«ΣΗΜΑΣΙΟΛΟΓΙΚΑ ΔΙΚΤΥΑ»

«Μελέτη του υποδείγματος αναπαράστασης της γνώσης και η υλοποίησή του με τη γλώσσα προγραμματισμού Common Lisp»

ΔΙΟΝΥΣΟΠΟΥΛΟΣ ΧΡΗΣΤΟΣ

15^{Ο} εξάμηνο 10796, xrhstosdion@hotmail.com

Επιβλέπων καθηγητής: Σταύρος Αδάμ

Λέξεις-Κλειδία: Common lisp, semantic nets, artificial intelligence, knowledge representation

ΠΕΡΙΛΗΨΗ

Τις τελευταίες δεκαετίες καλούμαστε να διαχειριστούμε ολοένα και μεγαλύτερους όγκους δεδομένων, όπου στην συνέχεια προσπαθούμε να τους διαχειριστούμε όσο πιο αποτελεσματικά μπορούμε. Αυτό διότι η γνώση που είναι κρυμμένη πίσω από αυτούς τους όγκους πληροφορίας κρίνεται εξαιρετικής σημασίας, μέσω αυτής βγάζουμε συμπεράσματα που μας βοηθούν στην επίλύση πολύπλοκων προβλημάτων. Έτσι σαν αποτέλεσμα βρισκόμαστε σε μια διαρκής αναζήτηση νέων τρόπων διαχείρισης της γνώσης αλλά και εξαγωγής συμπερασμάτων από αυτήν. Σκοπός της παρούσας διπλωματικής εργασίας είναι να αναδείξει τον τρόπο αναπαράστασης γνώσης που προσφέρουν τα σημασιολογικά δίκτυα, τα οποία δίνουν έμφαση στην εννοιολογική περιγραφή της πληροφορίας, κάνοντάς την έτσι αρκετά πιο κατανοητή.

Ένας επιπλέον σκοπός της παρούσας διπλωματικής εργασίας είναι να αναδείξει και μια γλώσσα προγραμματισμού διαφορετική σε σύγκριση με τις δημοφιλείς γλώσσες όπως η JAVA και η C++. Διότι κρύβεται μεγάλη αξία στη μάθησης μιας εναλλακτικής γλώσσας προγραμματισμού έναντι των συνηθισμένων. Η γλώσσα αυτή είναι η Common Lisp για την οποία θα δοθεί περιγραφική αναφορά στις δυνατότητές της και παρουσίαση κάποιων. Θα γίνει περιγραφή του συντακτικού, των συναρτήσεων και των Macro. Για τον προγραμματισμό σε μια γλώσσα προγραμματισμού χρειάζεται και ένα περιβάλλον με κάποια εργαλεία, τα οποία θα αναφερθούν, καθώς και η διαδικασία βήμα-βήμα για την εγκατάστασή τους. Επιπλέον θα δούμε κάποια από τα πλεονεκτήματα και μειονεκτήματα της γλώσσας αυτής, καθώς και κάποια συμπεράσματα από την γνωριμία αυτή. Τέλος θα επιχειρήσουμε να κατασκευάσουμε με την Clisp ένα σημασιολογικό δίκτυο.

Η παρακάτω διπλωματική εργασία με θέμα τα σημασιολογικά δίκτυα πραγματοποιήθηκε από τον φοιτητή Διονυσόπουλο Χρήστο, στα πλαίσια του πτυχιακού προγράμματος του τμήματος Μηχανικών Πληροφορικής Άρτας του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Ηπείρου.
Επιβλέπων καθηγητής ήταν ο κ. Σταύρος Αδάμ.

Υπογραφές

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Εδώ και χιλιάδες χρόνια ο άνθρωπος καταβάλλει τεράστια προσπάθεια ώστε να κατανοήσει από που, αλλά και πώς προέρχεται η νοημοσύνη του, χάριν της οποίας κατάφερε να ξεφύγει από την σκιά της απλής επιβίωσης και να κυριαρχήσει στον κόσμο του. Έτσι κατέληξε να κατανοεί, να προβλέπει και να χειρίζεται πράγματα πολύ μεγαλύτερα και πολύ πιο πολύπλοκα από την ίδια την νοημοσύνη του. Το πεδίο της τεχνητής νοημοσύνης, σπρώχνει τον άνθρωπο ακόμη πιο πέρα, επιχειρεί όχι μόνο να κατανοήσει αλλά και να *κατασκευάσει* νοήμονες οντότητες. Επίσης ας μη ξεχνάμε πως η τεχνητή νοημοσύνη μπορεί να έχει εφαρμογή σε οποιαδήποτε πεδίο της ανθρώπινης δραστηριότητας μιας και έχει τη δυνατότητα να συστηματοποιεί και να αυτοματοποιεί όλες τις διανοητικές εργασίες του ανθρώπου.

1.1 Ιστορία της τεχνητής νοημοσύνης

Η πρώτη εργασία που αναγνωρίζεται σήμερα ως τεχνητή νοημοσύνη χρονολογείται το 1943 από τους Warren McCulloch και Walter Pitts. Βασίστηκαν σε τρεις πηγές, στη βασική φυσιολογία και λειτουργία των νευρώνων του ανθρώπινου εγκεφάλου, σε μια ανάλυση της προτασιακής λογικής των Russell και Whitehead αλλά και στη θεωρία υπολογισμού του Turing. Πρότειναν ένα μοντέλο τεχνητών νευρώνων, το οποίο χαρακτήριζε τον κάθε νεύρωνα ως ενεργό η ανενεργό μεταβαίνοντας στην κάθε κατάσταση αποκρινόμενος ανάλογα την διέργεση από κάποιον επαρκή αριθμό γειτονικών νευρώνων. Αυτό που κατάφεραν να δείξουν ήταν πως οποιαδήποτε υπολογίσιμη συνάρτηση μπορούσε να υπολογιστεί από κάποιο δίκτυο συνδεδεμένων νευρώνων και οτι όλοι οι λογικοί σύνδεσμοι (AND, OR, NOT, κ.λπ.) μπορούσαν να υλοποιηθούν με απλές δικτυακές δομές. Επίσης υποστήριξαν πως κατάλληλα ορισμένα δίκτυα μπορούσαν να μαθαίνουν. Ακολούθησε ο Donald Hebb το 1949 με το βιβλίο του 'The organization of Behavior' όπου παρουσίασε έναν απλό κανόνα ενημέρωσης για την τροποποίηση των συνδετικών δυνάμεων μεταξύ νευρώνων. Ο κανόνας αυτός, σήμερα ονομάζεται μάθηση Hebb (Hebbian learning) και παραμένει μέχρι και σήμερα ένα σημαντικό μοντέλο.

Ο πρώτος υπολογιστής νευρωνικού δικτύου κατασκευάστηκε το 1951 από τον Marvin Minsky και τον Dean Edmonds, δύο μεταπτυχιακούς φοιτητές του μαθηματικού τμήματος του Princeton. Ονομαζόταν SNARC, χρησιμοποιούσε 3000 λυχνίες κενού και έναν μηχανισμό αυτόματου πιλότου από βομβαρδιστικό B-24 για να προσομοιώνει ενα δίκτυο 40 νευρώνων. Βέβαια η επιτροπή του Minsky για το Ph.D αμφέβαλλε για το αν μια τέτοια εργασία μπορούσε να θεωρηθεί μαθηματικά, αλλά όπως λέγεται ο von Neumann είπε 'Αν δεν είναι σήμερα, θα είναι κάποια μέρα.'

Ήταν όμως ο Alan Turing το 1950, εκείνος που διατύπωσε το πρώτο πλήρες όραμα για την τεχνητή νοημοσύνη, στο άρθρο του 'Computing Machinery and Intelligence' όπου παρουσίασε τη δοκιμασία Turing (την οποία θα περιγράψουμε παρακάτω), τη μηχανική μάθηση, τους γενετικούς αλγόριθμους και την ενισχυτική μάθηση [12].

1.2 Τι είναι η τεχνήτη νοημοσύνη

Υποστηρίξαμε ότι η τεχνητή νοημοσύνη είναι συναρπαστική όχι όμως τι ακριβώς ειναι. Ένας απλός και κατανοητός ορισμός δόθηκε από την Elaine Rich όπου ορίζει την τεχνητή νοημοσύνη ως την μελέτη που κάνουμε για τους υπολογιστές ώστε να κάνουν πράγματα, που μέχρι στιγμής, οι άνθρωποι κάνουν καλύτερα. Βέβαια ο παραπάνω ορισμός εκτός του ότι δεν είναι ευρέως αποδεκτός είναι και εφήμερος αφού αναφέρεται στη τρέχουσα κατάσταση της επιστήμης των υπολογιστών.
Έτσι παρακάτω παρουσιάζονται επιπλέον, οκτώ εγχειρίδια όπου επιχειρούν να ορίσουν την τεχνητή νοημοσύνη πιο περιγραφικά αλλά και ανάλογα με το είδος των διαδικασιών που παίρνουν μέρος.

Συστήματα που «σκέπτονται» σαν τον άνθρωπο	**Συστήματα που «σκέπτονται» ορθολογικά**
'Η συναρπαστική νέα προσπάθεια για να κάνουμε τουπς υπολογιστές να σκέπτονται . . . Μηχανές με νόηση, με την πλήρη και κυριολεκτική έννοια.' (Haugeland, 1985) 'Η αυτοματοποίηση των δραστηριοτήτων που συσχετίζουμε με την ανθρώπινη σκέψη, όπως η λήψη αποφάσεων, η επίλυση προβλημάτων, η μάθηση...' (Bellman, 1978)	'Η μελέτη των νοητικών ικανοτήτων με τη χρήση υπολογιστικών μοντέλων.' (Charniak και McDermott, 1985) 'Η μελέτη των υπολογιστικών εργασιών που μας δίνουν τη δυνατότητα να αντιλαμβανόμαστε, να συλλογιζόμαστε, και να ενεργούμε' (Winston, 1992)
Συστήματα που «ενεργούν» σαν τον άνθρωπο	**Συστήματα που «ενεργούν» ορθολογικά**
'Η τέχνη της δημιουργίας μηχανών που πραγματοποίουν λειτουργίες οι οποίες απαιτούν νοημοσύνη όταν πραγματοποιούνται από ανθρώπους.' (Kurzweil, 1990) 'Η μελέτη του πως μπορούμε να κάνουμε τους υπολογιστές να κάνουν πράγματα στα οποία, προς το παρόν, οι άνθρωποι είναι καλύτεροι,' (Rich and knight, 1991)	'Υπολογιστική νοημοσύνη είναι η μελέτη της σχεδίασης ευφυών πρακτόρων.' (Poole κ.α, 1998) 'Η τεχνητη νοημοσύνη ασχολείται με τη ευφυή συμπεριφορά των τεχνουργημάτων.' (Nilsson, 1998)

Οι ανωτέρω ορισμοί ενδιαφέρονται περισσότερο για τις διαδικασίες σκέψης και τη συλλογιστική, ενώ οι κάτω ασχολούνται με τη συμπεριφορά. Στα αριστερά έχουμε ορισμούς που μετρούν την επιτυχία με βάση την εγγύτητα προς τις ανθρώπινες επιδόσεις, ενώ δεξιά τη μετρούν σε σχέση με μια ιδανική έννοια νοημοσύνης, την ορθολογικότητα [9].

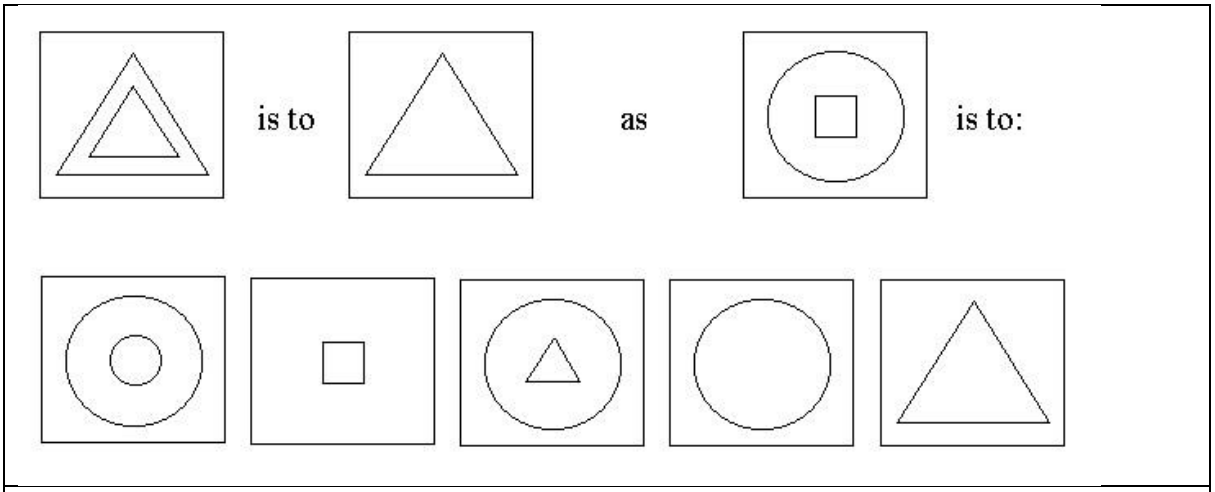
1.3 Η γέννηση της τεχνητής νοημοσύνης

Από το Princeton αποφοίτησε άλλη μια σημαντική φυσιογνωμία της τεχνητής νοημοσύνης, ο John McCarthy όπου τελικώς μετακόμισε στο Dartmouth College, τον επίσημο τόπο γέννησης του πεδίου. Ο McCarthy μαζί με τους Minsky, Claude Shannon και Nathaniel Rochester αποφάσισαν να οργανώσουν μια δίμηνη συνάντηση εργασίας στο Dartmouth όπου και προσκάλεσαν κάποιους Αμερικανούς ερευνητές που είχαν ενδιαφέρον για τη μελέτη της νοημοσύνης, τα νευρωνικά δίκτυα και τη θεωρία των αυτομάτων. Δύο ερευνητές από το Carnegie Tech έκλεψαν την παράσταση, ο Allen Newell και ο Herbert Simon παρουσίασαν το Logic Theorist (LT), ένα πρόγραμμα συλλογιστικής για το οποίο ο Simon είπε: 'Εφεύραμε ένα πρόγραμμα που να μπορεί να σκέπτεται μη αριθμητικά, επομένως λύσαμε το κλασικό πρόβλημα του δυισμού νου-σώματος.' Το πρόγραμμα μπορούσε να αποδείξει τα περισσότερα από τα θεωρήματα του Κεφαλαίου 2 του Principia Mathematica των Russel και Whitehead.
Στη συνάντηση συμφώνησαν να κρατήσουν την ονομασία που πρότεινε ο McCarthy για το πεδίο, η οποία φυσικά δεν ήταν άλλη από την '**Τεχνητή Νοημοσύνη**.' Κάπως έτσι κατάφεραν να έρθουν σε επαφή οι μεγαλύτερες φυσιογνωμίες του χώρου, οι οποίοι τα επόμενα 20 χρόνια μαζί με τους μαθητές και συνεργάτες τους, θα κυριαρχούσαν στο πεδίο.

Απόρροια της συνάντησης στο Dartmouth

Στην IBM, ο Nathaniel Rochester και οι συνεργάτες του κατάφεραν να δημιουργήσουν μερικά από τα πρώτα προγράμματα τεχνητής νοημοσύνης. Ο Herbert Gelernter δημιούργησε το Geometry Theorem Prover, το οποίο μπορούσε να αποδεικνύει θεωρήματα της γεωμετρίας τα οποία θεωρούνταν δύσκολα.

Ο Minsky επέβλεψε μια σειρά φοιτητών οι οποίοι ασχολήθηκαν με περιορισμένα προβλήματα που η επίλυσή τους απαιτούσε νοημοσύνη, τα πεδία αυτά ονομάστηκαν μικρόκοσμοι (microworlds). Ένας από αυτούς ο Tom Evans, έφτιαξε το ANALOGY (1968) όπου επέλυε προβλήματα γεωμετρικών αναλογιών που εμφανίζονται στα τεστ ευφυίας.



Παράδειγμα προβλήματος που μπορεί να επιλύσει το πρόγραμμα ANALOGY του Evans.

Lisp

Το 1958 ο John McCarthy μετακόμισε από το Dartmouth στο MIT, εκεί σε ένα υπόμνημα του, το MIT AI Lab Memo No. 1, όρισε τη γλώσσα υψηλού επιπέδου **Lisp**, η οποία θα γινόταν η κυρίαρχη γλώσσα προγραμματισμού της τεχνητής νοημοσύνης. Η Lisp ήταν για τον McCarthy το εργαλείο που χρειαζόταν, αλλά υπήρχε ένα ακόμη πρόβλημα, η πρόσβαση στους τότε δυσεύρευτους και δαπανηρούς υπολογιστικούς πόρους, το οποίο εν τέλει κατάφεραν να αντιμετωπίσουν όταν μαζί με κάποιους συναδέλφους στο MIT επινόησαν το χρονομερισμό (time sharing).
Μέσα στην ίδια χρονιά ο McCarthy περιέγραψε το πρώτο ολοκληρωμένο σύστημα Τεχνητής Νοημοσύνης, το Advice Taker, όπως και το Logic Theorist που ήταν σχεδιασμένο να χρησιμοποιεί γνώση για να αναζητά λύσεις σε προβλήματα.

GPS

Το 1976 ο Newell και ο Simon έφτιαξαν τον **'γενικό λυτή προβλημάτων'** (General Problem Solver – GPS), όπου ήταν σχεδιασμένο να μιμείται τα ανθρώπινα πρωτόκολλα επίλυσης προβλημάτων. Αποδείχτηκε πως η σειρά με την οποία το πρόγραμμα εξέταζε τους υποστόχους και τις διάφορες δυνατές ενέργειες ήταν παρόμοια με εκείνη που προσεγγίζει ο άνθρωπος τα ίδια προβλήματα. Το GPS ήταν το πρώτο πρόγραμμα που προσέγγισε την ανθρώπινη σκέψη, αλλά και βοήθησε τους Newell και Simon να διατυπώσουν την υπόθεση του '**φυσικού συστήματος συμβόλων'** (physical symbol system). Το νόημα της υπόθεσης αυτής δεν είναι άλλο από το ότι οποιοδήποτε σύστημα (άνθρωπος ή μηχανή) που έχει νοημοσύνη, πρέπει και να μπορεί να χειρίζεται δομές δεδομένων που αποτελούνται από σύμβολα. Έτσι λοιπόν ξεκίνησε η ανάγκη για καλύτερη **αναπαράσταση της γνώσης.**

1.4 Δοκιμασία Turing

Ο Alan Turing (1950) κατάφερε να σχεδιάσει και να παρέχει έναν ικανοποιητικό επιχειρησιακό ορισμό της νοημοσύνης. Πρότεινε μια δοκιμασία όπου βασιζόταν στην αδυναμία να γίνει δίακριση μεταξύ μηχανής και ανθρώπου. Ο υπολογιστής περνάει την δοκιμασία όταν ο άνθρωπος εξεταστής αφού έχει θέσει γραπτώς μερικές ερωτήσεις δεν μπορεί να συμπεράνει από τις γραπτές απαντήσεις αν συνομίλησε με άνθρωπο ή μηχανή.

Για να καταφέρει ένας υπολογιστής να περάσει τη δοκιμασία χρειάζεται να έχει τις εξής ικανότητες.

- **επεξεργασία φυσικής γλώσσας,** ώστε να μπορεί να επικοινωνεί
- **αναπαράσταση γνώσης,** ώστε να αποθηκεύει αυτά που γνωρίζει ή μαθαίνει
- **αυτοματοποιημένη συλλογιστική,** ώστε να χρησιμοποιεί τις αποθηκευμένες πληροφορίες για να απαντά ερωτήσεις και να παράγει συμπεράσματα

- **μηχανική μάθηση,** ώστε να προσαρμόζεται σε νέες περιστάσεις και να εντοπίζει ή να συμπεραίνει πρότυπα

Εμείς στην παρούσα εργασία θα επικεντρωθούμε στο πεδίο της αναπαράστασης της γνώσης και συγκεκριμένα με τα σημασιολογικά δίκτυα.

Άνθρωπος εναντίον μηχανής

Το 1997 πραγματοποιήθηκε η πρώτη αναμέτρηση μεταξύ μηχανής και ανθρώπου στις στρατηγικές ικανότητες, ύστερα από 6 παιχνίδια σκάκι ο υπερυπολογιστής της IBM, Deep Blue κατάφερε να νικήσει τον παγκόσμιο πρωταθλητή στο σκάκι Garry Kasparov. Την μέρα εκείνη η τεχνητή νοημοσύνη ξεπέρασε ακόμη ένα όριο, μια μηχανή μόλις είχε καταφέρει να νικήσει τον καλύτερο άνθρωπο σε ένα παιχνίδι που εδώ και καιρό θεωρείται ως το αποκορύφωμα της διανοητικής στρατηγικής.

Ο ενθουσιασμός που ακολούθησε αυτήν την πρώτη νίκη των μηχανών, είχε σαν αποτέλεσμα μέσα στους επόμενους τρεις μήνες να τεθεί ακόμη μεγαλύτερος στόχος. Το διεθνές συνέδριο Τεχνητής Νοημοσύνης που πραγματοποιήθηκε στην Ναγκόγια της Ιαπωνίας έβαλε σαν στόχο το 2050 μια ομάδα ποδοσφαίρου αποτελούμενη από αυτόνομα ρομπότ να νικήσει την καλύτερη ποδοσφαιρική ομάδα από ανθρώπους, με επίσημους κανόνες της FIFA.

1.5 Αναπαράσταση Γνώσης

Η αναπαράσταση της γνώσης είναι μια σημαντική περιοχή της τεχνητής νοημοσύνης, η οποία ασχολείται με το πως μπορεί να παρασταθεί αποδοτικότερα, γνώση γύρω από ένα πεδίο μέσα σε έναν υπολογιστή, με τελικό στόχο την επίλυση προβλημάτων. Η αναγκαιότητα αναπαράστασης της γνώσης, διαπιστώθηκε όταν άρχισαν οι πρώτες αποτυχίες στην προσπάθεια κατασκευής ευφυών συστημάτων με στόχο να γίνουν γενικοί λυτές προβλημάτων, όπως το παράδειγμα που αναφέραμε με το GPS των Newell και Simon.

Ένα οποιοδήποτε σύστημα, για να είναι νοήμον, πρέπει να έχει γνώση για τον κόσμο του καθώς και τα μέσα να βγάζει συμπεράσματα ή τουλάχιστον να δρα, με βάση αυτή τη γνώση. Στον άνθρωπο η γνώση λαμβάνεται από τα αισθητήριά του όργανα, έπειτα ο ανθρώπινος νους ταξινομεί, συγκρίνει, προσθέτει ή αφαιρεί αυτά που πρέπει και κάνει συσχετισμό αντικειμένου-έννοιας. Οι ιδιαιτερότητες των παραπάνω διαδικασιών ήταν πάντοντε αντικείμενο μελέτης από τον ίδιο τον άνθρωπο. Πως θα ήταν δυνατόν όμως και τι θα χρειαζόταν για να αποκτήσει ένας υπολογιστής ανάλογο τρόπο συλλογισμού; Τι ακριβώς είναι η αναπαράσταση της γνώσης;

Για να απαντήσουμε τις παραπάνω ερωτήσεις αλλά και να κάνουμε την έννοια της αναπαράστασης της γνώσης πιο εύκολα κατανοητή, θα περιγράψουμε τους πέντε ρόλους που μπορεί να παίξει [1].

1. **Η αναπαράσταση γνώσης είναι ένα υποκατάστατο των οντοτήτων του πραγματικού κόσμου.**

Ο συλλογισμός κάθε ευφυούς οντότητας θεωρείται εσωτερική διαδικασία αλλά τα περισσότερα πράγματα για τα οποία συλλογιζόμαστε είναι ή αφορούν πραγματικά αντικείμενα που υπάρχουν στον εξωτερικό κόσμο. Για παράδειγμα εάν ένα πρόγραμμα ή ένας άνθρωπος θελήσουν να συλλογιστούν τη συναρμολόγηση ενός ποδηλάτου, αναγκαστικά θα πρέπει να υπάρξει και ο συλλογισμός (εσωτερική διαδικασία) για την αλυσίδα, τους τροχούς, τα φρένα και άλλα εξαρτήματά του, που όμως υπάρχουν μόνο στον πραγματικό κόσμο.

Εαν λοιπόν δούμε την αναπαράσταση σαν υποκατάστατο, αυτομάτως προκύπτουν δύο ερωτήματα. Το πρώτο αφορά το υποκοκατάστατο, για ποιό λόγο υπάρχει; Πρέπει δηλαδή να υπάρχει ένα είδος αντιστοιχίας μεταξύ του αντικειμένου και του υποκατάστατού του. Το δεύτερο ερώτημα έχει να κάνει με την πιστότητα του υποκαταστάτου, πόσο καλός αντιπρόσωπος του αυθεντικού πραγματικά είναι; Ποιές ιδιότητες του αυθεντικού περιλαμβάνει και ποιές παραβλέπει; Γενικά η τέλεια πιστότητα είναι αδύνατη, τόσο στην πράξη όσο και επί της αρχής. Το μόνο πιστό αντίγραφο ενός αντικειμένου είναι το ίδιο το αντικείμενο. Όσες προσπάθειες αναπαράστασης και να κάνουμε θα υπάρχουν πάντα διαφορές από το πρωτότυπο αντικείμενο, τουλάχιστον στην τοποθεσία (την πραγματική θέση στον κόσμο) αν όχι σε κάτι άλλο.

2. **Η αναπαράσταση γνώσης είναι ένα σύνολο από οντολογικές δεσμεύσεις.**

Όπως αναφέραμε παραπάνω κάθε αναπαράσταση είναι ημιτελής, κάποιες ιδιότητες επιλέγονται ενώ κάποιες παραλείπονται. Έτσι επιλέγοντας να κάνουμε οποιαδήποτε αναπαράσταση αναπόφευκτα φτιάχνουμε και κάποια σύνολα αποφάσεων, όσον αφορά το πως και το τι βλέπει η αναπαράστασή μας στον κόσμο. Επομένως επιλέγοντας μια αναπαράσταση αυτόματα επιλέγουμε και ένα σύνολο οντολογικών δεσμεύσεων, τις οποίες μπορούμε να τις δούμε ως ένα «ζευγάρι γυαλία», όπου αποφασίζουν τι βλέπουμε, ξεδιαλύνοντας κάποια κομμάτια με το κόστος να θολώνουν κάποια άλλα. Αυτό το «ζευγάρι γυαλιά», εκτός από αναπόφευκτο είναι και εξαιρετικά χρήσιμο, εαν επιλεγούν σωστά οι ιδιότητες προς αναπαράσταση θα βοηθήσουν να επικεντρωθεί ο συλλογισμός μας σε ένα κομμάτι του φυσικού κόσμου. Διαφορετικά η απίστευτη πολυπλοκότητα του φυσικού κόσμου θα μπορούσε να κάνει τον οποιοδήποτε συλλογισμό από πολύ δύσκολο εως αδύνατο.

3. **Η αναπαράσταση γνώσης είναι μια αποσπασματική θεωρία ενός ευφυούς συλλογισμού.**

Ο ρόλος αυτός αποδίδεται στην αρχική σύλληψη της αναπαράστασης, όπου υποκινείται μόνο από κάποια διαίσθηση του πως ο άνθρωπος συλλογίζεται ευφυές, ή από την πεποίθηση του τι σημαίνει ευφυής συλλογισμός . Η θεωρία είναι αποσματική διότι ενσωματώνει μόνο ένα μερος της γνώσης η οποία είναι με τη σειρά της είναι μόνο ένα κομμάτι του πολύπλοκου φυσικού κόσμου.

Η θεωρία της αναπαράστασης ενός ευφυούς συλλογισμού δεν είναι τόσο εμφανής, για να γίνει πιο εύκολα κατανοητή μπορούμε να εξετάσουμε τα τρία κομμάτια από τα οποία αποτελείται.

i. Η εννοιολογική σύλληψη της αναπαράστασης του ευφυούς συλλογισμού
ii. Το σύνολο των συμπερασμάτων τα οποία η αναπαράσταση **εγκρίνει**
iii. Το σύνολο των συμπερασμάτων τα οποία η αναπαράσταση **προτείνει**

Τα κομμάτια αυτά μπορούν να θεωρηθούν και ως απαντήσεις της αναπαράστασης σε τρία θεμελιώδη ερωτήματα:

i. Τι σημαίνει να συλλογίζεσαι ευφυώς;
ii. Τι **μπορούμε** να συμπεράνουμε από αυτά που ξέρουμε;
iii. Τι **πρέπει** να συμπεράνουμε από αυτά που ξέρουμε;

Οι απαντήσεις σε αυτά τα ερωτήματα μας οδηγούν στην καρδιά και στην νοοτροπία μιας οποιασδήποτε αναπαράστασης.

4. Η αναπαράσταση γνώσης είναι μέσο για υπολογισμό.

Ο συλλογισμός για τις μηχανές, αλλά και για τους ανθρώπους είναι μια υπολογιστική διαδικασία. Για να χρησιμοποιήσουμε μια αναπαράσταση πρέπει αναγκαστικά να κάνουμε και υπολογισμούς. Έτσι σαν αποτέλεσμα, προκύπτουν αναπόφευκτα κάποια ερωτήματα όσον αφορά την υπολογιστική απόδοση, τα οποία κρίνονται απαραίτητα για οποιαδήποτε αναπαράσταση.

5. Η αναπαράσταση γνώσης είναι μέσο για την ανθρώπινη έκφραση.

Τελικώς η αναπαράσταση γνώσης είναι το μέσο για να εκφράσουμε διάφορα πράγματα στον κόσμο μας, το οποίο μέσο έχουμε ανάγκη και χρησιμοποιούμε εφόσον θέλουμε να επικοινωνούμε με τις μηχανές αλλά και μεταξύ μας. Έτσι, ο πέμπτος αυτός ρόλος αποσκοπεί στη χρήση από τους ίδιους ανθρώπους, ως μέσο έκφρασης και επικοινωνίας.

Συνέπειες της ατέλειας της αναπαράστασης

Δύο σημαντικά συμπεράσματα απορρέουν από την αδυναμία πιστής αναπαράστασης. Όταν έχουμε ατελή υποκατάστατα θα έχουμε και -αναπόφευκτα- λάθος συμπεράσματα, ανεξαρτήτως της διαδικασίας συλλογισμού, ανεξαρτήτως της αναπαράστασης που χρησιμοποιήθηκε. Δεύτερο συμπέρασμα είναι ότι λόγω της απίστευτα μεγάλης πολυπλοκότητας του φυσικού κόσμου, πρέπει, πάλι αναπόφευκτα, κατά την διαδικασία αναπαράστασης ενός αντικειμένου, να πούμε ψέμματα (τουλάχιστον από παράλειψη) για κάποιες από τις ιδιότητές του. Αυτό επίσης, θα μπορούσε να μεταφραστεί πως κάποια από τα τεχνητά αντικείμενα που αναπαραστήσαμε, δεν υπάρχουν στον φυσικό κόσμο.

1.6 Μοντέλα Αναπαράστασης της Γνώσης

- **Production Rules (Κανόνες Παραγωγής)**

Ένας τρόπος για αναπαράσταση της γνώσης είναι μια συλλογή από κανόνες παραγωγής, οι οποίοι είναι ζευγάρια συνθήκης-δράσης τα οποία ορίζουν μια συνθήκη όπου όταν αυτή ικανοποιηθεί προκαλεί την 'πυροδότηση' του κανόνα παραγωγής και τελικώς την πραγματοποίηση της δράσης.

- **Frames**

Ένα Frame είναι μια συλλογή από ιδιότητες και αντίστοιχες τιμές (καθώς και πιθανοί περιορισμοί στις τιμές) το οποίο περιγράφει μια οντότητα στον κόσμο. Ένα μοναδικό Frame σπάνια είναι χρήσιμο. Συνήθως χτίζουμε συστήματα Frames από κάποιες συλλογές Frames οι οποίες είναι συνδεδεμένες ανά μεταξύ τους, δυνάμει του γεγονότος πως κάποια τιμή μίας ιδιότητας ενός Frame μπορεί να είναι ένα άλλο Frame.

- **Σημασιολογικά δίκτυα**

Αρχικά τα σημασιολογικά δίκτυα είχαν σχεδιαστεί ως ένας τρόπος για την αναπαράσταση της σημασίας των Αγγλικών λέξεων. Σε ένα σημασιολογικό δίκτυο, η πληροφορία αποθηκεύεται σε σύνολα από κόμβους, οι οποίοι είναι συνδεδεμένοι με σύνολα από τόξα μαζί με ετικέτες, τα οποία τόξα αναπαριστούν τις σχέσεις μεταξύ των διαφόρων κόμβων. Μαθηματικά ένα σημασιολογικό δίκτυο, θα μπορούσε να οριστεί ως ένα κατευθυνόμενο γράφημα με ετικέτες.

2. LISP

Η Common Lisp είναι μια διάλεκτος προερχόμενη από τη γλώσσα Lisp, η οποία είναι η δεύτερη παλαιότερη γλώσσα προγραμματισμού που χρησιμοποιείται σήμερα, δημιουργήθηκε από τον John McCarthy και τους μαθητές του (1958), πρώτη είναι μόνο η FORTRAN κατά ένα χρόνο (Graham 1995).

Όταν πρωτοεμφανίστηκε κατέκλυσε τον κλάδο της τεχνητής νοημοσύνης, μιας και υποστήριζε το συμβολικό προγραμματισμό, αφού εκείνη την εποχή η τεχνητή νοημοσύνη ήταν επίσης συμβολική. Όμως μέχρι και σήμερα είναι ένα πολύ καλό εργαλείο στην επίλυση προβλημάτων για οποία δεν γνωρίζουμε την λύση τους, γεγονός που περιγράφει την τεχνητή νοημοσύνη τέλεια. Η Lisp είναι σχεδιασμένη για να εξελίσσεται συνεχώς, σε σημείο που να μην αναγνωρίζεται σε σχέση με πριν, αυτή η συνεχής αλλαγή είναι μεγάλο πλεονέκτημα, για αυτό και κατάφερε να «επιζήσει» πραγματικά πολλά χρόνια σε έναν κλάδο τόσο γρήγορα εξελισσόμενο όπου η διάρκεια ζωής νεοεισαχθέντων τεχνολογιών μειώνεται με αναλογικούς χρόνους.

Η Lisp υποστηρίζει τον διαδικαστικό, το συναρτησιακό, τον αντικειμενοστραφή προγραμματισμό ή συνδυασμό αυτών, δηλαδή παρέχει ευελιξία με πολλές δυνατότητες. Αυτό που την κάνει να ξεχωρίζει όμως είναι ο διαδραστικός προγραμματισμός, δηλαδή ο χρήστης μπορεί να τρέχει κομμάτια από το πρόγραμμα ή εντολές σε πραγματικό χρόνο και να παίρνει την έξοδο. Ήταν η Lisp αυτή που άνοιξε τον δρόμο στον διαδραστικό προγραμματισμό ώστε να αρχίσει να ενσωματώνεται και σε άλλες γλώσσες ή εφαρμογές.

Στον διαδραστικό προγραμματισμό ο χρήστης αφιερώνει λιγότερο χρόνο 'πείθωντας' τον μεταγλωττιστή να αφήσει να τρέξει το πρόγραμμά του, αυτό διότι βρίσκει τυχόν λάθη στον κώδικα γρηγορότερα σε πραγματικό χρόνο, δε χρειάζεται να γίνει μεταγλώττιση και έλεγχος σε ολόκληρο το πρόγραμμα αλλά είναι δυνατό να γίνει έλεγχος συνάρτηση – συνάρτηση. Έτσι περισσεύει περισσότερος χρόνος για να δουλέψει κάποιος στον κώδικά του (Seibel 2005).

Η παραπάνω διαδικασία επιτυγχάνεται με το χαρακτηριστικό σύστημα της Lisp 'διάβασε-αξιολόγησε-εκτύπωσε επανάληψη' (read-eval-print loop, REPL).
Ο χρήστης βλέπει στην οθόνη ενα μήνυμα τύπου κέλυφος εντολής Unix/Linux που μοιάζει έτσι

```
CL-USER>
```

Όπου μπορεί να πληκτρολογήσει εκφράσεις και να τις τρέξει ώστε να ξεκινήσει η διαδικασία της Lisp 'διάβασε-αξιολόγησε-εκτύπωσε' και μετά ξανά σε αναμονή για να γίνει επανάλυψη της διαδικασίας για την επόμενη έκφραση, αυτή η συνεχής κατάσταση ονομάζεται REPL ή Lisp Listener.

Όλες αυτές οι εκφράσεις που έχουν διαβαστεί-αξιολογηθεί-εκτυπωθεί επιτυχώς 'χτίζονται' μέσα στην γλώσσα και είναι προσβάσιμες εάν ο χρήστης το επιλέξει μέσω των ανάλογων functions. Όμως για να είναι αποτελεσματικότερος αλλά και πιο πρακτικός ο προγραμματισμός, χρειάζεται ένα περιβάλλον ή ενας επεξεργαστή κειμένου ο οποίος συνεργάζεται με το κέλυφος εντολών της Lisp ώστε η δημιουργία και η διαμόρφωση των εντολών που πρόκειται να τρέξουν στο REPL να γίνεται εύκολα. Αυτό το περιβάλλον το εξασφαλίζουν ο Emacs μαζί με το SLIME για τα οποία θα γίνει περιγραφή της εγκατάστασης και χρήσης τους στο επόμενο κεφάλαιο.

2.1 Συντακτικό

Το συντακτικό της Lisp είναι διαφορετικό από το συντακτικό γλωσσών προερχόμενες από την οικογένεια της Algol. Δύο σημαντικές διαφορές είναι, ότι η Lisp χρησιμοποιεί την 'Πολωνική Σημειογραφεία' (Polish notation) που σημαίνει συντακτικά πρώτα είναι οι τελεστές και μετά οι τελεσταίοι. Η άλλη διαφορά είναι ότι χρησιμοποιούνται παρενθέσεις για τον διαχωρισμό των εκφράσεων.

Οπότε εάν θέλαμε να κάνουμε την πράξη (2 + 2) , με την σύνταξη της Lisp θα γραφόταν (+ 2 2). Όπως είπαμε προηγουμένως το κέλυφος εντολών είναι σε αναμονή για να γράψουμε εκφράσεις , να τις αξιολογήσει και να εκτυπώσει το αποτέλεσμα, εαν τρέξουμε

```
CL-USER> (+ 2 2)
```

Θα μας επιστραφεί 4. Μπορούμε επίσης να κάνουμε πράξεις οι οποίες είναι σε εμφωλευμένες παρενθέσεις , στην έκφραση (+ 3 (* 2 3)) πρώτα θα εκτελεστεί η εσωτερική παρένθεση και μετά η εξωτερική.

2.1.1 Τύποι Δεδομένων

Η Lisp περιέχει διάφορους τύπους δεδομένων όπως strings, integers, characters, lists, symbols και άλλα, ας περιγράψουμε κάποια από αυτά.

Lists
Είναι ο πιο σημαντικός τύπος δεδομένων της Lisp από εκεί πήρε άλλωστε και το όνομά της (LISt Processing). Οι λίστες περικλείονται πάντα από παρενθέσεις, όταν μια λίστα διαβάζεται από το REPL κατασκευάζεται και αξιολογείται.
Μια λίστα απο 4 αριθμούς (1 5 10 15).
Εμφωλευμένη λίστα (2 3 (a b)).

Strings
Είναι ένα διάνυσμα από χαρακτήρες όπου αναγνωρίζεται όταν βρίσκετε ανάμεσα από διπλά εισαγωγικά, ‘‘Ενα String’’ όταν διαβάζετε από το REPL ξανά-εκτυπώνεται.

Symbols
Άλλος ένας κοινός τύπος δεδομένων στο συντακτικό της Lisp είναι τα σύμβολα, (a b c) μια λίστα με τρία σύμβολα. Για να εμφανίσουμε την τιμή από το σύμβολο ‘a’ το τρέχουμε στο REPL χωρίς απόστροφο και μας επιστρέφεται η τιμή του. Για να επιστραφεί το ίδιο το σύμβολο πρέπει να τρέξουμε ‘α. Τα στοιχεία αυτά αποτελούν το μεγαλύτερο κομμάτι του συντακτικού της Lisp.

2.2 Συναρτήσεις

Οι συναρτήσεις είναι το επίκεντρο στον προγραμματισμό με Lisp, ο χρήστης μπορεί να προσδιορίσει μια νέα συνάρτηση με την έκφραση defun τα αρχικά της οποίας σημαίνουν Define Function. Συνήθως χρειάζονται τρία ορίσματα για μια ολοκληρωμένη συνάρτηση, ένα όνομα για την συνάρτηση το οποίο θα χρησιμοποιούμε για να την καλούμε, μια λίστα για τις παραμέτρους και μια έκφραση που θα αποτελέσει το σώμα της. Μια συνάρτηση η οποία εκτυπώνει στην οθόνη Hello-world όταν την καλούμε έχει ως εξής,

```
(defun hello-world ()
  (print “Hello-world”))
```

Να σημειωθεί πως έχουμε αφήσει την λίστα με τις παραμέτρους κενή μιας και δεν χρειάζεται κάποια παράμετρος, η συνάρτηση καλείται με το όνομά της (hello-world). Για να γίνει η μεταγλώττιση της συνάρτησης ή γενικότερα οποιασδήποτε έκφρασης υπάρχουν δύο

τρόποι, ο ένας είναι να τρέξουμε την συνάρτηση κατευθείαν στο REPL και ο αλλός μέσω του SLIME όπου με κάποιες συντομεύσεις μπορούμε να κάνουμε μεταγλώττιση συγκεκριμένα κομμάτια κώδικα ή και ολόκληρα αρχεία με κώδικα. Εάν θελήσουμε να τροποποιήσουμε μια συνάρτηση, απλα μεταγλωττίζουμε την νέα έκδοση η οποία αντικαθιστά την προηγούμενη.
Υπάρχει εγκατεστημένο ήδη αρκετό υλικό από συναρτήσεις, τελεστές , κάποιους ειδικούς τελεστές όπως το IF, DO, print, μέσα σε κάθε εκτέλεση της Lisp.
Παρακάτω η εκτέλεση της dotimes συνάρτησης η οποία θα εκτελεστεί 5 φορές και κάθε φορα θα εκτυπώνει το τρέχον i.

```
(dotimes (i 5)
  (print i))
```

2.2.1 Αναδρομή

Ένα από τα πιο σημαντικά χαρακτηριστικά οποιασδήποτε προγραμματιστικής γλώσσας είναι η αναδρομή, ας δούμε μια συνάρτηση πως χρησιμοποιεί τη μέθοδο αυτή, συγκεκριμένα για τον υπολογισμό ενός παραγοντικού αριθμού.

```
(defun factorial (x)
   (if (plusp x)
     (* x (factorial (- x 1)))
     1))
```

Εάν καλέσουμε την συνάρτηση factorial με παράμετρο 5 θα μας επιστραφεί ο αριθμός 120. Ας κάνουμε μια ανάλυση για το πως λειτουργεί. Αρχικά οι συναρτήσεις if και plusp που είναι ήδη ενσωματωμένες μέσα στην Lisp έχουν την εξής λειτουργία, η plusp ελέγχει εάν ο αριθμός που ακολουθεί είναι θετικός, εάν είναι τότε επιστρέφει ‘αληθής’ και επιτρέπει στο if να εκτελέσει την έκφραση που ακολουθεί πρώτη, δηλαδή το (* x (factorial (- x 1))) διαφορετικά εάν ο αριθμός είναι αρνητικός ή και μηδέν το plusp θα επιστρέψει ‘ψευδής’ και η if θα εκτελέσει την δεύτερη έκφραση που ακολουθεί, δηλαδή το 1.

Χωρίς να έχουμε εμπειρία είναι δύσκολο να καταλάβουμε πόσες φορές θα γίνει αναδρομή και πόσες θα εκτελεστεί συνολικά η συνάρτηση. Ένας τρόπος να γίνει ευκολότερη η κατανόηση μιας οποιαδήποτε συνάρτησης είναι με την συνάρτηση trace, η οποία ενεργοποιείται εάν τρέξουμε στο κέλυφος εντολών (trace factorial).

Οπότε την επόμενη φορά που θα εκτελέσουμε την συνάρτηση factorial δεν θα εμφανιστεί απλά το αποτέλεσμα αλλά όλα τα βήματα που εκτελέστηκαν. Το (factorial 5) θα εμφανίσει

```
 0: (FACTORIAL 5)
  1: (FACTORIAL 4)
   2: (FACTORIAL 3)
    3: (FACTORIAL 2)
     4: (FACTORIAL 1)
      5: (FACTORIAL 0)
      5: FACTORIAL returned 1
```

```
    4: FACTORIAL returned 1
   3: FACTORIAL returned 2
  2: FACTORIAL returned 6
 1: FACTORIAL returned 24
0: FACTORIAL returned 120
```

Πράγμα που σημαίνει πως η εσωτερική συνάρτηση εκτελέστηκε 6 φορές , στην $6^{η}$ φορά το (if (plusp x) επέστρεψε 'ψευδής' αφού το x είχε τιμή 0 και εκτελέστηκε η εξωτερική συνάρτηση οπού η γραμμή (* x (factorial (- x 1))) είχε μια μορφή τύπου : (* 5 (4)(3)(2)(1)) μιας και η εσωτερική συνάρτηση είχε ήδη εκτελεστεί όλες τις φορές που έπρεπε.

2.3 Macros

Ο John Foderaro είχε αποκαλέσει την Lisp «μια προγραμματιζόμενη προγραμματιστική γλώσσα». Αυτό γιατί κάνοντας μικρές αλλαγές για να γράφονται συγκεκριμένα προγράμματα πιο εύκολα, μπορείς να μετατρέψεις την Lisp σε μια οποιαδήποτε γλώσσα προγραμματισμού. Ακριβώς έτσι άρχισαν να προκύπτουν πολλές και διάφορες υλοποιήσεις της Lisp , όπως η Common Lisp που χρησιμοποιούμε στην παρούσα εργασία. Το ρόλο αυτό τον έδωσαν τα Macros, τα οποία μπορούν και αλλάζουν τον τρόπο που θα μεταγλωττιστεί ένα πρόγραμμα.

Με μια έννοια θα μπορούσαμε να πούμε οτι τα Macros παράγουν κώδικα, όταν η μεταγλώττιση συναντάει ένα Macro τότε το 'ανοίγει' στο κώδικα στον οποίο το έχουμε ορίσει, η διαδικασία αυτή συνεχίζεται μέχρι να 'ανοιχτούν' ή αλλιώς αναδιπλωθούν όλα τα Macros σε ένα πρόγραμμα και μόνο τότε γίνεται η μεταγλώττιση των συναρτήσεων και των υπόλοιπων εκφράσεων.

Τα Macros προσδιορίζονται παρόμοια με τις συναρτήσεις, με μια έκφραση defmacro από το define macro, αλλά είναι αρκετά πιο δύσκολο να γραφτεί ένα Macro από μια συνάρτηση. Πρέπει να μπορέσει κάποιος να φανταστεί πως θα είναι ο κώδικας αφού ανοίξει το Macro.

Τα Macro έχουν την εξής δομή :

```
(defmacro name (parameter*)
   "προαιρετική περιγραφή του Macro."
   body-form*)
```

Πως θα ήταν μια απλή ύψωση στο τετράγωνο, σε macro και σε συνάρτηση:

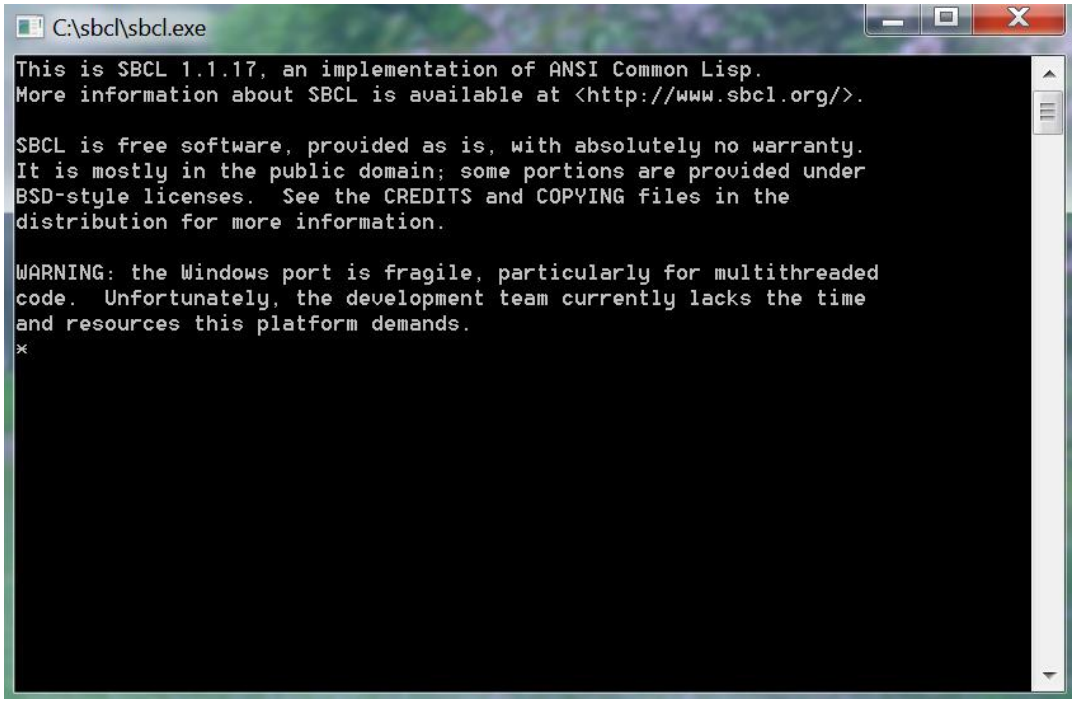
```
(defun sqr (x)
"ύψωση του χ στο τετράγωνο"
  (* x x))


(defmacro sqr (x)
"ύψωση του χ στο τετράγωνο"
 `(* ,x ,x))
```

Βλέπουμε πως ο κώδικας δεν διαφερει πολύ, οπότε ποιά είναι η διαφορά με τα Macro της Lisp; Η διαφορά είναι ότι η Lisp προτού μεταγλωττίσει τον κώδικα του προγραμμάτος μας θα ελέγξει εάν υπάρχουν Macro και θα τα ανοίξει, αφού τελειώσει με όλα τα Macro μόνο τότε θα ξεκινήσει η μεταγλώττιση ολόκληρου του κώδικα, για αυτό χρειάζεται και να διαβάσουμε το * ως String με το backquote, επειδή η μεταγλώττιση είναι αυτή που θα αναγνωρίσει τον τελεστή για την λειτουργία του πολλαπλασιασμού. Έτσι ακριβώς μπορούμε να καθοδηγούμε τον μεταγλωττιστή στο τι βλέπει και με ποιό τρόπο να τρέχει συγκεκριμένα προγράμματα, δηλαδή μπορούμε να γράψουμε προγράμματα που γράφουν προγράμματα. Επιπλέον σημαίνει ταχύτητα, στο απλό Macro του sqr δεν υπάρχει σημαντική διαφορά αλλά σε προγράμματα με πολλές γραμμές κώδικα και δύσκολές πράξεις η διαφορά γίνεται αισθητή.

2.4 Προετοιμασία περιβάλλοντος

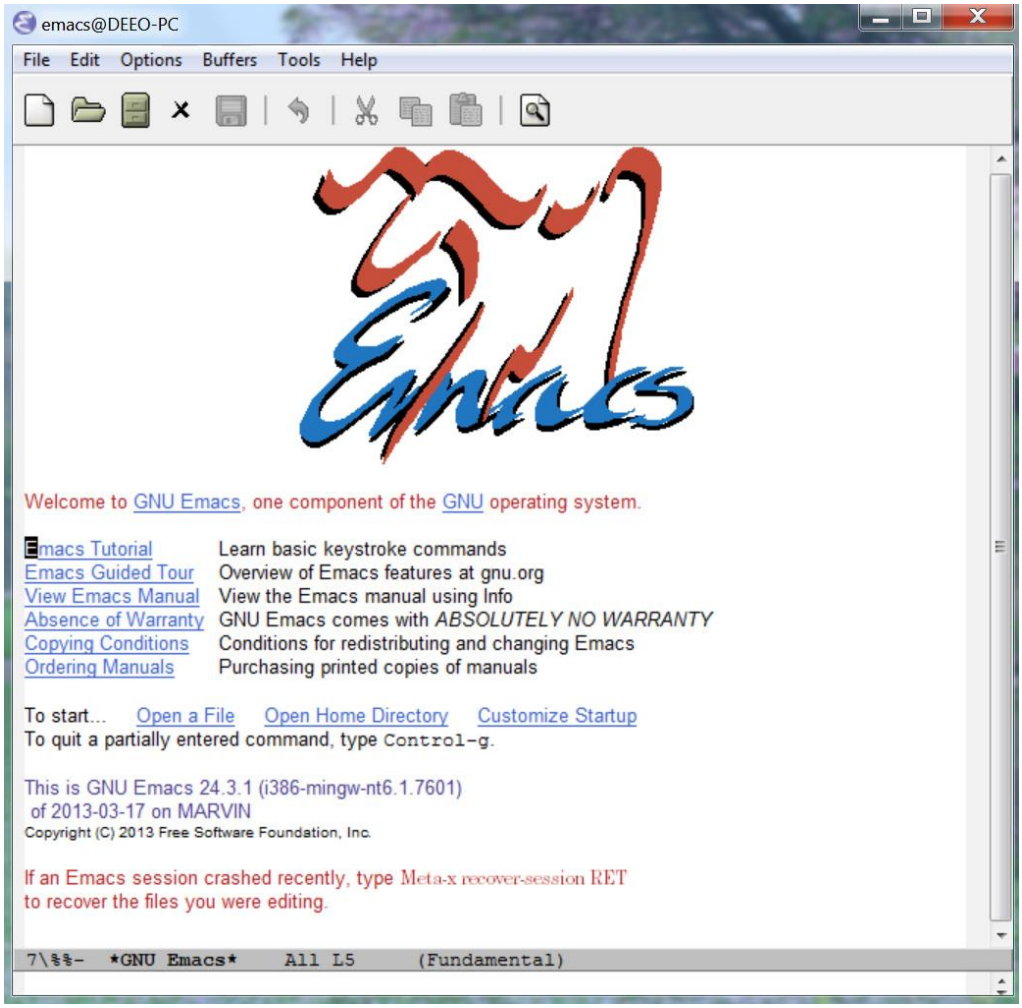
Σε αυτό το κεφάλαιο θα περιγράψουμε τα εργαλεία που θα χρησιμοποιήσουμε για τον προγραμματισμό με Common Lisp. Το βασικότερο είναι να εγκαταστήσουμε κάποια υλοποίηση της Common Lisp, εμείς θα επιλέξουμε την Steel bank Common Lisp, ύστερα τα Emacs, την quicklisp με την οποία κατεβάζουμε διάφορες βιβλιοθήκες αλλά και το SLIME (Superior Lisp Interaction Mode for Emacs).

Το πρώτο βήμα είναι να βρούμε μια υλοποίηση της Common Lisp, στο site http://www.sbcl.org/
Στον κατάλογο με τις λήψεις η Steel Bank Common Lisp προσφέρει την δικία της υλοποίηση ανάλογα με το λειτουργικό σύστημα που χρησιμοποιούμε. Σε αυτό το σημείο να τονίσουμε ότι για συστήματα παρόμοια με Unix/Linux η υποστήριξη είναι πολύ μεγαλύτερη μιας και τα windows δεν είναι τόσο δημοφιλή για προγραμματισμό σε Lisp. Εμείς πάντως θα δείξουμε την εγκατάσταση σε Windows 7. Κατεβάζουμε λοιπόν την κατάλληλη έκδοση και την εγκαθιστούμε, μόλις τρέξουμε το εκτελέσιμο αρχείο βλέπουμε μια οθόνη η οποία θυμίζει γραμμή εντολών.



Αυτό και μόνο το εργαλείο θα ήταν αρκετό για να προγραμματίσουμε σε Common Lisp, δίνοντας εκφράσεις γραμμή – γραμμή, τρέχοντας εντολές και παίρνοντας την ανάλογη έξοδο. Έτσι προκύπτει και ο διαδραστικός προγραμματισμός.
Όμως θα ήταν εξαιρετικά επίπονη διαδικασία να γράψουμε μεγάλα προγράμματα χωρίς κάποιο περιβάλλον που να μας παρέχει συγκεκριμένες δυνατότητες αλλά και κάποιες ευκολίες. Αυτόν τον ρόλο θα πάρουν ο Emacs μαζι με το Slime.

Ο Emacs είναι κάτι παραπάνω από ένας απλός Editor προσφέροντας πολλές δυνατότητες, θα μπορούσε κανείς να δαπανήσει αρκετούς μήνες για να τις εξερευνήσει.
Πηγαίνουμε στην σελίδα http://www.gnu.org/software/emacs/ και κατεβάζουμε την τελευταία έκδοση. Αφού εγκαταστήσουμε τον Emacs πρέπει να ‘πούμε’ στον υπολογιστή που μπορεί να τα βρει.
Δεξί κλικ στο my computer → Properties → Advanced system settings →Environment Variables
Και έπειτα πρέπει να βρούμε στην μεταβλητή PATH στο πεδίο Value τον προορισμό όπου έχουμε εγκαταστήσει τον Emacs στον υπολογιστή μας.



Το επόμενο βήμα είναι να εγκαταστήσουμε την quicklisp, η οποία μας επιτρέπει να κατεβάζουμε πακέτα και βιβλιοθήκες για την Common Lisp. Πηγαίνουμε στην σελίδα http://www.quicklisp.org/beta/ και κατεβάζουμε το αρχείο quicklisp.lisp το οποίο τοποθετούμε στον φάκελο εγκατάστασης της SBCL. Για την εγκατάστασή της quicklisp θα χρειαστούμε την γραμμή εντολών των Windows, αφού την ανοίξουμε πηγαίνουμε στην τοποθεσία όπου βρίσκεται η quicklisp με την εντολή cd και έπειτα τρέχουμε την εντολή :

sbcl --load quicklisp.lisp

Έτσι ανοίγουμε την έκδοση της Lisp μας, η οποία είναι η SBCL και με αυτήν φορτώνουμε το αρχείο quicklisp. Έπειτα ακολουθούμε τις οδηγίες εγκατάστασης που εμφανίζονται στην οθόνη μας.

Αμέσως μπορούμε να εκμεταλευτούμε τις δυνατότητες της quicklisp για να εγκαταστήσουμε το τελευταίο εργαλείο μας, το SLIME.

Αφού έχουμε ανοίξει την SBCL φορτώνοντας της quicklisp τρέχουμε:

(ql:quickload “quicklisp-slime-helper”)

έτσι πολύ απλά και γρήγορα γίνεται λήψη και εγκατάσταση αυτόματα.

Τώρα μένει να συνδέσουμε τον Emacs, το SLIME και την SBCL μεταξύ τους έτσι ώστε κάθε φορά που τρέχουμε τον Emacs να φορτώνουν και τα υπόλοιπα.
Πηγαίνουμε στον φάκελο εγκατάστασης των Emacs, ανοίγουμε έναν Editor και αντιγράφουμε μέσα τον παρακάτω κώδικα :

```
(load (expand-file-name "~/quicklisp/slime-helper.el"))
(setq inferior-lisp-program "sbcl")
```

Αποθηκεύουμε το αρχείο με την ονομασία .emacs και πλέον κάθε φορά που τρέχουμε το εκτελέσιμο αρχείο του Emacs φορτώνεται και το αρχείο .emacs οπού μέσα έχουμε γράψει τις εντολές για να φορτώνει και ολόκληρο το περιβάλλον μας. Η εγκατάσταση του περιβάλλοντός μας έχει ολοκληρώθεί, παρακάτω θα περιγράψουμε κάποιες συντομεύσεις Emacs οι οποίες είναι απαραίτητες για μεταγλώτιση, αποθήκευση, φόρτωση αρχείων και δίαφορες άλλες λειτουργίες.

2.5 Εντολές Emacs

Ο Emacs (Editing MACroS) είναι γραμμένος εξ ολοκλήρου σε κώδικα Lisp, παρέχουν σχεδόν όλες τις δυνατότητές τους με τη χρήση συντομεύσεων, διαφορετικών όμως από τις κοινές, αντιγραφή (CTRL-C), επικόλληση (CTRL-V). Εάν το θέλουμε όμως μπορούμε να τροποποιήσουμε συντομεύσεις με αυτές που βολεύει εμάς προσωπικά ή να δημιουργήσουμε καινούργιες.

Προτείνεται οι νέοι χρήστες να τρέξουν το Tutorial του Emacs και να το διαβάσουν. Είναι ξεκάθαρο και χρήσιμο. Ο Emacs λειτουργεί με συντομεύσεις οι οποίες κατά κύριο λόγο χρειάζονται πατημένο το κουμπί «ALT» ή το «CTRL» και ύστερα ανάλογα τον συνδυασμό πλήκτρων, προβαίνουν στην ανάλογη ενέργεια. Για λόγους πρακτικούς το alt θα το απεικονίζουμε M ενώ το ctrl με C, έτσι όταν ξέρουμε πως η συντόμευση C-c C-p πραγματοποιεί μεταγλώττιση και φόρτωση του αρχείου στο οποίο βρίσκετε ο κέρσορας του ποντικιού εννοούμε οτι κρατάμε πατημένο το ctrl ενώ πατάμε μια φορά το γράμμα “c” και ύστερα κρατάμε ξανά πατημένο το ctrl ενώ πατάμε “p”.

Με την παραπάνω νοοτροπία οι πιο κοινές συντομεύσεις Emacs έχουν ως εξής :

C-c C-p ; πραγματοποιεί μεταγλώτιση φόρτωση του αρχείου

C-c C-c ; Πραγματοποιεί μεταγλώττιση στην έκφραση όπου ο κέρσορας είναι στο τέλος

C-x-e ; Πραγματοποιεί αξιολόγηση στην έκφραση όπου ο κέρσορας είναι στο τέλος

C-x-f ; αναμένει πληκτρολόγηση ονόματος αρχείου ώστε να το φορτώσει

C-x-s ; αποθηκεύει το τρέχον αρχείο

C-x 3 ; Χωρίζει το τρέχον παράθυρο σε δυο

C-x 1 ; Επιστρέφει σε ένα παράθυρο μόνο

M-x ; αναμένει κάποιο όνομα προγράμματος ώστε να το εκκινήσει Π.χ M-x «slime»

Αυτές είναι κάποιες από τις βασικές εντολές του Emacs οι οποίες προσφέρουν ταχύτητα και άνεση στον προγραμματισμό με Common Lisp.

2.6 Πλεονεκτήματα- Μειονεκτήματα

Ασφάλεια με αυθόρμητο προγραμματισμό

Στις συνηθείς γλώσσες (JAVA, C++) όταν θέλουμε να τρέξουμε ένα πρόγραμμα, πρώτα πρέπει να περάσει ολόκληρο το πρόγραμμα από τον μεταγλωτιστή χωρίς κανένα σφάλμα, εάν υπάρξει (έστω και ένα μικρό) τότε όλος ο κώδικας είναι εντελώς άχρηστος, τουλάχιστον ώσπου διορθώσουμε το λάθος.
Το μεγάλο πλεονέκτημα στον προγραμματισμό με Lisp είναι το ότι μπορούμε να επικεντρωθούμε στον κώδικά μας κομμάτι – κομμάτι όπως για παράδειγμα με την εντολή C-c C-c, άρα όταν υπάρχει σφάλμα σε ένα σημείο τα υπόλοιπα κομμάτια κώδικα (εφ' όσον δεν χρησιμοποιούν αυτό το σημείο) έχουν την δυνατότητα να τρέξουν κανονικά και να επιστρέψουν την έξοδο της λειτουργίας τους.
Έτσι λοιπόν υπάρχει η ελευθερία και η ασφάλεια να 'πειράζουμε' τον κώδικα περισσότερο από άλλες γλώσσες, αφού δεν υπάρχει ο κίνδυνος να σταματάει κάθε φορά η λειτουργία ολόκληρου του προγράμματος.Ο προγραμματισμός γίνεται αυθόρμητος.

Απόδοτικότητα

Η κάθε έκφραση της Lisp είναι ανεξάρτητη από τις υπόλοιπες εκφράσεις, (εκτός αν γίνεται παραπομπή σε άλλο κομμάτι κώδικα μέσα στην ίδια την έκφραση) αυτό σημαίνει ότι οι διάφορες τροποποιήσεις ή προσθήκες στον κώδικα, επηρεάζουν συγκεκριμένα μέρη του προγράμματος. Επίσης επιτυγχάνεται ο εντοπισμός σφαλμάτων πολύ γρήγορα, αφού μπορούμε να απομονώσουμε την έκφραση που έχει λάθος και να δουλέψουμε μόνο σε αυτήν, έτσι ο προγραμματιστής εξοικονομεί χρόνο, αποδίδει καλύτερα.

Επανασχεδιασμός της γλώσσας

Από την στιγμή που τα Macro έχουν αυτήν την ιδιαίτερη λειτουργία να 'ανοίγουν' τον κώδικα τους πριν ξεκινήσει η μεταγλώττιση μας δίνεται η δυνατότητα να κάνουμε αλλαγές με την Lisp που σε άλλες γλώσσες μόνο ο ίδιος ο σχεδιαστής μπορεί να το κάνει. Ας υποθέσουμε πως ένας σχεδιαστής μιας γλώσσας ξεχάσει να υλοποιήσει τη λειτουργία του IF ή του While και

παρόλο που επικοινωνήσαμε μαζί του για να τα συμπεριλάβει στην επόμενη έκδοση , δεν θέλει να το κάνει, με τη Lisp μπορούμε να τα προσθέσουμε μόνοι μας. Εάν θέλουμε μπορούμε να τροποποιήσουμε και τις λειτουργίες των ήδη υπάρχοντων τελεστών, ή να φτιάξουμε παρόμοιες οι οποίες θα εξυπηρετούν κάποιο προσωπικό μας σκοπό.

Μειωμένη απόδοση όταν απαιτείται συνεργασία

Η Lisp παρέχει την δυνατότητα στον προγραμματιστή να διαμορφώσει την ίδια την γλώσσα όπως βολεύει τον ίδιο, να την φέρει ακριβώς στα μέτρα του. Δεν υπάρχουν πρότυπα αναγκαία να διατηρηθούν, οπότε αυτή η φοβερή ιδιότητα θα μπορούσε να λειτουργεί και σαν μειονέκτημα. Ο προγραμματιστής κάποιες φορές αλλάζει την Lisp φτιάχνοντας δικά του 'προσωπικά' πρότυπα, σε τέτοιο βαθμό ώστε ο κώδικας να είναι δυσνόητος από τρίτους προγραμματιστές Lisp.
Σε μεγάλα έργα, όπου επιβάλλεται αρκετοί άνθρωποι να συνεργαστούν και να διαβάσουν κώδικα από συναδέλφους, είτε δεν θα υπάρχει τόσο καλή απόδοση αφού ο καθένας θα έχει συνηθίσει την δικία του 'έκδοση' της Lisp είτε θα πρέπει να μη χρησιμοποιηθεί σε τέτοιο βαθμό η ελευθερία που παρέχει η συγκεκριμένη γλώσσα.

Υπερβολικά πολλές παρενθέσεις

Η διάταξή στο συντακτικό της Lisp χαρακτηρίζεται από τις παρενθέσεις, με αυτές διαχωρίζονται όλα τα στοιχεία της γλώσσας, οι συναρτήσεις, τα Macros. Μερικές φορές όμως ειδικά όταν υπάρχουν πολλές εμφωλευμένες εκφράσεις οι παρενθέσεις συσσωρρεύονται σε βαθμό να δημιουργείται σύγχυση ως προς το που ακριβώς αρχίζει και τελειώνει η κάθε έκφραση.

Ελλιπής υποστήριξη από περιβάλλοντα

Ίσως ένας από τους λόγους για τον οποίο πλεον η Lisp δεν χρησιμοποιείται ευρέως, είναι πως ποτέ δεν υπήρξε κάποιο περιβάλλον το οποίο να είναι δωρεάν, να τρέχει σε όλες τις πλατφόρμες και να διαθέτει υψηλή υποστήριξη στον προγραμματισμό. Οι υποστηρικτές της Lisp, επέλεξαν να διαχωριστούν σε πολλές πλατφόρμες με διαφορετικά πρότυπα η κάθε μια και το αποτέλεσμα ήταν σιγά – σιγά να «σπάει» όλο και περισσότερο η κοινότητα της Lisp σε μικρότερα κομμάτια.

2.7 Συμπεράσματα

Macros

Η Lisp όπως είδαμε βασίζεται κυρίως στον προγραμματισμό με συναρτήσεις αλλά σε πιο προχωρημένο επίπεδο, συναρτήσεις σε συνδυασμό με Macros τα οποία προσδίδουν μια μοναδική ιδιότητα στον προγραμματισμό.
Στο βιβλίο του ANSI Common Lisp, ο Graham (1995) υποστηρίζει πως όταν κάποιος γράφει Macros, επιλέγει τι βλέπει ο μεταγλωττιστής. Είναι σαν να τον ξαναγράφει. Έτσι χρειάζεται να σκέφτεται σαν ένας σχεδιαστής γλώσσας.
Επιπλέον τα Macros τρέχουν σε χρόνο μεταγλώττισης και όχι σε χρόνο αξιολόγησης (run time) όπως οι υπόλοιπες εκφράσεις, δηλαδή προσφέρει ταχύτητα στην κωδικοποιήση, αλλά και εναλλακτικές δυνατότητες στον προγραμματισμό.
Τα Macros όμως επειδή ακριβώς δίνουν την δυνατότητα να ξαναγράψει ο κάθε προγραμματιστής την Lisp σε πιο προσωπικό επιπέδο, τείνουν να δίνουν ένα χαρακτήρα στη Lisp πιο ατομικό, χωρίς την παρότρυνση για συνεργασία που υπάρχει σε άλλες γλώσσες. Έτσι η Lisp είναι μια γλώσσα που θα μπορούσε να επιλέγεται με κριτήριο τον αριθμό των ατόμων που πρέπει να συνεργαστούν σε ένα έργο.

Αξία μάθησης της Lisp

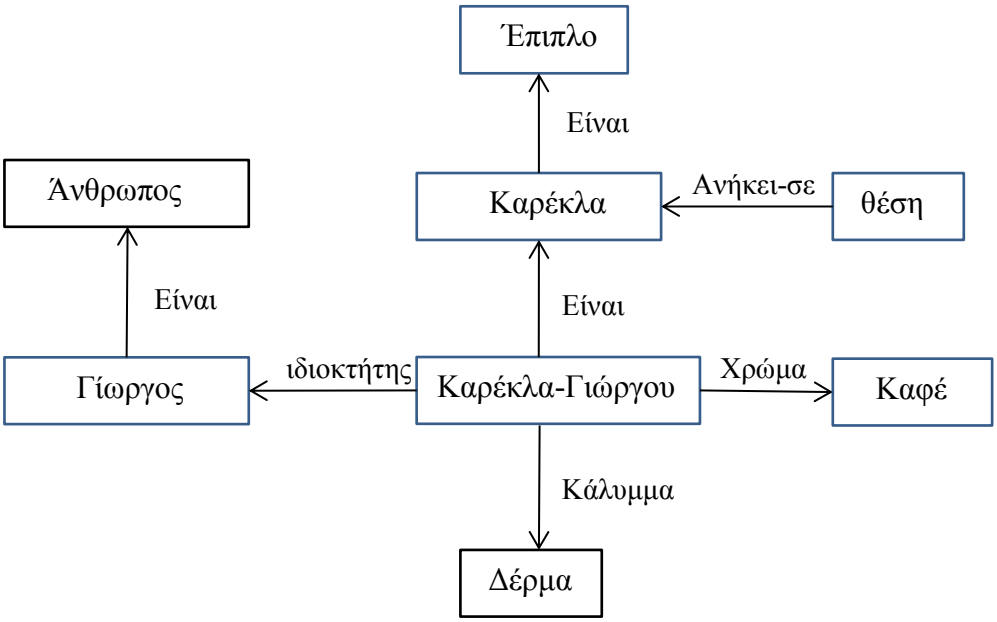
Η Lisp θα μπορούσε να πεί κάποιος όμως, πως δεν χρησιμοποιείται πλέον στο βαθμό που χρησιμοποιούνται άλλες γλώσσες όπως για παράδειγμα η Java και η C++ (μάλιστα με μεγάλη διαφορά). Άρα για ποιόν λόγο να προτιμήσει κάποιος να αφιερώσει τον χρόνο του για να μάθει την Lisp;

Στην έκθεση του «How to become a hacker» [7], ο Raymond είχε γράψει για την αξία που έχει η μάθηση της Lisp, πως όταν καταφέρεις να την καταλαβείς θα σε ‘αφήσει’ καλύτερο προγραμματιστή για τις επόμενες μέρες σου ακόμα και αν δεν χρησιμοποιήσεις ποτέ την ίδια την Lisp αρκετά. Έπειτα και από την δική μου συνάντηση με την Lisp δε μπορώ παρά να συμφωνήσω με τον Raymond, διότι οι γλώσσες προγραμματισμού μας μαθαίνουν να μην θέλουμε αυτό που δεν μπορούν να μας προσφέρουν. Πρέπει να σκέφτεσαι σύμφωνα με τη συγκεκριμένη γλώσσα που προγραμματίζεις ώστε να γράψεις κώδικα με αυτήν και είναι δύσκολο να θέλεις κάτι το οποίο δεν μπορείς να περιγράψεις. Δεν καλείσαι απλά να μεταφράσεις αυτά που έχεις μάθει από τη μια γλώσσα σε κάποια άλλη παραπλήσια, αλλά γνωρίζεις νέους τρόπους επίλυσης προβλημάτων. Η προσαρμογή είναι δυσκολότερη αλλά οι ορίζοντες διευρύνονται πραγματικά.

3. ΣΗΜΑΣΙΟΛΟΓΙΚΑ ΔΙΚΤΥΑ

3.1 Ορισμός σημασιολογικού δικτύου

Ένα σημασιολογικό δίκτυο είναι ένα γραφικό σύστημα με σκοπό την αναπαράσταση γνώσης με μορφή συνδεδεμένων κόμβων και τόξων με ετικέτες. Πιο συγκεκριμένα χρησιμοποιούνται για προτασιακή αναπαράσταση, για αυτό τα συναντάμε και ως προτασιακά δίκτυα. Οι πρώτες υλοποιήσεις σημασιολογικών δικτύων σε υπολογιστές έγιναν για την τεχνητή νοημοσύνη και για μηχανική μετάφραση, όμως νεότερες εκδόσεις τους χρησιμοποιούνταν από πολύ καιρό στη φιλοσοφία, την ψυχολογία και τη γλωσσολογία.



Σχήμα 1. Σημασιολογικό Δίκτυο

Στο παραπάνω σχήμα ένα σημασιολογικό δίκτυο περιέχει πληροφορίες για έννοιες όπως «Έπιπλο», «Άνθρωπος», «Καρέκλα», «Δέρμα», «Γίωργος». Με κατευθυνόμενα βέλοι και μια ετικέτα περιγράφουμε τις σχέσεις μεταξύ των διαφόρων εννοιών. Δηλαδή ορίζουμε πως ο «Γιώργος» Είναι «Άνθρωπος» ή πως η «Καρέκλα-Γιώργου» έχει χρώμα «Καφέ».

Φυσικά μέσα σε ένα πρόγραμμα, τα σημασιολογικά δίκτυα δε μπορούν να αναπαραστηθούν ως γραφήματα, συνήθως χρειάζεται μια σχέση ιδιότητας-τιμής σε δομή μνήμης. Για παράδειγμα στην LISP κάθε κόμβος θα ήταν ένα άτομο (atom), τα τόξα (βέλοι) θα ήταν ιδιότητες (properties), ενώ οι κόμβοι στην άλλη πλευρά των τόξων θα ήταν οι τιμές, όπως φαίνεται στο Σχήμα 2.

Άτομο	**Λίστα Ιδιοτήτων**
Καρέκλα	((Είναι Έπιπλο))
Καρέκλα-Γιώργου	((Είναι Καρέκλα) (Χρώμα Καφέ) (Κάλυμμα Δέρμα) (Ιδιοκτήτης Γιώργος))
Γιώργος	(Είναι Άνθρωπος)
Θέση	(Ανήκει-σε Καρέκλα)

Σχήμα 2. Σημασιολογικό Δίκτυο σε LISP

3.2 Είδη σημασιολογικών δικτύων

Όλα τα σημασιολογικά δίκτυα έχουν σαν κοινό στοιχείο, το ότι χρησιμοποιούν μια μορφή δηλωτικής γραφικής αναπαράστασης για να αναπαραστήσουν την γνώση, αλλά και για υποστήριξη σε αυτόματα συστήματα συλλογισμού, πάλι όσον αφορά την γνώση. Γενικά αυτά τα δίκτυα χωρίζονται σε έξι κατηγορίες, όπου κάποιες από αυτές είναι αυστηρά λογικά συστήματα. Παρακάτω ακολουθεί μια σύντομη περιγραφή τους.

- **Δίκτυα ορισμού** (Definitional networks): Δίνουν έμφαση στον υπό-τυπο ή την σχέση ‘είναι’ μεταξύ κάποιας έννοιας και κάποιας νέα ορισμένης έννοιας, που ήδη χρησιμοποιήσαμε στα παραπάνω παραδείγματα δικτύων. Το αποτέλεσμα αυτού του δικτύου είναι ένα είδος ιεραρχίας γενίκευσης ή υπαγωγής, που υποστηρίζει κανόνες κληρονομικότητας αφού αντιγράφει τις ιδιότητες ενός υπέρ-τυπου σε όλους τους υπότυπούς του. Εδώ εφόσον οι ορισμοί είναι αληθείς εξ ορισμού, οι πληροφοριές σε αυτά τα δίκτυα θεωρούνται αναγκαστικά αληθείς.

- **Δίκτυα ισχυρισμού** (Assertional networks): Είναι σχεδιασμένα για να εισάγονται προτάσεις. Διαφέρουν από τα δίκτυα ορισμού στο ότι η πληροφορία εδώ θεωρείται

ενδεχομένως σωστή, εκτός και αν είναι σαφώς σημειωμένο με έναν τροποποιητικό συντελεστή. Κάποια δίκτυα ισχυρισμών έχουν προταθεί ως μοντέλα θεμελιωδών κατασκευών για τις σημασιολογίες της φυσικής γλώσσας.

- **Δίκτυα συνεπαγωγής** (Implicational networks): Χρησιμοποιούν την συνεπαγωγή σαν την κύρια σχέση για την σύνδεση των κόμβων. Μπορούν να χρησιμοποιηθούν για την αναπαράσταση πρότυπων πίστης, τυχαιότητας ή συμπερασμάτων.

- **Εκτελέσιμα δίκτυα** (Executable networks): Περιλαμβάνουν κάποιον μηχανισμό ώστε να μπορεί να κάνει συμπερασμούς, να περάσει μηνύματα ή να ψάξει για πρότυπα και συνδέσμους.

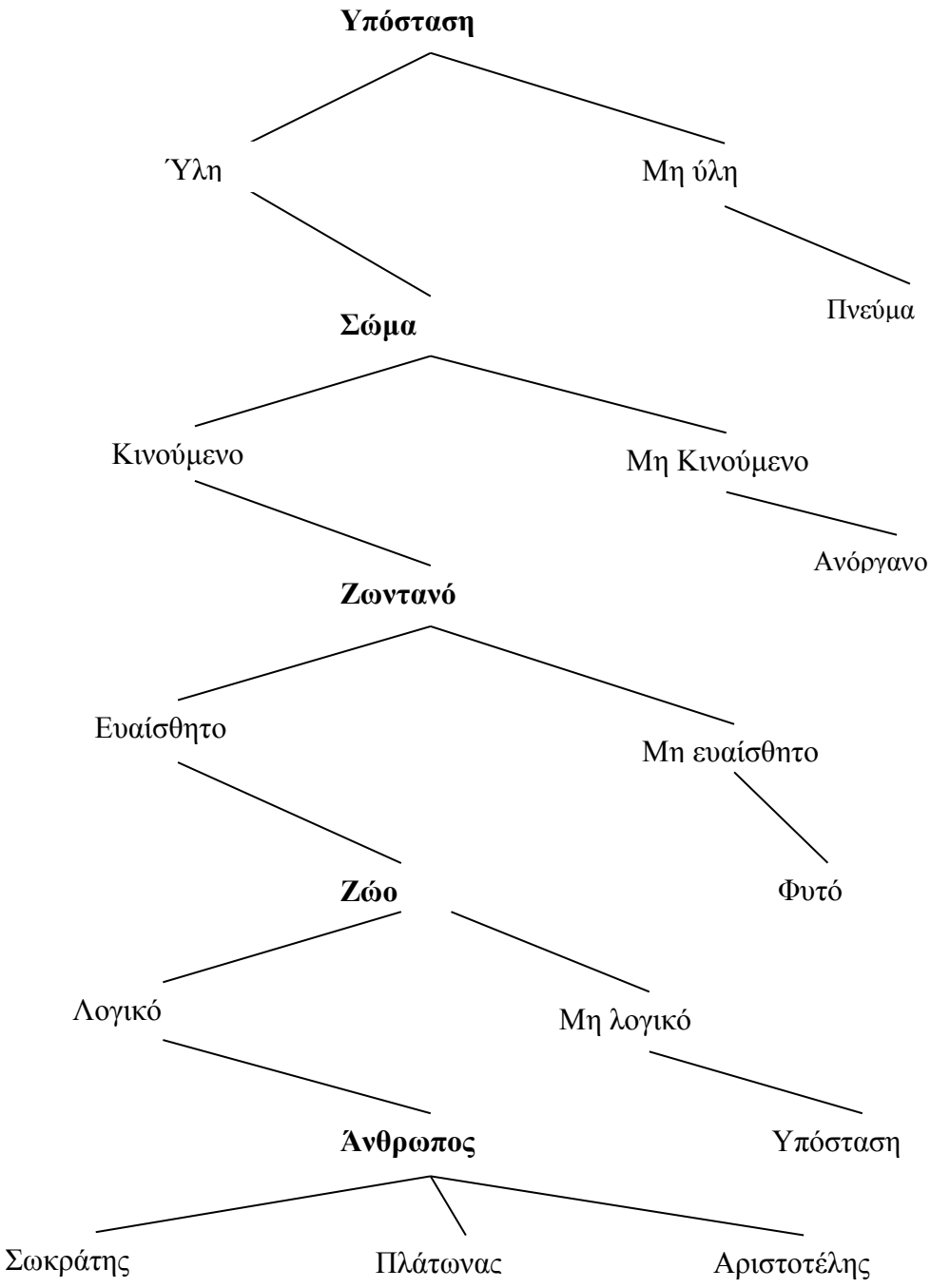
- **Δίκτυα μάθησης** (Learning networks): Χτίζουν ή ενισχύουν την αναπαράστασή τους αποκτώντας γνώση από παραδείγματα. Η νέα γνώση έχει τη δυνατότητα να αλλάξει το παλιό δίκτυο με πρόσθεση ή αφαίρεση κόμβων και τόξων ή αλλιώς τροποποιώντας τα ανάλογα βάρη σε αυτούς.

- **Υβριδικά δίκτυα** (Hybrid networks): Συνδυάζουν δύο ή παραπάνω από τα υπόλοιπα δίκτυα είτε ως ένα δίκτυο είτε ως ξεχωριστά αλλά αλληλεπιδρούμενα δίκτυα.

Μερικά από τα παραπάνω δίκτυα έχουν σχεδιαστεί αποκλειστικά για να εφαρμόσουν υποθέσεις για τους ανθρώπινους γνωστικούς μηχανισμούς, ενώ άλλα σχεδιάστηκαν για υπολογιστική αποδοτικότητα. Μερικές φορές, η υπολογιστική λογική μπορεί να δώσει τα ίδια αποτελέσματα με την ψυχολογία. Η διάκριση μεταξύ δικτύων ορισμού και προσθήκης, για παράδειγμα, έρχεται αρκετά κοντά στη διάκριση μεταξύ σημασιολογικής και προσωρινής μνήμης. Παρακάτω ακολουθεί η αναλυτική περιγραφή των έξι κατηγοριών σημασιολογικών δικτύων.

3.2.1 Δίκτυα ορισμού

Το παλαιότερο γνωστό σημασιολογικό δίκτυο σχεδιάστηκε τον 3^{ο} μ.Χ από τον Έλληνα φιλόσοφο Πορφυρίτη, στη μελέτη του πάνω στις κατηγορίες του Αριστοτέλη. Ο Πορφυρίτης το χρησιμοποίησε για να απεικονίσει την μέθοδο του Αριστοτέλη για τον ορισμό κατηγοριών, προσδιορίζοντας το γένος και γενικά χαρακτηριστικά που διαχωρίζουν τους διαφορετικούς υπό-τυπους ενός υπέρ-τυπου. Στο παρακάτω σχήμα φαίνεται το «δέντρο του Πορφυρίτη» όπως σχεδιάστηκε από τον Πέτρο της Ισπανίας (1239).

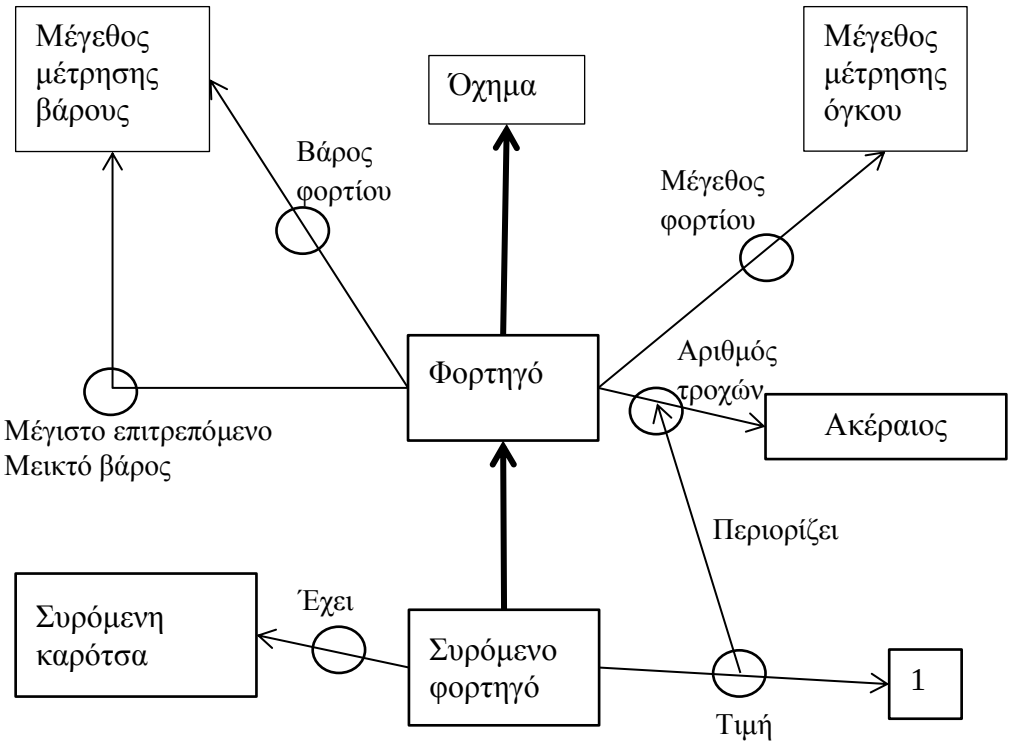


Σχήμα 3. Το δέντρο του Πορφυρίτη, όπως σχεδιάστηκε από τον Πέτρο της Ισπανίας

Παρά την χρονολογία του, το δέντρο του Πορφυρίτη αναπαριστά τον κοινό πυρήνα όλων των μοντέρνων ιεραρχιών που χρησιμοποιούνται για τον ορισμό τύπων εννοιών. Στο δέντρο, το γένος «Υπόσταση» με χαρακτηριστικό γνώρισμα «Ύλη» είναι το σώμα και με «όχι ύλη» είναι το πνεύμα. Ο μοντέρνος κανόνας κληρονομικότητας είναι μια ειδική περίπτωση των συλλογισμών

του Αριστοτέλη, όπου τα «Ζωντανά» αντικείμενα κληρονομούν την υλική υπόσταση από το «Σώμα» προσθέτοντας επίσης το χαρακτηριστικό «Κινούμενο». Ο άνθρωπος κληρονομεί την ευαισθησία, την κίνηση και την υλική υπόσταση, προσθέτοντας το χαρακτηριστικό «Λογικό». Οι μέθοδοι αυτοί, ορισμού και συλλογισμού του Αριστοτέλη χρησιμοποιούνται ακόμη και σήμερα στην τεχνητή νοημοσύνη, στον αντικειμενοστραφή προγραμματισμό αλλά και σε όλα τα λεξικά.

Ανάμεσα στα σύγχρονα συστήματα, η περιγραφική λογική εμπεριέχει τα χαρακτηριστικά από το δέντρο του Πορφυρίτη και παράλληλα προσθέτει διάφορες επεκτάσεις. Προέρχεται από μια προσέγγιση η οποία προτάθηκε από τον Woods (1975) και υλοποιήθηκε από τον Brachman (1979) σε ένα σύστημα όπου ονομάζεται «Γλώσσα Γνώσης Ένα» (KL-ONE). Στο παρακάτω σχήμα φαίνεται ένα KL-ONE δίκτυο το οποίο ορίζει τις έννοιες φορτηγό και συρόμενο φορτηγό ως υποέννοιες του οχήματος.



Σχήμα 4. Έννοιες όπως «Φορτηγό» και «Συρόμενο φορτηγό» σε δίκτυο KL-ONE

Στο παραπάνω σχήμα έχουμε 8 κόμβους και 9 τόξα, οπού το καθένα αναπαριστά διαφορετικού είδους συσχετίσεις. Ο κόμβος «18» σε αντίθεση με τους υπόλοιπους 7 έχει μια συγκεκριμένη τιμή και δεν ορίζει κάποια έννοια. Ο κόμβος «Ακέραιος» ορίζει σαν τύπο τους ακέραιους αριθμούς μόνο για το «Αριθμός τροχών». Οι έννοιες «Φορτηγό» και «Συρόμενο φορτηγό»

ορίζονται σχήμα ενώ οι έννοιες «Όχημα», «Συρόμενη καρότσα», «Μέγεθος μέτρηση βάρους» και «Μέγεθος μέτρησης όγκου» πρέπει να οριστούν σε ένα άλλο KL-ONE διάγραμμα.

Οι πληροφορίες που έχουμε στο σχήμα 4 μπορούν να αναπαραστηθούν ως συλλογισμοί του Αριστοτέλη με τις δυο παρακάτω προτάσεις-δηλώσεις.

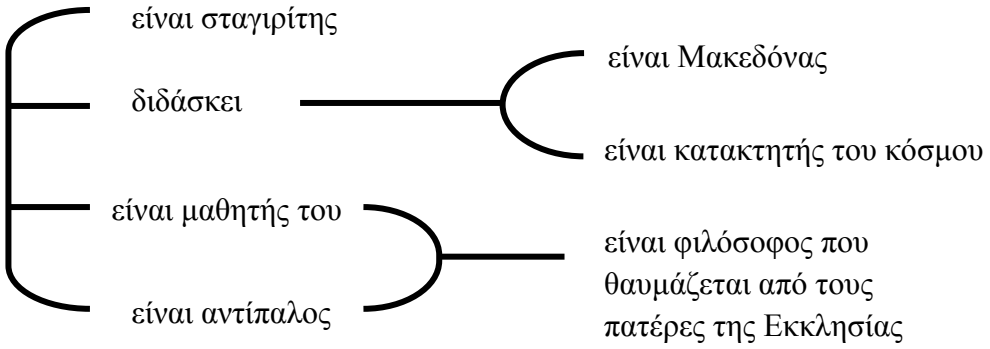
Κάθε **φορτηγό** είναι ένα **όχημα**.

Κάθε **συρόμενο φορτηγό** είναι ένα **φορτηγό** που αποτελείται από
μία **συρόμενη καρότσα**,
ένα **βάρος φορτίου,** όπου είναι ένα **μέγεθος μέτρησης βάρους**,
ένα **μέγιστο επιτρεπόμενο μεικτό βάρος,** όπου είναι ένα **μέγεθος μέτρησης βάρους**,
ένα **μέγεθος φορτίου,** όπου είναι ένα **μέγεθος μέτρησης όγκου**
και έναν **αριθμό τροχών,** όπου είναι ο **ακέραιος 18**.

3.2.2 Δίκτυα ισχυρισμού

Ο Gottlob Frege (1879) ανέπτυξε ένα σχήμα δέντρου για την πρώτη πλήρη έκδοση της λογικής πρώτης τάξης. Ο Charles Sanders Peirce (1880, 1885) ανέπτυξε ανεξάρτητα ένα αλγεβρικό σχήμα, το οποίο με μια αλλάγη στα σύμβολα από τον Peano (1889) έγινε το μοντέρνο σχήμα για τον υπολογισμό κατηγορημάτων.
Να σημειώσουμε πως ενώ στη κοινή γλώσσα οι σχέσεις των ποσοτήτων δηλώνονται με λέξεις σε ένα αλγεβρικό σχήμα δηλώνονται με σύμβολα/πρόσημα. Για παράδειγμα η πρόταση κάθε Α που διαιρείται με Β ισούται με 3Ψ σε αλγεβρικό σχήμα θα γραφόταν Α / Β = 3 Ψ.
Στο παρακάτω σχήμα φαίνεται ένα από τα σχεσιακά γραφήματά του, όπου αναπαριστά την πρόταση:

«Ένας σταγιρίτης δάσκαλος ενός Μακεδόνα κατακτητή του κόσμου είναι μαθητής και αντίπαλος ενός φιλοσόφου που θαυμάζεται από τους πατέρες της Εκκλησίας».



Σχήμα 5. Σχεσιακό γράφημα

Το σχήμα 5 περιλαμβάνει τρεις διακλαδισμένες γραμμές ταυτότητας, η κάθε μια από τις οποίες αντιστοιχεί υπαρξιακά σε μια ποσοτικοποιημένη μεταβλητή στο αλγεβρικό σχήμα. Οι λέξεις και οι φράσεις που υπάρχουν στα άκρα των γραμμών αντιστοιχούν στις σχέσεις ή στα κατηγορήματα του αλγεβρικού σχήματος. Με αυτές τις αντιστοιχίες το σχήμα 5 μπορεί να μεταφραστεί ως:

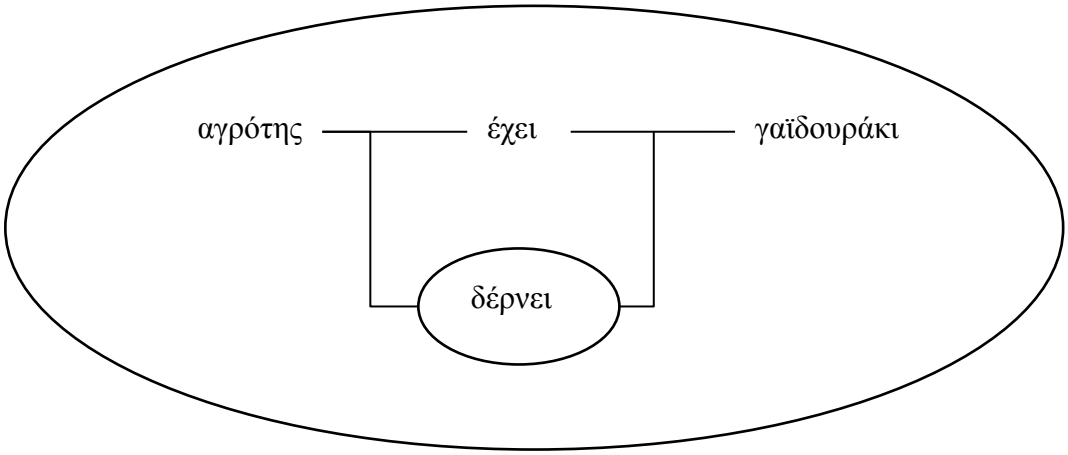
(∃x)(∃y)(∃z)(είναι_σταγιρίτης(x) ∧ διδάσκει(x, y) ∧ είναι_Μακεδόνας(y) ∧ είναι_κατακτητής_του_κόσμου(y) ∧ είναι_μαθητής_του(y,z) ∧ είναι_αντίπαλος_του(y,z) ∧ θαυμάζεται_από_τους_πατέρες_της_Εκκλησίας(z))

Όπως φαίνεται από τον παραπάνω τύπο, ένα σχεσιακό γράφημα μπορεί να αναπαριστά μόνο με δύο λογικούς τελεστές, την σύζευξη ∧ και τον υπαρξιακό ποσοδείκτη ∃.

Άλλοι τελεστές όπως η άρνηση ¬, η διάζευξη ∨, η συνεπαγωγή => και ο καθολικός ποσοδείκτης ∀ εκφράζονται πιο δύσκολα επειδή απαιτούν μεθόδους για την οροθεσία του πεδίου δράσης τους. Το πρόβλημα αναπαράστασης του πεδίου δράσης, όπου ο Peirce αντιμετώπισε στους γράφους του το 1882, εμφανίστηκε και στα πρώτα σημασιολογικά δίκτυα που χρησιμοποιήθηκαν στην τεχνητή νοημοσύνη 80 χρόνια αργότερα.

Το 1897, ο Peirce έκανε μια απλή αλλά ευφυέσταστη ανακάλυψη που έλυσε όλα τα προβλήματα με μιας. Εισήγαγε μια έλλειψη που μπορούσε να περικλείσει αλλά και να αρνηθεί ένα αυθαίρετα μεγάλο γράφημα ή υπογράφημα. Ύστερα, αυτές οι αρνήσεις σε συνδυασμό με τις συζεύξεις και τον υπαρξιακό ποσοδείκτη μπορούσαν να εκφράσουν όλους τους λογικούς τελεστές που χρησιμοποιούνταν στην σχεσιακή άλγεβρα. Η καινοτομία αυτή κατάφερε να μεταμορφώσει τα σχεσιακά γραφήματα σε «υπαρξιακά γραφήματα» όπου ο Peirce τα αποκάλεσε "η λογική του μέλλοντος". Για παράδειγμα, η συνεπαγωγή ⊃, μπορούσε να αναπαραστηθεί με την ενσωμάτωση δύο ελλείψεων, εφόσον (A=>B) είναι ισοδύναμο με ¬(A∧¬B). Στο παρακάτω Σχήμα δίνεται ένα υπαρξιακό γράφημα για την εξής πρόταση:

«Εάν ένας αγρότης έχει ένα γαϊδουράκι, τότε το δέρνει».



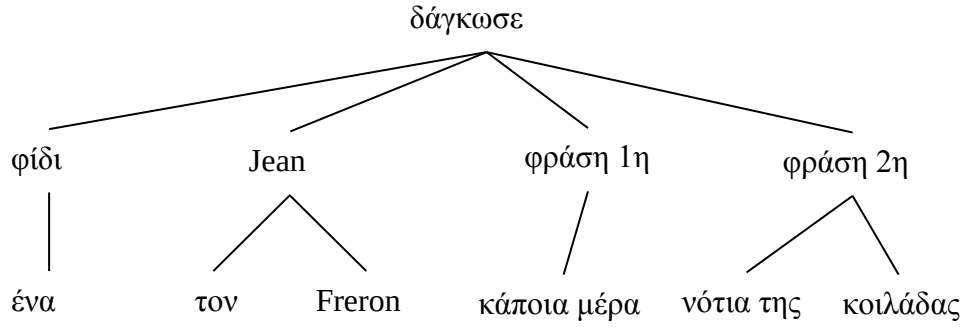
Σχήμα 6. Υπαρξιακό γράφημα

Το Σχήμα 6 μπορεί να μεταφραστεί στους παρακάτω αλγεβρικούς τύπους:
$\sim(\exists x)\ (\exists y)(\text{αγρότης}(\chi) \wedge \text{γαϊδουράκι}(y) \wedge \text{ιδιοκτήτης}(x,y) \wedge\sim \text{δέρνει}(x,y))$
$(\forall x)(\ \forall y)(\text{αγρότης}(x) \wedge \text{γαϊδουράκι}(y) \wedge \text{ιδιοκτήτης}(x,y) => \text{δέρνει}(x,y))$

Στην γλωσσολογία, ο Lucien Tesniere (1959) ανέπτυξε ένα γράφημα για το σύστημα του «γραμματική εξάρτηση». Στο παρακάτω Σχήμα φαίνεται ένα τέτοιο γράφημα για ένα επίγραμμα του Βολταίρου:

«Κάποια μέρα, νότια της κοιλάδας, ένα φίδι δάγκωσε τον Jean Freron».

Στην κορυφή βρίσκεται το ρήμα «δάγκωσε» από το οποίο κρέμονται οι λέξεις που εξαρτώνται από αυτό, το αντικείμενο «φίδι», το υποκείμενο «Jean» και οι δύο φράσεις.



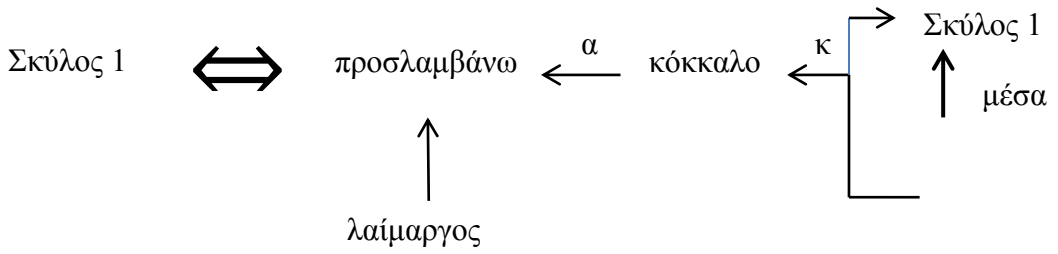
Σχήμα 7. Γράφημα εξάρτησης

Ο Tesniere είχε αποκτήσει μεγάλη επιρροή στις γλωσσολογικές θεωρίες οι οποίες δίνουν περισσότερο έμφαση στη σημασιολογία παρά στην σύνταξη. Ο David Hays (1964) παρουσίασε τη θεωρία εξάρτησης ως μια επίσημη εναλλακτική στους συντακτικούς συμβολισμούς του Chomsky όπου οι Klein και Simmons (1963) την υιοθέτησαν για ένα σύστημα μηχανικής μετάφρασης.

Υπό την επιρροή των Hays και Simmons, ο Roger Schank υιοθέτησε την προσέγγιση της εξάρτησης με την διαφορά ότι έδωσε μεγαλύτερη έμφαση στις έννοιες από ότι στις λέξεις. Στο παρακάτω σχήμα φαίνεται ένα γράφημα εννοιολογικής εξάρτησης για την εξής πρόταση:

«Ένας σκύλος τρώει λαίμαργα ένα κόκκαλο».

Ο Schank χρησιμοποίησε διαφορετικά είδη βελών για διαφορετικές σχέσεις, όπως το ⇔ όπου αντιστοιχεί στην σχέση ρήματος και υποκείμενου ή ένα βέλος σημειωμένο με «α» για το αντικείμενο. Αντικατέστησε το ρήμα «τρώει» με το «προσλαμβάνει» και το «λαίμαργα» με το «λαίμαργος». Πρόσθεσε ένα βέλος σημειωμένο με «κ» για την κατεύθυνση, ώστε να δείξει ότι το κόκκαλο πηγαίνει από ένα αδιευκρίνιστο μέρος μέσα στο σκύλο. Οι συμπερασμοί σε ένα τέτοιο γράφημα γίνονται με την ερμηνεία των κατευθυνόμενων βελών.



Σχήμα 8. Γράφημα εννοιολογικής εξάρτησης

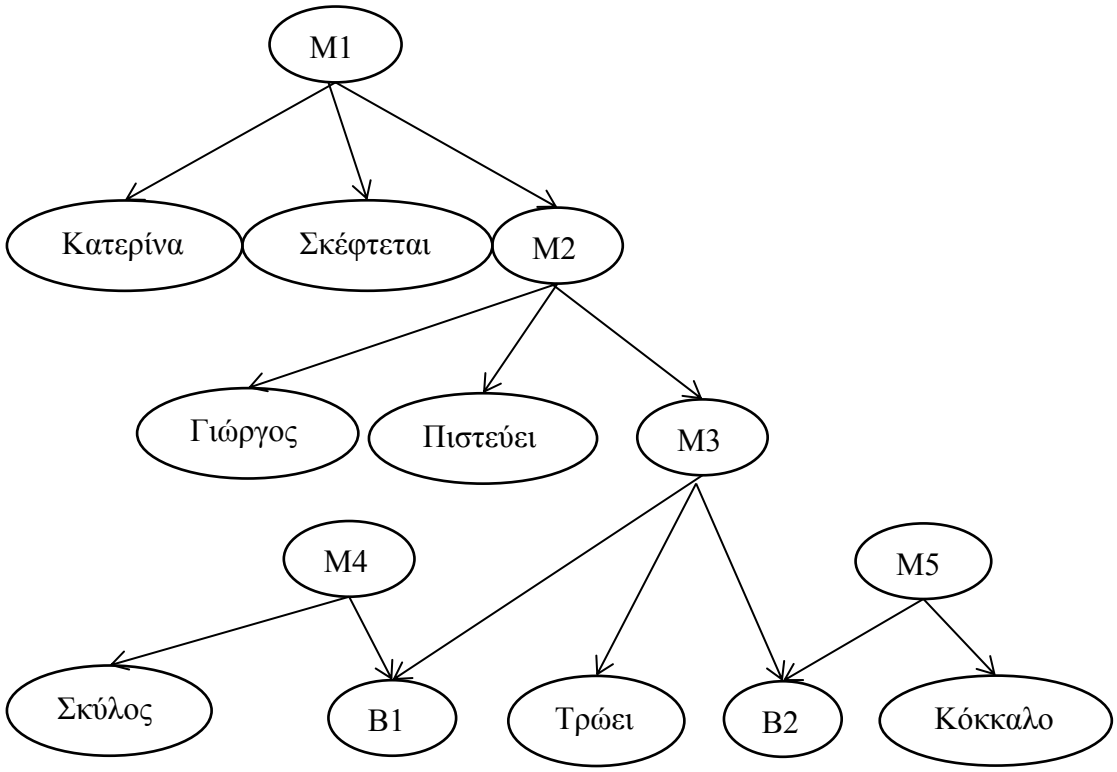
Οι εννοιολογικές εξαρτήσεις ήταν αρχικά κατάλληλες για την αναπαράσταση της πληροφορίας σε επίπεδο πρότασης, αλλά ο Schank και οι συνάδελφοί του αργότερα ανέπτυξαν συμβολισμούς για την αναπαράσταση μεγαλύτερων δομών, στα οποία οι εξαρτήσεις σε επιπέδο πρότασης ήταν ως ενσωματωμένα γραφήματα. Οι μεγαλύτερες δομές ονομάστηκαν σενάρια (scripts), πακέτα οργάνωσης μνήμης και πακέτα θεματικής οργάνωσης. Για τη μάθηση ή την εύρεση μεγαλύτερων δομών, αυτόματα, χρησιμοποιήθηκε ο συλλογισμός με περιπτώσεις (case based reasoning) για την εύρεση κοινών μοτίβων ανάμεσα στις εξαρτήσεις στα χαμηλά επίπεδα.

Λογικά, τα γραφήματα εξαρτήσεων του Tesniere έχουν την ίδια εκφραστική δύναμη με τους σχεσιακούς γράφους του Peirce του 1882, εφόσον οι μόνοι λογικοί τελεστές που μπορούν να τα αναπαραστήσουν είναι η σύζευξη και ο υπαρξιακός ποσοδείκτης. Ακόμα και όταν αυτά τα γραφήματα έχουν κόμβους σημειωμένους με άλλους τελεστές, αποτυγχάνουν να εκφράσουν το πεδίο τους σωστά. Η πιο επιτυχημένη προσέγγιση αναπαράστασης του πεδίου των λογικών τελεστών ήταν η μέθοδος της προσθήκης κόμβων σαφήνειας για την έκφραση προτάσεων. Οι λογικοί τελεστές θα συνδέονταν με τους προτασιακούς κόμβους και οι σχέσεις θα ήταν είτε συσχετισμένες με τους κόμβους, είτε ενσωματωμένες μέσα σε αυτούς. Με αυτά τα κριτήρια τα σχήματα των Frege, Peirce και Kamp θα μπορούσαν να ονομάζονται προτασιακά σημασιολογικά δίκτυα.

Το πρώτο προτασιακό σημασιολογικό δίκτυο που υλοποιήθηκε στην τεχνητή νοημοσύνη ήταν το σύστημα MIND, όπου αναπτύχθηκε από τον Stuart Shapiro (1971). Αργότερα εξελίχθηκε στο «σύστημα επεξεργασίας σημασιολογικών δικτύων» (SNePS), όπου χρησιμοποιήθηκε για να αναπαραστήσει ένα μεγάλο εύρος χαρακτηριστικών στην σημασιολογιά της φυσικής γλώσσας. Στο παρακάτω σχήμα η αναπαράσταση με το SNePS για την πρόταση:

«Η Κατερίνα σκέφτεται ότι ο Γιώργος πιστεύει πως ένας σκύλος τρώει ένα κόκκαλο»

Κάθε κόμβος από M1 έως M5 αναπαριστά μια ξεχωριστή πρόταση, το σχετικό περιεχόμενο της οποίας είναι συνδεδεμένο με τον προτασιακό κόμβο.



Σχήμα 9. Αναπαραστάσεις προτάσεων στο SNePS

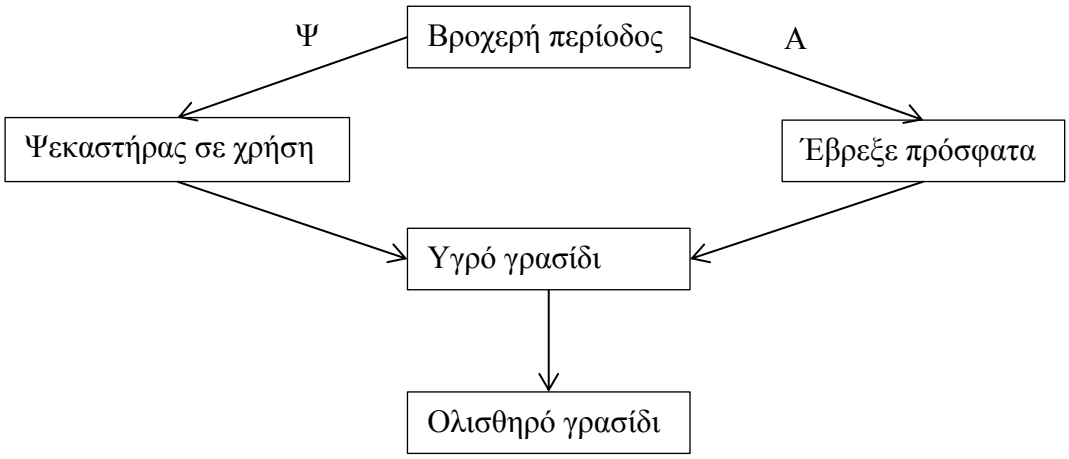
Η πρόταση M1 δηλώνει ότι η Κατερίνα έχει την εμπειρία του ρήματος «σκέφτομαι», του οποίου το αντικείμενο είναι μια άλλη πρόταση η M2. Για την M2, ο Γιώργος έχει την εμπειρία του ρήματος «πιστεύω» με αντικείμενο την πρόταση M3. Στην M3, ο αντιπρόσωπος είναι μια οντότητα B1, η οποία είναι μέλος της κλάσης «σκύλος», το ρήμα «τρώω» και αντικείμενο μια οντότητα B2 που ανήκει στην κλάση «κόκκαλο». Όπως φαίνεται στο σχήμα 9 οι προτάσεις μπορούν να χρησιμοποιηθούν σε κάποιο άλλο επίπεδο για να δηλώσουν κάτι για άλλες προτάσεις.

Τα εννοιολογικά γραφήματα είναι μια ποικιλία από προτασιακά σημασιολογικά δίκτυα, στα οποία οι σχέσεις είναι ενσωματωμένες μέσα σε προτασιακούς κόμβους. Εξελίχθηκαν ως ένας συνδυασμός από τα γλωσσολογικά χαρακτηριστικά των γραφημάτων εξαρτήσεων του Tesniere και των λογικών χαρακτηριστικών των υπαρξιακών γραφημάτων του Peirce, έχοντας επηρεαστεί αρκετά από την τεχνητή νοημοσύνη και την υπολογιστική γλωσσολογία [11].

3.2.3 Δίκτυα συνεπαγωγής

Τα δίκτυα συνεπαγωγής είναι μια ξεχωριστή περίπτωση ενός προτασιακού σημασιολογικού δικτύου στο οποίο η πρωταρχική σχέση είναι η συνεπαγωγή. Όμως και άλλες σχέσεις μπορεί να είναι ενσωματωμένες μέσα στους προτασιακούς κόμβους. Ανάλογα την ερμηνεία, τέτοια δίκτυα μπορεί να ονομάζονται δίκτυα πίστης, αιτιολογικά δικτυα, Bayesian

δίκτυα ή συστήματα διατήρησης της αλήθειας. Κάποιες φορές ο ίδιος γράφος μπορεί να χρησιμοποιηθεί για οποιαδήποτε ή και όλες τις παραπάνω αναπαραστάσεις. Το παρακάτω Σχήμα δείχνει τις πιθανές αιτίες για ολισθηρό γρασίδι: κάθε κουτί αναπαριστά μια πρόταση, τα βέλη έπειτα δείχνουν την συνεπαγωγή από την μια πρόταση στην άλλη. Εάν είναι εποχή βροχών, το βέλος με Α υπονοεί ότι πρόσφατα έβρεξε, διαφορετικά, το βέλος με Ψ υπονοεί πως έχει χρησιμοποιηθεί ο ψεκαστήρας. Στους κόμβους που φεύγει μόνο ένα βέλος, η αλήθεια της πρώτης πρότασης συνεπάγεται την αλήθεια και στην δεύτερη πρόταση, ενώ εάν είναι ψευδής η πρώτη πρόταση δεν υπονοείται τίποτα για τη δεύτερη.



Σχήμα 10. Δίκτυο συνεπαγωγής για τον συλλογισμό «ολισθηρό γρασίδι»

Το Σχήμα 10 αναπαριστά τη γνώση που θα χρειαζόταν κάποιος εφόσον περπάτησε στο γρασίδι, γλίστρησε και έπειτα θέλει να συλλογιστεί την αιτία. Μια πιθανή αιτία για το ολισθηρό γρασίδι θα ήταν να είναι βρεγμένο. Μπορεί να είναι βρεγμένο είτε γιατί ο ψεκαστήρας ήταν σε χρήση, είτε γιατί πρόσφατα έβρεξε. Εάν ήταν βροχερή περίοδος, ο ψεκαστήρας δεν θα ήταν σε χρήση, άρα θα πρέπει να είχε βρέξει.

Αυτό το είδος συλλογισμού που περιγράψαμε παραπάνω, μπορεί να εκτελεστεί από διάφορα συστήματα τεχνητής νοημοσύνης. Ο Chuck Rieger (1976) ανέπτυξε μια έκδοση των αιτιολογικών δικτύων, που την χρησιμοποίησε για ανάλυση περιγραφών προβλημάτων στα Αγγλικά και την μετάφρασή τους σε ένα δίκτυο που θα μπορούσε να υποστηρίξει τον συλλογισμό επιπέδων. Ο Benjamin Kuipers (1984, 1994), επηρεασμένος αρκετά από την προσέγγιση του Rieger, ανέπτυξε μεθόδους ποιοτικού συλλογισμού, που λειτουργούν ως γέφυρα ανάμεσα σε συμβολικές μεθόδους της τεχνητής νοημοσύνης και των διαφορικών εξισώσεων που χρησιμοποιούνται στην φυσική και την μηχανική. Ο Judea Pearl (1988, 2000), αφού ανέπτυξε τεχνικές για την εφαρμογή στατιστικής και πιθανότητας στην τεχνητή νοημοσύνη, εισήγαγε τα δίκτυα πίστης (belief networks), τα οποία είναι αιτιολογικά δίκτυα (causal networks) με συνδέσεις που έχουν σημειωμένες πιθανότητες.

Διαφορετικές μεθόδοι συλλογισμού μπορούν να έχουν εφαρμογή στο ίδιο βασικό γράφημα, μερικές φορές με σχόλια για να υποδεικνύουν τιμές αλήθειας ή πιθανότητας.
Παρακάτω ακολουθούν δύο από τις κυριότερες προσεγγίσεις:

- **Λογική.** Μεθόδοι λογικού συμπερασμού χρησιμοποιούνται στα συστήματα διατήρησης της αλήθειας (truth –maintenance systems - TMS). Ένα TMS θα άρχιζε από τους κόμβους των οποίων οι τιμές αλήθειας είναι γνωστές και έπειτα θα τις διαδώσει στο υπόλοιπο δίκτυο. Όσον αφορά την περίπτωση με το άτομο που γλιστρήσε στο γρασίδι, θα άρχιζε με την τιμή αλήθειας για το γεγονός ότι το γρασίδι είναι ολισθηρό και θα πήγαινε προς τα πίσω. Εναλλακτικά, ένα σύστημα TMS θα μπορούσε να ξεκινήσει με το γεγονός ότι τώρα είναι βροχερή περίοδος και θα προχωρούσε προς τα εμπρός. Με συνδυασμούς συλλογισμού προς τα εμπρός και προς τα πίσω το TMS διαδίδει τιμές αλήθειας στους κόμβους των οποίων η τιμή αλήθειας είναι άγνωστη. Εκτός από το συμπέρασμα σε νέα πληροφορία, ένα TMS μπορει να χρησιμοποιηθεί για να επιβεβαιώσει τη συνέπεια, να ψάξει συγκρούσεις, ή να βρει περιοχές όπου οι αναμενόμενες συνεπαγωγές δεν υπάρχουν. Οταν βρεθούν συγκρούσεις, η δομή του δικτύου μπορεί να μετατραπεί προσθέτοντας ή διαγράφοντας κόμβους. Το αποτέλεσμα είναι ένα είδος μη μονοτονικού συλλογισμού που ονομάζεται αναθεώρηση πίστης.

- **Πιθανότητα.** Τα περισσότερα χαρακτηριστικά των συλλογισμών προς τα εμπρός και προς τα πίσω που χρησιμοποιούνται σε ένα TMS μπορούν να υιοθετηθούν σε μια πιθανοτική προσέγγιση, εφόσον η αλήθεια θεωρηθεί σαν πιθανότητα με τιμή 1.0 και το ψέμα ως 0.0. Το συνεχές διάστημα πιθανοτήτων από 1.0 εως 0.0, όμως, απαιτεί πιο έξυπνη προσέγγιση και μεγαλύτερη πολυπλοκότητα στους υπολογισμούς. Η πιο λεπτομερής δουλειά πιθανοτικού συλλογισμού σε δίκτυο πίστης ή αιτιολογικό έχει γίνει από τον Pearl (2000). Για το παράδειγμα στο Σχήμα 10, η δίτιμη προσέγγιση αλήθεια – ψέμα είναι απλώς μια πρόχειρη προσέγγιση, αφού δεν βρέχει κάθε μέρα σε μια βροχερή περίοδο και ο ψεκαστήρας δεν χρησιμοποιείται απαραίτητα σε μια περίοδο που δεν βρέχει. Ο Pearl ανέλυσε διάφορες τεχνικές για την εφαρμογή στατιστικής του Bayes για την παραγωγή ενός αιτιολογικού δικτύου από την παρατήρηση δεδομένων και τον συλλογισμό πάνω σε αυτά.

Στα δύο παραπάνω συστήματα που περιγράψαμε, βασισμένα στη λογική και την πιθανότητα, η συσχετιστική πληροφορία που χρησιμοποιήθηκε για την εξαγωγή συνεπαγωγών, αγνοείται από τις διαδικασίες εξαγωγής συμπερασμάτων. Ο Doyle ανέπτυξε το πρώτο TMS εξάγοντας ένα υπογράφημα από συνεπαγωγές από κανόνες ενός έμπειρου συστήματος. Οι Martins και Shapiro (1988) εξήγαγαν ένα TMS από το SNePS κάνοντας ανάλυση μόνο στις δίτιμες (Boolean) συνδέσεις που συνδέουν προτασιακούς κόμβους. Παρόμοιες τεχνικές θα μπορούσαν να εφαρμοστούν και σε άλλα προτασιακά δίκτυα για την εξαγωγή ενός υπογραφήματος συνεπαγωγής όπου θα μπορούσε να αναλύθει από μεθόδους λογικής ή πιθανότητας.

Αν και τα δίκτυα συνεπαγωγής δίνουν έμφαση στην συνεπαγωγή, είναι ικανά να εκφράσουν όλες τις δίτιμες (Boolean) συνδέσεις επιτρέποντας μια σύζευξη εισόδων σε ένα προτασιακό κόμβο και μία διάζευξη εξόδων. Ο Gerhard Gentzen (1935) έδειξε πως μια συλλογή από συνεπαγωγές σε αυτήν την μορφή θα μπορούσε να εκφράσει όλη την προτασιακή λογική. Παρακάτω ακολουθεί η γενική μορφή μιας συνεπαγωγής γραμμένης στην προτασιακή μορφή του Gentzen: $p_1,...,p_n \Rightarrow q_1,...,q_m$. Τα p ονομάζονται πρόγονοι της συνεπαγωγής και τα q

συμπεράσματα. Ο γενικός κανόνας modus ponens δηλώνει πως όταν όλοι οι πρόγονοι είναι αληθείς τότε τουλάχιστον ένα από τα συμπεράσματα πρέπει να είναι αληθές. Σαν αποτέλεσμα, τα κόμματα έχουν το ρόλο του τελεστή ΚΑΙ, ανάμεσα στους προγόνους και τον ρόλο του τελεστή Ή ανάμεσα στα συμπεράσματα. Το αρχικό TMS του Doyle επέτρεπε μόνον έναν όρο στα συμπεράσματα. Η μορφή που έπαιρνε η συνεπαγωγή, η οποία λέγεται λογική Horn, χρησιμοποιείται ευρέως στα έμπειρα συστήματα. Για την υποστήριξη της πλήρους προτασιακής λογικής, οι μετέπειτα εκδόσεις του TMS γενικεύτηκαν, ώστε να επιτρέπουν πολλαπλούς τελεστές Ή στα συμπεράσματα [11].

3.2.3 Εκτελέσιμα δίκτυα

Τα εκτελέσιμα σημασιολογικά δίκτυα περιέχουν μηχανισμούς που μπορούν να αλλάξουν το ίδιο το δίκτυο. Οι εκτελέσιμοι μηχανισμοί διαχωρίζονται από δίκτυα που είναι στατικές δομές δεδομένων, που μπορούν να αλλάξουν μόνο μέσω ενός εξωτερικού ως προς το δίκτυο, προγράμματος. Υπάρχουν τρία είδη μηχανισμών που χρησιμοποιούνται συνήθως στα εκτελέσιμα σημασιολογικά δίκτυα:

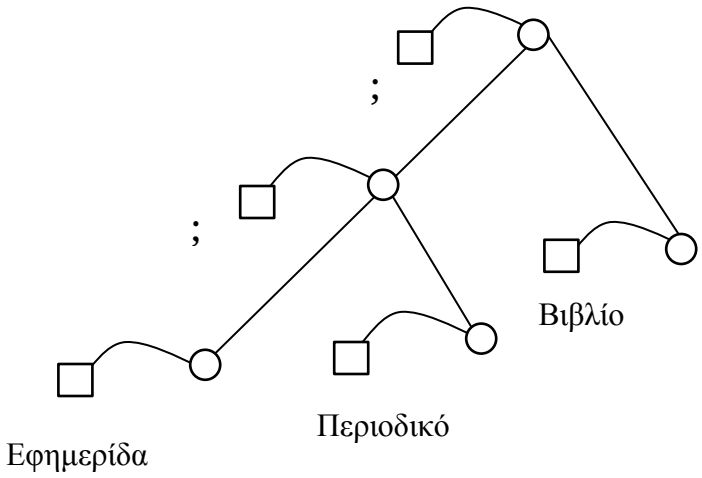
1. **Δίκτυα μετάδοσης μηνύματος (Message passing networks)** που μπορούν να περάσουν πληροφορία από τον ένα κόμβο στον άλλο. Για κάποια δίκτυα, η πληροφορία μπορεί να αποτελείται από ένα bit, που ονομάζεται σημάδι, έμβλημα ή σκανδάλη, για άλλα δίκτυα μπορεί να είναι ένα αριθμητικό βάρος ή ένα αυθαίρετα μεγάλο μήνυμα.

2. **Προσαρτημένες διαδικασίες (Attached procedures)** είναι προγράμματα που περιέχονται ή συσχετίζονται με ένα κόμβο και εκτελούν κάποιο κώδικα ή κανουν κάποιους υπολογισμούς με τις πληροφορίες σε ένα κόμβο ή σε ένα γειτονικό κόμβο.

3. **Μετασχηματισμοί γραφημάτων (Graph transformations)** που συνδυάζουν γραφήματα, τα τροποποιούν ή τα διασπούν σε μικρότερα γραφήματα. Σε τυπικά θεωρήματα απόδειξης τέτοιοι μετασχμηματισμοί εκτελούνται από ένα πρόγραμμα εξωτερικό ως προς τα γραφήματα. Όταν ενεργοποιούνται από τα ίδια τα γραφήματα, συμπεριφέρονται σα χημικές εξισώσεις που συνδυάζουν ή διασπούν μόρια.

Αυτοί οι τρείς μηχανισμοί μπορούν να συνδυαστούν με διάφορους τρόπους. Τα μηνύματα που περνούν από κόμβο σε κόμβο μπορούν να επεξεργαστούν από διαδικασίες που είναι συσχετισμένες με αυτούς τους κόμβους, όπως και οι μετασχηματισμοί γραφημάτων μπορούν να ενεργοποιηθούν από μηνύματα που εμφανίζονται σε κάποιους από τους κόμβους.

Μια σημαντική τάξη εκτελέσιμων δικτύων εμπνεύστηκε από τη δουλειά του ψυχολόγου Otto Selz (1913, 1922), ο οποίος δεν ήταν ικανοποιημένος από τις μη κατευθυντικές συσχετιστικές θεωρίες που υπήρχαν τότε. Σαν εναλλακτική, ο Selz πρότεινε τη σχηματική πρόβλεψη, ως μια κατευθυνόμενη από τον στόχο μέθοδο συγκέντρωσης της σκέψης, με λειτουργία την συμπλήρωση άδειων θέσεων σε μια πατέντα ή σχήμα. Στο παρακάτω Σχήμα φαίνεται ένα παράδειγμα ενός σχήματος του Selz όπου ζήτησε από τους δοκιμαστές να το συμπληρώσουν ενώ ο ίδιος συμπλήρωνε τα λεκτικά πρωτόκολλά τους [11].

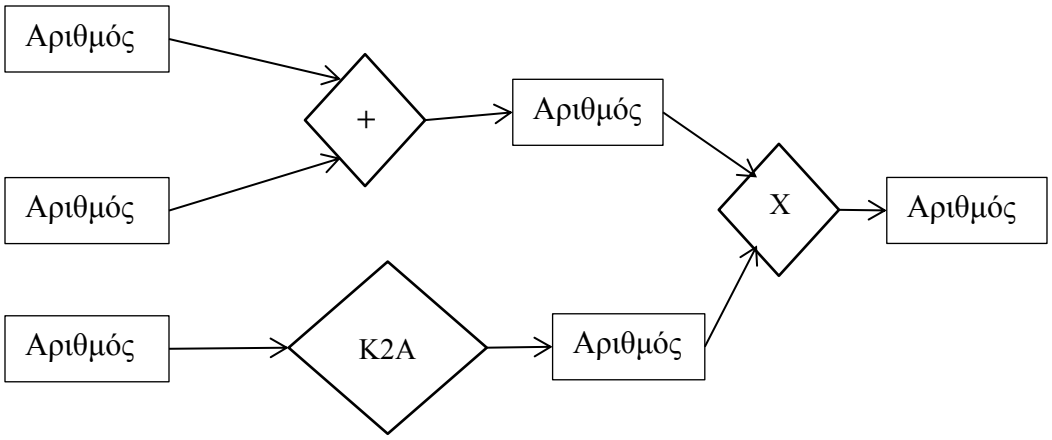


Σχήμα 10. Σχήμα που χρησιμοποίησε ο Selz στα πειράματά του.

Οι αναμενόμενες απαντήσεις στις άδειες θέσεις στο Σχήμα 10 είναι οι υπέρτυποι των λέξεων στους από κάτω κόμβους. Ο υπέρτυπος της εφημερίδας και του περιοδικού είναι «Περιοδικός» και ο υπέρτυπος του βιβλίου είναι «Έκδοση».

Η ομοιότητα ανάμεσα στη μέθοδο σχηματικής πρόβλεψης του Selz και τις κατευθυνόμενες από τον στόχο μεθόδους στην τεχνητή νοημοσύνη δεν είναι τυχαία. Δύο από τους πρωτεργάτες της τεχνητής νοημοσύνης, ο Herbert Simon και ο Allen Newell, έμαθαν για τις θεωρίες του Selz από έναν επισκέπτη που είχαν, τον ψυχολόγο και σκακιστή Adriaan de Groot (Simon 1981). Στην ανάλυσή του για το παίξιμο σκακιού, ο de Groot (1965) εφάρμοσε τις θεωρίες και τις μεθόδους ανάλυσης πρωτοκόλλου του Selz στις λεκτικές αναφορές των σκακιστών, από αρχάριους μέχρι επαγγελματίες. Οι Newel και Simon υιοθέτησαν την μέθοδο του Selz για ανάλυση πρωτοκόλλου στη μελέτη τους για ανθρώπινη επίλυση προβλημάτων. Ο μαθητής τους Ross Quillian (1966), συνδύασε τα δίκτυα του Selz με τα σημασιολογικά δίκτυα που χρησιμοποιούνταν στη μηχανική μετάφραση. Η πιο σημαντική καινοτομία του Quillian ήταν ο αλγόριθμος περάσματος του σημαδιού για διαδιδόμενες ενεργοποιήσεις, που ήταν μια μέθοδος για αναζήτηση σε νευρωνικά ή σημασιολογικά δίκτυα, ο οποίος υιοθετήθηκε σε μετέπειτα συστήματα όπως το NETL του Scott Fahlman (1979) και τους μαζικά παράλληλους αλγορίθμους του Hendler (1987, 1992) και Shastri (1991, 1992).

Τα πιο απλά δίκτυα με συσχετισμένες διαδικασίες είναι τα γραφήματα ροής πληροφορίας,οπού περιέχουν παθητικούς κόμβους που κρατούν την πληροφορία και ενεργούς κόμβους που παίρνουν την πληροφορία από κόμβους εισόδου και στέλνουν τα αποτελέσματα στους κόμβους εξόδου. Στο παρακάτω Σχήμα δίνεται ένα γράφημα ροής πληροφορίας όπου τα ορθογώνια αντιπροσωπεύουν τους παθητικούς κόμβους και τα διαμάντια τους ενεργητικούς. Οι ετικέτες στα ορθογώνια υποδεικνύουν τον τύπο δεδομένων (αριθμό ή κείμενο) ενώ οι ετικέτες στα διαμάντια το όνομα της συνάρτησης (+, x, ή μετατροπή από κείμενο σε αριθμό).



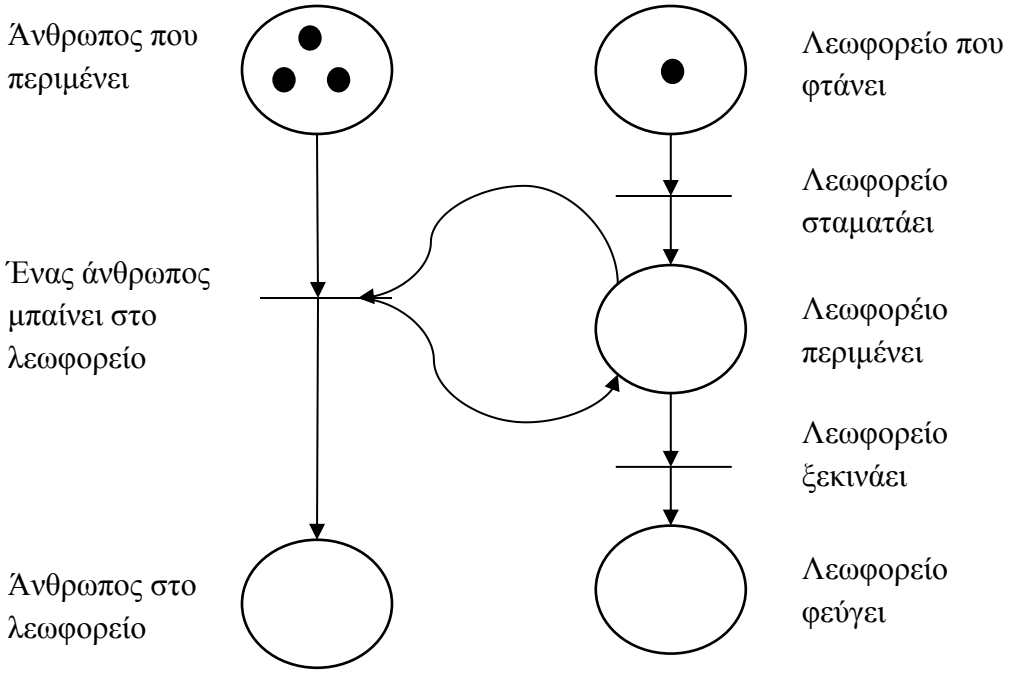
Σχήμα 11. Γράφημα ροής πληροφορίας

Για αριθμητικούς υπολογισμούς, τα διαγράμματα ροής της πληροφορίας δεν έχουν κάποιο πλεονέκτημα απέναντι στα αλγεβρικά σχήματα των συνηθών γλωσσών προγραμματισμού. Για παράδειγμα το σχήμα 11 θα είχε την ακόλουθη μορφή :

X = (A + B) * K2A(M)

Τα γραφικά σχήματα χρησιμοποιούνται συχνά σε ένα ενσωματωμένο περιβάλλον ανάπτυξης (IDE) για να συνδέουν πολλπαπλά προγράμματα και να σχηματίσουν ένα πλήρες σύστημα. Όταν τα γραφήματα ροής πληροφορίας συμπληρώνονται με μια γραφική μέθοδο για τον προσδιορισμό συνθηκών, όπως η if-then-else, και κάποιο τρόπο προσδιορισμού αναδρομικών συναρτήσεων, μπορούν να σχηματίσουν μια πλήρη γλώσσα προγραμματισμού, όμοια με τις γλώσσες συναρτησιακού προγραμματισμού.

Τα δίκτυα Petri, τα οποία εισήχθησαν από τον Carl Adam Petri (1962), είναι ο πλέον πιο χρησιμοποιούμενος φορμαλισμός που συνδυάζει το πέρασμα σημαδιού με διαδικασίες. Όπως τα διαγράμματα ροής πληροφορίας, τα δίκτυα Petri έχουν παθητικούς κόμβους που λέγονται μέρη (places) και ενεργούς κόμβους που λέγονται περάσματα (transitions). Επιπλέον, έχουν ένα σύνολο από κανόνες για σημείωση των μερών με τελείες, που λέγονται εμβλήματα (tokens) όπως και για εκτέλεση ή πυροδότηση των περασμάτων. Για την αναπαράσταση της ροής των εμβλημάτων, το παρακάτω σχήμα δείχνει ένα δίκτυο Petri για μια στάση λεωφορείου όπου τρία βλήματα αναπαριστούν ανθρώπους που περιμένουν και ένα έμβλημα το λεωφορείο που φτάνει.



Σχήμα 12. Δίκτυο Petri για μια στάση λεωφορείου

Στην πάνω αριστερή γωνία του Σχήματος 12, κάθε ένα από τα εμβλήματα αναπαριστά ένα άνθρωπο να περιμένει στην στάση του λεωφορείου. Το έμβλημα στην πάνω δεξιά γωνία αναπαριστά το λεωφορείο που φτάνει. Η μετάβαση με ετικέτα «Λεωφορείο σταματάει» αναπαριστά ένα γεγονός που πυροδοτείται αφαιρώντας ένα έμβλημα από το μέρος «Λεωφορείο που φτάνει» και προσθέτοντας ένα έμβλημα στο μέρος «Λεωφορείο περιμένει». Όταν το λεωφορείο περιμένει, η μετάβαση με ετικέτα «Ένας άνθρωπος που μπαίνει στο λεωφορείο» ενεργοποιείται επειδή έχει τουλάχιστον ένα έμβλημα σε κάθε μία από τις δύο εισόδους της. Πυροδοτείται, αφαιρώντας πρώτα ένα έμβλημα και από τις δύο εισόδους της και τοποθετώντας από ένα έμβλημα και στις δύο εξόδους της. Για όσο το λεωφορείο περιμένει και υπάρχουν ακόμη άνθρωποι που περιμένουν η μετάβαση θα συνεχίσει να πυροδοτείται, σταματάει όταν δεν υπάρχουν άλλοι άνθρωποι που περιμένουν ή όταν πυροδοτηθεί η μετάβαση «Το λεωφορείο ξεκινάει» μετακινώντας το έμβλημα στη μετάβαση «Λεωφορείο φεύγει».

Κάθε μέρος (κύκλος στο Σχήμα 12) σε ένα δίκτυο Petri αναπαριστά μια προϋπόθεση για τις μεταβάσεις που το χρησιμοποιούν ως είσοδο και ένα αποτέλεσμα για τις μεταβάσεις που το χρησιμοποιούν ως έξοδο. Ένα έμβλημα σε ένα μέρος σημαίνει ότι η αντίστοιχη συνθήκη είναι αληθής. Αφαιρώντας ένα έμβλημα από κάθε είσοδο, η πυροδότηση αφαιρεί και τις εγγραφές των προϋποθέσεων. Προσθέτοντας ένα έμβλημα σε κάθε έξοδο η πυροδότηση δηλώνει ότι η κάθε προϋπόθεση έγινε αληθής. Τα δίκτυα Petri μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουν την επεξεργασία που γίνεται σε έναν υπολογιστή στο υλικό ή στο λογισμικό μέρος. Είναι ιδιαίτερα χρήσιμα για το σχεδιασμό και την μοντελοποίηση κατανεμημένων παράλληλων επεξεργασιών. Στο Σχήμα 12, κάθε έμβλημα αναπαριστά πληροφορία μεγέθους ενός bit. Σε μια επέκταση, που ονομάζεται χρωματισμένο δίκτυο Petri, μπορούν να

συσχετιστούν αυθαίρετα μεγάλες ποσότητες πληροφορίας με κάθε έμβλημα (Jensen 1992). Με τέτοιες επεκτάσεις, τα δίκτυα Petri μπορούν να αναπαραστήσουν αυθαίρετα, πολλά γραφήματα ροής της πληροφορίας που τρέχουν παράλληλα ή να εξομοιώσουν τους δίαφορους αλγόριμους περάσματος συμβόλου που χρησιμοποιούνται στα σημασιολογικά δίκτυα στην κατά Quillian παράδοση.

Παρόλο που τα γραφήματα ροής της πληροφορίας και τα δίκτυα Petri δεν ονομάζονται συνήθως σημασιολογικά δίκτυα, παρόμοιες τεχνικές έχουν υλοποιηθεί στα σημασιολογικά δίκτυα με διαδικασίες. Στο πανεπιστήμιο του Toronto, ο Γιάννης Μυλόπουλος και οι μαθητές και συνεργάτες του έχουν υλοποιήσει μια σειρά από σημασιολογικά δίκτυα με συσχετισμένες ρουτίνες [4]. Τα συστήματά τους ενσωματώνουν δίκτυα ορισμού για τον ορισμό τάξεων, δίκτυα προσθήκης για δήλωση γεγονότων και ρουτίνες παρόμοιες με τις γλώσσες αντικειμενοστραφούς προγραμματισμού. Για τα εννοιολογικά γραφήματα, ο Sowa (1976, 1984) επέτρεψε μερικούς συσχετιστικούς κόμβους να αντικατασταθούν από κόμβους που δρούν ή αλλιώς «δράστες». Οι δράστες αυτοί, είναι συναρτήσεις οι οποίες σχηματίζουν το ισοδύναμο ενός γραφήματος ροής πληροφορίας.

Η τρίτη μέθοδος για την κατασκευή εκτελέσιμων δικτύων είναι να αφεθούν να μεγαλώνουν και να αλλάζουν δυναμικά. Ο Peirce και ο Selz μπορούν να θεωρηθούν και εδώ οι πρωτεργάτες για αυτήν την προσέγγιση. Ο Peirce είπε ότι οι συνεπαγωγικές διαδικασίες στα υπαρξιακά γραφήματα μπορούν να θεωρηθούν σαν μια κινούμενη εικόνα της σκέψης. Για την σχηματική πρόβλεψη, ο Selz θεώρησε ότι ένα σχήμα ήταν το αποτέλεσμα μιας νευρωνικής δραστηριότητας που παράγει μια λύση σε ένα πρόβλημα. Επίσημα, οι μετασχηματισμοί στα δίκτυα μπορούν να οριστούν χωρίς αναφορά στους μηχανισμούς που κάνουν τους μετασχηματισμούς. Στα δίκτυα Petri, για παράδειγμα ο ορισμός δηλώνει πως μια μετάβαση μπορεί να πυροδοτηθεί όταν καθένας από τους κόμβους εισόδους της έχει ένα έμβλημα. Ο μηχανισμός που εκτελεί την πυροδότηση μπορεί να είναι εσωτερικός ή εξωτερικός της μετάβασης. Για την υπολογιστική υλοποίηση, ίσως είναι πιο βολικό τα δίκτυα να χρησιμοποιούνται ως παθητικές δομές δεδομένων και να γραφτεί ένα πρόγραμμα που να τα διαχειρίζεται. Για την γνωστική θεωρία ωστόσο, οι μετασχηματισμοί θα μπορούσαν να ερμηνευτούν σαν πράξεις στο δίκτυο όπου ενεργοποιούνται και εκτελούνται από το ίδιο το δίκτυο [11].

3.2.4 Δίκτυα μάθησης

Ένα σύστημα μάθησης, φυσικό ή τεχνητό, ανταποκρίνεται στην καινούργια πληροφορία αλλάζοντας τις εσωτερικές του αναπαραστάσεις με ένα τρόπο που επιτρέπει στο σύστημα να ανταποκρίνεται πιο αποδοτικά στο περιβαλλον του. Τα συστήματα που χρησιμοποιούν αναπαραστάσεις δικτύων μπορούν να αλλάξουν τα δίκτυα με τρεις τρόπους:

1. **Μνήμη μηχανικής αποστήθισης(Rote memory).** Ο πιο απλός τρόπος μάθησης είναι η μετατροπή της νέας πληροφορίας σε ένα δίκτυο και η εισαγωγή της χωρίς άλλες αλλαγές στο υπάρχον δίκτυο.

2. **Αλλαγή βαρών (Changing Weights).** Κάποια δίκτυα αριθμών, όπου ονομάζονται βάρη και είναι συσχετισμένα με τους κόμβους και τα τόξα. Σε ένα συνεπαγωγικό δίκτυο, για

παράδειγμα, αυτά τα βάρη μπορεί να αναπαριστούν πιθανότητες και κάθε εμφάνιση του ίδιου τύπου δικτύου θα αύξανε την πιθανότητα της επανάληψής του.

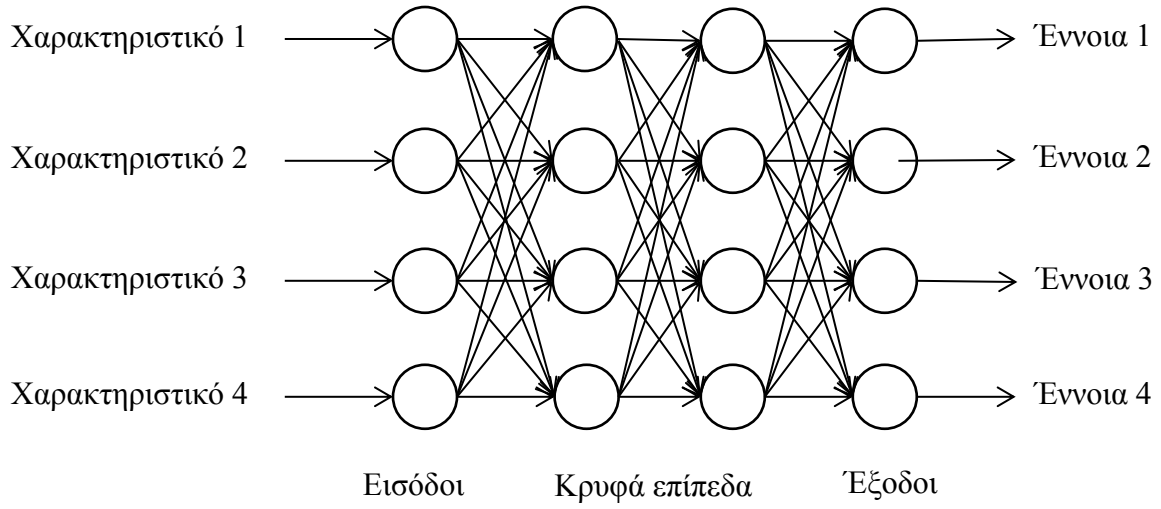
1. **Ανακατασκευή (Restructuring).** Ο πιο πολύπλοκος τρόπος μάθησης, πραγματοποιεί θεμελιώδεις αλλαγές στην δομή του δικτύου. Εφόσον ο αριθμός και τα είδη των δομικών αλλαγών είναι απεριόριστα, η μελέτη και η ταξινόμηση των μεθόδων ανακατασκευής γίνεται η πλέον πιο δύσκολη, αλλά ενδεχομένως να είναι η πιο ανταποδοτική.

Τα συστήματα που μαθαίνουν με αποστήθιση ή που αλλάζουν βάρη μπορούν να σταθούν μόνα τους, ενώ τα συστήματα που μαθαίνουν ανακατασκευάζοντας το δίκτυο χρειάζονται την βοήθεια ενός ή και δύο μεθόδων. Στον σημασιολογικό ιστό (semantic web), η μάθηση με αποστήθιση προσθέτοντας νέους κόμβους και τόξα είναι η βασική μέθοδος για προσθήκη γνώσης.

Εμπορικά, η μνήμη μηχανικής αποστήθισης είναι τεράστιας σημασίας, εφόσον η παγκόσμια οικονομία εξαρτάται από την διατήρηση αρχείων ακριβείας. Για τέτοιες εφαρμογές, η πληροφορία συνήθως αποθηκεύεταιι σε πίνακες, αλλά ο σημασιολογικός ιστός και οι σχετικές μεθόδοι, είναι πιο ευέλικτοι όσον αφορά τις πληροφορίες που δεν είναι τόσο σταθερές όσο ένας πίνακας. Ωστόσο οποιαδήποτε αναπαράσταση μπορέι να μετατραπεί στην άλλη. Για καλύτερη αποδοτικότητα και ευχρηστία, τα περισσότερα συστήματα βάσεων δεδομένων χρησιμοποιούν ευρετήρια για να επιταχύνουν την έρευνα, υποστηρίζουν γλώσσες τύπου ερωτήσεων, όπως η SQL, όπου κάνουν εξαγωγή και συνδυασμό της πληροφορίας για να απαντήσουν. Εφόσον ένα σύστημα μάθησης πρέπει να μπορεί να διακρίνει κοινά χαρακτηριστικά και εξαιρέσεις ανάμεσα σε όμοια παραδείγματα, ένα επιπλέον χαρακτηριστικό είναι απαραίτητο, η ικανότητα να μετράει την ομοιότητα και να ψάχνει την βάση δεδομένων για δίκτυα που είναι όμοια, αλλά όχι ολόιδια για κάθε παράδειγμα.

Τα νευρωνικά δίκτυα είναι μια ευρέως χρησιμοποιούμενη τεχνική μάθησης με αλλαγή βαρών των κόμβων ή των τόξων ενός δικτύου. Το σχήμα 13 δείχνει ένα τυπικό νευρωνικό δίκτυο, όπου οι οι εισόδοι του είναι μια σειρά αριθμών που δηλώνει την σχετική αναλογία των επιλεγμένων χαρακτηριστικών και του οποίου οι εξόδοι είναι ακόμη μια σειρά αριθμών που δηλώνει την πιθανότερη έννοια η οποία χαρακτηρίζεται από τον συνδυασμό των χαρακτηριστικών. Σε μια εφαρμογή, όπως η οπτική αναγνώριση χαρακτήρων, τα χαρακτηριστικά μπορούν να αναπαριστούν γραμμές, καμπύλες και γωνίες ενώ οι έννοιες μπορούν να αναπαριστούν τα γράμματα που έχουν τα χαρακτηριστικά αυτά.

Σε ένα τυπικό νευρωνικό δίκτυο, η δομή των κόμβων και των τόξων είναι σταθερή, οι μόνες αλλαγές που μπορούν να γίνουν είναι στα βάρη των τόξων. Όταν μια νέα είσοδος παρουσιαστεί στο νευρωνικό δίκτυο, τα βάρη στα τόξα συνδυάζονται με τα βάρη στα χαρακτηριστικά εισόδου για τον προσδιορισμό των βαρών στα κρυφά επίπεδα του δικτύου και τελικώς τα βάρη στις εξόδους. Στο στάδιο της μάθησης, λέγεται στο σύστημα εάν τα προβλεπόμενα βάρη είναι σωστά, διάφορες μεθόδοι ανάστροφης διάδοσης (backpropagation) χρησιμοποιούνται για να ρυθμίσουν τα βάρη στα τόξα όπου οδηγούν στο αποτέλεσμα.



Σχήμα 12. Νευρωνικό δίκτυο

Η μνήμη μηχανικής αποστήθισης είναι πλέον η καταλληλοτερη για εφαρμογές όπου απαιτείται ακριβής ανάκτηση των αρχικών δεδομένων, ενώ οι μέθοδοι αλλαγής βαρών είναι οι καλύτερες για την αναγνώριση προτύπων. Για περισσότερο ευέλικτους και δημιουργικούς τρόπους μάθησης, κάποιος τρόπος ανακατασκευής του δικτύου είναι απαραίτητος. Αλλά ο αριθμός των τρόπων για την αναδιοργάνωση του δικτύου είναι τόσο μεγάλος που το μεγαλύτερο μέρος των δυνατοτήτων είναι ανεξερεύνητο. Ακολοθούν μερικά παραδείγματα

1. Ο Patrick Winston (1975) χρησιμοποίησε μία έκδοση σχεσιακών γραφήμάτων για να περιγράψει δομές όπως τόξα και πύργοι. Δίνοντας στο πρόγραμμα του, θετικά και αρνητικά παραδείγματα και των δύο τύπων, γενικοποίησε τα γραφήματα ώστε να αποκομίσει ένα δίκτυο ορισμών για την ταξινόμηση όλων των τύπων που θεωρήθηκαν.

2. Ο Haas και ο Hendrix (1983) ανέπτυξαν το σύστημα NanoKlaus όπου μάθαινε δίκτυα ορισμού με παραδείγματα. Αντίθετα από το σύστημα του Winston, το οποίο απαιτούσε ένα σύνολο από παραδείγματα για να συμπεριλάβει όλα τα χαρακτηριστικά, το NanoKlaus συνέχιζε να ζητάει παραδείγματα μέχρι όλα τα χαρακτηριστικά να είναι αρκετά ώστε να γίνει η διάκριση όλων των τάξεων.

3. Ο George Lendaris (1988) ανέπτυκε ένα σύστημα μάθησης δύο σταδίων, όπου συνδύαζε σχεσιακά γραφήματα με νευρωνικά δίκτυα. Και τα δύο στάδια χρησιμοποιούσαν ένα νευρωνικό δίκτυο με αναστροφή διάδοσης (backpropagation), αλλά στο πρώτο στάδιο, οι εισόδοι ήταν χαρακτηριστικά, ενώ οι εξόδοι έννοιες όπως στο σχήμα 12. Στο δεύτερο στάδιο, κάθε είσοδος αναπαριστούσε ένα σχεσιακό γράφημα κατασκευασμένο από έννοιες οι οποίες είχαν αναγνωριστεί από το πρώτο στάδιο, οι εξόδοι αναπαριστούσαν

πολύπλοκες σκηνές οι οποίες είχαν περιγραφεί από τα γραφήματα. Το σύστημα αυτό των δύο σταδίων, είχε μια σημαντική μείωση στο ρυθμό λάθους και έναν πιο γρήγορο ρυθμό μάθησης σε σύγκριση με τα δίκτυα οπού αντιστοίχιζαν τα χαρακτηριστικά κατευθείαν στις σκηνές [11].

3.2.5 Υβριδικά δίκτυα

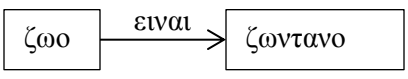
Τα περισσότερα πολύπλοκα συστήματα, συνήθως είναι υβριδικά. Για παράδειγμα, ένα σύστημα βάσης δεδομένων που χρησιμοποιείται για την αποθήκευση των δεδομένων ή μια γλώσσα προγραμματισμού για λεπτομερή υπολογισμό. Οι περισσότερες εφαρμογές στον σημασιολογικό ιστό είναι υβρίδια που συνδυάζουν αρκετές μεθόδους. Ένα παράδειγμα είναι το BabelNet [6], όπου συνδυάζει τεχνικές για τα σημασιολογικά δίκτυα, την μηχανική μετάφραση και τον σημασιολογικό ιστό για να μάθει και να συλλογιστεί με πληροφορίες από πολλές διαφορετικές γλώσσες της Wikipedia. Επίσης, οι περισσότερες αντικειμενοστραφής γλώσσες προγραμματισμού θα μπορούσαν να θεωρηθούν υβρίδια, για παράδειγμα η C++ είναι ένα υβρίδιο της σειριακής γλώσσας C και μιας γλώσσας ορισμού για τον προσδιορισμός κάποιας ιεραρχίας τάξεων ή τύπων [11].

4. ΥΛΟΠΟΙΗΣΗ ΣΗΜΑΣΙΟΛΟΓΙΚΟΥ ΔΙΚΤΥΟΥ ΣΕ LISP

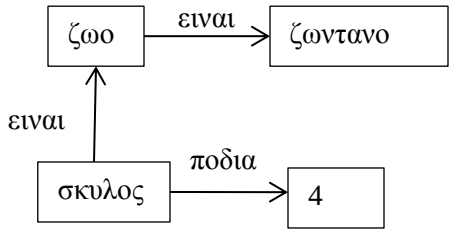
Όπως αναφέραμε στο προηγούμενο κεφάλαιο για να κατασκευάσουμε ένα σημασιολογικό δίκτυο σε οποιαδήποτε γλώσσα προγραμματισμού, χρειαζόμαστε κάποια σχέση τιμής-ιδιότητας. Η Lisp είναι μια βολική γλώσσα για την αναπαράσταση σημασιολογικών δικτύων αφού οι λίστες που χρησιμοποιεί, παρέχουν την δυνατότητα δημιουργίας αυθαίρετων αντικειμένων αλλά και την δέσμευσή τους σε σύμβολα, επιτρέποντας έτσι εύκολα συμπεράσματα καθώς και ορισμό των μεταξύ τους σχέσεων [5].

Στην Lisp αυτή την σχέση την παρέχουν οι λίστες ιδιοτήτων (property lists). Κάθε σύμβολο στην Lisp έχει μια λίστα ιδιοτήτων, στην οποία μπορούμε να αποθηκεύσουμε μια τιμή μαζί με έναν δείκτη, έπειτα χρησιμοποιούμαι τον ανάλογο δείκτη για να βρούμε τις τιμές που θέλουμε μέσα στην λίστα ιδιοτήτων.

Η ανάθεση κάποιας ιδιότητας μαζί με μια τιμη σε ένα σύμβολο γίνεται με τη χρήση των συναρτήσεων SET και GET. Για παράδειγμα για να αναπαραστήσουμε την πληροφορία πως το ‘ζώο’ είναι ‘ζωντανό’ θα γράψουμε: (setf (get ‘ζωο ‘ειναι) ‘ζωντανο) και θα δημιουργηθεί η παρακάτω σχέση :



Αν προσθέσουμε πως ο ‘σκυλος’ είναι ‘ζωο’ και έχει 4 ‘ποδια’ το σημασιολογικό δίκτυο θα επεκτεινόταν ως εξής:



Έτσι πλεον μπορούμε να συμπαιράνουμε πως και ο ‘σκυλος’ θα έχει την ιδιότητα ‘ζωντανο’ διότι κληρονομεί όλες τις ιδιότητες του συμβόλου ‘ζωο’.

Μπορούμε να συνεχίσουμε να επεκτείνουμε το σημασιολογικό δίκτυο με την προσθήκη νέων σχέσεων, αλλά πρώτα θα απλοποιήσουμε λίγο την διαδικασία προσθήκης δημιουργόντας μια συνάρτηση για την προσθήκη σχέσεων.

```
(defun add (symbol property value)
  (setf (get symbol property) value))

(add ‘ζωο ‘ειναι ‘ζωντανο)
(add ‘σκυλος ‘ειναι ‘ζωο)
(add ‘σκυλος ‘καλυμμα ‘δερμα)
(add ‘σκυλος ‘ποδια ‘4)
(add ‘σκυλος ‘ταξιδι ‘περπάτημα)
(add ‘σκυλος ‘ηχος ‘γαβγισμα)
(add ‘λαμπραντορ ‘ειναι ‘σκυλος)
(add ‘ροκυ ‘ειναι ‘λαμπραντορ)
(add ‘ροκυ ‘χρωμα ‘καφε)
(add ‘πουλι ‘ειναι ‘ζωο)
(add ‘καναρινι ‘ειναι ‘πουλι)
(add ‘καναρινι ‘ηχος ‘τραγουδι)
(add ‘καναρινι ‘χρωμα ‘κιτρινο)
(add ‘τουιτι ‘ειναι ‘καναρινι)
```

Για να έχει μια πιο ολοκληρωμένη λειτουργικότητα το σημασιολογικό δίκτυό μας, θα χρειαστεί και μια συνάρτηση για διαγραφή σχέσεων.

```
(defun remove-property (symbol property)
  (remprop symbol name)
```

Η Lisp έχει ενσωματωμένη την συνάρτηση «describe», η οποία αφού πάρει σαν παράμετρο ένα σύμβολο θα μας επιστρέψει όλες τις ιδιότητές του.

Το (describe ‘σκυλος) δηλαδή θα επιστρέψει :

ΗΧΟς -> ΓΑΒΓΙΣΜΑ
ΤΑΞΙΔΙ -> ΠΕΡΠΆΤΗΜΑ
ΠΟΔΙΑ -> 4
ΚΑΛΥΜΜΑ -> ΔΕΡΜΑ
ΕΙΝΑΙ -> ΖΩΟ

Η κληρονομικότητα χαρακτηριστικών είναι μια από τις σημαντικές λειτουργίες των σημασιολογικών δικτύων, για να την χρησιμοποιήσουμε χρειαζόμαστε απλά κάποια συνάρτηση που να υλοποιεί την συγκεκριμένη διαδικασία, μια τέτοια συνάρτηση έχουν δημιουργήσει οι Luger και Stubblefield [5].

```
(defun inherit-get (symbol property)
  (or (get symbol property)
     (get-from-parents (get symbol 'ειναι) property)))

(defun get-from-parents (parents property)
  (cond ((null parents) nil)
          ((atom parents) (inherit-get parents property))
          (t (or (get-from-parents (car parents) property)
               (get-from-parents (cdr parents) property)))))
```

Η συνάρτηση **inherit-get** πρώτα προσπαθεί να ανακτήσει την ιδιότητα από το σύμβολο που δώσαμε σαν παράμετρο και εάν αυτό αποτύχει καλεί την **get-from-parents** η οποία θα καλεί την ανάκτηση της ιδιότητα από τον επόμενο γονέα έως ότου τα καταφέρει ή δεν υπάρχει επόμενος γονέας.

Για παράδειγμα εάν καλέσουμε την **inherit-get** με παραμέτρους ‘ροκυ ‘καλυμμα θα προσπαθήσει να ανακτήσει το «κάλλυμα» από το σύμβολο «ροκυ», οταν αποτύχει θα το ζητήσει από το «λαμπραντορ» οπού θα αποτύχει ξανα και θα έχει επιτυχία μόνο όταν φτάσει στο επίπεδο «σκυλος», αφου εκεί ορίσαμε το «κάλλυμα».

```
0: (INHERIT-GET ΡΟΚΥ ΚΑΛΥΜΜΑ)
  1: (INHERIT-GET ΛΑΜΠΡΑΝΤΟΡ ΚΑΛΥΜΜΑ)
    2: (INHERIT-GET ΣΚΥΛΟς ΚΑΛΥΜΜΑ)
    2: INHERIT-GET returned ΔΕΡΜΑ
  1: INHERIT-GET returned ΔΕΡΜΑ
 0: INHERIT-GET returned ΔΕΡΜΑ
ΔΕΡΜΑ
```

Εάν ρωτήσουμε το δίκτυο πόσα πόδια έχει ο «ροκυ», θα ανατρέξει στο «λαμπραντορ» και έπειτα στο «σκυλος» όπου και θα επιστρέψει τον αριθμό 4.

```
 0: (INHERIT-GET ΡΟΚΥ ΠΟΔΙΑ)
  1: (INHERIT-GET ΛΑΜΠΡΑΝΤΟΡ ΠΟΔΙΑ)
    2: (INHERIT-GET ΣΚΥΛΟς ΠΟΔΙΑ)
```

```
    2: INHERIT-GET returned 4
  1: INHERIT-GET returned 4
 0: INHERIT-GET returned 4
4
```

Όμως, εάν εμείς θέλουμε να ορίσουμε πως ο «ροκυ», έχει μόνο 3 πόδια λόγω κάποιου ατυχήματος, μπορούμε να το κάνουμε;

```
(add ‘ροκυ ‘ποδια ‘3)
```

Η απάντηση είναι ναι, διότι αυτή τη φορά θα είναι δυνατόν να ανακτηθεί η ιδιότητα χωρίς να χρειαστεί να γίνει ερώτηση στους υπέρτυπους του «ροκυ». Έτσι όταν ξαναρωτήσουμε το δίκτυο πόσα πόδια έχει ο «ροκυ» θα επιστρέψει 3.

```
0: (INHERIT-GET ΡΟΚΥ ΠΟΔΙΑ)
 0: INHERIT-GET returned 3
3
```

Κατ’ αυτό τον τρόπο, βλέπουμε πως είναι δυνατόν όταν θέλουμε, να ορίζουμε κάποια γενικά χαρακτηριστικά στα διάφορα είδη που επιχειρούμε να περιγράψουμε, τα οποία και κληρονομούνται σε όλους τους υπότυπούς τους. Ενώ παράλληλα, μπορούμε να ορίζουμε και συγκεκριμένα χαρακτηριστικά λόγω κάποιας εξαίρεσης, σε κάποιον υπότυπο, ο οποίος έτσι θα χάνει την κλήρονομικότητα από την συγκεκριμένη ιδιότητα του υπέρτυπού του.

4.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Μειονεκτήματα

Σε γενικές γραμμές τα σημασιολογικά δίκτυα μπορούν να αναπαραστήσουν με μια καλή ακρίβεια τα γεγονότα του πραγματικού κόσμου, όμως τι γίνεται όταν επιχειρήσουμε να περιγράψουμε μια ιδέα ή μια πεποίθηση; Γρήγορα καταλαβαίνουμε πως δεν υπάρχουν μοναδικοί τρόποι για να πραγματοποιηθεί μια τέτοια αναπαράσταση και αυτό δημιουργεί προβλήματα όσον αφορά τη δομή των σημασιολογικών δικτύων.

Ακόμη μια δύσκολη ερώτηση για τα σημασιολογικά δίκτυα είναι αν θα μπορούσαν να αναπαραστήσουν το πέρασμα του χρόνου (χρονική συλλογιστική) σε ικανοποιητικό βαθμό.

Τέλος ένα σημαντικό μειονέκτημα των σημασιολογικών δικτύων, είναι πως μπορούν να αναπαραστήσουν τις διάφορες σχέσεις μεταξύ των κόμβων, όπως και παρατηρήσαμε, μόνο δυαδικά (binary), αυτή η ιδιότητα από μόνη της είναι σαφώς περιοστική.

Πλεονεκτήματα

Σύμφωνα με το παράδειγμα σημασιολογικού δικτύου όπου κατασκευάσαμε, μπορούμε να ορίσουμε προεπιλεγμένες τιμές σε κάποιες κατηγορίες, οι οποίες τιμές θα κληρονομούνται

στις υπο-κατηγορίες τους. Αλλά μπορούμε να ορίσουμε και κάποια συγκεκριμένη τιμή σε εξειδικευμένες κατηγορίες οι οποίες έτσι θα σταματούν να κληρονομούν την τιμή από την υπερ-κατηγορία.

Τα σημασιολογικά δίκτυα εξάγουν συμπεράσματα με έναν αρκετά διαφανή τρόπο και είναι εύκολα στην κατανόηση, χωρίς να χρειάζεται κάποιος να έχει εξειδικευμένες γνώσεις στο συγκεκριμένο πεδίο το οποίο περιγράφουν. Επίσης είναι εύκολο να μεταφραστούν και στη γλώσσα PROLOG.

Η σχέση που υπάρχει μεταξύ κόμβου και συνδέσμου στα σημασιολογικά δίκτυα, συλλαμβάνει πολύ καλα τη σχέση μεταξύ των συμβόλων και των δεικτών στον συμβολικό υπολογισμό.

Τέλος αξίζει να ειπωθεί πως τα σημασιολογικα δίκτυα έχουν αρκετά κοινά με την ίδια καθέ αυτού ψυχολογία της μνήμης.

Θα μπορούσαμε να συνεχίσουμε να επεκτείνουμε το παραπάνω σημασιολογικό δίκτυο έως ότου καταφέρουμε να περιγράψουμε τις έννοιες που θέλουμε σε έναν ικανοποιητικό βαθμό και στην συνέχεια με επιπλέον συναρτήσεις, μπορούμε να κάνουμε νέες ερωτήσεις στο δίκτυο αντλόντας νέα συμπεράσματα, δίνοντάς στο ίδιο δίκτυο διαφορετικό νόημα.

Όταν το σημασιολογικό δίκτυο καταφέρει να επεκταθεί αρκετά, θα μπορεί να εκλαμβάνεται και σαν την αναπαράσταση ενός «μικρού κόσμου», τον οποίο θα μπορούσε να κατανοήσει σε κάποιο βαθμό και μια μηχανή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Davis R., Shrobe H., & Szolovits P., "*What is a Knowledge Representation?*", AI Magazine, 14(1), pp.17-33, 1993

[2] Graham, P., "*ANSI Common Lisp*" Prentice Hall, 1996

[3] Graham, P., "*On Lisp*" Prentice Hall, 1993

[4]Levesque, H., & Mylopoulos J., "*A procedural semantics for semantic networks*", 1979

[5] Luger G. F., Stubblefield W. A., "*AI Algorithms, Data structures, and Idioms in Prolog, Lisp, and Java*" Pearson Education, 2009

[6] Navigli R., & Ponzetto S. P., "*BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network*", Artificial Intelligence 193, 217-250. 2012 Available at http://wwwusers.di.uniroma1.it/~navigli/pubs/AIJ_2012_Navigli_Ponzetto.pdf

[7] Raymond E. S., "*How To Become A Hacker"* 2001, Available at http://www.catb.org/esr/faqs/hacker-howto.html

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

[9] Russel S. J., & Norvig P., "*Τεχνητή νοημοσύνη μια σύγχρονη προσέγγιση*" (Άλβας Τ., Καρτσακλής Δ., & Σκουλαρίκης Φ., Μεταφρ.), Κλειδάριθμος (2ND ed.), 2005

[10] Seibel, P., "*Practical Common Lisp*" (3hd ed.) Apress, 2005

[11] Sowa J. F., "*Semantic networks*" 2002, Available at www.jfsowa.com/pubs/semnet.htm

[12] Turing A. M., "*Computing Machinery and Intelligence*" Mind 49: 433-460, 1950, Available at http://www.csee.umbc.edu/courses/471/papers/turing.pdf

http://common-lisp.net/project/slime/

http://www.gnu.org/software/emacs/

http://www.quicklisp.org/beta/

http://www.sbcl.org/