρουμε το αποφαινόμενο

C_3' : $q(f(\alpha)) \vee r(\alpha)$

Γενικότερα, εάν αντικαταστήσουμε το Y με f(X) στη C_1, παίρνουμε

C_1^* : $p(f(X)) \vee q(f(X))$

Πάλι, το C_1^* είναι ένα στιγμιότυπο της C_1. Αυτή την φορά το λεκτικό $p(f(X))$ στη C_1 είναι συμπληρωματικό προς το λεκτικό $\neg p(f(X))$ στη C_2. Επομένως, μπορούμε να πάρουμε από τις C_1^* και C_2 το αποφαινόμενο,

C_3 : $q(f(X)) \vee r(X)$

Η C_3' είναι ένα **στιγμιότυπο** της φράσης C_3. Αντικαθιστώντας τις μεταβλητές στις C_1 και C_2 με τους κατάλληλους όρους όπως παραπάνω, μπορούμε να δημιουργήσουμε νέες φράσεις από τις C_1 και C_2. Επιπλέον, η φράση C_3 είναι η **πιο γενική φράση** με την έννοια ότι όλες οι άλλες φράσεις που μπορούν να δημιουργηθούν με την παραπάνω διαδικασία είναι στιγμιότυπα της C_3. Η C_3 θα καλείται επίσης **αποφαινόμενο** (resolvent) των C_1 και C_2. Στη συνέχεια, θα πρέπει να σκεφτούμε πως να παράγουμε αποφαινόμενα από φράσεις (που πιθανόν περιέχουν μεταβλητές). Εφόσον η παραγωγή των αποφαινόμενων από φράσεις απαιτεί συχνά αντικατάσταση των μεταβλητών, δίνουμε τους παρακάτω ορισμούς.

Ορισμός 4.3 - Αντικατάσταση
Η **αντικατάσταση** (substitution) είναι ένα πεπερασμένο σύνολο της μορφής $\{t_1/v_1, ..., t_n/v_n\}$, όπου κάθε v_i είναι μια μεταβλητή, κάθε t_i είναι ένας όρος διαφορετικός από την v_i, και δεν υπάρχουν δύο στοιχεία στο που να έχουν την ίδια μεταβλητή μετά το σύμβολο '/'. Οταν τα t_1, ... , t_n είναι αρχικοποιημένοι όροι, η αντικατάσταση λέγεται **αρχικοποιημένη αντικατάσταση** (ground substitution). Η αντικατάσταση η οποία δεν περιέχει στοιχεία καλείται **κενή αντικατάσταση** (empty substitution) και δηλώνεται με το γράμμα ε. Θα χρησιμοποιήσουμε Ελληνικά γράμματα για να αναπαραστήσουμε τις αντικαταστάσεις.

Παράδειγμα 4.6 Τα παρακάτω δύο σύνολα είναι αντικαταστάσεις:

$\{f(Z)/X, Y/Z\}$ $\{\alpha/X, g(Y)/Y, f(g(b))/Z\}$

Ορισμός 4.4 Θεωρείστε ότι το $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ είναι μια αντικατάσταση και Ε είναι μια έκφραση. Τότε το Εθ είναι μια έκφραση που δημιουργείται από την Ε αντικαθιστώντας ταυτόχρονα κάθε εμφάνιση της μεταβλητής v_i, $1 \leq i \leq n$, στην Ε με τον όρο t_i. Η Εθ καλείται ένα **στιγμιότυπο της Ε**. (Επισημαίνουμε ότι ο ορισμός του στιγμιότυπου είναι συμβατός με αυτόν του αρχικοποιημένου στιγμιότυπου μιας φράσης, που ορίστηκε στο Κεφάλαιο 4.)

Παράδειγμα 4.7 Θεωρείστε $\theta = \{\alpha/X, f(b)/Y, c/Z\}$ και $E = p(X,Y,Z)$. Τότε $E\theta = p(\alpha, f(b), c)$.

Ορισμός 4.5 Θεωρείστε ότι το $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$ και το $\lambda = \{u_1/y_1, \ldots, u_m/y_m\}$ είναι δύο αντικαταστάσεις. Τότε η **σύνθεση** του θ και του λ είναι η αντικατάσταση, που δηλώνεται με $\theta\square\lambda$, και η οποία παράγεται από το σύνολο

$$\{t_1\lambda/x_1, \ldots, t_n\lambda/x_n, u_1/y_1, \ldots, u_m/y_m\}$$

διαγράφοντας κάθε στοιχείο $t_j\lambda/x_j$ για το οποίο $t_j\lambda = x_j$ και κάθε στοιχείο u_i/y_i για το οποίο το y_i περιέχεται ανάμεσα στα $\{x_1, x_2, \ldots, x_n\}$.

Παράδειγμα 4.8 Θεωρείστε τα

$$\theta = \{t_1/x_1, t_2/x_2\} = \{f(Y)/X, Z/Y\}$$

$$\lambda = \{u_1/y_1, u_2/y_2, u_3/y_3\} = \{\alpha/X, b/Y, Y/Z\}$$

Τότε έχουμε

$$\{t_1\lambda/x_1, t_2\lambda/x_2, u_1/y_1, u_2/y_2, u_3/y_3\} = \{f(b)/X, Y/Y, \alpha/X, b/Y, Y/Z\}$$

Ωστόσο, αφού $t_2\lambda = x_2$, $t_2\lambda/x_2$, δηλαδή, Y/Y, πρέπει να διαγραφεί από το σύνολο. Επιπροσθέτως, αφού τα y_1 και y_2 περιέχονται στα $\{x_1, x_2, x_3\}$, τα u_1/y_1 και u_2/y_2, τα οποία είναι τα α/X και b/Y, πρέπει να διαγραφούν. Επομένως παίρνουμε

$$\theta\square\lambda = \{f(b)/X, Y/Z\}$$

Θα πρέπει να σημειωθεί ότι η σύνθεση των αντικαταστάσεων είναι μεταβατική, και ότι η κενή αντικατάσταση ε είναι και δεξιά και αριστερή ταυτότητα. Αυτό σημαίνει ότι $(\theta\square\lambda)\square\mu = \theta\square(\lambda\square\mu)$ και $\varepsilon\square\theta = \theta\square\varepsilon$ για κάθε θ, λ και μ.

Στην διαδικασία απόδειξης με βάση την αρχή της απόφασης, για να αναγνωρίσουμε ένα συμπληρωματικό ζεύγος λεκτικών, πολύ συχνά πρέπει να ενώσουμε (ταιριάξουμε) δύο ή περισσότερες εκφράσεις. Αυτό σημαίνει ότι πρέπει να βρούμε μια αντικατάσταση η οποία να μπορεί να κάνει διάφορες εκφράσεις ταυτόσημες. Επομένως, πρέπει τώρα να δώσουμε τον ορισμό για την ενοποίηση των εκφράσεων.

Ορισμός 4.6 Η αντικατάσταση θ καλείται **ενοποιητής** (unifier) για το σύνολο $\{E_1, \ldots, E_\kappa\}$ αν και μόνο αν $E_1\theta = E_2\theta = \ldots = E_\kappa\theta$. Το σύνολο $\{E_1, \ldots, E_\kappa\}$ είναι **ενοποιήσιμο** (unifiable) αν υπάρχει ενοποιητής για αυτό.

Ορισμός 4.7 Ενας ενοποιητής σ για το σύνολο εκφράσεων $\{E_1, \ldots, E_\kappa\}$ είναι **ο πιο γενικός ενοποιητής** (most general unifier) αν και μόνο αν για κάθε ενοποιητή θ του συνόλου υπάρχει η αντικατάσταση λ τέτοια ώστε $\theta = \sigma\square\lambda$.

Παράδειγμα 4.9 Το σύνολο {p(α,Y), p(X,f(b))} είναι ενοποιήσιμο αφού η αντικατάσταση θ = {α/X, f(b)/Y} είναι ένας ενοποιητής του συνόλου.

4.4 Αλγόριθμος ενοποίησης

Σε αυτή την παράγραφο, θα παρουσιάσουμε έναν αλγόριθμο ενοποίησης για την εύρεση του πιο γενικού ενοποιητή για ένα πεπερασμένο ενοποιήσιμο σύνολο μη κενών εκφράσεων. Οταν το σύνολο δεν είναι ενοποιήσιμο, ο αλγόριθμος θα εντοπίσει αυτό το γεγονός.

Θεωρείστε τις p(α) και p(X). Αυτές οι δύο εκφράσεις δεν είναι ταυτόσημες: Η διαφορά είναι ότι το α εμφανίζεται στο p(α) αλλά το X εμφανίζεται στο p(X). Για να ενοποιήσουμε τις p(α) και p(X), πρέπει πρώτα να βρούμε τη διαφορά τους, και μετά να προσπαθήσουμε να την εξαλείψουμε. Για τις p(α) και p(X), η διαφορά τους είναι η {α,X}. Αφού το X είναι μεταβλητή, μπορεί να αντικατασταθεί από το α, και επομένως η διαφορά θα εξαλειφθεί. Πάνω σε αυτή την ιδέα βασίζεται ο αλγόριθμος ενοποίησης.

Ορισμός 4.8 Το **σύνολο διαφωνίας** (disagreement set) ενός μη κενού συνόλου εκφράσεων W, κατασκευάζεται ως εξής:
Βήμα 1ο. Ελέγχουμε από αριστερά πρός τα δεξιά έναν-έναν τους όρους των εκφράσεων του W, έως ότου εντοπίσουμε κάποια θέση στην οποία οι αντίστοιχοι όροι έχουν διαφορετικά σύμβολα ("διαφωνία").
Βήμα 2ο. Μετά αποσπούμε από κάθε έκφραση του W την υπο-έκφραση, η οποία ξεκινάει με το σύμβολο που καταλαμβάνει αυτή την θέση.
Βήμα 3ο. Στην συνέχεια συγκεντρώνουμε τις υποεκφράσεις στις οποίες υφίσταται αυτή η "διαφωνία" και τις κάνουμε στοιχεία του συνόλου διαφωνίας.

Παράδειγμα 4.10 Αν W είναι το {p(X,f(Y,Z)), p(X,α), p(X,g(h(k(X))))}, τότε το η πρώτη θέση συμβόλου στην οποία δεν είναι όλα τα στοιχεία του W ίδια, είναι η πέμπτη, αφού όλα έχουν τα τέσσερα πρώτα σύμβολα τους p(X, κοινά. Επομένως, το σύνολο διαφωνίας αποτελείται από τις αντίστοιχες υπο-εκφράσεις (οι όροι που είναι υπογραμμισμένοι) οι οποίες ξεκινούν από την πέμπτη θέση, και το οποίο είναι το σύνολο {f(Y,Z), α, g(h(k(X)))}

Ο Αλγόριθμος ενοποίησης

Η βασική ιδέα πίσω από τον αλγόριθμο ενοποίησης (Unification Algorithm) είναι η ακόλουθη: Θεωρείστε δύο εκφράσεις τις οποίες θέλουμε να ενοποιήσουμε, και δύο δείκτες οι οποίοι μετακινούνται πάνω στις εκφράσεις. Οι δείκτες τοποθετούνται αρχικά στο αριστερότερο σημείο των εκφράσεων και μετακινούνται προς τα δεξιά τυατόχρονα, έως ότου να δείχνουν σε δύο διαφορετικά σύμβολα. Γίνεται μια προσπάθεια να ενοποιηθούν οι υποεκφράσεις που ξεκινούν με αυτά τα σύμβολα, οπότε παράγεται ένα σύνολο αντικαταστάσεων το οποίο εφαρμόζεται και στις δύο εκφράσεις, και οι δείκτες μετακινούνται ταυτόχρονα προς τα δεξιά. Ο αλγόριθμος τερματίζει με επιτυχία όταν οι δείκτες φτάσουν (ταυτόχρονα) στο τέλος των εκφράσεων, και τερματίζει με αποτυχία αν υπάρξουν δύο υποεκφράσεις οι οποίες δεν ενοποιούνται.

Αλγόριθμος Ενοποίησης (α' έκδοση)

Βήμα 1 Θέτουμε $k = 0$, $W_k = W$, και $\sigma_k = \varepsilon$.
Βήμα 2 Αν το W_k δεν είναι μονοσύνολο, τότε σταμάτα. Ο σ_k είναι ο πιο γενικός ενοποιητής του W. Διαφορετικά, βρες το σύνολο διαφωνίας D_k του W_k.
Βήμα 3 Αν υπάρχουν στοιχεία v_k και t_k στο D_k τέτοια ώστε το v_k είναι μεταβλητή η οποία να μην εμφανίζεται στο t_k, πήγαινε στο Βήμα 4. Διαφορετικά, σταμάτα. Το W είναι ενοποιήσιμο.
Βήμα 4 Θεωρούμε $\sigma_{k+1} = \sigma_k \square \{t_k/v_k\}$ και $W_{k+1} = W_k\{t_k/v_k\}$. (Προσέξτε ότι $W_{k+1} = W\sigma_{k+1}$)
Βήμα 5 Θέτουμε $k = k+1$ και πηγαίνουμε στο Βήμα 2.

Παράδειγμα 4.11 Βρείτε τον πιο γενικό ενοποιητή για το
$W=\{p(\alpha, X, f(g(Y))), p(Z, f(Z), f(U))\}$

1. $\sigma_o=\varepsilon$ και $W_0 = W$. Αφού το W_0 δεν είναι μονοσύνολο, το σ_o δεν είναι ο πιο γενικός ενοποιητής για το W.
2. Το σύνολο διαφωνίας $D_0 = \{\alpha,Z\}$. Στο D_0 υπάρχει η μεταβλητή $v_o = Z$ η οποία δεν εμφανίζεται στο $t_o = \alpha$.
3. Εχουμε
$\sigma_1 = \sigma_o \square \{t_o / v_o\} = \varepsilon \square \{\alpha/ Z\}$,
$W_1 = W_0 \{t_o / v_o\}$
$= \{p(\alpha, X, f(g(Y))), p(Z, f(Z), f(U))\} \{\alpha/ Z\}$
$= \{p(\alpha, X, f(g(Y))), p(\alpha, f(\alpha), f(U))\}$.
4. Το W_1 δεν είναι μονοσύνολο, επομένως βρίσκουμε το σύνολο διαφωνίας D_1 του W_1:
$D_1 = \{X, f(\alpha)\}$
4. Από το D_1 βρίσκουμε ότι $v_1 = X$ και $t_1 = f(\alpha)$.
6. Εχουμε
$\sigma_2 = \sigma_1 \square \{t_1 / v_1\} = \{\alpha/Z\} \square \{f(\alpha)/X\} = \{\alpha/Z, f(\alpha)/X\}$,
$W_2 = W_1 \{t_1 / v_1\}$
$= \{p(\alpha, X, f(g(Y))), p(\alpha, f(\alpha), f(u))\} \{f(\alpha)/X\}$
$= \{p(\alpha, f(\alpha), f(g(Y))), p(\alpha, f(\alpha), f(U))\}$.
7. Το W_2 δεν είναι μονοσύνολο, επομένως βρίσκουμε το σύνολο διαφωνίας D_2 του W_2: $D_2 = \{g(Y), U)\}$. Από το D_2 βρίσκουμε ότι $v_2 = U$ και $t_2 = g(Y)$.
8. Εχουμε
$\sigma_3 = \sigma_2 \square \{t_2/ v_2\} = \{\alpha/Z, \{f(\alpha)/X\} \square \{g(Y)/U\} = \{\alpha/Z, f(\alpha)/X, g(Y)/U\}$,
$W_3 = W_2 \{t_2 / v_2\}$
$= \{p(\alpha, f(\alpha), f(g(Y))), p(\alpha, f(\alpha), f(U))\}\{g(Y)/U\}$
$= \{p(\alpha, f(\alpha), f(g(Y))), p(\alpha, f(\alpha), f(g(Y)))\}$
$= \{p(\alpha, f(\alpha), f(g(Y)))\}$.
9. Αφού το W_3 είναι μονοσύνολο, το $\sigma_3 = \{\alpha/Z, f(\alpha)/X, g(Y)/U\}$ είναι ο πιο γενικός ενοποιητής του W.

Παράδειγμα 4.12 Προσδιορίστε αν το σύνολο $W=\{q(f(\alpha),g(X)), q(Y,Y)\}$ είναι ή όχι ενοποιήσιμο.

1. Θέτουμε $\sigma_o = \varepsilon$ και $W_0 = W$.
2. Το W_0 δεν είναι μονοσύνολο, επομένως βρίσκουμε το σύνολο διαφωνίας D_0 του W_0: $D_0 = \{f(\alpha), Y\}$. Από το D_0 έχουμε ότι $v_o = Y$ και $t_0 = f(\alpha)$.
3. Εχουμε
$\sigma_1 = \sigma_o \square \{t_o / v_o\} = \varepsilon \square \{f(\alpha)/Y\} = \{f(\alpha)/Y\}$,
$W_1 = W_0 \{t_o / v_o\} = q(f(\alpha),g(X)), q(Y,Y)\}\{f(\alpha)/Y\}$
$= \{q(f(\alpha),g(X)), q(f(\alpha), f(\alpha))\}$.

4. Το W_1 δεν είναι μονοσύνολο, επομένως βρίσκουμε το σύνολο διαφωνίας D_1 του W_1: D_1 = {g(X), f(α)}.
4. Ωστόσο, δεν υπάρχει στοιχείο του D_1 που να είναι μεταβλητή. Επομένως ο αλγόριθμος ενοποίησης τερματίζεται και συμπεραίνουμε ότι το W δεν είναι ενοποιήσιμο.

Σημειώνουμε ότι ο παραπάνω αλγόριθμος ενοποίησης θα τερματίζεται πάντα για κάθε πεπερασμένο σύνολο εκφράσεων, γιατί διαφορετικά θα δημιουργηθεί μια άπειρη σειρά $W\sigma_0$, $W\sigma_1$, $W\sigma_2$, . . . πεπερασμένων μη κενών συνόλων εκφράσεων με την ιδιότητα ότι κάθε σύνολο επιτυχίας περιέχει μια λιγότερη μεταβλητή από το προηγούμενό του (δηλαδή, το $W\sigma_\kappa$ περιέχει το v_k ενώ το $W\sigma_{\kappa+1}$ όχι). Αυτό είναι αδύνατο αφού το W περιέχει μόνο πεπερασμένες διαφορετικές μεταβλητές.

Επισημάναμε προηγουμένως ότι αν το W είναι ενοποιήσιμο, ο αλγόριθμος ενοποίησης θα βρίσκει πάντα τον πιο γενικό ενοποιητή του W. Αυτό αποδεικνύεται στο παρακάτω θεώρημα.

Θεώρημα 4.2 (Αλγόριθμος Ενοποίησης) Αν το W είναι ένα πεπερασμένο μη κενό ενοποιήσιμο σύνολο εκφράσεων, τότε ο αλγόριθμος ενοποίησης θα τερματίζεται πάντα στο Βήμα 2, και το τελευταίο σ_κ θα είναι ο πιο γενικός ενοποιητής του W.

Αλγόριθμος ενοποίησης (β' έκδοση)

Μιά άλλη έκδοση του αλγορίθμου ακολουθεί στην επόμενη παράγραφο. Ο αλγόριθμος περιγράφεται από μια συνάρτηση που δέχεται σαν παραμέτρους λίστες από όρους και επιστρέφει μιά λίστα από ζευγάρια ή την τιμή false. Αν επιστραφεί η τιμή false, σημαίνει ότι ο αλγόριθμος απέτυχε να ενοποιήσει τις δύο λίστες όρων. Αν η λίστα είναι άδεια, σημαίνει πως το ταίριασμα ήταν επιτυχές, χωρίς απαιτούμενες αντικαταστάσεις, αλλιώς όλες οι απαιτούμενες αντικαταστάσεις περιέχονται στην λίστα που επιστρέφεται.

```
function Unify(var L1,L2:lists of terms):list of pairs;
begin
   if L1 or L2 is an atom
   then begin
      if L1,L2 identical
      then Unify:=NIL
      else if L1 is a variable
         then begin
            if L1 occurs in L2
            then Unify:=false
            else Unify:=(L2/L1);
         end
         else if L2 is a variable
            then begin
               if L2 occurs in L1
               then Unify:=false
               else Unify:=(L1/L2);
            end
            else Unify:=false;
   end
   if length(L1)<>Lenght(L2) then Unify:=false;
   SUBST:=NIL;
```

```
    for i:=1 to number of elements of L1 do
    begin
       S:=Unify(i'th element of L1, i'th element of L2);
       if S=false then Unify:=false;
       if S<>NIL
       then begin
          apply S to the remainder of both L1 and L2;
          SUBST:=APPEND(S,SUBST);
       end;
       Unify:=SUBST;
    end;
end;
```

4.5 Η αρχή της απόφασης για τη Κατηγορηματική Λογική

Εχοντας ασχοληθεί με τον αλγόριθμο ενοποίησης στην προηγούμενη παράγραφο, μπορούμε να εξετάσουμε την αρχή της απόφασης για την κατηγορική λογική πρώτης τάξεως.

Ορισμός 4.9 Αν δύο ή περισσότερα λεκτικά (με το ίδιο σύμβολο και 'πρόσημο') μιας φράσης C έχουν έναν πιο γενικό ενοποιητή σ, τότε το Cσ καλείται **παράγοντας** (factor) του C. Αν το Cσ είναι μοναδιαία φράση, τότε καλείται **μοναδιαίος παράγοντας** (unit factor) του C.

Παράδειγμα 4.13 Εστω C = $\underline{p(X)} \vee \underline{p(f(Y))} \vee \neg q(X)$. Τότε το πρώτο και το δεύτερο λεκτικό (υπογραμμισμένα) έχουν έναν πιο γενικό ενοποιητή σ = {f(Y)/X}. Επομένως, το Cσ = p(f(Y)) $\vee$ ¬q(f(Y)) είναι ένας παράγοντας του C.

Ορισμός 4.10 Εστω C_1 και C_2 δύο φράσεις, που καλούνται **γονικές φράσεις** (parent clauses), οι οποίες δεν έχουν κοινές μεταβλητές. Εστω L_1 και L_2 δύο λεκτικά των C_1 και C_2 αντίστοιχα. Αν οι L_1 και $\neg L_2$ έχουν έναν πιο γενικό ενοποιητή σ, τότε η φράση

$$(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$$

καλείται **δυαδικό αποφαινόμενο** (binary resolvent) των C_1 και C_2. Τα λεκτικά L_1 και L_2 καλούνται τα λεκτικά πάνω στα οποία εφαρμόζεται η αρχή της απόφασης.

Παράδειγμα 4.14 Εστω $C_1 = p(X) \vee q(X)$ και $C_2 = \neg p(\alpha) \vee r(X)$. Αφού το X εμφανίζεται και στην C_1 και στην C_2, μετονομάζουμε την μεταβλητή της C_2 και έχουμε $C_2 = \neg p(\alpha) \vee r(Y)$. Επιλέγουμε $L_1 = p(X)$ και $L_2 = \neg p(\alpha)$. Αφού $\neg L_2 = p(\alpha)$, τα L_1 και $\neg L_2$ έχουν πιο γενικό ενοποιητή το σ = {α/X}. Επομένως,

$$\begin{aligned}(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)= \\ = (\{p(\alpha),q(\alpha)\} - \{p(\alpha)\}) \cup (\{\neg p(\alpha),r(Y)\} - \{\neg p(\alpha)\}) \\ = \{q(\alpha)\} \cup \{r(Y)\} = \{q(\alpha),r(Y)\}= q(\alpha) \vee r(Y).\end{aligned}$$

Επομένως το $q(\alpha) \vee r(Y)$ είναι το δυαδικό αποφαινόμενο των C_1 και C_2. Τα p(X) και ¬p(X) είναι τα λεκτικά πάνω στα οποία εφαρμόζεται η αρχή της απόφασης.

Ορισμός 4.11 Το αποφαινόμενο των γονικών φράσεων C_1 και C_2 είναι ένα από τα ακόλουθα δυαδικά αποφαινόμενα:

1. δυαδικό αποφαινόμενο των C_1 και C_2
2. δυαδικό αποφαινόμενο των C_1 και ενός συντελεστή του C_2
3. δυαδικό αποφαινόμενο ενός συντελεστή του C_1 και του C_2
4. δυαδικό αποφαινόμενο ενός συντελεστή του C_1 και ενός συντελεστή του C_2

Παράδειγμα 4.15 Εστω $C_1 = p(X) \vee p(f(Y)) \vee r(g(Y))$ και $C_2 = \neg p(f(g(\alpha))) \vee q(b)$. Ενας παράγοντας του C_1 είναι $C_1' = p(f(Y)) \vee r(g(Y))$. Το δυαδικό αποφαινόμενο των C_1' και C_2 είναι το $r(g(g(\alpha))) \vee q(b)$. Επομένως, το $r(g(g(\alpha))) \vee q(b)$ είναι αποφαινόμενο των C_1 και C_2.

Η αποδεικτική διαδικασία στη ΚΛ

Η γενική αποδεικτική διαδικασία γίνεται με ένα αλγόριθμο που είναι ίδιος με τον αλγόριθμο απόδειξης με την αρχή της απόφασης που είδαμε στη Προτασιακή Λογική, με την διαφορά ότι η αναγνώριση και το ταίριασμα των συμπληρωματικών λεκτικών πραγματοποιείται από τον αλγόριθμο ενοποίησης.

Η αρχή της απόφασης, είναι ένας συμπερασματικός κανόνας ο οποίος παράγει αποφαινόμενα από ένα σύνολο προτάσεων. Αυτός ο κανόνας παρουσιάστηκε το 1965 από τον Robinson. Είναι περισσότερο αποτελεσματικός από τις προηγούμενες διαδικασίες όπως την άμεση εφαρμογή του θεωρήματος του Herbrand που χρησιμοποιήθηκε από τους Gilmore και Davis και Putnam. Επιπλέον, η αρχή της απόφασης είναι πλήρης. Αυτό σημαίνει ότι θα παράγει πάντα την κενή φράση $\square$ από ένα μη ικανοποιήσιμο σύνολο φράσεων. Πριν αποδείξουμε αυτή την πρόταση, θα δώσουμε ένα παράδειγμα πάνω στην επίπεδη γεωμετρία.

Παράδειγμα 4.16 Δείξτε ότι οι εσωτερικά εναλλασσόμενες γωνίες που σχηματίζονται από την διαγώνιο ενός τραπεζίου είναι ίσες.

Για να αποδείξουμε αυτό το θεώρημα, πρώτα ορίζουμε τις σταθερές και τα κατηγορήματα που θα χρησιμοποιήσουμε :

- Εστω ότι υπάρχουν τέσσερις σταθερές a,b,c,d οι οποίες αντιστοιχούν σε τέσσερα σημεία στο επίπεδο.
- Εστω το κατηγόρημα t(X,Y,U,V) που σημαίνει ότι το XYUV είναι ένα τραπέζιο με πάνω αριστερή κορυφή την X, πάνω δεξιά κορυφή την Y, κάτω δεξιά κορυφή την U και κάτω αριστερά κορυφή την V.
- Εστω το p(X,Y,U,V) που σημαίνει ότι η γραμμή XY είναι παράλληλη προς τη γραμμή UV.
- Εστω το e(X,Y,Z,U,V,W) που σημαίνει ότι η γωνία XYZ είναι ίση με τη γωνία UVW.

Αυτός ο ορισμός των κατηγορημάτων είναι ένας γρήγορος τρόπος γιά να ορίσουμε την σημασιολογία των σταθερών και των κατηγορημάτων της γλώσσας μας σε αντιστοιχία με τον πραγματικό κόσμο των τραπεζίων, των γωνιών, των σημείων, κ.λ.π. Οι νόμοι στους οποίους υπακούουν τα τραπέζια, οι γωνίες κ.λ.π. θα αποτυπωθούν με ένα σύνολο θεμελιωδών αξιωμάτων. Ετσι έχουμε τα παρακάτω αξιώματα:

- Ο ορισμός του τραπεζίου, που ορίζει ότι άν ένα τετράπλευρο είναι τραπέζιο, δηλαδή t(X,Y,U,V), τότε οι δύο πλευρές του είναι παράλληλες, δηλαδή p(X,Y,U,V).

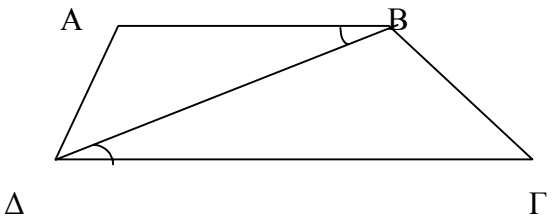
A_1 $(\forall X)\,(\forall Y)\,(\forall U)\,(\forall V)\,[t(X,Y,U,V) \rightarrow p(X,Y,U,V)]$

- Ο ορισμός της σχέσης των εσωτερικά εναλλασσόμενων γωνιών παράλληλων γραμμών, που ορίζει ότι αυτές είναι ίσες.

A_2 $(\forall X)\,(\forall Y)\,(\forall U)\,(\forall V)\,[p(X,Y,U,V) \rightarrow e(X,Y,V,U,V,Y)]$

- Το γεγονός ότι υπάρχει ένα συγκεκριμένο τραπέζιο, με κορυφές τις a,b,c,d.

A_3 $t(\alpha,b,c,d)$



Σχήμα 4.2 Το τραπέζιο του παραδείγματος

Από αυτά τα αξιώματα, είμαστε σε θέση να συμπεράνουμε ότι το $e(\alpha,b,d,c,d,b)$ αληθεύει, δηλαδή ότι

$$A_1 \wedge A_2 \wedge A_3 \rightarrow e(\alpha,b,d,c,d,b)$$

είναι μια έγκυρη σχέση. Αφού ζητάμε να αποδείξουμε αυτό με τη διαδικασία απαγωγής σε άτοπο, θεωρούμε ότι δεν ισχύει το συμπέρασμα και αποδεικνύουμε ότι το

$$A_1 \wedge A_2 \wedge A_3 \wedge \neg e(\alpha,b,d,c,d,b)$$

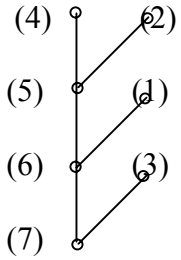
είναι μη ικανοποιήσιμο. Για να γίνει αυτό, το μετατρέπουμε σε μορφή φράσεων όπως η ακόλουθη:

$$S = \{\neg t(X_1,Y_1,U_1,V_1) \vee p(X_1,Y_1,U_1,V_1),\ \neg p(X_2,Y_2,U_2,V_2) \vee e(X_2,Y_2,V_2,U_2,V_2,Y_2),\ t(\alpha,b,c,d),\ \neg e(\alpha,b,d,c,d,b)\}.$$

Η παραπάνω μορφή S είναι ένα σύνολο τεσσάρων φράσεων. Θα δείξουμε τώρα ότι το σύνολο S είναι μη ικανοποιήσιμο με βάση την αρχή της απόφασης:

(1)	$\neg t(X_1,Y_1,U_1,V_1) \vee p(X_1,Y_1,U_1,V_1)$	φράση του S
(2)	$\neg p(X_2,Y_2,U_2,V_2) \vee e(X_2,Y_2,V_2,U_2,V_2,Y_2)$	φράση του S
(3)	$t(\alpha,b,c,d)$	φράση του S
(4)	$\neg e(\alpha,b,d,c,d,b)$	φράση του S
(5)	$\neg p(\alpha,b,c,d)$	αποφαινόμενο των (2) και (4)
(6)	$\neg t(\alpha,b,c,d)$	αποφαινόμενο των (1) και (5)
(7)	$\square$	αποφαινόμενο των (3) και (6).

Εφόσον η τελευταία φράση είναι η κενή φράση η οποία παράχθηκε από το S, συμπεραίνουμε ότι το S είναι μη ικανοποιήσιμο. Τα παραπάνω βήματα της απόδειξης μπορούν πολύ εύκολα να αναπαραστηθούν από ένα δένδρο που φαίνεται στο Σχήμα 4.3.



Σχήμα 4.3 Δέντρο παραγωγής του προβλήματος του τραπεζίου

Το δένδρο στο Σχήμα 4.3 καλείται **δένδρο παραγωγής**. Αυτό σημαίνει ότι ένα δένδρο παραγωγής ενός συνόλου φράσεων S είναι ένα (προς τα πάνω) δένδρο, στο οποίο σε κάθε αρχικό κόμβο του βρίσκεται μια φράση του S, και σε κάθε μη αρχικό κόμβο του βρίσκεται το αποφαινόμενο των φράσεων των αμέσως προηγούμενων κόμβων του. Καλούμε το δένδρο παραγωγής, ένα δένδρο παραγωγής του r αν το r είναι η φράση που βρίσκεται στη ρίζα του δένδρου παραγωγής. Αφού ένα δένδρο παραγωγής είναι απλώς το δένδρο που αναπαριστά την παραγωγή, στη συνέχεια θα χρησιμοποιήσουμε τους όρους **παραγωγή** και δένδρο παραγωγής για την παραγωγή της αυτής έννοιας.

4.6 Παραδείγματα που χρησιμοποιούν την αρχή της απόφασης

Σε αυτό το κεφάλαιο θα χρησιμοποιήσουμε παραδείγματα για να αναπαραστήσουμε το πως η αρχή της απόφασης μπορεί να είναι αποδοτική στην απόδειξη θεωρημάτων.

Παράδειγμα 4.17 Υποθέστε ότι οι τιμές των μετοχών μειώνονται εφόσον τα τραπεζικά επιτόκια αυξάνουν. Υποθέστε επίσης ότι οι περισσότεροι άνθρωποι που επενδύουν σε μετοχές στεναχωριούνται όταν παρατηρείται πτώση στις μετοχές. Δεδομένου ότι τα τραπεζικά επιτόκια αυξάνουν, αποδείξτε ότι οι περισσότεροι άνθρωποι που επενδύουν σε μετοχές είναι λυπημένοι.

Εστω ότι επιλέγουμε τα άτομα p,s,u να έχουν την ακόλουθη σημασία :

p = ‘τα τραπεζικά επιτόκια αυξάνουν’
s = ‘οι τιμές των μετοχών μειώνονται’
u = ‘οι περισσότεροι άνθρωποι που επενδύουν σε μετοχές στεναχωριούνται’

Εχουμε λοιπόν τις τέσσερις παρακάτω δηλώσεις. πρέπει να αποδείξουμε ότι η (4’) μπορεί να αποδειχθεί από τις (1’),(2’),(3’) :

(1΄) $p \rightarrow s$
(2΄) $s \rightarrow u$
(3΄) p
(4΄) u

Για να δείξουμε ότι η (4΄) προέρχεται από τις (1΄), (2΄) και (3΄), θα μετατρέψουμε πρώτα όλες τις δηλώσεις σε κανονική διαζευκτική μορφή. Επομένως έχουμε

(1) $\neg p \vee s$
(2) $\neg s \vee u$
(3) p
(4) u

Αποδεικνύουμε ότι το u είναι λογική συνέπεια των (1), (2) και (3) χρησιμοποιώντας την διαδικασία απαγωγής σε άτοπο. Θεωρούμε ότι δεν ισχύει η (4) και έχουμε τα ακόλουθα:

(1) $\neg p \vee s$
(2) $\neg s \vee u$
(3) p
(4) $\neg u$ άρνηση του συμπεράσματος
(5) s αποφαινόμενο των (3) και (1)
(6) u αποφαινόμενο των (5) και (2)
(7) □ αποφαινόμενο των (6) και (4)

Παράδειγμα 4.18 Οταν το κοινοβούλιο αρνείται να ψηφίσει καινούριους νόμους, τότε η απεργία μιάς επιχείτισης δεν σταματάει εκτός αν διαρκέσει περισσότερο από ένα χρόνο και ο διευθυντής της βιομηχανίας παραιτηθεί. Δεδομένου ότι η απεργία μόλις άρχισε και το κοινοβούλιο αρνείται να ψηφίσει καινούριους νόμους, θα σταματήσει ;

Εστω ότι επιλέγουμε τα άτομα p,q,r,s να έχουν την ακόλουθη σημασία :

p = 'το κοινοβούλιο αρνείται να ψηφίσει καινούριους νόμους'
q = 'η απεργία σταματάει'
r = 'ο διευθυντής της βιομηχανίας παραιτείται'
s = 'η απεργία διαρκεί παραπάνω από ένα χρόνο''

Εχουμε τις τρεις προτάσεις F_1, F_2 και F_3, όπως φαίνεται παρακάτω. Θέλουμε να αποδείξουμε ότι η $\neg q$ είναι λογική συνέπεια των $F_1 \wedge F_2 \wedge F_3$.

F_1 : $(p \rightarrow (\neg q \vee (s \wedge r)))$
F_2 : p
F_3 : $\neg s$
G : $\neg q$

Θεωρούμε ότι δεν ισχύει η $\neg q$, δηλαδή ότι ισχύει η q, και μετατρέπουμε το $F_1 \wedge F_2 \wedge F_3$ σε κανονική διαζευκτική μορφή. Ετσι δημιουργείται το παρακάτω σύνολο προτάσεων:

(1) $\neg p \vee \neg q \vee r$
(2) $\neg p \vee \neg q \vee s$ από την F_1
(3) p από την F_2
(4) $\neg s$ από την F_3
(5) q άρνηση του συμπεράσματος

Χρησιμοποιώντας την αρχή της απόφασης, παίρνουμε τις παρακάτω αποδείξεις

(6) $\neg q \vee s$ αποφαινόμενο των (3) και (2)
(7) s αποφαινόμενο των (6) και (5)
(8) □ αποφαινόμενο των (7) και (4)

Παράδειγμα 4.19 Θεωρείστε το παρακάτω σύνολο από προτάσεις :

F_1 : $(\forall X)\,(c(X) \rightarrow (w(X) \wedge r(X)))$
F_2 : $(\exists X)\,(c(X) \wedge o(X))$
G : $(\exists X)\,(o(X) \wedge r(X))$

Το πρόβλημά μας είναι να δείξουμε ότι το G είναι λογική συνέπεια των F_1 και F_2. Μετατρέπουμε τα F_1, F_2 και $\neg G$ στην κανονική διαζευκτική μορφή και παίρνουμε τις παρακάτω πέντε φράσεις:

(1) $\neg c(X_1) \vee w(X_1)$
(2) $\neg c(X_2) \vee r(X_2)$ από την F_1
(3) $c(a)$
(4) $o(a)$ από την F_2
(5) $\neg q(X_5) \vee \neg r(X_5)$ από την την άρνηση του G.

Το παραπάνω σύνολο φράσεων είναι μη ικανοποιήσιμο. Αυτό μπορεί να αποδειχτεί με την αρχή της απόφασης ως εξής:

(6) $r(\alpha)$ αποφαινόμενο των (3) και (2)
(7) $\neg r(\alpha)$ αποφαινόμενο των (5) και (4)
(8) $\square$ αποφαινόμενο των (7) και (6)

Αρα, το G είναι λογική συνέπεια των F_1 και F_2.

Παράδειγμα 4.20 *Προτάσεις* **:** Μερικοί ασθενείς συμπαθούν όλους τους γιατρούς. Κανένας ασθενής δεν συμπαθεί ένα τσαρλατάνο. *Συμπέρασμα* : κανένας γιατρός δεν είναι τσαρλατάνος.Εχουμε:

$p(X)$: ο X είναι ασθενής
$d(X)$: ο X είναι γιατρός
$q(X)$: ο X είναι τσαρλατάνος
$l(X,Y)$: ο X συμπαθεί τον Y

F_1 : $(\exists X)\,(p(X) \wedge (\forall Y)\,(\,d(Y) \rightarrow l(X,Y)\,)\,)$
F_2 : $(\forall X)\,(p(X) \rightarrow (\forall Y)\,(\,q(Y) \rightarrow \neg l(X,Y)\,)\,)$
G : $(\forall X)\,(d(X) \rightarrow \neg q(X))$

Οι F_1, F_2 και $\neg G$ μετατρέπονται στις παρακάτω φράσεις:

(1) $p(a)$
(2) $\neg d(X_2) \vee l(a, X_2)$ (1),(2) από την F_1
(3) $\neg p(X_3) \vee \neg q(Y_3) \vee \neg l(X_3, Y_3)$ από την F_2
(4) $d(b)$
(5) $q(b)$ (4),(5) από την άρνηση του G

Χρησιμοποιώντας την αρχή της απόφασης, παίρνουμε τα παρακάτω:

(6) $l(a,b)$ αποφαινόμενο των (4) και (2)
(7) $\neg q(Y_7) \vee \neg l(\alpha, Y_7)$ αποφαινόμενο των (3) και (1)
(8) $\neg l(\alpha,b)$ αποφαινόμενο των (5) και (7)
(9) $\square$ αποφαινόμενο των (6) και (8)

Η αρχή της απόφασης είναι αρκετά φυσική. Ας προσπαθήσουμε τώρα να μεταφράσουμε τα παραπάνω στα Ελληνικά.

(a) Από την F_1, μπορούμε να συμπεράνουμε ότι υπάρχει ένας ασθενής α που συμπαθεί κάθε γιατρό (φράσεις (1) και (2)).
(b) Ας υποθέσουμε ότι το συμπέρασμα δεν αληθεύει. Αυτό σημαίνει ότι υποθέτουμε ότι ο b είναι και γιατρός και τσαρλατάνος (φράσεις (4) και (5)).
(c) Αφού ο ασθενής α συμπαθεί κάθε γιατρό, ο α συμπαθεί τον b (φράση (6)).
(d) Αφού ο α είναι ασθενής, ο α δεν συμπαθεί κανένα τσαρλατάνο (φράση (7)).
(e) Ωστόσο, ο b είναι τσαρλατάνος. Επομένως, ο α δεν συμπαθεί τον b (φράση (8)).
(f) Αυτό είναι αδύνατο λόγω της (c). Επομένως, ολοκληρώσαμε την απόδειξη.

Παράδειγμα 4.21 *Προτάσεις* **:** Οι τελωνειακοί υπάλληλοι ψάχνουν κάθε έναν που μπαίνει στην χώρα και δεν είναι VIP. Μερικά 'βαποράκια' μπήκαν στην χώρα και αυτοί που τους έψαξαν ήταν 'βαποράκια'. Κανένα 'βαποράκι' δεν είναι VIP. *Συμπέρασμα* : Κάποιοι τελωνειακοί είναι βαποράκια'. Εχουμε :

e(X) : Είσοδος του X στη χώρα
v(X) : ο X είναι VIP (very important person!)
s(X,Y): ο Y ψάχνει (searches) τον X
c(X) : ο X είναι τελωνειακός (customs)
p(X) : ο X είναι 'βαποράκι' (pusher)

Οι προτάσεις προσδιορίζονται από τις παρακάτω φόρμες:

F_1 : $(\forall X)\ (\ (e(X) \wedge \neg v(X)) \rightarrow (\exists Y)\ (\ s(X,Y) \wedge c(Y)\ )\ )$
F_2 : $(\exists X)\ (p(X) \wedge e(X) \wedge (\forall Y)\ (\ s(X,Y) \rightarrow p(Y)\ )\ )$
F_3 : $(\forall X)\ (p(X) \rightarrow \neg v(X))$

και το συμπέρασμα είναι

G : $(\exists X)\ (\ p(X) \wedge c(X)\ )$

Μετατρέποντας τις προτάσεις σε φράσεις παίρνουμε

(1) $\neg e(X_1) \vee v(X_1) \vee s(X_1, f(X_1))$
(2) $\neg e(X_2) \vee v(X_2) \vee c(f(X_2))$
(3) $p(a)$
(4) $e(a)$
(5) $\neg s(a,Y_5) \vee p(Y_5)$
(6) $\neg p(X_6) \vee \neg v(X_6)$

Η άρνηση του συμπεράσματος είναι η

(6) $\neg p(X_7) \vee \neg c(X_7)$

Η διαδικασία απόδειξης με βάση την αρχή της απόφασης είναι η παρακάτω:

(8) $\neg v(\alpha)$ από την (3) και (6)
(9) $v(\alpha) \vee c(f(\alpha))$ από την (2) και (4)
(10) $c(f(\alpha))$ από την (8) και (9)

(11) $v(\alpha) \vee s(\alpha,f(\alpha))$ από την (1) και (4)
(12) $s(\alpha,f(\alpha))$ από την (8) και (11)
(13) $p(f(\alpha))$ από την (12) και (5)
(14) $\neg c(f(\alpha))$ από την (13) και (7)
(15) $\square$ από την (10) και (14)

Επομένως, επιβεβαιώσαμε το συμπέρασμα.

Παράδειγμα 4.22 Η πρόταση είναι ότι όποιος αποταμιεύει χρήματα κερδίζει τους τόκους. Το συμπέρασμα είναι ότι αν δεν υπάρχει τόκος, κανένας δεν αποταμιεύει χρήματα.

Εστω ότι τα s(X,Y), m(X), i(X) και e(X,Y) δηλώνουν 'ο X αποταμιεύει Y', 'το X είναι χρήματα', 'το X είναι τόκος' και 'ο X κερδίζει Y', αντίστοιχα. Τότε η πρόταση συμβολίζεται ως εξής:

$(\forall X) ((\exists Y)(s(X,Y) \wedge m(Y)) \rightarrow (\exists Y) (i(Y) \wedge e(X,Y)))$

και το συμπέρασμα είναι

$\neg(\exists X) i(X) \rightarrow (\forall X) (\forall Y) (s(X,Y) \rightarrow \neg m(Y))$

Μεταφράζοντας την πρόταση σε φράσεις, έχουμε

(1) $\neg s(X_1, Y_1) \vee \neg m(Y_1) \vee i(f(X_1))$
(2) $\neg s(X_2, Y_2) \vee \neg m(Y_2) \vee e(X_2,f(X_2))$

Η άρνηση του συμπεράσματος είναι η

(3) $\neg i(X_3)$
(4) $s(\alpha,b)$
(5) $m(b)$

Η απόδειξη που δίνεται παρακάτω είναι πολύ απλή:

(6) $\neg s(X_6, Y_6) \vee \neg m(Y_6)$ από την (3) και (1)
(7) $\neg m(b)$ από την (6) και (4)
(8) $\square$ από την (7) και (5)

Επομένως το συμπέρασμα επιβεβαιώθηκε.

Παράδειγμα 4.23 *Πρόταση* : Οι φοιτητές είναι πολίτες. *Συμπέρασμα* : Οι ψήφοι των φοιτητών είναι ψήφοι πολιτών.

Εστω ότι τα s(X), c(X) και v(X,Y) δηλώνουν ότι 'ο X είναι μαθητής', 'ο X είναι πολίτης' και 'X είναι μια ψήφος του Y' αντίστοιχα. Τότε η πρόταση και το συμπέρασμα συμβολίζονται ως εξής:

$(\forall Y) (s(Y) \rightarrow c(Y))$ (πρόταση 7)
$(\forall X) ((\exists Y) (s(Y) \wedge v(X,Y)) \rightarrow (\exists Z) (c(Z) \wedge v(X,Z))$ (συμπέρασμα)

Η κανονική διαζευκτική μορφή της πρότασης είναι η παρακάτω:

(1) $\neg s(Y_1) \vee c(Y_1)$

Αφού

$\neg (\forall X) ((\exists Y) (s(Y) \wedge v(X,Y)) \rightarrow (\exists Z) (c(Z) \wedge v(X,Z))$
$|=|$ $\neg (\forall X) (\neg (\exists Y) (s(Y) \wedge v(X,Y)) \vee (\exists Z) (c(Z) \wedge v(X,Z))$
$|=|$ $\neg (\forall X) ((\forall Y) \neg s(Y) \vee \neg v(X,Y)) \vee (\exists Z) (c(Z) \wedge v(X,Z))$
$|=|$ $\neg (\forall X) (\forall Y) (\exists Z) (\neg s(Y) \vee \neg v(X,Y) \vee (c(Z) \wedge v(X,Z))$
$|=|$ $(\exists X) (\exists Y)(\forall Z) (s(Y) \wedge v(X,Y) \wedge (\neg c(Z) \vee \neg v(X,Z))$
$|=|$ $(\exists X) (\exists Y)(\forall Z) (s(b) \wedge v(a,b) \wedge (\neg c(Z) \vee \neg v(a,Z))$

έχουμε τρεις φράσεις για την άρνηση του συμπεράσματος,

(2) $s(b)$
(3) $v(\alpha,b)$
(4) $\neg c(Z) \vee \neg v(\alpha,Z)$

Η απόδειξη συνεχίζεται ως εξής:

(5) C(b) από την (1) και (2)
(6) $\neg V(\alpha,b)$ από την (5) και (4)
(7) □ από την (6) και (3)

Παρόλο που η παραπάνω απόδειξη είναι σχεδόν μηχανική, είναι στην πραγματικότητα πολύ φυσική. Πράγματι, μπορούμε να μεταφράσουμε την παραπάνω απόδειξη στα Ελληνικά ως εξής:

Ας υποθέσουμε ότι ο b είναι μαθητής, α είναι η ψήφος του b και α δεν είναι η ψήφος κανενός πολίτη. Αφού ο b είναι μαθητής, πρέπει να είναι και πολίτης. Επιπλέον, η α δεν πρέπει να είναι ψήφος του b αφού ο b είναι πολίτης. Αυτό είναι αδύνατο. Επομένως ολοκληρώσαμε την απόδειξη.

Παράδειγμα 4.24 Εχουμε τις παρακάτω προτάσεις σε φυσική γλώσσα :
1. Κάποιος θα παρακολουθήσει το έργο, εάν έχει έγχρωμη τηλεόραση και δεν είναι απασχολημένος.
2. Κάποιος θα καταγράψει το έργο, εάν έχει video και είναι απασχολημένος.
3. Κάποιος θα παρακολουθήσει το έργο εάν το έχει καταγράψει.
4. Ο Παύλος έχει έγχρωμη τηλεόραση.
5. Η Μαρία έχει έγχρωμη τηλεόραση.
6. Ο Πέτρος έχει έγχρωμη τηλεόραση.
7. Η Σοφία έχει video.
8. Η Μαίρη έχει video.
9. Ο Γιάννης έχει video.
10. Ο Πέτρος έχει video.

Ας υποθέσουμε ακόμα ότι δίνουμε τις παρακάτω ονομασίες :

watch(X,Y) για την πρόταση "Ο X θα παρακολουθήσει το Y"
has(X,Y) για την πρόταση "Ο X έχει το Y"
busy(X) για την πρόταση "Ο X είναι απασχολημένος"
record(X,Y) για την πρόταση "Ο X θα καταγράψει το Y"

Μετά από αυτά οι παραπάνω προτάσεις στη ΚΛ γίνονται ως εξής :

1. $(\forall X)$ (watch(X,film) $\leftarrow$ has(X,color_TV) $\wedge \neg$busy(X))
2. $(\forall X)$ (record(X,film) $\leftarrow$ has(X,video) $\wedge$ busy(X))
3. $(\forall X)$ (watch(X,film) $\leftarrow$ record(X,film))
4. has(paul,color_TV)
5. has(mary,color_TV)
6. has(peter,color_TV)
7. has(sofia,video)
8. has(mary,video)
9. has(john,video)
10. has(peter,video)

Η προς απόδειξη πρόταση είναι η watch(X,Y). Μετατρέπουμε τις παραπάνω προτάσεις σε φράσεις. Επίσης πρέπει να πάρουμε την άρνηση του θεωρήματος και αφού τη μετατρέψουμε και αυτή σε φράση την εισάγουμε στο σύνολο των διαθέσιμων προτάσεων. Τελικά δημιουργούμε το εξής σύνολο προτάσεων.

(1) watch(X_1,film) $\vee \neg$has(X_1,color_TV) $\vee$ busy(X_1)
(2) record(X_2,film) $\vee \neg$has(X_2,video) $\vee \neg$busy(X_2)
(3) watch(X_3,film) $\vee \neg$record(X_3,film)
(4) has(paul,color_TV)
(5) has(mary,color_TV)
(6) has(peter,color_TV)
(7) has(sofia,video)
(8) has(mary,video)
(9) has(john,video)
(10) has(peter,video)
(11) $\neg$watch(X_{11},Y_{11})

Εφαρμόζουμε την αρχή της απόφασης σε ζευγάρια κανόνων εώς ότου φτάσουμε σε αντίφαση. Θα έχουμε όλες τις δυνατές απαντήσεις αν εφαρμόσουμε την μέθοδο εξαντλητικά.

Από (11) και (1) με την αντικατάσταση { film/Y_{11} , X_{12}/X_1, X_{12}/X_{11}} παίρνω την (12) :
(12) $\neg$has(X_{12},color_TV) $\vee$ busy(X_{12}),

Από (2) και (12) με την αντικατάσταση { X_{13}/X_2, X_{13}/X_{12}} παίρνω την (13) :
(13) $\neg$has(X_{13},color_TV) $\vee$ record(X_{13},film) $\vee \neg$has(X_{13},video)

Από (3) και (13) με την αντικατάσταση { X_{14}/X_3, X_{14}/X_{13}} παίρνω την (14) :
(14) watch(X_{14},film) $\vee \neg$has(X_{14},color_TV) $\vee \neg$has(X_{14},video)

Από (11) και (14) με την αντικατάσταση { film/Y_{11} , X_{15}/X_{14}, X_{15}/X_{11}} παίρνω την (15) :
(15) $\neg$has(X_{15},color_TV) $\vee \neg$has(X_{15},video)

Ο μόνος τρόπος να φτάσουμε σε αντίφαση είναι να βρούμε δύο προτάσεις της μορφής has(X,color_TV) και has(X,video) με την ίδια τιμή γιά τη μεταβλητή X. Τέτοιες προτάσεις είναι οι (5) και (8) που θα δώσουν την απάντηση X=mary, και οι (6) και (10) που θα δώσουν την απάντηση X=peter. Μάλιστα, εάν εφαρμόσουμε τη μέθοδο εξαντλητικά θα βρούμε δύο

τέτοια ζευγάρια λύσεων, δηλαδή η μέθοδος θα παράγει τις απαντήσεις mary, peter από δύο φορές την κάθε μία. Αυτό φαίνεται άν παρατηρήσουμε πως η πρόταση (11) μπορεί αρχικά να συνδυαστεί είτε με την πρόταση (1), είτε με την πρόταση (3). Το συμπέρασμά είναι ότι βάση της αρχής της απόφασης όλες οι δυνατές απαντήσεις είναι {mary,peter} και μόνο αυτές.

Παράδειγμα 4.25 Συνεχίζοντας το παράδειγμα 4.16 από τον χώρο των συνόλων έχουμε :
Μετονομασία μεταβλητών και ένωση σε ένα σύνολο:
1: $\{\neg a(X_1,S_1) \vee \neg a(X_1,T_1) \vee a(X_1,\tau(S_1,T_1))$,
2: $\neg a(X_2,\tau(S_2,T_2)) \vee a(X_2,S_2)$,
3: $\neg a(X_3,\tau(S_3,T_3)) \vee a(X_3,S_3)$,
4: $a(f(S_4,T_4),S_4) \vee u(S_4,T_4)$,
5: $\neg a(f(S_5,T_5),T_5) \vee u(S_5,T_5)$,
6: $\neg u(\tau(a,b),a)\}$
Απόδειξη της κενής πρότασης
7. από 5,6 με $\sigma = \{\tau(a,b)/S_5, a/T_5\}$ παράγεται η $\neg a(f(\tau(a,b),a),a)$
8. από 4,6 με $\sigma = \{\tau(a,b)/S_4, a/T_4\}$ παράγεται η $a(f(\tau(a,b),a),\tau(a,b))$
9. από 2,8 με $\sigma = \{f(\tau(a,b), a)/X_2, a/S_2, b/T_2\}$ παράγεται η $a(f(\tau(a,b),a),a)$
10. από 7,9 με $\sigma = \{\}$ παράγεται η κενή πρόταση

Παράδειγμα 4.26 Χρησιμοποιείστε την αρχή της απόφασης για την παραγωγή της άδειας πρότασης από το παρακάτω σύνολο:
1: $\{M(a, s(c), s(b))$
2: $P(a)$
3: $M(X_3, X_3, s(X_3))$
4: $\neg M(X_4, Y_4, Z_4) \vee M(Y_4, X_4, Z_4)$
5: $\neg M(X_5, Y_5, Z_5) \vee D(X_5, Z_5)$
6: $\neg P(X_6) \vee \neg M(Y_6, Z_6, U_6) \vee \neg D(X_6, U_6) \vee D(X_6, Y_6) \vee D(X_6, Z_6)$
7: $\neg D(a, b)\}$
8. από 1,5 με $\sigma = \{a/X_5, s(c)/Y_5, s(c)/Z_5\}$
$D(a, s(b))$
1: από 2,6, $\sigma = \{a/X_6\}$
$\neg M(Y_9, Z_9, U_9) \vee \neg D(a, U_9) \vee D(a, Y_9) \vee D(a, Z_9)$
2: από 9,8 με $\sigma = \{s(b)/U_9\}$
$\neg M(Y_{10}, Z_{10}, s(b)) \vee D(a, Y_{10}) \vee D(a, Z_{10})$
3: από 3,10 με $\{X_3/Y_{10}, X_3/Z_{10}, b/X_3\}$ δηλαδή $\sigma = \{b/X_3, b/Y_{10}, b/Z_{10}\}$
$D(a, b) \vee D(a, b)$
4: Με 'παραγόντιση' στην 11
$D(a, b)$
5: από 7,12 με $\sigma = \varnothing$

5

Στρατηγικές της Αρχής της Απόφασης

5.1 Εισαγωγή

Οι γενικοί αλγόριθμοι που περιγράψαμε στα προηγούμενα κεφάλαια τόσο για τον Προτασιακό Λογισμό, όσο και για τον Κατηγορικό λογισμό επιλέγουν προτάσεις τελείως τυχαία. Αυτό έχει σαν αποτέλεσμα την κατανάλωση χρόνου και χώρου (μνήμης) για αποφαινόμενες προτάσεις οι οποίες δεν συνεισφέρουν στην απόδειξη. Γίνεται δηλαδή μια 'τυφλή' αναπαραγωγή προτάσεων, που πολλές φορές οδηγεί σ' αυτό που ονομάζεται συνδυαστική έκρηξη.

Στο κεφάλαιο αυτό θα περιγράψουμε κατ'αρχήν κάποια γενικά θέματα για την έννοια των διαδικασιών απόδειξης, και στη συνέχεια θα περιγραφούν μερικές ειδικές στρατηγικές της αρχής της απόφασης που βελτιώνουν την απόδοση ελαττώνοντας την αναπαραγωγή άχρηστων προτάσεων.

5.2 Διαδικασίες απόδειξης

Αν θέλουμε να περιγράψουμε μαθηματικά μιά διαδικασία απόδειξης, τότε πρέπει να θεωρήσουμε το σύνολο N των φυσικών, ένα αρχικό σύνολο προτάσεων Δ και τη κλάση D όλων των συνόλων προτάσεων. Μια διαδικασία απόδειξης είναι μια συνάρτηση step :

$$\text{step: } D \times N \rightarrow D$$

Προφανώς,

$$\text{step: } (\Delta,1) = \Delta$$

Το επόμενο βήμα (next) αποτυπώνει το σύνολο των προτάσεων που παράγονται μετά την εφαρμογή κάποιας διαδικασίας απόδειξης. Κάθε φορά που εφαρμόζουμε ένα αποδεικτικό βήμα ένα νέο σύνολο προτάσεων προκύπτει : Το προηγούμενο σύνολο μαζί με το σύνολο των προτάσεων που έχουν παραχθεί κατά το αποδεικτικό βήμα. Υπάρχουν αποδεικτικές διαδικασίες που βασίζονται μόνο στο τελευταίο κάθε φορά σύνολο. Τέτοιες ονομάζονται

διαδικασίες απόδειξης τύπου Markov (Markov inference procedures) και έχουν το χαρακτηριστικό ότι δεν ενδιαφέρονται γιά την ενδιάμεση ιστορία της απόδειξης.

(a) Markov inference procedures

$$\text{next: } D \rightarrow D$$

και

$$\text{step}(\Delta,n) = \begin{cases} \Delta & , n=1 \\ \text{next}(\text{step}(\Delta,n-1)), & n>1 \end{cases}$$

Τέτοιες διαδικασίες απόδειξης δεν μας απασχολούν. Μας απασχολούν πιό πολύπλοκες διαδικασίες που ονομάζονται **αυξητικές διαδικασίες απόδειξης** (incremental inference procedures) :

(β) incremental inference procedures

$$\text{new: } D \times N \rightarrow D$$

και

$$\text{step}(\Delta,n) = \begin{cases} \Delta & , n=1 \\ \text{append}(\text{step}(\Delta,n-1), \text{new}(\Delta,n-1)), & n>1 \end{cases}$$

Στη συνέχεια θα δώσουμε ένα τέτοιο παράδειγμα αυξητικής διαδικασίας απόδειξης

Παράδειγμα 5.1 Ενα παράδειγμα αυξητικής διαδικασίας απόδειξης

Ορίζουμε τις συναρτήσεις step και new ως εξής :

$$\text{step}(\Delta,n) = \begin{cases} \Delta & , n=1 \\ \text{append}(\text{step}(\Delta,n-1), \text{new}(\Delta,n-1)), & n>1 \end{cases}$$

$$\text{new}(\Delta,n) = \begin{cases} \Delta, & n=1 \\ [x]\ \text{mp}(\text{first}(\text{fast}(\Delta,n-1)), \text{first}(\text{slow}(\Delta,n-1)), x) & \\ [\ ], & \text{otherwise} \end{cases}$$

$$\text{fast}(\Delta,n) = \begin{cases} \Delta, & n=1 \\ \text{append}(\text{step}(\Delta,n-1), \text{new}(\Delta,n)), & \text{fast}(\Delta,n-1)=\text{slow}(\Delta,n-1) \\ \text{append}(\text{rest}(\text{fast}(\Delta,n-1)), \text{new}(\Delta,n)), & \text{otherwise} \end{cases}$$

$$\text{slow}(\Delta,n) = \begin{cases} \Delta, & n=1 \\ \text{append}(\text{rest}(\text{slow}(\Delta,n-1)), \text{new}(\Delta,n)), & \text{fast}(\Delta,n-1)=\text{slow}(\Delta,n-1) \\ \text{append}(\text{slow}(\Delta,n-1)), \text{new}(\Delta,n)), & \text{otherwise} \end{cases}$$

Η συνάρτηση mp είναι το γνωστό μας Modus Ponens :

$$\left.\begin{array}{l} A \\ \dfrac{A \rightarrow B}{B} \end{array}\right\} \text{Modus Ponens}$$

Θα μπορούσαμε να γράψουμε τις παραπάνω συναρτήσεις λίγο πιό απλά :

$$\text{step}_n = \begin{cases} \Delta, & n=1 \\ \text{step}_{n-1} \cup \text{newn}_{-1}, & n>1 \end{cases}$$

$$new_n = \begin{cases} \Delta, & n=1 \\ \{x\}, \text{ if } MP(first(fast_{n-1}), first(slow_{n-1}), x) \\ 0, \text{ otherwise} \end{cases}$$

$$fast_n = \begin{cases} \Delta, & n=1 \\ step_{n-1} \cup newn \text{ if } fast_{n-1}=slow_{n-1} \\ rest(fast_{n-1}) \cup new_n, \text{ otherwise} \end{cases}$$

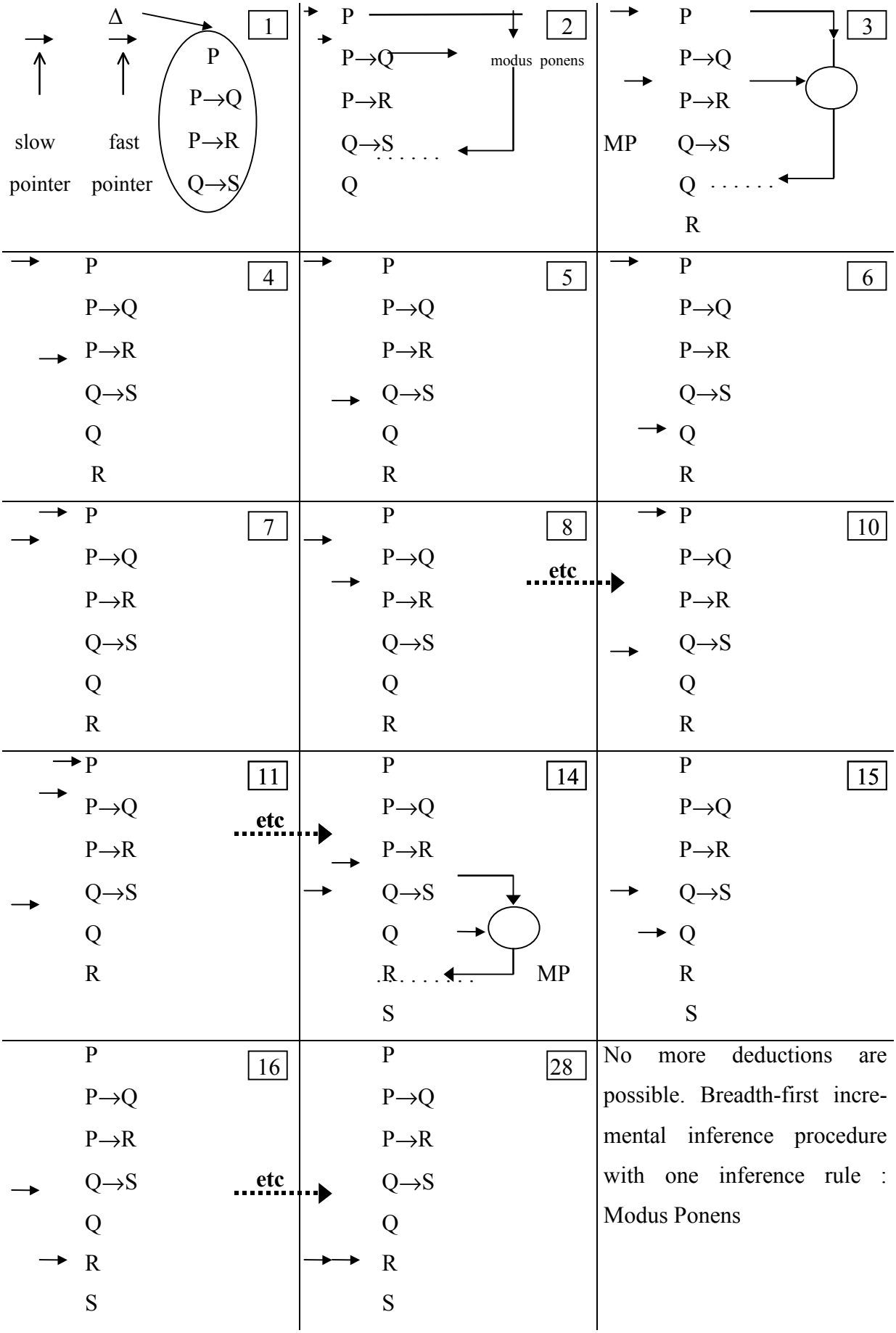
$$slow_n = \begin{cases} \Delta, & n=1 \\ rest(slow_{n-1}) \cup new_n \text{ if } fast_{n-1}=slow_{n-1} \\ slow_{n-1} \cup new_n, \text{ otherwise} \end{cases}$$

- $fast_n$, $slow_n$ είναι υποσύνολα του $step_n$. Οι δείκτες αυτοί μετακινούνται πάνω στο σύνολο, ο ένας γρήγορα (fast) και ο άλλος αργά (slow). Χρησιμεύουν στο να παίρνουμε όλες τις προτάσεις του συνόλου με τη σειρά χωρίς να ξεχνάμε κάποια φράση, άρα και κάποια πιθανή παραγωγή νέας φράσης.
- το new περιέχει την παραγόμενη νέα φράση.
- το $step_n$ είναι μια ακολουθία συνόλων προτάσεων που κάθε νέος όρος της περιέχει το προηγούμενο σύνολο προτάσεων και την παραγόμενη φράση.
- οι συναρτήσεις αυτές επιτρέπουν τον έλεγχο για εφαρμογή του Modus Ponens μεταξύ όλων των δυνατών ζευγαριών προτάσεων.
- Modus Ponens(A, A→B, B) ή MP(A→B, A, B)

Ας εφαρμόσουμε αυτή τη διαδικασία απόδειξης στο αρχικό σύνολο

$$\Delta = \{P, P \rightarrow Q, P \rightarrow R, Q \rightarrow S\}$$

Τα διαδοχικά βήματα μετά την εκτέλεση αυτής της διαδικασίας φαίνονται στο σχήμα της επόμενης σελίδας.



No more deductions are possible. Breadth-first incremental inference procedure with one inference rule : Modus Ponens

Είδαμε πως είναι δυνατόν να αναπαραστήσει κάποιος μιά διαδικασία απόδειξης μέσω συναρτήσεων. Στη συνέχεια θα ξεφύγουμε από τη μοντελοποίηση μέσω συναρτήσεων των διαδικασιών απόδειξης και θα επανέλθουμε στο γνωστό μοντέλο της λογικής, όπως το είδαμε στα προηγούμενα κεφάλαια.

5.3 Στρατηγικές διαγραφής

Οι στρατηγικές διαγραφής απορρίπτουν προτάσεις με συγκεκριμένες ιδιότητες, πριν αυτές χρησιμοποιηθούν.

Ενα λεκτικό σε ένα σύνολο αξιωμάτων είναι **γνήσιο** (pure) εάν και μόνο εάν δεν εμφανίζεται αλλού στο σύνολο αξιωμάτων με αντίθετο πρόσημο. Αν θεωρήσουμε για παράδειγμα το ακόλουθο σύνολο αξιωμάτων.

$$F = \{\neg P \vee \neg Q \vee \neg R, \neg P \vee S, \neg Q \vee S, P, Q, \neg R\}$$

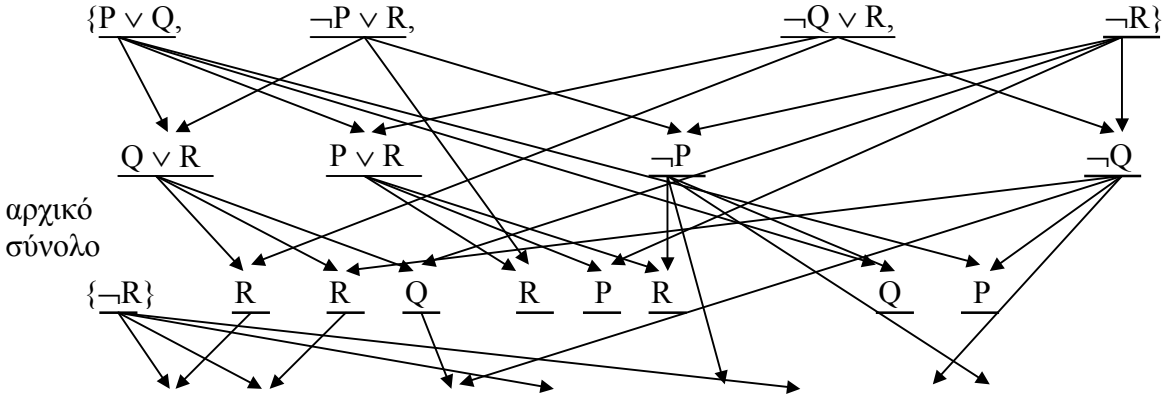
Το λεκτικό S δεν εμφανίζεται σε καμιά φράση σαν ¬S, οπότε οι προτάσεις που το περιέχουν είναι άχρηστες στην απόδειξη και πρέπει να διαγραφούν. Η στρατηγική της διαγραφής αυτής αναφέρεται σαν **διαγραφή των γνήσιων λεκτικών** (pure-literal elimination).

Η παρουσία ή η απουσία ταυτολογιών σε ένα σύνολο αξιωμάτων κανένα ρόλο δεν παίζει στην αποδεικτική διαδικασία. Η φράση $P(X) \vee Q(Y) \vee \neg Q(Y) \vee R(Z)$ είναι μια ταυτολογία. Η στρατηγική που διαγράφει τις ταυτολογίες ονομάζεται **διαγραφή των ταυτολογιών** (tautology elimination).

Μια άλλη περίπτωση διαγραφής είναι η **διαγραφή υποπροτάσεων** (subsumption elimination). Μια φράση Φ είναι υποφράση (subsumes) μιας φράσης Ψ εάν υπάρχει μια αντικατάσταση (substitution) σ τέτοια ώστε $\Phi\sigma \subseteq \Psi$. Η έννοια του υποσυνόλου έχει το νόημα ότι η Φσ περιέχεται μέσα στην Ψ. Για παράδειγμα η φράση $\Phi = P(X) \vee Q(Y)$ είναι υποφράση της φράσης $\Psi = P(a) \vee Q(V) \vee R(W)$ γιατί για την αντικατάσταση $\sigma = \{a/X, Y/V\}$ η Φσ περιέχεται μέσα στην Ψ.

5.4 Η γενική περίπτωση

Θεωρείστε το σύνολο :



Η γενική μεθοδολογία γιά την παραγωγή της κενής φράσης απαιτεί την τυχαία επιλογή φράσεων. Για να είναι πλήρης, η εφαρμογή του γενικού αλγορίθμου που διαλέγει όλους τους γενικούς συνδυασμούς φράσεων και προσπαθεί να παράγει την κενή φράση. Εάν χρησιμοποιούσαμε αυτή την γενική μεθοδολογία για να βρούμε την κενή φράση θα χρειαζόμασταν 24 χρήσεις!

Βέβαια στο συγκεκριμένο παράδειγμα είμαστε τυχεροί γιατί η κενή φράση αποδεικνύεται με πολλούς τρόπους και έτσι μπορεί να διαλέξουμε το σωστό μονοπάτι αρκετά νωρίς. Δεν είναι έτσι πάντα. Υπάρχουν περιπτώσεις που η κενή φράση αποδεικνύεται με πολύ λίγους τρόπους και η επιλογή των φράσεων παίζει πολύ σημαντικό ρόλο.

Οταν προσπαθούμε να εφαρμόσουμε τον γενικό αλγόριθμο στο χαρτί, πάντα κάνουμε κάποιες επιλογές που προβλέπουμε ότι θα μας οδηγήσουν στην κενή φράση. Ομως αν θέλουμε να υλοποιήσουμε τον αλγόριθμο σε μιά μηχανή, πρέπει να βρούμε τρόπους να πραγματοποιήσουμε αυτές τις στρατηγικές επιλογές. Αυτοί οι τρόποι ονομάζονται στρατηγικές της αρχής της απόφασης.

Χωρίς τέτοιες στρατηγικές τα προγράμματα οδηγούνται σε αυτό που λέγεται συνδυαστική έκρηξη. Η συνδυαστική έκρηξη οφείλεται στην διαρκώς αυξανόμενη παραγωγή νέων προτάσεων χωρίς να βρίσκεται η κενή πρόταση. Η τυφλή παραγωγή οδηγεί σε τεράστια σύνολα, με αποτέλεσμα, πολύ γρήγορα να εξαντλείται ο χώρος της μνήμης του υπολογιστικού συστήματος.

Οι στρατηγικές που θα περιγραψουμε υιοθετούν μερικές από τις επιλογές μας που φαίνεται να οδηγούν με μεγαλύτερη ασφάλεια στη κενή πρόταση. Μερικές όμως από αυτές τις στρατηγικές φαίνεται ότι δεν είναι πλήρεις. Ας θυμηθούμε λίγο την έννοια της πληρότητας. Πλήρες λέγεται ένα σύστημα απόδειξης όταν μπορεί να αποδείξει ότι επαληθεύεται στη σημασιολογία. Δηλαδή,

$$\Delta \models \varphi \Rightarrow \Delta \vdash \varphi$$

Αυτό ισοδύναμα γράφεται,

$$\Delta \cup \{\neg \varphi\} \models \text{`ψευδές'} \Rightarrow \Delta \cup \{\neg\varphi\} \vdash$$

Εαν δεν είναι πλήρες ένα τέτοιο σύστημα, τότε υπάρχουν σύνολα προτάσεων μη ικανοποιήσιμα από τα οποία δεν μπορεί να παραχθεί η κενή πρόταση. Δηλαδή,

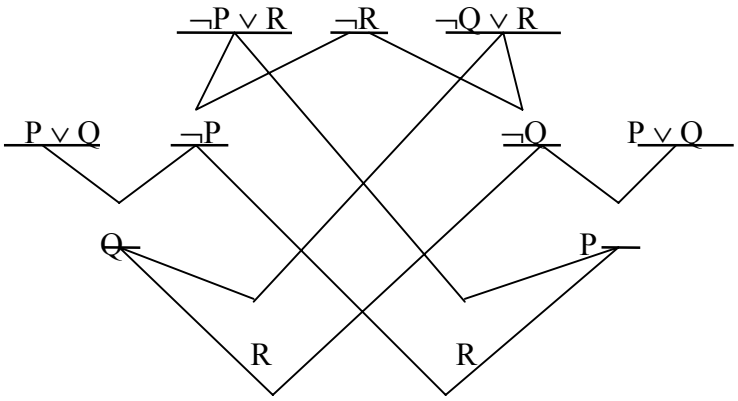
$$\Delta \cup \{\neg \varphi\} \models \text{`ψευδές'} \Rightarrow \Delta \cup \{\neg\varphi\} \vdash$$

Αυτό μπορεί να είναι ιδιαίτερα ενοχλητικό γιατί το σύστημα δεν θα μπορεί να αποδείξει μερικά από τα θεωρήματα. Το πρόβλημα αυτό μπορεί να λυθεί άν προσπαθήσουμε σε ένα σύστημα να εφαρμόσουμε παραπάνω από μιά στρατηγικές, με κάποια προτεραιότητα η κάθε μία. Στη συνέχεια θα μελετήσουμε τέτοιες στρατηγικές και τις ιδιότητές τους όσον αφορά στην πληρότητα.

5.5 Αρχή της απόφασης με μονάδες (Unit Resolution)

Μοναδιαία φράση (unit clause) είναι η φράση που αποτελείται από ένα λεκτικό. **Μοναδιαίο αποφαινόμενο** (unit resolvent) είναι το αποφαινόμενο δύο γονικών προτάσεων, από τις οποίες η μία τουλάχιστον είναι μοναδιαία φράση. **Μοναδιαία παραγωγή** (unit deduction) είναι η παραγωγή στην οποία όλα τα παραγόμενα είναι μοναδιαία αποφαινόμενα. **Μοναδιαία απαγωγή** (unit refutation) είναι η μοναδιαία παραγωγή της κενής φράσης. Εστω για παράδειγμα το εξής σύνολο προτάσεων:

$\{P \vee Q, \neg P \vee R, \neg Q \vee R, \neg R\}$.



Μια αποδεικτική διαδικασία που χρησιμοποιεί αυτή την στρατηγική φαίνεται στο σχήμα αριστερά.

Οι διαδικασίες που βασίζονται στην αρχή της απόφασης με μονάδες είναι εύκολο να υλοποιηθούν και είναι συνήθως αρκετά αποδοτικές (efficient). Αρκεί να σημειωθεί ότι κάθε φορά που γίνεται εφαρμογή της αρχής της απόφασης μεταξύ δύο προτάσεων από τις οποίες η μία είναι μοναδιαία φράση, η αποφαινόμενη φράση έχει λιγότερα λεκτικά από την μια γονική φράση. Αυτό βοηθάει στην συγκέντρωσή μας σε μονοπάτι,όπου ο αριθμός των λεκτικών συνεχώς ελαττώνεται, έως ότου τελικά παραχθεί η κενή φράση.

Δυστυχώς η στρατηγική αυτή δεν είναι πλήρης (complete). Για παράδειγμα το σύνολο $\{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ ενώ οδηγεί σε κενή φράση με τον γενικό αλγόριθμο, δεν οδηγεί στην κενή φράση με την δεδομένη στρατηγική καθώς καμία από τις προτάσεις δεν είναι μοναδιαία. Υπάρχει όμως ένα είδος φράσεων, που λέγονται φράσεις τύπου Horn, για τις οποίες η συγκεκριμένη στρατηγική είναι πλήρης.

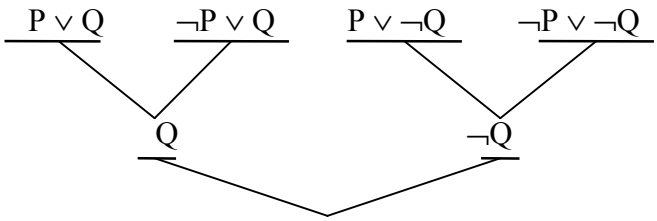
5.7. Αρχή της απόφασης στην είσοδο (Input Resolution)

Με την στρατηγική αυτή απαιτούμε μια τουλάχιστον από τις γονικές φράσεις να είναι φράση του αρχικού συνόλου προτάσεων.

Εχει αποδειχθεί ότι η αρχή της απόφασης στην είσοδο είναι ισοδύναμη στρατηγική με την αρχή της απόφασης με μονάδες, δηλαδή ότι μπορεί να γίνει με την μια στρατηγική το ίδιο μπορεί να γίνει και με την άλλη.

Ετσι, η αρχή της απόφασης στην είσοδο είναι επίσης πλήρης για προτάσεις Horn, αλλά για γενικές προτάσεις δεν είναι βέβαιο ότι μπορεί να δώσει λύσεις ακόμη και αν δίνει λύση ο γενικός αλγόριθμος.

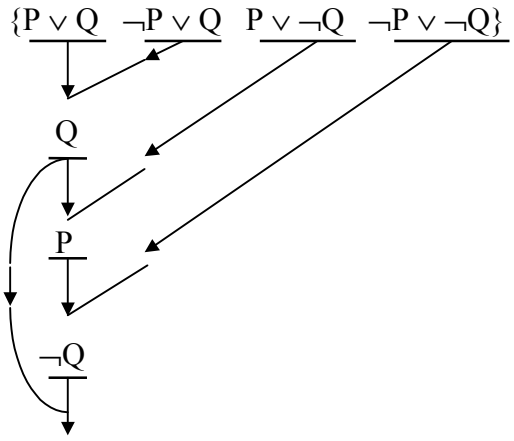
Για παράδειγμα το σύνολο των προτάσεων {P ∨ Q, ¬P ∨ Q, P ∨ ¬Q, ¬P ∨ ¬Q} και πάλι δεν μπορεί να οδηγηθεί σε λύση, γιατί όπως φαίνεται από το παρακάτω σχήμα η κενή φράση συμπεραίνεται από τις προτάσεις Q και ¬Q που όμως δεν ανήκουν στο αρχικό σύνολο προτάσεων.



5.8 Γραμμική αρχή της απόφασης (Linear Resolution)

Η γραμμική αρχή της απόφασης είναι μια ελαφρή γενικοποίηση της αρχής της απόφασης στην είσοδο. **Γραμμικό αποφαινόμενο** (linear resolvent) είναι εκείνο του οποίου τουλάχιστον μία από τις γονικές προτάσεις είτε είναι φράση του αρχικού συνόλου, είτε είναι πρόγονος της άλλης γονικής φράσης.

Η στρατηγική αυτή παίρνει το όνομά της από το γραμμικό σχήμα των αποδείξεων που δημιουργεί. **Μια γραμμική παραγωγή** (linear deduction) ξεκινάει με μια φράση του αρχικού συνόλου, που ονομάζεται **κορυφαία** (top clause) και παράγει μια **γραμμική αλυσίδα** από αποφαινόμενες προτάσεις.



5.9 Αρχή της απόφασης με σύνολο υποστήριξης

Οταν έχουμε ένα σύνολο αξιωμάτων και θέλουμε να αποδείξουμε μια φράση, την αντιστρέφουμε και την εισάγουμε στο σύνολο των αξιωμάτων. Το σύνολο των αξιωμάτων μαζί με την αντίθεση της προς απόδειξη φράσης είναι μια αντίφαση και οδηγεί στην κενή φράση. Είναι δηλαδή όπως λέμε ένα σύνολο προτάσεων ανικανοποιήσιμο (unsatisfiable).

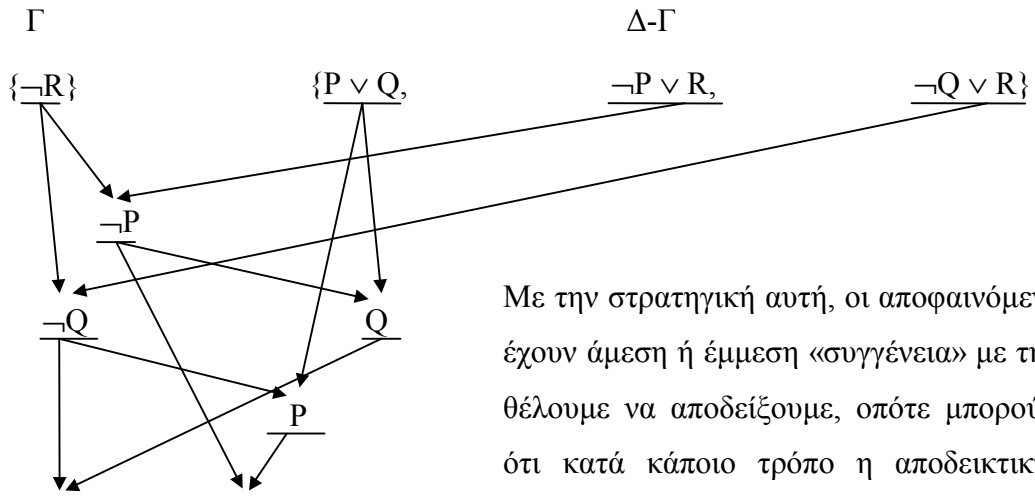
Το σύνολο όμως των αξιωμάτων είναι ικανοποιήσιμο (satisfiable) γιατί αν ήταν αλλιώς, αν ήταν από μόνο του ανικανοποιήσιμο θα μπορούσαμε να αποδείξουμε οποιαδήποτε φράση ή την αντίθετή της, γιατί η κενή φράση θα παραγόταν έτσι κι αλλιώς.

Ενα υποσύνολο Γ ενός συνόλου Δ λέγεται **σύνολο υποστήριξης** (set of support) εάν και μόνο εάν το σύνολο Δ-Γ είναι ικανοποιήσιμο.

Πρακτικά, το σύνολο υποστήριξης μπορεί να είναι το σύνολο των προτάσεων (clauses) που παράγεται από την άρνηση της προς απόδειξης φράσης.

Αποφαινόμενο του συνόλου υποστήριξης (set of support resolvent) είναι εκείνη η φράση της οποίας τουλάχιστον μια από τις γονικές προτάσεις ανήκει στο σύνολο υποστήριξης Γ, ή είναι απόγονος του συνόλου Γ.

Ας υποθέσουμε για παράδειγμα το σύνολο των αξιωμάτων $\{P \vee Q, \neg P \vee R, \neg Q \vee R\}$ και η φράση που θέλουμε να αποδείξουμε είναι η R. Την αντιστρέφουμε ($\neg R$) και έχουμε το αρχικό σύνολο Δ: $\Delta = \{P \vee Q, \neg P \vee R, \neg Q \vee R, \neg R\}$ από το οποίο το Γ δηλαδή το σύνολο υποστήριξης είναι $\Gamma = \{\neg R\}$.

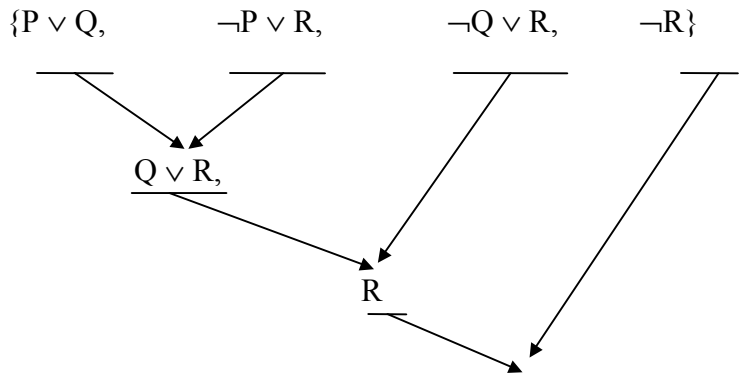


Με την στρατηγική αυτή, οι αποφαινόμενες προτάσεις έχουν άμεση ή έμμεση «συγγένεια» με την φράση που θέλουμε να αποδείξουμε, οπότε μπορούμε να πούμε ότι κατά κάποιο τρόπο η αποδεικτική διαδικασία οδηγείται από τον στόχο (Goal) είναι «goal-driven» ή ότι κινείται από το στόχο ανάποδα προς την κενή φράση (backward). Τους όρους αυτούς θα τους δούμε αναλυτικά σε επόμενα κεφάλαια.

5.10 Ταξινομημένη αρχή της απόφασης (Ordered Resolution)

Η στρατηγική αυτή είναι πολύ περιορισμένη, γιατί κάθε φράση θεωρείται σαν ένα γραμμικά ταξινομημένο σύνολο (linearly ordered set). Σ' αυτά τα σύνολα η αρχή της απόφασης επιτρέπεται μόνο στο πρώτο κάθε φορά λεκτικό.

Οι αποφαινόμενες προτάσεις (μπορούμε να τις ονομάζουμε και συμπεράσματα) διατηρούν την σειρά των λεκτικών των γονικών προτάσεων, και μάλιστα πρώτα μπαίνουν τα λεκτικά της φράσης που περιέχει το θετικό άτομο που απαλείφεται, και στην συνέχεια τα λεκτικά της φράσης που περιέχουν το αρνητικό άτομο που απαλείφεται.



Στο σχήμα αυτό φαίνονται οι μόνες δυνατές αποφαινόμενες προτάσεις με την στρατηγική αυτή. Η στρατηγική της ταξινομημένης αρχής της απόφασης είναι πολύ αποτελεσματική. Οπως φαίνεται και στο σχήμα χρειάστηκαν μόνο τρεις χρήσεις της αρχής της απόφασης για να φτάσουμε στην κενή φράση.

Δυστυχώς, η ταξινομημένη αρχή της απόφασης είναι πλήρης μόνο για φράσεις τύπου Horn.

5.11 Κατευθυνόμενη αρχή της απόφασης (Directed Resolution)

Η κατευθυνόμενη αρχή της απόφασης είναι μια μορφή ταξινομημένης αρχής της απόφασης στην οποία το σύνολο των αξιωμάτων αποτελείται από κατευθυνόμενες φράσεις (directed clauses).

Μια **κατευθυνόμενη φράση** είναι μια φράση τύπου Horn στην οποία το θετικό λεκτικό εμφανίζεται είτε στην αρχή είτε στο τέλος της φράσης.

Στην κατευθυνόμενη αρχή της απόφασης η φράση που θέλουμε να αποδείξουμε έχει την μορφή σύζευξης θετικών λεκτικών. Σκοπός είναι να βρεθούν οι τιμές των μεταβλητών έτσι ώστε μετά την αντικατάσταση των τιμών η σύζευξη των λεκτικών να είναι **αποδείξιμη** (provable) από το σύνολο των αξιωμάτων.

Στην στρατηγική αυτή χρησιμοποιείται η έννοια της κατεύθυνσης στη διαδικασία απόδειξης. Εμφανίζονται δύο τύποι κατευθύνσεων: '**προς τα μπρος**' (forward) και '**προς τα πίσω**' (backward).

Για να επιτευχθεί αυτός γίνεται ένας πρώτος μετασχηματισμός των προτάσεων που είναι αξιώματα. Ο μετασχηματισμός αυτός σχετίζεται με τη χρήση του ‘συνεπάγεται’ (← ή →). Ετσι, για τα διάφορα είδη προτάσεων έχουμε τους ακόλουθους μετασχηματισμούς:

$\neg\Phi_1 \vee ... \vee \neg\Phi_n \vee \Psi$ γίνεται $\Phi_1, ..., \Phi_n \Rightarrow \Psi$
$\Psi \vee \neg\Phi_1 \vee ... \vee \neg\Phi_n$ γίνεται $\Psi \Leftarrow \Phi_1, ..., \Phi_n$
$\neg\Phi_1 \vee ... \vee \neg\Phi_n$ γίνεται $\Phi_1, ..., \Phi_n \Rightarrow$
ή $\Leftarrow \Phi_1, ..., \Phi_n$

Στην προς τα εμπρός χρήση της αρχής της απόφασης (forward resolution) θετικά συμπεράσματα παράγονται από θετικά δεδομένα.

Αντίθετα, στην προς τα πίσω χρήση της αρχής της απόφασης (backward resolution), αρνητικές προτάσεις παράγονται από άλλες αρνητικές προτάσεις. (Τα αρνητικά λεκτικά μπορούν να θεωρηθούν και οι στόχοι).

Μια ‘**προς τα εμπρός**’ φράση (forward clause) είναι εκείνη που έχει ένα θετικό λεκτικό στο τέλος.

$$\neg\Phi_1 \vee ... \vee \neg\Phi_n \vee \Psi \text{ ή } \Phi_1, ..., \Phi_n \Rightarrow \Psi.$$

Οι ‘προς τα εμπρός’ προτάσεις ενεργοποιούν ‘προς τα εμπρός’ συμπερασματολογία στην κατευθυνόμενη αρχή της απόφασης.

1.	$\neg M(X) \vee P(X)$	ή	$M(X) \rightarrow P(X)$
2.	$M(a)$		$M(a)$
3.	$\neg P(Z)$		$P(Z) \rightarrow$
1, 2: 4.	$P(a)$		$P(a)$
3, 4: 5.			

Η χρήση της κατεύθυνσης στην διαδικασία απόδειξης έτσι όπως ορίζεται εδώ, σημαίνει ότι σε «προς τα εμπρός» συμπερασματολογία δεν μπορούμε να χρησιμοποιήσουμε τις προτάσεις 1, 3. Πρέπει οπωσδήποτε, να ακολουθήσουμε την κατεύθυνση από αριστερά προς τα δεξιά.

Στην ‘προς τα πίσω’ συμπερασματολογία συμβαίνουν αντίστοιχα πράγματα. Ετσι

1.	$P(X) \vee \neg M(X)$	ή	$P(X) \leftarrow M(X)$
2.	$M(a)$		$M(a)$
3.	$\neg P(Z)$		$\leftarrow P(Z)$
1, 3: 4.	$\neg M(Z)$		$\leftarrow M(Z)$
2, 4: 5.			

Πάλι η κατεύθυνση που ακολουθούμε είναι από αριστερά προς τα δεξιά, και έτσι είναι αδύνατον να εφαρμόσουμε την αρχή της απόφασης μεταξύ των προτάσεων 1, 2.

Οταν μερικές προτάσεις είναι ‘προς τα εμπρός’ και μερικές άλλες ‘προς τα πίσω’ μπορούμε να έχουμε συνδυασμό των κατευθύνσεων. Ας το δούμε αυτό με ένα παράδειγμα:

	1.	$\neg P(X) \vee \neg Q(X) \vee R(X)$	ή	$P(X), Q(X) \rightarrow R(X)$
	2.	$\neg M(X) \vee P(X)$		$M(X) \rightarrow P(X)$
	3.	$Q(X) \vee \neg N(X)$		$Q(X) \leftarrow N(X)$
	4.	M(a)		M(a)
	5.	M(b)		M(b)
	6.	N(b)		N(b)
	7.	$\neg R(Z)$		$R(Z) \rightarrow$
2, 4:	8.	P(a)		P(a)
2, 5:	9.	P(b)		P(b)
1, 8:	10.	$\neg Q(a) \vee R(a)$		$Q(a) \rightarrow R(a)$
1, 9:	11.	$\neg Q(b) \vee R(b)$		$Q(b) \rightarrow R(b)$
3,10:	12.	$\neg N(a) \vee R(a)$		$N(a) \rightarrow R(a)$
3,11:	13.	$\neg N(b) \vee R(b)$		$N(b) \rightarrow R(b)$
6,13:	14.	R(b)		R(b)
7,14:	15.			

Το αν θα τοποθετήσουμε το θετικό λεκτικό στην μια ή στην άλλη άκρη της φράσης είναι θέμα του ποια κατεύθυνση είναι περισσότερο αποδοτική. Μπορούμε για σύγκριση να δούμε ένα παράδειγμα. Υποθέτουμε ότι έχουμε τις παρακάτω προτάσεις:

1. έντομο(X) → ζώο(X)
2. θηλαστικό(X) → ζώο(X)
3. μερμήγκι(X) → έντομο(X)
4. μέλισσα(X) → έντομο(X)
5. αράχνη(X) → έντομο(X)
6. λιοντάρι(X) → θηλαστικό(X)
7. τίγρης(X) → θηλαστικό(X)
8. ζέμπρα(X) → θηλαστικό(X)

Εάν ξέρουμε ότι η ντόλυ είναι μια ζέμπρα, μπορούμε να αποδείξουμε ότι η ντόλυ είναι ζώο;

9. ζέμπρα(ντόλυ)

10. ζώο(ντόλυ) →

8, 9:11. θηλαστικό(ντόλυ)

2,11:12. ζώο(ντόλυ)

10,12:13.

Βλέπουμε λοιπόν ότι με λίγα λογικά βήματα καταλήγουμε στην κενή φράση. Σ' αυτή την περίπτωση λέμε ότι ο **χώρος αναζήτησης** (search space) είναι μικρός. Αν όμως αντιστρέψουμε τις προτάσεις και τις κάνουμε:

1. ζώο(Χ) ← έντομο(Χ)
2. ζώο(Χ) ← θηλαστικό(Χ)
3. έντομο(Χ) ← μερμήγκι(Χ)
4. έντομο(Χ) ← μέλισσα(Χ)
5. έντομο(Χ) ← αράχνη(Χ)
6. θηλαστικό(Χ) ← λιοντάρι(Χ)
7. θηλαστικό(Χ) ← τίγρης(Χ)
8. θηλαστικό(Χ) ← ζέμπρα(Χ)
9. ζέμπρα(ντόλυ)
10. ← ζώο(ντόλυ)

1,10:11. ← έντομο(ντόλυ)

2,10:12. ← θηλαστικό(ντόλυ)

11,3:13. ← μερμήγκι(ντόλυ)

11,4:14. ← μέλισσα(ντόλυ)

11,5:15. ← αράχνη(ντόλυ)

12,6:16. ← λιοντάρι(ντόλυ)

12,7:17. ← τίγρης(ντόλυ)

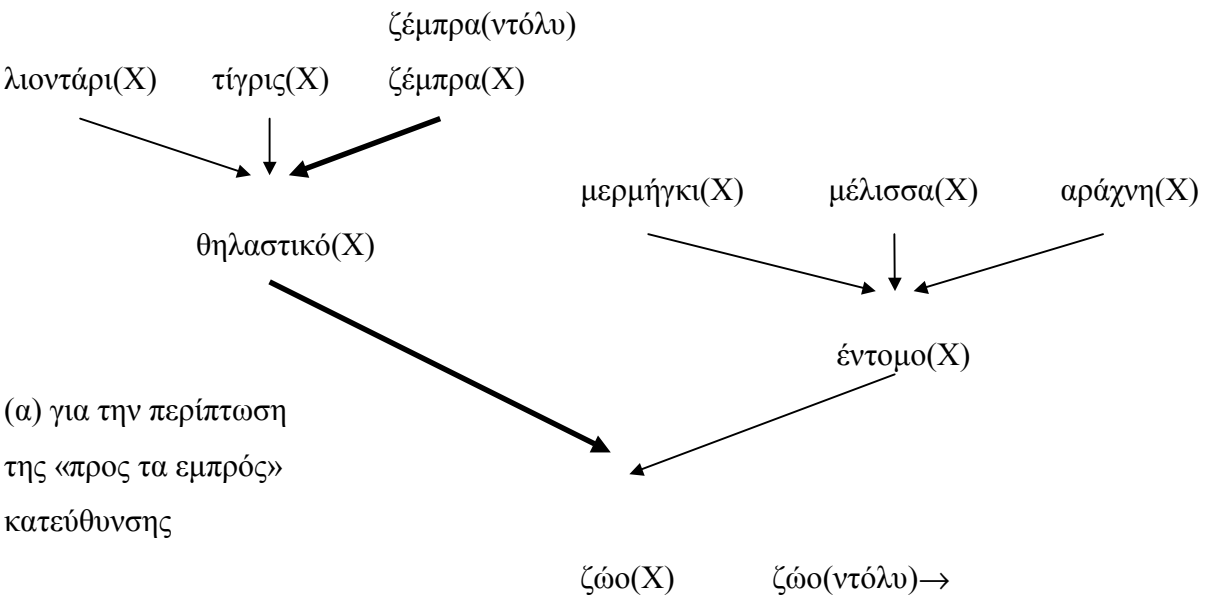
12,8:18. ← ζέμπρα(ντόλυ)

9,18:19.

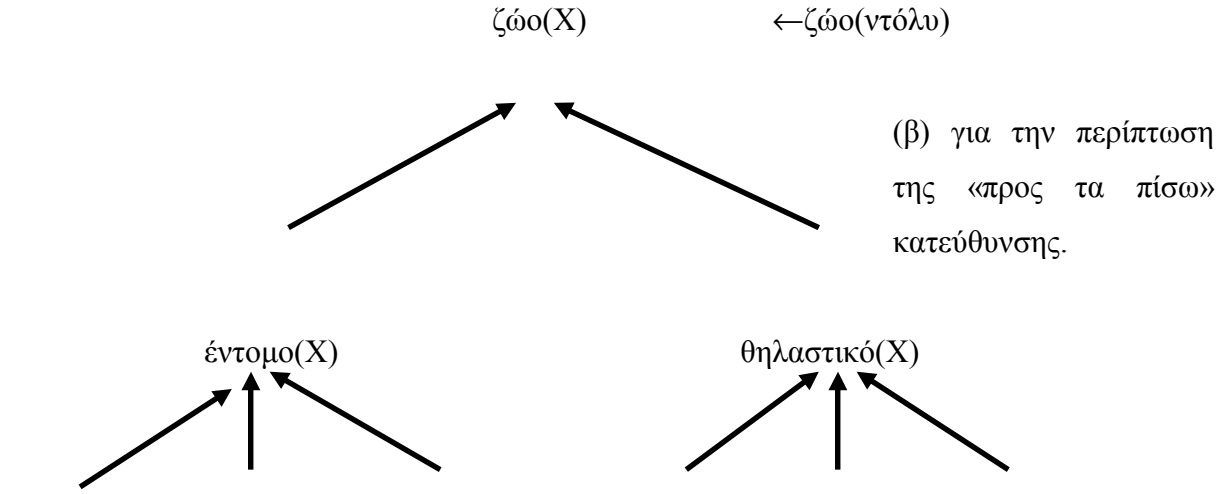
Βλέπουμε λοιπόν ότι σ' αυτή την περίπτωση χρειάστηκε πολύ περισσότερα λογικά βήματα για να φτάσουμε στην κενή φράση. Η διαφορά οφείλεται σ' αυτό που ονομάζεται συντελεστής διακλάδωσης (branching factor) σε κάθε μια από τις περιπτώσεις. Ο συντελεστής αυτός φαίνεται σχηματικά στο παρακάτω σχήμα. Στην πρώτη περίπτωση το δεδομένο ζέμπρα(ντόλυ) μας κατευθύνει σωστά προς την απάντηση ζώο(ντόλυ). Είναι μια στρατηγική που κατευθύνεται από τα δεδομένα (data-driven). Στην περίπτωση (β) ο

συντελεστής διακλάδωσης είναι μεγάλος και το γεγονός ότι η ντόλυ είναι ζέμπρα είναι κατά κάποιον τρόπο «κρυμμένο» από τη διαδικασία απόδειξης. Θα αποκαλυφθεί μόνο όταν η διαδικασία απόδειξης αναρωτηθεί για το αν ισχύει η φράση ζέμπρα(ντόλυ). Μέχρι την στιγμή εκείνη η πληροφορία αυτή φαίνεται άσχετη καθώς αναρωτιόμαστε για άλλα πράγματα π.χ. αν έντομο(ντόλυ), θηλαστικό(ντόλυ), κ.λπ.

Υπάρχουν βέβαια προβλήματα στα οποία η «προς τα πίσω» κατεύθυνση που οδηγείται από τον στόχο (goal driven) είναι πιο αποδοτική. Αυτό εξαρτάται από τον τύπο των προτάσεων.



(α) για την περίπτωση της «προς τα εμπρός» κατεύθυνσης



(β) για την περίπτωση της «προς τα πίσω» κατεύθυνσης.

Αν θεωρήσουμε τα θετικά μοναδιαία λεκτικά σαν τα δεδομένα (τα γεγονότα) τότε η προς τα εμπρός συμπερασματολογία λέγεται και ‘κατευθυνόμενη από τα δεδομένα’ (data-driven) και έχει το νόημα ότι εφόσον τηρούνται κάποιες συνθήκες τότε μπορούμε να βγάλουμε κάποια συμπεράσματα. Ετσι η φράση $\neg\Phi_1 \vee ... \vee \neg\Phi_n \vee \Psi$ που γίνεται $\Phi_1, ..., \Phi_n \Rightarrow \Psi$ ερμηνεύεται σαν ‘εάν ισχύουν τα $\Phi_1, ..., \Phi_n$ τότε ισχύει και το Ψ’.

Στην προς τα πίσω συμπερασματολογία θεωρούμε τα αρνητικά μοναδιαία λεκτικά σαν στόχους. Γι’ αυτό η συμπερασματολογία αυτή ονομάζεται «κατευθυνόμενη από τους στόχους» (goal driven). Η φράση $\neg\Phi_1 \vee ... \vee \neg\Phi_n \vee \Psi$ που γίνεται $\Psi \Leftarrow \Phi_1, \Phi_2, ..., \Phi_n$ ερμηνεύεται σαν “για να αποδείξω τον στόχο Ψ αρκεί να αποδείξω τους στόχους $\Phi_1, ..., \Phi_n$.”

5.12 Σειριακή ικανοποίηση περιορισμών

Η σειριακή ικανοποίηση περιορισμών (Sequential Constraint Satisfaction) είναι μια χρήση της ταξινομημένης αρχής της απόφασης, για να λύσει ένα σημαντικό πρόβλημα ερωταπαντήσεων. Οι ερωτήσεις είναι και εδώ συζεύξεις θετικών λεκτικών, αλλά το σύνολο των αξιωμάτων αποτελείται ολοκληρωτικά από θετικά λεκτικά που έχουν μόνο σταθερές.

Παράδειγμα:

γονέας(πέτρος, γιώργος)
γονέας(δήμητρα, γιώργος)
γονέας(άννα, δημήτρης)
γονέας(κώστας, δημήτρης)
γονέας(δημήτρης, νίκος)
γονέας(ελένη, νίκος)
γονέας(ελένη, μαρία)
επάγγελμα(γιώργος, μαθηματικός)
επάγγελμα(πέτρος, φυσικός)
επάγγελμα(δήμητρα, φιλόλογος)
επάγγελμα(άννα, βιοτέχνης)
επάγγελμα(κώστας, βιοτέχνης)
επάγγελμα(δημήτρης, υπάλληλος)
επάγγελμα(ελένη, φωτογράφος)
επάγγελμα(νίκος, υπάλληλος)
φτωχός(νίκος)

όπου
γονέας (X,Y) σημαίνει
ο/η X είναι γονέας
του/της Y.

Ας υποθέσουμε ότι θέλουμε να βρούμε τους γονείς και το επάγγελμα των φτωχών ατόμων που υπάρχουν στα δεδομένα. Μπορούμε να ρωτήσουμε

$\text{γονέας}(X,Y) \wedge \text{επάγγελμα}(Y,Z) \wedge \text{φτωχός}(Y)$ (1)
ή $\text{επάγγελμα}(Y,Z) \wedge \text{γονέας}(X,Y) \wedge \text{φτωχός}(Y)$ (2)
ή $\text{επάγγελμα}(Y,Z) \wedge \text{φτωχός}(Y) \wedge \text{γονέας}(X,Y)$ (3)
. . .
ή $\text{φτωχός}(Y) \wedge \text{επάγγελμα}(Y,Z) \wedge \text{γονέας}(X,Y)$ (4)

Η σειρά των λεκτικών στην ερώτηση που κάνουμε (στην φράση που πρέπει να αποδειχθεί δηλαδή), παίζει μεγάλο ρόλο στην απόδοση του συστήματος. Ας υποθέσουμε λοιπόν ότι έχουμε τόσες πολλές προτάσεις που χρειαζόμαστε μια βάση δεδομένων για να τις

αποθηκεύσουμε. Σ' αυτή την βάση δεδομένων υπάρχουν στοιχεία για 100 ανθρώπους άρα 100 προτάσεις της μορφής επάγγελμα (..., ...) και (όλοι έχουν γονείς) 200 προτάσεις της μορφής γονέας (..., ...). Απ' αυτούς βέβαια δεν είναι όλοι φτωχοί, αλλά ας πούμε οι 10, οπότε υπάρχουν 10 προτάσεις της μορφής φτωχός (...).

Εάν ρωτήσουμε με την μορφή (1) πρέπει να εξετάσουμε και τις 200 προτάσεις της μορφής γονέας (X,Y) και για κάθε μια από αυτές από μια φορά για το επάγγελμα του Y και για τον καθένα Y θα εξετάσουμε αν είναι φτωχός.

Εάν ρωτήσουμε με την μορφή (2) εξετάζουμε μία-μία τις προτάσεις επάγγελμα (Y,Z) και για κάθε Y τους γονείς του, άρα δύο προτάσεις της μορφής γονέας(X,Y), και για κάθε Y εξετάζουμε αν είναι φτωχός. Πάλι κάνουμε 100 x 2 - 200 βήματα.

Με την ίδια λογική για την ερώτηση με την μορφή (3) κάνουμε 100 βήματα (όσες οι προτάσεις επάγγελμα (X,Y)), ενώ για την ερώτηση με την μορφή (4) μόνο 20 βήματα! Υπάρχουν 10 φτωχοί που έχουν δυο γονείς ο καθένας!!!

Το παράδειγμα αυτό υποδηλώνει έμμεσα ένα ευρεστικό κανόνα (heuristic rule) για προβλήματα σειριακής ικανοποίησης περιορισμών. Ο κανόνας αυτός λέγεται 'φτηνότερος πρώτος κανόνας' (cheapest first rule) και ο οποίος λέει ότι: Τα λεκτικά της ερώτησης πρέπει να δοκιμασθούν κατά σειρά μικρότερου συνόλου λύσεων.

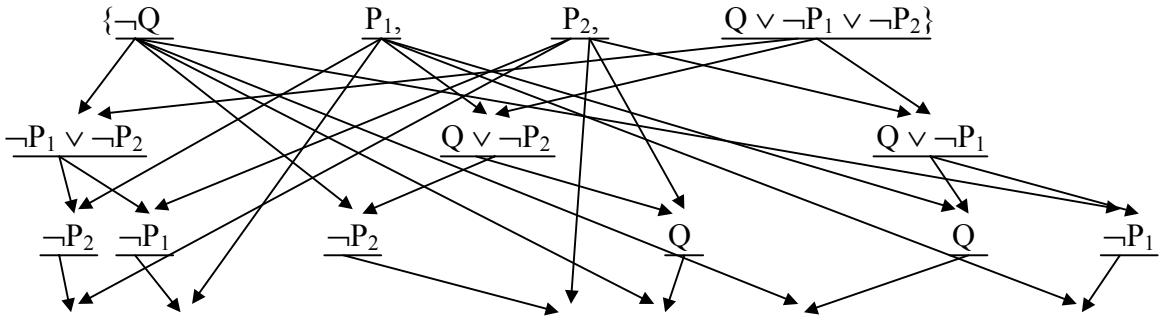
Και αυτός βέβαια ο κανόνας δεν ισχύει πάντα, γι' αυτό έχει αναπτυχθεί αρκετή θεωρία, θεωρήματα, κ.λπ. για το πώς να βρίσκει κανείς την καλύτερη δυνατή σειρά.

5.13 Ταξινομημένη αρχή της απόφασης στην είσοδο

Είναι συνδυασμός και των δύο στρατηγικών (ταξινομημένη αρχή της απόφασης και αρχή της απόφασης στην είσοδο). Η στρατηγική της ταξινομημένης αρχής της απόφασης στην είσοδο (Ordered Input Resolution) έχει μεγάλη σημασία γιατί αποτελεί την βάση της συμπερασματολογίας της Prolog.

Η Prolog (Programming in Logic) είναι μια σύγχρονη γλώσσα λογικού προγραμματισμού που διαθέτει όμως και στοιχεία πέραν της λογικής έτσι ώστε γίνεται μια γλώσσα προγραμματισμού γενικής χρήσης με ενδιαφέρουσες εφαρμογές.

Με την αρχή της απόφασης στην είσοδο, όταν προσπαθούμε να φτάσουμε στην κενή φράση από μια φράση n-λεκτικών, ακόμη και όταν κάθε λεκτικό μπορεί να απαλειφθεί με ένα μόνο τρόπο, υπάρχουν n! παραγωγές της κενής φράσης.



3! = 6 παραγωγές του

Αυτό είναι αντιαποδοτικό. Θα μας αρκούσε να βρίσκαμε μόνο μια φορά την κενή φράση. Εάν περιορίσουμε την τυχαία επιλογή των λεκτικών στα οποία θα εφαρμοσθεί η αρχή της

απόφασης, και ορίσουμε μια αυστηρή σειρά π.χ. από αριστερά προς τα δεξιά, τότε πετυχαίνουμε αυτό που θέλουμε. Οπως φαίνεται και στο σχήμα που ακολουθεί, πολύ γρήγορα βρίσκουμε την κενή φράση.

Με τον τρόπο αυτό λειτουργεί και η Prolog. Στην Prolog έχουμε τρεις τύπους προτάσεων: Κανόνες (rules), γεγονότα (facts) και την ερώτηση που κάνουμε στο σύστημα. Οι τύποι αυτοί έχουν την εξής μορφή:

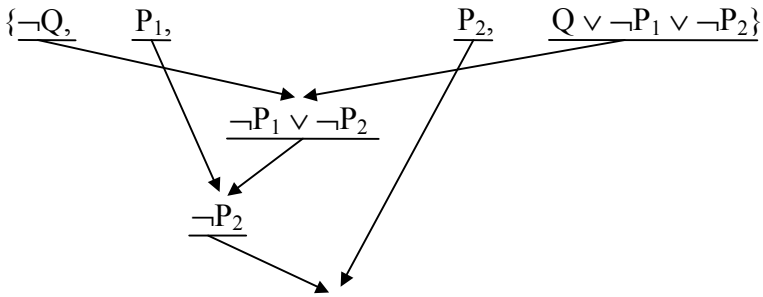
Q: - $P_1, P_2, ..., P_n$. (κανόνας)
P_1. (γεγονός)
?Q (ερώτηση)

Οι ισοδύναμες φράσεις είναι

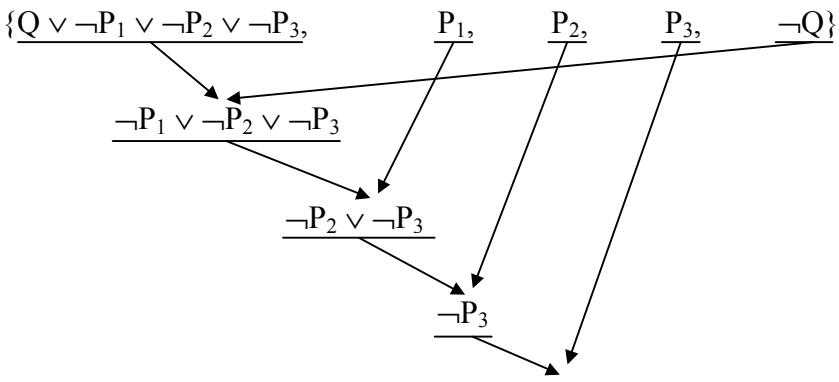
$Q \vee \neg P_1 \vee ... \vee \neg P_n$ (κανόνας)
P_1 (γεγονός)
$\neg Q$ (ερώτηση)

Για παράδειγμα, μπορούμε στην Prolog να έχουμε τα εξής:

Q: - P_1, P_2, P_3.
P_1.
P_2. Η ερώτηση ?Q είναι και ο στόχος (goal).
P_3.
?Q



Για να αποδείξουμε το ?Q βρίσκουμε τον πρώτο κανόνα και κάνουμε στόχο την πρώτη συνθήκη P_1. Η συνθήκη P_1 όμως αποδεικνύεται γιατί υπάρχει το γεγονός P_1. Στην συνέχεια κάνουμε στόχο την δεύτερη συνθήκη P_2, η οποία επίσης αποδεικνύεται γιατί υπάρχει το γεγονός P_2. Ομοια αποδεικνύεται και η συνθήκη P_3. Αφού αποδείξαμε όλες τις συνθήκες (υποστόχους, subgoals) του πρώτου κανόνα, έχουμε αποδείξει και τον στόχο Q, άρα απαντήσαμε και στην ερώτηση. Με ταξινομημένη αρχή της απόφασης στην είσοδο θα είχαμε την παρακάτω απόδειξη.



Που είναι η ίδια ακριβώς διαδικασία!!!

Ομως υπάρχουν μερικά σημεία διαφοράς. Οι κανόνες της Prolog είναι κανόνες ‘προς τα πίσω’ ή ‘καθοδηγούμενοι από τους στόχους’. Η συμπερασματολογία της λοιπόν έχει αυτό το επιπρόσθετο χαρακτηριστικό. Αυτή η ιδιότητα μπορεί να βγει και από την εξής παρατήρηση: Στην Prolog όλοι οι κανόνες έχουν την μορφή

$$Q: - P_1, ..., P_n$$

που είναι ισοδύναμη με την φράση

$$Q \vee \neg P_1 \vee ... \vee \neg P_n.$$

Στα αριστερά δηλαδή κάθε κανόνα εμφανίζεται ένα θετικό λεκτικό, το οποίο μπορεί να απλοποιηθεί με τη χρήση της αρχής της απόφασης, μόνο με ένα αρνητικό λεκτικό. Αρνητικά λεκτικά είναι οι στόχοι της ερώτησης ($\neg Q$) ή άλλοι αργότερα παραγόμενοι υποστόχοι ($\neg P_1$ κ.λπ.). Ετσι η μορφή των κανόνων της Prolog μας ωθεί να αναζητούμε κανόνες που ταιριάζουν με τους στόχους, να δημιουργούμε νέους υποστόχους (τα $\neg P_1$ $\neg P_2$ κ.λπ.) έως ότου όλοι οι στόχοι συναντήσουν ένα γεγονός (μοναδιαίο θετικό λεκτικό) και επιλυθούν.

Αντίθετα η ταξινομημένη αρχή της απόφασης στην είσοδο μπορεί να έχει προτάσεις της μορφής $\neg P_1 \vee \neg P_2 \vee Q$ που, εφόσον έχουμε αριστερά προς τα δεξιά κίνηση, είναι κανόνες συμπερασματολογίας. Η ταξινομημένη λοιπόν αρχή της απόφασης στην είσοδο είναι γι’ αυτό τον λόγο πιο γενική.

Ομως δεν είναι μόνο αυτός ο λόγος της εξειδίκευσης στην συμπερασματολογία της Prolog. Η Prolog ‘ψάχνει’ υποψήφιες προτάσεις από πάνω προς τα κάτω. Μεγαλύτερη προτεραιότητα έχουν οι προτάσεις που βρίσκονται πιο ψηλά στην λίστα των προτάσεων του λογικού προγράμματος. Αυτή η λεπτομέρεια έχει τόση πολλή σημασία, που υπάρχουν προγράμματα Prolog, που αν τους αλλάξουμε την σειρά των προτάσεων δεν τρέχουν σωστά!!!

Περισσότερα στοιχεία για την συμπερασματολογία της Prolog, τις στρατηγικές ελέγχου, και την Prolog σαν γλώσσα προγραμματισμού θα δούμε σε επόμενο κεφάλαιο.

6

Μη μονότονη συμπερασματολογία

6.1 Εισαγωγή

Στα προηγούμενα κεφάλαια πήραμε μια ιδέα για το πώς μπορούμε να αναπαριστάνουμε και να επεξεργαζόμαστε την γνώση με την προτασιακή λογική, την κατηγορική λογική α' τάξης, την Prolog κ.λπ.

Με βάση αυτά μπορούμε να πούμε πως σε ένα τυπικό σύστημα Τεχνητής Νοημοσύνης που χρησιμοποιεί την κατηγορική λογική σαν βάση, εκφράζουμε την γνώση για το θέμα που επεξεργαζόμαστε, σαν ένα σύνολο προτάσεων που κατά κάποιο τρόπο αποτελεί το βασικό σύνολο των πεποιθήσεων (base set of beliefs) του συστήματος. Ας συμβολίζουμε αυτό το σύνολο με το γράμμα Δ.

Το σύνολο Δ ονομάζεται και βάση γνώσης (Knowledge-base) σε αντίθεση με μια συλλογή δεδομένων που ονομάζεται βάση δεδομένων (Database).

Για να απαντήσει ερωτήσεις, ή για να εξάγει τη γνώση της υπάρχουσας βάσης γνώσης, το σύστημα πρέπει να αποφασίσει αν μια πρόταση Φ είναι λογική συνέπεια της βάσης Δ. Ενας τρόπος να γίνει αυτό είναι η χρήση της αρχής της απόφασης στο σύνολο $\Delta \wedge \neg\Phi$ έως ότου παραχθεί η κενή πρόταση.

Τα πλεονεκτήματα του μοντέλου αυτού είναι η αυστηρή και καλά-ορισμένη μεθοδολογία, η ακρίβεια και η συνέπεια των συλλογισμών και η ευκολία με την οποία μπορούμε να εκφράσουμε προτάσεις της φυσικής γλώσσας σε καλοσχηματισμένες προτάσεις (Wffs) της κατηγορικής λογικής.

Υπάρχουν όμως και μερικά μειονεκτήματα, ή καλύτερα περιορισμοί. Οι μάλλον πιο σημαντικοί είναι οι παρακάτω:

Α. Η γλώσσα του κατηγορικού λογισμού α' τάξης δεν μπορεί βέβαια να «κωδικοποιήσει» όλη την γνώση που διαθέτουμε. Μπορούμε βέβαια να «αποτυπώσουμε» λογικούς συλλογισμούς, κανόνες, γεγονότα κ.λπ. αλλά υπάρχουν και ποιοτικές πτυχές της γνώσης που δεν τις πλαισιώνουμε.
Παράδειγμα είναι οι εξαιρέσεις (exceptions). Ξέρουμε για παράδειγμα ότι όλα τα πουλιά πετούν:

$$(\forall X)\ (\text{πουλί}(X) \rightarrow \text{πετάει}(X))$$

Ομως οι πιγκουΐνοι είναι πουλιά που δεν πετούν:

$$(\forall X)\ (\text{πουλί}(X) \wedge \neg\text{πιγκουΐνος}(X) \rightarrow \text{πετάει}(X))$$

Υπάρχουν όμως και άλλα πουλιά που δεν πετούν : Τα μωρά πουλιά, τα νεκρά πουλιά, τα πουλιά που τους έχουν σπάσει τα φτερά κ.λπ.

Στην πραγματικότητα σχεδόν όλες οι προτάσεις με καθολικούς ποσοδείκτες έχουν εξαιρέσεις. Οι εξαιρέσεις αυτές μπορεί να είναι προσωρινές. Χρειαζόμαστε λοιπόν κάποιο κανόνα συμπερασματολογίας που να μπορεί να βγάζει «προσωρινά» συμπεράσματα, τα οποία μάλιστα με την άφιξη νέων πληροφοριών να υπάρχει η δυνατότητα να αναθεωρηθούν.

Β. Οι κανόνες συμπερασματολογίας όπως η αρχή της απόφασης ποτέ δεν παράγουν κάτι νέο, «νέα γνώση» για τον κόσμο. Ο λόγος γι' αυτό είναι ο ορισμός της λογικής συνέπειας. Αν η πρόταση Φ είναι λογική συνέπεια του συνόλου Δ, τότε η Φ δεν αναπαριστά κάτι νέο, που δεν το αναπαριστά το Δ.

Η κλασσική λογική βέβαια βοηθάει ώστε ένα μεγάλο μέρος της γνώσης να αναπαριστάνεται έμμεσα μέσω των προτάσεων του Δ και της συμπερασματολογίας πάνω στο Δ, αλλά προσθέτοντας νέες προτάσεις στο Δ απλώς αυξάνουμε την ποσότητα της γνώσης που αναπαριστάνουμε.

Θα θέλαμε όμως να μπορούμε να αναθεωρούμε και την αντίληψη του κόσμου μας, με νέα στοιχεία, έτσι που το μοντέλο της γνώσης να ανταποκρίνεται πιο καλά στα φαινόμενα του πραγματικού κόσμου.

Γ. Ενα άλλο σημαντικό θέμα είναι η ακρίβεια. Τα κατηγορήματα παίρνουν εξ ορισμού δύο δυνατές τιμές: Αλήθεια (T) ή Ψέμα (F). Πολλές φορές όμως οι πληροφορίες μας για ένα θέμα είναι αβέβαιες, ανακριβείς, ασαφείς. Η αναπαράσταση και επεξεργασία τέτοιων πληροφοριών ξεφεύγει από τα όρια της κλασσικής λογικής.

Δ. Ενας άλλος περιορισμός βρίσκεται σ' αυτή καθεαυτή την χρήση λογικών προτάσεων για την αναπαράσταση της γνώσης. Τα αντικείμενα (objects) του γνωστικού πεδίου είναι συνήθως οι παράμετροι των κατηγορημάτων, τα κατηγορήματα είναι σχέσεις αντικειμένων και οι προτάσεις (clauses) είναι γενικότερες συσχετίσεις. Με τον τρόπο αυτό η γνώση για τα διάφορα αντικείμενα αναπαριστάνεται με ένα «κατανεμημένο» τρόπο. Υπάρχουν μοντέλα αναπαράστασης γνώσης πιο δομημένα, όπου η γνώση για ένα αντικείμενο είναι συγκεντρωμένη, τα αντικείμενα ομαδοποιούνται σε ιεραρχίες ή συγγενικές ομάδες με βάση το ένα χαρακτηριστικό ή το άλλο κ.λπ. Εκεί η αναπαράσταση και επεξεργασία της γνώσης έχει τελείως διαφορετικό νόημα.

Στο κεφάλαιο αυτό θα προσπαθήσουμε να ξεπεράσουμε μερικούς από τους παραπάνω περιορισμούς. Θα παραθέσουμε μερικούς άλλους τρόπους, ή και κανόνες συμπερασματολογίας που αντιμετωπίζουν μερικά από τα παραπάνω προβλήματα. Η παράθεση αυτή βρίσκεται και πάλι μέσα στα πλαίσια της λογικής. Σε επόμενα κεφάλαια θα διερευνήσουμε και άλλες όψεις του θέματος «αναπαράσταση και επεξεργασία της γνώσης».

Με τους κανόνες συμπερασματολογίας (inference rules) που θα περιγράψουμε τα συμπεράσματα της διαδικασίας απόδειξης θα εξαρτώνται από τις προτάσεις που περιέχονται στην βάση γνώσης. Με διαγραφή κάποιων προτάσεων ή εισαγωγή νέων θα «αλλάζουν» και τα συμπεράσματα. Για τον λόγο αυτό χρησιμοποιείται και ο όρος «μη μονοτονική» (non monotonic) συμπερασματολογία.

6.2 Η υπόθεση του κλειστού κόσμου (The Closed-World Assumption, CWA)

Ας υποθέσουμε ότι έχουμε ένα σύνολο Σ το οποίο αποτελείται από κατηγορήματα που δείχνουν με ποιές χώρες συνορεύει η χώρα μας.

Σ = {γειτονεύει(Ελλάδα,Αλβανία)
γειτονεύει(Ελλάδα,Γιουγκοσλαβία),
γειτονεύει(Ελλάδα,Βουλγαρία),
γειτονεύει(Ελλάδα,Τουρκία)}

Από ένα τέτοιο σύνολο στη σημασιολογία της ΚΛ η παρακάτω πρόταση δεν αληθεύει:

Φ = ¬γειτονεύει(Ελλάδα,Αμερική)

Ο κοινός νους όμως με δεδομένα τα γεγονότα του Σ μπορεί να συμπεράνει ότι η Ελλάδα δεν γειτονεύει με την Αμερική. Ομως Σ|≠Φ !!! Δηλαδή κάτι τέτοιο δεν είναι λογικό συμπέρασμα του Σ. Αρα ο ανθρώπινος νους (μερικές φορές) σκέπτεται με μη ορθό τρόπο (ως προς την σημασιολογία της ΚΛ). Δημιουργείται λοιπόν εδώ ένα σοβαρό θέμα. Πως θα μοντελοποιήσουμε αυτή την ασυνέπεια της ΚΛ ως προς τον κοινό νου ; Την λύση μας τη δίνει η υπόθεση του κλειστού κόσμου.

Εάν συμβολίσουμε με Δ το σύνολο των αρχικών αξιωμάτων μιας βάσης γνώσης (proper axioms) τότε το σύνολο των προτάσεων που είναι λογικά συμπεράσματα του Δ, ονομάζεται θεωρία του Δ «κλεισμένη» κάτω από το λογικό συμπέρασμα. Αυτό συμβολίζεται ως εξής:

Φ ∈ T[Δ] εάν και μόνο εάν Δ ⊨Φ, όπου
Φ είναι μια πρόταση,
Δ το σύνολο των αρχικών αξιωμάτων
T[Δ] η θεωρία (Theory) που παράγεται από το Δ κάτω από το λογικό συμπέρασμα
Δ ⊨Φ σημαίνει η πρόταση Φ είναι λογικό συμπέρασμα της βάσης Δ.

Θυμίζουμε ότι ένα κατηγόρημα που περιέχει μόνο συναρτησιακά σύμβολα και σταθερές, δηλ. δεν περιέχει μεταβλητές, λέγεται **στοιχειώδες** ή αρχικοποιημένο (ground).

Στην υπόθεση του κλειστού κόσμου υποθέτουμε ότι εκτός του συνόλου Δ έχουμε και ένα σύνολο '**υποθετικών πεποιθήσεων**' (assumed beliefs) Δasm, τέτοιο ώστε:

¬P ∈ Δasm εάν και μόνο εάν το στοιχειώδες άτομο P ∉ T[Δ].

Ετσι λέμε ότι:

Φ ∈ CWA[Δ] αν και μόνο αν {Δ∪Δasm} ⊨Φ.

Δηλαδή επεκτείνουμε κατά κάποιο τρόπο την θεωρία Τ[Δ].
Παράδειγμα 6.1.

$\Delta = \{P(a), P(a) \rightarrow Q(a), P(b)\}$

Παρατηρούμε ότι το Q(b) δεν μπορεί να αποδειχθεί από το Δ, έτσι μπορούμε να πούμε ότι

$\neg Q(b) \in \Delta asm$ γιατί $Q(b) \notin T[\Delta]$

Αρα $\Delta asm = \{\neg Q(b)\}$.

Παράδειγμα 6.2

Θέλουμε σε μια βάση Δ να αποτυπώσουμε τις γειτονικές χώρες. Ετσι γράφουμε:

γειτονική (ελλάδα, βουλγαρία)
γειτονική (αμερική, καναδάς)
γειτονική (μεξικό, γουτεμάλα)
⋮

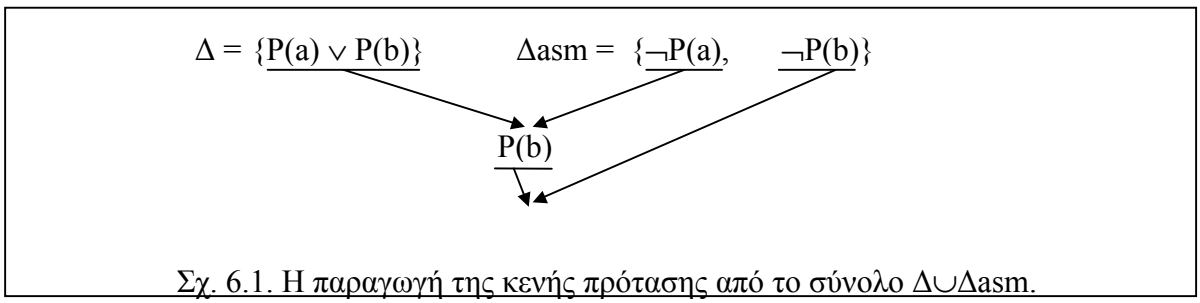
Ομως θέλουμε ταυτόχρονα να μπορούμε να συμπεράνουμε ότι αν δύο χώρες δεν εγγράφονται γειτονικές, τότε είναι μη γειτονικές! Αυτό μπορεί να γίνει με το CWA. Δηλαδή αν ρωτήσουμε ?γειτονική(ιαπωνία, περσία), εφόσον το σύστημα δεν μπορεί να αποδείξει ότι είναι γειτονικές μπορεί να συμπεράνει:

¬γειτονική(ιαπωνία, περσία).

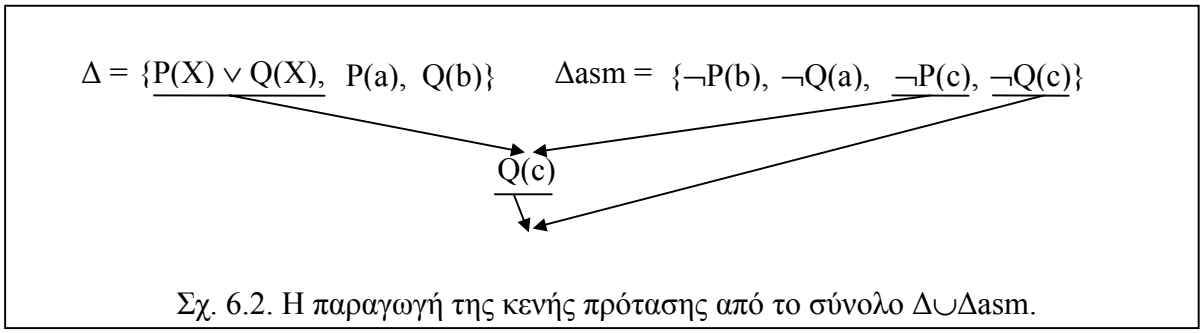
Στην υπόθεση του κλειστού κόσμου, δεν είναι απαραίτητο, ούτε πολλές φορές εφικτό, να καταγράψουμε το Δasm. Γι' αυτό προσπαθούμε να βρούμε αλγορίθμους για να συμπεραίνουμε το Δasm όταν χρειάζεται. Μια τέτοια λύση δίδεται στην Prolog με το ειδικό κατηγόρημα not.

Ενδιαφέρουσες παρατηρήσεις για το CWA

- Η συμπερασματολογία με CWA εξαρτάται από τα συντακτικά χαρακτηριστικά του Δ. Ετσι αν το $\Delta = \{\neg P(a), \neg P(a) \rightarrow \neg Q(a), \neg P(b)\}$ τότε το $\Delta asm = \{\ \}$, δηλαδή *δεν* εισάγουμε στο Δasm το Q(b) επειδή το ¬Q(b) δεν μπορεί να αποδειχθεί από το Δ. Κάντε σύγκριση με το παράδειγμα 6.1. Ας μην ξεχνάμε ότι στο Δasm εισάγουμε μόνο αρνητικά (negated) άτομα.
- Πολλές φορές γεννάται το ερώτημα της συνέπειας (consistency) του Δ∪Δasm. Αν για παράδειγμα $\Delta = \{P(a) \vee P(b)\}$ τότε $\Delta asm = \{\neg P(a), \neg P(b)\}$ γιατί ούτε το P(a) ούτε το P(b) μπορούν να αποδειχθούν από το Δ. Ομως Δ∪Δasm παράγει την κενή φράση από μόνο του, χωρίς να χρειάζεται να θέσουμε μια πρόταση προς απόδειξη (Σχ. 6.1.)



Σχ. 6.1. Η παραγωγή της κενής πρότασης από το σύνολο Δ∪Δasm.



Σχ. 6.2. Η παραγωγή της κενής πρότασης από το σύνολο Δ∪Δasm.

- Η συνέπεια του Δ∪Δasm εξαρτάται και από το σύνολο των στοιχειωδών όρων (ground terms, σταθερές και συναρτήσεις που περιέχουν μόνο σταθερές), που θεωρούμε ότι περιέχει ο κόσμος αναφοράς.

Παράδειγμα 6.4

Για το σύνολο Δ = {P(X) ∨ Q(X), P(a), Q(b)} εάν θεωρήσουμε ότι τα μόνα δυνατά αντικείμενα είναι τα a και b, τότε Δasm = {¬P(b), ¬Q(a)} και Δ∪Δasm είναι συνεπές.

Αν όμως θεωρήσουμε ότι τα δυνατά αντικείμενα είναι a, b, c τότε Δasm = {¬P(b), ¬Q(a), ¬P(c), ¬Q(c)} και Δ∪Δasm είναι ασυνεπές (βλέπε Σχ. 6.2.).

Για να ξεπεράσουμε αυτή τη δυσκολία του καθορισμού των στοιχειωδών αντικειμένων του κόσμου αναφοράς, κάνουμε άλλη μια υπόθεση, που αναφέρεται στην βιβλιογραφία σαν «Domain Closure Assumption» ή DCA. Θα μπορούσαμε να την μεταφράσουμε σαν Υπόθεση Κλειστότητας του Κόσμου Αναφοράς που αναλύεται σε επόμενη παράγραφο.

- **CWA με αναφορά σε ένα κατηγόρημα P**

Μπορούμε να εξειδικεύσουμε το CWA, εφαρμόζοντάς το μόνο σε ένα κατηγόρημα P της βάσης Δ. Αυτό σημαίνει ότι το Δasm θα περιέχει αρνητικά άτομα του P μόνο.

Δ = {Q(X) → P(X), Q(a), R(b) ∨ P(b)}
Δasm = {¬P(b)}
Δ∪Δasm συνεπές.

- **CWA με αναφορά σε ένα σύνολο κατηγορημάτων**

Μπορούμε ακόμη να εφαρμόσουμε το CWA σε ένα σύνολο κατηγορημάτων του Δ (υποσύνολο όλων των κατηγορημάτων του Δ). Είναι ενδιαφέρον ότι ενώ το CWA[Δ] μπορεί να είναι συνεπές με αναφορά στο κάθε κατηγόρημα ξεχωριστά, μπορεί να είναι ασυνεπές με αναφορά σε όλα τα κατηγορήματα μαζί.

Δ = {P ∨ Q}, Δasm = {¬P, ¬Q} Δ∪Δasm ασυνεπές, ως προς P, Q
Δ = {P ∨ Q}, Δasm = {¬P} Δ∪Δasm συνεπές, ως προς P
Δ = {P ∨ Q}, Δasm = {¬Q} Δ∪Δasm συνεπές, ως προς Q.

6.3 Υπόθεση Κλειστότητας του Κόσμου Αναφοράς (Domain Closure Assumption DCA)

Η υπόθεση αυτή λέει ότι:

'*Τα μόνο αντικείμενα του κόσμου αναφοράς D είναι εκείνα τα στοιχειώδη αντικείμενα (ground terms) που μπορούν να σχηματισθούν χρησιμοποιώντας τις σταθερές και τα συναρτησιακά σύμβολα του Δ*'.

Παράδειγμα 6.5

$\Delta = \{p(X) \vee q(f(X), p(a), q(f(b))\}$, σταθερές $C = \{a, b\}$, συναρτησιακά $F = \{f\}$

Τα δυνατά αντικείμενα: $P = \{a, b, f(a), f(b), f(f(a)), f(f(b)), ...\}$

Παράδειγμα 6.6

$\Delta = \{p(X) \wedge q(X), p(b), q(a), p(c)\}$, σταθερές $C = \{a, b, c\}$, συναρτησιακά $F = \varnothing$

Τα δυνατά αντικείμενα: $P = \{a, b, c\}$

Αν η βάση αξιωμάτων Δ δεν περιέχει συναρτησιακά σύμβολα, παρά μόνο τις σταθερές t_1, t_2, ..., t_n, (οι οποίες εμφανίζονται σε διάφορα κατηγορήματα του Δ), τότε το DCA μπορεί να εκφρασθεί σαν ένα αξίωμα:

$C = \{t_1, ..., t_n\}, F = \varnothing$ DOMAIN CLOSURE AXIOM

DCA: $(\forall X)\ ((X = t_1) \vee (X = t_2) \vee ... \vee (X = t_n))$

Αν θέλουμε να εισάγουμε και συναρτησιακά σύμβολα το αξίωμα αυτό περιέχει άπειρους όρους, και γι' αυτό πρέπει να παρασταθεί με πρόταση κατηγορικού λογισμού δεύτερης τάξης.

6.4 Υπόθεση των μοναδικών ονομάτων (Unique-names assumption - UNA)

Μια άλλη υπόθεση που κάνουμε είναι η UNA που αναφέρει ότι :

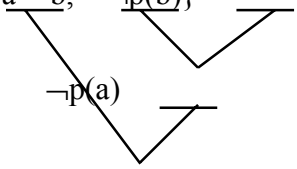
'*Αν δύο στοιχειώδεις όροι δεν μπορούν να αποδειχθούν ίσοι, μπορούν να θεωρηθούν άνισοι*'.

Βέβαια η UNA προϋποθέτει κάποια θεωρία Ισότητας όρων, την οποία δεν έχουμε εισάγει στις σημειώσεις. Θεωρίες για συμπερασματολογία πάνω στην ισότητα (Equality Reasoning) επιτρέπουν την ενοποίηση όρων που είναι σταθερές και συναρτησιακά σύμβολα, όχι μόνο με μεταβλητές, αλλά και με άλλες σταθερές ή συναρτησιακά σύμβολα, εφόσον έχουν αποδειχθεί ίσα (ή θεωρούνται ίσα). Δηλαδή, συστήματα που επιτρέπουν συμπερασματολογία σαν αυτή του παρακάτω παραδείγματος.

Παράδειγμα 6.7

Δ = {P(a), a = b}, θεώρημα: p(b)

Λύση Δ' = {p(a), a = b, ¬p(b)}



Χωρίς τέτοιες θεωρίες η UNA ισχύει από μόνη της (by default), αλλά την αναφέρουμε εδώ για λόγους πληρότητας. Για δύο σταθερές a, b η UNA μπορεί να εκφρασθεί σαν το γεγονός ¬a=b ή a≠b.

$C = \{t_1, ..., t_n\}, F = \varnothing$	UNIQUE NAMES AXIOM
UNA:	$(\forall t_i)\,(\forall t_j)\,(t_i \in C_\wedge\, t_j \in C_\wedge\, i \neq j \rightarrow t_i \neq t_j)$

Μετά από όλες αυτές τις ενδιαφέρουσες παρατηρήσεις για το CWA, μένει το ερώτημα αν μπορούμε να ξέρουμε τίποτα για την συνέπεια του Δ∪Δasm. Αποδεικνύεται το εξής:

'*Εάν η βάση Δ αποτελείται από φράσεις τύπου Horn, και είναι συνεπής, τότε το CWA[Δ] είναι επίσης συνεπές*'.

6.5 Συμπλήρωση Κατηγορημάτων (Predicate Completion)

Ας υποθέσουμε ότι Δ = {P(a)}, δηλαδή P(a) είναι η μοναδική πρόταση του Δ και το a είναι το μοναδικό αντικείμενο που ικανοποιεί το P. Αυτά γράφονται ως εξής :

$$(\forall X)(\, X=a \Rightarrow P(X))$$

Η υπόθεση ότι δεν υπάρχουν άλλα αντικείμενα που ικανοποιούν το P, παρά μόνο το a του P(a) μπορεί να διατυπωθεί με την πρόταση:

$$(\forall X)(\, P(X) \Rightarrow X=a)$$

Η πρόταση αυτή λέγεται πρόταση συμπλήρωσης (completion formula) για το P. Η πρώτη πρόταση υπονοείται από την σημασιολογία της Κατηγορικής λογικής α' τάξης. Η δεύτερη πρόταση λέγεται πρόταση συμπλήρωσης (completion formula) για το P.

Η συμπλήρωση του P στο Δ συμβολίζεται με COMP[Δ;P] και είναι:
$COMP[\Delta;P] \equiv \{(\forall X)\,(\, P(X) \Rightarrow X=a)\} \cup \Delta$

Παράδειγμα 6.8

$\Delta = \{p(a), p(b)\}$
$COMP[\Delta;p] \equiv \{(\forall x)\ (p(x) \rightarrow x=a \vee x=b)\} \cup \Delta$

$\Delta = \{p(a), q(b), q(c)\}$
$COMP[\Delta;p] \equiv \{(\forall x)\ (q(x) \rightarrow x=b \vee x=c)\} \cup \Delta$

Την συμπλήρωση κατηγορημάτων την εφαρμόζουμε σε προτάσεις που μπορούν να μετατραπούν στην μορφή

$$Q_1 \wedge Q_2 \wedge ... \wedge Q_m \Rightarrow P(t_1, t_2, ..., t_n)$$

όπου τα t_i είναι σταθερές, μεταβλητές, συναρτησιακά σύμβολα κ.λπ. Θεωρούμε ακόμη ότι κανένα από τα Q_i δεν είναι P. Τέτοιες προτάσεις τις μετατρέπουμε στην μορφή:

$$X_1=t_1 \wedge X_2=t_2 \wedge ... \wedge X_n=t_n \wedge Q_1 \wedge ... \wedge Q_n \Rightarrow P(X_1, X_2, ..., X_n)$$

Η προηγούμενη πρόταση γράφεται σαν $E \Rightarrow P(X)$.

Συμπλήρωση Κατηγορημάτων: Αλγόριθμος μετατροπής

α) Εστω ότι έχουμε προτάσεις της μορφής:
$q_1 \wedge q_2 \wedge ... \wedge q_m \rightarrow p(t_1, ..., t_n)$ όπου $t_1, ..., t_n$ σταθερές.

β) τις μετατρέπουμε στην παρακάτω μορφή
$(\forall x_1) ... (\forall x_n)\ (x_1=t_1 \wedge x_2=t_2 \wedge ... \wedge x_n=t_n \wedge q_1 \wedge ... \wedge q_m \rightarrow p(x_1, ..., x_n))$
που γράφεται και $(\forall \vec{x})\ (e \rightarrow p(\vec{x}))$

γ)
$$\left.\begin{array}{l}(\forall \vec{x})\ (e_1 \rightarrow p(\vec{x})) \\ (\forall \vec{x})\ (e_2 \rightarrow p(\vec{x})) \\ \vdots \\ (\forall \vec{x})\ (e_k \rightarrow p(\vec{x}))\end{array}\right\} \quad (\forall \vec{x})\ (e_1 \vee e_2 \vee ... \vee e_\kappa \rightarrow p(\vec{x}))$$

δ) Η πρόταση συμπλήρωσης γίνεται:
$(\forall \vec{x})\ (p(\vec{x}) \rightarrow e_1 \vee e_2 \vee ... \vee e_\kappa)$

ε) $COMP[\Delta;p] = \{((\forall \vec{x})\ (p(\vec{x}) \rightarrow e_1 \vee e_2 \vee ... \vee e_\kappa))\} \cup \Delta$

Παράδειγμα 6.9

$\Delta = \{$στρουθοκάμηλος(X) $\Rightarrow$ πουλί(X), πουλί(tweety), $\neg$στρουθοκάμηλος(sam)$\}$

Γράφουμε τις δύο πρώτες προτάσεις ως εξής:

στρουθοκάμηλος(X) $\Rightarrow$ πουλί(X)

$X=tweety \Rightarrow$ πουλί(X)

στρουθοκάμηλος(X) $\vee$ X=tweety $\Rightarrow$ πουλί(X)

Δηλαδή τα μόνα πουλιά είναι οι στρουθοκάμηλοι και ο tweety. Ετσι, αφού υπολογίσουμε την πρόταση συμπλήρωσης, έχουμε

COMP[Δ;πουλί] ≡ Δ∪{πουλί(X) $\Rightarrow$ στρουθοκάμηλος(X) $\vee$ X=tweety}.

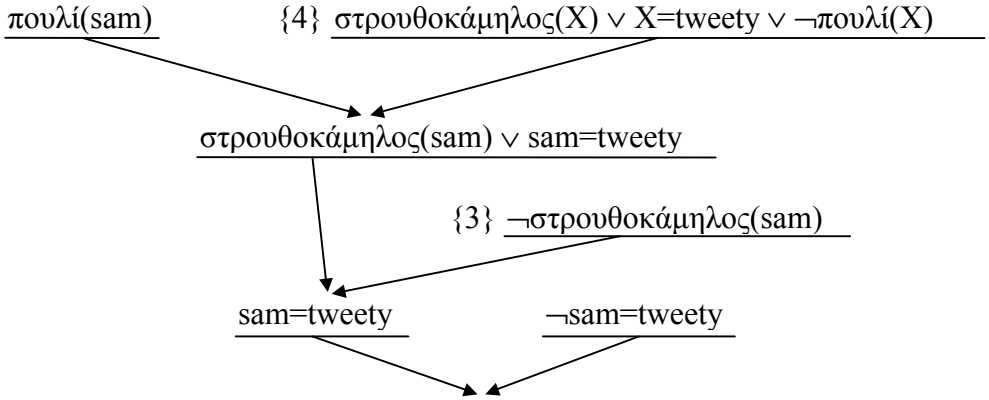
Στο επαυξημένο σύνολο προτάσεων μπορούμε να αποδείξουμε ότι ο sam δεν είναι πουλί! Η απόδειξη ακολουθεί, (αφού μετατρέψουμε τις προτάσεις σε φράσεις):

Δ = {πουλί(X) $\vee$ ¬στρουθοκάμηλος(X) {1},
πουλί(tweety) {2},
¬στρουθοκάμηλος(sam) {3}}

Πρόταση συμπλήρωσης:

στρουθοκάμηλος(X) $\vee$ X=tweety $\vee$ ¬πουλί(X) {4}

Εάν θέλουμε να αποδείξουμε ¬πουλί(sam), εισάγουμε το πουλί(sam) στα αξιώματα, και εφαρμόζουμε την αρχή της απόφασης.



Χρησιμοποιήσαμε άλλη μια πρόταση, την ¬sam=tweety, δηλαδή ισχύει sam≠tweety, που είναι άμεση συνέπεια της υπόθεσης UNA (σελ. VI.9.)

Παράδειγμα 6.10

Δ = {καρέκλα(x) → κάθισμα(x), σκαμπό(x) → κάθισμα(x),
καρέκλα(a), τραπέζι(b), σκαμπό(c), γάτα(d),
¬καρέκλα(d), ¬τραπέζι(d), ¬σκαμπό(d)}

Να αποδειχθεί ότι το d δεν είναι κάθισμα.

Φ ⊨| ¬κάθισμα(d)

Θα προσπαθήσουμε να αποδείξουμε ότι

$\Delta \cup \{\neg\Phi\} \vdash_{\text{Resolution}}$

$\Delta \cup \{\neg\Phi\} = \{$κάθισμα($x_1$) ∨ ¬καρέκλα($x_1$), (1)
κάθισμα(x_2) ∨ ¬σκαμπό(x_2), (2)
καρέκλα(a), (3)
τραπέζι(b), (4)
σκαμπό(c), (5)
γάτα(d), (6)
¬καρέκλα(d) (7)
¬τραπέζι(d) (8)
¬σκαμπό(d) (9)
κάθισμα(d)} (10)

Παρατήρηση : Δεν γίνεται να παράγουμε την κενή πρόταση γιατί δεν εμφανίζεται σε καμιά πρόταση το άτομο ¬κάθισμα(d).

1. Λύση με χρήση του CWA.

C = {a, b, c, d}, Φ = ¬κάθισμα(d)

Δasm = {¬σκαμπό(a), ¬τραπέζι(a), ¬γάτα(a),
¬καρέκλα(b), ¬σκαμπό(b), ¬γάτα(b), ¬κάθισμα(b),
¬καρέκλα(c), ¬τραπέζι(c),
¬κάθισμα(d)}

Δ ∪ Δasm = ...

Αρα τώρα εύκολα $\boxed{\text{CWA}[\Delta] \cup \{\neg\Phi\} \vdash_{\text{Resolution}}}$

Η δυσκολία σ' αυτή την περίπτωση ήταν να βρούμε το Δasm, γιατί έπρεπε να κάνουμε έλεγχο για όλα τα αντικείμενα του C και όλες τις σχέσεις του Δ.

2. Λύση με χρήση του COMP.

... πρόταση συμπλήρωσης ως προς κάθισμα:

κάθισμα(x) → καρέκλα(x) ∨ σκαμπό(x)

Αρα COMP[Δ;κάθισμα] ∪ {¬Φ} =

1: {καρέκλα(x_1) ∨ σκαμπό(x_1) ∨ ¬κάθισμα(x_1),
2: κάθισμα(x_2) ∨ ¬καρέκλα(x_2),
3: κάθισμα(x_3) ∨ ¬σκαμπό(x_3),
4: καρέκλα(a),
5: τραπέζι(b),
6: σκαμπό(c),
7: γάτα(d),
8: ¬καρέκλα(d),

9: ¬τραπέζι(d),
10: ¬σκαμπό(d),
11: κάθισμα(d)}
12: από 1, 11 με σ = {d/x_1} καρέκλα(d) ∨ σκαμπό(d)
13: από 10, 12 καρέκλα(d)
14: από 8, 13

COMP[Δ;κάθισμα] ∪ {¬Φ} ├$_{Resolution}$

- **Ενδιαφέρουσες μορφές του COMP**
- Εάν Δ = {P(X)} τότε το P(X) - που αληθεύει για οποιαδήποτε τιμή του X - μπορεί να γραφεί σαν ∀X true ⇒ P(X) και η πρόταση συμπλήρωσης γίνεται P(X) ⇒ true.

- Εάν αντίθετα το Δ δεν περιέχει το P μπορούμε να γράψουμε ∀X false ⇒ P(X) και η πρόταση συμπλήρωσης γίνεται ∀X P(X) ⇒ false ή ¬P(X) δηλαδή «κανένα αντικείμενο δεν ικανοποιεί το P».

- Στις απλές περιπτώσεις τα CWA και COMP βγάζουν τα ίδια αποτελέσματα. Γενικά όμως δεν ισχύει. Εκείνο που γενικά ισχύει είναι ότι

CWA + DCA συνεπάγεται COMP
και COMP + UNA συνεπάγεται CWA.

- Η συμπλήρωση COMP είναι μη μονότονη γιατί αν προσθέσουμε στο Δ κάποια άλλη πληροφορία αλλάζει η πρόταση συμπλήρωσης. Ετσι:

Δ = { στρουθοκάμηλος(X) ⇒ πουλί(X),
πιγκουΐνος(X) ⇒ πουλί(X),
πουλί(tweety),
¬στρουθοκάμηλος(sam)}

Πρόταση συμπλήρωσης:

πουλί(X) ⇒ στρουθοκάμηλος(X) ∨ πιγκουΐνος(X) ∨ X=tweety.
Ετσι δεν μπορούμε πλέον να αποδείξουμε ότι ¬πουλί(sam), γιατί ο sam δεν είναι μεν στρουθοκάμηλος, αλλά θα μπορούσε να είναι πιγκουΐνος!!!

6.6 Συμπεράσματα

Εξετάσαμε δύο πολύ απλές περιπτώσεις μη μονότονης λογικής, την υπόθεση του κλειστού κόσμου, και την συμπλήρωση κατηγορήματος.

Τις περιπτώσεις αυτές τις είδαμε στην απλή τους μορφή, και ίσως δεν φαίνεται η σημασία τους. Η αλήθεια είναι ότι η μη μονότονη συμπερασματολογία βρίσκεται ακόμη σε «πρωτόγονη φάση». Πρακτικά έχουν αναπτυχθεί πολύ λίγα συστήματα που να την

υλοποιούν. Θεωρητικά υπάρχουν πολλά ανοικτά θέματα. Ερευνητικά, δεν έχει μελετηθεί η εφαρμογή της σε πραγματικά προβλήματα - δεν έχει ακόμα ωριμάσει αρκετά.
Υπάρχουν και άλλα συστήματα μη μονότονης συμπερασματολογίας, όπως η περιχάραξη (circumscription), οι θεωρίες τυπικής τιμής (default theories), κ.λ.π.

Ομως έχουν αναπτυχθεί και λογικές οι οποίες ξεφεύγουν από τους περιορισμούς της κλασικής κατηγορηματικής λογικής α' τάξης. Ερωτήματα που έχουν απαντηθεί είναι :

- Γιατί δίτιμη λογική;

γιατί όχι τρίτιμη, τετράτιμη, ..., πολύ-τιμη λογική,

- Γιατί να μη χρησιμοποιηθούν κάποια μέτρα αβεβαιότητας;
  - Θεωρία της μαρτυρίας (Evidence Theory)
  - Συντελεστές βεβαιότητας (Certainty factors)
  - Μέτρα Αξιοπιστίας και Ευλογοφάνειας (Credibility - Plausibility measures)
  - Μέτρα Αναγκαιότητας - Δυνατότητας (Necessity - Possibility measures)
  - Μέτρα Πιθανότητας (Possibility measures)
  - Ασαφής Λογική
- Πως θα μοντελοποιήσουμε ποιοτικά την αναγκαιότητα και τη δυνατότητα ;

Λογική του τρόπου (Modal logic)

- Πως θα μοντελοποιήσουμε τον χρόνο ;

Χρονικές λογικές (Temporal logics)

- Αλλα θέματα ;

Γνώση και πεποίθηση (Knowledge and Belief)
Μεταγνώση και ενδοσκόπηση κ.λπ.

7

Περί Prolog και Λογικού Προγραμματισμού

7.1 Εισαγωγή

Στο κεφάλαιο αυτό θα ασχοληθούμε με τη γλώσσα Λογικού Προγραμματισμού Prolog. Η λέξη Prolog βγαίνει από τα αρχικά PROgramming in LOGic, δηλαδή προγραμματίζοντας στη Λογική. Είναι μιά γλώσσα τέταρτης γενιάς, που έχει αναπτυχθεί τα τελευταία κυρίως χρόνια, αν και αμόμη δεν έχει βρει 'το δρόμο της', δεν έχει δηλαδή καθιερωθεί μεταξύ των άλλων συμβατικών γλωσσών προγραμματισμού.

Ο λόγος για αυτό δεν είναι τόσο οι ελείψεις της (αν και εμφανίζει τέτοιες), όσο η άγνοια όσον αφορά αυτή τη γλώσσα των υπευθύνων των εταιρειών παραγωγής λογισμικού (software houses) και των προγραμματιστών.

Ο προγραμματισμός με Prolog απαιτεί κάποιες στοιχειώδεις γνώσεις Λογικής. Οχι τόσο γιά την εκμάθηση των συγκεκριμένων κατηγορημάτων της Prolog, όσο γιά την βαθύτερη κατανόηση του τρόπου λειτουργίας της.

Δεν πρέπει βέβαια να αποκρύψουμε και το γεγονός ότι οι περισσότεροι διερμηνείς και μεταφραστές της Prolog δεν περιέχουν δυνατότητες επικοινωνίας με βάσεις δεδομένων, εντολές χειρισμού γραφικών, κ.λ.π. Ετσι, λόγω των χαμηλών δυνατοτήτων διαπροσωπείας (interface) που προσφέρει αυτή η γλώσσα έχει παραμείνει σαν εργαλείο κυρίως σε ερευνητικούς και ακαδημαϊκούς χώρους. Βέβαια οι περισσότεροι μεταφραστές της Prolog παρέχουν την δυνατότητα σύνδεσης με άλλες γλώσσες προγραμματισμού όπως η Pascal και η C, και έτσι, αν θέλει κανείς, μπορεί να ξεπεράσει, με κάποιο κόπο, αυτές τις δυσκολίες.

Η Prolog είναι γλώσσα Λογικού Προγραμματισμού και δίνει πολύ μεγάλες δυνατότητες σε ένα προγραμματιστη να λύσει προβλήματα γρήγορα και με κομψό τρόπο. Ομως ο προγραμματισμός σ'αυτή τη γλώσσα έχει τελείως διαφορετικό νόημα από τον συμβατικό προγραμματισμό που ακολουθείται σε διαδικαστικές γλώσσες όπως η Pascal και η C. Αν λοιπόν θέλετε να προγραμματίσετε σε Prolog ξεχάστε τον τρόπο που προγραμματίζατε.

Στην Prolog τα προγράμματα είναι ορισμοί! Είναι ορισμοί κατηγορημάτων σε σχέση με το εννοιολογικό περιεχόμενο των κατηγορημάτων. Μάλιστα είναι στις περισσότερες περιπτώσεις αναδρομικοί ορισμοί, όπως αυτοί που δίνουμε σε μερικές περιοχές των μαθηματικών (όπως η Λογική).

Η Prolog όμως ήταν η πρώτη γλώσσα που άνοιξε τις πόρτες της Τεχνητής Νοημοσύνης στο ευρύτερο κοινό και έτσι κατέχει σήμερα την πρώτη θέση ανάμεσα στις γλώσσες λογικού προγραμματισμού.

Ενας από τους λόγους της γενικότερης επιτυχίας της, σαν γλώσσα λογικού προγραμματισμού, είναι ότι η Prolog προσφέρει εκτός του λογικού της μέρους, και πλήθος από μεταλογικά (metalogical) και πέραν του λογικού προγραμματισμού (extralogical) χαρακτηριστικά, που την καθιστούν αρκετά ικανή στο θέμα της επικοινωνίας με τον υπολογιστή.

Από την πλευρά του λογικού προγραμματισμού, η Prolog βασίζεται στην ταξινομημένη αρχή της απόφασης στην είσοδο και ο προγραμματισμός της γίνεται με φράσεις τύπου Horn (Horn clauses). Το μέρος εκείνο της Prolog που αναλαμβάνει να υλοποιήσει την αρχή της απόφασης, την επιλογή των υποψήφιων γονικών προτάσεων, και την επιλογή των υποψήφιων για ταίριασμα ατόμων ονομάζεται μηχανή συμπερασματολογίας (inference engine).

Το σύνολο των λογικών προτάσεων που κάθε φορά εκτελεί η Prolog ονομάζεται λογικό πρόγραμμα (logic program) ή και βάση γνώσης (knowledge base). Βέβαια η έννοια βάση γνώσης αναφέρεται συνήθως σε κάτι γενικότερο, που είναι η δομή εκείνη στην οποία έχουμε αποθηκεύσει την γνώση. Στην περίπτωση του λογικού προγραμματισμού βάση γνώσης είναι το λογικό πρόγραμμα.

7.2 Δομή ενός Λογικού Προγράμματος

7.2.1. Λογικά προγράμματα (Logic programs)

Ορισμός 7.1
Ενα Λογικό πρόγραμμα μπορεί να ορισθεί σαν μιά τριάδα (T,N,P), όπου T είναι ένας πεπερασμένος αριθμός όρων, N είναι ένας πεπερασμένος αριθμός από κατηγορηματικά σύμβολα που έχουν σαν παραμέτρους όρους που ανήκουν στο T, P είναι ένας πεπερασμένος αριθμός από φράσεις τύπου Horn, που περιέχουν κατηγορήματα που ανήκουν στο σύνολο N.

Ενα λογικό πρόγραμμα L είναι ένα σύνολο λογικών προτάσεων που κατηγοριοποιούνται σε γεγονότα (facts), κανόνες (rules) και ερωτήσεις (queries).

Τα **γεγονότα** είναι προτάσεις που περιέχουν μόνο ένα θετικό λεκτικό, και εκφράζουν ατομικές αλήθειες (δηλαδή απλές προτάσεις - άτομα - που η αλήθεια τους δεν εξαρτάται από άλλες προτάσεις), π.χ.

```
human(socrates).
cartoon(ran_tan_plan).
```

Οι **κανόνες** είναι προτάσεις που περιέχουν ένα θετικό λεκτικό και τουλάχιστον ένα αρνητικό λεκτικό. Ομως οι προτάσεις αυτές στην Prolog γράφονται με την ισοδύναμη μορφή :

R_o:-R_1,R_2,...,R_n.

Το Ro αναφέρεται σαν *συμπέρασμα* του κανόνα ή *κεφαλή* του κανόνα. Τα Ri, i=1,...,n αναφέρονται σαν *συνθήκες* του κανόνα, ή σαν *παιδιά* του κανόνα, ή σαν κατηγορήματα του *σώματος* του κανόνα.

Ο κανόνας (1) διαβάζεται ως εξής : Γιά να αποδειχθεί το Ro αρκεί να αποδειχθούν τα R1,R2,... Αυτή η ερμηνεία του κανόνα είναι **δηλωτική** (declarative) γιατί θυμίζει τους ορισμούς.. Ενας ορισμός είναι δηλωτικός στην φύση του γιατί ορίζει μιά έννοια με βάση κάποιες άλλες πιό στοιχειώδεις έννοιες.

Κάθε φορά που προσπαθούμε να αποδείξουμε ένα κατηγόρημα Q το θεωρούμε σαν στόχο. Η Prolog επιλέγει ένα κανόνα που έχει σαν συμπέρασμα (κεφαλή) ένα άλλο κατηγόρημα Q' το οποίο μπορεί να ταιριάξει (ενοποίηση) με το Q. Τα παιδιά του Q' θα γίνουν στην συνέχεια το καθένα με την σειρά του καινούριοι στόχοι.

Οι κανόνες κωδικοποιούν ορισμούς, ή εκφράζουν την άποψή μας γιά τις συνθήκες που πρέπει να ικανοποιούνται γιά να βγάλουμε κάποιο συμπέρασμα. Ενα παράδειγμα κανόνα είναι το ακόλουθο :

```
grandparent(X,Y):-parents(X,V,U),parents(V,Y,W).
```

Οι **ερωτήσεις** που κάνουμε στην Prolog είναι κατά κάποιο τρόπο προτάσεις που τις αναθέτουμε να αποδείξει, κάτι σαν 'θεωρήματα', δεδομένων των 'αξιωμάτων' που περιέχονται στο λογικό πρόγραμμα. Η μέθοδος απόδειξης που χρησιμοποιεί η Prolog είναι **αντίστοιχη** της απαγωγής σε άτοπο (refutation procedure, όπου refutation είναι η παραγωγή της κενής πρότασης). Σύμφωνα όμως με αυτή την μέθοδο απόδειξης, εισάγουμε την άρνηση της προς απόδειξη πρότασης στο αρχικό σύνολο των προτάσεών, και αρχίζουμε να αποδεικνύουμε προτάσεις έως ότου αποδείξουμε την κενή πρόταση (που είναι ψευδής) οπότε φτάνουμε σε άτοπο.

Με βάση αυτά αν θέλουμε να αποδείξουμε μιά πρόταση $\neg Q_1 \vee \neg Q_2 \vee ... \vee \neg Q_n$, πρέπει να την αντιστρέψουμε $(Q_1 \wedge Q_2 \wedge ... \wedge Q_n)$ και να την εισάγουμε στο αρχικό σύνολο προτάσεων.

7.2.2 Ενοποίηση (unification)

Στις συμβατικές γλώσσες διαθέτουμε την έννοια της ανάθεσης τιμής, με την οποία δίνουμε τιμές στις μεταβλητές. Στην Prolog όμως δεν διαθέτουμε τέτοιες μεταβλητές ούτε τέτοια έννοια ανάθεσης. Γιά παράδειγμα στην Prolog δεν μπορούμε να γράψουμε x=x+1, δηλαδή αυτό που στις συμβατικές γλώσσες σημαίνει ότι η μεταβλητή x έχει σαν τιμή την τιμή που είχε προηγουμένως συν ένα, δεν έχουμε δηλαδή ανάθεση με την κλασσική της έννοια.

Αντίθετα στην Prolog έχουμε την έννοια της ενοποίησης. Η ενοποίηση, είναι μιά πιό γενική λειτουργία από την απλή ανάθεση γιατί αν η ανάθεση "δένει" τις μεταβλητές με κάποιες τιμές, η ενοποίηση μπορεί να συνδέσει μεταβλητές με σταθερές, άλλες μεταβλητές, ή ακόμη και συναρτήσεις που περιέχουν σταθερές ή μεταβλητές.

7.3 Συσχέτιση με την Κατηγορηματική Λογική

Ας θυμηθούμε όμως πρώτα ορισμένα στοιχεία : Η αρχή της απόφασης στην είσοδο είναι ένας περιορισμός της αρχής της απόφασης, που απαιτεί μιά από τις γονικές προτάσεις να είναι μιά πρόταση εισόδου, δηλαδή να μην βρίσκεται μεταξύ των προτάσεων που παράγονται κατά την διάρκεια της αποδεικτικής διαδικασίας.

Η αρχή της απόφασης στην είσοδο δεν είναι πλήρης (complete). Για παράδειγμα οι προτάσεις $P \vee Q$, $P \vee \neg Q$, $\neg P \vee Q$, $\neg P \vee \neg Q$, δεν οδηγούνται σε αντίφαση με την στρατηγική αυτή, ενώ οδηγούνται με την γενική αρχή της απόφασης. Ομως η αρχή της απόφασης στην είσοδο είναι πλήρης γιά ένα υποσύνολο προτάσεων που ονομάζονται φράσεις τύπου Horn. Μια γενική πρόταση έχει την παρακάτω μορφή.

$$R_0 \vee R_1 \vee ... \vee R_j \vee \neg R_{j+1} \vee ... \vee \neg R_n \quad (7.1)$$

Η πρόταση αυτή περιέχει j+1 θετικά άτομα και n-j αρνητικά άτομα.

Φράσεις τύπου Horn (Horn clauses) είναι εκείνες οι οποίες περιέχουν το πολύ ένα θετικό άτομο. Δηλαδή οι φράσεις Horn είναι των παρακάτω τριών μορφών.

$R0 \vee \neg R1 \vee ... \vee \neg Rn$
$R0$
$\neg R1 \vee ... \vee \neg Rn$

Από αυτές ο πρώτος τύπος αντιστοιχεί στους κανόνες, ο δεύτερος στα γεγονότα και ο τρίτος στις ερωτήσεις της Prolog.

Η ταξινομημένη αρχή της απόφασης στην είσοδο είναι μιά περαιτέρω περιορισμένη στρατηγική κατά την οποία τα επιλεγόμενα αντιτιθέμενα άτομα των γονικών προτάσεων, δεν επιλέγονται τυχαία, αλλά με μιά προκαθορισμένη σειρά, γιά παράδειγμα από αριστερά προς τα δεξιά. Οπως είπαμε παραπάνω η Prolog βασίζεται στην ταξινομημένη αρχή της απόφασης στην είσοδο. Θα δείξουμε στίς παρακάτω παραγράφους πώς γίνεται αυτό.

Στην Prolog ξεκινάμε από την πρόταση της ερώτησης την οποία και θεωρούμε σαν την πρώτη παραγόμενη πρόταση. Επιλέγουμε το πρώτο κάθε φορά άτομο της παραγόμενης πρότασης, και το ονομάζουμε παρόντα στόχο (current goal). Στην συνέχεια, προσπαθούμε να βρούμε ένα πιθανό ταίριασμα του παρόντος στόχου με το συμπέρασμα κάποιου κανόνα (την κεφαλή κάποιου κανόνα). Εάν κάποιο τέτοιο ταίριασμα βρεθεί, ενεργοποιούμε τον αλγόριθμο ενοποίησης, και εάν αυτός επιτύχει, αντικαθιστούμε το πρώτο άτομο της παραγόμενης πρότασης με το σύνολο των ατόμων του σώματος του κανόνα.

Για παράδειγμα, ας υποθέσουμε ότι έχουμε κάποια παραγόμενη πρόταση

$$\neg Q_1 \vee ... \vee \neg Q_n$$

και υπάρχει στο λογικό πρόγραμμα κάποιος κανόνας της μορφής

$$Q_1 \vee \neg P_1 \vee ... \vee \neg P_m$$

Τα δύο αντιτιθέμενα άτομα $\neg Q_1$ και Q_1 θα εξαλειφθούν, και η νέα παραγόμενη πρόταση θα γίνει

$$\neg P_1 \vee ... \vee \neg P_m \vee \neg Q_2 \vee ... \vee \neg Q_n$$

Παρατηρούμε δηλαδή ότι, όταν επιλέγονται κανόνες, η παραγόμενη προτάση τείνει να εκτείνεται συνεχώς. Ο μόνος τρόπος να αρχίσει να περιορίζεται είναι, αν αντί γιά κανόνες επιλεχτούν γεγονότα. Τα γεγονότα δεν διαθέτουν παρά μόνο ένα άτομο, με αποτέλεσμα η παραγόμενη πρόταση να συρρικνωθεί.

Οταν η παραγόμενη πρόταση μείνει κενή, σημαίνει πως ,σύμφωνα με την αρχή της απόφασης, έχουμε φτάσει σε αντίφαση, οπότε η αρχική πρόταση (η ερώτηση) έχει αποδειχθεί. Στην περίπτωση της Prolog, που διαθέτουμε και παραμέτρους, καθώς αποδεικνύεται η αρχική πρόταση παίρνουν συγχρόνως τιμές και οι μεταβλητές της ερώτησης.

Ενα σημαντικό χαρακτηριστικό της μηχανής συμπερασματολογίας της Prolog, είναι η δυνατότητα που έχει σε περίπτωση αποτυχίας να επαναφέρει την παραγόμενη πρόταση στην προηγούμενη μορφή της, και να αναζητήσει μιά άλλη εναλλακτική λύση (alternative solution).

Αυτό επιτυγχάνεται μέσω ενός μηχανισμού οπισθοδρόμησης (backtracking mechanism), που αναλαμβάνει αφ'ενός την διαδικασία επαναφοράς και αφ'ετέρου την εκ νέου αναζήτηση υποψήφιου κανόνα.

Οπως φαίνεται από τα παραπάνω, η μηχανή συμπερασματολογίας της Prolog χρησιμοποιεί μεθοδολογία κατευθυνόμενη από τους στόχους.

Ενα άλλο χαρακτηριστικό της μηχανής συμπερασματολογίας της Prolog που δεν έχουμε μέχρι τώρα θίξει, είναι η στρατηγική επιλογής υποψήφιων στόχων. Η στρατηγική αυτή είναι μιά από αριστερά προς τα δεξιά επιλογή των νέων στόχων (left to right selection of the goals), συνδυασμένη με μιά κατά βάθος στρατηγική αναζήτησης. Η αναζήτηση των υποψήφιων γιά ταίριασμα κανόνων μέσα από το λογικό πρόγραμμα γίνεται απλά από την αρχή του λογικού προγράμματος πρός το τέλος.

Ας δούμε ένα παράδειγμα λογικού προγράμματος, και ας παρακολουθήσουμε βήμα-βήμα την εκτέλεση του προγράμματος από την Prolog.

Παράδειγμα 7.1 Ας υποθέσουμε γιά παράδειγμα ότι έχουμε το παρακάτω πρόγραμμα Prolog. Ρωτάμε την Prolog ?A(X,Y).

```
a(X,Y) :-    b(X,Y).          /*  1 */
b(X,Y) :-    d(X),e(Y).       /*  2 */
d(c).                                 /*  3 */
d(d).                                 /*  4 */
d(X)   :-    c(X,Z),g(Z).             /*  5 */
c(c,a).                               /*  6 */
g(a).                                 /*  7 */
c(d,b).                          /*  8 */
g(b).                                 /*  9 */
e(f).                                 /* 10 */
e(g).                                 /* 11 */
a(a,b).                          /* 12 */
b(g,h).                          /* 13 */
```

Η Prolog θα ταιριάξει το a(X,Y) με την κεφαλή του κανόνα 1, και το b(X,Y) του κανόνα 1 με την κεφαλή του κανόνα 2. Στην συνέχεια επιλέγει το d(X) του κανόνα 2 και βρίσκει την λύση d(c) από το γεγονός 3. Η Prolog στην συνέχεια θα βρεί την λύση e(f) από το γεγονός 10, και τελικά θα απαντήσει a(c,f).

Αν ζητήσουμε από την Prolog και άλλες λύσεις θα οπισθοδρομήσει στον τελευταίο στόχο και θα ψάξει γιά κάποια άλλη εναλλακτική λύση γιά το e(X). Μιά τέτοια λύση θα βρεθεί από το γεγονός 11, η e(g), οπότε η νέα λύση γιά το a(X,Y) θα είναι η a(c,g).

Γιά την επόμενη λύση, η Prolog θα οπισθοδρομήσει στο d(X) του κανόνα 2, θα πετύχει στο d(d) του γεγονότος 4, και θα ξαναδοκιμάσει το e(Y) από την αρχή. Ετσι, με τον ίδιο τρόπο που βρήκε τις δύο πρώτες λύσεις, θα βρεί τώρα τις λύσεις a(d,f) και a(d,g).

Γιά το d(X) του κανόνα 2, υπάρχει και τρίτος εναλλακτικός δρόμος : ο κανόνας 5. Με τον κανόνα αυτό υπεισέρχονται στην αποδεικτική διαδικασία και άλλοι στόχοι, οι c(X,Z), g(Z).

Αναλύοντας την αποδεικτική διαδικασία με τον τρόπο που υποδείξαμε παραπάνω, μπορούμε να βρούμε όλες τις δυνατές λύσεις γιά τον αρχικό στόχο a(X,Y).

Την αποδεικτική διαδικασία μπορούμε εύκολα να την αποτυπώσουμε σε δομές δενδρικής μορφής, όπου στην ρίζα του δένδρου υπάρχει ο αρχικός στόχος, και στις διακλαδώσεις οι κανόνες. Στα φύλλα του δένδρου εμφανίζονται τα γεγονότα. Τα δένδρα αυτά λεγονται και δένδρα απόδειξης.

7.4 Υλοποίηση της Prolog και ερευνητικές κατευθύνσεις

Εχουν προταθεί πολλές μεθοδολογίες γιά την υλοποίηση ενός διερμηνέα (interpreter) Prolog. Οπως είναι γνωστό, τον βασικό αλγόριθμο γιά ένα μεταφραστή Prolog τον εμπνεύστηκαν οι A.Colmerauer και Ph.Roussel όταν και πρωτο-υλοποιηθήκε η Prolog στην Μασσαλία το 1972.

Δεν υπάρχουν δημοσιεύσεις του πλήρους βασικού αλγορίθμου, αν και υπάρχει πλήθος αναφορών που σχετίζονται με προσεγγίσεις στην επιμέρους υλοποίηση της Prolog. Οι προσεγγίσεις αυτές διακρίνονται μεταξύ τους στο τρόπο διαχείρισης των δομών στον αλγόριθμο ενοποίησης (structure sharing versus copying).

Εχουν επίσης δημοσιευτεί διάφοροι απλοί αλγόριθμοι υλοποίησης της Prolog, μέσα από διαφορετικές προσεγγίσεις. Το βασικό θέμα λοιπόν της υλοποίησης της Prolog έχει σε μεγάλο βαθμό ερευνηθεί, και για τον λόγο αυτό οι ερευνητικές δραστηριότητες σήμερα στρέφονται πρός άλλες κατευθύνσεις.

Μιά από τις κατευθύνσεις αυτές είναι η ενσωμάτωση της έννοιας της παραλληλίας στην Prolog και η υλοποίηση παράλληλης Prolog (Concurrent Prolog). Ο παραλληλισμός αυτός μπορεί να είναι είτε εγγενής στο σύστημα είτε να καθορίζεται από τον ίδιο τον προγραμματιστή.

Αλλες κατευθύνσεις είναι η ανάπτυξη μιάς γλώσσας Prolog με χαρακτηριστικά ενός πλήρη αποδείκτη θεωρημάτων, η ενσωμάτωση αναπαράστασης και επεξεργασίας αβέβαιης γνώσης στην Prolog, η βελτίωση του μηχανισμού οπισθοδρόμησης κλπ.

Στο τέλος του κεφαλαίου δίνουμε ένα απλό αλγόριθμο σε μορφή διαγράμματος ροής, ο οποίος συμπεριφέρεται όπως ακριβώς η Prolog. Ο αλγόριθμος είναι απλοποιημένος, από την άποψη ότι δεν επιτρέπουμε στα κατηγορήματα να έχουν παραμέτρους. Επίσης δεν λαμβάνουμε υπόψη μας ειδικού τύπου κατηγορήματα.

7.5 Επιπρόσθετα χαρακτηριστικά της Prolog

Εκτός των χαρακτηριστικών της που προέρχονται από τον κατηγορικό λογισμό πρώτης τάξης, η Prolog διαθέτει και ένα πλήθος επιπρόσθετα κατηγορήματα πολυ χρήσιμα, είτε κανείς την χρησιμοποιεί σαν μιά γλώσσα λογικού προγραμματισμού είτε σαν μιά οποιαδήποτε άλλη γλώσσα προγραμματισμού. Μπορούμε να ομαδοποιήσουμε αυτά τα κατηγορήματα σε διάφορες κατηγορίες όπως :

- Κατηγορήματα για αριθμητικές πράξεις (arithmetic predicates).
- Κατηγορήματα επισκόπησης δομών (structure inspection).
- Μεταλογικά Κατηγορήματα (metalogical predicates).
- Κατηγορήματα ελέγχου (control predicates).

- Κατηγορήματα συστήματος (Είσοδος/Εξοδος, συστήματα αρχείων κλπ.).
- Και άλλα επιπρόσθετα κατηγορήματα που οι εκάστοτε υλοποιήσεις μπορεί να διαθέτουν.

Θα αναφερθούμε μόνο σε ένα σημαντικό κατηγόρημα ελέγχου της Prolog, στο cut ("!"). Το cut όταν χρησιμοποιηθεί αλλάζει την διαδικαστική συμπεριφορά των λογικών προγραμμάτων. Η Prolog χειρίζεται το cut σύμφωνα με τον παρακάτω ορισμό.

Ορισμός 7.2
"*Σαν στόχος το cut πάντοτε επιτυγχάνει, αλλά δεσμεύει την Prolog σε όλες τις επιλογές που έχει κάνει από την στιγμή που ο αμέσως προηγούμενος στόχος είχε ενοποιηθεί με την κεφαλή του κανόνα στον οποίο βρίσκεται το cut*".

Παράδειγμα 7.2
Ας υποθέσουμε γιά παράδειγμα ότι έχουμε το παρακάτω πρόγραμμα Prolog. Ρωτάμε την Prolog ?A(X,Y).

```
1.  A(X,Y):-B(X,Y).
2.  B(X,Y):-D(X),!,E(Y).
3.  D(c).
4.  D(d).
5.  D(X):-C(X,Z),G(Z).
6.  C(c,a).
7.  G(a).
8.  C(d,b).
9.  G(b).
10. E(f).
11. E(g).
12. A(a,b).
13. B(g,h).
```

Η Prolog θα ταιριάξει το A(X,Y) με την κεφαλή του κανόνα 1, και το B(X,Y) του κανόνα 1 με την κεφαλή του κανόνα 2. Στην συνέχεια επιλέγει το D(X) του κανόνα 2 και βρίσκει την λύση D(c) από το γεγονός 3.

Εξετάζει το cut (!) το οποίο και επιτυγχάνει, αλλά συγχρόνως δεσμεύεται τόσο στην επιλογή της λύσης D(c), όσο και στην επιλογή του κανόνα 2. Η Prolog στην συνέχεια θα βρεί την λύση E(f) από το γεγονός 10, και τελικά θα απαντήσει A(c,f).

Αν ζητήσουμε όμως από την Prolog και άλλες λύσεις θα δώσει ακόμη δύο την A(c,g) και την A(a,b). Γιά να βρεί την λύση A(c,g) η Prolog ενεργεί ως εξής: Αναζητεί μιά ακόμη λύση για το E(Y), και βρίσκει το E(g) (γεγονός 11). Στην συνέχεια όμως όταν προσπαθήσει να κάνει οπισθοδρόμηση (backtracking) πάνω από το cut θα αγνοήσει άλλες λύσεις του D(X) και άλλες εναλλακτικές προτάσεις γιά το B(X,Y), και θα επιστρέψει στο A(X,Y). Γιά το A(X,Y) θα αναζητήσει εναλλακτικές προτάσεις, και θα βρεί το γεγονός 12, δηλαδή την λύση A(a,b).

Βλέπουμε λοιπόν ότι με την χρησιμοποίηση του cut (!), η Prolog έχει "παγώσει" τις επιλογές της τόσο γιά όλα τα κατηγορήματα μεταξύ του cut και της κεφαλής του κανόνα, όσο και εναλλακτικές προτάσεις του ίδιου του κανόνα στον οποίο βρίσκεται. Η λειτουργικότητα του cut φαίνεται κατά την στιγμή που ενεργείται ο μηχανισμός οπισθοδρόμησης πάνω στο cut.

Ας σημειώσουμε μόνο ότι το ίδιο πρόγραμμα χωρίς το cut θα έδινε τις ακόλουθες λύσεις: A(g,h), A(c,f), A(c,g), A(d,f), A(d,g), A(c,f), A(c,g), A(d,f), A(d,g), A(a,b). (παράδειγμα 7.1)

Ενα άλλο σημείο που είναι άξιο προσοχής είναι η άρνηση όπως υλοποιείται από την Prolog. Η άρνηση της Prolog ονομάζεται άρνηση μέσω αποτυχίας (negation by failure). Η άρνηση μέσω αποτυχίας υποστηρίζει την συμπερασματολογία σύμφωνα με την υπόθεση του κλειστού κόσμου (closed world assumption).

Ορισμός 7.3. Η υπόθεση του κλειστού κόσμου στη Prolog
Η υπόθεση του κλειστού κόσμου υποστηρίζει ότι "*γιά ένα κατηγόρημα τα στιγμιότυπα γιά τα οποία ισχύει το κατηγόρημα είναι μόνο εκείνα που μπορούν να αποδειχθούν με την αποδεικτική διαδικασία της Prolog*".

Ορισμός 7.4. Η άρνηση μέσω αποτυχίας
Σύμφωνα με την υπόθεση του κλειστού κόσμου, η αποτυχία να αποδειχθεί ένα κατηγόρημα (με συγκεκριμένες παραμέτρους) συνεπάγεται ότι η αρνησή του ισχύει. Η υπόθεση του κλειστού κόσμου έχει ερευνηθεί θεωρητικά σε μεγάλη έκταση.

Η άρνηση της Prolog εκφράζεται με το κατηγόρημα not και μπορεί να ορισθεί χρησιμοποιώντας το cut και με κατηγορικό λογισμό δεύτερης τάξης, με τον τρόπο που εξηγούμε παρακάτω. Ας σημειωθεί ότι στο κατηγορικό λογισμό δεύτερης τάξης, μπορούμε να χειριστούμε κατηγορήματα σαν μεταβλητές.

Ορισμός 7.5. Η άρνηση μέσω αποτυχίας

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

Στον ορισμό αυτό το P είναι κάποιο κατηγόρημα που πιθανόν και αυτό να έχει κάποιες παραμέτρους. Απλά χειριζόμαστε το κατηγόρημα P σαν μιά συνάρτηση (function, δομή, structure). Το κατηγόρημα call(P) μετατρέπει την παράμετρό του P σε στόχο, δηλαδή ισοδυναμεί με το κατηγόρημα P. Το κατηγόρημα fail πάντοτε αποτυγχάνει.

Ας δούμε ένα παράδειγμα με την χρησιμοποίηση του not. Μάλιστα γιά να έχουμε ένα συγκριτικό παράδειγμα της συμπερασματολογίας της Prolog με την συμπερασματολογία ενός συστήματος που χρησιμοποιεί την γενική αρχή της απόφασης, θα χρησιμοποιήσουμε το παράδειγμα 5.25 με την αντικατάσταση της άρνησης "¬" με την άρνηση "not" της Prolog.

Παράδειγμα 7.3

```
 1. watch(X,film) :- has(X,color_TV), not(busy(X)).
 2. record(X,film):- has(X,video), busy(X).
 3. watch(X,film) :- record(X,film).
 4. has(paul,color_TV).
 5. has(mary,color_TV).
 6. has(peter,color_TV).
 7. has(sofia,video).
 8. has(mary,video).
 9. has(john,video).
10. has(peter,video).
```

Η ερώτηση που δίνουμε στο σύστημα είναι ?watch(X,Y).

Το not(busy(X)) επιτυγχάνει γιά τις τιμές εκείνες του X που αποτυγχάνει το busy(X). Με την ερώτηση watch(X,Y) η Prolog θα επιλέξει τον κανόνα 1, και θα προσπαθήσει να αποδείξει το has(X,color_TV). Θα επιτύχει και με τα τρία γεγονότα 4,5,6 και καθώς δεν υπάρχουν στη βάση γνώσης κανόνες γιά το busy(X), θα οδηγηθεί σε αποτυχίες γιά τα busy(paul), busy(mary), busy(peter), οπότε θα επιτύχει το not(busy(X)) και στις τρείς περιπτώσεις και θα δώσει τις απαντήσεις

1. watch(paul,film),
2. watch(mary,film),
3. watch(peter,film).

Στην συνέχεια θα επιλέξει τον κανόνα 3, αλλά θα γυρίσει χωρίς απάντηση, καθώς θα συναντήσει το busy(X) γιά το οποίο δεν διαθέτει γνώση.
Οι απαντήσεις αυτές περιέχουν (λανθασμένα) την επιπρόσθετη απάντηση watch(paul,film). Αυτό οφείλεται αφ'ενός στην διαφορετική έννοια που έχει η άρνηση στην Prolog και αφ'ετέρου στην περιορισμένη δυνατότητα συμπερασματολογίας της γλώσσας αυτής.

7.6 Περιορισμοί της Prolog (ως προς την ΚΛ)

Η Prolog δεν μπορεί να θεωρηθεί ένας πλήρης αποδείκτης θεωρημάτων, και αυτό οφείλεται σε τρείς κύριους περιορισμούς της.

Α) Αλγόριθμος ενοποίησης χωρίς τον έλεγχο εμφάνισης (occurs check).
Β) Απεριόριστη κατά βάθος στρατηγική αναζήτησης.
C) Ατελής μηχανισμός συμπερασματολογίας (incomplete inference system).

Στις επόμενες παραγράφους θα συζητήσουμε τους τρείς αυτούς περιορισμούς.

Α) Αλγόριθμος ενοποίησης χωρίς τον έλεγχο εμφάνισης

Ορισμός 7.6
Ο έλεγχος εμφάνισης (occurs check) είναι ο έλεγχος που γίνεται στον αλγόριθμο ενοποίησης όταν πρόκειται να γίνει ενοποίηση μιάς μεταβλητής και ενός όρου και συνίσταται στην εξακρίβωση της ύπαρξης της μεταβλητής μέσα στον όρο.

Ο αλγόριθμος ενοποίησης με τον έλεγχο εμφάνισης δεν επιτρέπει την ενοποίηση μεταξύ μιάς μεταβλητής και ενός σύνθετου όρου που περιέχει την μεταβλητή.

Η αιτιολογία είναι πως αν κάτι τέτοιο ήταν επιτρεπτό, θα υπήρχαν περιπτώσεις που θα καταλήγαμε σε άπειρους όρους. Γιά παράδειγμα αν προσπαθήσουμε να ενοποιήσουμε τους όρους Y, s(Y), τότε παρατηρούμε ότι το Y παίρνει την τιμή s(Y), τό οποίο όμως είναι το s(s(Y)), που είναι το s(s(s(Y))) κλπ. Τελικά το Y είναι ένας άπειρος όρος της μορφής s(s(s(s(...)))).

Ο έλεγχος εμφάνισης είναι ένας χρονοβόρος έλεγχος δεδομένου ότι πρέπει να γίνεται πολύ συχνά κατά την διαδικασία απόδειξης. Ετσι γιά λόγους ταχύτητας, πολλές υλοποιήσεις

της Prolog αποφεύγουν την εκτέλεση αυτού του ελέγχου. Για εφαρμογές όμως ενός πλήρους αποδείκτη θεωρημάτων είναι απαραίτητος.

Β) Απεριόριστη κατά βάθος στρατηγική αναζήτησης

Η απεριόριστη κατά βάθος στρατηγική αναζήτησης της Prolog, μπορεί να δημιουργήσει σημαντικά προβλήματα στην διαδικασία απόδειξης. Το κυριότερο από αυτά τα προβλήματα είναι οι αριστερές αναδρομές. Σέ ένα πρόγραμμα που περιέχει αριστερές αναδρομές υπάρχει ο κίνδυνος το πρόγραμμα να εκτελεί επ'άπειρο ένα κύκλο κανόνων χωρίς να μπορεί μέσω κάποιας επιτυχίας ή αποτυχίας, να βγεί από την αναδρομή.

Παράδειγμα 7.4 Ας υπόθέσουμε ένα πρόγραμμα Prolog της μορφής :

1. successor(X,Y) :- successor(X,Z), parent(Y,Z).
2. successor(X,Y) :- parent(Y,X).
3. parent(j,b).
4. parent(j,l).
5. parent(b,a).
6. parent(b,p).
7. parent(c,j).
8. parent(d,c).

Με ένα τέτοιο λογικό πρόγραμμα η Prolog θα πέσει σε ένα άπειρο κύκλο, καθώς ο κανόνας 1 θα καλεί επ'άπειρο τον εαυτό του χωρίς ποτέ να μπορεί να αποδείξει αν το successor(X,Y) επιτυγχάνει η αποτυγχάνει. Για την αποφυγή του προβλήματος ο κανόνας 1 πρέπει να γραφτεί με την εξής μορφή :

1. successor(X,Y) :- parent(Y,Z) ,successor(X,Z).

Η εναλλαγή αυτή των κατηγορικών συμβόλων λύνει το παρόν μόνο πρόβλημα. Υπάρχουν όμως περιπτώσεις που κάτι τέτοιο δεν είναι δυνατό. Γιά να λυθεί αυτό το πρόβλημα έχει προταθεί η αντικατάσταση της απεριόριστης κατά βάθος στρατηγικής αναζήτησης, με μιά περιορισμένη. Θέτουμε δηλαδή κάποιο μέγιστο βάθος, και άν φτάνοντας σ'αυτό το βάθος δεν έχει αποδειχθεί ο αρχικός στόχος (top goal), αναζητούμε μέσω του μηχανισμού οπισθοδρόμησης άλλους εναλλακτικούς κανόνες γιά την εύρεση της λύσης.

Μιά τέτοια προσέγγιση λύνει το πρόβλημα, μπορεί όμως να καταλήξει σε μεγάλες άκαρπες προσπάθειες, καθώς μπορεί να έχουμε ήδη δοκιμάσει πολλά μονοπάτια πριν βρούμε αυτό που οδηγεί στην απάντηση. Αντ'αυτού έχει προταθεί μιά τακτική βαθμιαίας κατά βάθος αναζήτησης (staged depth-first) με σταδιακά αυξανόμενα επιτρεπτά βάθη.

C) Ατελής μηχανισμός συμπερασματολογίας

Οπως συζητήσαμε σε προηγούμενες παραγράφους, ο μηχανισμός συμπερασματολογίας της Prolog είναι πλήρης γιά προτάσεις Horn, αλλά ατελής γιά γενικές προτάσεις (Παράδειγμα 7.3). Οι γενικές προτάσεις δεν είναι πάντοτε απαραίτητες αλλά σε μερικές περιπτώσεις είναι αναπόφευκτες. Γι'αυτές τις τελευταίες περιπτώσεις η Prolog, είναι ανεπαρκής. Λύσεις σε αυτά τα προβλήματα έχουν δοθεί [Sti84, Pan88b], αλλά η εξέτασή τους υπερβαίνει τα πλαίσια αυτών των σημειώσεων.

7.7 Εφαρμογές της Prolog

Η γλώσσα Prolog μπορεί να χρησιμοποιηθεί σε όλες τις περιοχές που είναι εφαρμόσιμος και ο κατηγορικός λογισμός πρώτης τάξης. Επίσης, λόγω των πέρα της λογικής χαρακτηριστικών της μπορεί να βρεί εφαρμογή σε όλο το εύρος της επιστήμης της Πληροφορικής (σαν μιά γλώσσα προγραμματισμού).

Δεδομένου δε ότι, όπως έχει υποστηρίξει ο Kowalski "**Αλγόριθμος = Λογική + Ελεγχος**", η Prolog περισσότερο από κάθε άλλη γλώσσα προγραμματισμού προσφέρει ένα εύκολο τρόπο στον χρήστη της να εκφράσει το λογικό μέρος των αλγορίθμων. Τον έλεγχο τον προσφέρει η ίδια αυτοματοποιημένο, μέσα από τον μηχανισμό συμπερασματολογίας.

Υπάρχει πλήθος δημοσιεύσεων και βιβλίων που προσφέρουν έναν ανεξάντλητο αριθμό προγραμμάτων και εφαρμογών της Prolog. Ενα πολύ καλό βιβλίο που καλύπτει σε μεγάλη έκταση πλήθος εφαρμογών είναι αυτό των Sterling and Shapiro, όπως αναφέρεται και στην εισαγωγή των σημειώσεων. Οι εργασίες του Kowalski δίνουν στον αναγνώστη μια βαθύτερη αντίληψη της Prolog και γενικότερα του λογικού προγραμματισμού. Μέρος του βιβλίου του ίδιου συγγραφέα προσφέρει επίσης ένα σημαντικό αριθμό γενικών εφαρμογών λογικού προγραμματισμού που μπορούν να υλοποιηθούν με Prolog.

Εχει ακόμα προταθεί ένα σύστημα ανάπτυξης μεταφραστή (compiler) με Prolog, συστήματα επεξεργασίας συσχετιστικών βάσεων δεδομένων κλπ. Μιά σημαντική ακόμη εργασία είναι και η ανάπτυξη ενός διερμηνέα (interpreter) της Prolog σε Prolog.

Εξ'άλλου με δεδομένες τις επεκτάσεις που συζητήσαμε παραπάνω είναι δυνατόν να χρησιμοποιήσει κανείς την Prolog γιά συστήματα που απαιτούν τον πλήρη κατηγορικό λογισμό πρώτης τάξης. Τέτοιες εφαρμογές έχουν επίσης προταθεί και αφορούν την χρησιμοποίηση της Prolog σαν αποδείκτη θεωρημάτων γιά διάγνωση λαθών (fault diagnosis). Μετά την ανάπτυξη συστημάτων παράλληλης Prolog, άρχισαν να γίνονται και αναφορές εφαρμογών της.

7.8 Κατευθύνσεις της διαδικασίας απόδειξης

Μια σημαντική ακόμη άποψη της γενικότερης συμπερασματολογίας αναφέρεται στην κατεύθυνση της διαδικασίας απόδειξης. Η άποψη αυτή αναφέρεται στη μεθοδολογία αναζήτησης (search method) της λύσης σε ένα δεδομένο πρόβλημα. Κάθε πρόβλημα μπορεί να θεωρηθεί ότι αποτελείται από κάποιες αρχικές καταστάσεις, κάποιες τελικές καταστάσεις, και κάποιους κανόνες μετακίνησης μεταξύ των καταστάσεων.

Σε ένα λογικό πρόγραμμα σαν αρχική κατάσταση μπορούμε να θεωρήσουμε τα δεδομένα του προγράμματος, δηλαδή τις προτάσεις εκείνες που περιέχουν συμπεράσματα χωρίς συνθήκες. Τις προτάσεις αυτές τις ονομάζουμε γεγονότα. Τελική κατάσταση είναι ο στόχος (goal), δηλαδή η προς απόδειξη πρόταση.

Ολες οι υπόλοιπες προτάσεις αποτελούν τους κανόνες μετακίνησης μεταξύ των αρχικών και τελικών καταστάσεων, και θα τις ονομάσουμε απλά κανόνες. Τα συμπεράσματα των κανόνων μπορούμε να πούμε ότι αντιστοιχούν σε κάποιες ενδιάμεσες καταστάσεις μεταξύ αρχικών και τελικών.

Οπως αναφέραμε και στο προηγούμενο κεφάλαιο, υπάρχουν δύο κύριες κατευθύνσεις συμπερασματολογίας στην αναζήτηση της λύσης, η από πάνω προς τα κάτω (top-down) συμπερασματολογία, που ξεκινά από την τελική κατάσταση και οδηγείται προς τις αρχικές, και η από κάτω προς τα πάνω (bottom-up) συμπερασματολογία, που ξεκινά από τις αρχικές καταστάσεις και οδηγείται προς την τελική.

Η από πάνω προς τα κάτω συμπερασματολογία είναι μιά συμπερασματολογία καθοδηγούμενη από τους στόχους (goal-driven) που θέλουμε να αποδείξουμε. Γιά κάθε στόχο αναζητούμε τις συνθήκες του, τις οποίες και καθιστούμε νέους στόχους, κ.ο.κ. Είναι μιά μεθοδολογία οδηγούμενη από τους στόχους προς τα πίσω (backward reasoning), πρός τα τμήματα της γνώσης που είναι καθορισμένα χωρίς συνθήκες (γεγονότα, facts). Ετσι, θα μπορούσαμε να πούμε ότι η από πάνω προς τα κάτω συμπερασματολογία είναι μιά αναλυτική συμπερασματολογία.

Αντίθετα η από κάτω προς τα πάνω συμπερασματολογία, είναι μιά συμπερασματολογία καθοδηγούμενη από τα δεδομένα (data-driven) της γνώσης, των οποίων την ισχύ δεχόμαστε χωρίς τον καθορισμό κάποιων συνθηκών. Από κάθε τέτοιο κβάντο γνώσης προχωράμε μέσω των συνθηκών των προτάσεων σε συμπεράσματα, τα οποία και αποτελούν νέα στοιχεία που θα εκπληρώσουν συνθήκες προτάσεων. Προχωρώντας κατ'αυτό τον τρόπο φτάνουμε κάποια στιγμή στον ζητούμενο στόχο. Η μεθοδολογία αυτή οδηγείται προς τον στόχο, γι'αυτό θεωρείται και προς τα εμπρός συμπερασματολογία (forward reasoning). Με βάση αυτά θα μπορούσαμε να πούμε ότι η από κάτω προς τα πάνω συμπερασματολογία, είναι μιά συνθετική συμπερασματολογία.

Τα υπάρχοντα συστήματα χρησιμοποιούν τον ένα τύπο ή τον άλλο τύπο συμπερασματολογίας με τα αντίστοιχα πλεονεκτήματα/μειονεκτήματα.

Η συμπερασματολογία που καθοδηγείται από τους στόχους είναι τυφλή ως πρός τα γεγονότα του γνωστικού πεδίου, και ελπίζει πως επιλέγοντας τους κανόνες θα βρεί τελικά κάποια γεγονότα του λογικού προγράμματος που θα ικανοποιήσουν τις συνθήκες των κανόνων.

Αντίθετα η συμπερασματολογία που καθοδηγείται από τα δεδομένα είναι τυφλή ως πρός τον τελικό στόχο, και ενώ αποδεικνύει συνεχώς και νέα συμπεράσματα, ελπίζει πως κάποια στιγμή κάποιο από αυτά θα είναι και ο ζητούμενος στόχος. Υπάρχουν βέβαια συστήματα που χρησιμοποιούν μιά υβριδική προσέγγιση, εκμεταλλευόμενα τα πλεονεκτήματα και των δύο μεθοδολογιών.

Εκτός της διαδικασίας απόδειξης με την αρχή της απόφασης, υπάρχουν και άλλες διαδικασίες απόδειξης γιά τον κατηγορικό λογισμό πρώτης τάξης. Μιά τέτοια είναι η διαδικασία απόδειξης με χρησιμοποίηση γραφημάτων. Οι καταστάσεις που περιγράψαμε στην αρχή του κεφαλαίου μπορούν να θεωρηθούν και σαν κόμβοι ενός γραφήματος. Η αναζήτηση της λύσης είναι, σ'αυτή την περίπτωση, η αναζήτηση ενός μονοπατιού ανάμεσα στους κόμβους που οδηγεί από τις αρχικές καταστάσεις στις τελικές. Οι μεθοδολογίες αναζήτησης συνεχίζουν να ισχύουν και σ'αυτή την διαδικασία απόδειξης.

7.9 Στρατηγικές επιλογής υποψήφιων για επέκταση κόμβων

Εχουν καθορισθεί και επιπρόσθετες στρατηγικές οι οποίες καθοδηγούν τη γενικότερη αποδεικτική διαδικασία. Οι στρατηγικές αυτές αφορούν στην επιλογή των υποψήφιων γιά επέκταση κόμβων. Θα αναφέρουμε εδώ τις δύο σημαντικότερες στρατηγικές που είναι η αναζήτηση κατά βάθος (depth-first search strategy), και η αναζήτηση κατά πλάτος (breadth-first search strategy).

Η αναζήτηση κατά βάθος προσπαθεί κατ'αρχήν να βρεί μιά λύση γιά τις πρώτες συνθήκες των κανόνων που επιλέγει. Δηλαδή γιά κάθε νέο κανόνα που εμπλέκεται στην αποδεικτική διαδικασία, επιλέγει το πρώτο κατηγόρημα, και αναζητεί ένα άλλο κανόνα που ταιριάζει μ'αυτό. Προχωρώντας έτσι σταματάει όταν φτάσει σε ένα γεγονός, οπότε συνεχίζει την ίδια διαδικασία με το επόμενο κατά σειρά κατηγόρημα του προηγούμενου κανόνα.

Η αναζήτηση κατά πλάτος, πρίν προχωρήσει στην απόδειξη μιάς συνθήκης του παρόντος κανόνα, αναζητεί υποψήφιους γιά ταίριασμα κανόνες γιά κάθε μιά από τις συνθήκες του παρόντος κανόνα.

Εκτός από τις δύο αυτές στρατηγικές υπάρχουν και άλλες πιό πολύπλοκες στρατηγικές επιλογής υποψήφιων για επέκταση κόμβων. Για περισσότερες λεπτομέρειες πάνω σε τέτοιες στρατηγικές ο αναγνώστης μπορεί να ανατρέξει στην αναφορά.

7.10 Ευρεστικές συναρτήσεις

Θα κάνουμε μια ακόμα αναφορά σε μεθόδους αναζήτησης της απόδειξης σχολιάζοντας τις ευρεστικές συναρτήσεις (Heuristic functions). Η ευρεστικότητα είναι μιά τεχνική που βοηθάει στην ανακάλυψη λύσεων αν και δεν εγγυάται ότι δεν θα οδηγήσει πρός μιά λανθασμένη κατεύθυνση. Μια ευρεστική συνάρτηση είναι μιά συνάρτηση που δίνει ένα μέτρο επιθυμητότητας των κόμβων του υπό κατασκευή δέντρου απόδειξης.

Αυτό σημαίνει ότι ένα σύστημα συμπερασματολογίας συνοδευόμενο από μιά ευρεστική συνάρτηση, μπορεί σε κάθε βήμα να επιλέγει την καλύτερη εναλλακτική λύση και να κατευθύνει έτσι την αποδεικτική διαδικασία.

Οι ευρεστικές συναρτήσεις είναι ιδιαίτερα χρήσιμες σε περιπτώσεις που τα δέντρα απόδειξης είναι μεγάλα και υπάρχουν πολλές εναλλακτικές λύσεις, οπότε είναι σημαντικό να βρεθεί νωρίς το σωστό μονοπάτι γιά την λύση.

7.11 Ενας απλός αλγόριθμος ενός διερμηνέα της Prolog

Στο σημείο αυτό, θα περιγραφεί ένας πολύ απλός αλγόριθμος, που θα δωθεί σε μορφή διαγράμματος ροής, ο οποίος μπορεί να χρησιμοποιηθεί για την παραγωγή λύσεων, πολύ απλών λογικών προγραμμάτων, τα οποία περιέχουν προτάσεις Horn, δεν εμφανίζονται παράμετροι στα κατηγορήματα (προτασιακός λογισμός) και δεν χρησιμοποιούν ειδικά κατηγορήματα της PROLOG.

Υποθέτουμε ότι αποθηκεύουμε το λογικό πρόγραμμα σε ένα πίνακα lp(i,j), όπου το i αντιστοιχεί στον α/α των κανόνων-γεγονότων, και το j στον α/α των κατηγορημάτων για κάθε κανόνα. Μετράμε και τις τελείες. Για παράδειγμα, το λογικό πρόγραμμα “test” αποθηκεύεται στον πίνακα lp(i,j), όπως φαίνεται στο σχήμα 7.3. Υποθέτουμε ακόμα πως διαθέτουμε ένα άλλο δισδιάστατο πίνακα S_λ, που διαθέτει 6 πεδία ($goal_\lambda$, r_λ, t_λ, sup_λ, sub_λ, $pred_\lambda$), που λειτουργεί σαν σωρός (stack). Το λ είναι δείκτης στην “παρούσα” θέση στον σωρό, το ν είναι δείκτης στην πρώτη ελεύθερη θέση του σωρού.

ερώτηση ? P
δηλαδή top-goal:-P.
P: -A, B, C.
A.
B: -G, H.
C.
G.
H.

i	j → 1	2	3	4	5
↓ 1	top-goal	P	•		
2	P	A	B	C	•
3	A	•			
4	B	G	H	•	
5	C	•			
6	G	•			
7	H	•			
8	-|				

Σχ. 7.3 Το λογικό πρόγραμμα "test" και η αποθήκευσή του στον πίνακα lp(i,j).

$goal_\lambda$	r_λ	t_λ	sup_λ	sub_λ	$pred_\lambda$
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

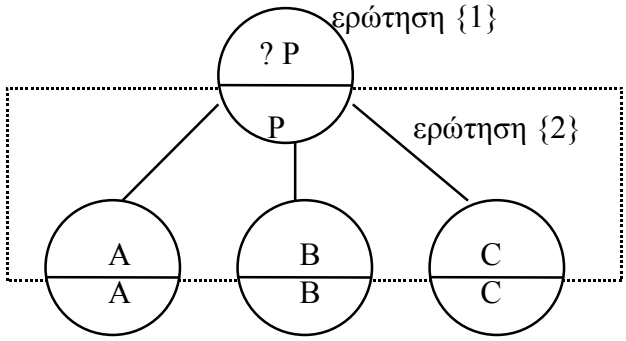
Σχ. 7.4 Ένα "στιγμιότυπο" του σωρού S.

Το $goal_\lambda$ είναι ο παρών στόχος (π.χ. top-goal, P, A, B κ.λ.π.). Το ζεύγος (r_λ, t_λ) αντιστοιχεί στο (i,j) του lp και δείχνει τη θέση μέσα στο lp(i,j) στην οποία έχει φτάσει η απόδειξη στον κανόνα του $goal_\lambda$. Τα r_λ, t_λ είναι rule number, term number αντίστοιχα. Το sup_λ (superior) δείχνει στον πατέρα του $goal_\lambda$ στο δένδρο απόδειξης. Το sub_λ (subordinate) δείχνει στο τελευταίο παιδί του $goal_\lambda$ στο δένδρο απόδειξης. Το $pred_\lambda$ (predecessor) δείχνει στον αριστερό αδελφό του $goal_\lambda$ στο δένδρο απόδειξης. Ο ίδιος ο σωρός S στο σύνολό του υλοποιεί το δένδρο απόδειξης σε μια γραμμική μορφή.

Για παράδειγμα το παρακάτω λογικό πρόγραμμα:

2. P: - A, B. C.
3. A.
4. B.
5. C.

με την ερώτηση ? P έχει το εξής δένδρο απόδειξης:



Αυτό υλοποιείται ως εξής:

lp	1	2	3	4	5
1	? P	P	•		
2	P	A	B	C	•
3	A	•			
4	B	•			
5	C	•			
6	-|				

S	goal	r	t	sup	sub	pred
1	? P	1	3	0	2	0
2	P	2	5	1	5	0
3	A	3	2	2	0	0
4	B	4	2	2	0	3
5	C	5	2	2	0	4

Ας σημειωθεί ότι το σύμβολο "-|" υποδηλώνει το τέλος του λογικού προγράμματος. Τα sup, sub, και pred δείχνουν σε θέσεις στον σωρό, και τα r, t δείχνουν σε θέσεις στο lp(i,j). Ο αλγόριθμος που περιγράφουμε στην επόμενη σελίδα παράγει όλα τα δένδρα απόδειξης (όλες τις λύσεις).

Εάν ακολουθήσουμε τον αλγόριθμο βήμα-βήμα για το παράδειγμα του σχήματος 7.3, θα αρχίσει να δημιουργείται σιγά-σιγά ο σωρός S:

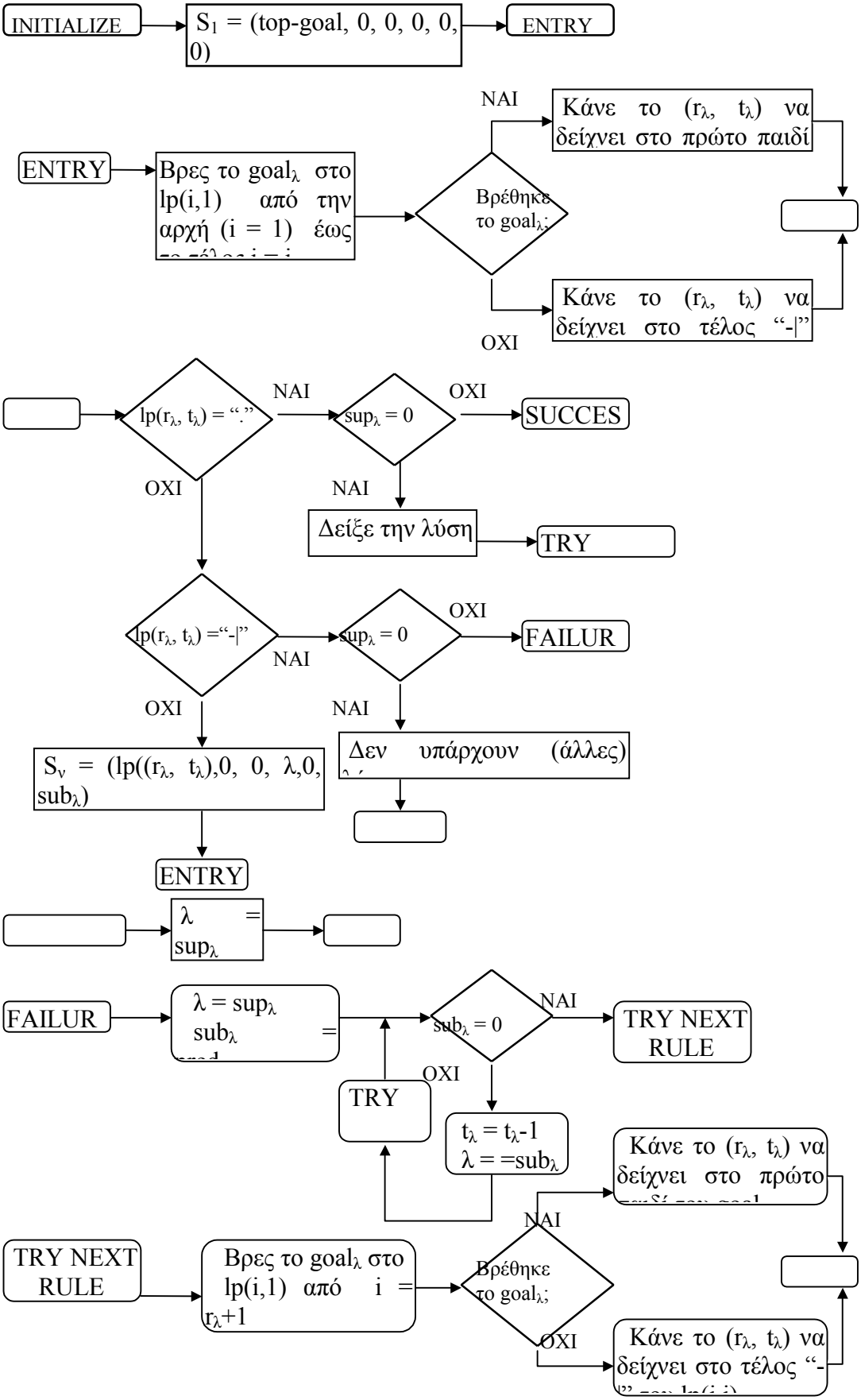
λ = 1 2 ~~3~~2 4 κ.λ.π.
ν = 2 3 4 5 κ.λ.π.

	$goal_\lambda$	r_λ	t_λ	sup_λ	sub_λ	$pred_\lambda$
1	top-goal	~~0~~1	~~0~~2	0	~~0~~2	0
2	P	~~0~~2	~~0~~2~~0~~3	1	~~0~~3~~0~~4	0
3	A	~~0~~3	~~0~~2	2	0	0
4	B	0	0	2	0	3
	.	.	.	.	.	.
	.	.	.	.	.	.
	.	.	.	.	.	.

κ.λ.π.

Οι τελικές τιμές των λ, ν, r_λ, t_λ, sup_λ, sub_λ, $pred_\lambda$ θα παρουσιασθούν μόλις τελειώσει ο αλγόριθμος. Ο σωρός στο αριστερό σχήμα έχει ενημερωθεί έως την στιγμή που θα βάλουμε και το B μέσα του.

Ο αλγόριθμος του διερμηνέα της Prolog



8

Βασικές έννοιες λογικών προγραμμάτων

8.1 Εισαγωγή

Ενα λογικό πρόγραμμα είναι ένα σύνολο από αξιώματα ή κανόνες οι οποίοι καθορίζουν σχέσεις ανάμεσα σε αντικείμενα. Υπολογισμός ενός λογικού προγράμματος είναι ένα συμπέρασμα από τα αποτελέσματα ενός προγράμματος. Ενα πρόγραμμα καθορίζει ένα σύνολο αποτελεσμάτων, τα οποία αποτελούν το νόημά του. Η ικανότητα του λογικού προγραμματισμού είναι να κατασκευάζει σαφή και κομψά προγράμματα τα οποία έχουν το επιθυμητό νόημα.

Οι βασικές έννοιες του λογικού προγραμματισμού, όροι και δηλώσεις, είναι κληρονομημένες από την λογική. Υπάρχουν τρεις βασικές δηλώσεις: τα γεγονότα (facts), οι κανόνες (rules) και οι ερωτήσεις (queries). Υπάρχει μια μοναδική δομή δεδομένων: ο λογικός όρος (logical term).

8.2 Γεγονότα

Το πιο απλό είδος της δήλωσης ονομάζεται *γεγονός (fact)*. Τα γεγονότα είναι ένα μέσο καθορισμού μιας σχέσης που ισχύει ανάμεσα στα αντικείμενα. Ενα παράδειγμα είναι:

father(abraham,isaac).

Αυτό το γεγονός λέει ότι ο Abraham είναι ο πατέρας του Isaac, ή ότι ο συσχετισμός *πατέρας (father)* ισχύει ανάμεσα στις μονάδες, οι οποίες ονομάζονται *abraham* και *isaac*. Αλλο ένα όνομα για μια σχέση είναι το *κατηγόρημα (predicate)*. Τα ονόματα των μονάδων είναι γνωστά ως *άτομα (atoms)*. Παρόμοια το *plus(2,3,5)* εκφράζει την σχέση του 2 συν 3, η οποία είναι το 5. Η γνωστή σχέση, *πρόσθεση (plus)* μπορεί να πραγματοποιηθεί διαμέσου ενός συνόλου γεγονότων τα οποία καθορίζουν τον πίνακα της πρόσθεσης. Ενα αρχικό τμήμα αυτού του πίνακα είναι:

plus(0,0,0).	*plus(0,1,1).*	*plus(0,2,2).*	*plus(0,3,4).*
plus(1,0,1).	*plus(1,1,2).*	*plus(1,2,3).*	*plus(1,3,4).*

Η επέκταση αυτού του πίνακα (ο οποίος δεν καταγράφεται εδώ ολόκληρος), με τις υπόλοιπες σχέσεις, π.χ. *plus(2,3,5).*, *plus(2,2,4),*, κ.τ.λ., θα θεωρηθεί ο ορισμός της σχέσης *πρόσθεση (plus)* σ'ολόκληρο το κεφάλαιο.

father(terach,abraham).	*male(terach).*
father(terach,nachor).	*male(abraham).*

```
father(terach,haran).          male(nachor).
father(abraham,isaac).              male(haran).
father(haran,lot).                  male(isaac).
father(haran,milcah).          male(lot).
father(haran,yiscah).

                                         female(sarah)
mother(sarah,isaac).           female(milcah).
                                         female(yiscah)
```

Πρόγραμμα 8.1 Μιά βάση δεδομένων με οικογενειακές σχέσεις από την Βίβλο

Ενα πεπερασμένο σύνολο από γεγονότα συγκροτεί ένα *πρόγραμμα (program)*. Αυτή είναι η πιο απλή μορφή ενός λογικού προγράμματος. Ενα σύνολο από γεγονότα είναι επίσης η περιγραφή μιας κατάστασης. Αυτή η επίγνωση είναι η βάση για τον προγραμματισμό βάσης δεδομένων, που θα συζητηθεί στο επόμενο κεφάλαιο. Ενα παράδειγμα βάσης δεδομένων των σχέσεων μιας οικογένειας από την Βίβλο είναι δοσμένο στο Πρόγραμμα 8.1. Τα κατηγορήματα *πατέρας (father), μητέρα (mother), αρσενικό (male)* και *θηλυκό (female)* εκφράζουν ολοφάνερα τις σχέσεις.

8.3 Απλές Ερωτήσεις

Η δεύτερη μορφή μιας δήλωσης στον λογικό προγραμματισμό είναι η *ερώτηση (query)*. Οι ερωτήσεις είναι το μέσο απόδοσης πληροφοριών από ένα λογικό πρόγραμμα. Με μια ερώτηση, ρωτάμε αν ισχύει κάποιος συσχετισμός ανάμεσα σε αντικείμενα. Για παράδειγμα η ερώτηση *father(abraham,isaac)?* ρωτάει, εάν η σχέση *πατέρας (father)* ισχύει ανάμεσα στον *abraham* και στον *isaac*. Με δεδομένο τα γεγονότα του Προγράμματος 8.1, η απάντηση είναι *ναι (yes)*.

Συντακτικά, οι ερωτήσεις και τα γεγονότα μπορούν να διαφοροποιηθούν από τα συμφραζόμενα. Η τελεία υποδεικνύει ένα γεγονός "*P*.", ενώ το ερωτηματικό υποδεικνύει μια ερώτηση ("*P?* ").

Στόχο (goal) ονομάζουμε την οντότητα χωρίς την τελεία ή το ερωτηματικό. Ενα γεγονός "*A*." δηλώνει ότι ο στόχος *A* είναι αληθής. Η ερώτηση "*A?*" ρωτάει εάν ο στόχος είναι αληθής. Μια απλή ερώτηση αποτελείται από έναν απλό στόχο.

Απαντώντας σε μια ερώτηση ***Q?*** αναφορικά με ένα πρόγραμμα ***P*** είναι σαν να ρωτάμε αν ο στόχος ***Q*** αποδεικνύεται από το πρόγραμμα ***P*** ή αν ο στόχος ***Q*** είναι λογικό συμπέρασμα του προγράμματος ***P***.

Τα λογικά συμπεράσματα έχουν παραχθεί με την εφαρμογή συμπερασματικών κανόνων. Ο πιο απλός κανόνας συμπεράσματος είναι η ***ταυτότητα (identity)***: από το *P* συμπεραίνουμε *P*. Μια ερώτηση είναι ένα λογικό συμπέρασμα ενός ίδιου γεγονότος.

Για να απαντήσουμε σε απλές ερωτήσεις εργαζόμαστε ως εξής : ψάχνουμε για ένα γεγονός μέσα στο πρόγραμμα, το οποίο συμπεραίνει την ερώτηση. Αν βρεθεί ένα γεγονός ίδιο με την ερώτηση, η απάντηση είναι *ναι (yes)*.

Η απάντηση *όχι (no)* δίνεται αν δεν βρεθεί ένα γεγονός ίδιο με την ερώτηση, γιατί τότε το γεγονός δεν είναι λογικό συμπέρασμα του προγράμματος. Αυτή η απάντηση δεν αντανακλά στην αλήθεια της ερώτησης. Απλώς λέει ότι αποτύχαμε να αποδείξουμε την ερώτηση αναφορικά με το πρόγραμμα αυτό. Αυτό μπορεί να οφείλεται στην υπόθεση του κλειστού κόσμου, στο οτι δηλαδή ο κόσμος τον οποίο γνωρίζουμε είναι μόνο αυτός που υπάρχει μέσα στο περιορισμένο λογικό πρόγραμμα στο οποίο απευθύνουμε ερωτήσεις. Και οι

δύο ερωτήσεις *female(abraham)?* και *plus(1,1,2)* θα απαντήσουν *όχι (no)* αναφορικά με το Πρόγραμμα 8.1.

8.4 Λογικές μεταβλητές

Μια λογική μεταβλητή αντιπροσωπεύει μια ακαθόριστη μονάδα και χρησιμοποιείται ανάλογα. Μελετείστε την χρήση της στις ερωτήσεις. Υποθέτουμε ότι θέλουμε να γνωρίσουμε τίνος πατέρας ήταν ο *abraham*. Ενας τρόπος είναι να ρωτήσουμε μια σειρά ερωτήσεων, *father(abraham,lot)?, father(abraham,milcah)?,* κ.τ.λ. μέχρι να δοθεί μια απάντηση *ναι (yes)*. Μια μεταβλητή επιτρέπει έναν καλύτερο τρόπο για να εκφράσει την ερώτηση *father(abraham,X)?* της οποίας η απάντηση είναι *X = isaac*.

Ετσι, οι *μεταβλητές είναι ένα είδος περίληψης πολλών ερωτήσεων*. Μια ερώτηση περιέχοντας μια μεταβλητή ρωτάει αν υπάρχει μια τιμή της μεταβλητής που να κάνει την ερώτηση ένα λογικό συμπέρασμα του προγράμματος.

Οι μεταβλητές στον λογικό προγραμματισμό συμπεριφέρονται διαφορετικά από τις μεταβλητές στις συμβατικές γλώσσες προγραμματισμού. Αυτές αντιπροσωπεύουν περισσότερο μια ακαθόριστη αλλά μοναδική οντότητα, παρά μια θέση μνήμης.

Εχοντας παρουσιάσει τις μεταβλητές, μπορούμε να ορίσουμε τους *όρους (terms),* τη μοναδική δομή δεδομένων στα λογικά προγράμματα. Ο ορισμός είναι επαγωγικός. Οι σταθερές και οι μεταβλητές είναι όροι. Επίσης όροι είναι και οι σύνθετοι όροι ή οι δομές. Ενας σύνθετος όρος περιέχει έναν *συναρτητή (functor)* (ονομάζεται ο κύριος συναρτητής του όρου) και μια σειρά από ένα ή περισσότερα ορίσματα, τα οποία είναι όροι. Ενας *συναρτητής (functor)* χαρακτηρίζεται από το όνομά του, το οποίο είναι ένα άτομο και από την πολλαπλότητα ή τον αριθμό των ορισμάτων του. Συντακτικά οι σύνθετοι όροι έχουν την μορφή $f(t_1, t_2, \ldots, t_n)$ όπου ο συναρτητής έχει όνομα *f*, είναι *πολλαπλότητας n*, και τα t_i είναι τα ορίσματα. Παραδείγματα των σύνθετων όρων περιλαμβάνουν τα: *s(0), hot(milk), name(john,doe), list(a,like(b,nil)), foo(X)* και *tree(tree(nil,3,nil),5,R)*.

Οι ερωτήσεις, οι στόχοι και πιο γενικά οι όροι όπου οι μεταβλητές δεν εμφανίζονται ονομάζονται *στοιχειώδεις (ground)*. Οπου οι μεταβλητές εμφανίζονται καλούνται *μη στοιχειώδεις (nonground)*. Για παράδειγμα, *foo(a,b)* είναι *στοιχειώδεις*, ενώ η *bar(X)* δεν είναι.

Ορισμός: Μια ***αντικατάσταση (substitution)*** είναι ένα πεπερασμένο σύνολο (πιθανόν άδειο) από ζευγάρια της μορφής $X_i = t_i$, όπου X_i είναι μια μεταβλητή, t_i είναι ένας όρος, $X_i \neq X_j$ για κάθε $i \neq j$, και το X_i δεν βρίσκεται στο t_j, για κάθε i και j.

Ενα παράδειγμα μιας αντικατάστασης που αποτελεί ένα μοναδικό ζευγάρι είναι $\{X = isaac\}$. Αντικαταστάσεις μπορούν να εφαρμοστούν στους όρους. Το αποτέλεσμα από την εφαρμογή μιας αντικατάστασης θ σε έναν όρο *A*, που φαίνεται από το $A\theta$, είναι ο όρος ο οποίος πετυχαίνεται από την αντικατάσταση κάθε εμφάνισης του *X* με το *t* στο *A*, για κάθε ζευγάρι $X = t$ μέσα στο θ.

Το αποτέλεσμα της εφαρμογής $\{X = isaac\}$ στον όρο *father(abraham,X)* είναι ο όρος *father(abraham,isaac)*.

Ορισμός: Το *A* είναι ένα ***στιγμιότυπο (instance)*** του *B* αν υπάρχει μια αντικατάσταση θ τέτοια ώστε $A = B\theta$.

Ο στόχος *father(abraham,isaac)* είναι ένα στιγμιότυπο του *father(abraham,X)* μέσα από την αντικατάσταση *{X = isaac}*. Παρόμοια ο *mother(sarah,isaac)* είναι ένα στιγμιότυπο του *mother(X,Y)* μέσα από την αντικατάσταση *{X = sarah, Y = isaac}*.

8.5 Υπαρξιακές ερωτήσεις

Αν εμφανίζονται μεταβλητές μέσα στις ερωτήσεις, τότε υπονοείται ότι αυτές βρίσκονται στην εμβέλεια ενός υπαρξιακού ποσοδείκτη. Τέτοιες ερωτήσεις θα μπορούσαμε να τις ονομάζουμε 'υπαρξιακές ερωτήσεις' (existential queries). Αυτό σημαίνει, ότι η ερώτηση *father(abraham,X)?* διαβάζεται: "Υπάρχει ένα *X* τέτοιο που ο *abraham* να είναι ο πατέρας του *X*;".

Πιο γενικά, μια ερώτηση $p(T_1, T_2, \ldots, T_n)?$ η οποία περιλαμβάνει τις μεταβλητές X_1, $X_2, \ldots, X_k$ διαβάζεται: "Υπάρχουν $X_1, X_2, \ldots, X_n$ έτσι ώστε $p(T_1, T_2, \ldots, T_n)$;". Για ευκολία, οι υπαρξιακοί ποσοδείκτες έχουν παραληφθεί.

Ο δεύτερος κανόνας συμπεράσματος που παρουσιάζουμε είναι η ***γενίκευση (generalization)***: μια ερώτηση *P* που περιέχει μεταβλητές είναι ένα λογικό συμπέρασμα ενός στιγμιότυπου από αυτό, το $P\theta$, για κάθε αντικατάσταση θ. Το γεγονός *father(abraham,isaac)* υπονοεί ότι υπάρχει ένα *X* τέτοιο ώστε το *father(abraham,X)* να είναι αληθές, δηλαδή *X = isaac*.

Λειτουργικά, για να απαντήσουμε μια υπαρκτή, μη στοιχειώδη ερώτηση χρησιμοποιώντας ένα πρόγραμμα με γεγονότα, βρίσκουμε ένα γεγονός το οποίο είναι ένα στιγμιότυπο της ερώτησης. Η απάντηση, ή η *λύση*, είναι αυτό το στιγμιότυπο. Η απάντηση είναι *όχι (no)* αν δεν υπάρχει κατάλληλο γεγονός στο πρόγραμμα.

Απαντώντας σε μη στοιχειώδεις ερωτήσεις, εκτελείται ένας υπολογισμός του οποίου η έξοδος είναι ένα στιγμιότυπο της ερώτησης. Εμείς μερικές φορές αντιπροσωπεύουμε αυτό το στιγμιότυπο με μια αντικατάσταση όπου αν εφαρμοστεί σε μια ερώτηση, καταλήγει σε μια λύση του στιγμιότυπου.

Γενικά, μια υπαρκτή ερώτηση ίσως έχει αρκετές λύσεις. Το Πρόγραμμα 8.1 δείχνει ότι ο Haran είναι ο πατέρας τριών παιδιών. Κατά συνέπεια η ερώτηση *father(haran,X)?* έχει τις λύσεις *{X = lot}*, *{X = milcah}, {X = yiscah}*. Αλλη ερώτηση με πολλαπλές λύσεις είναι η *plus(X,Y,4)?* η οποία βρίσκει αριθμούς των οποίων το άθροισμα ισούται με το 4. Οι λύσεις είναι, για παράδειγμα, *{X = 0, Y = 4}* και *{X = 1, Y = 3}*. Σημειώστε ότι οι διαφορετικές μεταβλητές *X* και *Y* ανταποκρίνονται σε διαφορετικά (πιθανόν) αντικείμενα.

Μια ενδιαφέρουσα παραλλαγή της τελευταίας ερώτησης είναι η *plus(X,X,4)?* η οποία εμμένει στο ότι οι δύο αριθμοί των οποίων το άθροισμα ισούται με το 4, είναι ίδιοι. Αυτή έχει μοναδική απάντηση *{X = 2}*.

8.6 Καθολικά Γεγονότα

Οι μεταβλητές είναι επίσης χρήσιμες στα γεγονότα. Υπέθεσε όλα τα Βιβλικά πρόσωπα, που συμπαθούν τον Pomegranates. Αντί να συμπεριλάβουμε στο πρόγραμμα ένα κατάλληλο γεγονός για κάθε μονάδα:

```
likes(abracham, pomegranates)
likes(sarah, pomegranates).
...
```

ένα γεγονός συμπάθειας *likes(X, pomegranates)* μπορεί να τα πει όλα. Κάνοντας χρήση αυτής της μεθόδου, *οι μεταβλητές είναι ένα μέσο περίληψης πολλών γεγονότων*. Το γεγονός *time(0, X, 0)* συνοψίζει όλα τα γεγονότα τα οποία ξεκινούν με το ότι 0 φορές κάποιος αριθμός ισούται με το 0.

Μεταβλητές μέσα σε γεγονότα βρίσκονται στην εμβέλεια ενός καθολικού ποσοδείκτη, το οποίο σημαίνει διαισθητικά ότι το γεγονός *likes(X, pomegranates)* δηλώνει ότι για όλα τα *X*, το *X* συμπάθεια τον pomegranates. Γενικά, ένα γεγονός $p(T_1, \ldots, T_n)$ το οποίο ερμηνεύεται: ότι όλα τα $X_1, \ldots, X_n$, όπου τα X_i είναι μεταβλητές, που εμφανίζονται μέσα στο γεγονός, $p(T_1, \ldots, T_n)$ είναι αληθή. Λογικά, από ένα τέτοιο γεγονός, κάποιος μπορεί να συνάγει κάθε στιγμιότυπο από αυτό. Για παράδειγμα, από *likes(X, pomegranates)* συμπεραίνουμε *likes(abraham, pomegranates)*.

Αυτός είναι ο τρίτος συμπερασματικός κανόνας, ο οποίος ονομάζεται ***αρχικοποίηση (instantiation)***. Από μια καθολικό ποσοδείκτη δήλωσης *P* συμπεραίνουμε ένα στιγμιότυπο $P\theta$ από αυτό, για κάθε αντικατάσταση θ.

Δύο ακαθόριστα αντικείμενα, τα οποία προσδιορίζονται από μεταβλητές, μπορούν εξαναγκασμένα να είναι όμοια, χρησιμοποιώντας το ίδιο όνομα μεταβλητής. Το γεγονός *plus(0,X,X)* εκφράζει ότι το *0* είναι μια αριστερή ταυτότητα της πρόσθεσης. Αυτό ερμηνεύεται πως για όλες τις τιμές του *X*, *0* συν *X* ισούται με *X*. Μια παρόμοια χρήση γίνεται όταν μεταφράζουμε την Αγγλική δήλωση "καθένας συμπαθεί τον εαυτό του" με το *likes(X,X)*.

Η απάντηση σε στοιχειώδη ερώτηση που βασίζεται σε ένα γεγονός το οποίο περιέχει μεταβλητές είναι προφανής. Ψάξτε για ένα γεγονός για το οποίο η ερώτηση είναι ένα στιγμιότυπο. Για παράδειγμα, η απάντηση στο *plus(0,2,2)?* η οποία είναι *ναι (yes)*, βασίζεται πάνω στο γεγονός *plus(0,X,X)*. Η απάντηση σε μια μη στοιχειώδη ερώτηση χρησιμοποιώντας ένα μη στοιχειώδες γεγονός εμπεριέχει ένα νέο ορισμό: ένα κοινό στιγμιότυπο με δύο όρους.

Ορισμός: Το *C* είναι ένα κοινό στιγμιότυπο του *A* και του *B* αν αυτό είναι ένα στιγμιότυπο του *B*. Με άλλα λόγια, αν υπάρχουν αντικαταστάσεις θ_1 και θ_2, τότε η $C = A\theta_1$, είναι συντακτικώς ταυτόσημη με τη $B\theta_2$.

Για παράδειγμα, οι στόχοι *plus(0,1,Y)* και *plus(0,X,X)* έχουν ένα κοινό στιγμιότυπο *plus(0,1,1)*. Εφαρμόζοντας την αντικατάσταση $\{Y = 1\}$ στο *plus(0,1,Y)* και την αντικατάσταση $\{X = 1\}$ στο *plus(0,X,X)* και οι δύο παράγουν το *plus(0,1,1)*.

Ως επί το πλείστον, προκειμένου να απαντήσουμε μια ερώτηση χρησιμοποιώντας ένα γεγονός, ψάχνουμε για ένα κοινό στιγμιότυπο της ερώτησης και του γεγονότος. Η απάντηση είναι το κοινό στιγμιότυπο αν υπάρχει, διαφορετικά η απάντηση είναι *όχι (no)*.

Η απάντηση σε μια υπαρκτή ερώτηση βασισμένη σε ένα γεγονός που περιέχει μεταβλητές, χρησιμοποιώντας ένα κοινό στιγμιότυπο, εμπλέκει δύο κανόνες συμπερασματολογίας. Το στιγμιότυπο συμπεραίνεται από το γεγονός του κανόνα της αρχικοποίησης και η ερώτηση συμπεραίνεται από το στιγμιότυπο του κανόνα της γενίκευσης.

8.7 Συζευκτικές Ερωτήσεις.

Μια σημαντική προέκταση στο είδος των ερωτήσεων, οι οποίες συζητήθηκαν μέχρι εδώ, είναι οι συζευκτικές ερωτήσεις. Συζευκτική ερώτηση είναι μια σύζευξη στόχων που εκφράζονται σε μια ερώτηση, για παράδειγμα *father(terach,X), father(X,Y)?* ή γενικά, $Q_1, \ldots, Q_n?$. Οι απλές ερωτήσεις είναι μια ειδική περίπτωση συζευκτικών ερωτήσεων όταν υπάρχει ένας απλός στόχος.

Στις πιο απλές συζευκτικές ερωτήσεις όλοι οι στόχοι είναι στοιχειώδεις, για παράδειγμα *father(abraham,isaac), male(lot).* Η απάντηση σε αυτή την ερώτηση χρησιμοποιώντας το Πρόγραμμα 8.1 είναι *ναι (yes)* καθώς και οι δύο στόχοι αποδεικνύονται από το πρόγραμμα. Γενικά, η ερώτηση Q_1, . . ., Q_n*?* όπου κάθε Q_i είναι ένας στοιχειώδης στόχος, έχει απαντηθεί με *ναι (yes)*, αναφορικά με το πρόγραμμα *P* αν κάθε Q_i εξυπακούεται από το *P*. Ως εκ τούτου οι στοιχειώδεις συζευκτικές ερωτήσεις δεν είναι πολύ ενδιαφέρουσες.

Οι συζευκτικές ερωτήσεις είναι ενδιαφέρουσες όταν υπάρχει μια ή περισσότερες *διαμοιραζόμενες μεταβλητές*, μεταβλητές που εμφανίζονται μέσα σε δύο διαφορετικούς στόχους ερωτήσεων. Ενα παράδειγμα είναι η ερώτηση *father(haran,X), male(X)?*. Η έκταση μιας μεταβλητής σε μια συζευκτική ερώτηση, είναι όλη η σύζευξη. Κατά συνέπεια η ερώτηση *p(X), q(X)?* εκφράζει: "Υπάρχει ένα *X* έτσι ώστε να ισχύει και τόσο *p(X)*, όσο και *q(X)?*". Οπως στις απλές ερωτήσεις, οι μεταβλητές στις συζευκτικές ερωτήσεις βρίσκονται στην εμβέλεια ενός ή περισσοτέρων υπαρξιακών ποσοδεικτών.

Οι διαμοιραζόμενες μεταβλητές χρησιμοποιούνται σαν ένα μέσο εξαναγκασμού μιας απλής ερώτησης, έτσι ώστε να περιορίσει την έκταση μιας μεταβλητής. Εχουμε κιόλας δει ένα παράδειγμα με την ερώτηση *plus(X,X,4)?* όπου το 4 είναι η λύση των αριθμών θα προστεθούν, η οποία περιοριζόταν στους αριθμούς που ήταν ίδιοι. Θεωρούμε την ερώτηση *father(haram,X), male(X)?*. Εδώ οι λύσεις της ερώτησης *father(haram,X)?* είναι περιορισμένες στα παιδιά τα οποία είναι αρσενικά. Το Πρόγραμμα 8.1 δείχνει ότι υπάρχει μόνο μια λύση, {*X = lot*}. Αντίστοιχα αυτή η ερώτηση μπορεί να εξεταστεί ως περιοριστική λύση στην ερώτηση *male(X)?*, στις μονάδες οι οποίες έχουν τον *Haran* για πατέρα.

Μία ελαφρώς διαφορετική χρήση μιας διαμοιραζόμενης μεταβλητής μπορεί να γίνει αντιληπτή στην ερώτηση *father(terach,X), father(X,Y)?*. Από την μια πλευρά αυτό περιορίζουμε τους υιούς του *terach* σε εκείνους οι οποίοι είναι οι ίδιοι πατέρες. Από την άλλη πλευρά θεωρούμε μονάδες *Y*, των οποίων οι πατέρες είναι υιοί του *terach*. Υπάρχουν αρκετές λύσεις, για παράδειγμα {*X = abraham, Y = isaac*}, και {*X = haran, Y = lot*}.

Μια συζευκτική ερώτηση είναι ένα λογικό συμπέρασμα ενός προγράμματος *P*, αν όλοι οι στόχοι μέσα στην σύζευξη είναι λογικά συμπεράσματα του *P*, δεδομένου ότι οι διαμοιραζόμενες μεταβλητές έχουν γίνει στιγμιότυπα στις ίδιες τιμές μέσα σε διαφορετικούς στόχους.

Λειτουργικά, για να λύσουμε μια συζευκτική ερώτηση A_1, A_2, . . ., A_n*?* χρησιμοποιώντας ένα πρόγραμμα *P*, βρίσκουμε μια αντικατάσταση θ έτσι ώστε η $A_1\theta$ και . . . και η $A_n\theta$ να είναι στοιχειώδη στιγμιότυπα για τα γεγονότα μέσα στο *P*. Η ίδια αντικατάσταση εφαρμόστηκε σε όλους τους στόχους, εξασφαλίζοντας ότι οι στοιχειώδεις μεταβλητές είναι κοινές καθ' όλη την ερώτηση.

Για παράδειγμα, υποθέστε την ερώτηση *father(haran,X), male(X)?* αναφορικά με το Πρόγραμμα 8.1 Εφαρμόζοντας την αντικατάσταση {*X = lot*} στην ερώτηση, καταλήγουμε σε ένα στοιχειώδες στιγμιότυπο *father(haram, lot), male(lot)?,* το οποίο είναι ένα συμπέρασμα του προγράμματος.

8.8 Κανόνες

Ενδιαφέρουσες συζευκτικές ερωτήσεις καθορίζουν σχέσεις ανεξάρτητα άλλων παραγόντων. Η ερώτηση *father(haran,X), male(X)?* ρωτάει για έναν γιο του *Haran*. Η ερώτηση *father(haran,X), father(X,Y)?* ρωτάει για εγγόνια του *Terach*. Αυτό μας φέρνει στην τρίτη και πιο σημαντική δήλωση στον λογικό προγραμματισμό, ο *κανόνας (rule)*, ο οποίος μας καθιστά ικανούς στο να καθορίσουμε νέες σχέσεις στους όρους των υφιστάμενων σχέσεων. Οι κανόνες είναι δηλώσεις της μορφής:

$$A \leftarrow B_1, B_2, \ldots, B_n$$

όπου $n \geq 0$. Το A είναι η κεφαλή του κανόνα και τα B_i είναι το σώμα του. Τόσο το A, όσο και τα B_i είναι στόχοι. Οι κανόνες, τα γεγονότα και οι ερωτήσεις καλούνται επίσης **φράσεις Horn** (Horn clauses) ή εν συντομια **φράσεις**. Σημειώστε ότι ένα γεγονός είναι απλώς μια ειδική περίπτωση ενός κανόνα όταν $n = 0$. Τα γεγονότα καλούνται επίσης **μοναδιαίες φράσεις** (unit clauses). Υπάρχει επίσης ένα ειδικό όνομα για φράσεις με έναν στόχο μέσα στο σώμα, δηλαδή όταν $n = 1$. Μια τέτοια φράση καλείται **επαναληπτική φράση** (iterative clause). Οσο για τις μεταβλητές που εμφανίζονται μέσα σε κανόνες, θεωρούνται ότι βρίσκονται στην εμβέλεια καθολικών ποσοδεικτών που η έκτασή τους είναι όλος ο κανόνας.

Ενας κανόνας ο οποίος εκφράζει την σχέση υιός (son) είναι:

son(X,Y) ← father(Y,X), male(X).

Ομοίως μπορεί να καθορίσει ένας κανόνας για την σχέση της κόρης:

daughter(X,Y) ← father(X,Y), female(X).

Ενας κανόνας για την σχέση παππού είναι ο εξής:

grandfather(X,Z) ← father(X,Y), father(Y,Z).

Οι κανόνες μπορούν να εξεταστούν με δύο τρόπους. Πρώτα, είναι ένα μέσο με το οποίο εκφράζονται νέες ή σύνθετες ερωτήσεις χρησιμοποιώντας απλές ερωτήσεις. Μια ερώτηση *son(X,haran)?* στο πρόγραμμα το οποίο περιλαμβάνει τον προηγούμενο κανόνα είναι μεταφρασμένο στην ερώτηση *father(haran,X), male(X)?* σύμφωνα με τον κανόνα και λύνεται όπως προηγουμένως. Η ερμηνεία κανόνων με αυτό τον τρόπο, αποτελεί την **διαδικαστική ερμηνεία (*procedural meaning*)**. Η διαδικαστική ερμηνεία για τον κανόνα του παππού είναι: "Να απαντήσουμε στην ερώτηση: *είναι ο X ο παππούς του Y?*, απαντώντας τη συζευκτική ερώτηση: *είναι ο X ο πατέρας του Z και ο Z ο πατέρας του Y"*.

Η δεύτερη άποψη για τους κανόνες βγαίνει από την μετάφραση του κανόνα όπως ένα λογικό αξίωμα. Το ανάποδο βέλος "←" χρησιμοποιείται για να δείξει λογική συνέπεια. Ο κανόνας *υιός (son)* ερμηνεύεται ως: Ο *X* είναι υιός του *Y*, αν ο *Y* είναι ο πατέρας του *X* και ο *X* είναι άρρεν". Κάτω από αυτό το πρίσμα οι κανόνες είναι ένα μέσο καθορισμού νέων ή σύνθετων σχέσεων χρησιμοποιώντας άλλες, απλές σχέσεις. Το κατηγόρημα *υιός (son)* καθορίζεται με τα κατηγορήματα *πατέρας (father)* και *άρρενό (male)*. Η ερμηνεία αυτή ενός κανόνα είναι γνωστή ως η **δηλωτική ερμηνεία (*declarative meaning*)**. Η δηλωτική ερμηνεία του κανόνα *παππού (grandfather)* είναι: "Για όλα τα *X, Y* και *Z*, ο *X* είναι παππούς του Y αν ο X είναι ο πατέρας του Z και ο Z είναι ο πατέρας του Y".

Αν και τυπικά όλες οι μεταβλητές μέσα σε μια φράση βρίσκονται στην εμβέλεια καθολικού ποσοδείκτη, εμείς θα αναφερόμαστε κάποιες φορές στις μεταβλητές οι οποίες εμφανίζονται μέσα στο σώμα μιας πρότασης, αλλά όχι μέσα στο κεφάλι, σαν να ήταν στην εμβέλεια ενός υπαρξιακού ποσοδείκτη μέσα στο σώμα. Για παράδειγμα, ο κανόνας *παππούς (grandfather)*

grandfather(X,Z) ← father(X,Y), father(Y,Z).

μπορεί να διαβαστεί:

"*Για όλα τα X και Y, ο X είναι ο παππούς του Y, αν υπάρχει ένα Z τέτοιο ώστε ο X να είναι ο πατέρας του Z και ο Z ο πατέρας του Y*".

Η αιτιολογία για αυτόν τον τυπικό μετασχηματισμό δεν θα δοθεί και εμείς θα τον χρησιμοποιούμε σαν ένα πλεονέκτημα. Οπου αυτό είναι πηγή παρανόησης, ο αναγνώστης μπορεί να καταφεύγει πίσω στην τυπική ερμηνεία μιας φράσης, μέσα στην οποία όλες οι μεταβλητές είναι καθολικά ποσοτικοποιημένες, εξωτερικά.

Το να ενσωματώσουμε κανόνες συμπερασματολογίας μέσα στο πλαίσιο της λογικής απόδειξης στα πλαίσια του λογικού προγραμματισμού, χρειαζόμαστε την αρχή του Γενικευμένου Modus Ponens (M.P.).

Ορισμός: Η αρχή του ***Γενικευμένου Modus Ponens*** *(M.P.)* λέει ότι από τον κανόνα

$$R = (A \leftarrow B_1, B_2, \ldots, B_n.)$$

και τα γεγονότα B_1'. B_2'. ... B_n'.

το A' μπορεί να καταλήξει συμπέρασμα, αν

$$A' \leftarrow B_1', B_2', \ldots, B_n'.$$

είναι ένα στιγμιότυπο του R.

Το γενικευμένο *Modus Ponens (MP)* περιλαμβάνει τους συμπερασματικούς κανόνες της ταυτότητας (identity) και της αρχικοποίησης (instantiation) σαν ειδικές περιπτώσεις. Είμαστε τώρα σε θέση να δώσουμε έναν ολοκληρωμένο ορισμό της έννοιας ενός λογικού προγράμματος και της συσχετισμένης έννοιας του λογικού συμπεράσματός του.

Ορισμός: Ενα *λογικό πρόγραμμα (logic program)* είναι ένα πεπερασμένο σύνολο κανόνων.

Ορισμός: Ενας στόχος G είναι λογικό συμπέρασμα ενός προγράμματος P, αν υπάρχει μια φράση μέσα στο P με ένα στοιχειώδες στιγμιότυπο $G' \leftarrow B_1, \ldots, B_n$, $n \geq 0$, τέτοια ώστε τα $B_1, \ldots, B_n$ να είναι λογικά συμπεράσματα του P και το G' είναι ένα στιγμιότυπο του G.

Σημειώστε ότι ο στόχος G είναι ένα λογικό συμπέρασμα του προγράμματος P αν και μόνο αν το G συνεπάγεται από το P, με έναν πεπερασμένο αριθμό εφαρμογών στον κανόνα του γενικευμένου Modus Ponens (MP).

Θεωρείστε την ερώτηση *son(S,haran)?* αναφορικά με το Πρόγραμμα 8.1 η οποία ορίζεται από τον κανόνα για το *υιό* (*son*). Η αντικατάσταση {*X = lot, Y = haran*} που εφαρμόστηκε στον κανόνα, δίνει το στιγμιότυπο *son(lot,haran)* ← *father(haram,lot), male(lot)*. Και οι δύο στόχοι στο σώμα αυτού του κανόνα είναι γεγονότα στο Πρόγραμμα 8.1. Κατά συνέπεια η γενικευμένη αρχή του *Modus Ponens (M.P.)* υποδηλώνει την ερώτηση της οποίας η απάντηση είναι *S = lot*.

Λειτουργικά, η απάντηση στις ερωτήσεις, απεικονίζεται στον ορισμό του λογικού συμπεράσματος. Φανταστείτε ένα στοιχειώδες στιγμιότυπο ενός στόχου και ένα στοιχειώδες στιγμιότυπο ενός κανόνα και αναδρομικά απαντήστε τη συζευκτική ερώτηση στο σώμα

αυτού του κανόνα. Δηλαδή, για να επαληθεύσετε έναν στόχο A, σε ένα πρόγραμμα P, διαλέξτε έναν κανόνα $A_1 \leftarrow B_1, B_2, \ldots, B_n$ μέσα στο πρόγραμμα P, και θεωρείστε αντικατάσταση θ τέτοια ώστε $A = A_1\theta$ και $B_i\theta$ στοιχειώδη για $1 \leq i \leq n$. Ύστερα αναδρομικά επαληθεύστε κάθε $B_i\theta$.

Ενα άλλο θέμα που σχετίζεται με τους κανόνες είναι το κατά πόσο είναι πλήρεις. Κατά πόσο δηλαδή εκφράζουν με συνέπεια αυτό που πραγματικά θέλουμε να πούμε. Για παράδειγμα, ο κανόνας που έχει δοθεί για το *υιό (son)* είναι σωστός, αλλά είναι ελλειπής ο ορισμός της σχέσης. Ετσι, δεν μπορούμε να συμπεράνουμε ότι ο *Isaac* είναι υιός του *Sarah*. Αυτό το οποίο λείπει, είναι ότι το παιδί μπορεί να είναι υιός μιας μητέρας όπως υιός ενός πατέρα. Ενας νέος κανόνας που εκφράζει την σχέση μπορεί να προστεθεί, συγκεκριμένα:

son(X,Y) ← mother(Y,X), male(X)

Παρόμοια, για να καθορίσουμε την συγγένεια *παππούς (grandparent)* θα μπορούσαμε να πάρουμε τέσσερις κανόνες για να συμπεριλάβουμε και τις δύο περιπτώσεις *πατέρα (father)* και *μητέρας (mother)*:

grandparent(X,Z) ← father(X,Y), father(Y,Z).
grandparent(X,Z) ← father(X,Y), mother(Y,Z).
grandparent(X,Z) ← mother(X,Y), father(Y,Z).
grandparent(X,Z) ← mother(X,Y), mother(Y,Z).

Υπάρχει ένας καλύτερος, πιο συνεπτυγμένος, τρόπος για να εκφράσουμε αυτούς τους κανόνες. Χρειαζόμαστε να ορίσουμε την βοηθητική σχέση, *πατέρας (parent)* όπως είναι ένας πατέρας ή μια μητέρα. Μέρος αυτής της δυνατότητας του λογικού προγραμματισμού είναι το να αποφανθεί στο τι ενδιάμεσα κατηγορήματα να καθορίσει για να επιτύχει μια πλήρη, κομψή δημιουργία ενός αξιώματος μιας σχέσης. Αυτοί οι κανόνες οι οποίοι ορίζουν το *γονέα (parent)* είναι ευθείς, συλλαμβάνοντας τον ορισμό ενός γονέα ως πατέρα ή μητέρα. Τα λογικά προγράμματα ενσωματώσουν εναλλακτικούς ορισμούς, ή περισσότερους τεχνικούς διαχωρισμούς, έχοντας εναλλακτικούς κανόνες, όπως για τον γονέα:

parent(X,Y) ← father(X,Y).
parent(X,Y) ← mother(X,Y).

Οι κανόνες για τον *υιό (son)* και τον *γονέα (grandparent)* είναι τώρα, περισσότερο ειδικευμένοι:

son(X,Y) ← parent(Y,X), male(X).
grandparent(X,Y) ← parent(X,Z), parent(Z,X).

Μια συλλογή κανόνων με το ίδιο κατηγόρημα στην κεφαλή, όπως το ζευγάρι των κανόνων του γονέα, ονομάζεται *διαδικασία (procedure)*. Θα δούμε αργότερα ότι κάτω από λογική μεταγλώττιση αυτών των κανόνων στην Prolog, τέτοια συλλογή κανόνων είναι όντως το ανάλογο των διαδικασιών ή υπορουτινών μέσα σε συμβατικές γλώσσες προγραμματισμού.

8.9 Ενας απλός θεωρητικός διερμηνέας.

Μια λειτουργική διαδικασία απάντησης ερωτήσεων έχει ανεπίσημα περιγραφεί και προοδευτικά αναπτυχθεί στις προηγούμενες ενότητες. Εμείς θα ασχοληθούμε εδώ με τις λεπτομέρειες, προκειμένου για να παρουσιάσουμε έναν θεωρητικό διερμηνέα για λογικά προγράμματα. Σύμφωνα με τον περιορισμό του *Modus Ponens (M.P.)* στους στοιχειώδεις στόχους, ο διερμηνέας απαντά μόνο σε στοιχειώδεις ερωτήσεις.

Ο θεωρητικός διερμηνέας εκτελεί *ναι/όχι (yes/no)* υπολογισμούς. Αυτός παίρνει ένα πρόγραμμα *P* και στοιχειώδη ερώτηση *Q* και δίνει ως έξοδο *ναι (yes)* αν *Q* είναι συμπέρασμα του *P* και αλλιώς δίνει *όχι (no)*. Ο διερμηνέας μπορεί ωστόσο να αποτύχει να τερματίσει την εκτέλεση του αλγορίθμου αν ο στόχος δεν αποτελεί συμπέρασμα του προγράμματος, οπότε σε τέτοια περίπτωση δεν παράγει καμια απάντηση. Κάτι τέτοιο συμβαίνει αν ο διερμηνέας πέσει σε άπειρα επαναλαμβανόμενο βρόγχο (ατέρμονη αναδρομή). Τα βήματα του διερμηνέα δίνονται στο Σχήμα 8.1.

Ο τρέχων στόχος σε κάθε φάση του υπολογισμού ονομάζεται ***αποφαινόμενο (resolvent)***. Ενα ***ίχνος (trace)*** του διερμηνέα είναι η ακολουθία των αποφαινόμενων που παράγονται κατά την διάρκεια του υπολογισμού, μαζί με τις επιτυχημένες επιλογές. Θεωρήστε την ερώτηση *son(lot,haran)?* αναφορικά με το πρόγραμμα 8.2, που αποτελείται από ένα υποσύνολο των γεγονότων του προγράμματος 8.1 μαζί με κανόνες που ορίζουν τον *υιό (son)* και την *κόρη (daughter)*. Το σχήμα 8.2 είναι ένα ίχνος της διαδικασίας απάντησης της ερώτησης.

Το ίχνος αποτελεί μιά έμμεση απόδειξη της στοιχειώδους ερώτησης από το πρόγραμμα. Μια περισσότερο βολική αναπαράσταση της απόδειξης είναι το δέντρο απόδειξης. Καθορίζουμε τις απαραίτητες έννοιες.

Ορισμός: Σ*τοιχειώδης μείωση (ground reduction)* του στόχου *G* από το πρόγραμμα *P* είναι η αντικατάσταση του *G* από το σώμα ενός στοιχειώδους στιγμιότυπου ενός κανόνα του *P*, του οποίου η κεφαλή πανομοιότυπη με τον επιλεγμένο στόχο.

Δηλαδή :

G?
G' ← A1,A2,...,An.

Εστω θ τέτοιο ώστε G = G'θ. Τότε ο στόχος G εξαφανίζεται και αντικαθίσταται από την στοιχειώδη συζευκτική ερώτηση

A1θ,A2θ,...,Anθ?

η οποία αποτελεί μια σύζευξη νέων στόχων

A1θ?, A2θ? ,..., Anθ?

Μια μείωση είναι το βασικό υπολογιστικό βήμα στον λογικό προγραμματισμό. Αυτό ανταποκρίνεται σε μια εφαρμογή του γενικευμένου *Modus Ponens (M.P.)*. Αυτό επίσης ανταποκρίνεται σε μια επανάληψη του (*loop while)* του διερμηνέα στο Σχήμα 8.1. Ο στόχος που αντικαταστάθηκε μέσα σε μια μείωση, είναι ***μειωμένος (redused)***, και οι νέοι στόχοι είναι ***παραγόμενοι (derived)***.

```
father(abraham,isaac).
father(haran,lot).
father(haran,milcah).
father(haran,yiscah).
female(milcah).
female(yiscah).
male(lot).
```

male(isaac).
son(X,Y) ← father(Y,X), male(X).
daughter(X,Y) ← father(Y,X), male(X).

Πρόγραμμα 8.2: Οι σχέσεις της Βιβλικής οικογένειας.

Είσοδος: Μια περιοχή ερώτησης *Q* και ένα πρόγραμμα *P*.

Εξοδος: *yes* αν μια απόδειξη της *Q* από το *P* είχε βρεθεί, αλλιώς *όχι (no)*.

Αλγόριθμος: Αρχικοποίησε το διαλυτικό της *Q*

While τα διαλυτικά $A_1, \ldots, A_n$ δεν είναι άδεια
begin
διάλεξε ένα προορισμό A_i, $1 \le i \le n$, και
μια περιοχή περιπτώσεων από μιας πρόταση
$A \leftarrow B_1, B_2, \ldots, B_k$, $k \ge 0$ στο *P*,
τέτοιο που το $A = A_i$
(αν καμια τέτοια πρόταση δεν υπάρχει,
βγές από το loop);
$A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_n$
end
Αν το διαλυτικό είναι άδειο, τότε δώσε στην έξοδο *ναι (yes); αλλιώς δώσε έξοδο όχι (no)*.

Σχήμα 8.1: Ενας θεωρητικός διερμηνέας για λογικό πρόγραμμα

Συνδέουμε τώρα αυτές τις έννοιες με το παράδειγμα ίχνους στο σχήμα 8.2. Αυτές είναι τρεις μειώσεις μέσα στο ίχνος. Η πρώτη μειώνει τον στόχο *son(lot,haram)* και δημιουργεί δύο παραγόμενους στόχους, *father(kharam,lot)* και *male(lot)*. Η δεύτερη μείωση είναι του *father(haram, lot)* χωρίς να δημιουργεί παραγόμενους στόχους. Η τρίτη μείωση επίσης, δεν δημιουργεί παραγόμενους στόχους μειώνοντας το *male(lot)*.

Ενα ***δέντρο απόδειξης (proof tree)*** αποτελείται από κόμβους και τόξα. Η ρίζα του δέντρου απόδειξης για μια απλή ερώτηση είναι η ίδια η ερώτηση. Οι κόμβοι αναπαριστάνουν τους μειωμένους στόχους κατά την διάρκεια του υπολογισμού. Καθώς όμως κάθε μειωμένος στόχος έχει συνταιριάξει με την κεφαλή ενός κανόνα ή με ένα γεγονός, οι κόμβοι του δέντρου απόδειξης αναπαριστάνουν ταυτόχρονα κεφαλές των επιλεγμένων κανόνων ή επιλεγμένων γεγονότων. Οι κόμβοι λοιπόν του δέντρου είναι οι στόχοι οι οποίοι έχουν παραχθεί κατά την διάρκεια του υπολογισμού. Εάν ο στόχος δεν συνταίριαξε με κεφαλή κανόνα αλλά με γεγονός, τότε ο κόμβος είναι και φύλλο του δέντρου (καθώς δεν υπάρχουν παραγόμενοι στόχοι). Από έναν κόμβο φεύγει ένα ή περισσότερα καθοδηγούμενα τόξα. Κάθε τέτοιο τόξο συνδέει ένα μειωμένο στόχο με ένα παραγόμενο στόχο. Το δέντρο απόδειξης για μια

συζευκτική ερώτηση είναι η συλλογή των δέντρων απόδειξης κάθε μεμονωμένου στόχου της σύζευξης.

Είσοδος: *son(lot,haran)?* και το Πρόγραμμα 8.2
Το αποφαινόμενο δεν είναι άδειο
Διάλεξε *son(lot,haran)* (η μόνη επιλογή)
Διάλεξε *son(lot,haran)* ← *father(haran,lot), male(lot).*
Το νέο αποφαινόμενο είναι *father(haran,lot), male(lot)?*
Το αποφαινόμενο δεν είναι άδειο
Διάλεξε *father(haran,lot)*
Διάλεξε *father(haran,lot).*
Το νέο αποφαινόμενο είναι *male(lot)?*
Το αποφαινόμενο δεν είναι άδειο
Διάλεξε *male(lot)*
Διάλεξε *male(lot).*
Το νέο αποφαινόμενο είναι άδειο

Εξοδος: *ναι (yes)*

Σχήμα 8.2: Η ιχνηλασία του διερμηνέα

Υπάρχουν δύο επιλογές μέσα στον διερμηνέα. Πρέπει κατ'αρχήν να επιλεγεί ο στόχος που θα είναι υποψήφιος για μείωση. Αυτό συμβαίνει διότι από μιά απλή ερώτηση, εφόσον επιλεγεί κανόνας, όλοι οι παραγόμενοι στόχοι, που βρίσκονται στο σώμα του κανόνα, είναι και οι υποψήφιοι γιά μείωση στόχοι. Πρέπει λοιπόν με κάποιο κριτήριο να επιλεγεί κάποιος από αυτούς για μείωση.Πρέπει επίσης να επιλεγεί μιά φράση του προγράμματος (γεγονός ή κανόνας) η οποία μετα από μιά αρχικοποίηση των μεταβλητών της μπορεί να ταιριάξει με τον επιλεγμένο για μείωση στόχο. Τότε λέμε ότι ένα κατάλληλο στοιχειώδες στιγμιότυπο μιάς φράσης του προγράμματος ταιριάζει με τον επιλεγμένο προς μείωση στόχο.

Οι δύο επιλογές έχουν πολύ διαφορετικό χαρακτήρα. Η επιλογή του στόχου που μειώνεται είναι αυθαίρετη. Σε κάθε δοσμένο αποφαινόμενο όλοι οι στόχοι πρέπει να είναι μειωμένοι. Μπορεί να δειχθεί ότι, στον καθαρό λογικό προγραμματισμό, η σειρά των μειώσεων δεν έχει σημασία, προκειμένου να βρούμε μια απόδειξη. Αυτό ισχύει, γιατί αν υπάρχει μια απόδειξη για έναν στόχο, τότε υπάρχει μια απόδειξη με όποια σειρά κι αν γίνουν οι μειώσεις. Σε όρους της απόδειξης δέντρου, αυτό σημαίνει ότι η σειρά των κλαδιών είναι άσχετη. Στην Prolog βέβαια η επιλογή αυτή δεν είναι αυθαίρετη αλλά ακολουθεί μιά σειρά από αριστερά προς τα δεξιά.

Αντίθετα, η επιλογή της φράσης και ενός κατάλληλου στοιχειώδους στιγμιότυπο είναι κρίσιμη. Γενικά, υπάρχουν αρκετές επιλογές μιας φράσης και η λάθος επιλογή μπορεί να οδηγήσει σε λάθος μονοπάτια απόδειξης, τα οποία δηλαδή δεν οδηγούν τελικά στην απόδειξη του στόχου. Αλλες επιλογές μπορεί να οδηγήσουν σε ατέρμονες αναδρομικές κλήσεις. Η επιλογή στον λογικό προγραμματισμό θεωρείται ότι γίνεται μη ντετερμινιστικά (nondeterministically). Και πάλι όμως, στην Prolog, ακολουθείται μια συγκεκριμένη πολύ απλή στρατηγική επιλογής φράσεων. Η σειρά επιλογής είναι διατεταγμένη από πάνω προς τα κάτω. Αυτό σημαίνει ότι γιά κάθε νέο στόχο, το πρόγραμμα των φράσεων εξετάζεται από πάνω προς τα κάτω έως ότου βρεθεί μιά φράση της οποίας ένα στιγμιότυπο μπορεί να ταιριάξει με τον στόχο.

Η έννοια της **μη ντετερμινιστικής επιλογής** χρησιμοποιείται στον ορισμό πολλών μοντέλων υπολογισμού, όπως για παράδειγμα στα πεπερασμένα αυτόματα και στις μηχανές Turing, και αποτελεί μια πολύ ισχυρή θεωρητική έννοια. Μια μη ντετερμινιστική επιλογή είναι μια ακαθόριστη επιλογή από έναν αριθμό εναλλακτικών (επιλογών), οι οποίες υποτίθεται ότι φτιάχνονται με έναν "διορατικό" τρόπο: αν κάποια, μόνο από τις εναλλακτικές, οδηγούν σε έναν επιτυχή υπολογισμό (στην δική μας περίπτωση, να βρίσκει μια απόδειξη), τότε μια από αυτές είναι επιλεγμένη. Τυπικά, η έννοια είναι ορισμένη όπως παρακάτω:

"*Ενας υπολογισμός που περιλαμβάνει μη ντετερμινιστικές επιλογές ορίζεται ότι επιτυγχάνει αν υπάρχει τουλάχιστον μια ακολουθία μη ντετερμινιστικών επιλογών που οδηγούν σε επιτυχία*".

Φυσικά, καμια πραγματική μηχανή δεν μπορεί να εκτελεί απευθείας αυτόν τον ορισμό. Εν τούτοις, αυτό μπορεί να προσεγγιστεί με έναν χρήσιμο τρόπο, όπως περατώνεται στην Prolog.

Ο διερμηνέας που δόθηκε στο Σχήμα 8.1 μπορεί να επεκταθεί στην απάντηση της περιοχής υπαρξιακών ερωτήσεων με ένα επιπρόσθετο αρχικό βήμα: Μάντεψε ένα στοιχειώδες στιγμιότυπο αυτής της ερώτησης. Αυτό το βήμα είναι ίδιο με το βήμα του διερμηνέα που υπολογίζει στοιχειώδη στιγμιότυπα των κανόνων. Είναι δύσκολο γενικά να μαντέψουμε ένα κατάλληλο στοιχειώδες στιγμιότυπο, αφού αυτό θα σήμαινε ότι γνωρίζουμε εκ των προτέρων ποιές είναι οι σωστές απαντήσεις. Στην πραγματικότητα χρειάζεται κάποιος μηχανισμός που θα φροντίζει να βρίσκει ένα στοιχειώδες στιγμιότυπο με κάποιο μηχανιστικό τρόπο χωρίς να είναι απαραίτητο εμείς να το μαντεύουμε.

Ενα σημαντικό μέγεθος που παρέχεται από τα δέντρα απόδειξης είναι ο αριθμός των κόμβων σε ένα δέντρο. Αυτό δείχνει πόσα πολλά βήματα μείωσης εκτελούνται σε έναν υπολογισμό.

8.9 Η ερμηνεία ενός λογικού προγράμματος

Πώς μπορούμε να ξέρουμε αν ένα λογικό πρόγραμμα ορίζει τις έννοιες ακριβώς με τον τόπο που θέλουμε; Πως μπορούμε να γνωρίζουμε αν το λογικό πρόγραμμα είναι ορθό ή λανθασμένο; Προκειμένου να απαντήσουμε σε τέτοιες ερωτήσεις, πρέπει να ορίσουμε ποιά είναι η ερμηνεία ενός λογικού προγράμματος.

Ορισμός: Η ***ερμηνεία (meaning),*** *M(P)*, ενός λογικού προγράμματος *P*, είναι το σύνολο των στοιχειωδών μοναδιαίων φράσεων (δηλ. στοιχειωδών κατηγορημάτων) που συμπεραίνονται από το *P*.

Από τον ορισμό αυτόν, έπεται ότι η ερμηνεία ενός λογικού προγράμματος που αποτελείται από στοιχειώδη μόνο γεγονότα, όπως το Πρόγραμμα 8.1, είναι το ίδιο το πρόγραμμα. Με άλλα λόγια, για απλά προγράμματα, το πρόγραμμα "σημαίνει ακριβώς ότι αυτό λέει".
Οταν το λογικό πρόγραμμα περιέχει και κανόνες ή καθολικά γεγονότα, τότε πρέπει να διατρέξουμε 'νοερά' το πρόγραμμα και να βρούμε όλες τις δυνατές στοιχειώδεις μοναδιαίες φράσεις που συμπεραίνονται από το πρόγραμμα. Αυτό βέβαια δεν είναι τόσο εύκολο γιά ένα σύνθετο λογικό πρόγραμμα. Δεν είναι Με βάση τον παραπάνω ορισμό, η ερμηνεία του λογικού προγράμματος 8.2 είναι το παρακάτω σύνολο στοιχειωδών μοναδιαίων φράσεων :

{ *father(abraham,isaac), father(haran,lot), father(haran,milcah), father(haran,yiscah),.*
female(milcah), female(yiscah), male(lot), male(isaac), son(isaac, abraham),

son(lot, haran), daughter(milcah, haran), daughter(yiscah, haran) }

Αυτό το παράδειγμα δείχνει πως η ερμηνεία ενός προγράμματος περιέχει ρητά ότιδήποτε δηλώνεται έμμεσα.

Ορίσαμε και εξηγήσαμε την έννοια της ερμηνείας ενός λογικού προγράμματος. Δηλαδή, μπορούμε τώρα να παράγουμε την ερμηνεία ενός οποιουδήποτε λογικού προγράμματος. Ομως αυτό δεν αρκεί. Ο λόγος είναι ότι πρέπει να συγκρίνουμε την ερμηνεία ενός προγράμματος με την προσδοκώμενη σημασία του.

Ορισμός: Η **προσδοκώμενη σημασία (intended meaning)** ενός προγράμματος είναι ένα σύνολο από στοιχειώδεις μοναδιαίες φράσεις που ορίζονται από τον συγγραφέα του προγράμματος.

Οταν γράφουμε ένα πρόγραμμα Prolog προσπαθούμε με διαδοχικές αποσφαλματώσεις και διορθώσεις του πηγαίου κώδικα να το κάνουμε να απαντά με ορθό τρόπο στις ερωτήσεις που του κάνουμε. Τι είναι οι ερωτήσεις που του κάνουμε; Είναι στιγμιότυπα της προσδοκώμενης σημασίας. Γιατί το πρόγραμμα δεν απαντά εξαρχής σωστά; Οχι βέβαια γιατί ο υπολογιστής λειτουργεί με λάθος τρόπο αλλά διότι η ερμηνεία του προγράμματος που γράψαμε δεν ταυτίζεται με την προσδοκώμενη σημασία του.

Ας υποθέσουμε ότι έχουμε ένα πρόγραμμα με τις σχέσεις *πατέρας*, *μητέρα*, *άρρεν*, *θήλυ* και θέλουμε να γράψουμε τον ορισμό της κόρης (daughter). Η προσδοκώμενη σημασία του προγράμματος είναι :

{ father(peter, george), father(peter, john), father(peter,maria), mother(helen,georgia), mother(helen,roula), male(peter), male(george), male(john), female(helen), female(maria), female(georgia), female(roula), daughter(maria, peter), daughter(georgia, helen), daughter(roula, helen) }

Το πρόγραμμα που γράφουμε λοιπόν έχει ως εξής :

```
father(peter, george).
father(peter, john).
father(peter,maria).
mother(helen,georgia).
mother(helen,roula).
male(peter).
male(george).
male(john).
female(helen).
female(maria).
female(roula).
daughter(X,Y) ← father(Y,X), female(X).
```

Ομως το πρόγραμμα αυτό έχει την παρακάτω ερμηνεία :

{ father(peter, george), father(peter, john), father(peter,maria), mother(helen,georgia), mother(helen,roula), male(peter), male(george), male(john), female(helen), female(maria), female(roula), daughter(maria, peter) }

Παρατηρούμε ότι η στοιχειώδης μοναδιαία φράση *daughter(roula, helen)* δεν αποδεικνύεται από το πρόγραμμα. Σκεφτόμαστε λίγο και καταλήγουμε στο συμπέρασμα ότι ξεχάσαμε να εισάγουμε τον παρακάτω κανόνα στο λογικό πρόγραμμα :

daughter(X,Y) ← mother(Y,X), female(X).

Πράγματι, τώρα το λογικό πρόγραμμα είναι πληρέστερο. Η ερμηνεία του προγράμματος είναι τώρα η παρακάτω :

{ *father(peter, george), father(peter, john), father(peter,maria), mother(helen,georgia),*
mother(helen,roula), male(peter), male(george), male(john), female(helen), female(maria),
female(roula), daughter(maria, peter), daughter(roula, helen) }

Η ενσωμάτωση του επιπλέον κανόνα έφερε δραματική αλλαγή στην ερμηνεία του λογικού προγράμματος. Παρόλα αυτά η στοιχειώδης μοναδιαία φράση *daughter(georgia, helen)* δεν αποδεικνύεται από το πρόγραμμα. Σκεφτώμαστε λίγο και καταλήγουμε στο συμπέρασμα ότι ξεχάσαμε να εισάγουμε τον παρακάτω γεγονός στο λογικό πρόγραμμα :

female(georgia).

Τώρα η ερμηνεία του προγράμματος είναι πανομοιότυπη με την προσδοκώμενη σημασία του. Οπότε το λογικό προγραμμα αντιπροσωπεύει αυτό ακριβώς που θέλαμε να αναπαραστήσουμε.

Με τους παραπάνω ορισμούς και τα παραδείγματα η σχέση ανάμεσα στην ερμηνεία και στην προσδοκώμενη σημασία ενός προγράμματος γίνεται πιό σαφής. Μπορούμε να ελέγξουμε αν είναι ορθό οτιδήποτε εκφράζει το πρόγραμμα, ή αν το πρόγραμμα εκφράζει οτιδήποτε εμείς θέλουμε αυτό να πει.

Λέμε ότι ένα πρόγραμμα είναι ***ορθό (correct)*** αναφορικά προς κάποια προσδοκώμενη σημασία *M* αν η ερμηνεία του, *M(P)*, είναι ένα υποσύνολο του *M*. Με άλλα λόγια, ένα ορθό πρόγραμμα δεν καταλήγει σε απροσδόκητα συμπεράσματα (μπορεί όμως να μην καταλήγει σε όλα τα προσδοκώμενα συμπεράσματα).

Ενα πρόγραμμα είναι ***πλήρες (complete)*** αναφορικά με την *M* αν η *M* είναι ένα υποσύνολο της *M(P)*. Με άλλα λόγια, ένα πλήρες πρόγραμμα καταλήγει σε όλα τα προσδοκώμενα συμπεράσματα (μπορεί επίσης να καταλήγει και σε άλλα συμπεράσματα).

Ετσι, ένα πρόγραμμα *P* είναι ορθό και πλήρες αναφορικά με μιά προσδοκώμενη ερμηνεία *M* αν *M = M(P)*.

Η ορθότητα και η πληρότητα που εισάγονται έτσι θυμίζουν τις αντίστοιχες έννοιες στην προτασιακή και κατηγορηματική λογική και πράγματι αποτελούν αντίστοιχες έννοιες.

Αν και η ιδέα της αλήθειας δεν είναι ορισμένη πλήρως εδώ, θα πούμε ότι ένα στοιχειώδες ενδεχόμενο είναι *αληθές (true)* αναφορικά με την προσδοκώμενη ερμηνεία αν αυτό είναι ένα μέλος της, αλλιώς είναι *ψευδές (false)*.

Καθ' όλη την διάρκεια των σημειώσεων, όταν χρησιμοποιούνται σημαντικά κατηγορήματα και σταθερές, η προσδοκώμενη ερμηνεία των προγραμμάτων υπονοείται διαισθητικά από την επιλογή των ονομάτων.

9

Προγραμματισμός Βάσεων Γνώσης

9.1 Εισαγωγή

Υπάρχουν δύο βασικοί τρόποι χρησιμοποίησης των λογικών προγραμμάτων: ορίζοντας μια βάση γνώσης ή λογική βάση δεδομένων, και χρησιμοποιώντας δομές δεδομένων. Αυτό το κεφάλαιο μελετά τον προγραμματισμό βάσεων δεδομένων. Μια λογική βάση δεδομένων αποτελείται από ένα σύνολο γεγονότων και κανόνων. Σε ένα σύνολο γεγονότων κάποιος μπορεί να ορίσει σχέσεις, όπως στις σχεσιακές βάσεις δεδομένων. Οι κανόνες μπορούν να ορίσουν πολύπλοκα σχεσιακά ερωτήματα, όπως στη σχεσιακή άλγεβρα. Ενα λογικό πρόγραμμα μπορεί να εκφράσει τις λειτουργίες που σχετίζονται με τις σχεσιακές βάσεις δεδομένων.

9.2 Απλές Βάσεις Δεδομένων

Ξεκινάμε ανατρέχοντας στο Πρόγραμμα 8.1, την βιβλική βάση δεδομένων, και την αύξησή της με κανόνες που εκφράζουν τις σχέσεις οικογένειας. Η ίδια η βάση δεδομένων είχε τέσσερα βασικά κατηγορήματα, *father/2*, *mother/2*, *male/1*, και *female/1*. Υιοθετούμε την παραδοχή από την θεωρία βάσεων δεδομένων και δίνουμε για κάθε σχέση, ένα *σχήμα σχέσης* το οποίο καθορίζει το ρόλο κάθε θέσης στη σχέση (ή ορίσματος στον στόχο) που πρόκειται να αναπαραστήσει. Σχήματα σχέσης για τα παραπάνω τέσσερα κατηγορήματα είναι, αντίστοιχα, τα *father(Father,Child)*, *mother(Mother,Child)*, *male(Person)*, και *female(Person)*. Τα μνημονικά ονόματα μιλάνε από μόνα τους.

Υιοθετούμε την τυπογραφική παραδοχή ότι τα σχήματα σχέσης δίνονται σε πλάγια γράμματα (italics). Οι μεταβλητές δίνονται με μνημονικά ονόματα στους κανόνες, αλλά συνήθως είναι *X* και *Y* όταν μελετάμε ερωτήσεις (queries). Ονόματα που αποτελούνται από πολλές λέξεις γράφονται είτε με κολλημένες τις λέξεις μεταξύ τους ή με χρήση του συμβόλου '_', (underscore), μεταξύ των λέξεων. Κάθε νέα λέξη σε μια μεταβλητή ξεκινά με κεφαλαίο γράμμα, για παράδειγμα, NieceOrNephew, ενώ οι λέξεις διαχωρίζονται με underscores για κατηγόρημα και όνομα συνάρτησης, για παράδειγμα, schedule_conflict.

Νέες σχέσεις κατασκευάζονται από αυτές τις βασικές σχέσεις ορίζοντας κατάλληλους κανόνες. Κατάλληλα σχήματα σχέσεων για τις σχέσεις που παρουσιάζονται στο προηγούμενο κεφάλαιο είναι:

```
son(Son,Parent),
daughter(Daughter,Parent),
parent(Parent,Child),
grandparent(Grandparent,Grandchild).
```

Από λογικής απόψεως, είναι ασήμαντο ποιές σχέσεις καθορίζονται από γεγονότα και ποιές από κανόνες. Για παράδειγμα, αν η διαθέσιμη βάση δεδομένων αποτελείται από τα γεγονότα parent, male, και female, οι κανόνες που ορίζουν τις σχέσεις son και grandparent εξακολουθούν να είναι σωστές. Πρέπει να γραφούν νέοι κανόνες για τις σχέσεις που δεν καθορίζονται πλέον από τα γεγονότα, δηλαδή τις father και mother. Κατάλληλοι κανόνες είναι οι εξής:

father(Dad,Child) ← parent(Dad,Child), male(Dad).
mother(Mum,Child) ← parent(Mum,Child), female(Mum).

Μπορούμε ακόμη να αποκτήσουμε ενδιαφέροντες κανόνες κάνοντας άμεσες τις σχέσεις που υπάρχουν μόνο σαν έμμεσες στη βάση δεδομένων. Για παράδειγμα, εφόσον γνωρίζουμε το πατέρα και την μητέρα ενός παιδιού, ξέρουμε και ποια ζευγάρια απέκτησαν απογόνους, ή για να χρησιμοποιήσουμε τον βιβλικό όρο, τεκνοποίησαν. Αυτό δεν δίνεται άμεσα από την βάση δεδομένων αλλά ένας απλός κανόνας μπορεί να γραφεί για να ανακτήσει αυτή την πληροφορία. Το σχήμα της σχέσης είναι *procreated(Man,Woman)*:

procreated(Man,Woman) ← father(Man,Child), mother(Woman,Child).

Αυτό διαβάζεται ως εξής: "Ο *Man* και η *Woman* τεκνοποίησαν αν υπάρχει ένα *Child* τέτοιο ώστε ο *Man* να είναι ο πατέρας του *Child* και η *Woman* να είναι η μητέρα του *Child*".

Ενα άλλο παράδειγμα για πληροφορίες που μπορούν να ανακτηθούν από τις απλές υπάρχουσες πληροφορίες είναι οι σχέσεις αδελφών - brothers και sisters. Δίνουμε έναν κανόνα για τον *brother(Brother,Sibling).*

Brother(Brother,Sib) ← parent(Parent,Brother), parent(Parent,Sib), male(Brother).

Αυτό διαβάζεται ως εξής: "Ο *Brother* είναι αδελφός του *Sib* αν ο *Parent* είναι πατέρας και των δύο *Brother* και *Sib*, και ο *Brother* είναι αρσενικός."

Υπάρχει ένα πρόβλημα στον ορισμό του αδελφού που δόθηκε παραπάνω. Η ερώτηση *brother(X,X)?* ικανοποιείται για κάθε αρσενικό παιδί *X*, που δεν συμφωνεί με τον τρόπο που αντιλαμβανόμαστε την σχέση αδελφού. Για να αποφευχθούν τέτοιες περιπτώσεις από την έννοια του προγράμματος εισάγουμε ένα κατηγόρημα '≠', που συντάσσεται σε ενδοδιάταξη : Term1 ≠ Term2. Το Term1 ≠ Term2 είναι αληθές εάν το Term1 είναι διαφορετικό από το Term2. Προς το παρόν, περιορίζεται σε σταθερούς όρους. Κατ' αρχήν, μπορεί να οριστεί από έναν πίνακα *X≠Y* για κάθε δύο διαφορετικά άτομα *X* και *Y* στον τομέα του ενδιαφέροντος. Το σχήμα 9.1 δίνει τον κατάλληλο πίνακα για το Πρόγραμμα 8.1.

abraham≠isaac	abraham≠haran	abraham≠lot
abraham≠milcah	abraham≠yiscah	isaac≠haran
isaac≠lot	isaac≠milcah	issac≠yiscah
haran≠lot	haran≠milcah	haran≠yiscah
lot≠milcah	lot≠yiscah	milcah≠yiscah

Σχήμα 9.1: Ορίζοντας την ανισότητα

Ο νέος κανόνας για τον αδελφό είναι:

brother(Brother,Sib) ←

parent(Parent,Brother),
parent(Parent,Sib),
male(Brother), Brother ≠ Sib.

Οσο πιο πολλές σχέσεις υπάρχουν, τόσο πιο εύκολο είναι να οριστούν πολύπλοκότερες σχέσεις. Το Πρόγραμμα 9.1 ορίζει τις σχέσεις *uncle(Uncle, NieceOrNephew)*, *sibling(Sib1, Sib2)*, και *cousin(Cousin1, Cousin2)*.

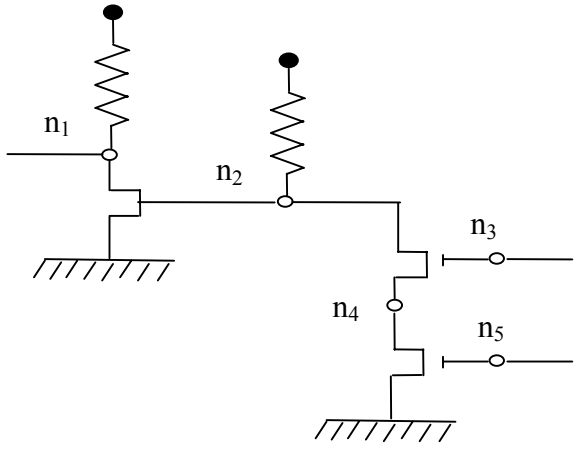
uncle(Uncle,Person) ←
brother(Uncle,Parent), parent(Parent,Person).
sibling(Sib1,Sib2) ←
parent(Parent,Sib1), parent(Parent,Sib2), Sib1≠Sib2.
cousin(Cousin1,Cousin2) ←
parent(Parent1,Cousin1),
parent(Parent2,Cousin2),
sibling(Parent1,Parent2).

Πρόγραμμα 9.1: Ορίζοντας τις σχέσεις οικογένειας

Μια άλλη σχέση που δεν είναι σαφής στην βάση δεδομένων μια οικογένειας είναι αν η γυναίκα είναι μητέρα. Αυτό καθορίζεται χρησιμοποιώντας τις σχέσεις *mother/2*. Το νέο σχήμα σχέσης είναι mother(Woman) και ορίζεται από τον κανόνα:

mother(Woman) ← mother(Woman,Child).

Αυτό διαβάζεται: Μια *Woman* είναι μητέρα εάν είναι μητέρα κάποιου *Child*. Προσέξτε ότι, έχουμε χρησιμοποήσει το ίδιο όνομα κατηγόρημα, *mother*, για να περιγράψουμε δύο διαφορετικές σχέσεις *mother*. Το κατηγόρημα mother δέχεται διαφορετικό αριθμό ορισμάτων, λ.χ., έχει διαφορετική πολλαπλότητα στις δυο περιπτώσεις. Γενικά, το ίδιο όνομα κατηγορήματος δηλώνει μια διαφορετική σχέση, όταν έχει διαφορετικό αριθμό ορισμάτων.



Εικόνα 9.2: Ενα λογικό κύκλωμα

Αλλάζουμε παραδείγματα, για να μη μας περιπλέξει το παράδειγμα των οικογενειακών σχέσεων.Εστω ότι έχουμε να περιγράψουμε απλά λογικά κυκλώματα. Ενα κύκλωμα μπορεί να θεωρηθεί με δυο τρόπους. Ο πρώτος είναι το τοπολογικό σχέδιο των φυσικών συστατικών του μερών, που συνήθως περιγράφονται με ένα διάγραμμα κυκλώματος. Ο δεύτερος είναι η αλληλεπίδραση των λειτουργικών μονάδων του. Και οι δυο απόψεις μπορούν να

τοποθετηθούν εύκολα μέσα σε ένα λογικό πρόγραμμα. Το διάγραμμα κυκλώματος αναπαρίσταται από μια συλλογή γεγονότων, ενώ οι κανόνες περιγράφουν τα λειτουργικά συστατικά του μέρη.

Το Πρόγραμμα 9.2 είναι μια Βάση Δεδομένων που δίνει μια απλοποιημένη άποψη της λογικής πύλης *and* σχεδιασμένης στην Εικόνα 9.2. Τα γεγονότα είναι οι σύνδεσμοι των συγκεκριμένων αντιστάσεων και transistors που αποτελούν το κύκλωμα. Το σχήμα σχέσεων για τις αντιστάσεις είναι *resistor(End1,End2)* και για τα transistors είναι transistor(Gate,Source,Drain).

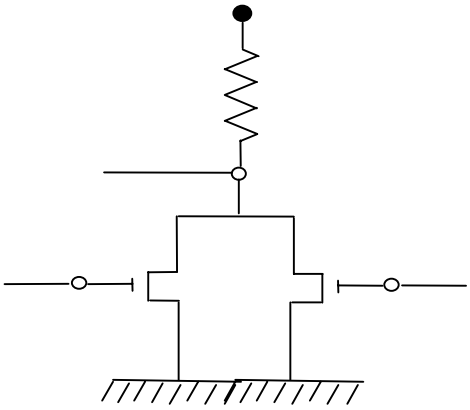
Το πρόγραμμα δείχνει το στυλ σχολιασμού των λογικών προγραμμάτων που θα ακολουθήσουμε σε όλο το βιβλίο. Κάθε ενδιαφέρουσα διαδικασία προηγείται από ένα σχήμα σχέσης για την διαδικασία, και αγγλικό κείμενο που ορίζει την σχέση. Συστήνουμε αυτό το στυλ σχολιασμού, το οποίο δίνει έμφαση στο επεξηγηματικό διάβασμα των προγραμμάτων, και για προγράμματα σε Prolog.

```
resistor(power,n1).
resistor(power,n2).
transistor(n2,ground,n1).
transistor(n3,n4,n2).
transistor(n5,ground,n4).

irverter(Input,Output) ←
    Output είναι το αντίστροφο του Input.
    inverter(Input,Output) ←
        transistor(Input,ground,Output),
        resistor(power,Output).

nand_gate(Input1,Input2,Output) ←
    Output είναι το λογικό nand τουInput1 και τουInput2.
    nand_gate(Input1,Input2,Output) ←
        transistor(Input1,X,Output),
        transistor(Input2,ground,X),
        resistor(power,Output).

and_gate(Input1,Input2,Output) ←
    Output είναι το λογικό and τουInput1 και του Input2.
    and_gate(Input1,Input2,Output) ←
        nand_gate(Input1,Input2,X),
        inverter(X,Output).
```

Πρόγραμμα 9.2: Το κύκλωμα για μια λογική πύλη and

Συγκεκριμένες διατάξεις αντιστάσεων και transistors ικανοποιούν ρόλους που διαφαίνονται μέσω των κανόνων που ορίζουν τα λειτουργικά μέρη του κυκλώματος. Το κύκλωμα παριστάνει μια πύλη *and*, η οποία δέχεται δύο σήματα εισόδου και παράγει ως έξοδο το λογικό *and* αυτών των σημάτων. Ενας τρόπος για να φτιάξουμε μια πύλη *and* είναι να συνδέσουμε μια πύλη *nand* με έναν αντιστροφέα. Σχήματα σχέσης για αυτά τα τρία μέρη είναι τα *and_gate(Input1,Input2,Output)*, *nand_gate(Input1,Input2,Output)*, και *inverter(Input, Output)*.

Για να εκτιμήσουμε το Πρόγραμμα 9.2, ας διαβάσουμε τον κανόνα του αντιστροφέα. Αυτός δηλώνει ότι ένας αντιστροφέας αποτελείται από ένα transistor με την πηγή συνδεδεμένη στο έδαφος, και μια αντίσταση της οποίας η μία άκρη συνδέεται με την παροχή ρεύματος. Η πύλη του transistor είναι η είσοδος στον αντιστροφέα, ενώ η ελεύθερη άκρη της αντίστασης πρέπει να συνδέεται με τη γείωση του transistor, η οποία αποτελεί την έξοδο του αντιστροφέα.



Σχήμα 9.3: Μία πύλη OR

Εστω η ερώτηση *and_gate(In1,In2,Out)?* στο Πρόγραμμα 9.2. Δίνει την απάντηση {*In1=n3,In2=n5,Out=n1*}. Αυτή η λύση επιβεβαιώνει ότι το κύκλωμα που περιγράφθηκε από τα γεγονότα είναι μια πύλη *and*, και υποδεικνύει τις εισόδους και την έξοδο.

9.3 Δομημένα δεδομένα και αφαίρεση δεδομένων

Ενας περιορισμός του Προγράμματος 9.2 για την περιγραφή της πύλης *and* είναι η αντιμετώπιση του κυκλώματος ως μαύρο κουτί. Δεν υπάρχει καμία ένδειξη της δομής του κυκλώματος στην απάντηση της ερώτησης *and_gate*, παρόλο που η δομή έχει έμμεσα χρησιμοποιηθεί για την εύρεση της απάντησης. Οι κανόνες μας λένε ότι το κύκλωμα αναπαριστά μια πύλη *and*, αλλά η δομή της πύλης *and* υπάρχει μόνο έμμεσα.Αυτό διορθώνεται προσθέτοντας ένα επιπλέον όρισμα σε καθέναν από τους στόχους της βάσης δεδομένων. Για ομοιομορφία, το επιπλέον όρισμα γίνεται το πρώτο όρισμα. Τα χαμηλού επιπέδου γεγονότα απλά αποκτούν ένα στοιχείο αναγνώρισης (identifier). Προχωρώντας από αριστερά προς τα δεξιά στο διάγραμμα του Σχήματος 9.2, ονομάζουμε τις αντιστάσεις *r1* και *r2*, και τα transistors *t1*, *t2*, και *t3*.
Τα ονόματα των λειτουργικών μερών θα πρέπει να αντικατοπτρίζουν την δομή τους. Ενας αντιστροφέας αποτελείται από ένα transistor και μια αντίσταση. Για να τον αναπαραστήσουμε χρειαζόμαστε δομημένα δεδομένα. Η τεχνική είναι να χρησιμοποιήσουμε έναν σύνθετο όρο, *inv(T,R)*, όπου *T* και *R* είναι αντίστοιχα τα ονόματα των μερών του αντιστροφέα, δηλαδή του transistor και της αντίστασης. Ανάλογα, το όνομα της πύλης *nand* θα είναι *nand(T1,T2,R)*, όπου *T1*, *T2*, και *R* τα ονόματα των δύο transistors και της αντίστασης που συνθέτουν μια πύλη *nand*. Τελικά, μια πύλη *and* μπορεί να ονομαστεί με

τους όρους ενός αντιστροφέα και μιας πύλης *nand*. Ο μορφοποιημένος κώδικας που περιλαμβάνει τα ονόματα εμφανίζεται στο Πρόγραμμα 9.3.

Η ερώτηση *and_gate(G,In1,In2,Out)?* έχει λύση την *{G=and(nand(t2,t3,r2), inv(t1,r1)), In1=n3,In2=n5,Out=n1}*. Οι In1, In2 και Out έχουν τις προηγούμενες τιμές τους. Η περίπλοκη δομή για το *G* αντικατοπτρίζει επακριβώς την λειτουργική σύνθεση της πύλης *and*.

```
resistor(R,Node1,Node2) ←
  R είναι μια αντίσταση μεταξύ των Node1 και Node2.
        resistor(r1,power,n1).
        resistor(r2,power,n2).

transistor(T,Gate,Source,Drain) ←
   T είναι ένα transistor του οποίου η πύλη είναι η Gate,
  πηγή η Source, και γείωση η Drain.
        transistor(t1,n2,ground,n1).
        transistor(t2,n3,n4,n2).
        transistor(t3,n5,ground,n4).

inverter(I,Input,Output) ←
   I είναι ένας αντιστροφέας που αντιστρέφει το Input σε Output.
        inverter(inv(T,R),Input,Output) ←
              transistor(T,Input,ground,Output),
              resistor(R,power,Output).

nand_gate(Nand,Input1,Input2,Output) ←
   Nand είναι μια πύλη που σχηματίζει το λογικό nand, Output, των Input1 και Input2.
        nand_gate(nand(T1,T2,R),Input1,Input2,Output) ←
              transistor(T1,Input1,X,Output),
              transistor(T2,Input2,Ground,X),
              resistor(R,Power,Output).

and_gate(And,Input1,Input2,Output) ←
   And είναι μια πύλη που σχηματίζει το λογικό and, Output,
   των Input1 και Input2.
        and_gate(and(N,I),Input1,Input2,Output) ←
              nand_gate(N,Input1,Input2,X),
              inverter(I,X,Output).
```

Πρόγραμμα 9.3: Το κύκλωμα βάσης Δεδομένων με ονόματα

Η δόμηση των δεδομένων είναι σημαντική γενικά στον προγραμματισμό και ειδικά στον λογικό προγραμματισμό. Χρησιμοποιείται για την οργάνωση των δεδομένων κατά έναν ουσιαστικό τρόπο. Οι κανόνες μπορούν να γραφούν πιο αφηρημένα, αγνοώντας ασήμαντες λεπτομέρειες. Τα περισσότερα τμηματικά προγράμματα μπορούν να επιτευχθούν με αυτό το τρόπο, καθώς μια αλλαγή στην αναπαράσταση των δεδομένων δεν σημαίνει αλλαγή σε όλο το πρόγραμμα, όπως φαίνεται και από το παρακάτω παράδειγμα.

Εστω οι ακόλουθοι δύο τρόποι αναπαράστασης ενός γεγονότος σχετικά με μια διάλεξη πάνω στην πολυπλοκότητα που δόθηκε την Δευτέρα από τις 9 ως τις 11 από τον David Harel στο κτίριο Feinberg, αίθουσα A:

course(complexity,monday,9,11,david,harel,feinberg,a).
και
course(complexity,time(monday,9,11),lecturer(david,harel), location,feinberg,a)).

Το πρώτο γεγονός αναπαριστά το *course* ως μια σχέση μεταξύ οκτώ στοιχείων - ένα όνομα μαθήματος, μια μέρα, μια ώρα εκκίνησης, μια ώρα τερματισμού, το όνομα του Λέκτορα, το επώνυμο του Λέκτορα, ένα κτίριο και μια αίθουσα.

Το δεύτερο γεγονός θέτει το *course* ως σχέση τεσσάρων στοιχείων - ένα όνομα, έναν χρόνο, έναν λέκτορα και μια τοποθεσία με περισσότερους όρους. Ο χρόνος αποτελείται από μια μέρα, έναν χρόνο εκκίνησης και έναν χρόνο τερματισμού, οι Λέκτορες έχουν όνομα και επώνυμο, και οι τοποθεσίες προσδιορίζονται από ένα κτίριο και μια αίθουσα. Το δεύτερο γεγονός αντικατοπτρίζει πιο κομψά τις σχέσεις που υπάρχουν.

Η έκδοση του *course* με τέσσερα ορίσματα δίνει την δυνατότητα να γραφούν πιο συνοπτικοί κανόνες αφαιρώντας τις λεπτομέρειες που δεν έχουν σχέση με την ερώτηση. Το Πρόγραμμα 9.4 αποτελείται από μερικά παραδείγματα. Ο κανόνας *occupied* θεωρεί ένα κατηγόρημα μικρότερο ή ίσο, που αναπαρίσταται με έναν δυαδικό ένθετο τελεστή ≤.

Οι κανόνες που δεν σχετίζονται με τις συγκεκριμένες τιμές ενός δομημένου ορίσματος δεν χρειάζεται να "γνωρίζουν" πως είναι δομημένο το όρισμα. Για παράδειγμα, οι κανόνες για *duration* και *teaches* αναπαριστούν σαφώς τον χρόνο ως *time(Day,Start,Finish)*, διότι οι χρόνοι *Day* ή *Start* ή *Finish* του μαθήματος είναι επιθυμητοί. Σε αντίθεση, ο κανόνας για τον *lecturer* δεν είναι. Αυτό οδηγεί σε μεγαλύτερη τμηματικότητα, καθώς η αναπαράσταση του χρόνου μπορεί να αλλάξει χωρίς να επηρεάζει τους κανόνες που δεν τον ερευνούν.

```
lecturer(Lecturer,Course) ←
        course(Course,Time,Lecturer,Location).
duration(Course,Lenght) ←
        course(Course,time(Day,Start,Finish),Lecturer,Location)
        plus(Start,Lenght,Finish).
teaches(Lecturer,Day) ←
         course(Course,time(Day,Start,Finish),Lecturer,Location).
occupied(Room,Day,Time) ←
         course(Course,time(Day,Start,Finish),Lecturer,Room),
         Start ≤ Time, Time ≤ Finish.
```

Πρόγραμμα 9.4: Κανόνες εκμάθησης

Δεν έχουμε ορισμένους κανόνες για να αποφασίσουμε αν θα χρησιμοποιήσουμε δεδομένα ή όχι. Εκεί όπου όλα τα δεδομένα είναι απλά, η μη-χρησιμοποίηση δομημένων δεδομένων επιτρέπει την ομοιόμορφη αναπαράσταση. Τα πλεονεκτήματα των δομημένων δεδομένων είναι η συμπαγής αναπαράσταση, η οποία αντικατοπτρίζει με μεγαλύτερη ακρίβεια την άποψή μας για μια περίπτωση και η τμηματικότητα. Πιστεύουμε ότι η εμφάνιση ενός προγράμματος είναι σημαντική, ιδιαίτερα όταν αφορά δύσκολα προβλήματα. Μια καλή δόμηση των δεδομένων μπορεί να φέρει διαφοροποίηση όταν προγραμματίζουμε περίπλοκα προβλήματα.

Μερικοί από τους κανόνες στο Πρόγραμμα 9.4 ανασύρουν δυαδικές σχέσεις, δηλ. σχέσεις μεταξύ δύο οντοτήτων, από μια άλλη σχέση πιό περίπλοκη. Ολες οι πληροφορίες για το μάθημα θα μπορούσαν να έχουν γραφεί σαν δυαδικές σχέσεις όπως παρακάτω:

day(complexity,monday).

start_time(complexity,9).
finish_time(complexity,11).
lecturer(complexity,harel).
building(complexity,feinberg).
room(complexity,a).

Οι κανόνες τότε θα εκφράζονταν διαφορετικά, επανερχόμενοι στο προηγούμενο στυλ του να φτιάχνουμε τους έμμεσους κανόνες άμεσους. Για παράδειγμα:

teaches(Lecturer,Day) ← lecturer(Course,Lecturer),day(Course,Day).

9.4 Αναδρομικοί κανόνες

Οι κανόνες που περιγράφθηκαν ως τώρα ορίζουν νέες σχέσεις με βάση τις ήδη υπάρχουσες. Μια ενδιαφέρουσα επέκταση είναι οι αναδρομικοί ορισμοί των σχέσεων που ορίζουν σχέσεις με βάση τους εαυτούς τους. Κατά ένα τρόπο μπορούμε να δούμε τους αναδρομικούς κανόνες ως μια γενίκευση ενός συνόλου μη-αναδρομικών κανόνων.

Εστω μια σειρά κανόνων που ορίζουν τους προγόνους - παππούδες και γιαγιάδες, προπαππούδες και προγιαγιάδες, κ.λπ.:

grandparent(Ancestor,Descendant) ←
parent(Ancestor,Person),
parent(Person,Descendant).
grandparent(Ancestor,Descendant) ←
parent(Ancestor,Person),
grandparent(Person,Descendant).
greatgreatgrandparent(Ancestor,Descendant) ←
parent(Ancestor,Person),
greatgrandparent(Person,Descendant).

Μπορούμε να δούμε ένα σαφές υπόδειγμα, το οποίο μπορεί να εκφραστεί σε έναν κανόνα που ορίζει τη σχέση του *ancestor(Ancestor,Descendant)*:

ancestor(Ancestor,Descendant) ←
parent(Ancestor,Person),
ancestor(Person,Descendant).

Ο κανόνας είναι μια γενίκευση των παραπάνω κανόνων.

Ενα λογικό πρόγραμμα για το *ancestor* απαιτεί ακόμη έναν μη-αναδρομικό κανόνα, η επιλογή του οποίου επηρεάζει την έννοια του προγράμματος. Αν χρησιμοποιείται το γεγονός *ancestor(X,X)*, ορίζοντας τη σχέση του ancestor να είναι αυτοπαθής, οι άνθρωποι θα θεωρηθούν ως πρόγονοί τους. Αυτή δεν είναι η διαισθητική έννοια του προγόνου. Το Πρόγραμμα 9.5 είναι ένα λογικό πρόγραμμα που ορίζει τη σχέση προγόνου, όπου οι γονείς θεωρούνται πρόγονοι.

Η σχέση *ancestor* είναι το μεταβατικό κλείσιμο (transitive closure) της σχέσης *parent*. Γενικά, η εύρεση του μεταβατικού κλεισίματος μιας σχέσης βρίσκεται εύκολα σε ένα λογικό πρόγραμμα, χρησιμοποιώντας έναν αναδρομικό κανόνα.

Εστω το πρόβλημα δοκιμής της συνδετικότητας σε έναν προσανατολισμένο γράφο. Ενας προσανατολισμένος γράφος μπορεί να αναπαρασταθεί σαν ένα λογικό πρόγραμμα με μια συλλογή γεγονότων. Ενα γεγονός edge(Node1,Node2) υπάρχει στο πρόγραμμα αν υπάρχει μια ακμή από τον Node1 στον Node2 του γράφου. Το Σχήμα 9.4 δίνει έναν γράφο, ενώ το Πρόγραμμα 9.6 είναι η περιγραφή του με ένα λογικό πρόγραμμα.

Δύο κόμβοι συνδέονται αν υπάρχει μια σειρά ακμών που μπορεί να διασχιστεί από τον πρώτο κόμβο ως τον δεύτερο. Δηλαδή, η σχέση *connected(Node1,Node2),* η οποία είναι αληθής αν οι *Node1* και *Node2* συνδέονται. Για παράδειγμα, οι *a* και *e* συνδέονται σε έναν γράφο του Σχήματος 9.4, αλλά οι *b* και *f* δεν συνδέονται. Το Πρόγραμμα 9.7 ορίζει την σχέση. Η έννοια του προγράμματος είναι το σύνολο των στόχων *connected(X,Y)*, όπου *X* και *Y* συνδέονται. Παρατηρήστε ότι το *connected* είναι μια μεταβατική αυτοπαθής σχέση, εξαιτίας της επιλογής του βασικού γεγονότος.

```
ancestor(Ancestor,Descendant) ←
        Ancestor είναι ένας πρόγονος του Descendant.

ancestor(Ancestor,Descendant) ←
        parent(Ancestor,Descendant).
ancestor(Ancestor,Descendant) ←
        parent(Ancestor,Person), ancestor(PersonDescendant).
```

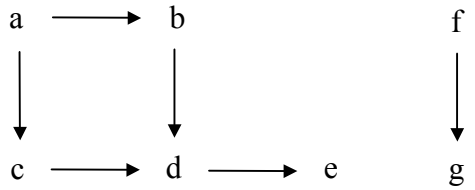
Πρόγραμμα 9.5: Η σχέση προγόνου

```
edge(a,b).    edge(a,c).    edge(b,d).
edge(c,d).    edge(d,e).    edge(f,g).
```

Πρόγραμμα 9.6: Ενας προσανατολισμένος γράφος

```
connected(Node,Node) ←
        O Node1 συνδέεται με τον Node2 στον
        γράφο που ορίζεται από την σχέση  edge/2.

Connected(Node,Node).
Connected(Node1,Node2) ← edge(Node1,Link), connected(Link,Node2).
```

Πρόγραμμα 9.7: Το μεταβατικό κλείσιμο της σχέσης edge.



Εικόνα 9.4: Ενας απλός γράφος

9.5 Λογικά προγράμματα και το μοντέλο σχεσιακών βάσεων δεδομένων

Τα λογικά προγράμματα μπορούν να θεωρηθούν ως μια δυναμική επέκταση στο μοντέλο σχεσιακών βάσεων δεδομένων, με την δυναμικότητα αυτή να πηγάζει από την ικανότητα να καθορίζουν κανόνες. Πολλές από τις θεωρίες που έχουν προταθεί έχουν ουσιαστικές αναλογίες με τους όρους των βάσεων δεδομένων. Το αντίστροφο είναι επίσης αληθές. Οι βασικές λειτουργίες της σχεσιακής άλγεβρας εύκολα εκφράζονται μέσα στον λογικό προγραμματισμό. Πέντε βασικές λειτουργίες ορίζουν την σχεσιακή άλγεβρα: ένωση, διαφορά συνόλου, Καρτεσιανό γινόμενο, προβολή και επιλογή. Σας δείχνουμε πως μεταφράζεται η καθεμιά σε λογικό πρόγραμμα.

Η ένωση δημιουργεί μια σχέση πολλαπλότητας *n* από δύο σχέσεις *r* και *s*, και οι δυο πολλαπλότητας *n*. Η νέα σχέση, που δηλώνεται εδώ ως *r_union_s*, είναι η ένωση των *r* και *s*. Ορίζεται άμεσα σαν λογικό πρόγραμμα από δύο κανόνες:

$r\_union\_s(X_1,...,X_n) \leftarrow r(X_1,...,X_n).$
$r\_union\_s(X_1,...,X_n) \leftarrow s(X_1,...,X_n).$

Η διαφορά συνόλου περιλαμβάνει και άρνηση. Διαισθητικά, ένας στόχος *not G* αληθεύει σύμφωνα με ένα πρόγραμμα *P*, αν το *G* δεν είναι λογική συνέπεια του *P*. Η άρνηση στα λογικά προγράμματα δεν μελετάται εδώ, όπου εμφανίζονται περιορισμοί στον διαισθητικό ορισμό. Ο ορισμός ωστόσο είναι σωστός, αν έχουμε να κάνουμε μόνο με βασικά γεγονότα, όπως στην περίπτωση με σχεσιακές βάσεις δεδομένων.

Ο ορισμός του *r_diff_s* πολλαπλότητας *n*, όπου *r* και *s* έχουν πολλαπλότητα *n* είναι :

$r\_diff\_s(X_1,...,X_n) \leftarrow r(X_1,...,X_n),\ not\ s(X_1,...,X_n).$
$r\_diff\_s(X_1,...,X_n) \leftarrow s(X_1,...,X_n),\ not\ r(X_1,...,X_n).$

Το Καρτεσιανό γινόμενο μπορεί να οριστεί με έναν μόνο κανόνα. Αν το *r* είναι μια σχέση πολλαπλότητας *m*, και *s* είναι μια σχέση πολλαπλότητας *n*, τότε *r_x_s* είναι μια σχέση πολλαπλότητας m+n που ορίζεται από

$$r\_x\_s(X_1,...,X_m,X_{m+1},...,X_{m+n}) \leftarrow r(X_1,...,X_m),\ s(X_{m+1},...,X_{m+n}).$$

Η προβολή περιλαμβάνει τον σχηματισμό μιας νέας σχέσης που αποτελείται μόνο από μερικά από τα ορίσματα μιας ήδη υπάρχουσας σχέσης. Αυτό είναι σαφές για οποιαδήποτε σχέση. Για παράδειγμα, η προβολή r3 επιλέγοντας το πρώτο και το τρίτο όρισμα μιας σχέσης πολλαπλότητας 3 είναι

$$r13(X_1,X_3) \leftarrow r(X_1,X_2,X_3).$$

Η επιλογή είναι το ίδιο σαφής για κάθε περίπτωση. Εστω μια σχέση που αποτελείται από πλειάδες των οποίων το τρίτο μέρος είναι μεγαλύτερο από το δεύτερο, και μια σχέση όπου το πρώτο μέρος είναι *Smith* ή *Jones*. Και στις δύο περιπτώσεις μια σχέση *r* πολλαπλότητας 3 χρησιμοποιείται για να το υποδείξει. Το πρώτο παράδειγμα δημιουργεί μαι σχέση *r1*:

$$r1(X_1,X_2,X_3) \leftarrow r(X_1,X_2,X_3),\ X_2 > X_3.$$

Το δεύτερο παράδειγμα δημιουργεί μια σχέση r2, που απαιτεί μια διαζευκτική σχέση, smith_or_jones:

$r2(X_1,X_2,X_3) \leftarrow r(X_1,X_2,X_3), smith\_or\_jones(X_1).$
$smith\_or\_jones(smith).$
$smith\_or\_jones(jones).$

Μερικές από τις παραγόμενες λειτουργίες της σχεσιακής άλγεβρας συνδέονται πιο στενά με τις δομές του λογικού προγραμματισμού. Αναφέρουμε δύο, τομή και φυσική σύνδεση (natural join). Αν *r* και *s* είναι σχέσεις πολλαπλότητας *n*, η τομή r_meet_s είναι επίσης πολλαπλότητας n και ορίζεται με έναν μόνο κανόνα.

$$r\_meet\_s(X_1,...,X_n) \leftarrow r(X_1,...,X_n), s(X_1,...,X_n).$$

Μια φυσική σύνδεση είναι ακριβώς μια συνδετική ερώτηση με διαμοιραζόμενες μεταβλητές.

10

Αναδρομικός Προγραμματισμός

10.1 Εισαγωγή

Τα προγράμματα του προηγούμενου κεφαλαίου επεξεργάζονται και ανακτούν πληροφορίες από περιορισμένο αριθμό δομών δεδομένων. Γενικά, αποκτούμε μαθηματική δύναμη θεωρώντας άπειρες ή πιθανά άπειρες δομές. Πάρα πολλά στιγμιότυπα δομών ακολουθούν τις πιθανά άπειρες δομές ως ειδικές περιπτώσεις. Τα λογικά προγράμματα μεγαλώνουν λοιπόν αυτή τη δύναμη χρησιμοποιώντας αναδρομικούς τύπους δεδομένων.

Οι λογικοί όροι μπορούν να διακριθούν σε τύπους. Ενας *τύπος* είναι ένα (πιθανώς άπειρο) σύνολο όρων. Μερικοί τύποι ορίζονται μόνο από μοναδιαίες σχέσεις. Μια σχέση *p/1* ορίζει τον τύπο *p* να είναι το σύνολο των *X* τέτοιων ώστε *p(X)*.

Για παράδειγμα, τα κατηγορήματα *male/1* και *female/1* που χρησιμοποιήθηκαν προηγουμένως ορίζουν τους τύπους *male* και *female* αντίστοιχα.

Πιο πολύπλοκοι τύποι μπορούν να οριστούν από αναδρομικά λογικά προγράμματα. Τέτοιοι τύποι ονομάζονται *αναδρομικοί τύποι*. Οι τύποι που ορίζονται από μοναδιαία αναδρομικά προγράμματα καλούνται *απλοί αναδρομικοί τύποι*. Ενα πρόγραμμα που ορίζει έναν τύπο ονομάζεται *ορισμός τύπου*.

Σε αυτό το κεφάλαιο, παρουσιάζουμε λογικά προγράμματα που ορίζουν σχέσεις σε απλούς αναδρομικούς τύπους, όπως οι ακέραιοι, λίστες και δυαδικά δέντρα, και επίσης προγράμματα σε πιο πολύπλοκους τύπους, όπως τα πολυώνυμα.

10.2 Αριθμητική

Ο πιο απλός αναδρομικός τύπος, οι φυσικοί αριθμοί, προέρχεται από τις βάσεις των Μαθηματικών. Η αριθμητική βασίζεται στους φυσικούς αριθμούς. Αυτή η ενότητα δίνει λογικά προγράμματα για την εφαρμογή της αριθμητικής.

Στην ουσία, τα προγράμματα Prolog για την εφαρμογή της αριθμητικής διαφέρουν σημαντικά από τα λογικά τους αντίστοιχα, όπως θα δούμε σε επόμενα κεφάλαια. Παρόλα αυτά, η μελέτη των λογικών προγραμμάτων είναι χρήσιμη. Υπάρχουν δύο λόγοι. Κατά πρώτον, οι πράξεις της αριθμητικής μελετώνται συνήθως λειτουργικά παρά σχεσιακά. Η παρουσίαση παραδειγμάτων για μια τόσο γνωστή περιοχή τονίζει την ανάγκη της αλλαγής στην σκέψη για την σύνθεση λογικών προγραμμάτων. Κατά δεύτερον, είναι πιο φυσικό να μελετάμε τα μαθηματικά ζητήματα που βρίσκονται από κάτω, όπως η ορθότητα και η πληρότητα των προγραμμάτων.

natural_number(*X*) ←

Χ είναι ένας φυσικός αριθμός.

natural_number(0).
natural_number(s(X)) ← natural_number(X).

Πρόγραμμα 10.1: Ορίζοντας τους φυσικούς αριθμούς

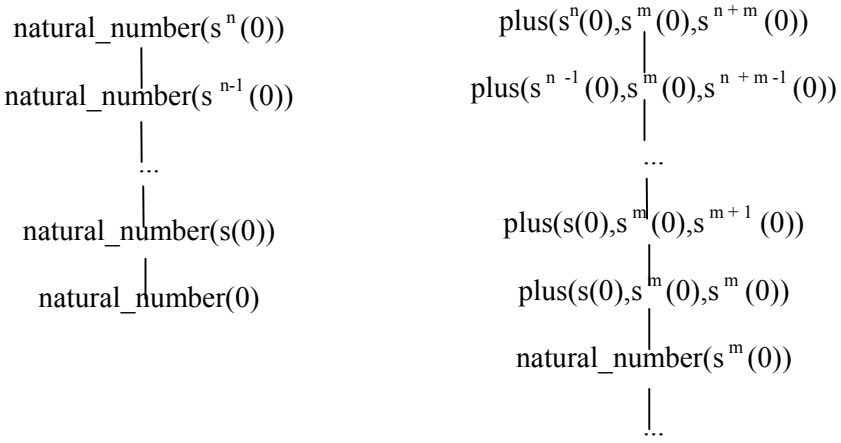
Οι φυσικοί αριθμοί κατασκευάζονται από δύο δομές, το σταθερό σύμβολο *0* και την συνάρτηση διαδοχής *s* με arity *1*. Κατόπιν, όλοι οι φυσικοί αριθμοί δίνονται αναδρομικά ως *0*, *s(0)*, *s(s(0))*, *s(s(s(0)))*,.... Συμφωνούμε ότι το $s^n(0)$ δηλώνει τον ακέραιο *n*, δηλαδή, τις *n* εφαρμογές της συνάρτησης διαδοχής στο *0*.

Οπως στο προηγούμενο κεφάλαιο, για κάθε κατηγόρημα δίνουμε ένα σχήμα σχέσης, μαζί με την προτιθέμενη ερμηνεία του κατηγορήματος. Θυμηθείτε ότι ένα πρόγραμμα *P* είναι *ορθό* σε σχέση με μια προτιθέμενη ερμηνεία *M*, αν η ερμηνεία του *P* είναι υποσύνολο του *M*. Είναι *πλήρες* αν το *M* είναι υποσύνολο της ερμηνείας του *P*. Είναι ορθό και πλήρες αν η ερμηνεία του ταυτίζεται με το M. Η απόδειξη της ορθότητας δηλώνει πως οτιδήποτε εξάγεται από το πρόγραμμα είναι το επιθυμητό. Η απόδειξη της πληρότητας δηλώνει πως οτιδήποτε επιθυμητό εξάγεται από το πρόγραμμα. Δύο συνήθεις αποδείξεις πληρότητας και ορθότητας δίνονται σε αυτή την ενότητα.

Ο απλός τύπος ορισμού των φυσικών αριθμών υλοποιείται με ωραίο τρόπο στο λογικό πρόγραμμα, Πρόγραμμα 10.1. Το σχήμα σχέσης που χρησιμοποιείται είναι *natural_number(X)*, με επιθυμητή ερμηνεία ότι ο *X* είναι φυσικός αριθμός. Το πρόγραμμα αποτελείται από μια μοναδιαία πρόταση και μια επαναληπτική πρόταση (μια πρόταση με ένα μοναδικό στόχο στο σώμα). Ενα τέτοιο πρόγραμμα καλείται *ελάχιστο επαναληπτικό*.

Πρόταση: Το Πρόγραμμα 10.1 είναι ορθό και πλήρες σε σχέση με το σύνολο των στόχων $natural\_number(s^i(0))$, για $i \geq 0$.

Απόδειξη:
(1) Ορθότητα. Εστω *N* ένας φυσικός αιθμός. Θα δείξουμε ότι ο στόχος *natural_number(N)*, μπορεί να εξαχθεί από το πρόγραμμα δίνοντας ένα σαφές δέντρο απόδειξης. Είτε το *N* είναι *0* είτε του τύπου $s^N(0)$. Το δέντρο απόδειξης για τον στόχο *natural_number(0)*, είναι ασήμαντο και εύκολο. Το δέντρο απόδειξης για τον στόχο *natural_number(s(...s(0)...))*, περιλαμβάνει *N* απλοποιήσεις (υποβιβασμούς), χρησιμοποιώντας τον κανόνα του Προγράμματος 10.1, για να φτάσουμε στο γεγονός *natural_number(0)*, όπως φαίνεται στο αριστερό μισό της Εικόνας 10.1.
(2) Πληρότητα. Εστω ότι το *natural_number(X)* μπορεί να εξαχθεί από το Πρόγραμμα 10.1, με *n* επαγωγές. Αποδεικνύουμε ότι το *natural_number(X)* είναι η επιθυμητή ερμηνεία του προγράμματος με επαγωγή στο *n*. Αν *n=0*, τότε πρέπει να έχει αποδειχθεί χρησιμοποιώντας μια μοναδιαία προταση, που υπονοεί ότι το *X=0*. Αν *n>0*, τότε ο στόχος πρέπει να είναι του τύπου $natural\_number(s(X_1))$, αφού μπορεί να εξαχθεί από το πρόγραμμα, και επιπλέον το $natural\_number(X_1)$ μπορεί να εξαχθεί από *n-1* επαγωγές. Από την επαγωγική υπόθεση, το X_1 είναι η σκοπούμενη ερμηνεία του προγράμματος, λ.χ., $X_1=s^k(0)$ για κάποια $k \geq 0$.



Εικόνα 10.1: Δέντρα απόδειξης που δίνουν την πληρότητα των προγραμμάτων

Οι φυσικοί αριθμοί έχουν φυσική διάταξη. Το Πρόγραμμα 9.2. είναι ένα λογικό πρόγραμμα που ορίζει την σχέση μικρότερο από ή ίσο σύμφωνα με την διάταξη αυτή. Δηλώνουμε την σχέση με ένα δυαδικό ένθετο σύμβολο, έναν *τελεστή*, ≤ ακολουθώντας την μαθηματική του χρήση. Η έκφραση $0 \leq X$ είναι ένας όρος με παράγοντα *≤/2*, και ορίσματα *0* και *X*, και είναι συντακτικά ισοδύναμη με το *'≤'(0,X)*.

Το σχήμα σχέσης είναι $N_1 \leq N_2$. Η σκοπούμενη ερμηνεία του Προγράμματος 10.2 είναι όλα τα βασικά γεγονότα $X \leq Y$, όπου *X* και *Y* φυσικοί αριθμοί και το *X* εναι μικρότερο ή ίσο του *Y*.

Ο αναδρομικός ορισμός του ≤ δεν είναι "υπολογιστικά εφικτός". Το δέντρο απόδειξης που δείχνει ότι ένας συγκεκριμένος *N* είναι μικρότερος ή ίσος από έναν συγκεκριμένο *M* έχει *M+2* κόμβους. Συνήθως εξετάζουμε αν ένας αριθμός είναι μικρότερος από έναν άλλο σαν μια μοναδιαία πράξη, ανεξάρτητη του μεγέθους των αριθμών. Πράγματι η Prolog δεν ορίζει την αριθμητική σύμφωνα με αξιώματα που παρουσιάζονται σ' αυτή την ενότητα, αλλά χρησιμοποιεί άμεσα τις υπάρχουσες αριθμητικές δυνατότητες του υπολογιστή.

$X \leq Y \leftarrow$
X και Y είναι φυσικοί αριθμοί, έτσι ώστε ο X να είναι μικρότερος ή ίσος του Y.

$0 \leq X \leftarrow$ natural_number(X).
$s(X) \leq s(Y) \leftarrow X \leq Y$.

natural_number(X) ← βλέπε Πρόγραμμα 10.1

Πρόγραμμα 10.2: Η σχέση μικρότερο ή ίσο

Η πρόσθεση είναι μια βασική πράξη που ορίζει μια σχέση μεταξύ δύο φυσικών αριθμών και το άθροισμά τους. Στην Ενότητα 8.1 ένας πίνακας της σχέσης plus

δίνεται για όλους τους σημαντικούς φυσικούς αριθμούς. Ενα αναδρομικό πρόγραμμα εκφράζει τη σχέση κομψά και συνοπτικά, και δίνεται σαν Πρόγραμμα 10.3. Η προτιθέμενη ερμηνεία του Προγράμματος 10.3 είναι το σύνολο των γεγονότων *plus(X,Y,Z)* όπου *X*, *Y*, και *Z* είναι φυσικοί αριθμοί και *X* + *Y*=*Z*.

plus(*X,Y,Z*) ←
X, Y και Z φυσικοί αριθμοί, έτσι ώστε Z να είναι το άθροισμα των X και Y.

plus(0,X,X) ← natural_number(X).
plus(*s(X)*,Y,s(Z)) ← plus(X,Y,Z).
natural_number(X) ← βλέπε Πρόγραμμα 10.1

Πρόγραμμα 10.3: Πρόσθεση

Πρόταση: Τα Προγράμματα 10.1 και 10.3 αποτελούν ένα ορθό και πλήρες αξίωμα της πρόσθεσης, σύμφωνα με την καθιερωμένη προτιθέμενη ερμηνεία του *plus/3*.

Απόδειξη: (1) Πληρότητα. Εστω *X*, *Y*, και *Z* φυσικοί αριθμοί τέτοιοι ώστε *X* + *Y*=*Z*. Δίνουμε ένα δέντρο απόδειξης για τον στόχο *plus(X,Y,Z)*. Αν το *X* ισούται με *0*, τότε το *Y* ισούται με το *Z*. Εφόσον το Πρόγραμμα 10.1 είναι ένα πλήρες αξίωμα των φυσικών αριθμών, υπάρχει ένα δέντρο απόδειξης για το *natural_number(Y)*, το οποίο εύκολα επεκτείνεται σε ένα δέντρο απόδειξης για το *plus(0,Y,Y)*. Αλλιώς, το *X* ισούται με $s^n(0)$ για κάποιο *n*. Αν το *Y* ισούται με $s^m(0)$, τότε το *Z* ισούται με $s^{n+m}(0)$. Το δέντρο απόδειξης στο δεξί μισό της Εικόνας 10.1 δείχνει την πληρότητα.

(2) Ορθότητα. Εστω ότι το *plus(X,Y,Z)* ανήκει στην ερμηνεία. Ενα απλό επαγωγικό όρισμα στο μέγεθος του *X*, παρόμοιο με αυτό που χρησιμοποιήθηκε στην προηγούμενη πρόταση, δείχνει ότι *X* + *Y*=*Z*. ■

Η πρόσθεση συνήθως θεωρείται ως συνάρτηση δύο ορισμάτων και όχι μια τριμελής σχέση. Γενικά, τα λογικά προγράμματα που αντιστοιχούν σε συναρτήσεις *n* ορισμάτων ορίζουν *n + 1* μέλη σχέσεων. Ο υπολογισμός της τιμής μιας συνάρτησης επιτυγχάνεται θέτωντας μια ερώτηση με n συγκεκριμένα ορίσματα και την θέση του ορίσματος να αντιστοιχεί στην τιμή της συνάρτησης με τα αόριστα ορίσματα. Η λύση στο ερώτημα είναι η τιμή της συνάρτησης με τα δοσμένα ορίσματα. Για να κάνουμε πιο σαφή την αναλογία, δίνουμε έναν συναρτησιακό ορισμό της πρόσθεσης που αντιστοιχεί στο λογικό πρόγραμμα.

0+X = X.
S(X)+Y = s(X+Y).

Ενα πλεονέκτημα που έχουν τα σχεσιακά προγράμματα ως προς τα συναρτησιακά προγράμματα είναι η δυνατότητα να έχει το πρόγραμμα πολλαπλές χρήσεις. Για παράδειγμα, το ερώτημα *plus(s(0),s(0),s(s(0)))?* σημαίνει έλεγχος εάν *1+1=2*. (Προσπαθούμε να χρησιμοποιούμε τον πιο ευανάγνωστο αριθμητικό συμβολισμό όταν αναφέρουμε αριθμούς.) Οσον αφορά το ≤, το πρόγραμμα για το *plus* δεν είναι αποτελεσματικό. Το δέντρο απόδειξης που επαληθεύει ότι το άθροισμα του *N* με το *M* είναι *N+M* έχει *N+M+2* κόμβους.

Θέτωντας το ερώτημα *plus(s(0),s(0),X)?*, ένα παράδειγμα της καθιερωμένης χρήσης, υπολογίζει το άθροισμα του *1* με το *1*. Παρόλα αυτά, το πρόγραμμα μπορεί να χρησιμοποιηθεί εξίσου εύκολα για την αφαίρεση θέτωντας ένα ερώτημα της

μορφής *plus(s(0),X,s(s(s(0))))?* Η τιμή του *X* που υπολογίζεται είναι η διαφορά μεταξύ του *3* και του *1*, δηλαδή, το *2*. Παρόμοια, κάνοντας μια ερώτηση με το πρώτο όρισμα μη-συγκεκριμένο, και το δεύτερο και τρίτο συγκεκριμένο, επίσης γίνεται η αφαίρεση.

Μια νεώτερη χρήση εκμεταλλεύεται την πιθανότητα μιας ερώτησης να έχει *πολλαπλές λύσεις*. Η ερώτηση *plus(X,Y,σ(s(s(0))))?* διαβάζεται ως εξής: "Υπάρχουν αριθμοί Χ και Υ που δίνουν άθροισμα 3." Με άλλα λόγια, βρες μια διαμέριση του αριθμού *3* μέσα στο άθροισμα των δύο αριθμών, *X* και *Y*. Υπάρχουν διάφορες λύσεις.

Μια ερώτηση με πολλαπλές λύσεις γίνεται πιο ενδιαφέρουσα όταν οι ιδιότητες των μεταβλητών της ερώτησης είναι περιορισμένες. Υπάρχουν δύο τύποι περιορισμού: χρησιμοποιώντας περισσότερους συνδέσμους στην ερώτηση, και συγκεκριμενοποιώντας μεταβλητές στην ερώτηση. Είδαμε τέτοια παραδείγματα όταν θέταμε ερωτήσεις σε μια βάση δεδομένων. Η Ασκηση (ii) στο τέλος της ενότητας ζητά να ορίσετε το κατηγόρημα *even(X)*, το οποίο είναι αληθές αν το *X* είναι άρτιος αριθμός. Θεωρώντας ένα τέτοιο κατηγόρημα, η ερώτηση *plus(X,Y,N),even(X),even(Y)?* δίνει μια διαμέριση του *N* σε δύο άρτιους αριθμούς. Ο δεύτερος τύπος περιορισμού δηλώνεται με την ερώτηση *plus(s(s(X)),s(s(Y)),N)?* που επιμένει ότι καθένας από τους αριθμούς που δίνει άθροισμα το *N* είναι γνήσια μεγαλύτερο του ένα.

Σχεδόν όλα τα λογικά προγράμματα έχουν πολλαπλές χρήσεις. Εστω το Πρόγραμμα 10.2 για το ≤, για παράδειγμα. Η ερώτηση *s(0)≤ s(s(0))?* ελέγχει αν το *1* είναι μικρότερο ή ίσο του *2*. Η ερώτηση *X ≤ s(s(0))?* βρίσκει αριθμούς *X* μικρότερους ή ίσους του *2*. Ακόμη, υπολογίζει ζευγάρια αριθμών μικρότερα ή ίσα του ενός από το άλλο με την ερώτηση *X ≤ Y?*

Το Πρόγραμμα 10.3 που ορίζει την πρόσθεση δεν είναι μοναδικό. Για παράδειγμα, το λογικό πρόγραμμα

```
plus(X,0,X) ← natural_number(X).
plus(X,s(Y),s(Z)) ← plus(X,Y,Z).
```

έχει ακριβώς την ίδια ερμηνεία με το Πρόγραμμα 10.3 για το *plus*. Δύο προγράμματα αναμένονται εξαιτίας της συμμετρικότητας μεταξύ των δύο πρώτων ορισμάτων. Μια απόδειξη ορθότητας και πληρότητας που δίνεται για το Πρόγραμμα 10.3 εφαρμόζεται σ' αυτό το πρόγραμμα αντιστρέφοντας τους ρόλους των συμμετρικών ορισμάτων.

Η ερμηνεία του προγράμματος για το plus δεν θα άλλαζε ακόμη και αν αποτελούνταν από των συνδυασμό των δύο προγραμμάτων. Παραταύτα, το σύνθετο πρόγραμμα δεν είναι επιθυμητό. Υπάρχουν πολλά διαφορετικά δέντρα απόδειξης για τον ίδιο στόχο. Είναι σημαντικό, τόσο για λόγους απόδοσης κατά τον χρόνο εκτέλεσης όσο και για λόγους συνοπτικότητας του κειμένου, τα αξιώματα των λογικών προγραμμάτων να είναι ελάχιστα.

Ορίζουμε ως *συνθήκη τύπου* να είναι η κλήση σε ένα κατηγόρημα που ορίζει τον τύπο. Για φυσικούς αριθμούς, μια συνθήκη τύπου είναι οποιοσδήποτε στόχος του τύπου *natural_number(X)*.

Στην πράξη, και το Πρόγραμμα 10.2 και το 10.3 απλοποιούνται παραλείποντας το σώμα του βασικού κανόνα, *natural_number(X)*. Χωρίς αυτή τη δοκιμή, τα γεγονότα όπως *0 ≤ α* και *plus(0,α,α)*, όπου α μια τυχαία σταθερά, θα ανήκει στις ερμηνείες του προγράμματος. Οι συνθήκες τύπου είναι αναγκαίες για ορθά προγράμματα. Παρόλα αυτά, οι συνθήκες τύπου ξεφεύγουν από την απλότητα των προγραμμάτων, και επηρεάζουν το μέγεθος των δέντρων απόδειξης. Επομένως,

στα επόμενα ίσως παραληφθούν ορισμένες συνθήκες τύπου από τα παραδείγματα προγραμμάτων.

Τα βασικά προγράμματα που παρουσιάστηκαν είναι οι βασικοί λίθοι για την κατασκευή πιο περίπλοκων σχέσεων. Ενα τυπικό παράδειγμα είναι ο ορισμός του πολλαπλασιασμού ως επαναληπτική πρόσθεση. Το Πρόγραμμα 10.4 αντικατοπτρίζει αυτή τη σχέση. Το σχήμα σχέσης είναι *times(X,Y,Z)* που σημαίνει ότι, *X* φορές το *Y* ισούται με *Z*.

Η εκθετικότητα ορίζεται ως επαναληπτικός πολλαπλασιασμός. Το Πρόγραμμα 10.5 για το *exp(N,X,Y)* εκφράζει τη σχέση ότι $X^N = Y$. Είναι ανάλογο του Προγράμματος 10.4 για *times(X,Y,Z)*, με τα *exp* και *times* να αντικαθιστούν τα *times* και *plus*, αντίστοιχα. Οι βασικές περιπτώσεις για ύψωση σε δύναμη είναι $X^0 = 1$ για όλες τις θετικές τιμές του *X*, και $0^N = 0$ για θετικές τιμές του *N*.

Ενας ορισμός της παραγοντικής συνάρτησης για φυσικούς αριθμούς χρησιμοποιεί τον ορισμό του πολλαπλασιασμού. Θυμηθείτε ότι N!=N · (N-1) · ... · 2 · 1. Το κατηγόρημα *factorial(N,F)* συσχετίζει έναν αριθμό *N* με το παραγοντικό του *F*. Το Πρόγραμμα 10.6 είναι το αξίωμά του.

Δεν ορίζονται αναδρομικά όλες οι σχέσεις που αφορούν τους φυσικούς αριθμούς. Οι σχέσεις μπορούν να οριστούν, επίσης, με το στυλ των προγραμμάτων προηγουμένου κεφαλαίου. Ενα παράδειγμα είναι και το Πρόγραμμα 10.7 που βρίσκει το ελάχιστο δύο αριθμών με τη σχέση *minimum(N1,N2,Min)*.

times(X,Y,Z) ← X, Y και Z φυσικοί αριθμοί, έτσι ώστε Z να είναι το γινόμενο των X και Y.

```
times(0,X,0).
times(s(X),Y,Z) ← times(X,Y,W), plus(W,Y,Z).
plus(X,Y,Z) ←Βλέπε Πρόγραμμα 10.3.
```

Πρόγραμμα 10.4: Ο πολλαπλασιασμός ως επαναληπτική πρόσθεση

exp(N,X,Y) ← N, X, και Y φυσικοί αριθμοί, έτσι ώστε Y ισούται με το X υψωμένο στο N.

```
exp(s(X),0,0).
exp(0,s(X),s(0)).
exp(s(N),X,Y) ← exp(N,X,Z), times(Z,X,Y).
times(X,Y,Z) ← Βλέπε Πρόγραμμα 10.4
```

Πρόγραμμα 10.5: Η εκθετικότητα ως επαναληπτικός πολλαπλασιασμός

factorial(N,F) ← Το F ισούται με το N παραγοντικό.

```
factorial(_,s(0)).
factorial(s(N),F) ← factorial(N,F1), times(s(N),F1,F).
Times(X,Y,Z) ← Βλέπε Πρόγραμμα 10.4
```

Πρόγραμμα 10.6: Υπολογισμός Παραγοντικών

minimum(N_1, N_2, *Min*) ← Το ελάχιστο των φυσικών αριθμών *N1* και *N2* είναι το *Min*.

minimum(N_1, N_2, N_1) ← $N_1 \leq N_2$.
minimum(N_1, N_2, N_2) ← $N_2 \leq N_1$.

Πρόγραμμα 10.7: Το ελάχιστο δύο αριθμών

mod(X,Y,Z) ← όπου Z είναι το υπόλοιπο της ακέραιας διαίρεσης του X δια του Y.

mod(X,Y,Z) ← Z < Y, times(Y,Q,W), plus(W,Z,X).

Πρόγραμμα 10.8α: Ενας μη αναδρομικός ορισμός του modulus

mod(X,Y,Z) ← όπου Z είναι το υπόλοιπο της ακέραιας διαίρεσης του X δια του Y.

mod(X,Y,X) ← X < Y.
mod(X,Y,Z) ← plus(X1,Y,X), mod(X1,Y,z).

Πρόγραμμα 10.8β: Ενας αναδρομικός ορισμός του modulus

Η σύνθεση ενός προγράμματος για τον καθορισμό του υπολοίπου της ακέραιας διαίρεσης αποκαλύπτει ένα πολύ ενδιαφέρον φαινόμενο - διαφορετικοί μαθηματικοί ορισμοί της ίδιας σύληψης μεταφράζονται σε διαφορετικά λογικά προγράμματα. Τα προγράμματα 10.8α και 10.8β δίνουν δύο διαφορετικούς ορισμούς της σχέσης mod(X,Y,Z), η οποία είναι αληθής εάν το Z είναι η τιμή του X modulo Y, ή με άλλα λόγια το Z είναι το υπόλοιπο του X δια του Y. Τα προγράμματα χρησιμοποιούν τη σχέση < όπως καθορίζεται στη άσκηση (i) στο τέλος του τμήματος.

Το πρόγραμμα 10.8α παρουσιάζει τον άμεσο μετασχηματισμό ενός μαθηματικού ορισμού ο οποίος είναι μία λογική πρόταση, σε ένα λογικό πρόγραμμα. Το πρόγραμμα ισοδυναμεί με ένα υπαρκτό ορισμό του ακεραίου υπολοίπου: " Το Z είναι η τιμή του X mod Y εάν το Z είναι αυστηρώς μικρότερο του Y και υπάρχει ένας αριθμός Q τέτοιος ώστε X = Q.Y+Z. Γενικώς, οι μαθηματικοί ορισμοί μεταφράζονται εύκολα σε λογικά προγράμματα.

Μπορούμε να σχετίσουμε το πρόγραμμα 10.8α με κατασκευαστικά μαθηματικά. Αν και φαίνεται ένας υπαρκτός ορισμός, είναι επίσης κατασκευαστικός, εξ'αιτίας της κατασκευαστικής φύσης των <, plus και times. Ο αριθμός Q, για παράδειγμα,που υπάρχει στον ορισμό θα υπολογιστεί ρητώς από το times σε κάθε χρήση του mod.

Σε αντίθεση με το πρόγραμμα 10.8α, το πρόγραμμα 10.8β ορίζεται αναδρομικά. Αποτελεί έναν αλγόριθμο για την εύρεση του ακεραίου υπολοίπου, βασισμένο στην επαναλαμβανόμενη αφαίρεση. Ο πρώτος κανόνας λέει ότι το X mod Y είναι X εάν το X είναι αυστηρώς μικρότερο του Y. Ο δεύτερος κανόνας λέει ότι το X mod Y είναι το ίδιο με το X-Y mod Y. Το αποτέλεσμα κάθε υπολογισμού για τον καθορισμό του modulus είναι η συνεχής αφαίρεση του Y από το X εώς ότου το X γίνει μικρότερο του Y, οπότε είναι και η σωστή τιμή.

Η μαθηματική συνάρτηση X mod Y δεν ορίζεται όταν το Y είναι μηδέν. Ούτε το πρόγραμμα 10.8α ούτε το πρόγραμμα 10.8β έχουν στόχους mod(X,0,Y) στην ερμηνεία τους για καμμία τιμή X ή Y. Ο έλεγχος "<" εγγυάται κάτι τέτοιο.

Το υπολογιστικό μοντέλο δίνει ένα τρόπο διάκρισης μεταξύ των δύο προγραμμάτων για το mod. Δεδομένων συγκεκριμένων X, Y και Z που ικανοποιούν το mod, μπορούμε να συγκρίνουμε το μέγεθος των δένδρων απόδειξής τους. Γενικά, τα δένδρα απόδειξης που παράγονται από το πρόγραμμα 10.8β θα είναι μικρότερα από τα δένδρα απόδειξης που παράγονται από το πρόγραμμα 10.8α. Με αυτή την έννοια το πρόγραμμα 10.8β είναι πιο "αποδοτικό".

Αφήνουμε τις περισσότερο αυστηρές συζητήσεις για την αποδοτικότητα μέχρις ότου μιλήσουμε για τις λίστες, όπου οι γνώσεις που θα έχουμε αποκτήσει μέχρι το σημείο εκείνο θα χρησιμοποιηθούν σε προγράμματα Prolog.

ackermann(X,Y,A) ← A είναι η τιμή της συνάρτησης Ackermann για τους φυσικούς αριθμούς X και Y.

```
ackermann(0,N,s(n)).
ackermann(s(M),0,Val) ← ackermann(M,s(0),Val).
ackermann(s(M),s(N),Val) ←
        ackermann(s(M),N,Val1), ackermann(M,Val1,Val).
```

Πρόγραμμα 10.9: Συνάρτιση του Ackermann

Ενα άλλο παράδειγμα άμεσης μετάφρασης ενός μαθηματικού ορισμού σε λογικό πρόγραμμα, είναι η συγγραφή ενός προγράμματος για τον ορισμό της συνάρτησης του Ackermann. Η συνάρτηση αυτή είναι το πιο απλό παράδειγμα μιας αναδρομικής συνάρτησης η οποία δεν είναι αρχικώς αναδρομική. Είναι μία συνάρτηση δύο ορισμάτων, ορισμένη από τρεις περιπτώσεις:

ackermann(0,N) = N+1.
ackermann(M,0) = ackermann(m-1,1).
ackermann(M,N) = ackermann(m-1,ackermann(m,N-1)).

Το πρόγραμμα 10.9 είναι μία μετάφραση του συναρτησιακού ορισμού σε λογικό πρόγραμμα. Το κατηγόρημα ackermann(M,N,A) έχει την έννοια A=ackermann(M,N). Ο τρίτος κανόνας περιέχει δύο κλήσεις της συνάρτησης ackermann, η μία για τον υπολογισμό της τιμής του δεύτερου ορίσματος.

Αυτό γίνεται περισσότερο φανερό στο συναρτησιακό ορισμό. Γενικά, η συναρτησιακή παράσταση είναι πιο αναγνώσιμη για καθαρά συναρτησιακούς ορισμούς όπως η συνάρτηση Ackermann. Ενα άλλο παράδειγμα φαίνεται στο πρόγραμμα 10.8α. Εκφράζοντας ότι X=Q.Y+Z, μία πρόταση που αφορά συναρτήσεις, είναι λίγο περίεργη με σχεσιακά λογικά προγράμματα.

Το τελικό παράδειγμα σε αυτό το τμήμα είναι ο Ευκλείδιος αλγόριθμος για την εύρεση του μέγιστου κοινού διαιρέτη δύο φυσικών αριθμών, μετασχηματισμένος σε λογικό πρόγραμμα. Οπως το παράδειγμα 10.8β, είναι ένα αναδρομικό πρόγραμμα όχι βασισμένο στην αναδρομική δομή των αριθμών. Το σχεσιακό πρόγραμμα είναι το gcd(X,Y,Z), με την ερμηνεία ότι το Z είναι ο μέγιστος κοινός διαιρέτης (ή ο gcd) των φυσικών αριθμών X και Y. Χρησιμοποιεί οποιδήποτε από τα προγράμματα 10.8α ή 10.8β, για το mod.

gcd(X,Y,Z) ← Z είναι ο ΜΚΔ των φυσικών αριθμών X και Y.

gcd(X,Y,Gcd) ← mod(X,Y,Z), gcd(Y,Z,Gcd).
gcd(x,0,X) ← X > 0.

Πρόγραμμα 10.10: Ο ευκλείδιος αλγόριθμος

Ο πρώτος κανόνας στο πρόγραμμα 10.10 είναι η λογική ουσία του Ευλείδιου αλγόριθμου. Ο ΜΚΔ των X και Y είναι ο ίδιος με τον ΜΚΔ των Y και X mod Y. Μία απόδειξη ότι το πρόγραμμα 10.10 είναι σωστό βασίζεται στην απόδειξη της παραπάνω μαθηματικής πρότασης για τους μέγιστους κοινούς διαιρέτες. Η απόδειξη ότι ο Ευκλείδιος αλγόριθμος είναι σωστός επίσης βασίζεται σε αυτό το αποτέλεσμα.
Το δεύτερο γεγονός στο πρόγραμμα 10.10 είναι το βασικό γεγονός. Πρέπει να διευκρινιστεί ότι το X είναι μεγαλύτερο από το μηδέν για να προβλέψει την περίπτωση gcd(0,0,0). Ο ΜΚΔ του 0 και 0 δεν είναι καλά ορισμένος.

10.3 Λίστες

Η βασική δομή για την αριθμητική είναι ο μοναδιαίος διαδοχικός functor. Αν και μπορούν να οριστούν πολύπλοκες αναδρομικές συναρτήσεις όπως η συνάρτηση του Ackermann, η χρήση της μοναδιαίας αναδρομικής δομής είναι περιορισμένη. Αυτό το τμήμα ασχολείται με τη δυαδική δομή, τη λίστα.

Το πρώτο όρισμα της λίστας αποτελεί ένα στοιχείο και το δεύτερο όρισμα είναι αναδρομικά το υπόλοιπο της λίστας. Οι λίστες είναι ικανοποιητικές για τους περισσότερους υπολογισμούς - όπως επιβεβαιώνει και η επιτυχία της γλώσσας προγραμματισμού LISP, η οποία χρησιμοποιεί τις λίστες ως τη βασική δομή δεδομένων. Οι περισσότερο πολύπλοκες δομές μπορούν να παραταθούν με λίστες, αν και είναι πιο βολική η χρήση διαφορετικών δομών όπου κρίνεται σκόπιμο.

Για τις λίστες, όπως άλλωστε και με τους αριθμούς, είναι απαραίτητο ένα σταθερό σύμβολο για τον τερματισμό της αναδρομής. Αυτή η "άδεια λίστα", που αναφέρεται ως κενή (nil), θα συμβολίζεται εδώ με το σύμβολο[]. Χρειαζόμαστε επίσης ένα functor πολλαπλότητας 2. Ιστορικά ο συνήθης functor για τη λίστα είναι ο "." (λέγεται dot), που ξεπερνά τη χρήση της τελείας. Είναι βολικό να καθορίσουμε μία ξεχωριστή ειδική σύνταξη. Ο όρος .(X,Y) συμβολίζεται [X|Y]. Τα συστατικά του έχουν ειδικά ονόματα: το X καλείται κεφαλή και το Y ουρά. Ο όρος [X| Y] ισοδυναμεί με το cons pair της LISP. Ο αντίστοιχος όρος για την κεφαλή και την ουρά είναι car και cdr.

Η εικόνα 10.2 παρουσιάζει τη σχέση μεταξύ λιστών γραμμένων με διαφορετική σύνταξη. Η πρώτη στήλη περιέχει λίστες με τον functor τελεία και είναι ο τρόπος που θεωρούνται ως όροι στα λογικά προγράμματα. Η δεύτερη στήλη δίνει την αντίστοιχη της σύνταξης με τελεία, σύνταξη με αγκύλες. Η τρίτη στήλη είναι μία βελτίωση της σύνταξης της δεύτερης, με την οποία κρύβουμε την αναδρομική δομή της λίστας. Σε αυτή την σύνταξη, οι λίστες γράφονται σαν μία ακολουθία στοιχείων που περιέχονται σε αγκύλες και διαχωρίζονται από κόματα. Η άδεια λίστα που χρησιμοποιείται για να τερματίσει την αναδρομική δομή, αποκρύπτεται. Παρατηρήστε τη χρήση του συμβολισμού "cons-pair" στην τρίτη στήλη, όταν η λίστα έχει ουρά μεταβλητή.

Οι όροι που σχηματίζονται με τον functor τελεία είναι πιο γενικοί από τις λίστες. Το πρόγραμμα 10.11 καθορίζει μία λίστα ακριβώς. Δηλωτικά ορίζει ότι: "

Λίστα είναι είτε η άδεια λίστα, είτε μία cons pair της οποίας η ουρά είναι μία λίστα." Το πρόγραμμα 10.11 είναι ανάλογο του προγράμματος 10.1 που ορίζει τους φυσικούς αριθμούς, και είναι ο απλός ορισμός τύπου των λιστών.

Η εικόνα 10.3 δείχνει ένα δένδρο απόδειξης για το στόχο list([a,b,c]). Στο δένδρο απόδειξης υπάρχουν βασικά στιγμιότυπα του προγράμματος 10.11, για παράδειγμα, list([a,b,c]) ←. list ([b,c]).

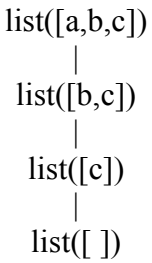
Αναφερόμαστε το συγκεκριμένο στιγμιότυπο, επειδή τα στιγμιότυπα λιστών σε μορφή cons pair μπορεί να προκαλούν σύγχιση. Το [a,b,c] είναι ένα στιγμιότυπο του [X| Xs] με τον περιορισμό {X=a, Xs=[b,c] }.

```
list(Xs) ←
        Xs είναι λίστα.

list([ ]).
list([X|Xs]) ← list(Xs).
```

Πρόγραμμα 10.11: Ορισμός της λίστας

Τυπική σύνταξη	Cons pair σύνταξη	Σύνταξη στοιχείου
.(a,[ ])	[a| [ ]]	[a]
.(a,.b,[ ]))	[a| [b| [ ]]]	[a,b]
.(a,.(b,.(c,[ ])))	[a| [b| [c | [ ]]]]	[a,b,c]
.(a,.(b,X))	[a| [b| X]]	[a,b| X]

Εικόνα 10.2: Ισοδύναμοι τύποι λιστών



Εικόνα 10.3: Δένδρο απόδειξης για την επαλήθευση λίστας

Επειδή η λίστες είναι πλουσιότερες δομές δεδομένων από τους αριθμούς, υπάρχει μεγάλη ποικιλία ενδιαφερόντων σχέσεων που μπορούμε να καθορίσουμε με αυτές. Η πιο βασική λειτουργία των λιστών, είναι ίσως ο καθορισμός εάν ένα στοιχείο ανήκει στη λίστα. Το κατηγόρημα που εκφράζει αυτή τη σχέση είναι το member(Element,List). Το πρόγραμμα 10.12 είναι ένας αναδρομικός ορισμός του member/2.

Δηλωτικά, η ανάγνωση του προγράμματος 10.12 είναι προφανής. Το X είναι ένα στοιχείο μιας λίστας, εάν είναι η κεφαλή της λίστας σύμφωνα με την πρώτη συνθήκη, ή εάν είναι ένα μέλος της ουράς της λίστας σύμφωνα με τη δεύτερη

συνθήκη. Ο σκοπός του προγράμματος είναι να καλύψει όλα τα βασικά στιγμιότυπα member(X,Xs), όπου το X είναι ένα στοιχείο του Xs.

```
member(Element,List) ←
        Element είναι ένα στοιχείο της Λίστας.

member(X,[X| Xs]).
member(X,[Y| Ys]) ← member(X,Ys).
```

Πρόγραμμα 10.12: Μέλος της Λίστας

```
prefix(Prefix,List) ←
      Prefix είναι ένα πρόθεμα της Λίστας.

prefix([ ],Ys).
prefix([X| Xs],[X|Ys})← prefix(Xs,Ys).

suffix(Suffix,List) ←
      Suffix είναι το επίθεμα της Λίστας.

suffix(Xs,Ys).
suffix(Xs,[Y| Ys]) ←.. suffix(Xs,Ys).
```

Πρόγραμμα 10.13: Προθέματα και επιθέματα της λίστας

Παραλείπουμε την κατάσταση τύπου στην πρώτη συνθήκη. Εναλλακτικά θα μπορούσε να γραφτεί

```
member(X,[X|Xs]) ← list(Xs).
```

Το πρόγραμμα αυτό έχει πολλές ενδιαφέρουσες εφαρμοφές. Οι βασικές του λειτουργίες είναι ο έλεγχος εάν ένα στοιχείο ανήκει στη λίστα με μία ερώτηση του τύπου member(b,[a,b,c])?, η εύρεση ενός στοιχείου της λίστας με μία ερώτηση όπως η member(X,[a,b,c])?, και η εύρεση μιας λίστας που περιέχει ένα στοιχείο με ερώτηση του τύπου member(b,X)? . Αυτή η τελευταία ερώτηση μπορεί να φαίνεται περίεργη, αλλά υπάρχουν προγράμματα που βασίζονται σε αυτή τη χρήση του member.

Χρησιμοποιούμε τις παρακάτω συμβάσεις όπου είναι δυνατόν, όταν ονομάζουμε μεταβλητές σε προγράμματα που χρησιμοποιούν λίστες. Εάν το X χρησιμοποιείται για να συμβολίσει την κεφαλή της λίστας, τότε το Xs θα συμβολίζει την ουρά της. Γενικότερα, τα πολλαπλά ονόματα μεταβλητών θα συμβολίζουν λίστες στοιχείων ενώ τα απλά ονόματα θα συμβολίζουν ξεχωριστά στοιχεία. Τα αριθμητικά επιθέματα θα συμβολίζουν πολλαπλές λίστες. Τα σχέδια σχέσεων θα εξακολουθούν να περιλαμβάνουν μνημονικά ονόματα.

Το επόμενο παράδειγμά μας είναι ένα κατηγόρημα sublist(Sub,List) που καθορίζει εάν toSub είναι υπολίστα της λίστας List. Η υπολίστα απαιτεί τα στοιχεία της να είναι διαδοχικά: η [a,b] είναι υπολίστα της [a,b,c,d], όχι όμως και η [a,c].

Είναι χρήσιμο να καθορίσουμε δύο ειδικές περιπτώσεις υπολιστών, για να

κάνουμε τον ορισμό της υπολίστας πιο εύκολο. Είναι καλή αρχή για τη σύνθεση λογικών προγραμμάτων, να καθορίζουμε νοηματικές σχέσεις σαν βοηθητικά κατηγορήματα.

Οι δύο περιπτώσεις που εξετάζονται είναι οι αρχικές υπολίστες ή τα προθέματα μιας λίστας και τερματικές υπολίστες ή επιθέματα μιας λίστας. Τα προγράμματα είναι ενδιαφέροντα από τη δική του πλευρά το καθένα.

Το κατηγόρημα prefix(Prefix,List) είναι αληθές εάν το Prefix είναι μία αρχική υπολίστα της Λίστας List, για παράδειγμα είναι αληθές ότι prefix([a,b],[a,b,c]) . Το αντίστοιχο κατηγόρημα suffix(Sufix,List) καθορίζει εάν το Suffix είναι τερματική υπολίστα της Λίστας List. Για παράδειγμα, είναι αληθές ότι suffix([b,c],[a,b,c]). Και τα δύο κατηγορήματα καθορίζονται στο Πρόγραμμα 10.13. Ο τύπος κατάστασης list(Xs) θα έπρεπε να προτεθεί σε κάθε βασικό γεγονος κάθε κατηγορήματος για να δώσει τη σωστή ερμηνεία.

Μία αυθαίρετη υπολίστα μπορεί να καθοριστεί με τη βοήθεια των προθεμάτων και των επιθεμάτων: ονομαστικά ως ένα επίθεμα ενός προθέματος ή σαν ένα πρόθεμα ενός επιθέματος. Το πρόγραμμα 10.14α εκφράζει το λογικό κανόνα ότι το Xs είναι μία υπολίστα της Ys εάν υπάρχει Ps τέτοιο ώστε το Ps να είναι ένα πρόθεμα του Ys και το Xs να είναι ένα επίθεμα του Ps. Το πρόγραμμα 10.14β είναι ο δυϊκός ορισμός της υπολίστας ως πρόθεμα ενός επιθέματος.

```
sublist(Sub,List) ←
      Sub είναι μία υπολίστα της Λίστας.
α:    Επίθεμα ή πρόθεμα
        sublist(Xs,Ys) ← prefix(Ps,Ys), suffix(Xs,Ps).
β:    Πρόθεμα ή επίθεμα
        sublist(Xs,Ys) ← prefix(Xs,Ss), suffix(Ss,Ys).
γ:    Αναδρομικός ορισμός της υπολίστας
        sublist(Xs,Ys) ← prefix(Xs,Ys).
        sublist(Xs,[Y| Ys]) ← sublist(Xs,Ys).
δ:    Επίθεμα ή πρόθεμα, χρησιμοποιώντας το append
        sublist(Xs,AsXsBs) ←
              append(As,XsBs,AsXsBs), append(Xs,Bs,XsBs).
ε:    Πρόθεμα ή επίθεμα, χρησιμοποιώντας το append
        sublist(Xs,AsXsBs) ←
              append(AsXs,Bs,AsXsBs), append(As,Xs,AsXs).
```

Πρόγραμμα 10.14: Καθορισμός υπολιστών μιας λίστας

```
append(Xs,Ys,XsYs) ←
      Xs,Ys είναι το αποτέλεσμα της σύζευξης των Xs και Ys.
append([ ],Ys,Ys).
append([X| Xs],Ys,[X| Zs]) ← apend(Xs,Ys,Zs).
```

Πρόγραμμα 10.15: Σύζευξη δύο λιστών

```
append([a,b],[c,d],[a,b,c,d])
                   |
```

append([b],[c,d],[b,c,d])
|
append([ ],[c,d],[c,d])

Εικόνα 10.4: Ενα δένδρο απόδειξης για τη σύζευξη δύο λιστών

Το κατηγόρημα prefix μπορεί επίσης να χρησιμοποιηθεί σαν η βάση του αναδρομικού ορισμού της υπολίστας. Αυτό φαίνεται στο Πρόγραμμα 10.14γ. Ο βασικός κανόνας λέει ότι ένα πρόθεμα μίας λίστας είναι υπολίστα της λίστας. Ο αναδρομικός κανόνας λέει ότι η υπολίστα μίας ουράς μίας λίστας είναι υπολίστα της ίδιας της λίστας.

Το κατηγόρημα member μπορεί να θεωρηθεί σαν ειδική περίπτωση υπολίστας με βάση τον κανόνα

member(X,Xs) ← sublist([X],Xs).

Η βασική πράξη με τις λίστες είναι η σύζευξη δύο λιστών σε μία τρίτη λίστα. Ετσι καθορίζεται μία σχέση append(Xs,Ys,Zs) μεταξύ δύο λιστών Xs, Ys και με αποτέλεσμα Zs την ένωση αυτών των δύο. Ο κώδικας για το append, το πρόγραμμα 10.15, είναι όμοιο σε δομή με το βασικό πρόγραμμα για την άθροιση δύο αριθμών, το πρόγραμμα 10.3 που ονομάσαμε plus.

Η εικόνα 10.4 δείχνει ένα δένδρο απόδειξης για τον στόχο append([a,b],[c,d],[a,b,c,d]). Η δομή του δένδρου φανερώνει ότι το μέγεθός του είναι γραμμικό στο μέγεθος της πρώτης λίστας. Γενικά, αν το Xs είναι λίστα n στοιχείων, το δένδρο απόδειξης για το append(Xs,Ys,Xs) έχει n+1 κόμβους.

Υπάρχουν πολλές χρήσεις για το append, παρόμοιες με εκείνες του plus. Η βασική χρήση είναι για τη σύζευξη δύο λιστών, θέτοντας ένα ερώτημα της μορφής append([a,b,c],[d,e],Xs)? με απάντηση Xs=[a,b,c,d,e]. Μία ερώτηση όπως append(Xs,[c,d],[a,b,c,d] βρίσκει τη διαφορά Xs=[a,b] μεταξύ των λιστων [c,d] και [a,b,c,d]. Αντίθετα με το plus, το append δεν είναι συμμετρικό για τα δύο πρώτα ορίσματά του, και γι'αυτό υπάρχουν δύο διακριτές εκδόσεις για την εύρεση της διαφοράς μεταξύ δύο λιστών.

```
reverse(List,Tsil) ←
    Tsil είναι το αποτέλεσμα την αντιστροφής της λίστας List.

α: Απλή αντιστροφή
    reverse([ ],[ ]).
    reverse([X| Xs],Zs) ← reverse(Xs,Ys), append(Ys,[X],Zs).

β: Υπολογιστική αντιστροφή
    reverse(Xs,Ys) ← reverse(Xs,[ ],Ys).

    reverse([X| Xs],Acc,Ys) ← reverse(Xs,[X| Acc],Ys).
    reverse([ ],Ys,Ys).
```

Πρόγραμμα 10.16: Αντιστροφή λίστας

Η ανάλογη διαδικασία με το διαχωρισμό ενός αριθμού είναι η διάσπαση της λίστας.

Η ερώτηση append(As,Bs,[a,b,c,d])? , για παράδειγμα, ψάχνει για τις λίστες AS, Bs, ώστε η σύζευξη των As και Bs να δίνει την λίστα [a,b,c,d]. Οι ερωτήσεις σχετικά με τη διάσπαση της λίστας γίνονται πιο ενδιαφέρουσες με το μερικό καθορισμό της φύσης της διασπασμένης λίστας. Τα κατηγορήματα member, sublist, prefix και suffix που παρουσιάστηκαν πριν μπορούν όλα να καθοριστούν με τη χρήση του append, με την παρακολούθηση της διαδικασίας της διάσπασης της λίστας.

Οι πιο ευθείς ορισμοί είναι για τα prefix και suffix, οι οποίοι απλώς καθορίζουν ποιό από τα δύο διεσπασμένα κομμάτια, έχει ενδιαφέρον:

prefix(Xs,Ys) ← append(Xs,As,Ys).
suffix(Xs,Ys) ← append(As,Xs,Ys).

Το sublist μπορεί να γραφεί με τη χρήση δύο στόχων append. Υπάρχουν δύο διακριτές μεταβλητές, που δίδονται στα Προγράμματα 10.14α και 10.14β, όπου τα suffix και prefix αντικαθίστανται από στόχους append.

Το member μπορεί να καθοριστεί με τη χρήση του append, όπως φαίνεται στη συνέχεια:

member(X,Ys) ← append(As,[X| Xs],Ys).

Αυτό σημαίνει ότι το X είναι μέλος του Ys αν το Ys μπορεί να χωριστεί σε δύο λίστες, όπου η X είναι η κεφαλή της δεύτερης λίστας.

Ενας παρόμοιος νόμος μπορεί να γραφεί για να εκφράσει τη σχέση adjacent(X,Y,Zs) όπου δύο στοιχεία X και Y είναι συνεχόμενα σε μία λίστα Zs:

adjacent(X,Y,Zs) ←.. append(As,[X,Y|Ys],Zs).

Μία άλλη σχέση που εκφράζεται εύκολα μέσω του append είναι ο καθορισμός του τελευταίου στοιχείου μιας λίστας. Το επιθυμητό σχήμα για το δεύτερο όρισμα στο append, μία λίστα με ένα στοιχείο, γίνεται μέσα από τον κανόνα:

last(X,XS) ← append(As,[X],Xs).

length(Xs,N) ←
    Η λίστα Xs έχει N στοιχεία.

length([ ],0).
length([x| Xs],s(N)) ← length(Xs,N).

Πρόγραμμα 10.17: Καθορισμός του μήκους μιας λίστας.

Επαναλαμβανόμενες εφαρμογές του append μπορούν να χρησιμοποιηθούν για τον ορισμό ενός κατηγορήματος reverse(List,Tsil). Η ερμηνεία του reverse είναι ότι η λίστα Tsil περιέχει τα στοιχεία της λίστας List, με αντίστροφη σειρά. Ενα παράδειγμα του στόχου για την ερμηνεία του προγράμματος είναι το reverse([a,b,c],[c,b,a]). Η απλή έκδοση που δίνεται στο πρόγραμμα 10.16α, είναι το λογικό ισοδύναμο της διατύπωσης της αναστροφής σε οποιαδήποτε γλώσσα: αναδρομικά αντιστρέψτε την ουρά της λίστας, και στη συνέχεια προσθέστε το πρώτο στοιχείο στο τέλος της αντιστραμένης ουράς.

Υπάρχει και εναλλακτικός τρόπος για τον καθορισμό του reverse χωρίς την

άμεση κλήση του append. Ορίζουμε ένα βοηθητικό κατηγόρημα reverse(Xs,Ys,Zs) το οποίο είναι αληθές εάν το Zs είναι το αποτέλεσμα της πρόσθεσης του Ys στα στοιχεία του Xs αντίστραμένα. Αυτό ορίζεται στο Πρόγραμμα 10.16β. Το κατηγόρημα reverse/3 σχετίζεται με το reverse/2 από την πρώτη συνθήκη του προγράμματος 10.16β.

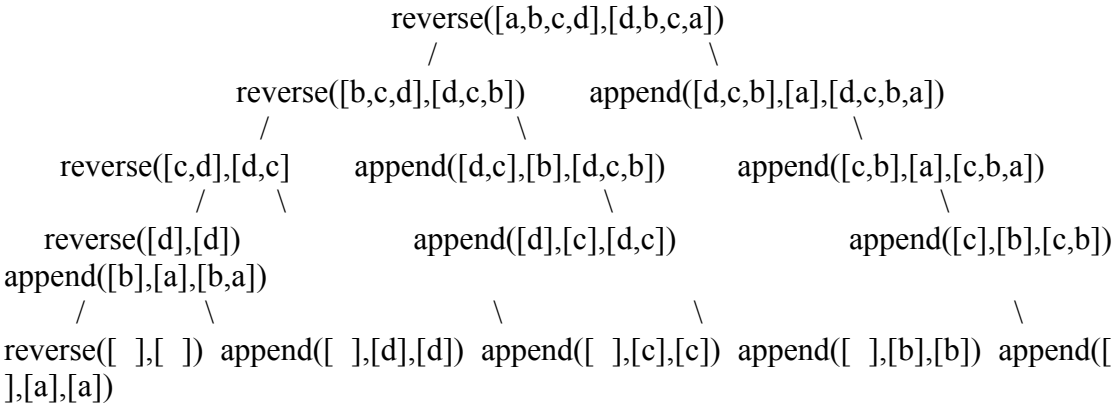
Το Πρόγραμμα 10.16β είναι πιο αποτελεσματικό από το Πρόγραμμα 10.16α. Παρατηρείστε την Εικόνα 10.5 με τα δένδρα απόδειξης για το στόχο reverse([a,b,c,d],[d,c,b,a]) χρησιμοποιώντας και τα δύο προγράμματα. Γενικά το μέγεθος του δένδρου απόδειξης για το Πρόγραμμα 10.16α είναι τετραγωνική συνάρτηση, όσον αφορά τον αριθμό των στοιχείων που θα αντιστραφούν, ενώ αυτό του προγράμματος 10.16β είναι γραμμικό.

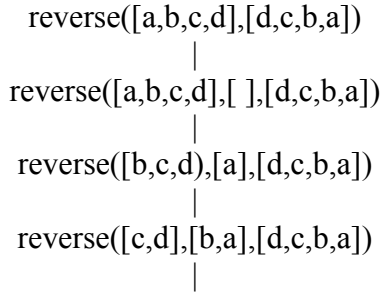
Το πλεονέκτιμα του προγράμματος 10.16β είναι η χρήση μιας καλύτερης δομής δεδομένων για την αναπαράσταση της σειράς των στοιχείων, την οποία συζητάμε με περισσότερη λεπτομέρεια στα Κεφάλαια 7 και 15.

Το τελευταίο πρόγραμμα σε αυτή την ενότητα, το Πρόγραμμα 10.17, εκφράζει μία σχέση μεταξύ αριθμών και λιστών, χρησιμοποιώντας την αναδρομική δομή του καθενός. Το κατηγόρημα length(Xs,N) είναι αληθές εάν το Xs έχει μήκος N, δηλαδή εάν περιέχει N στοιχεία, όπου N είναι ένας φυσικός αριθμός. Για παράδειγμα, το length([a,b],s(s(0))) φανερώνει ότι το [a,b] έχει δύο στοιχεία από την ερμηνεία του προγράμματος.

Ας εξετάσουμε τις πολλάπλές χρήσεις του Προγράμματος 10.17. Η ερώτηση length([a,b],X)? υπολογίζει το μήκος της λίστας [a,b], ίσο με 2. Κατά αυτόν τον τρόπο το length θεωρείται συνάρτηση λίστας με το συναρτησιακό ορισμό

```
length([ ]) = 0
length([X| Xs]) = s(length(Xs)).
```





reverse([d],[c,b,a],[d,c,b,a})
|
reverse([ ],[d,c,b,a],[d,c,b,a])

Εικόνα 10.5: Δένδρο απόδειξης για την αντιστροφή λίστας

Η ερώτηση length([a,b],s(s(0)))? ελέγχει εάν η λίστα [a,b] έχει μήκος 2. Η ερώτηση length(Xs,s(s(0)))? παράγει μία λίστα μήκους 2 με μεταβλητές για στοιχεία.

10.4 Σύνθεση αναδρομικών προγραμμάτων

Μέχρι στιγμής δεν έχει δοθεί καμμία εξήγηση σχετικά με το πως έχουν συνθεθεί τα λογικά προγράμματα των παραδειγμάτων. Ως ένα βαθμό, ισχυριζόμαστε ότι η σύνθεση λογικών προγραμμάτων είναι αποτέλεσμα μάθησης και όσμωσης, και σίγουρα πολλής εξάσκησης. Για τις απλές σχέσεις, η καλύτερη αξιωματοποίηση έχει μία αισθητική κομψότητα που φαίνεται προφανώς ορθή όταν γράφεται. Μέσω της λύσεως των ασκήσεων, ο αναγνώστης θα μάθει ότι υπάρχει διαφορά μεταξύ της αναγνώρισης και της σύνθεσης ορθών λογικών προγραμμάτων.

Αυτή η ενότητα περιέχει περισσότερα παραδείγματα προγραμμάτων που αφορούν λίστες. Κατά την παρουσίασή τους όμως, δίνεται περισσότερη έμφαση στο πώς τα προγράμματα πρέπει να κατασκευαστούν. Δύο αρχές παρουσιάζονται: πως αναμιγνείονται η διαδικαστική (procedural) και η δηλωτική (declarative) σκέψη, και πως αναπτύσσονται τα προγράμματα top-down.

Εχουμε δείξει τη δυϊκή ανάγνωση των προτάσεων: δηλωτική και διαδικαστική. Πώς αυτές συνδέονται όταν γράφουμε προγράμματα; Στην πράξη, σκεφτόμαστε διαδικαστικά όταν προγραμματίζουμε. Ομως, σκεφτόμαστε δηλωτικά όταν έχουμε να κάνουμε με θέματα αλήθειας και ερμηνείας. Ενας τρόπος να αναμίξουμε αυτά τα δύο είναι να γράφουμε διαδικαστικά και μετά να μεταφράζουμε το αποτέλεσμα σε δηλωτικές προτάσεις. Κατασκευάστε ένα πρόγραμμα με τον ένα τρόπο σκέψης: ύστερα αναρωτηθείτε αν ο εναλλακτικός τρόπος σκέψης δίνει δηλωτική έννοια. Εφαρμόζουμε αυτό σε ένα πρόγραμμα για τη διαγραφή στοιχείων από μια λίστα.

Το πρώτο, και πλέον βασικό βήμα, είναι ο προσδιορισμός της σκοπούμενης σημασίας της σχέσης. Υπάρχουν τρία ορίσματα που αφορούν τη διαγραφή στοιχείου από μία λίστα: το στοιχείο X που θα διεγραφεί, η λίστα L1 η οποία μπορεί να έχει εμφανίσεις του X και η λίστα L2 με όλα τις εμφανίσεις των διαγεγραμένων X. Μία κατάλληλη σχεσιακή πρόταση είναι η delete(L1,X,L2). Η φυσική ερμηνεία είναι όλα τα βασικά στιγμιότυπα όπου η L2 είναι η λίστα L1 με όλες τις εμφανίσεις του X απομακρισμένες.

Κατά τη συγγραφή του προγράμματος, είναι ευκολότερο να σκεφτόμαστε μία συγκεκριμένη λειτουργία. Θεωρήστε την ερώτηση delet([a,b,c,b],b,X)?, ένα τυπικό παράδειγμα της εύρεσης του αποτελέσματος της διαγραφής ενός στοιχείου από μία λίστα. Η απάντηση εδώ είναι X=[a,c]. Το πρόγραμμα θα είναι αναδρομικό στο πρώτο όρισμα. Ας σκεφτούμε τώρα διαδικαστικά.

Αρχίζουμε με το αναδρομικό μέρος. Η συνήθης μορφή των αναδρομικών ορισμάτων για λίστες είναι [X| Xs]. Υπάρχουν δύο εκδοχές που πρέπει να διακρίνουμε. Μια όπου το X είναι στοιχείο είναι ένα στοιχείο που θα διαγραφεί και

μία όπου το X είναι στοιχείο που δεν θα διαγραφεί. Στην πρώτη περίπτωση το αποτέλεσμα της αναδρομικής διαγραφής του X από την Xs είναι η επιθυμητή απάντηση της ερώτησης. Ο σωστός κανόνας είναι

delete([X| Xs],X,Ys]) ←. delete(Xs,X,Ys).

Η δηλωτική ανάγνωση αυτού του κανόνα είναι: "Η διαγραφή του X από την [X|Xs] δίνει Ys εάν η διαγραφή του X από την Xs είναι ίση με Ys." Η κατάσταση όπου η κεφαλή της λίστας και το στοιχείο που θα διαγραφεί είναι το ίδιο, καθορίζεται από κοινή μεταβλητή στην κεφαλή του κανόνα.

Η δέυτερη περίπτωση όπου το στοιχείο που θα διαγραφεί είναι διάφορο του X, της κεφαλής δηλαδή της λίστας, είναι παρόμοια. Το απαιτούμενο αποτέλεσμα είναι μία λίστα της οποίας η κεφαλή είναι X και η ουρά είναι το αποτέλεσμα της αναδρομικής διαγραφής του στοιχείου. Ο κανόνας είναι

delete([X| Xs],Z,[X| Ys]) ← X≠Z, delete(Xs,Z,Ys).

Η δηλωτική ανάγνωση του κανόνα είναι: "Η διαγραφή του Z από την [X|Xs} δίνει αποτέλεσμα [X|Ys] αν το Z είναι διάφορο του X και η διαγραφή του Z από το Xs δίνει αποτέλεσμα Ys." Σε αντίθεση με τον προηγούμενο κανόνα, η περίπτωση όπου η κεφαλή της λίστας και το στοιχείο που θα διαγραφεί είναι διαφορετικά, φαίνεται ρητώς στον κορμό του κανόνα.

Η βασική περίπτωση είναι η εξής. Κανένα στοιχείο δεν μπορεί να διαγραφεί από άδεια λίστα και το αποτέλεσμα είναι επίσης άδεια λίστα. Αυτό δίνει το γεγονός delete([ ],X,[ ]). Το ολοκληρομένο πρόγραμμα φαίνεται στο Πρόγραμμα 10.18.

```
delete(List,X,HasNoXs) ←
     Η λίστα HasNoXs είναι το αποτέλεσμα της μετακίνησης όλων
      των εμφανίσεων του X από τη Λίστα List.

delete([X| Xs],X,Ys) ← delete(Xs,X,Ys).
delete([X| Xs],Z,[X|Ys]) ← X≠Z, delete(Xs,Z,Ys).
delete([ ],X,[ ]).
```

Πρόγραμμα 10.18: Διαγραφή όλων των εμφανίσεων ενός στοιχείου από μία λίστα

```
select(X,HasXs,OneLessXs) ←
    Η λίστα OneLessXs είναι το αποτέλεσμα της μετακίνησης μιας εμφά-
    νισης του X από τη λίστα HasXs.

select(X,[X| Xs],Xs).
select(X,[Y| Ys],[Y| Zs]) ← select(X,Ys,Zs).
```

Πρόγραμμα 10.19: Επιλογή ενός στοιχείου από τη λίστα

Ας ξαναδούμε το πρόγραμμα που έχουμε γράψει, εξετάζοντας εναλλακτικούς σχηματισμούς. Παραλείποντας τον όρο X≠Z από το δεύτερο κανόνα του Προγράμματος 10.18 πέρνουμε μία παραλαγή του delete. Αυτή η παραλαγή έχει

λιγότερο φυσική ερμηνεία, αφού οποιοσδήποτε αριθμός εμφανίσεων ενός στοιχείου μπορεί να διαγραφεί. Για παράδειγμα, τα delete([a,b,c,b],b,[a,c]), delete([a,b,c,b],b,[a,c,b]), delete([a,b,c,b],b,[a,b,c]) και delete([a,b,c,b,b,[a,b,c,b]) είναι όλα στη λογική της παραλαγής αυτής.

Τόσο το Πρόγραμμα 10.18 όσο και η πάνω παραλλαγή περιέχουν στην ερμηνεία τους περιπτώσεις, όπου το στοιχείο που θα διαγραφεί δεν ανήκει σε καμμία λίστα, για παράδειγμα η delete([a],b,[a]) είναι αληθής. Υπάρχουν περιπτώσεις όπυ κάτι τέτοιο δεν είναι επιθημητό. Το πρόγραμμα 10.19 καθορίζει μία σχέση select(X,L1,L2) , η οποία έχει διαφορετική προσέγγιση για τα στοιχεία που δεν εμφανίζονται στη λίστα, Η ερμηνεία του select(X,L1,L2) είναι όλες οι βασικές περιπτώσεις όπου L2 είναι η λίστα L1 στην οποία ακριβώς μία εμφάνιση του X έχει απομακρυνθεί.

Το πρόγραμμα είναι μία διασταύρωση του Προγράμματος 10.12 για το member και του Προγράμματος 10.18 για το delete. Η δηλωτική ανάγνωση είναι:"Το X επιλέγεται από την [X| Xs] για να δώσει Xs ή το X επιλέγεται από την [Y|Zs] για να δώσει [Y|Zs] αν το X επιλέγεται από την Ys για να δώσει Zs." Χρησιμοποιούμε το select για να κατασκευάσουμε ένα απλό πρόγραμμα για την ταξινόμιση λιστών, που παρουσιάζεται παρακάτω.

Μεγάλη ώθηση στον προγραμματισμό έχει δωθεί από την έμφαση στη μεθοδολογία του top-down σχεδιασμού, όπως επίσης και από τον βηματικό εξευγενισμό. Γενικά, η μεθοδολογία είναι να ορίσεις το γενικό πρόβλημα, να το διασπάσεις σε υποπροβλήματα και τέλος να λύσεις τα κομμάτια. Ο top-down σχεδιασμός είναι ένας φυσικός τρόπος για να συνθέτουμε λογικά προγράμματα. Η περιγραφή των προγραμμάτων μέσα σε αυτό το βιβλίο θα είναι κυρίως top-down. Το υπόλοιπο αυτού του τμήματος περιγράφει τη σύνθεση δύο προγραμμάτων για την ταξινόμηση μιας λίστας: την ταξινόμηση μετάθεσης και τη γρήγορη ταξινόμηση. Ο top-down χαρακτήρας τους τονίζεται ιδιαίτερα.

Ενας λογικός καθορισμός της ταξινόμησης είναι η εύρεση ενός κανόνα μετάθεσης της λίστας. Αυτό μπορεί να γραφεί άμεσα σαν λογικό πρόγραμμα. Η βασική σχέση είναι η sort(Xs,Ys), όπου η Ys είναι η λίστα που περιέχει τα στοιχεία της Xs ταξινομημένα με αύξουσα σειρά:

sort(Xs,Ys) ← permutation(Xs,Ys), ordered(Ys).

Ο υψηλού επιπέδου στόχος της ταξινόμησης έχει αποσυντεθεί. Τώρα πρέπει να καθορίσουμε τα permutation και ordered.

Ο έλεγχος του εάν μία λίστα βρίσκεται σε αύξουσα σειρά μπορεί να εκφραστεί με δύο συνθήκες που δίδονται παρακάτω. Το γεγονός λέει ότι μία λίστα με ένα στοιχείο είναι οποσδήποτε ταξινομημένη. Ο κανόνας λέει ότι μία λίστα είναι ταξιμομημένη αν το πρώτο στοιχείο είναι μικρότερο ή ίσο από το το δεύτερο και αν η υπόλοιπη λίστα, αρχίζοντας από το δεύτερο στοιχείο, είναι ταξινομημένη:

ordered([X]).
ordered([X,Y| Ys]) ← X≤Y, ordered([Y| Ys]).

Το πρόγραμμα για το permutation είναι πιο εκλεπτισμένο. Μία άποψη της διαδικασίας μετάθεσης μίας λίστας είναι η μη-ντετερμινιστική επιλογή ενός στοιχείου ως πρώτου της μεταθετημένης λίστας και η αναδρομική μετάθεση της υπόλοιπης λίστας. Μεταφράζουμε αυτή την άποψη σε λογικό πρόγραμμα για το permutation, χρησιμοποιώντας το Πρόγραμμα 10.19 για το select. Το βασικό γεγονός λέει ότι η

άδεια λίστα είναι μοναδική μετάθεση του εαυτού της:

```
permutation(Xs,[Z| Zs]) ← select(Z,Xs,Ys), permutation(Ys,Zs).
permutation([ ],[ ]).
```

Μία άλλη διαδικαστική άποψη για την γέννηση μεταθέσεων λιστών είναι η αναδρομική μετάθεση της ουράς της λίστας και η εισαγωγή της κεφαλής σε αυθαίρετη θέση. Αυτή η άποψη μπορεί επίσης να μεταφραστεί άμεσα σε κώδικα. Το βασικό μέρος είναι όμοιο με την προηγούμενη έκδοση:

```
permutation([X| Xs],Zs) ← permutation(Xs,Ys), insert(X,Ys,Zs).
permutation([ ],[ ]).
```

Το κατηγόρημα insert μπορεί να οριστεί με τη βοήθεια του Προγράμματος 10.19 για το select.

```
insert((X,Ys,Zs) ← select(X,Zs,Ys).
```

Και οι δύο εκδόσεις για το permutation έχουν καθαρές δηλωτικές αναγνώσεις.

Το "απλό" πρόγραμμα ταξινόμησης, το οποίο ονομάζουμε μεταθετική ταξινόμηση, το έχουμε συγκεντώσει στο Πρόγραμμα 10.20. Είναι ένα παράδειγμα του παρήγαγε-και-δοκίμαζε ((generate-and-test) προτύπου.

Το πρόβλημα της ταξινόμησης λιστών είναι καλά μελετημένο. Η μετάθεση λίστας δεν είναι καλή μέθοδος για την ταξινόμηση λιστών στην πράξη. Πολύ καλύτεροι αλγόριθμοι προκύπτουν από την εφαρμογή της στρατηγικής "διαίρει και βασίλευε" στην ταξινόμηση. Ο στόχος μας προκύπτει με το να διαιρέσουμε τη λίστα σε δύο κομμάτια, να ταξινομήσουμε αναδρομικά τα δύο κομμάτια και τέλος να ενώσουμε τα δύο κομμάτια ώστε να προκύψει η ταξινομημένη λίστα. Οι μέθοδοι για τη διαίρεση και την ένωση της λίστας πρέπει να περιγραφούν. Υπάρχουν δύο ακραίες θέσεις. Η πρώτη είναι να κάνουμε τη διαίρεση δύσκολη και την ένωση εύκολη. Αυτή η προσέγγιση προκύπτει από τον αλγόριθμο quicksort. Δίνουμε ένα λογικό πρόγραμμα για το quicksort παρακάτω. Η δεύτερη θέση είναι να κάνουμε την διαίρεση εύκολη και την ένωση δύσκολη. Αυτή είναι η προσέγγιση της ταξινόμησης με συγχώνευση (merge sort) και η ταξινόμηση εισαγωγής στο Πρόγραμμα 10.21.

```
sort(Xs,Ys) ←
    Η λίστα Ys είναι μία ταξιμομημένη μετάθεση της λίστας Xs.

sort(Xs,Ys) ← permutation(Xs,Ys), ordered(Ys).
permutation(Xs, [Z| Zs]) ← select(Z,Xs,Ys), permutation(Ys,Zs).
permutation([ ],[ ]).

ordered([X]).
ordered([X,Y| Ys]) ← X ≤ Y, ordered([Y| Ys]).
```

Πρόγραμμα 10.20: Μεταθετική ταξινόμηση

```
sort(Xs,Ys) ←
    Η λίστα Ys είναι μία ταξινομημένη μετάθεση της λίστας Xs.
```

```
sort([X| Xs],Ys) ← sort(Xs,Ys), insert(X,Zs,Ys).
sort([ ],[ ]).

insert(X,[ ],[X]).
insert(X,[Y| Ys],[Y| Zs]) ← X > Y, insert(X,Ys,Zs).
insert(X,[Y| Ys],[X,Y| Ys]) ← X ≤ Y.
```

Πρόγραμμα 10.21: Ταξινόμηση με εισαγωγή

Στην ταξινόμηση εισαγωγής (insertion sort), ένα στοιχείο, (τυπικά το πρώτο) μετακινείται από την λίστα. Η υπόλοιπη λίστα ταξινομείται αναδρομικά. Στη συνέχεια το στοιχείο εισάγεται, διατηρώντας τη διάταξη της λίστας.

```
sort(Xs,Ys) ←
    Η λίστα Ys είναι μία ταξινομημένη μετάθεση της λίστας Xs.
quicksort([X| Xs],Ys) ←
        partition(Xs,X,Littles,Bigs),
        quicksort(Littles,Ls),
        quicksort(Bigw,Bs),
        append(Ls,[X|Bs],Ys).
quicksort([ ],[ ]).
partition([X| Xs],Y,[X| Ls],Bs) ← X ≤ Y, partition(Xs,Y,Ls,Bs).
partition([X| Xs],Y,Ls,[X| Bs]) ← X > Y, partition(Xs,Y,Ls,Bs).
partition([ ],Y,[ ]).
```

Πρόγραμμα 10.22: Quicksort

Το quicksort διαιρεί τη λίστα διαλέγοντας ένα τυχαίο στοιχείο της και χωρίζοντας τα υπόλοιπα στοιχεία της λίστας σε μικρότερα ή μεγαλύτερα από το επιλεγμένο στοιχείο. Η ταξινομημένη λίστα συντίθεται από τα μικρότερα στοιχεία, ακολουθούμενα από το επιλεγμένο στοιχείο και τέλος από τα μεγαλύτερά του. Το πρόγραμμα που περιγράφουμε επιλέγει το πρώτο στοιχείο της λίστας ως βάση για τον διαχωρισμό.

Το Πρόγραμμα 10.22 ορίζει το sort χρησιμοποιώντας τον αλγόριθμο quicksort. Ο αναδρομικός κανόνας για το sort λέει: "Η Ys είναι η ταξινομημένη εκδοχή της [X| Xs] εάν τα Littles και Bigs είναι το αποτέλεσμα του διαχωρισμού του Xs σύμφωνα με το X, Ls και Bs είναι το αποτέλεσμα της ταξινόμησης των Littles και Bigs αναδρομικά, και Ys είναι το απτέλεσμα της προσάρτισης της [X| Bs] στην Ls.

Ο διαχωρισμός μίας λίστας είναι άμεσος, και είναι ίδιος με το πρόγραμμα για τη διαγραφή στοιχείων. Υπάρχουν δύο συνθήκες που πρέπει να λάβουμε υπ'όψη μας: όταν η τρέχουσα κεφαλή της λίστας είναι μικρότερη από το στοιχείο που χρησιμοποιείται για το διαχωρισμό, και όταν η κεφαλή είναι μεγαλύτερη από το στοιχείο διαχωρισμού. Η δηλωτική ανάγνωση της πρώτης συνθήκης του partition είναι: " Ο διαχωρισμός μίας λίστας της οποίας η κεφαλή είναι το X και η ουρά είναι η Xs σύμφωνα με το στοιχείο Y, δίνει τις λίστες [X|Littles] και Bigs, εάν το X είναι μικρότερο ή ίσο του Y και ο διαχωρισμός της Xs σύμφωνα με το στοιχείο Y δίνει τις λίστες Littles και Bigs." Η δεύτερη συνθήκη για το partition έχει ανάλογη ερμηνεία. Η βασική συνθήκη είναι ότι η άδεια λίστα διαχωρίζεται σε δύο άδειες λίστες.

10.5 Δυαδικά δένδρα

Ο επόμενος αναδρομικός τύπος δεδομένων που εξετάζουμε είναι τα δυαδικά δένδρα. Αυτές οι δομές έχουν σημαντική θέση σε πολλούς αλγορίθμους.

Τα δυαδικά δένδρα παριστάνονται από το τριαδικό functor tree(Element,Left,Right), όπου Element είναι το στοιχείο του κόμβου και Left και Right είναι το αριστερό και δεξί υποδένδρο του κόμβου αντίστοιχα. Το άδειο δένδρο παριστάνεται με τη βοήθεια του void. Για παράδειγμα το δέντρο

```
  a
 / \
b   c
```

παριστάνεται ως tree(a,tree(b,void,void),tree(c,void,void)).

Τα λογικά προγράμματα που χειρίζονται τα δυαδικά δένδρα είναι όμοια με εκείνα που χειρίζονται τις λίστες. Οπως και με τους αριθμούς ή τις λίστες, αρχίζουμε με ορισμό του τύπου του δυαδικού δένδρου. Δίνεται στο Πρόγραμμα 10.23. Παρατηρείστε ότι το πρόγραμμα είναι διπλά αναδρομικό, δηλαδή υπάρχουν δύο στόχοι στον κορμό του αναδρομικού κανόνα με το ίδιο κατηγόρημα σαν κεφαλή του κανόνα. Αυτό οφείλεται στη διπλά αναδρομική φύση των δυαδικών δένδρων, και θα εμφανίζεται και στα υπόλοιπα προγράμματα αυτού του τμήματος.

Ας γράψουμε κάποια προγράμματα επεξεργασίας δένδρων. Το πρώτο μας παράδειγμα ελέγχει εάν ένα στοιχείο εμφανίζεται σε ένα δένδρο. Η σχεσιακή πρόταση είναι η tree_member((Element,Tree).

```
binary_tree(Tree) ←
        Το Tree είναι ένα δυαδικό δένδρο.

binary_tree(void).
binary_tree(tree(Element,Left,Right)) ←
        binary_tree(Left), binary_tree(Right).
```

Πρόγραμμα 10.23: Ορισμός δυαδικού δένδρου

```
tree_member(Element,Tree) ←
        Element είναι ένα στοιχείο του δυαδικού δένδρου Tree.


tree_member(X,tree(X,Left,Right)).
tree_member(X,tree(Y,Left,Right)) ← tree_member(X,Right).
tree_member(X,tree(Y,Left,Right)) ← tree_member(X,Right).
```

Πρόγραμμα 10.24: Ελεγχος εάν το στοιχείο είναι μέλος του δένδρου

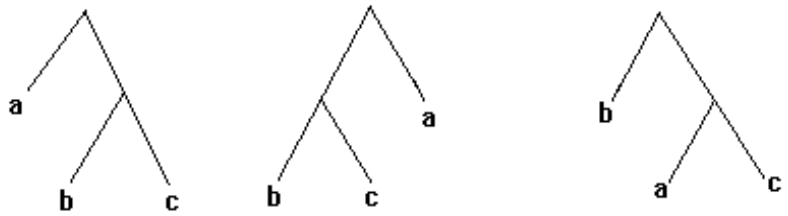
```
isotree(Tree1,Tree2) ←
        Τα Tree1 και Tree2 είναι ισομορφικά δυαδικά δένδρα

isotree(void,void).
isotree(tree(X,Left1,Right1),tree(X,Left2,Right2) ←
        isotree(Left1,Left2), isotree(Right1,Right2)).
isotree(tree(X,Left1,Right1),tree(X,Left2,Right2) ←
        isotree(Left1,Right2), isotree(Right1,Left2)).
```

Πρόγραμμα 10.25: Καθορισμός του εάν τα δένδρα είναι ισομορφικά

Η σχέση είναι αληθής αν το Element είναι ένας κόμβος του δένδρου. Το Πρόγραμμα 10.24 περιέχει τον ορισμό. Η δηλωτική ανάγνωση του προγράμματος είναι: "το X είναι μέλος του δένδρου αν είναι στοιχείο του κόμβου (από το γεγονός) ή αν είναι μέλος του αριστερού ή του δεξιού υποδένδρου (από τους δύο αναδρομικούς κανόνες)."

Οι δύο κλάδοι ενός δυαδικού δένδρου είναι διακριτοί, αλλά για πολλές εφαρμογές η διάκριση δεν είναι σημαντική ως προϋπόθεση. Συνεπώς μία χρήσιμη έννοια είναι ο ισομορφισμός, που καθορίζει αν δύο όχι ταξινομημένα δένδρα είναι ουσιαστικά τα ίδια. Δύο δυαδικά δένδρα T1 και T2 είναι ισομορφικά αν το T2 μπορεί να κατασκευαστεί από την αναδιάταξη των κλαδιών των υποδένδρων του T1. Η εικόνα 10.6 δειχνει τρία απλά δυαδικά δένδρα. Τα δύο πρώτα είναι ισομορφικά, όχι όμως και το τρίτο.

Ο ισομορφισμός είναι μία ισοδύναμη σχέση με ένα απλό αναδρομικό ορισμό. Δύο κενά δένδρα είναι ισομορφικά. Αλλιώς δύο δένδρα είναι ισομορφικά αν έχουν τα ίδια στοιχεία στον κόμβο και τα αριστερά και δεξιά υποδένδρα τους είναι ισομορφικά, ή το αριστερό υποδένδρο του ενός είναι ισομορφικό με το δεξί υποδένδρο του άλλου και τα υπόλοιπα δύο υποδένρα είναι επίσης ισομορφικά.



Εικόνα 10.6: Σύγκριση δένδρων για ισομορφισμό

Το Πρόγραμμα 10.25 ορίζει ένα κατηγόρημα isotree(Tree1,Tree2) το οποίο είναι αληθές εάν τα Tree1 και Tree2 είναι ισομορφικά. Το κατηγόρημα είναι συμμετρικό για τα ορίσματά του.

Τα προγράμματα που σχετίζονται με τα δυαδικά δένδρα περιέχουν διπλή αναδρομή, μία για κάθε κλάδο του δένδρου. Η διπλή αναδρομή μπορεί να δηλωθεί με

δύο τρόπους. Τα προγράμματα μπορούν να έχουν δύο ξεχωριστές υποθέσεις, όπως στο Πρόγραμμα 10.24 για το tree_member. Σε αντίθεση, το Πρόγραμμα 10.12 που ελέγχει για τα μέλη μιας λίστας έχει μόνο μία αναδρομική περίπτωση. Εναλλακτικά ο κορμός της αναδρομικής συνθήκης έχει δύο αναδρομικές κλήσεις, όπως με καθέναν από τους αναδρομικούς κανόνες για το isotree του Προγράμματος 10.25.

Μπορεί να γραφεί ένα πρόγραμμα για την αντικατάσταση στοιχείων μέσα σε δυαδικά δένδρα. Το κατηγόρημα substitute(X,Y,ldTree,NewTree) είναι αληθές εάν το NewTree είναι το αποτέλεσμα της αντικατάστασης όλων των εμφανίσεων του X με Y στο OldTree. Μία αξιωματοποίηση του substitute/4 δίδεται στο Πρόγραμμα 10.26.

Πολλές εφαρμογές που σχετίζονται με δένδρα απαιτούν πρόσβαση στα στοιχεία που εμφανίζονται στους κόμβους. Η κεντρική ιδέα είναι αυτή ενός δένδρου προσπελάσιμου, το οποίο είναι μία ακολουθία των κόμβων του δένδρου σε μια προκαθορισμένη σειρά. Υπάρχουν τρεις περιπτώσεις για τη γραμμική διάταξη της προσπέλασης: η προδιάταξη, όπου η τιμή του κόμβου είναι πρώτη, μετά είναι οι κόμβοι του αριστερού υποδένδρου, ακολουθούμενοι από τους κόμβους του δεξιού υποδένδρου, η ενδοδιάταξη, όπου οι αριστεροί κόμβοι είναι πρώτοι, ακολουθούμενοι από τον ίδιο τον κόμβο και τους δεξιούς κόμβους, και η μεταδιάταξη όπου ο κόμβος έρχεται μετά το αριστερό και το δεξί υποδένδρο.

Ενας ορισμός για κάθε μία από τις τρεις προσπελάσεις δίνεται στο Πρόγραμμα 10.27. Η αναδρομική δομή είναι ίδια. Η μόνη διαφορά μεταξύ τους είναι η σειρά των στοιχείων που συνθέτονται απο τους διάφορους στόχους append.

```
substitute(X,Y,TreeX,TreeY) ←
        Το δυαδικό δέντρο Tree Y είναι το αποτέλεσμα της αντικατάστασης
        όλων των συμβαίνοντων του X στο δυαδικό δέντρο TreeX από το Y.
substitute(X,Y,void,void).
substitute(X,Y,tree(X,Left,Right)),tree(Y,Left1,Right1))                ←
substitute(X,Y,Left,Left1), substitute(X,Y,Right,Right1).
substitute(X,Y,tree(Z,Left,Right)),tree(Z,Left1,Right1)) ←
        X≠Z, substitute(X,Y,Left,Left1), substitute(X,Y,Right,Right1).
```

Πρόγραμμα 10.26: Αντικατάσταση ενός όρου σε ένα δέντρο

```
pre_order(Tree,Pre) ←
        Pre is a pre-order traversal of the binary tree Tree

pre_order(tree(X,L,R),Xs) ←
        pre_order(L,Ls),pre_order(R,Rs),append([X|Ls].Rs,Xs).
pre_order(void,[ ]).

in_order(Tree,In)←
In is an in-order traversal of the binary tree Tree.

in_order(tree(X,L,R),Xs) ←
        in_order(L,Ls), in_order(R,Rs),append(Ls,[X|Rs],Xs).
in_order(void,[ ]).

post_order(tree(X,L,R),Xs) ←
        post_order(L,Ls),
        post_order(R,Rs),
```

```
        append(Rs,[X],Rs1);
        append(Ls,Rs1,Xs).
post_order(void,[ ]).
```

Πρόγραμμα 3.27 :Διασταύρωση του δυαδικού δέντρου

10.6 Χειρισμός συμβολικών εκφράσεων

Τα λογικά προγράμματα που επεξηγούνται ως τώρα σε αυτό το κεφάλαιο έχουν χειριστεί φυσικούς αριθμούς, λίστες και δυαδικά δέντρα. Ο τρόπος προγραμματισμού παρουσιάζεται περισσότερο γενικά. Το κεφ. αυτό δίνει 4 παραδείγματα αναδρομικών προγραμματων - ένα πρόγραμμα που ορίζει πολυώνυμα, ένα πρόγραμμα για συμβολική διαφοροποίηση, ένα πρόγραμμα για την λύση του προβλήματος του πύργου του Hanoi, και ένα πρόγραμμα για την εξέταση της ικανοποίησης της φόρμουλας Boolean.

Το πρώτο παράδειγμα είναι ένα πρόγραμμα για την αναγνώριση πολυωνύμων σε κάποιον όρο X. Τα πολυώνυμα ορίζονται επαγωγικά.Το X από μόνο του είναι ένα πολυώνυμο του X, όπως είναι κάθε σταθερά. Αθροίσματα διαφορές και παράγωγα πολυωνύμων του X είναι πολυώνυμα του X. Οπως επίσης είναι και πολυώνυμα υψωμένα σε δύναμη ενός φυσικού αριθμού και διαίρεση ενός πολυώνυμου από μια σταθερά.

Ενα παράδειγμα πολυώνυμου του x είναι το x^2 - 3x + 2 . Αυτό προκύπτει από το άθροισμα των πολύωνύμων x^2 - 3x και 2, όπου x^2 - 3x αναγνωρίζεται

Ενα λογικό πρόγραμμα, για την αναγνώριση πολυωνύμων αποκτάται, εκφράζοντας τους κανόνες που δόθηκαν συνοπτικά παραπάνω στην κατάλληλη μορφή. Το πρόγραμμα 3.28 ορίζει την σχέση polynomial(Expression, X) η οποία είναι αληθής αν η “Expression” είναι ένα πολυώνυμο του X. Δίνουμε 2 κανόνες από το πρόγραμμα. Η σταθερά polynomial(X,X) λέει ότι ο όρος X είναι ένα πολυώνυμο από μόνο του. Ο κανόνας λέει ότι το άθροισμα Term 1 + Term 2 είναι πολυώνυμα του X αν τα Term 1 και Term 2 είναι πολυώνυμα του X.

```
        polynomial(Term1+ Term2,x) ←
                polynomial(Term1,X), polynomial(Term2,X)
        polynomial (Expression,X) ←
                η “Expression” είναι ένα πολυώνυμο του X          polynomial (X,X)
        polynomial(Term,X) ←
                σταθερά (Term).
        polynomial (Term1+Term2,X) ←
                polynomial(Term1,X), polynomial (Term 2,X)          polynomial
(Term1-Term2,X) ←
                polynomial(Term1,X), polynomial(Term 2,X).
        polynomial(Term1*Term2,X) ←
                polynomial (Term1,X), polynomial(Term2,X).
        Polynomial(Term1/Term2,X) ←
                Polynomial(Term1,X), constant(Term2).
```

```
polynomial(Term↑N,X) ←
        natural number(N), polynomial(Term,X).
```

Πρόγραμμα 10.28 : Αναγνωρίζοντας πολυώνυμα

Αλλοι τύποι που χρησιμοποιούνται στο πρόγραμμα 3.28 είναι η χρήση της constant για την αναγνώριση σταθερών, και ο δυαδικός functor ↑ που δηλώνει δείκτες. Ο όρος X↑Y δηλώνει το X^Y.

Το επόμενο παράδειγμα είναι ένα πρόγραμμα για παραγώγιση. Το σχετικό σχήμα είναι derivative(Expression,X, DifferentiatedExpression). Η έννοια του derivative είναι ότι η DifferentiatedExpression είναι η παράγωγος της Expression ως προς την X.

Οσο για το πρόγραμμα 10.28 για την αναγνώριση πολωνύμων, ένα λογικό πρόγραμμα για την διαφοροποίηση είναι απλά μια συλλογή του σχετικού κανόνα διαφοροποίησης, γραμμένο σε σωστή σύνταξη. Για παράδειγμα το γεγονός:

```
derivative (X,X,s,(0)).
```

εκφράζει ότι η παράγωγος του X ως προς τον εαυτό του είναι 1. Το γεγονός:

```
derivative (sin (X),X, cos (X)).
```

διαβάζεται : "Η παράγωγος του sin (X) ως προς X είναι cos (X)". Ενα αντιπροσωπευτικό δείγμα λειτουργιών και των παραγώγων τους δίνονται στο πρόγραμμα 10.29

Τα αθροίσματα και οι παράγωγοι των όρων έχουν μετατραπεί χρησιμοποιώντας τον κανόνα άθροισης και κανόνα παραγώγισης αντιστοίχως. Ο κανόνας άθροισης δηλώνει ότι η παράγωγος του αθροίσματος είναι το άθροισμα των παραγώγων. Η κατάλληλη πρόταση είναι :

```
derivative( F + G,X, DF + DG) ←
        derivative( F,X DF), derivative(G,X,DG)
```

```
derivative(Expression,X,DiferentiatedExpression) ←
        DifferentiatedExpression is the derivative of Expression with respect to X.

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).

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+DF∗G) ←
        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).
```

Πρόγραμμα 10.29 Κανόνες παραγώγισης

Ο κανόνας παραγώγισης για το γινόμενο είναι λίγο πιο περίπλοκος αλλά η λογική πρόταση είναι απλώς ένας μαθηματικός ορισμός .

```
derivative(F∗G,X,F∗DG+DF∗G) ←
        derivative(F,X,DF), derivative(G,X,DG).
```

Το πρόγραμμα 10.29 επίσης περιέχει προσθετικούς και διαιρετικούς κανόνες. Ο αλυσιδωτός κανόνας είναι λίγο περισσότερο περίπλοκος. Δηλώνει ότι η παράγωγος του f(g(x)) ως προς χ είναι η παράγωγος του αφαιρετικού f(g(x)) ως προς g(x), επί την παράγωγο του g(x) ως προς χ. Οπως αναφέρθηκε εμπλέκει την ποσότητα σε σχέση με τις λειτουργικότητες, και βρίσκεται έξω από τα όρια του λογικού προγραμματισμού που εμείς έχουμε ήδη παρουσιάσει.

hanoi (N,A,B,C, Moves) ← Το "Moves" είναι μια επανάληψη κινήσεων για να λυθεί το puzzle των πύργων του Hanoi με N δίσκους και 3 στύλους A,B, και C.

```
hanoi(s(0),A,B,C,[A to B]).
hanoi(s(N), A,B,C,Moves) ←
        hanoi (N,A,AC,B,Ms1),
        hanoi(N,C,B,A,Ms2),
        append(Ms1, [A to B |Ms2], Moves).
```

Πρόγραμμα 10.30 Πύργος του Hanoi

Παρόλ'αυτά μια πτυχή του αλυσιδωτού κανόνα είναι δυνατή για κάθε συγκεκριμένη λειτουργία. Για π.χ. δίνουμε τον κανόνα για την μετατροπή του X^N και sin(X) :

```
derivative(U↑s(N),X,s(N)∗U↑N∗DU)←derivative(U,X,DU).
derivative(sin(U),X,cos(U) ∗DU) ← derivative(U,X,DU).
```

Η δυσκολία, διατύπωσης του αλυσιδωτού κανόνα όσο αφορά τις διαφοροποίησεις, ανακύπτει από την επιλογή παρουσίασης των όρων. Και τα δύο προγράμματα 10.28 και 10.29 χρησιμοποιούν την "φυσική" παρουσίαση τους μέσα από τα μαθηματικά όπου οι όροι παρουσιάζονται από μόνοι τους. Ενας όρος όπως ο sin (X) παρουσιάζεται χρησιμοποιώντας την ενοποιητική σταθερά sin. Αν έχει χρησιμοποιηθεί μια διαφορετική παρουσίαση , π.χ. unary_term (sin, X) όπου το

όνομα της δομής έχει γίνει δεκτό, τότε το πρόβλημα με την αλυσίδα κανόνων εξαφανίζεται. Η αλυσίδα κανόνων μπορεί τότε να πάρει την μορφή ως εξής :

derivative(unary_term(F,U),X,DF∗DU) ←
    derivative(unary_term(F,U),U,DF),derivative(U,X,DU).

Σημειώσε ότι όλοι οι κανόνες στο πρόγραμμα 10.29 θα έπρεπε να αναπροσαρμοστούν σε όρους αυτής της νέας απεικόνισης και θα εμφανίζονται έτσι λιγότερο φυσικοί.

Οι άνθρωποι, παίρνουν σαν δεδομένο την αυτόματη απλοποίηση των εκφράσεων όταν έχουμε μετατροπές εκφράσεων. Η απλοποίηση λείπει από το πρόγραμμα 10.29. Η απάντηση στην ερώτηση

derivative (3∗X + 2, X, D) ? είναι D = (3∗1 + 0∗x) + 0.

Θα μπορούσαμε κατευθείαν να απλοποιήσουμε το D σε 3 αλλά δεν καθορίζεται από το λογικό πρόγραμμα.

Το επόμενο παράδειγμα είναι η λύση στο πρόβλημα του πύργου του Hanoi, που είναι ένα τυπικό εισαγωγικό παράδειγμα για την χρήση της επανάληψης. Το πρόβλημα είναι η μεταφορά ενός πύργου με n δίσκους από τον ένα στήλο στον άλλο με την βοήθεια ενός βοηθητικού στήλου και όπου ο μεγαλύτερος δίσκος δεν μπορεί ποτέ να τοποθετηθεί πάνω από έναν μικρότερο δίσκο.

Υπάρχει ένας μύθος που συνδέεται με το παιχνίδι. Κάπου κρυμμένο στα περίχωρα του Hanoi, σε ένα αφανές δυτικό χωριό όπου ο μύθος πρώτα, λέχθηκε είναι ένα μοναστήρι. Οι καλόγεροι εκεί εκτελούν ένα μάθημα που τους ανατέθηκε από τον Θεό όταν ο κόσμος δημιουργήθηκε - λύνοντας το προαναφερθέν πρόβλημα με 3 χρυσούς στήλους και 64 χρυσούς δίσκους. Την στιγμή που θα ολοκληρώσουν το μάθημα, ο κόσμος θα καταρρεύσει και θα γίνει σκόνη. Καθώς η προφανής λύση στο πρόβλημα με τους n δίσκους θέλει, 2^n - 1 κινήσεις, δεν χρειάζεται να χάσουμε τον ύπνο μας με αυτήν την πιθανότητα. Ο αριθμός 2^{64} είναι βολικά μεγάλος.

Το σχετικό σχέδιο για την λύση του προβλήματος είναι Hanoi (N,A,B, C, Moves). Είναι αληθές αν το Moves είναι η συχνότητα των κινήσεων, για την μεταφορά του πύργου των N δίσκων από τη στήλη A στην στήλη B χρησιμοποιώντας την στήλη C ως βοηθητική. Αυτή είναι μια επέκταση σε συνηθισμένα προβλήματα, που δεν υπολογίζουν την συχνότητα των κινήσεων αλλά μάλλον εκτελούν αυτές. Η αντιπροσώπευση των κινήσεων χρησιμοποιούν έναν δυαδικό functor to, γραμμένο σαν ένα χειριστή εκτύπωσης. Ο όρος X to Y δηλώνει ότι ο δίσκος που βρίσκεται στην κορυφή της στήλης X μεταφέρεται στην στήλη Y. Το πρόγραμμα για την λύση του προβλήματος δίνεται στο πρόγραμμα 10.30.

Το δηλωτικό διάβασμα στο κέντρο της λύσης, ο επαναληπτικός κανόνας στο πρόγραμμα 10.30 είναι "το Moves είναι η συχνότητα των κινήσεων των s(N) δίσκων από τον στήλο A στον στήλο B χρησιμοποιώντας τον C στήλο ως βοηθητικό, αν Ms1 είναι η λύση για την μεταφορά των N στήλων από το A στο C χρησιμοποιώντας το B, το Ms2 είναι η λύση για την μεταφορά N στήλων από το C στο B χρησιμοποιώντας το A και το Moves είναι το αποτέλεσμα της πρόσθεσης του (A to B/Ms2) στο Ms1.

Η επανάληψη τελειώνει μετακινώντας έναν στύλο. Μια πιό απλή μέθοδος αλλά λιγότερο ακριβής, βασίζεται στο ότι η επανάληψη δεν μετακινεί δίσκους. Η κατάλληλη σταθερά είναι η :

Hanoi (O,A,B,C,[ ]).

Το τελευταίο παράδειγμα, αφορά τον τύπο Boolean. Ο τύπος Boolean είναι ένας όρος που ορίζεται ακολούθως :

Οι σταθερές true και false είναι τύπου Boolean. Αν τα X και Y είναι τύπου Boolean θα είναι επίσης τα X∨Y, X∧Y, και ¬X, όπου ∨ και ∧ είναι δυαδικοί τελεστές ενδοδιάταξης για την διάζευξη και σύζευξη αντιστοιχως, και το ¬ είναι ένα πρόθεμα που χρησιμοποιούμε για την άρνηση.

Ο τύπος Boolean F είναι αληθής εάν

F = "true"
F = X∧Y, and both X and Y are true
F = X∨Y, and either X or Y (or both) are true
F = ¬X, and X is false.

Ο τύπος Boolean formula είναι ψεύδης εάν :

F = "false"
F = X∧Y, and either X or Y (or both) are false
F = X∨Y, and either X or Y (or both) are false
F = ¬X, and X is true.

satisfiable (Formula) ← Υπάρχει ένα αληθές στιγμιότυπο της Boolean έκφρασης Formula.

satisfiable(true).
satisfiable (X∧Y) ←satisfiable(X), satisfiable (Y).
satisfiable(X∨Y) ←satisfiable(X).
satisfiable (X∨Y← satisfiable (Y).
satisfilable (¬X) ← invalid (X).

invalid(Formula) ← Υπάρχει ένα ψευδές στιγμιότυπο της Boolean έκφρασης Formula.

invalid(false).
invalid (X∨Y) ←invalid(X), invalid(Y).
invalid(X∧Y) ← invalid(X).
invalid (X∧Y← invalid(Y).
invalid(¬Y) ← satisfiable (Y).

Πρόγραμμα 10.31 Ικανοποιησιμότητα του τύπου Boolean

11

Απλά παραδείγματα σε Arity Prolog

11.1 Εισαγωγή

Στο κεφάλαιο αυτό δίνουμε ορισμένα απλά παραδείγματα εφαρμογής της Arity Prolog. Στόχος μας είναι να αναδείξουμε κυρίως τα λογικά χαρακτηριστικά της Prolog. Μεταξύ των παραδειγμάτων όμως θα δούμε και μερικά θέματα που θα αφορούν μη λογικά χαρακτηριστικά της Prolog.

11.2 Οικογενειακό δένδρο

Το πρόγραμμα αυτό αποτυπώνει τις σχέσεις συγγένειας των μελών μιας οικογένειας. Αφού ορίσαμε σαν γεγονότα τις σχέσεις "γονέων-παιδιών", ορίσαμε κανόνες που καθορίζουν τις σχέσεις συγγένειας για τα αδέλφια, τα ξαδέλφια, τον πατέρα, τη μητέρα, τους θείους, τις θείες, τους παπούδες και τις γιαγιάδες.

Το πρόβλημα που εμφανίζεται αφορά τις πολλαπλές απαντήσεις που δίνει το πρόγραμμα όταν ζητάγαμε τη συγγένεια μεταξύ δυο προσώπων.

Προσθέτοντας την εντολή "X≠Y" στον κανόνα "brothers(X,Y):-parents(X,Z,W), parents(Y,Z,W)." αποφεύγουμε τις απαντήσεις της μορφής "X=a,Y=a". Δεν καταφέρνουμε όμως να αποφύγουμε τις απαντήσεις της μορφής "X=a,Y=b" και "X=b,Y=a".

Χρησιμοποιήσαμε την εντολή "!" (cut) για να αποφύγουμε τις πολλαπλές απαντήσεις στις σχέσεις "θείοι" και "θείες".

Εξάλλου στο πρόγραμμα που ακολουθεί θα βρείτε διάφορα τεχνάσματα και τρόπους που αφορούν στην εκτύπωση δεδομένων στην οθόνη ή στην αποθήκευσή τους σε αρχείο, τρόπους γιά να συλλέγετε σε μιά λίστα όλες τις λύσεις, κ.λ.π.

Μελετείστε το πρόγραμμα αυτό, καθώς και άλλα προγράμματα γιά να μπορέσετε να πραγματοποιήσετε κάπως πιό πρακτικά και χρήσιμα προγράμματα σε Prolog.

```
/*****************************************************************
*******/
  help:-cls,
       write('*-----------------------------------------------------*'),nl,
       write('* File: FAMILY2.ARI      Author: Themis Panayiotopoulos *'),nl,
       write('* This is a PROLOG program which contains information   *'),nl,
       write('* about children and their parents, and finds family    *'),nl,
       write('* relations among relatives.                            *'),nl,
       write('*-----------------------------------------------------*'),nl,
       write('* Possible Queries :                                    *'),nl,
       write('*   parents(Child,Father,Mother)                        *'),nl,
       write('*   brothers(X,Y)                                       *'),nl,
       write('*   cousins(X,Y)                                        *'),nl,
       write('*   uncle(X,Y)       (Y is uncle of X)                  *'),nl,
       write('*   aunt(X,Y)        (Y is aunt of X)                   *'),nl,
       write('*   grandfather(X,Y)  (Y is grandfather of X)           *'),nl,
       write('*   grandmother(X,Y)  (Y is grandmother of X)           *'),nl,
       write('*   male(X)          (X is male)                        *'),nl,
       write('*   female(X)        (X is female)                      *'),nl,
       write('*   married(Husband,Wife)                               *'),nl,
       write('*-----------------------------------------------------*'),nl,
       write('Press any key ...'),get0_noecho(_),cls,
       write('* Special Queries :                                     *'),nl,
       write('*   ppar1. (to write to file parents.txt all the        *'),nl,
       write('*          triples : Child - Father - Mother).          *'),nl,
       write('*   pbro1. (To write to file brother1.txt all the       *'),nl,
       write('*          brothers).                                   *'),nl,
       write('*   pbro2. (To write to file brother2.txt all the       *'),nl,
       write('*          brothers, once for each pair).               *'),nl,
       write('*   pcou1. (To write to file cousins1.txt all the       *'),nl,
       write('*          cousins).                                    *'),nl,
       write('*   pcou2. (To write to file cousins2.txt all the       *'),nl,
       write('*          cousins, once for each pair).                *'),nl,
       write('*-----------------------------------------------------*'),nl.

  parents(pavlos,christos,zoi).
  parents(kostas,christos,zoi).
  parents(christos,grandfather_kostas,grandmother_gianna).
  parents(zoi,grandfather_pavlos,grandmother_kalliopi).
  parents(c_lalis,u_kostas,a_dina).
  parents(c_thanasis,u_kostas,a_dina).
  parents(c_popi,u_kostas,a_dina).
  parents(u_kostas,grandfather_pavlos,grandmother_kalliopi).
  parents(c_boubis,u_thanasis,a_marika).
  parents(c_helen,u_thanasis,a_marika).
  parents(u_thanasis,grandfather_pavlos,grandmother_kalliopi).
  parents(c_kostas,u_dionisis,a_lemonia).
  parents(c_kalli,u_dionisis,a_lemonia).
  parents(a_lemonia,grandfather_pavlos,grandmother_kalliopi).
  parents(c_maria,u_nikos,a_sofia).
```

```
parents(a_sofia,grandfather_pavlos,grandmother_kalliopi).
parents(c_eleni,u_christos,a_sotiria).
parents(a_sotiria,grandfather_pavlos,grandmother_kalliopi).
parents(c_pavlakis,u_leonidas,a_roula).
parents(c_popitsa,u_leonidas,a_roula).
parents(c_daria,u_leonidas,a_roula).
parents(u_leonidas,grandfather_pavlos,grandmother_kalliopi).
parents(c_dina,u_giannis,a_vassiliki).
parents(c_giorgos,u_giannis,a_vassiliki).
parents(a_vassiliki,grandfather_kostas,grandmother_gianna).
parents(c_vagelis,u_stelios,a_rina).
parents(a_rina,grandfather_kostas,grandmother_gianna).
parents(c_gianna,u_nikos1,a_asimo).
parents(c_konstantina,u_nikos1,a_asimo).
parents(u_nikos1,grandfather_kostas,grandmother_gianna).
parents(grandfather_pavlos,grandgrandfather_thanasis,
      grandgrandmother_sotiria).
parents(granduncle_miltiadis,grandgrandfather_thanasis,
      grandgrandmother_sotiria).
parents(granduncle_nikos,grandgrandfather_thanasis,
      grandgrandmother_sotiria).
parents(secondaunt_sofia,granduncle_nikos,grandaunt_nikena).
parents(secondunclе_panayiotis,granduncle_miltiadis,
      grandaunt_glykeria).
parents(secondcousin_miltiadis,secondunclе_panayiotis,
      secondaunt_asimina).
parents(secondcousin_kostas,secondunclе_sotiris,
      secondaunt_sofia).
parents(secondcousin_varvara,secondunclе_sotiris,
      secondaunt_sofia).
parents(secondcousin_nikos,secondunclе_sotiris,
      secondaunt_sofia).

brothers(X,Y):- parents(X,Z,W),parents(Y,Z,W),X\=Y.

male(Y):- parents(P,Y,W).

female(Y):- parents(P,W,Y).

married(X,Y):- parents(P,X,Y),!.

uncle(X,Y):- parents(X,Z,W),brothers(Z,Y),male(Y).
uncle(X,Y):- parents(X,Z,W),brothers(W,Y),male(Y).
uncle(X,Y):- parents(X,Z,W),brothers(Z,Q),married(Y,Q).
uncle(X,Y):- parents(X,Z,W),brothers(W,Q),married(Y,Q).

aunt(X,Y):- parents(X,Z,W),brothers(Z,Q),married(Q,Y).
aunt(X,Y):- parents(X,Z,W),brothers(W,Q),married(Q,Y).
aunt(X,Y):- parents(X,Z,W),brothers(Z,Y),female(Y).
aunt(X,Y):- parents(X,Z,W),brothers(W,Y),female(Y).
```

```
  grandfather(X,Y):- parents(X,A,B),parents(A,Y,Z).
  grandfather(X,Y):- parents(X,A,B),parents(B,Y,Z).

  grandmother(X,Y):- parents(X,A,B),parents(A,Z,Y).
  grandmother(X,Y):- parents(X,A,B),parents(B,Z,Y).

  cousins(X,Y):- parents(X,A,B),parents(Y,C,D),brothers(A,C).
  cousins(X,Y):- parents(X,A,B),parents(Y,C,D),brothers(A,D).
  cousins(X,Y):- parents(X,A,B),parents(Y,C,D),brothers(B,C).
  cousins(X,Y):- parents(X,A,B),parents(Y,C,D),brothers(B,D).

  secondcousins(X,Y):- parents(X,A,B),parents(Y,C,D),cousins(A,C).
  secondcousins(X,Y):- parents(X,A,B),parents(Y,C,D),cousins(A,D).
  secondcousins(X,Y):- parents(X,A,B),parents(Y,C,D),cousins(B,C).
  secondcousins(X,Y):- parents(X,A,B),parents(Y,C,D),cousins(B,D).

  kouniadoi(X,S):- married(S,R),brothers(X,R),male(R).
  kouniadoi(X,A):- married(N,A),brothers(X,N),female(A).
  kouniadoi(Z,D):- married(D,L),brothers(Z,L),male(D).
  kouniadoi(Z,R):- married(L,R),brothers(L,Z),female(R).

  sinifades(Z,A):- married(X,Z),married(N,A),brothers(X,N),
                   female(Z),female(A).

  batsanakides(Z,S):- married(X,Z),married(S,R),brothers(X,R),
                      female(Z),male(S).

  batsanakides(X,N):- married(X,Z),married(N,S),brothers(Z,S),
                      male(X),male(N).

  batsanakides(X,D):- married(X,Z),married(K,D),brothers(K,Z),
                      male(X),female(D).

/*****************************************************************
*****/
  ppar1:-create(H,'parents.txt'),
         bagof(X,queryparent(X),L),
         printheader(H,parents),
         printlist(H,L,1),
         printfooter(H),
         close(H),!.
  queryparent(parents(X,Y,Z)) :- parents(X,Y,Z).

/*****************************************************************
***/
  pbro1:-create(H,'brother1.txt'),
         bagof(X,qbro1(X),L),
         printheader(H,brothers),
         printlist(H,L,1),
```

```
        printfooter(H),
        close(H).
qbro1(brothers(X,Y)) :- brothers(X,Y).

pbro2 :- findbrothers,printbrothers,deletebrothers,!.

findbrothers:- brothers(X,Y), not br(Y,X), assert(br(X,Y)), fail.
findbrothers.

printbrothers:-create(H,'brother2.txt'),
        bagof(X,qbro2(X),L),
        printheader(H,brothers),
        printlist(H,L,1),
        printfooter(H),
        close(H).
qbro2(brothers(X,Y)) :- br(X,Y).

deletebrothers:- br(X,Y),retract(br(X,Y)),fail.
deletebrothers.

/*************************************************************************
*****/
pcou1:-create(H,'cousins1.txt'),
        bagof(X,qcou1(X),L),
        printheader(H,cousins),
        printlist(H,L,1),
        printfooter(H),
        close(H),!.
qcou1(cousins(X,Y)) :- cousins(X,Y).

pcou2 :- findcousins,printcousins,deletecousins,!.

findcousins:- cousins(X,Y), not co(Y,X), assert(co(X,Y)), fail.
findcousins.

printcousins:-create(H,'cousins2.txt'),
        bagof(X,qcou2(X),L),
        printheader(H,brothers),
        printlist(H,L,1),
        printfooter(H),
        close(H).
qcou2(cousins(X,Y)) :- co(X,Y).

deletecousins:- co(X,Y),retract(co(X,Y)),fail.
deletecousins.

/*************************************************************************
*****/
printheader(H,X):-write(H,' Query : ?- '),write(H,X),nl(H),
            write(H,'==================='),nl(H),
```

```
            write(H,' A n s w e r s  :'),nl(H),
            write(H,'-------------------------------------------'),
            nl(H).

 printfooter(H):- write(H,'-------------------------------------------'),
            nl(H).

 printlist(H,[],N).
 printlist(H,[X|L],N):-write(H,N),write(H,'. '),write(H,X),nl(H),
                write(N),write('. '),write(X),nl,
                M is N+1,printlist(H,L,M).

/******************* E N D  O F  P R O G R A M  ****************/
```

11.3 Αναγνώριση όρων

11.3.1 Πρόγραμμα που αναγνωρίζει αν ένας όρος είναι σταθερά (integer ή atom), μεταβλητή ή string.

```
subterm(X):-var(X),write('<Variable>'),write(X),!.
subterm(X):-atom(X),write('<Constant>:'),write(X),!.
subterm(X):-integer(X),write('<Constant>:'),write(X),!.
subterm(X):-string(X),write('<string>:'),write(X),!.
```

Στο παραπάνω πρόγραμμα χρησιμοποιούμε τα κατηγορήματα διαχείρησης όρων atom(+X),integer(+X),var(+X) και string(+X).

11.3.2 Πρόγραμμα που αναγνωρίζει αν ο όρος είναι functor.

```
member2([X|_],[]).
member2([X|L],[L1]):-L1=L.
subterm(X):-functor(X,M,N),X=..L,[O|P]=L,member2([O|P],[L1]),list(L1).
list([X|L]):-subterm(X),list(L).
list([]):-write('<EndOfFunctor>').
```

Στο προηγούμενο πρόγραμμα χρησιμοποιούμε το κατηγόρημα διαχείρησης όρων functor(X,M,N) όπου X είναι ο FUNCTOR ,M είναι το όνομα του Functor και N ο αριθμός των παραμέτρων του. Μετατρέπουμε τον Functor σε λίστα με το κατηγόρημα UNIV (X=..L) και μέσω του κατηγορήματος member2 δημιουργούμε τη λίστα L1 που είναι η L χωρίς το πρώτο στοιχείο. Με το κατηγόρημα list εξετάζουμε τους όρους της λίστας L1.

11.3.3 Πρόγραμμα που αναγνωρίζει αν ο όρος είναι list.

```
subterm([X|L]):-subterm(X),lst(L),!.
lst([X|L]):-subterm(X),lst(L),!.
lst([]):-write('<EndOfList>').
```

Στο παραπάνω πρόγραμμα εξετάζουμε τους όρους μιας λίστας ,εως ότου καταλήξουμε στη κενή λίστα.

11.3.4 Πρόγραμμα που αναγνωρίζει όρους και τους εκτυπώνει κατάλληλα

```
member2([X|_],[]).
member2([X|L],[L1]):-L1=L.
subterm(X):-var(X),write('<Variable>'),write(X),!.
subterm(X):-atom(X),write('<Constant>:'),write(X),!.
subterm(X):-integer(X),write('<Constant>:'),write(X),!.
subterm(X):-string(X),write('<string>:'),write(X),!.
subterm([X|L]):-write('<list>:'),subterm(X),nl,lst(L),!.
lst([X|L]):-write('      :'),subterm(X),nl,lst(L),!.
lst([]):-write('<EndOfList>').
subterm(X):-functor(X,M,N),X=..L,write('<Functor>:'),write(M),nl,
          write('       '),write('<arguments>:'),
          [O|P]=L,member2([O|P],[L1]),list(L1).
list([X|L]):-subterm(X),nl,write('          :'),list(L).
list([]):-write('<EndOfFunctor>').

printsubterm:-create(H,'subterm.txt'),printheader(H,subterm),
                  printfooter(H),close(H),!.

printheader(H,X):-write(H,' Query:? '),write(H,X),nl(H),
               write(H,'Answers'),nl(H),write(H,'    '),nl(H).

printfooter(H):-write('Γράψε τον όρο'),nl,read(X),
             write(H,'Ο αρχικός όρος είναι'),
                   write(H,X),nl(H),tell_h(H),
             subterm(X),told,subterm(X).
```

Εκτέλεση του προγράμματος :

```
 Query:? subterm
Answers

Ο αρχικός όρος είναι  [5,t,$8u7y12jz$,f(a,g(c,d)),[k,l,m]]
<list>:<Constant>:5
     :<Constant>:t
     :<string>:8u7y12jz
     :<Functor>:f
       <arguments>:<Constant>:a
          :<Functor>:g
       <arguments>:<Constant>:c
          :<Constant>:d
          :<EndOfFunctor>
          :<EndOfFunctor>
     :<list>:<Constant>:k
     :<Constant>:l
     :<Constant>:m
```

<EndOfList>
<EndOfList>

11.4 Φυσικοί και άλλοι αριθμοί

Οι ορισμοί γιά το μικρότερο ή ίσο, την πρόσθεση και τον πολλαπλασιασμό στους φυσικούς είναι :

LESSEQUAL
$\forall X \in N \Rightarrow 0 \leq X.$
$\forall X \in N$ και $\forall Y \in N$ τέτοια ώστε $X \leq Y \Rightarrow s(X) \leq s(Y)$

PLUS
$\forall X \in N^* \Rightarrow X+0=X$
$\forall X \in N$ και $\forall Y \in N$ ισχύει $s(X)+Y=s(X+Y)$

TIMES
$\forall X \in N \Rightarrow X*0=0$
$\forall X \in N$ και $\forall Y \in N$ τέτοια ώστε $X*Y=W \Rightarrow s(X)*Y=Z$ όπου $Z=W+Y$

11.4.1 Λύση με χρήση του συναρτησιακού s(X), δηλαδή successor(X)

LESSEQUAL

```
(1) lessequal(0,X).
(2) lessequal(s(X),s(Y)):-lessequal(X,Y).
```

Στη σχέση (1) δίνουμε σαν γεγονός ότι κάθε φυσικός X είναι μεγαλύτερος ή ίσος από το 0.Στη σχέση (2) λέμε ότι ο επόμενος του X είναι μεγαλύτερος ή ίσος του επόμενου του Y αν ο X είναι μεγαλύτερος ή ίσος του Y.

PLUS

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

Στη σχέση (1) ορίζουμε σαν άθροισμα οποιουδήποτε φυσικού X με το 0 το X ,ενώ στη σχέση (2) θέτουμε σαν ικανή συνθήκη του να είναι το άθροισμα, του Y και του επόμενου του X , ο επόμενος του Z ,το να είναι άθροισμα του X και του Y ο Z.

TIMES

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

Στη σχέση (1) δίνουμε σαν γεγονός ότι το γινόμενο του X με το 0 είναι 0 ενώ στη σχέση (2) λέμε ότι το γινομένο του επόμενου του X με τον Y είναι Z αν το γινόμενο του X με τον Y είναι W, όπου W η διαφορά του Z με τον Y.

11.4.2 Ισοδύναμοι ορισμοί χωρίς τη χρήση του συναρτησιακού.

ΠΡΟΣΘΕΣΗ

Στο παρακάτω πρόγραμμα πραγματοποιούμε τη πρόσθεση δυο αριθμών X και Y χρησιμοποιώντας μια σχέση με πέντε ορίσματα. Το X είναι ο πρώτος προσθετέος, το Y ο δεύτερος, το X2 είναι μια βοηθητική μεταβλητή που παίρνει αρχική τιμή 0 και κάθε φορά που εφαρμόζεται ο κανόνας προστίθεται στην προηγούμενη τιμή του συν ένα,το Y1 είναι σταθερά και ισούται με την μονάδα κατά τη διάρκεια του προγράμματος και τέλος το Z είναι το άθροισμα των X και Y.

Το πρόγραμμα λειτουργεί ως εξής:

Έστω ότι του ζητάμε να υπολογίσει το άθροισμα των X και Y. Θέτει X2=1,X3=X+1 και όσο το X2 είναι διάφορο του Y,επαναλαμβάνει τη διαδικασία. (Ουσιαστικά προσθέτει στο X τόσες μονάδες ,όσες το άθροισμά τους να δίνει το Y.)

```
plus(X,Y,X1,Y1,Z):-X2 is X1+1,X3 is Y1+X,X2=\=Y,plus(X3,Y,X2,Y1,Z).
plus(X,Y,X1,Y1,Z):-X2 is X1+1,Z is X+Y1,X2=Y.
```

ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ

Στο παρακάτω πρόγραμμα πραγματοποιούμε το πολλαπλασιασμό δύο αριθμών X και Y χρησιμοποιώντας μια σχέση με πέντε ορίσματα. Έστω Z το γινόμενο του X με το Y,η μεταβλητή X1 παίρνει αρχική τιμή 0 και κάθε φορά που χρησιμοποιείται ο κανόνας, η μεταβλητή X1 αυξάνεται κατά μια μονάδα.Στη μεταβλητή W δίνεται αρχική τιμή η τιμή του X και το W παραμένει σταθερό κατά τη διάρκεια του προγράμματος.

Το πρόγραμμα λειτουργεί ως εξής:

Έστω ότι ζητάμε να υπολογιστεί το γινόμενο των X και Y.Θέτει X2=1,X3=X∗ 1

και όσο το X2 είναι διάφορο του Y,επαναλαμβάνει τη διαδικασία.(Ουσιαστικά πολλαπλασιάζει το X με τόσες μονάδες, όσες το άθροισμά τους να δίνει το Y.)

```
times(X,0,X1,W,0).
times(X,Y,X1,W,Z):-X2 is X1+1,X3 is W*X2,X2=\=Y,times(X3,Y,X2,W,Z).
times(X,Y,X1,W,Z):-X2 is X1+1,Z is W*X2,X2=Y.
```

11.4.3. Ορισμός της αφαίρεσης :

Μπορούμε να χρησιμοποιήσουμε το PLUS για να υπολογίσουμε

(α) το άθροισμα Z δύο αριθμών X και Y ως εξής :

```
PLUS(X,Y,Z)
```

π.χ. PLUS(3,2,Z) επιστρέφει Z=5

(β) τη διαφορά Z δύο αριθμών X και Y ως εξής :

```
PLUS(X,Z,Y)
```

π.χ. PLUS(3,Z,5) επιστρέφει Z=2

11.4.4 Πρόγραμμα για το εκθετικό exp(X,Y,Z) :

```
n(0).
n(s(X)):-n(X).
times(0,X,0).
times(s(X),Y,Z):-times(X,Y,W),plus(W,Y,Z).
exp(X,0,s(0)).
exp(s(0),Y,s(0)).
exp(X,s(Y),Z):-exp(X,Y,Z1),times(Z1,X,Z).
```

Με το παραπάνω πρόγραμμα υπολογίζουμε τη τιμή Z που παίρνει το X αν το υψώσουμε στη Y. Το πρόγραμμα χρησιμοποιεί τους φυσικούς αριθμούς και τον ορισμό του πολλαπλασιασμού σε αυτούς. Αρχικά δίνουμε σαν γεγονότα οτι το X στη μηδενική ισούται με τη μονάδα και η μονάδα σε οποιαδήποτε δύναμη ισούται με τη μονάδα.

Ο κανόνας λέει οτι το X υψωμένο στην s(Y),δηλαδή στη (Y+1),ισούται με Z,αν το X υψωμένο στην Y ισούται Z1,όπου Z είναι το γινόμενο του Z1 με τοX.

11.4.5 Αντίστοιχοι ορισμοί σε άλλο σύνολο :

Αναφερόμαστε στο σύνολο των διανυσμάτων που βρίσκονται σε κάποιο Ευκλείδειο χώρο τα οποία αναπαριστούμε εδώ με μια λίστα που περιέχει τις συντεταγμένες τους. Ορίζουμε λοιπόν σ' αυτό το σύνολο τις πράξεις της πρόσθεσης και του εσωτερικού γινομένου μεταξύ δυο διανυσμάτων,καθώς επίσης και τον πολλαπλασιασμό διανύσματος με πραγματικό αριθμό,τον βαθμωτό πολλαπλασιασμό. Επίσης ορίζουμε και υπολογίζουμε μέσω ενός προγράμματος το μέτρο ενός διανύσματος και μέσω του μέτρου των διανυσμάτων βρίσκουμε αν κάποιο διάνυσμα είναι μικρότερο ή ίσο από κάποιο άλλο.

i)Η **Πρόσθεση** δυο διανυσμάτων γίνεται προσθέτοντας τις αντίστοιχες συντεταγμένες τους.Στο πρόγραμμα επιτυγχάνεται αφού έχουμε θεωρήσει τα διανύσματα ως λίστες και προσθέσουμε τα αντίστοιχα στοιχεία τους.

*ΟΡΙΣΜΟΣ:*Για κάθε $x=(x_1,x_2,...,x_n)$ και $y=(y_1,y_2,...,y_n)$ με $x,y \in \Re^n$ ισχύει

$$x + y =(x_1+y_1,x_2+y_2,...,x_n+y_n) \text{ και } (x + y) \in \Re^n .$$

ΠΡΟΓΡΑΜΜΑ:

```
prosthesi([ ],[ ][ ]):-!.
prosthesi([A|B],[X|Y],[Z|W]):-Z is A+X,prosthesi(B,Y,W).
```

ii)Ο **Βαθμωτός πολλαπλασιασμός** μεταξύ ενός διανύσματος και ενός πραγματικού αριθμού γίνεται πολλαπλασιάζοντας τον πραγματικό αριθμό με κάθε στοιχείο της λίστας με την οποία συμβολίζουμε το διάνυσμα.

*ΟΡΙΣΜΟΣ:*Για κάθε $x=(x_1,x_2,...,x_n)\in \Re^n$ και $\forall a\in \Re$ το διάνυσμα
ax=(ax1,ax2,...,axn) είναι το διάνυσμα που προκύπτει όταν πολλαπλασιάσουμε το a με το x.

ΠΡΟΓΡΑΜΜΑ:
```
poll(X,[],[]):-!.
poll(X,[A|B],[Z|W]):-Z is A*X,poll(X,B,W).
```

iii)**Εσωτερικό γινόμενο** μεταξύ δυο διανυσμάτων.

*ΟΡΙΣΜΟΣ:*Για $x=(x_1,x_2,...,x_n)$ και $y=(y_1,y_2,...,y_n)\in \Re^n$ το εσωτερικό γινόμενο είναι ο αριθμός $x\cdot y=\sum x_i\cdot y_i$ για i=1,...,n.

ΠΡΟΓΡΑΜΜΑ:
```
inpro([ ],[ ],0).
inpro([X|Xs],[Y|Ys],Z):-inpro(Xs,Ys,Zs),Z is Zs+(X*Y).
```

iv)Παρακάτω ορίζουμε το **μέτρο** και την σχέση **μικρότερο ίσο** στο σύνολο των διανυσμάτων

ΟΡΙΣΜΟΣ: Το μέτρο ενός διανύσματος $x=(x_1,x_2,...,x_n)\in \Re^n$ δίνεται από τον τύπο

$$\|x\|=\sqrt{\sum x_i^{\,2}}$$

Ορίζουμε τώρα ότι ένα διάνυσμα x είναι μικρότερο ή ίσο από ένα διάνυσμα y αν ισχύει:

$$\|x\| \le \|y\|$$

ΠΡΟΓΡΑΜΜΑ:
```
metro([X|Y],M):-sum2list([X|Y],N),M is N^(1/2).
sum2list([ ],0).
sum2list([X|L],N):-sum2list(L,M),N is M+(X^2).
lsq([A|B],[C|D]):-metro([A|B],X),metro([C|D],Y),X=<Y.
lsq(0,X).
```

11.5 Παραγοντικό - Ackermann - Fibbonacci - ΜΚΔ - ΕΚΠ

11.5.1 Επαγωγικός ορισμός του παραγοντικού:

$0!=1$
$(n+1)!=(n+1)*n!$

Πρόγραμμα υλοποίησης παραγοντικού:

```
n(0).
n(s(X)):-n(X).
plus(0,X,X):-n(X).
plus(s(X),Y,s(Z)):-plus(X,Y,Z).
times(0,X,0).
times(s(X),Y,Z):-times(X,Y,W),plus(W,Y,Z).
factorial(0,s(0)).
factorial(s(N),F):-factorial(N,X),times(s(N),X,F).
```

Λειτουργία του προγράμματος:

Το πρόγραμμα για να υπολογίσει το παραγοντικό F ενός αριθμού N ,

factorial(N,F)

αρχικά υπολογίζει το γινόμενο του N με το (N-1)! το οποίο έχει βρει μέσω του επαγωγικού ορισμού του παραγοντικού.

Για να υπολογίσει όμως το γινόμενο Z δύο αριθμών X και Y ,

times(X,Y,Z)

υπολογίζει το γινόμενο W του X -1 και Y ,επαγωγικά, και Z δίνει το άθροισμα των W και Y.

Επίσης για να υπολογίσει το άθροισμα Z δύο αριθμών X και Y,

plus(X,Y,Z)

υπολογίζει το άθροισμα Z των X και Y, επαγωγικά, μέσω του επαγωγικού ορισμού των φυσικών αριθμών.

11.5.2. Επαγωγικός ορισμός της συνάρτησης του ACKERMANN:

a(0,n)=n+1
a(m,0)=a(m-1,1)
a(m,n)=a(m-1,a(m,n-1))

Πρόγραμμα υλοποίησης της συνάρτησης του ACKERMANN:

```
ack(0,N,X):-X is N+1.
ack(M,0,X):-K is M-1,ack(K,0,X).
ack(M,N,X):-K is N-1,L is M-1,ack(M,K,Y),ack(L,Y,X).
```

Λειτουργία του προγράμματος:

Η συνάρτηση του ACKERMANN είναι μια συνάρτηση δύο μεταβλητών ,και είναι από το $N\times N\rightarrow N$. Επίσης είναι τρίκλαδη και λειτουργεί ως εξής :

(1) Έστω ότι ζητάμε να υπολογίσει τη τιμή του ACK(0,N) ,τότε θα μας επιστρέψει το (N+1),δηλαδή τον επόμενο του N.

(2) Αν ζητήσουμε να υπολογίσει τη τιμή του ACK(N,0), θα υπολογίσει τη τιμή του ACK((N-1),0), όμως για να υπολογίσει αυτή, θα πρέπει να υπολογίσει τη τιμή του ACK((N-2),0) και ομοίως μέχρι να πρέπει να υπολογίσει τη τιμή του ACK((N-N),0) το οποίο είναι το ACK(0,0) που από το (1) είναι 1. Έτσι αντιστρέφοντας την ως τώρα πορεία εξισώνει τη γνωστή πλέον τιμή του ACK(0,0) με το ACK(1,0), μετά αυτό με το ACK(2,0), όσπου να φτάσει στη τιμή του ACK(N,0) που είναι και η ζητούμενη. Αρα ζητώντας τη τιμή ACK(N,0) για οποιοδήποτε N επιστρέφει πάντα τη τιμή 1.

(3) Αν ζητήσουμε τη τιμή X του ACK(M,N), υπολογίζει τη τιμή Y του ACK(M,(N-1)) και τότε τη τιμή X του ACK((M-1),Y). Ο ορισμός είναι αναδρομικός, δηλαδή για να υπολογίσει τη τιμή Y του ACK(M,(N-1)) θα χπησιμοποιήσει το (3), ομοίως και για να υπολογίσει τη τιμή X του ACK((M-1),Y).

11.5.3 Επαγωγικός ορισμός της ακολουθίας FIBONACCI :

f(n) = 1 για n=1
f(n) = 1 για n=1
f(n) =f(n-1)+f(n-2)

Υλοποίηση του προγράμματος :

```
fib(0,1).
fib(1,1).
fib(N,X):-M1 is N-1,M2 is N-2, fib(M1,X1), fib(M2,X2), X is X1+X2.
```

11.5.4 Αλγόριθμος του Ευκλείδη για τον Μέγιστο Κοινό Διαιρέτη δυο αριθμών :

Εστω ότι θέλουμε να υπολογίσουμε τον Μ.Κ.Δ. των X και Y (έστω Z).
Αν X≥Y(διαφορετικά αντιμεταθέτουμε τις τιμές των X και Y), πάρε το υπόλοιπο της ακέραιης διαίρεσης X δια Y (X mod Y).
Βάλε όπου νέο X =Y και όπου νέο Y=X mod Y.
(δηλαδή Μ.Κ.Δ.(X,Y)=Μ.Κ.Δ.(Y, X modY).
Συνέχισε έτσι, μέχρι να βρείς νέο Y=0.

Υλοποίηση του προγράμματος

```
mkd(X,Y,Z):-X>Y,W is X mod Y,W=\=0,mkd(Y,W,Z).
mkd(X,Y,Y):-X>Y,W is X mod Y,W=0.
```

11.5.5 Αλγόριθμος για το Ελάχιστο Κοινό Πολλαπλάσιο δυο αριθμών:

Εστω ότι θέλουμε να υπολογίσουμε το Ε.Κ.Π. τών X και Y(έστω Z). Χρησιμοποιούμε μια ακόμη μεταβλητή X1 η οποία κρατά την (αρχική) τιμή του X.
Εστω X1≥Y, (διαφορετικά αντιμεταθέτουμε τις τιμές τους), πάρε το υπόλοιπο W της διαίρεσης του X1προς Y. Βάλε όπου X1 το X1+X και συνέχισε έτσι μέχρι να βρείς υπόλοιπο 0.

Υλοποίηση του προγράμματος :

```
ekp(X,Y,X1,Z):-X>Y,W is X mod Y,W=\=0,X2 is X+X1,ekp(X2,Y,X1,Z).
ekp(X,Y,X1,X):-X>Y,W is X mod Y,W=0.
```

ΠΕΡΙΕΧΟΜΕΝΑ