ΣΧΟΛΗ : ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ
ΤΜΗΜΑ: ΤΗΛΕΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
ΘΕΜΑ: ΑΝΑΓΝΩΡΙΣΗ ΣΥΝΤΑΚΤΙΚΑ ΕΓΚΥΡΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ ΜΕ ΤΗ ΒΟΗΘΕΙΑ OCR

ΕΠΙΒΛΕΠΩΝ:
ΤΣΟΥΛΟΣ ΙΩΑΝΝΗΣ

ΣΠΟΥΔΑΣΤΗΣ:
ΒΑΡΒΑΡΑΣ ΙΩΑΝΝΗΣ

2005

Στην οικογένειά μου,

για την ανεξάντλητη αγάπη και υπομονή της.

Το τέλειο δε φτάνεται αλλά και το εφικτό δεν εξαντλείται.
Το διαχρονικό όμως κερδίζεται,
τελικά η πράξη είναι ανάγκη, επιβάλλεται και κρίνεται.

ΕΥΧΑΡΙΣΤΙΕΣ

Στον καθένα από όλους συναδέλφους και φίλους μου, οι οποίοι βοήθησαν στην διαμόρφωση της εργασίας αυτής με τις συμβουλές, την ενθάρρυνση και την άμεση συμμετοχή τους, απευθύνω τις ευχαριστίες μου για το ενδιαφέρον και τις ιδέες τους. Εδώ πρέπει να αναφέρω την βοήθεια και συμπαράσταση του καλού φίλου Δημοσθένη Βάκα.

Ιδιαίτερα θέλω να ευχαριστήσω τον εισηγητή κ. Ιωάννη Τσούλο για την ανεκτίμητη βοήθειά του.

ΠΕΡΙΛΗΨΗ

Πτυχιακή εργασία με αντικείμενο τη σχεδίαση και υλοποίηση συστήματος οπτικής αναγνώρισης χαρακτήρων, με δυνατότητες ταυτόχρονης μεταγλώττισης. Επιχειρείται μια σύντομη περιγραφή των τεχνικών νευρωνικών δικτύων. Στη συνέχεια γίνεται αναλυτική παρουσίαση των πολυεπίπεδων perceptrons (MLP). Γίνεται λεπτομερής παρουσίαση των δομικών στοιχείων και των ιδιοτήτων του προαναφερθέντος μοντέλου καθώς και μια ανάλυση των βασικών αλγόριθμων εκπαίδευσης. Κατόπιν γίνεται εισαγωγή στις βασικές έννοιες των μεταγλωττιστών και ακολουθεί παρουσίαση των χαρακτηριστικών των γλωσσών προγραμματισμού. Ακολουθεί παρουσίαση της φάσης της λεκτικής ανάλυσης, του μεταεργαλείου flex το οποίο χρησιμοποιείται για την δημιουργία λεξιλογικών αναλυτών, της φάσης της συντακτικής ανάλυσης και του μεταεργαλείου bison το οποίο χρησιμοποιείται για την ανάπτυξη συντακτικών αναλυτών. Στο τέλος παρουσιάζονται τα τμήματα της εφαρμογής που αναπτύχθηκε.

ΠΡΟΛΟΓΟΣ

Τα τελευταία χρόνια η επιστήμη της επεξεργασίας δεδομένων, γνώρισε μία τρομερή ανάπτυξη η οποία σε συνδυασμό με άλλες επιστήμες όπως η μικροηλεκτρονική, μας έδωσαν πολλά ισχυρά εργαλεία για την πραγματοποίηση εργασιών που πριν μερικά χρόνια φάνταζαν αδύνατα. Από τις πιο σημαντικές και ίσως η πιο σημαντική είναι ο ηλεκτρονικός υπολογιστής Η/Υ. Παρά τις αδιαμφισβήτητες ικανότητες των Η/Υ όμως, υπάρχει μία διαχωριστική γραμμή ανάμεσα στα προβλήματα που μπορούν να επιλυθούν παραγωγικά με χρήση Η/Υ και σε αυτά που ο ανθρώπινος εγκέφαλος είναι ανυπέρβλητος.

Ο Η/Υ μπορεί να επιλύσει εύκολα προβλήματα που μπορούν να αναλυθούν σε απλές λογικές πράξεις, ενώ ο ανθρώπινος εγκέφαλος είναι αξεπέραστος στην επίλυση προβλημάτων που εμπεριέχουν υψηλό βαθμό απροσδιοριστίας. Η ενσωμάτωση χαρακτηριστικών του ανθρώπινου εγκεφάλου στους Η/Υ, τους προσδίδει ικανότητα επίλυσης προβλημάτων με υψηλό βαθμό απροσδιοριστίας. Τα Τεχνικά Νευρωνικά Δίκτυα ΤΝΔ προσπαθούν να επιλύσουν αυτό το πρόβλημα.

Για να μπορέσουν όμως οι Η/Υ να κάνουν τις απαραίτητες εργασίες για την παραγωγή των κατάλληλων προγραμμάτων, χρειάζεται μια περιγραφή τους που ονομάζεται Πρόγραμμα. Το κάθε πρόγραμμα είναι γραμμένο σε μια κατάλληλη γλώσσα που ονομάζεται γλώσσα προγραμματισμού. Το πρόβλημα είναι ότι οι Η/Υ τη μόνη γλώσσα που αναγνωρίζουν είναι η Γλώσσα Μηχανής, που είναι δυσνόητη και η συγγραφή και συντήρηση μεγάλων προγραμμάτων σε γλώσσα μηχανής είναι σχεδόν αδύνατη. Αυτό το πρόβλημα το επιλύουν οι Μεταγλωττιστές με αυτόματο τρόπο. Επιτρέπουν τη συγγραφή προγραμμάτων σε μια γλώσσα υψηλού επιπέδου κατανοητή από τον άνθρωπο, την οποία μεταφράζουν στη γλώσσα μηχανής του υπολογιστή, ώστε να είναι άμεσα εκτελέσιμη από τον υπολογιστή.

Η εφαρμογή που αναπτύξαμε είναι ένα “scanner” με δυνατότητες μεταγλωττιστή. Έστω ότι θέλουμε μία σελίδα που περιέχει τον κώδικα μιας γλώσσας προγραμματισμού, να την αποθηκεύσουμε στον ηλεκτρονικό μας υπολογιστή και ταυτόχρονα να βρούμε διάφορα λάθη γραμματικά ή συντακτικά. Η εφαρμογή μας προσπαθεί να επιλύσει δύο πολύ σημαντικά προβλήματα.

Το πρώτο είναι η μεταφορά χωρίς λάθη ως προς το σύνολο και την διάταξη των χαρακτήρων του κειμένου στον Η/Υ και η αποθήκευσή του σε μορφή αναγνωρίσιμη και επεξεργάσιμη από τον Η/Υ. Αρχικά με τη βοήθεια ενός "scanner" μεταφέρουμε το κείμενο στον υπολογιστή σε δυαδική μορφή με 0 – 1. Η μεταφορά αυτή γίνεται με ορισμένη μορφή (format). Στη συνέχεια το αρχείο με τη δυαδική αποτύπωση του αρχείου, με τη βοήθεια ΤΝΔ μετατρέπεται σε αρχείο κειμένου το οποίο μπορούμε να το επεξεργαστούμε.

Το δεύτερο είναι η μεταγλώττιση του αρχείου που προέκυψε κατά την πρώτη φάση. Από την πρώτη φάση θα πάρουμε ένα αρχείο κειμένου που περιέχει τον κώδικας που θέλουμε να μεταγλωττίσουμε. Ο μεταγλωττιστής κατασκευάζεται με διάφορους τρόπους, αλλά με τη βοήθεια των μεταεργαλείων Flex και Bison μπορούμε να τον υλοποιήσουμε σχετικά εύκολα.

ΠΕΡΙΕΧΟΜΕΝΑ ΚΑΙ ΟΡΓΑΝΩΣΗ

Η παρούσα εργασία χωρίζεται σε εφτά κεφάλαια. Τα πρώτα έξι παρέχουν στον αναγνώστη ένα στοιχειώδες θεωρητικό υπόβαθρο, προκειμένου να κατανοήσει την σημασία των νευρωνικών δικτύων και των μεταγλωττιστών. Στο έβδομο κεφάλαιο γίνεται παρουσίαση της εργασίας μας και των μερών από τα οποία αποτελείται. Στο τέλος ακολουθεί σε παράρτημα ο κώδικας που χρησιμοποιήθηκε για την παραγωγή του software. Ακολουθεί αναλυτική παρουσίαση των κεφαλαίων.

Στο **κεφάλαιο 1** γίνεται μια εισαγωγή στα νευρωνικά δίκτυα. Προηγείται μια σύντομη ιστορική αναδρομή και ακολουθεί η παρουσίαση των βασικών εννοιών των νευρωνικών δικτύων. Περιγράφονται οι κατηγορίες τους, ο τρόπος λειτουργίας τους, ο τρόπος ανάπτυξής τους και οι εφαρμογές τους.

Στο **κεφάλαιο 2** παρουσιάζονται τα πολυεπίπεδα τεχνικά νευρωνικά δίκτυα. Αρχικά παρουσιάζεται η δομή και ο τρόπος λειτουργίας τους. Στη συνέχεια τα μεγέθη που χρησιμοποιούνται και ο αλγόριθμος λειτουργίας τους. Στο τέλος υπάρχουν πληροφορίες για το μέγεθός τους, τις μεθόδους βελτίωσης του τρόπου λειτουργίας τους και τις εφαρμογές στις οποίες μπορούμε να τα χρησιμοποιήσουμε.

Στο **κεφάλαιο 3** γίνεται εισαγωγή στους μεταγλωττιστές. Αρχικά παρουσιάζονται κάποιες βασικές έννοιες των γλωσσών προγραμματισμού. Στη συνέχεια γίνεται η

παρουσίαση των βασικών εννοιών σχετικά με τους μεταγλωττιστές, τα είδη τους και αναλύεται ο τρόπος ανάπτυξης και παραγωγής τους.

Στο **κεφάλαιο 4** γίνεται εισαγωγή σε έννοιες που σχετίζονται με τις τυπικές γλώσσες που είναι χρήσιμες για την κατασκευή των μεταγλωττιστών: κανονικές γλώσσες και εκφράσεις, πεπερασμένα αυτόματα, γλώσσες και γραμματικές χωρίς συμφραζόμενα, αυτόματα στοίβας και κατηγορικές γραμματικές.

Στο **κεφάλαιο 5** περιγράφεται η φάση της λεκτικής ανάλυσης. Παρουσιάζονται οι λεκτικές μονάδες και ο τρόπος αναγνώρισής τους με τη βοήθεια διαγραμμάτων μετάβασης. Στη συνέχεια περιγράφεται η σχεδίαση και υλοποίηση ενός λεκτικού αναλυτή με βάση τα διαγράμματα μετάβασης. Στο τέλος περιγράφεται η ανάπτυξη ενός λεκτικού αναλυτή με τη βοήθεια του μεταεργαλείου flex.

Στο **κεφάλαιο 6** περιγράφεται η φάση της λεκτικής ανάλυσης. Περιγράφεται ο τρόπος ανάπτυξης των δύο βασικών κατηγοριών συντακτικών αναλυτών, τους αναλυτές από πάνω προς τα κάτω (top - down) και από κάτω προς τα πάνω (bottom - up). Στο τέλος περιγράφεται η ανάπτυξη ενός συντακτικού αναλυτή με τη βοήθεια του μεταεργαλείου bison.

Στο **κεφάλαιο 7** περιγράφεται η εφαρμογή που αναπτύξαμε. Αρχικά παρουσιάζεται το λεξικό και η γραμματική τη γλώσσας, καθώς και τα αρχεία εισόδου στα μεταεργαλεία flex και bison. Στη συνέχεια παρουσιάζεται ο τρόπος με τον οποίο ψηφιοποιήθηκαν τα αρχεία εισόδου στο νευρωνικό και στο τέλος περιγράφεται η δομή και η λειτουργία του νευρωνικού δικτύου.

Στο **παράρτημα Α** αρχικά τοποθετήθηκε ο κώδικας από τα αρχεία εισόδου στο flex, στη συνέχεια ο κώδικας του αρχείου εισόδου στο bison και στο τέλος ο κώδικας του νευρωνικού δικτύου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ.....v

ΠΡΟΛΟΓΟΣ.....vi

ΠΕΡΙΕΧΟΜΕΝΑ ΚΑΙ ΟΡΓΑΝΩΣΗ vii

ΠΕΡΙΕΧΟΜΕΝΑ.....ix

1 ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ.....1

1.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ.....1

1.2 ΓΕΝΙΚΑ.....1

1.3 ΚΑΤΗΓΟΡΙΕΣ ΤΝΔ.....3

1.4 ΕΚΠΑΙΔΕΥΣΗ ΤΩΝ ΤΝΔ.....4

1.5 ΕΛΕΓΧΟΣ ΑΠΟΔΟΣΗΣ ΤΩΝ ΤΝΔ.....6

1.6 ΕΦΑΡΜΟΓΕΣ ΤΩΝ ΤΝΔ.....7

1.7 Η ΤΕΧΝΟΛΟΓΙΑ ΚΑΤΑΣΚΕΥΗΣ ΤΩΝ ΤΝΔ.....**7**

2 ΠΟΛΥΕΠΙΠΕΔΑ PERCEPTRONS (MLP).....9

2.1 ΠΟΛΥΕΠΙΠΕΔΑ PERCEPTRONS (Back error propagation BP).....9

2.1.1 Ορισμός μεγεθών και εκπαίδευση του BP.....11

2.1.2 Αλγόριθμος του λειτουργίας του back – propagation.....11

2.2 ΤΕΡΜΑΤΙΣΜΟΣ ΤΗΣ ΕΚΠΑΙΔΕΥΣΗΣ.....13

2.3 ΤΟ ΜΕΓΕΘΟΣ ΤΟΥ ΤΝΔ.....14

2.4 ΒΕΛΤΙΩΣΗ ΤΟΥ BP ΚΑΙ ΕΥΡΕΤΙΚΟΙ ΚΑΝΟΝΕΣ.....15

2.5 ΕΦΑΡΜΟΓΕΣ ΤΟΥ BP.....16

3 ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ.....17
3.1 ΟΡΙΣΜΟΙ.....17
3.2 ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ.....18
3.3 ΕΙΔΗ ΜΕΤΑΓΛΩΤΤΙΣΤΩΝ ΚΑΙ ΣΥΝΑΦΗ ΕΡΓΑΛΕΙΑ.....19
3.4 ΚΑΤΑΣΚΕΥΗ ΕΝΟΣ ΜΕΤΑΓΛΩΤΤΙΣΤΗ.....20
3.4.1 Ορισμός του προβλήματος.....20
3.4.2 Απαιτήσεις.....21
3.4.3 Φάσεις της μεταγλώττισης.....22
3.4.4 Επιλογές κατά την υλοποίηση.....23
3.4.5 Έλεγχος ορθότητας.....25

4 ΤΥΠΙΚΕΣ ΓΛΩΣΣΕΣ.....26
4.1 ΓΕΝΝΗΤΙΚΑ ΚΑΙ ΑΝΑΓΝΩΡΙΣΤΙΚΑ ΜΟΝΤΕΛΑ.....26
4.1.1 Ιεραρχία Chomsky.....26
4.1.2 Αναγνωριστές.....27
4.2 ΚΑΝΟΝΙΚΕΣ ΓΛΩΣΣΕΣ.....29
4.2.1 Πεπερασμένα Αυτόματα.....30
4.3 ΓΛΩΣΣΕΣ ΧΩΡΙΣ ΣΥΜΦΡΑΖΟΜΕΝΑ.....31
4.3.1 Γραμματικές χωρίς συμφραζόμενα.....31
4.3.2 Αυτόματα στοίβας.....35
4.4 ΚΑΤΗΓΟΡΙΚΕΣ ΓΡΑΜΜΑΤΙΚΕΣ.....36

5 ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ.....38
5.1 ΛΕΚΤΙΚΕΣ ΜΟΝΑΔΕΣ.....38
5.2 ΑΝΑΓΝΩΡΙΣΗ ΛΕΚΤΙΚΩΝ ΜΟΝΑΔΩΝ.....39
5.2.1 Μια γενική μέθοδος.....40

5.2.2 Μια ειδική μέθοδος με βάση τα διαγράμματα μετάβασης.....40
5.3 ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ ΜΕ ΒΑΣΗ ΤΑ ΔΙΑΓΡΑΜΜΑΤΑ ΜΕΤΑΒΑΣΗΣ.....41
5.4 ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ ΜΕ ΤΟ ΜΕΤΑΕΡΓΑΛΕΙΟ FLEX.....43
5.4.1 Περιγραφή κανονικών εκφράσεων στο flex.....45
5.4.2 Ένα παράδειγμα χρήσης του flex.....46
5.4.3 Το περιβάλλον εκτέλεσης του flex.....47

6 ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ.....49
6.1 ΕΙΣΑΓΩΓΗ.....49
6.1.1 Βοηθητικές έννοιες: FIRST και FOLLOW.....50
6.2 ΣΥΝΤΑΚΤΙΚΟΙ ΑΝΑΛΥΤΕΣ ΑΠΟ ΠΑΝΩ ΠΡΟΣ ΤΑ ΚΑΤΩ.....51
6.2.1 Γραμματικές LL(1).....52
6.2.1.1 Αντικατάσταση.....53
6.2.1.2 Αριστερή παραγοντοποίηση.....53
6.2.1.3 Απαλοιφή αριστερής αναδρομής.....53
6.3 ΣΥΝΤΑΚΤΙΚΟΙ ΑΝΑΛΥΤΕΣ ΑΠΟ ΚΑΤΩ ΠΡΟΣ ΤΑ ΠΑΝΩ.....53
6.3.1 Γραμματικές LR(1).....55
6.4 ΣΥΝΤΑΚΤΙΚΟΙ ΑΝΑΛΥΤΕΣ LALR(L) ΜΕ ΤΟ ΜΕΤΑΕΡΓΑΛΕΙΟ BISON.....55
6.4.1 Περιγραφή κανόνων παραγωγής στο bison.....58
6.4.2 Χειρισμός συντακτικών σφαλμάτων στο bison.....59
6.5 ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΕΚΤΕΛΕΣΗΣ ΤΟΥ BISON.....59

7 Η ΕΦΑΡΜΟΓΗ.....61

7.1 Η ΓΛΩΣΣΑ ΕΦΑΡΜΟΓΗΣ……..61
7.2 ΤΟ ΛΕΞΙΚΟ ΚΑΙ ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ FLEX…..61
7.3 Η ΓΡΑΜΜΑΤΙΚΗ ΚΑΙ ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ BISON…….62
7.4 ΨΗΦΙΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ……63
7.5 ΤΟ ΤΕΧΝΗΤΟ ΝΕΥΡΩΝΙΚΟ ΔΙΚΤΥΟ…..64
7.6 ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΥΕΛΕΣΜΑΤΑ……..65

ΠΑΡΑΡΤΗΜΑ……67
Α.1 ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ FLEX……67
Α.2 ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ BISON……69
Α.3 ΤΟ ΝΕΥΡΩΝΙΚΟ ΔΙΚΤΥΟ……71

ΒΙΒΛΙΟΓΡΑΦΙΑ……77

ΔΙΕΥΘΥΝΣΕΙΣ INTERNET……78
ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ……78
ΝΕΥΡΩΝΙΚΑ……78

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ……80

ΚΕΦΑΛΑΙΟ 1

ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

1.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Οι σύγχρονες προσπάθειες για την κατανόηση των δομών και των λειτουργιών του εγκεφάλου άρχισαν πριν 100 περίπου χρόνια ενώ η συστηματοποίηση τους ξεκίνησε με τις εργασίες των McCulloch και Pitts[1] και τις ψυχολογικές θεωρίες του Hebb[2] και λίγο αργότερα τις εργασίες του Rosenblatt (Perceptron)[3] και του Widrow (Adaline)[4].

Στα χρόνια που ακολούθησαν η γνώση κωδικοποιήθηκε σε ένα ξεχωριστό γνωστικό πεδίο, αυτό των Τεχνητών Νευρωνικών Δικτύων (ΤΝΔ). Η αρχική αισιοδοξία μετατράπηκε σε απαισιοδοξία μετά την έκδοση του περίφημου βιβλίου "Perceptrons" των Minsky και Papert[5]. Στην δεκαετία του '70 οι προσπάθειες στον τομέα των ΤΝΔ συνεχίστηκαν με μειωμένους ρυθμούς και το ενδιαφέρον μετατοπίστηκε στην προσέγγιση τύπου "μαύρου κουτιού". Ορισμένοι ερευνητές που συνέχισαν να εργάζονται με ΤΝΔ στην δεκαετία του '70 είναι οι Grossberg[6] και Kohonen[7].

Η αναθέρμανση του γνωστικού πεδίου πραγματοποιήθηκε στις αρχές της δεκαετίας του '80, έχοντας ως σημείο αναφοράς τις εργασίες του Hopfield, με αποτέλεσμα σήμερα να αναφερόμαστε σε μια πραγματική έκρηξη που πυροδοτείται από την προσέγγιση των ΤΝΔ με συγγενείς τεχνολογίες (Γενετικοί αλγόριθμοι, Ασαφής λογική) και πιστεύεται ότι θα δώσει εξαιρετικά εργαλεία ανάπτυξης.

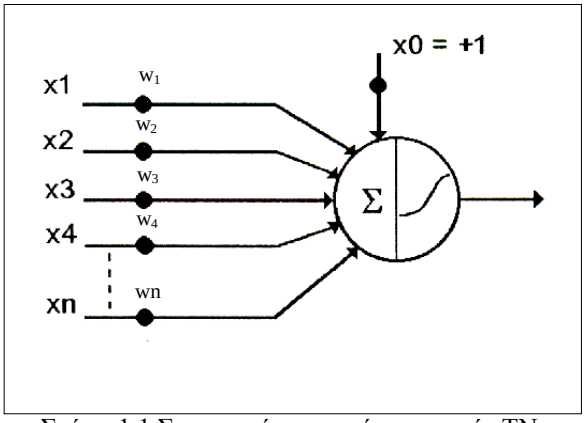
1.2 ΓΕΝΙΚΑ

Ένα ΤΝΔ είναι ένα υπολογιστικό σύστημα που εκτελεί ορισμένες από τις χαρακτηριστικές λειτουργίες των πραγματικών νευρωνικών δικτύων (ΠΝΔ). Μερικά από τα πλεονεκτήματα των ΤΝΔ είναι:

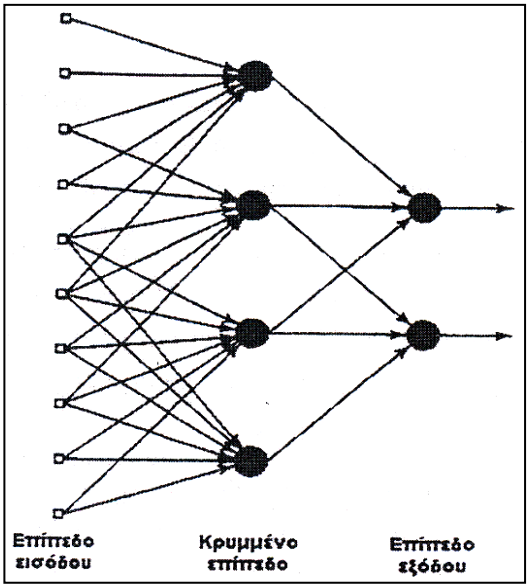
- Παράλληλη επεξεργασία.
- Μνήμη.

- Ικανότητα εκμάθησης τυχαίων συναρτήσεων εισόδου εξόδου.
- Ικανότητα προσαρμογής.
- Απλές υπολογιστικές μονάδες (Τεχνικοί Νευρώνες).

Ένα ΤΝΔ αποτελείται από τεχνητούς νευρώνες (ΤΝ) που αποτελούν την υπολογιστική μηχανή του (Σχήμα 1.1) και είναι οργανωμένοι σε επάλληλα στρώματα (επίπεδα, layers).



Σχήμα 1.1 Σχηματική αναπαράσταση ενός ΤΝ με μεταβλητά βάρη εισόδου (w_1,w_2,…wn) και μια σταθερή είσοδο (κατώφλι).



Σχήμα 1.2 Δίκτυο πρόσθιας τροφοδότησης

Συνήθως υπάρχει ένα επίπεδο εισόδου (με ή χωρίς υπολογιστικές ικανότητες) όπου εισάγεται το εκπαιδευτικό διάνυσμα εισόδου ή το διάνυσμα προς ταξινόμηση, ένα ή περισσότερα κρυφά επίπεδα όπου γίνεται η γραμμική ή μη γραμμική επεξεργασία των πληροφοριών και τέλος, ένα επίπεδο εξόδου που συνήθως, διαθέτει υπολογιστική ικανότητα και μεταφέρει τα αποτελέσματα στον έξω κόσμο. Στο σχήμα 1.2 απεικονίζεται ένα μη ανατροφοδοτούμενο δίκτυο. Τα δίκτυα αυτά δεν περιέχουν κλειστά μονοπάτια ή συνδέσεις προς προηγούμενα επίπεδα . Ο αριθμός των ΤΝ μπορεί να κυμαίνεται από μερικούς ΤΝ έως μερικές χιλιάδες ΤΝ.

Συνήθως όλοι οι ΤΝ υλοποιούν την ίδια συνάρτηση μεταφοράς, χωρίς αυτό να είναι δεσμευτικό. Η σύνδεση των ΤΝ υλοποιείται με "βάρη" που σταθμίζουν την έξοδο κάθε ΤΝ. Οι τιμές των βαρών των συνδέσεων αποτελούν τη γνώση που είναι αποθηκευμένη στο δίκτυο και καθορίζουν τη λειτουργία του. Η συνδεσμολογία καθώς και η συνάρτηση μεταφοράς που υλοποιεί κάθε ΤΝ μπορεί να ποικίλλει. Ακόμη εφαρμόζεται κάποιος

αλγόριθμος εκπαίδευσης ο οποίος τροποποιεί τις τιμές των βαρών, έτσι ώστε το ΤΝΔ να υλοποιήσει την επιθυμητή απεικόνιση των διανυσμάτων εισόδου στα διανύσματα εξόδου. Όλα αυτά τα χαρακτηριστικά στοιχειοθετούν ένα μοντέλο ΤΝΔ. Μερικά γνωστά μοντέλα ΤΝΔ είναι του Hopfield, του Kohonen, το backpropagation, το counterpropagation και το ART.

Ορισμένα μοντέλα ΤΝΔ μοιάζουν σημαντικά με τα ΠΝΔ, όμως κανένα δεν προσομοιώνει όλες τις λειτουργίες των ΠΝΔ. Άλλωστε πολλές λειτουργίες των ΠΝΔ δεν είναι σαφώς γνωστές. Τέλος σε ορισμένα ΤΝΔ ο κατασκευαστής τους δίνει κάποιες ικανότητες οι οποίες δεν απαντώνται καθόλου στα ΠΝΔ.

1.3 ΚΑΤΗΓΟΡΙΕΣ ΤΝΔ

Τα ΤΝΔ μπορούν να χωριστούν σε δύο γενικές κατηγορίες με βάση τη συνάρτηση μεταφοράς που υλοποιούν οι ΤΝ:

- Τα **στατικά** όπως είναι το BP και το RBF.
- Τα **δυναμικά** όπως είναι το Hopfield.

Τα στατικά ΤΝΔ υλοποιούν συναρτήσεις της μορφής $y = f(\chi)$ όπου το χ είναι πραγματικός αριθμός και η συνάρτηση y λαμβάνει συνήθως τιμές στο διάστημα *[0,1]* ή σε κάποιο άλλο διάστημα.

Τα δυναμικά (dynamic) ΤΝΔ διαθέτουν ως συνάρτηση μεταφοράς των ΤΝ τους μια εξίσωση διαφορών ή ακόμη και κάποια διαφορική εξίσωση. Πρόκειται για ιδιαίτερα χρήσιμα ΤΝΔ με σημαντικές εφαρμογές σε συστήματα ελέγχου αεροπλάνων, πυραύλων, διαστημοπλοίων, ρομπότ κλπ.

Τα δυναμικά ΤΝΔ μπορούμε να τα διακρίνουμε σε αυτά που:

- Δεν διαθέτουν ανάδραση δυναμικών (feedforward dynamics). Δεν περιέχουν συνδέσεις προς προηγούμενα επίπεδα.
- Διαθέτουν ανατροφοδότηση εξόδου (output feedback).
- Διαθέτουν ανατροφοδότηση καταστάσεων (state feedback).

Επίσης τα ΤΝΔ ταξινομούνται ως προς τον τρόπο λειτουργίας τους με δύο διαφορετικούς τρόπους: είτε ως **αυτοσυσχετιζόμενα** είτε ως **ετεροσυσχετιζόμενα**. Στον πρώτο τύπο ΤΝΔ το επιθυμητό διάνυσμα εξόδου είναι ίδιο με το διάνυσμα εισόδου,

δηλαδή, το εκπαιδευμένο ΤΝΔ ανακαλεί κάποιο αποθηκευμένο πρότυπο διάνυσμα, ακόμη και αν εισαχθεί στο επίπεδο εισόδου κάποια ελλιπής μορφή του αρχικού προτύπου (BSB, HN). Στον δεύτερο τύπο ΤΝΔ το επιθυμητό διάνυσμα εξόδου είναι διαφορετικό από το διάνυσμα εισόδου. Με αυτόν τον τρόπο κατασκευάζεται μια αυθαίρετη σχέση μεταξύ των διανυσμάτων εισόδου και των διανυσμάτων εξόδου (LVQ, SOM κατηγοριών, CP, BP και παραλλαγές του). Τέλος, ορισμένοι τύποι ΤΝΔ μπορούν να λειτουργήσουν και με τους δύο προαναφερθέντες τρόπους (BP).

1.4 ΕΚΠΑΙΔΕΥΣΗ ΤΩΝ ΤΝΔ

Η έννοια της εκπαίδευσης είναι πολύ ευρεία. Σε γενικές γραμμές η εκπαίδευση μπορεί να οριστεί ως η κατάλληλη χρήση πληροφοριών για βελτίωση της συμπεριφοράς ενός συστήματος ή για την επίλυση ενός προβλήματος. Στην πιο συνηθισμένη περίπτωση των προβλημάτων απεικόνισης (συσχέτιση προτύπων εισόδου - εξόδου) η εκπαίδευση μπορεί να οριστεί ως η τροποποίηση των παραμέτρων (βαρών) του ΤΝΔ, ώστε χρησιμοποιώντας ένα σύνολο δεδομένων να πλησιάσουμε σταδιακά την επιθυμητή συμπεριφορά συγκρίνοντας την τρέχουσα απόκριση του δικτύου με την επιθυμητή απόκριση.

Υπάρχει μια πληθώρα αλγορίθμων οι οποίοι χρησιμοποιούνται για την "εκπαίδευση" των ΤΝΔ. Στην πράξη η εκπαίδευση κάποιου ΤΝΔ ισοδυναμεί με την προσαρμογή των τιμών των βαρών σύμφωνα με τους κανόνες που ορίζει ο αλγόριθμος εκπαίδευσης, μέχρι αυτά να λάβουν τις κατάλληλες τιμές. Οι μεθοδολογίες εκπαίδευσης που χρησιμοποιούνται είναι:

- Με εποπτεία από εξωτερική πηγή (supervised).
- Χωρίς εποπτεία (unsupervised).
- Με ανταγωνισμό των ΤΝ.

Κατά την **εκπαίδευση με εποπτεία** παρουσιάζονται στο ΤΝΔ ζευγάρια διανυσμάτων εισόδου - εξόδου και το ΤΝΔ προσπαθεί να κατασκευάσει μια σχέση (συνάρτηση) των διανυσμάτων όσο το δυνατόν πιστότερη. Πιο συγκεκριμένα, κάποιος αλγόριθμος καθοδήγησης από εξωτερική πηγή ακολουθεί μια επαναληπτική διαδικασία βήμα προς βήμα σύμφωνα με την οποία τροποποιούνται οι τιμές των βαρών που συνδέουν τους ΤΝ. Οι παράγοντες που επηρεάζουν την μεταβολή των βαρών είναι:

- Οι τρέχουσες τιμές των βαρών.
- Οι είσοδοι (σταθμισμένο άθροισμα) των ΤΝ.
- Οι έξοδοι (δραστηριότητα) των ΤΝ.
- Οι επιθυμητές έξοδοι.

Έστω ένα σύνολο εκπαίδευσης με *N* ζεύγη της μορφής *(χ,d)*, όπου *χ* είναι το πρότυπο (διάνυσμα εισόδου) και *d* το διάνυσμα των επιθυμητών εξόδων για το πρότυπο *χ*. Το σήμα σφάλματος στην έξοδο του νευρώνα *j* στην $n^{οστη}$ επανάληψη καθορίζεται από τον τύπο:

$$e_j(n) = d_j(n) - y_j(n) \tag{1.1}$$

όπου y_j και d_j είναι η πραγματική και η επιθυμητή έξοδος του νευρώνα j που αντιστοιχούν στην είσοδο *χ(n)*. Ορίζουμε την στιγμιαία τιμή του τετραγωνικού σφάλματος για τον νευρώνα εξόδου *j* ως:

$$de_j(n) = \frac{1}{2}\boldsymbol{e}_j^2(n) \tag{1.2}$$

Αντίστοιχα ορίζουμε το άθροισμα των τεραγωνικών σφαλμάτων όλων των νευρώνων εξόδου ως:

$$G(n) = \frac{1}{2}\sum_{j\in C}\boldsymbol{e}_j^2(n) \tag{1.3}$$

Η μέση τιμή των σφαλμάτων για ένα σύνολο *N* προτύπων είναι:

$$G_{αν} = \frac{1}{N}\sum_{n=1}^{N} G(n) \tag{1.4}$$

Η επαναληπτική διαδικασία σταματάει όταν ελαχιστοποιηθεί η συνάρτηση σφάλματος $G_{αν}$. Το βασικό μειονέκτημα αυτής της μεθόδου είναι η πιθανότητα παγίδευσης του ΤΝΔ σε κάποιο τοπικό ελάχιστο.

Κατά την **εκπαίδευση χωρίς εποπτεία** παρουσιάζονται στο ΤΝΔ μόνον τα διανύσματα εισόδου τα οποία ταξινομούνται σε κατηγορίες ανάλογα με την σχετική ομοιότητα τους (αυτοοργάνωση του ΤΝΔ). Πιο συγκεκριμένα κάποιος αλγόριθμος χωρίς εποπτεία πραγματοποιεί ψευδοτυχαίες μεταβολές των τιμών των βαρών. Από τις μεταβολές διατηρούνται μόνον αυτές που βελτιώνουν (ελαχιστοποιούν) την συνάρτηση σφάλματος $G_{αν}$. Η εφαρμογή κατάλληλης στρατηγικής για την επιλογή του μεγέθους των μεταβολών των τιμών των βαρών οδηγεί στην εύρεση του ολικού ελαχίστου της συνάρτησης σφάλματος. Το σημαντικότερο μειονέκτημα αυτής της μεθόδου είναι ο

μεγάλος χρόνος που απαιτείται για την εύρεση του ολικού ελαχίστου.

Κατά την **εκπαίδευση με ανταγωνισμό** οι ΤΝ συναγωνίζονται για το ποιος θα μείνει σε λειτουργία. Με τη μέθοδο αυτή σταδιακά απενεργοποιούνται νευρώνες οι οποίοι δεν είναι ενεργοί και δεν επηρεάζουν την απόδοση του δικτύου. Στην απλούστερη περίπτωση απομένει δραστηριοποιημένος μόνον ο ΤΝ που έχει την μεγαλύτερη δραστηριοποίηση ενώ οι υπόλοιποι ΤΝ καταστέλλονται. Οι διάφορες μέθοδοι εκπαίδευσης μπορεί να χρησιμοποιηθούν σε συνδυασμό.

Πρέπει να υπογραμμιστεί ότι επειδή απαιτούνται πολλοί επαναληπτικοί υπολογισμοί για την εκπαίδευση των ΤΝΔ, συνήθως αυτή πραγματοποιείται σε χρόνο πριν από την χρησιμοποίηση του ΤΝΔ για την λύση του προβλήματος εφαρμογής (εκπαίδευση off line).

1.5 ΕΛΕΓΧΟΣ ΑΠΟΔΟΣΗΣ ΤΩΝ ΤΝΔ

Ο έλεγχος της απόδοσης κάποιου ΤΝΔ μπορεί να γίνεται με κάποια συνάρτηση σφάλματος, είτε στο εκπαιδευτικό σύνολο διανυσμάτων, είτε σε "άγνωστο" σύνολο διανυσμάτων τα οποία δεν έχει δει προηγουμένως το ΤΝΔ. Για την πιστοποίηση των αποτελεσμάτων εφαρμόζονται διάφορες μέθοδοι όπως η αναδιάταξη των διανυσμάτων και ο τυχαίος διαχωρισμός των συνόλων εκπαίδευσης και ελέγχου.

Αρκετό ενδιαφέρον για τον έλεγχο απόδοσης παρουσιάζουν η "ανάκληση" (recall) και η "ακρίβεια" (precision) ενός ΤΝΔ. Η ανάκληση ορίζεται ως ο λόγος του αριθμού των διανυσμάτων κάποιας κατηγορίας που ταξινομούνται σωστά από το ΤΝΔ, ως προς τον πραγματικό αριθμό των διανυσμάτων που ανήκουν σε αυτήν την κατηγορία. Η ακρίβεια ορίζεται ως ο λόγος του αριθμού των διανυσμάτων κάποιας κατηγορίας που ταξινομούνται σωστά από το ΤΝΔ, ως προς τον συνολικό αριθμό των διανυσμάτων που ταξινομούνται σε αυτή την κατηγορία (ορθά ή λανθασμένα). Για παράδειγμα, αν σε μια κατηγορία ανήκουν 50 διανύσματα και το ΤΝΔ ταξινομεί σωστά τα 40, τότε η ανάκληση είναι 40/50, ενώ εάν σε αυτή την κατηγορία ταξινομήσει π.χ. 7 διανύσματα που ανήκαν σε άλλες κατηγορίες, τότε η ακρίβεια είναι 40/57.

1.6 ΕΦΑΡΜΟΓΕΣ ΤΩΝ ΤΝΔ

Τα ΤΝΔ χρησιμοποιούνται είτε αυτόνομα είτε σε συνδυασμό με άλλες τεχνικές ως εναλλακτικός τρόπος επίλυσης πολλών προβλημάτων. Τέτοια προβλήματα είναι:

- Η βελτιστοποίηση διαδικασιών (optimisation).
- Η αναγνώριση προτύπων (pattern recognition).
- Η επεξεργασία σήματος (signal processing).
- Το προσαρμοστικό φιλτράρισμα (adaptive filtering).
- Η συμπίεση δεδομένων (data compression).
- Η αναγνώριση και επεξεργασία της ανθρώπινης φωνής.
- Η επεξεργασία εικόνας.

Ακόμη τα ΤΝΔ χρησιμοποιούνται:

- Στην διαλεύκανση των μηχανισμών που διέπουν την λειτουργία του εγκέφαλου και ιδιαίτερα στην ανάλυση της αντίληψης, της αίσθησης και της κινητικότητας αλλά και την ερμηνεία γνωστικών και ψυχολογικών φαινομένων.
- Στην επεξεργασία βιοηλεκτρικών σημάτων(ΗΚΓ, ΗΕΓ, ΗΜΓ).
- Στην ανάλυση των μηχανισμών της όρασης και την επεξεργασία ιατρικής εικόνας.
- Άλλες εφαρμογές (βιολογία, βιοχημεία, ιατρική γνωμάτευση).

1.7 Η ΤΕΧΝΟΛΟΓΙΑ ΚΑΤΑΣΚΕΥΗΣ ΤΩΝ ΤΝΔ

Τα ΤΝΔ μπορούν να υλοποιηθούν με λογισμικό (software) ή να κατασκευαστούν απευθείας σε ολοκληρωμένο κύκλωμα πολύ μεγάλης κλίμακας ολοκλήρωσης.

Ο πιο προσιτός τρόπος για να κατασκευάσει κάποιος ένα ΤΝΔ είναι να χρησιμοποιήσει ως προγραμματιστικό εργαλείο οποιαδήποτε γλώσσα προγραμματισμού τέταρτης γενιάς η οποία υποστηρίζει πράξεις πινάκων (πχ. Pascal, C, C++). Ακόμη υπάρχουν στο εμπόριο εξειδικευμένα εργαλεία που διευκολύνουν την ανάπτυξη ΤΝΔ. Τέτοια περιβάλλοντα διευκολύνουν την ανάπτυξη επιτρέποντας την αλληλεπιδραστική επικοινωνία χρήστη και ΤΝΔ μέσα από κυλιόμενα μενού, τόσο κατά την φάση της ανάπτυξης, όσο και κατά την φάση του ελέγχου. Ακόμη για αυξημένη ταχύτητα αριθμητικών υπολογισμών το

αναπτυξιακό περιβάλλον μπορεί να συνδυάζεται με κάποιον αριθμητικό επιταχυντή σε hardware .

Η κατασκευή ΤΝΔ με τεχνολογία VLSI παρουσιάζει αυξημένο κόστος αλλά και ιδιαίτερο ενδιαφέρον, επειδή κατασκευάζοντας τις παράλληλες δομές των ΤΝΔ σε ολοκληρωμένα κυκλώματα (ΟΚ) επιτυγχάνεται εξαιρετικά υψηλή ταχύτητα επεξεργασίας των πληροφοριών. Μερικά από τα χαρακτηριστικά που πρέπει να διαθέτει ένα ΤΝΔ που κατασκευάζεται σε ΟΚ είναι τα εξής:

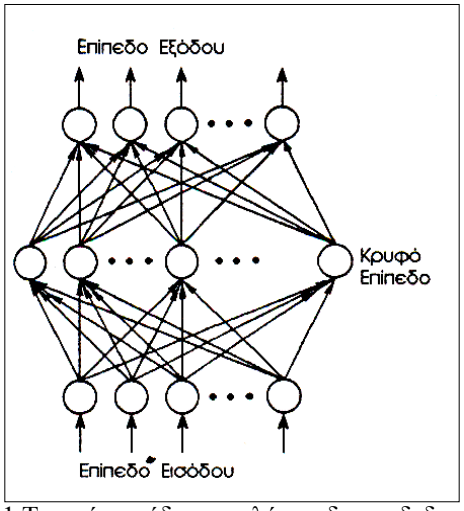
- Να επιτρέπει την δημιουργία διαφορετικών συνδεσμολογιών των ΤΝ έτσι ώστε να μπορεί το ΤΝΔ να χρησιμοποιηθεί σε διαφορετικές εφαρμογές.
- Η απόδοση του πρέπει να είναι σταθερή και ανεξάρτητη από το μέγεθος του ΤΝΔ.
- Το κόστος παραγωγής του πρέπει να είναι λογικό. Αυτή η απαίτηση πληρείται αν χρησιμοποιηθεί τεχνολογία κατασκευής CMOS, η οποία είναι ευρέως διαθέσιμη στο εμπόριο.

ΚΕΦΑΛΑΙΟ 2

ΠΟΛΥΕΠΙΠΕΔΑ PERCEPTRONS (MLP)

2.1 ΠΟΛΥΕΠΙΠΕΔΑ PERCEPTRONS (Back error propagation BP)

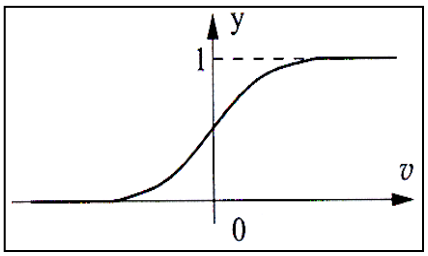
Το BP είναι σήμερα το πιο δημοφιλές ΤΝΔ και έγινε γνωστό από τις εργασίες των Rumelhart, Hinton και Williams[5] ενώ παρόμοια διάταξη ΤΝΔ είχε προταθεί παλαιότερα και από άλλους ερευνητές. Η ονομασία του οφείλεται στον τρόπο με τον οποίο μεταφέρεται το σφάλμα από το επίπεδο εξόδου προς τα προηγούμενα επίπεδα επεξεργασίας του ΤΝΔ και συγκεκριμένα από την ανατροφοδότηση του σφάλματος από το επίπεδο εξόδου προς τα προηγούμενα επίπεδα επεξεργασίας. Το BP διαθέτει σημαντικές δυνατότητες δημιουργίας μη γραμμικών σχέσεων μεταξύ των διανυσμάτων εισόδου και των διανυσμάτων εξόδου. Για να καταστεί δυνατή η απεικόνιση, γίνεται βαθμιαία μείωση του συνολικού σφάλματος ως προς τα βάρη, με χρήση "τοπικών" πληροφοριών. Επομένως το BP ικανοποιεί την απαίτηση για τοπική επεξεργασία της πληροφορίας που τίθεται από τα ΠΝΔ .



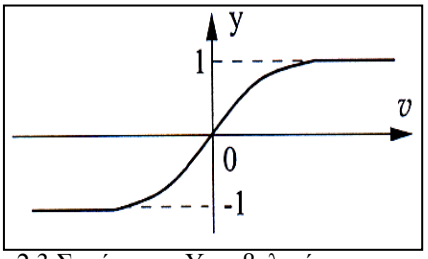
Σχήμα 2.1 Τυπικό παράδειγμα πλήρως διασυνδεδεμένου BP.

Τυπικά το BP αποτελείται από το επίπεδο εισόδου, ένα τουλάχιστον κρυφό επίπεδο και το επίπεδο εξόδου. Μπορούμε να χρησιμοποιήσουμε όσα κρυφά επίπεδα επιθυμούμε.

Κάθε επίπεδο είναι πλήρως συνδεδεμένο με μεταβλητά βάρη με το επόμενο. Η βασική διαφορά των ΤΝ του ΒΡ από των Perceptrons είναι ότι οι πρώτοι διαθέτουν κάποια σιγμοειδή συνάρτηση μεταφοράς, ενώ τα δεύτερα διαθέτουν έναν κβαντιστή. Κβαντιστής ή αλλιώς βηματική συνάρτηση είναι η συνάρτηση όπου αν η είσοδος είναι θετική το αποτέλεσμα είναι 1 (ή +1), ενώ αν η είσοδος είναι αρνητική το αποτέλεσμα είναι 0 (ή -1).



Σχήμα 2.2 Σιγμοειδής Λογιστική Συνάρτηση



Σχήμα 2.3 Συνάρτηση Υπερβολικής εφαπτομένης

Οι σιγμοειδείς συναρτήσεις είναι δύο τύπων:

- Η *λογιστική* συνάρτηση με τύπο:

$$y(u) = 1 / (1+\exp(-gu)) \quad (2.1)$$

η οποία παρέχει τιμές στο (0,1) και g είναι μια παράμετρος που καθορίζει την κλήση της σιγμοειδούς (Σχήμα 2.2).

- Η συνάρτηση *υπερβολικής εφαπτομένης* με τύπο:

$$y(u) = \tanh(gu) \quad (2.2)$$

η οποία παρέχει τιμές στο διάστημα (-1, 1) (Σχήμα 2.3).

Το ΒΡ είναι αλγόριθμος που λειτουργεί με εποπτεία και μπορεί να λειτουργήσει ως αυτοσυσχετιζόμενο ή ετεροσυσχετιζόμενο ΤΝΔ. Η κλασική διάταξη ενός ΒΡ φαίνεται στο Σχήμα 2.1. Τα στάδια επεξεργασίας που ακολουθεί το ΒΡ είναι:

- Εισαγωγή του εκπαιδευτικού διανύσματος εισόδου και υπολογισμός των σταθμισμένων αθροισμάτων εισόδου καθώς και των εξόδων των ΤΝ, μέχρι το επίπεδο εξόδου, όπου παράγεται το διάνυσμα εξόδου.
- Υπολογισμός του μέτρου της διαφοράς (σφάλματος) του διανύσματος εξόδου που υπολογίστηκε κατά το στάδιο 1, από το επιθυμητό διάνυσμα εξόδου, καθώς επίσης και της αναγκαίας μεταβολής των βαρών.
- Υπολογισμός του σφάλματος των ΤΝ των κρυφών επιπέδων και της αντίστοιχης μεταβολής των βαρών.

- Μεταβολή όλων των βαρών με πρόσθεση των αντίστοιχων τιμών που υπολογίστηκαν προηγουμένως.

2.1.1 Ορισμός μεγεθών και εκπαίδευση του BP

Οι συμβολισμοί που χρησιμοποιούνται σε αυτό το τμήμα είναι οι εξής:

$w^{(l)}$ = διάνυσμα των βαρών των συνάψεων του επιπέδου l

$\theta^{(l)}$ = διάνυσμα πόλωσης του επιπέδου l

$\upsilon^{(l)}$ = διάνυσμα των τιμών ενεργοποίησης των νευρώνων του επιπέδου l

$y^{(l)}$ = διάνυσμα των σημάτων εξόδου των νευρώνων του επιπέδου l. Ο δείκτης l των επιπέδων παίρνει τιμές από l = 0 για το επίπεδο εισόδου, μέχρι και l = L για το επίπεδο εξόδου.

$\delta^{(l)}$ = διάνυσμα των τοπικών κλίσεων των νευρώνων του επιπέδου l

$e^{(l)}$ = διάνυσμα σφαλμάτων που περιέχει τα $e_1, e_2, \ldots, e_q$

2.1.2 Ο αλγόριθμος back – propagation

Τα στάδια του αλγορίθμου είναι:

1. **Αρχικοποίηση.** Ορισμός της αρχιτεκτονικής του δικτύου και αρχικοποίηση των βαρών των συνδέσεων με μικρές τυχαίες τιμές ομοιόμορφα κατανεμημένες (συνήθως στοδιάστημα (-1,1)).
2. **Παρουσίαση των προτύπων εκπαίδευσης**. Παρουσίαση στο δίκτυο μιας εποχής προτύπων εκπαίδευσης. Για κάθε πρότυπο του συνόλου εκτέλεσε τα στάδια: 3 και 4 που ακολουθούν.
3. **Υπολογισμοί ευθείας φοράς.** Θεωρούμε το πρότυπο εκπαίδευσης [x(n) , d(n)] όπου x(n) είναι το διάνυσμα εφαρμογής στην είσοδο του δικτύου και d(n) είναι το επιθυμητό διάνυσμα εξόδου του δικτύου. Η έξοδος του νευρώνα j του επιπέδου l είναι

$$u_j^{(l)}(n) = \sum_{i=0}^{p} u_{ji}^{(l)}(n)\, y_i^{(l-1)}(n) \qquad (3.1)$$

όπου $y_i^{(l-1)}(n)$ είναι η έξοδος του νευρώνα i του προηγούμενου επιπέδου l-1, κατά την επανάληψη n και $w_{ji}^{(l)}(n)$ είναι το βάρος της σύναψης του νευρώνα j στο επίπεδο l που τροφοδοτείται από το νευρώνα i του επιπέδου l-1. Για i = 0 έχουμε $y_0 = -1$ και $w_{j0} = \theta_j$ όπου θ_j είναι η τιμή της πόλωσης. Χρησιμοποιούμε τη λογιστική συνάρτηση για το νευρώνα j του επιπέδου l:

$$y_j^{(n)} = \frac{1}{1+\exp(-u_j^{(l)}(n))} \tag{3.2}$$

Αν ο νευρώνας j ανήκει στο πρώτο κρυμμένο επίπεδο (l = 1) θέστε $y_i^{(0)}(n) = x_j(n)$ όπου $x_j(n)$ είναι το j-οστο στοιχείο του διανύσματος εισόδου χ(n). Αν ο νευρώνας j ανήκει στο επίπεδο εξόδου (l = L) θέστε $y_j^{(L)}(n) = \mathbf{o}_j(n)$ όπου **o**(n) το επιθυμητό διάνυσμα εξόδου. Έτσι υπολογίστε το σφάλμα $e_j(n) = d_j(n) - o_j(n)$ όπου $d_j(n)$ είναι το j-οστό στοιχείο του επιθυμητού διανύσματος εξόδου d(n).

4. **Υπολογισμοί αντίστροφης φοράς**. Υπολογίστε τα δ (τοπικές κλίσεις) προχωρώντας αντίστροφα στο δίκτυο ανά επίπεδο:

$$\delta_j^{(L)} = e_j^{(L)} o_j(n)[1-o_j(n)] \tag{3.3}$$

$$\delta_j^{(l)} = y_j^{(l)}[1-y_j^{(l)}]\sum_k \delta_k^{(l+1)}(n)\, w_{kj}^{(l+1)}(n) \tag{3.4}$$

Προσαρμόστε τα βάρη του επιπέδου l σύμφωνα με τον κανόνα δέλτα:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[\, w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n)\, y_j^{(l-1)}(n) \tag{3.5}$$

όπου *n* είναι ο ρυθμός μάθησης και α είναι σταθερά momentum.

5. **Επανάληψη**. Επανέλαβε τους υπολογισμούς για νέες εποχές προτύπων έως ότου οι παράμετροι του δικτύου σταθεροποιηθούν, και η μέση τιμή του σφάλματος G_{au} γίνει ελάχιστη. Η σειρά της παρουσίασης των προτύπων σε κάθε εποχή πρέπει να είναι τυχαία. Η σταθερά α και ο ρυθμός μάθησης η συνήθως μειώνονται με το πέρασμα των εποχών.

2.2 ΤΕΡΜΑΤΙΣΜΟΣ ΤΗΣ ΕΚΠΑΙΔΕΥΣΗΣ

Η εκπαίδευση μπορεί να σταματήσει όταν:

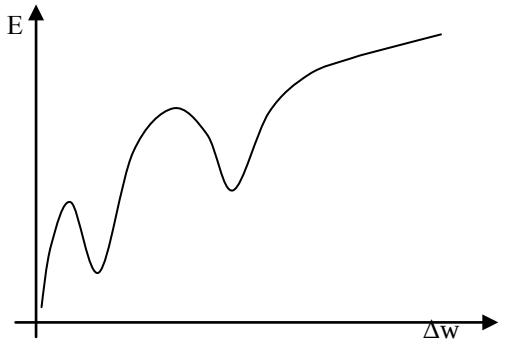
- Η μεταβολή της συνάρτησης σφάλματος είναι ικανοποιητικά μικρή ή μηδέν (εξ ορισμού μηδενίζεται στο σημείο ολικού ελαχίστου).
- Η συνάρτησης σφάλματος τείνει στο μηδέν.
- Το ΤΝΔ μάθει όλα τα εκπαιδευτικά διανύσματα, δηλαδή τα ταξινομήσει στην σωστή κατηγορία (αυτό δεν εγγυάται την ικανότητα γενίκευσης σε άγνωστα διανύσματα).
- Συμπληρωθεί κάποιος συγκεκριμένος αριθμός επαναλήψεων, χωρίς να υπάρχει εγγύηση ότι σε αυτό το σημείο βρίσκεται κάποιο τοπικό ελάχιστο.

Το μειονέκτημα όλων αυτών των μεθόδων είναι ότι εξαρτώνται από τις εκάστοτε παραμέτρους του αλγορίθμου εκπαίδευσης.

Μια άλλη μέθοδος τερματισμού της εκπαίδευσης, που δεν περιορίζεται από τις παραμέτρους του αλγορίθμου εκπαίδευσης και ακόμη μπορεί να βελτιώσει την ικανότητα γενίκευσης του ΤΝΔ, είναι η διαεπικύρωση (Cross Validation). Σύμφωνα με αυτήν, τα διανύσματα χωρίζονται σε δύο σύνολα, ένα εκπαιδευτικό και ένα ελέγχου. Καθώς προχωράει η εκπαίδευση η απόδοση βελτιώνεται στο σύνολο εκπαίδευσης αλλά και στο σύνολο ελέγχου. Μετά από κάποιον αριθμό επαναλήψεων η βελτίωση στο σύνολο ελέγχου σταματάει ή και χειροτερεύει και αρχίζει να μειώνεται η απόδοση του ΤΝΔ. Σε αυτό το σημείο πρέπει να σταματήσει η εκπαίδευση, διαφορετικά το ΤΝΔ θα υπερεκπαιδευτεί. Υπερεκπαίδευση του δικτύου είναι η κατάσταση κατά την οποία το δίκτυο αντί να μαθαίνει, απομνημονεύει τα δεδομένα, οπότε μειώνεται και η ικανότητα γενίκευσής του. Το μειονέκτημα της μεθόδου είναι ότι απαιτεί περισσότερους υπολογισμούς και δεν μπορεί να εφαρμοστεί αν δεν διατίθεται μεγάλος αριθμός διανυσμάτων.

Τέλος λόγω μεγάλης πολυπλοκότητας της επιφάνειας σφάλματος (φαράγγια-κοιλάδες), οι αλγόριθμοι εκπαίδευσης που υλοποιούν αναζήτηση με χρήση της κλίσης της επιφάνειας σφάλματος, έχουν την τάση να παγιδεύονται σε τοπικά ελάχιστα αδυνατώντας να βρουν την βέλτιστη δυνατή λύση. Ακόμη και η εύρεση κάποιου τοπικού ελαχίστου μπορεί να αποδειχτεί εξαιρετικά χρονοβόρα. Το γεγονός αυτό οφείλεται στο ότι η

εκπαίδευση είναι γρήγορη στις περιοχές που παρουσιάζουν μεγάλη κλίση αλλά πολύ αργή στις περιοχές όπου η υπερεπιφάνεια σφάλματος είναι σχεδόν επίπεδη. Αυτές οι επίπεδες περιοχές μοιάζουν με τοπικό ελάχιστο (Σχήμα 2.4).



Σχήμα 2.4 Διάγραμμα Υπερεπιφάνειας Σφάλματος Ε.

Αν η λύση που βρέθηκε δεν είναι ικανοποιητική πρέπει να δώσουμε νέες αρχικές τιμές στα βάρη και ενδεχομένως στις άλλες παραμέτρους του ΤΝΔ και να αναζητήσουμε κάποια άλλη λύση. Δυστυχώς δεν υπάρχει εγγύηση ότι το ΤΝΔ θα εκπαιδευτεί ικανοποιητικά σε πεπερασμένο χρόνο ή ότι θα βρούμε το ολικό ελάχιστο της συνάρτησης σφάλματος επειδή η εκπαίδευση είναι ένα πρόβλημα του τύπου NP-complete. Πάντως μετά από αρκετές προσπάθειες είναι δυνατή η εύρεση κάποιας ικανοποιητικής λύσης.

2.3 ΤΟ ΜΕΓΕΘΟΣ ΤΟΥ ΤΝΔ

Η επιλογή του σωστού μεγέθους του ΤΝΔ είναι εξαιρετικής σημασίας διότι αν επιλεγεί κάποιο ΤΝΔ μικρού μεγέθους τότε αυτό δεν θα μπορεί να σχηματίσει ένα καλό μοντέλο του προβλήματος. Αν πάλι, επιλεγεί κάποιο ΤΝΔ μεγάλου μεγέθους τότε αυτό θα μπορεί να εκπαιδευτεί αλλά δεν θα μπορεί να γενικεύσει ικανοποιητικά. Έχει αποδειχτεί ότι κάποιο απεριόριστου μεγέθους ΤΝΔ μπορεί να δημιουργήσει οποιαδήποτε συνεχή μη γραμμική απεικόνιση με απειροστή ακρίβεια. Ωστόσο δεν είναι δυνατόν να γνωρίζουμε το σωστό πεπερασμένο μέγεθος κάποιου ΤΝΔ για ένα συγκεκριμένο πρόβλημα και βέβαια για κάθε πρόβλημα χρειάζεται διαφορετικού μεγέθους ΤΝΔ. Σε κάθε περίπτωση το ζητούμενο είναι το μέγεθος του ΤΝΔ να προσεγγίσει καλύτερα το πρόβλημα ή αν τα εκπαιδευτικά διανύσματα είναι λίγα, τουλάχιστον τις ιδιότητες του συνόλου εκπαίδευσης.

Στην περίπτωση που αγνοούμε την δομή του προβλήματος μπορούμε απλά να

δοκιμάσουμε διαφορετικά μεγέθη ΤΝΔ. Μια μέθοδος είναι να ξεκινήσουμε με το ελάχιστο δυνατό μέγεθος ΤΝΔ και σταδιακά να προσθέτουμε ΤΝ μέχρι να αρχίσει να μειώνεται η απόδοση. Κάθε ΤΝΔ εκπαιδεύεται με καινούριες αρχικές συνθήκες. Παρόμοια μέθοδος είναι η προσθήκη ΤΝ κατά την διάρκεια της εκπαίδευσης, ξεκινώντας με έναν ΤΝ.

Μια άλλη μέθοδος είναι να ξεκινήσουμε από κάποιο μεγάλο ΤΝΔ εφόσον βέβαια υπάρχει κάποια εκτίμηση του τι είναι μεγάλο και σταδιακά να αποκόπτουμε τους ΤΝ ή τα βάρη που δεν παίζουν σημαντικό ρόλο στην επίλυση του προβλήματος.

Στην πράξη δεν χρησιμοποιούνται σχεδόν ποτέ περισσότερα από δύο κρυφά επίπεδα ενώ ο αριθμός των ΤΝ των/του κρυφού επιπέδου δεν πρέπει να υπερβαίνει τον αριθμό των εκπαιδευτικών διανυσμάτων εισόδου. Το άνω όριο για τον αριθμό των ΤΝ του κρυφού επιπέδου ενός ΤΝΔ με ένα κρυφό επίπεδο, είναι της τάξεως του αριθμού των εκπαιδευτικών διανυσμάτων (πχ. 1000 εκπαιδευτικά διανύσματα εισόδου σημαίνει αριθμό ΤΝ κρυφού επιπέδου στην περιοχή του 1000). Στην πραγματικότητα ο αριθμός των ΤΝ των/του κρυφού επιπέδου πρέπει να είναι σημαντικά μικρότερος από τον αριθμό των εκπαιδευτικών διανυσμάτων εισόδου, διαφορετικά το ΤΝΔ "αποστηθίζει" τα εκπαιδευτικά διανύσματα εισόδου και δεν γενικεύει ικανοποιητικά. Ακόμη μπορούμε να θέσουμε το άνω όριο του αριθμού των ΤΝ του κρυφού επιπέδου ως συνάρτηση της διάστασης του υπερχώρου των διανυσμάτων εισόδου.

2.4 ΒΕΛΤΙΩΣΗ ΤΟΥ ΒΡ ΚΑΙ ΕΥΡΕΤΙΚΟΙ ΚΑΝΟΝΕΣ

Παρά τις αξιόλογες δυνατότητες τις οποίες διαθέτει το ΒΡ χρειάζεται μεγάλους χρόνους για να συγκλίνει και έτσι καθίσταται μη εφαρμόσιμο σε εφαρμογές μεγάλης κλίμακας. Για να βελτιώσουμε την ταχύτητα σύγκλισής του μπορούμε να χρησιμοποιήσουμε:

- Περισσότερες και αναλυτικότερες πληροφορίες για την μεταβολή των βαρών όπως η χρήση μερικών παραγώγων δεύτερης τάξης.
- Κάποια ευρετική μέθοδο (heuristic).

Τέτοια μέθοδος είναι η προσθήκη ενός παράγοντα ορμής (momentum), ο οποίος πολλαπλασιάζεται με την μεταβολή του βάρους Δw_{ji} που έγινε στον προηγούμενο χρονικό

κύκλο: $\alpha \cdot [\, w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)]$

Ακόμη μπορούμε να συνδυάσουμε το ΒΡ με κάποια μέθοδο στατιστικού υπολογισμού μεταβολής των βαρών. Σε αυτή την περίπτωση, η μεταβολή κάποιου βάρους υπολογίζεται ως το άθροισμα της μεταβολής που ορίζει το ΒΡ και κάποιας ψευδοτυχαίας τιμής $\Delta w_{ji}[s]_{ran}$.

Σε ορισμένες περιπτώσεις η σύγκλιση του ΒΡ επιταχύνεται αν προσθέσουμε κάποιο πολλαπλάσιο του τοπικού σφάλματος β*e_i[l-1] του i ΤΝ του επιπέδου l-1 στην τιμή της δραστηριοποίησης του, πριν μεταβάλουμε βάρος.

Τέλος οι ΤΝ του επιπέδου εξόδου δεν είναι απαραίτητο να διαθέτουν μη γραμμική συνάρτηση μεταφοράς. Αν όμως χρησιμοποιήσουμε ως συνάρτηση μεταφοράς των ΤΝ του κρυφού επιπέδου κάποια γραμμική συνάρτηση, δεν προκύπτει κανένα πλεονέκτημα από την χρήση κρυφών επιπέδων και το ΒΡ είναι ισοδύναμο με τον κανόνα Δέλτα των Widrow και Hoff[8]. Λόγω αυτής της ομοιότητας, το ΒΡ ονομάζεται και "γενικευμένος κανόνας δέλτα" ("δ" είναι η ονομασία που χρησιμοποιείται πολλές φορές αντί του όρου "τοπικό σφάλμα").

2.5 ΕΦΑΡΜΟΓΕΣ ΤΟΥ ΒΡ

Το ΒΡ μπορούμε να το χρησιμοποιήσουμε για:

- Υλοποίηση λογικών δυαδικών συναρτήσεων.
- Τμηματοποίηση (segmentation) του υπερχώρου των διανυσμάτων εισόδου.
- Κατασκευή μη γραμμικών μετασχηματισμών για προσέγγιση συναρτήσεων.
- Προσέγγιση aposteriori πιθανοτήτων για ταξινόμηση (ταξινόμηση διανύσματος στην κατηγορία από την οποία υπάρχει η μεγαλύτερη πιθανότητα να προήλθε).

Άλλες εφαρμογές του ΒΡ είναι το φιλτράρισμα του ηλεκτροεγκεφαλογραφήματος ΗΚΓ και η πρόβλεψη των τιμών κλεισίματος του χρηματιστηρίου[9], η μετατροπή γραπτού κειμένου σε ήχο (Nettalk)[10], ο διαχωρισμός φωνηέντων και συμφώνων[11], ο διαχωρισμός του επιστρεφόμενου σήματος κάποιου μεταλλικού κυλίνδρου από το σήμα επιστροφής ενός αντίστοιχου μεγέθους βράχου που καταφθάνει σε μια συσκευή sonar[12], η εφαρμογή του ΒΡ σε τεχνικές τεχνητής όρασης[13], η αναγνώριση ομιλίας[14] και χειρόγραφου κειμένου[15] καθώς επίσης και η επεξεργασία ιατρικής εικόνας[16].

ΚΕΦΑΛΑΙΟ 3

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

3.1 ΟΡΙΣΜΟΙ

Ένα οποιοδήποτε μη κενό και πεπερασμένο σύνολο Σ αποτελούμενο από σύμβολα ονομάζεται *αλφάβητο* (alphabet). Για παράδειγμα, τα αλφάβητα ASCII, Unicode κ.λπ. είναι σήμερα αρκετά διαδεδομένα στους ηλεκτρονικούς υπολογιστές. Κάθε στοιχείο ενός αλφαβήτου Σ ονομάζεται *σύμβολο* του αλφαβήτου.

Μια πεπερασμένη παράθεση από σύμβολα ονομάζεται *συμβολοσειρά.* Η συμβολοσειρά που δεν περιέχει κανένα σύμβολο ονομάζεται *κενή συμβολοσειρά* και παριστάνεται με ϵ. Ο αριθμός των συμβόλων που αποτελούν μια συμβολοσειρά α ονομάζεται *μήκος* της συμβολοσειράς και παριστάνεται με |α|.

Έστω ένα αλφάβητο Σ. *Γλώσσα* (language) επί του αλφαβήτου Σ ονομάζουμε κάθε σύνολο συμβολοσειρών του Σ. Οι γλώσσες συμβολίζονται συνήθως με κεφαλαία λατινικά γράμματα, όπως *L, S, T,* κ.λπ. Πολλές φορές παραλείπουμε τον καθορισμό του αλφαβήτου της γλώσσας, αν αυτό εννοείται.

Γραμματική (grammar) ονομάζεται ένα σύστημα παραγωγής συμβολοσειρών *G* που ορίζεται από μια διατεταγμένη τετράδα της μορφής *(T, N, P, S)* όπου:

- *T* είναι ένα αλφάβητο του οποίου τα μέλη ονομάζονται *τερματικά σύμβολα* (terminal symbols).
- *N* είναι ένα αλφάβητο του οποίου τα μέλη ονομάζονται *μη τερματικά σύμβολα* (non terminal symbols). Τα σύνολα *T* και *N* πρέπει να είναι ξένα μεταξύ τους.
- *P* είναι ένα πεπερασμένο σύνολο *κανόνων παραγωγής.* Οι κανόνες παραγωγής είναι διατεταγμένα ζεύγη *(α, β)* συμβολοσειρών του αλφαβήτου *T U N,* δηλαδή:

$$P \subseteq (T \cup N)^* \times (T \cup N)^*$$

Ένας κανόνας παραγωγής συμβολίζεται συνήθως με $\alpha \rightarrow \beta$. Η συμβολοσειρά α ονομάζεται *αριστερό* ή *παράγον* μέλος του κανόνα, ενώ η συμβολοσειρά *β δεξιό* ή *παραγόμενο* μέλος του κανόνα.

- *S* είναι ένα στοιχείο του *N,* το οποίο ονομάζεται *αρχικό σύμβολο* της γραμματικής.

Ονομάζουμε *γλώσσα* της γραμματικής *G* το σύνολο των συμβολοσειρών *L(G)* που μπορούν να παραχθούν με αυτό τον τρόπο. Η διαδικασία αυτή ορίζεται πιο αυστηρά στη συνέχεια.

Κάθε γραμματική *G = (T,* N, *P, S)* ορίζει μια γλώσσα $L(G) \in T^*$, η οποία αποτελείται ακριβώς από τις συμβολοσειρές που παράγονται από την *G.* Λέμε τότε ότι η γραμματική *G παράγει* τη γλώσσα *L(G)*. Η γλώσσα αυτή ορίζεται ως:

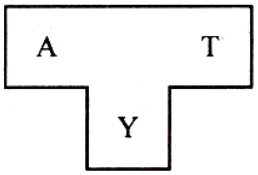
$$L(G) = \{ \alpha \in T^* \mid S =>^+ \alpha \}$$

Μια γλώσσα *L* θα λέγεται *τυπική* (formal) αν υπάρχει γραμματική *G* που να την παράγει.

3.2 ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Ο *μεταγλωττιστής* (compiler) είναι ένα πρόγραμμα το οποίο δέχεται ως είσοδο το κείμενο ενός προγράμματος, γραμμένο σε μια γλώσσα L_A και παράγει ως έξοδο ένα ισοδύναμο πρόγραμμα γραμμένο σε μια γλώσσα L_T. Η διαδικασία αυτή ονομάζεται *μεταγλώττιση* (compilation). Η γλώσσα L_A ονομάζεται *αρχική γλώσσα* (source language), ενώ η γλώσσα L_T ονομάζεται *τελική γλώσσα* (target language). Οι δυο αυτές γλώσσες είναι συχνά πολύ διαφορετικές: η L_A είναι συνήθως μια γλώσσα προγραμματισμού υψηλού επιπέδου, π.χ. η Pascal, και η L_T είναι συνήθως η γλώσσα μηχανής κάποιου συγκεκριμένου υπολογιστή. Το πρόγραμμα που δίνεται ως είσοδος στο μεταγλωττιστή λέγεται *αρχικό πρόγραμμα* (source program) και το ισοδύναμο πρόγραμμα που παράγεται στην έξοδό του ονομάζεται *τελικό πρόγραμμα* (target program).

Ο υπολογιστής όπου γίνεται η μεταγλώττιση και ο υπολογιστής όπου γίνεται η εκτέλεση των προγραμμάτων είναι συνήθως ο ίδιος. Αν πρόκειται για δυο διαφορετικούς υπολογιστές, ο μεταγλωττιστής ονομάζεται *διαμεταγλωττιστής* (cross compiler).



Σχήμα 3.1 Ο μεταγλωττιστής ως διάγραμμα Τ

Για να εκφρασθούν ρητά οι τρεις γλώσσες που χαρακτηρίζουν ένα μεταγλωττιστή

έχουν επινοηθεί και χρησιμοποιούνται δυο συμβολισμοί. Ο πρώτος χρησιμοποιεί διαγράμματα σε μορφή κεφαλαίου Τ (Σχήμα 3.1). Ο δεύτερος χρησιμοποιεί ένα κεφαλαίο γράμμα με δυο πάνω δείκτες, που συμβολίζουν την αρχική και τελική γλώσσα αντίστοιχα, και έναν κάτω δείκτη που συμβολίζει τη γλώσσα υλοποίησης, π.χ. $C_Y^{A,T}$

3.3 ΕΙΔΗ ΜΕΤΑΓΛΩΤΤΙΣΤΩΝ ΚΑΙ ΣΥΝΑΦΗ ΕΡΓΑΛΕΙΑ

Οι περισσότεροι μεταγλωττιστές έχουν ως αρχική γλώσσα μια συγκεκριμένη γλώσσα προγραμματισμού υψηλού επιπέδου και η τελική γλώσσα τους είναι η γλώσσα μηχανής ενός συγκεκριμένου υπολογιστή. Οι μεταγλωττιστές αυτοί ονομάζονται *απλοί* και χρησιμοποιούνται ευρέως για την ανάπτυξη λογισμικού.

Την αντίστροφη εργασία, όπως υποδηλώνει και το όνομά τους, κάνουν οι *αντίστροφοι μεταγλωττιστές* (decompilers). Σε αυτούς, αρχική γλώσσα είναι η γλώσσα μηχανής ενός υπολογιστή και τελική γλώσσα είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου. Οι αντίστροφοι μεταγλωττιστές είναι χρήσιμοι για την κατανόηση και την αποσφαλμάτωση προγραμμάτων γραμμένων σε γλώσσα μηχανής.

Μια τρίτη πολύ χρήσιμη κατηγορία μεταγλωττιστών είναι οι *μεταμεταγλωττιστές* (meta-compilers), που χρησιμοποιούνται για την υλοποίηση μεταγλωττιστών. Οι μετα-μεταγλωττιστές χωρίζονται σε δυο υποκατηγορίες, που διαφέρουν ως προς τη λειτουργία τους: *οι προσαρμοζόμενοι μεταμεταγλωττιστές* (adjustable meta-compilers), *οι μεταμεταγλωττιστές γεννήτορες* (meta-compiler generators).

Τρεις ειδικές περιπτώσεις προγραμμάτων που συναντώνται συχνά στην πράξη και μπορούν να θεωρηθούν μεταγλωττιστές είναι οι εξής: οι *προεπεξεργαστές,* οι *συμβολομεταφραστές* και οι *γεννήτορες προγραμμάτων*. Το κοινό στοιχείο στις δυο πρώτες περιπτώσεις είναι ότι η αρχική γλώσσα του μεταγλωττιστή δε διαφέρει πολύ από την τελική. Οι *γεννήτορες προγραμμάτων* (program generators) χρησιμοποιούνται για την αυτόματη κατασκευή προγραμμάτων βασισμένων σε τυπικές προδιαγραφές. Τα μεταεργαλεία flex και bison για την αυτόματη κατασκευή λεκτικών και συντακτικών αναλυτών αντίστοιχα αποτελούν παραδείγματα γεννητόρων προγραμμάτων.

Στην ανάπτυξη λογισμικού είναι επίσης χρήσιμα εργαλεία που δεν είναι μεταγλωττιστές, αλλά είτε είναι συναφή είτε χρησιμοποιούνται σε συνδυασμό με αυτούς.

Τα ποιο διαδεδομένα από αυτά είναι: οι *διερμηνείς* (interpreters), *η βιβλιοθήκη χρόνου εκτέλεσης* (run-time library), *οι διαχειριστές βιβλιοθηκών* (library managers), *οι συνδέτες* (linkers), *οι φορτωτές* (loaders), *οι εκδότες προγραμμάτων* (program editors), *οι εντοπιστές σφαλμάτων* (debuggers) και οι *στατιστικοί αναλυτές* (profilers).

Πολλοί απλοί μεταγλωττιστές χρησιμοποιούν εξωτερικούς προεπεξεργαστές ή και συμβολομεταφραστές στην υλοποίησή τους. Οι περισσότεροι μεταγλωττιστές χρησιμοποιούν εξωτερικούς συνδέτες και βιβλιοθήκες. Επίσης, πολλοί μεταγλωττιστές συνεργάζονται με εκδότες προγραμμάτων, εντοπιστές σφαλμάτων, στατιστικούς αναλυτές και άλλα βοηθητικά εργαλεία. Σε μερικές περιπτώσεις όλα αυτά είναι ενσωματωμένα σε ένα *ολοκληρωμένο περιβάλλον προγραμματισμού* (integrated programming environment).

3.4 ΚΑΤΑΣΚΕΥΗ ΕΝΟΣ ΜΕΤΑΓΛΩΤΤΙΣΤΗ

Οι μεταγλωττιστές είναι συνήθως προγράμματα πολύπλοκα. Η κατασκευή τους είναι μια επίπονη εργασία, εφικτή μόνο όταν γίνεται συστηματικά. Παραδοσιακά, η κατασκευή των μεταγλωττιστών γινόταν και εξακολουθεί να γίνεται χειρωνακτικά. Ο αυτόματος τρόπος κατασκευής δε χρησιμοποιείται ιδιαίτερα στην πράξη, κυρίως λόγω των αδυναμιών που παρουσιάζουν οι σημερινοί μεταμεταγλωττιστές. Με την τελειοποίηση των τελευταίων όμως, είναι επόμενο αυτός ο τρόπος να επικρατήσει. Ο πλέον διαδεδομένος στις μέρες μας τρόπος κατασκευής μεταγλωττιστών είναι υβριδικός: ορισμένα τμήματα του μεταγλωττιστή, όπως ο λεκτικός και ο συντακτικός αναλυτής, κατασκευάζονται αυτόματα χρησιμοποιώντας ειδικά εργαλεία, ενώ ο υπόλοιπος μεταγλωττιστής κατασκευάζεται χειρωνακτικά.

3.4.1 Ορισμός του προβλήματος

Στην περίπτωση της κατασκευής ενός μεταγλωττιστή, ο ορισμός του προβλήματος συνίσταται στον ορισμό δύο γλωσσών: της αρχικής και της τελικής γλώσσας. Γενικά, για να ορισθεί μια γλώσσα πρέπει να ορισθεί *η σύνταξη* (syntax) και *η σημασιολογία* (semantics) της. Η σύνταξη προσδιορίζει ποιες συμβολοσειρές ανήκουν στη γλώσσα, δηλαδή αποτελούν έγκυρα προγράμματα. Η σημασιολογία ορίζει ποια είναι η σημασία

των συντακτικά σωστών προγραμμάτων. Έτσι, ο ορισμός του προβλήματος της κατασκευής ενός μεταγλωττιστή συνίσταται στον ορισμό της σύνταξης και της σημασιολογίας της αρχικής και της τελικής γλώσσας.

Η περιγραφή της σύνταξης των τεχνητών γλωσσών γίνεται με απλό και κατανοητό τρόπο. Για το σκοπό αυτό έχουν αναπτυχθεί τυπικοί συμβολισμοί όπως είναι *η μορφή Backus Naur* (Backus Naur Form - BNP) και τα *συντακτικά διαγράμματα.* Ο ορισμός της σημασιολογίας είναι πολύ πιο δύσκολος. Αναλυτική παρουσίασή τους θα γίνει στο επόμενο κεφάλαιο.

3.4.2 Απαιτήσεις

Οι βασικές απαιτήσεις από ένα μεταγλωττιστή είναι:

- Να λειτουργεί *σωστά,* δηλαδή να μεταφράζει σωστά ένα πρόγραμμα της αρχικής γλώσσας σε ένα ισοδύναμο πρόγραμμα της τελικής γλώσσας.
- Να μπορεί ο μεταγλωττιστής να μεταφράζει προγράμματα *αυθαίρετα μεγάλου μεγέθους,* όταν φυσικά το επιτρέπει η διαθέσιμη μνήμη του υπολογιστή.

Εκτός από αυτές τις δύο βασικές απαιτήσεις υπάρχουν και ορισμένες άλλες που είναι στην ουσία περιορισμοί στη λύση αυτού του προβλήματος. Μερικές από τις επιπρόσθετες αυτές απαιτήσεις είναι οι ακόλουθες:

- Να παράγει *αποδοτικό κώδικα,* δηλαδή τα μεταγλωττισμένα προγράμματα να εκτελούνται γρήγορα και να είναι μικρά σε μέγεθος.
- Να έχει μικρό *χρόνο μεταγλώττισης.*
- Να έχει *μικρές απαιτήσεις* σε μνήμη κατά τη μεταγλώττιση. Το μέγεθος του μεταγλωττιστή και οι απαιτήσεις του σε μνήμη δεν είναι τόσο σημαντικά σήμερα, όταν οι περισσότεροι υπολογιστές διαθέτουν μνήμη πολλών megabytes.
- Να δίνει *κατανοητά διαγνωστικά μηνύματα,* ούτως ώστε να διευκολύνει τον προγραμματιστή στη διόρθωση των σφαλμάτων.
- Να έχει τη δυνατότητα *ανάκαμψης ύστερα από ανακάλυψη σφαλμάτων,* ώστε να μπορεί να υποδεικνύει στον προγραμματιστή όσο το δυνατόν περισσότερα από τα σφάλματα που υπάρχουν στο πρόγραμμα.
- Να είναι *μεταφέρσιμος.* Ένα πρόγραμμα ονομάζεται μεταφέρσιμο αν είναι αρκετά

απλό, από πλευράς χρόνου και κόπου, το να τρέξει σε διαφορετικό τύπο υπολογιστών. Οι περιορισμοί αυτοί είναι αντικρουόμενοι και ο κατασκευαστής βρίσκεται ενίοτε σε πραγματικά δύσκολη θέση προκειμένου να πάρει αποφάσεις. Συχνά στην πράξη οι περιορισμοί ιεραρχούνται και ικανοποιούνται κατά προτεραιότητα.

3.4.3 Φάσεις της μεταγλώττισης

Ένας μεταγλωττιστής μπορεί να σχεδιαστεί με πολλούς τρόπους. Η πείρα μέχρι τώρα έχει δείξει ότι η δομή που είναι κατάλληλη για να χρησιμοποιηθεί ως μια βάση σχεδιασμού μεταγλωττιστών, είναι η μεταγλώττιση να χωρίζεται σε επιμέρους *φάσεις* (phases), κάθε μια από τις οποίες δέχεται ως είσοδο ένα πρόγραμμα ισοδύναμο με το αρχικό, σε κάποια μορφή, και δίνει ως έξοδο το ίδιο πρόγραμμα σε μια άλλη μορφή.

Κατά τη διάρκεια κάθε φάσης της μεταγλώττισης ενδέχεται να εντοπισθούν σφάλματα. Στην περίπτωση αυτή καλείται ο *χειριστής σφαλμάτων* (error handler). Οι φάσεις στην κατασκευή ενός μεταγλωττιστή είναι η λεκτική ανάλυση, η συντακτική ανάλυση, η σημασιολογική ανάλυση, η παραγωγή του ενδιάμεσου κώδικα, η βελτιστοποίηση και τέλος η παραγωγή του τελικού κώδικα.

- Στη φάση της *λεκτικής ανάλυσης* διαβάζεται από το αρχικό πρόγραμμα, ένας - ένας οι χαρακτήρας κάθε φορά, και γίνεται μια ομαδοποίηση των χαρακτήρων σε *λεκτικές μονάδες* (tokens) που δίνονται ως είσοδος στην επόμενη φάση.
- Κατά τη *συντακτική ανάλυση* διαβάζονται οι λεκτικές μονάδες και ομαδοποιούνται σε συντακτικές μονάδες βάσει μιας γραμματικής που περιγράφει τη σύνταξη της αρχικής γλώσσας. Στο τέλος αυτής της φάσης έχει σχηματισθεί ένα δέντρο που απεικονίζει τη συντακτική δομή του αρχικού προγράμματος. Το δέντρο αυτό ονομάζεται *συντακτικό δέντρο* (syntax tree).
- Στη φάση της *σημασιολογικής ανάλυσης*, που συχνά ονομάζεται *στατική σημασιολογία (static semantics)*, ελέγχεται το πρόγραμμα για σημασιολογικά σφάλματα και συλλέγονται πληροφορίες που χρειάζονται για την παραγωγή του ενδιάμεσου κώδικα. Το αποτέλεσμα αυτής της φάσης είναι και πάλι το συντακτικό δέντρο, πιθανώς τροποποιημένο και συμπληρωμένο με πληροφορίες σημασιολογικής φύσης.

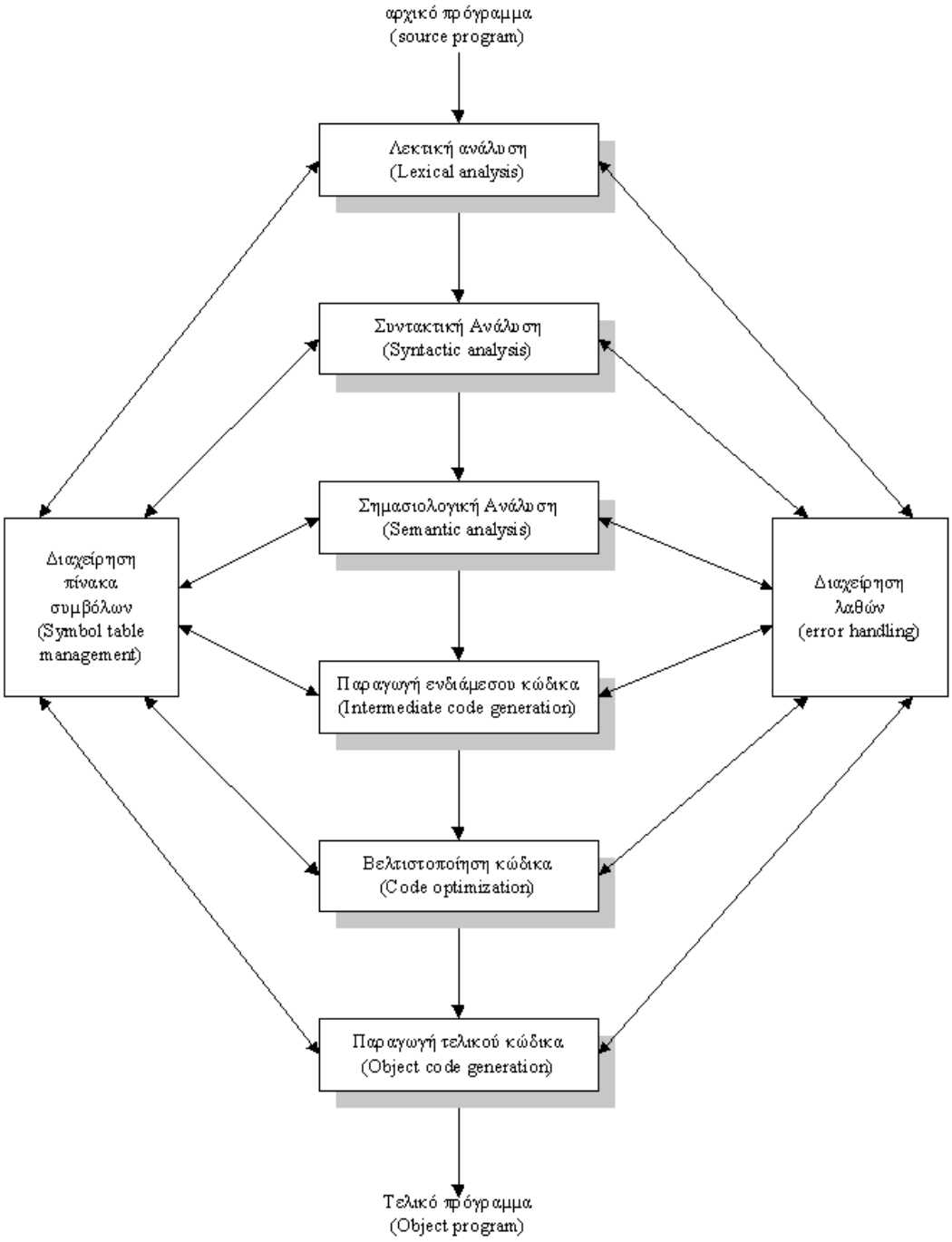
- Στη φάση *παραγωγής ενδιάμεσου κώδικα* παράγεται κώδικας σε μια *ενδιάμεση γλώσσα (intermediate language)* με βάση το συντακτικό δέντρο. Ο ενδιάμεσος κώδικας μπορεί να θεωρηθεί ως ένα ισοδύναμο πρόγραμμα για μια αφηρημένη μηχανή εκτέλεσης.
- Η *βελτιστοποίηση* έχει ως σκοπό να κάνει αποδοτικότερο το τελικό πρόγραμμα. Κύρια μέτρα απόδοσης, σε αυτή την περίπτωση είναι η ταχύτητα εκτέλεσης του τελικού προγράμματος και το μέγεθός του. Η βελτιστοποίηση είναι φυσικά προαιρετική σε ένα μεταγλωττιστή και μπορεί να εφαρμοστεί στον ενδιάμεσο ή στον τελικό κώδικα.
- Κατά την φάση *παραγωγής του τελικού κώδικα,* μετατρέπεται ο ενδιάμεσος κώδικας σε τελικό κώδικα. Το αποτέλεσμά της είναι το τελικό μεταγλωττισμένο πρόγραμμα.

3.4.4 Επιλογές κατά την υλοποίηση

Για την υλοποίηση των πρώτων μεταγλωττιστών αρχικά χρησιμοποιήθηκε γλώσσα μηχανής ή συμβολική γλώσσα. Αργότερα, για το σκοπό αυτό χρησιμοποιήθηκαν γλώσσες προγραμματισμού υψηλού επιπέδου, όπως η FORTRAN, η ALGOL, η Pascal, η C, η C++ και η Java. Είναι δυνατόν να χρησιμοποιηθεί για την υλοποίηση ενός μεταγλωττιστή, η ίδια η αρχική του γλώσσα, π.χ. ένας μεταγλωττιστής Pascal να υλοποιηθεί σε Pascal.

Η υλοποίηση ενός μεταγλωττιστή δεν ακολουθεί πιστά τις φάσεις, αλλά οργανώνεται σε ένα ή περισσότερα *περάσματα* (passes). Κάθε πέρασμα υλοποιεί μια ή περισσότερες φάσεις. Διαβάζει την είσοδό του, που είτε είναι το αρχικό πρόγραμμα ή το αποτέλεσμα του προηγούμενου περάσματος, επιτελεί τη λειτουργία του και αποθηκεύει τα αποτελέσματά του, που είτε είναι το τελικό πρόγραμμα ή είναι ενδιάμεσα αρχεία που θα διαβαστούν από το επόμενο πέρασμα.

Στην πράξη χρησιμοποιείται ένα δεύτερο είδος ομαδοποίησης, όπου ορισμένες φάσεις σχετίζονται μόνο με την αρχική γλώσσα και είναι ανεξάρτητες από την τελική. Οι φάσεις αυτές (ή τα περάσματα που τις περιέχουν) αποτελούν το *εμπρόσθιο τμήμα* (front-end) του μεταγλωττιστή, το οποίο τυπικά περιλαμβάνει τη λεκτική, τη συντακτική και τη σημασιολογική ανάλυση, την παραγωγή και τη βελτιστοποίηση ενδιάμεσου κώδικα.



Σχήμα 3.2 Δομή ενός μεταγλωττιστή

Αντίθετα, άλλες φάσεις σχετίζονται μόνο με την τελική γλώσσα και τοποθετούνται στο *οπίσθιο τμήμα*(back-end) του μεταγλωττιστή. Αυτό περιλαμβάνει την παραγωγή και τη βελτιστοποίηση τελικού κώδικα. Η διαχείριση του πίνακα συμβόλων και η διαχείριση σφαλμάτων είναι λειτουργίες που μοιράζονται ανάμεσα στα δυο αυτά τμήματα.

3.4.5 Έλεγχος ορθότητας

Οι μεταγλωττιστές είναι βασικά εργαλεία ανάπτυξης λογισμικού και όπως προαναφέρθηκε, η ορθότητα της λειτουργίας τους πρέπει να είναι εξασφαλισμένη. Για το λόγο αυτό, στην κατασκευή ενός μεταγλωττιστή δίνεται πάντα μεγάλη έμφαση στη διαδικασία του ελέγχου της ορθότητάς του.

Αν οι μεταγλωττιστές έπρεπε να επεξεργάζονται μόνο σωστά προγράμματα, η σχεδίαση και η υλοποίηση τους θα ήταν κατά πολύ απλούστερη. Ωστόσο, πάρα πολύ συχνά, γράφονται λανθασμένα προγράμματα και ένας καλός μεταγλωττιστής οφείλει να βοηθά τον προγραμματιστή στον εντοπισμό και στη διόρθωση των σφαλμάτων, σ' αυτές τις περιπτώσεις.

Ένα πρόγραμμα μπορεί να περιέχει σφάλματα πολλών ειδών. Τα κυριότερα από αυτά κατατάσσονται σε πέντε κατηγορίες: *λεκτικά σφάλματα, συντακτικά σφάλματα, σημασιολογικά σφάλματα, σφάλματα εκτέλεσης* και *λογικά σφάλματα.*

Η εκτύπωση διαγνωστικών μηνυμάτων είναι η κύρια μέριμνα του τμήματος του μεταγλωττιστή που ασχολείται με τη διαχείριση των σφαλμάτων. Τα διαγνωστικά μηνύματα διακρίνονται σε τέσσερις κατηγορίες κατά φθίνουσα σειρά σοβαρότητας: *εσωτερικά σφάλματα* (internal errors), που υποδηλώνουν κάποιο σοβαρό πρόβλημα στη λειτουργία του μεταγλωττιστή, *σφάλματα* (errors), που υποδεικνύουν ότι το αρχικό πρόγραμμα δεν είναι έγκυρο λεκτικά, συντακτικά ή σημασιολογικά *προειδοποιητικά μηνύματα* (warnings) που επισημαίνουν στον προγραμματιστή ότι ενδέχεται να έχει παραβιαστεί κάποιος κανόνας της γλώσσας, *απλά μηνύματα* (plain messages), τα οποία συνήθως είναι ανεξάρτητα από τη μεταγλώττιση.

Είναι όμως σημαντικό, ο εντοπισμός του πρώτου σφάλματος να μην έχει ως αποτέλεσμα τη διακοπή της μεταγλώττισης. Τις περισσότερες φορές, η μεταγλώττιση μπορεί να συνεχιστεί, επιτρέποντας έτσι τον εντοπισμό και άλλων σφαλμάτων. Η διαδικασία αυτή ονομάζεται *ανάνηψη από σφάλματα* (error recovery).

ΚΕΦΑΛΑΙΟ 4

ΤΥΠΙΚΕΣ ΓΛΩΣΣΕΣ

4.1 ΓΕΝΝΗΤΙΚΑ ΚΑΙ ΑΝΑΓΝΩΡΙΣΤΙΚΑ ΜΟΝΤΕΛΑ

Στην κατασκευή μεταγλωττιστών, αλλά και γενικότερα στην επιστήμη υπολογιστών, αναγκαζόμαστε να περιοριστούμε σε αρκετά στενότερες κλάσεις γλωσσών. Η ευρύτερη από αυτές είναι η κλάση των *υπολογίσιμων γλωσσών* (computable languages), δηλαδή των γλωσσών για τις οποίες μπορούν να κατασκευαστούν προγράμματα που να αποφαίνονται αν μια τυχαία συμβολοσειρά ανήκει ή όχι σε αυτές. Στην κατασκευή μεταγλωττιστών, συχνά περιοριζόμαστε ακόμη περισσότερο και χρησιμοποιούμε γλώσσες οι οποίες όχι απλώς είναι υπολογίσιμες, αλλά υπολογίσιμες μέσα στα πλαίσια του εύλογου χρόνου που μπορεί να περιμένει ένας άνθρωπος, αλλά και της μνήμης που διαθέτει ένας μεταγλωττιστής.

4.1.1 Ιεραρχία Chomsky

Οι γραμματικές και αντίστοιχα οι γλώσσες που παράγονται από αυτές, κατατάσσονται σε τέσσερις κλάσεις ανάλογα με τη μορφή των κανόνων τους. Οι κλάσεις αυτές, συνθέτουν μια ιεραρχία με την έννοια ότι κάθε κλάση είναι υπερσύνολο των επομένων της. Η ιεραρχία αυτή ονομάζεται *ιεραρχία Chomsky* (Chomsky hierarchy), από το γλωσσολόγο Noam Chomsky που την όρισε:

- *Γραμματικές τύπου 0.* Στην κλάση αυτή ανήκουν όλες οι γραμματικές, χωρίς κανένα περιορισμό στους κανόνες τους.
- *Γραμματικές τύπου 1* ή *γραμματικές με συμφραζόμενα* (context-sensitive grammars). Στην κλάση αυτή ανήκουν οι γραμματικές με κανόνες της μορφής $\alpha \rightarrow \beta$, όπου η συμβολοσειρά α περιέχει τουλάχιστον ένα μη τερματικό σύμβολο και $|\alpha| \leq |\beta|$.
- *Γραμματικές τύπου 2* ή *γραμματικές χωρίς συμφραζόμενα* (context-free grammars). Στην κλάση αυτή ανήκουν οι γραμματικές με κανόνες της μορφής $A \rightarrow \alpha$, όπου A ένα μη τερματικό σύμβολο και α συμβολοσειρά.

- *Γραμματικές τύπου 3* ή *κανονικές γραμματικές* (regular grammars). Στην κλάση αυτή ανήκουν οι γραμματικές με κανόνες που έχουν μια από τις εξής μορφές: $A \rightarrow aB$, $A \rightarrow$ a, ή $A \rightarrow \epsilon$, όπου A και B μη τερματικά σύμβολα και a τερματικό σύμβολο.

Μια γλώσσα ονομάζεται τύπου n (n = 0,1,2,3) αν υπάρχει γραμματική τύπου *n* που να την παράγει. Από τον παραπάνω ορισμό είναι προφανές ότι το σύνολο των γλωσσών τύπου m, με m > *n*, είναι υποσύνολο των γλωσσών τύπου *n*.

4.1.2 Αναγνωριστές

Ας θεωρήσουμε μια αφηρημένη μηχανή *M,* η οποία παίρνει ως είσοδο συμβολοσειρές ενός ορισμένου αλφαβήτου Σ και δίνει ως έξοδο "ναι" ή "όχι". Μια τέτοια μηχανή ονομάζεται *αναγνωριστής* (recognizer). Το σύνολο των συμβολοσειρών για τις οποίες η μηχανή απαντά "ναι" ονομάζεται *γλώσσα* του αναγνωριστή και συμβολίζεται με *L(M)*. Στην περίπτωση αυτή λέμε ότι ο αναγνωριστής *αναγνωρίζει* τη γλώσσα *L(M).*

Οι αφηρημένες μηχανές, που συχνά ονομάζονται και αυτόματα (automata), κατατάσσονται ανάλογα με την πολυπλοκότητα και την υπολογιστική δύναμη. Στη θεωρία υπολογισμού ιδιαίτερη θέση έχουν οι ακόλουθοι τέσσερις τύποι αφηρημένων μηχανών και οι ισοδύναμές τους:

- *Η μηχανή Turing* (Turing machine)[17], που σχεδιάστηκε από τον Alan Turing στη δεκαετία του 1940, είναι το γενικότερο δυνατό υπολογιστικό μοντέλο. Έχει τις δυνατότητες ενός πλήρους σύγχρονου υπολογιστή, χωρίς περιορισμούς χρόνου και μνήμης. Αποτελεί τη βάση της θεωρίας υπολογισμού και της θεωρίας πολυπλοκότητας. Είναι ένα θεωρητικό μοντέλο το οποίο δεν κατασκευάστηκε ποτέ. Η μηχανή Turing αποτελείται από μια ταινία άπειρου μήκους και μια κεφαλή ανάγνωσης-εγγραφής που μπορεί να κινείται οπουδήποτε πάνω σε αυτή την ταινία. Θεωρούμε ότι σε διακριτές θέσεις πάνω στην ταινία μπορούν να γράφονται χαρακτήρες. Κάθε στιγμή, η μηχανή μπορεί να βρίσκεται σε μια κατάσταση που ανήκει σε ένα πεπερασμένο σύνολο καταστάσεων. Αρχικά, η μηχανή βρίσκεται σε κάποια αρχική κατάσταση, η ταινία περιέχει μια συμβολοσειρά και η κεφαλή βρίσκεται στην αρχή της συμβολοσειράς. Στη συνέχεια, ανάλογα με το χαρακτήρα που βρίσκεται στη θέση της κεφαλής και με την τρέχουσα κατάσταση, συμβαίνουν τα ακόλουθα: (i) η μηχανή μεταφέρεται σε μια νέα κα-

τάσταση ή τερματίζει, (ii) ένας νέος χαρακτήρας γράφεται στη θέση της ταινίας όπου βρίσκεται η κεφαλή, και (iii) η κεφαλή μετακινείται ένα χαρακτήρα αριστερά ή δεξιά ή παραμένει στάσιμη. Η περίπτωση τερματισμού της μηχανής συνοδεύεται είτε από την αναγνώριση της συμβολοσειράς ή από την απόρριψή της.

- *Η γραμμικά περιορισμένη μηχανή Turing* (linearly bounded Turing Machine) είναι μια μηχανή Turing με έναν πρόσθετο περιορισμό μνήμης: Αν η συμβολοσειρά που δεχεται ως είσοδο έχει μήκος *n,* η μηχανή μπορεί να χρησιμοποιεί το πολύ *O(n)* θέσεις μνήμης.
- *Το αυτόματο στοίβας* (push-down automaton).
- *Το πεπερασμένο αυτόματο* (finite automaton).

Οι δύο τελευταίοι τύποι αναγνωριστών περιγράφονται αναλυτικά στην συνέχεια του κεφαλαίου, λόγω της συνάφειας που έχουν με τους μεταγλωττιστές.

Υπάρχει μια ενδιαφέρουσα αντιστοιχία ανάμεσα στις κλάσεις των τυπικών γλωσσών που περιγράφηκαν στην προηγούμενη ενότητα και στους παραπάνω τύπους αφηρημένων μηχανών (αναγνωριστών). Μια γλώσσα είναι τύπου *n (n* = 0,1,2,3), αν και μόνο αν, μπορεί να αναγνωριστεί από μια μηχανή του αντίστοιχου τύπου όπως φαίνεται στον πίνακα 4.1.

Πίνακας 4.1. Αντιστοιχία τυπικών γλωσσών και αναγνωριστικών μηχανών.

n	Κλάση τυπικών γλωσσών	Αντίστοιχος τύπος μηχανών
0	Υπολογίσιμες γλώσσες	Μηχανές Tuήng
1	Γλώσσες με συμφραζόμενα	Γραμμικά περιορισμένες μηχανές Turing
2	Γλώσσες χωρίς συμφραζόμενα	Αυτόματα στοίβας
3	Κανονικές γλώσσες	Πεπερασμένα αυτόματα

Ας σημειωθεί ότι ο ορισμός μιας γλώσσας με τη βοήθεια κάποιας αφηρημένης μηχανής, είναι ποιοτικά διαφορετικός από τον ορισμό της με τη βοήθεια μιας γραμματικής. Η γραμματική δίνει ένα μοντέλο, που περιγράφει πώς μπορεί να παραχθεί κάθε συμβολοσειρά της γλώσσας και για το λόγο αυτό οι γραμματικές ονομάζονται *γεννητικά μοντέλα* (generating models) για τον ορισμό γλωσσών. Αντίθετα, οι αφηρημένες μηχανές επικεντρώνονται στο πώς μπορεί να επαληθευθεί αν μια συμβολοσειρά ανήκει ή όχι σε μία

γλώσσα και για το λόγο αυτό ονομάζονται *αναγνωριστικά μοντέλα* (recognizing models).

4.2 ΚΑΝΟΝΙΚΕΣ ΓΛΩΣΣΕΣ

Οι κανονικές γλώσσες αποτελούν απαραίτητα εργαλεία στην κατασκευή μεταγλωττιστών και συγκεκριμένα στην υλοποίηση της φάσης της λεκτικής ανάλυσης. Στις κανονικές γλώσσες, ανήκουν οι *κανονικές εκφράσεις,* ένας εναλλακτικός τρόπος περιγραφής μιας κανονικής γλώσσας που χρησιμοποιείται στην πράξη περισσότερο από τις κανονικές γραμματικές, λόγω της κομψότητας και της λακωνικότητάς του και τα πεπερασμένα αυτόματα που, όπως ήδη αναφέρθηκε, είναι οι αφηρημένες μηχανές που αναγνωρίζουν κανονικές γλώσσες.

Οι *κανονικές εκφράσεις* (regular expressions), είναι ο τρόπος που παραδοσιακά χρησιμοποιείται για την περιγραφή των λεκτικών μονάδων μιας γλώσσας προγραμματισμού. Συμβολίζονται με τα μικρά λατινικά γράμματα *r,* s, *t,* κ.λ.π. Κάθε κανονική έκφραση *r* προσδιορίζει μια γλώσσα που συμβολίζεται με *L(r).*

Έστω ένα αλφάβητο Σ. Οι κανονικές εκφράσεις πάνω στο Σ ορίζονται αναδρομικά ως εξής:

- Η κενή συμβολοσειρά *ε* είναι κανονική έκφραση και $L(\epsilon) = \{\epsilon\}$.
- Κάθε σύμβολο a του αλφαβήτου Σ είναι κανονική έκφραση και $L(a) = \{ a \}$.
- Αν *r* και s είναι κανονικές εκφράσεις, τότε η *(rs)* είναι κανονική έκφραση και αντιστοιχεί στην παράθεση των *r* και *s*. Δηλαδή $L(rs) = L(r)L(s)$.
- Αν *r* και s είναι κανονικές εκφράσεις, τότε η *(r|s)* είναι κανονική έκφραση και αντιστοιχεί στη διάζευξη των *r* και s. Δηλαδή $L(r|s) = L(r) \cup L(s)$.
- Αν *r* είναι κανονική έκφραση, τότε η (r^*) είναι κανονική έκφραση και αντιστοιχεί στην κλειστότητα Kleene της *r*. Δηλαδή $L(r^*) = (L(r))^*$.

Επομένως, για την κατασκευή μιας κανονικής έκφρασης, τα βασικά δομικά στοιχεία είναι τα σύμβολα του αντίστοιχου αλφαβήτου και η κενή συμβολοσειρά, ενώ οι επιτρεπόμενες πράξεις είναι η παράθεση, η διάζευξη και το κλείσιμο Kleene.

4.2.1 Πεπερασμένα Αυτόματα

Τα *πεπερασμένα αυτόματα* (finite automata), είναι αφηρημένες μηχανές που μπορούν να χρησιμοποιηθούν ως αναγνωριστές για κανονικές γλώσσες. Διακρίνουμε τρία είδη πεπερασμένων αυτομάτων, που διαφέρουν ανάλογα με τον τρόπο που πραγματοποιούνται οι μεταβάσεις από κατάσταση σε κατάσταση:

- τα ντετερμινιστικά πεπερασμένα αυτόματα (ΝΠΑ). Ένα *ντετερμινιστικό πεπερασμένο αυτόματο* (deterministic finite automaton) είναι μια διατεταγμένη πεντάδα $M = (A, Q, \delta, q_0, P)$, όπου:
    - A είναι το *αλφάβητο* του αυτομάτου,
    - Q είναι ένα μη κενό και πεπερασμένο *σύνολο καταστάσεων,*
    - $\delta : Q \times A \rightarrow Q$ είναι η *συνάρτηση μετάβασης,*
    - $q_0 \in Q$ είναι μια ειδική κατάσταση που ονομάζεται *αρχική κατάσταση,* και $F \subseteq Q$ είναι ένα υποσύνολο του συνόλου των καταστάσεων, τα στοιχεία του οποίου ονομάζονται *τελικές καταστάσεις.*

Η λειτουργία ενός ΝΠΑ γίνεται ως εξής. Αρχικά, το αυτόματο βρίσκεται στην κατάσταση q_0. Από τη συμβολοσειρά εισόδου διαβάζεται ένα σύμβολο κάθε φορά. Αν κάποια στιγμή το αυτόματο βρίσκεται στην κατάσταση q και διαβαστεί το σύμβολο a, τότε το αυτόματο μεταβαίνει στην κατάσταση $\delta(q,$ a). Η διαδικασία συνεχίζεται μέχρι να εξαντληθεί η συμβολοσειρά εισόδου. Στην περίπτωση αυτή, αν η κατάσταση στην οποία βρίσκεται το αυτόματο μετά το τέλος της συμβολοσειράς εισόδου ανήκει στο σύνολο F των τελικών καταστάσεων, τότε το αυτόματο αναγνωρίζει τη συμβολοσειρά εισόδου. Διαφορετικά την απορρίπτει.

- Τα μη ντετερμινιστικά πεπερασμένα αυτόματα (ΜΠΑ). Ένα *μη ντετερμινιστικό πεπερασμένο αυτόματο* (nondeterministic finite automaton), ορίζεται όπως και ένα ΝΠΑ, με τη διαφορά ότι η συνάρτηση μετάβασης δ δεν επιστρέφει μια μοναδική κατάσταση, αλλά ένα σύνολο πιθανών επόμενων καταστάσεων.
- Τα μη ντετερμινιστικά πεπερασμένα αυτόματα με μηδενικές μεταβάσεις (ΜΠΑ - ϵ). Ένα *μη ντετερμινιστικό πεπερασμένο αυτόματο με μηδενικές μεταβάσεις* (nondeterministic finite automaton with zero transitions) ορίζεται όπως ένα ΜΠΑ, με

τη διαφορά ότι επιτρέπονται και "αυθόρμητες" μεταβάσεις, χωρίς να διαβάζονται χαρακτήρες από τη συμβολοσειρά εισόδου.

Τα τρία αυτά είδη πεπερασμένων αυτομάτων, έχουν παρεμφερείς ορισμούς και είναι ισοδύναμα από πλευράς υπολογιστικής ικανότητας.

4.3 ΓΛΩΣΣΕΣ ΧΩΡΙΣ ΣΥΜΦΡΑΖΟΜΕΝΑ

Οι γλώσσες χωρίς συμφραζόμενα, είναι ιδιαίτερα χρήσιμες για τον ορισμό της σύνταξης των γλωσσών προγραμματισμού και για την κατασκευή συντακτικών αναλυτών. Θα μελετήσουμε τις γραμματικές χωρίς συμφραζόμενα, που αποτελούν το γεννητικό μοντέλο για αυτές τις γλώσσες και συνήθως χρησιμοποιούνται στην περιγραφή μιας γλώσσας προγραμματισμού και στη συνέχεια τα αυτόματα στοίβας, που είναι οι αναγνωριστές αυτών των γλωσσών και βρίσκουν άμεση εφαρμογή στην κατασκευή μεταγλωττιστών.

4.3.1 Γραμματικές χωρίς συμφραζόμενα

Θα ονομάζουμε *αριστερότερη παραγωγή* (leftmost derivation) μιας συμβολοσειράς, την παραγωγή εκείνη στην οποία, σε κάθε βήμα αντικαθίσταται το αριστερότερο μη τερματικό σύμβολο. Αντίστοιχα, θα ονομάζουμε *δεξιότερη παραγωγή* (rightmost derivation) μιας συμβολοσειράς, την παραγωγή στην οποία σε κάθε βήμα αντικαθίσταται το δεξιότερο μη τερματικό σύμβολο. Αν η συμβολοσειρά β παράγεται από τη συμβολοσειρά α με κάποια αριστερότερη παραγωγή, τότε γράφουμε α $=>_L \beta$. Ομοίως, αν η β παράγεται από την α με κάποια δεξιότερη παραγωγή, τότε γράφουμε α $=>_R \beta$. Κάθε παραγωγή μιας γραμ-ματικής χωρίς συμφραζόμενα, μπορεί να μετατραπεί σε αριστερότερη ή δεξιότερη παρα-γωγή, αλλάζοντας απλώς τη σειρά αντικατάστασης των μη τερματικών συμβόλων και διατηρώντας ίδιους τους κανόνες που χρησιμοποιούνται σε κάθε αντικατάσταση.

Ένας εναλλακτικός τρόπος παράστασης μιας παραγωγής, είναι το *συντακτικό δέντρο* (parse tree). Συγκεκριμένα, συντακτικό δέντρο μιας γραμματικής G ονομάζεται ένα δεντρο που πληρεί τις παρακάτω προϋποθέσεις:

- Η ρίζα του δέντρου περιέχει το αρχικό μη τερματικό σύμβολο της γραμματικής.
- Κάθε κόμβος του δέντρου που δεν είναι φύλλο περιέχει ένα μη τερματικό σύμβολο

της γραμματικής.

- Κάθε φύλλο του δέντρου περιέχει ένα τερματικό σύμβολο της γραμματικής.
- Οι απόγονοι κάθε κόμβου του δέντρου που δεν είναι φύλλο, υπακούουν τους κανόνες παραγωγής της γραμματικής.

Έστω η γραμματική $G_1 = (T, N, P, S)$ όπου:

$T = \{a, b, c\}$

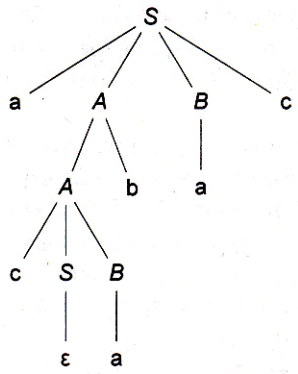
$N = \{S,A,B\}$

$$P = \begin{Bmatrix} S \to aAbc & A \to cSB & B \to bB \\ S \to \in & A \to Ab & B \to \alpha \end{Bmatrix}$$

Μια παραγωγή μπορεί να είναι η ακόλουθη:

S => $aABc$ => $aAbBc$ => $acSBbBc$ => $acSabBc$ => acabBc => acabac

η οποία παράγει τη συμβολοσειρά acabac. Οι προτασιακοί τύποι σ' αυτή την παραγωγή είναι επτά και το αντίστοιχο συντακτικό δέντρο είναι το ακόλουθο:



Η αριστερότερη και η δεξιότερη παραγωγή που αντιστοιχούν σε αυτό το συντακτικό δέντρο είναι:

$S =>_L aABc =>_L aAbBc =>_L acSBbBc =>_L acBbBc$

$=>_L$ acabBc $=>_L$ acabac

$S =>_R aABc =>_R$ aAac $=>_R$ aAbac $=>_R acSBbac$

$=>_R$ acSabac $=>_R$ acabac

Ένας συμβολισμός για γραμματικές που χρησιμοποιήθηκε πρώτη φορά για την περιγραφή της σύνταξης της γλώσσας προγραμματισμού ALGOL 60 και έγινε αργότερα πολύ δημοφιλής, είναι η *μορφή Backus Naur* (Backus Naur Form - BNF). Σύμφωνα με αυτό το συμβολισμό:

- Τα μη τερματικά σύμβολα περικλείονται σε γωνιακές παρενθέσεις, π.χ. <expr>.
- Το σύμβολο → στους κανόνες παραγωγής αντικαθίσταται από το σύμβολο ::=.
- Σε έναν κανόνα παραγωγής μπορούν να συνδυαστούν τα δεξιά μέλη περισσότερων κανόνων παραγωγής για το ίδιο μη τερματικό σύμβολο, χωρισμένα με το σύμβολο |. Για παράδειγμα, ο κανόνας παραγωγής <digit> ::= 0 | 1 είναι ισοδύναμος με τους δυο κανόνες παραγωγής <digit> ::= 0 και <digit> ::= 1.

Ο συμβολισμός που περιγράφηκε είναι ο καθαρός συμβολισμός BNF. Στην πράξη χρησιμοποιούνται διάφορες παραλλαγές του. Μια τέτοια παραλλαγή είναι ότι αντί του συμβόλου ::= χρησιμοποιείται το →. Μια άλλη παραλλαγή, είναι ότι τα τερματικά σύμβολα γράφονται με μικρά γράμματα και τα μη τερματικά σύμβολα γράφονται με κεφαλαία γράμματα.

Για λόγους συντομογραφίας, ο συμβολισμός BNF τροποποιήθηκε με αποτέλεσμα να προκύψει η *διευρυμένη μορφή Backus Naur* (Extended Backus Naur Form - EBNF)[17]. Και πάλι υπάρχουν διάφορες παραλλαγές αυτού του συμβολισμού. Μια τέτοια παραλλαγή είναι η ακόλουθη, που διαφέρει από το συμβολισμό BNF στα ακόλουθα σημεία:

- Τα τερματικά σύμβολα, γράφονται μέσα σε εισαγωγικά. Έτσι οι γωνιακές παρενθέσεις στα μη τερματικά σύμβολα περιττεύουν και χρησιμοποιούνται προαιρετικά.
- Επιτρέπεται η χρήση παρενθέσεων. Ο κανόνας $A ::= \alpha(\beta)\gamma$ είναι ισοδύναμος με τους δυο κανόνες $A ::= \alpha B\gamma$ *και* $B ::= \beta$.
- Συμβολοσειρές που περικλείονται σε αγκύλες είναι προαιρετικές. Ο κανόνας $A ::= \alpha[\beta]\gamma$ είναι ισοδύναμος με τον κανόνα $A ::= \alpha(\beta)\gamma \mid \alpha\gamma$.
- Επιτρέπεται η χρήση των συμβόλων * και +, που έχουν την ίδια ερμηνεία όπως και στις κανονικές εκφράσεις. Το σύμβολο * υποδηλώνει μηδέν ή περισσότερες εμφανίσεις του προηγούμενου συμβόλου, επομένως ο κανόνας $A ::= \alpha u^*\beta$ είναι ισοδύναμος με τους δυο κανόνες $A ::= \alpha B\beta$ και $B ::= \varepsilon \mid uB$. Το σύμβολο + υποδηλώνει μια ή περισσότερες εμφανίσεις του προηγούμενου συμβόλου και άρα ο κανόνας $A ::= \alpha u^+\beta$ είναι ισοδύναμος με $A ::= \alpha uu^*\beta$.

Στα παραπάνω, εννοείται ότι το μη τερματικό σύμβολο B δεν εμφανίζεται αλλού στη γραμματική.

Με τη χρήση του συμβολισμού EBNF, είναι δυνατόν σε μια γραμματική να αντικατασταθούν πλήρως οι αναδρομικοί κανόνες παραγωγής με κανόνες που περιέχουν τους

επαναληπτικούς τελεστές * και +.

Τα *συντακτικά διαγράμματα* (syntax diagrams), είναι ένας τρίτος δυνατός συμβολισμός για τον ορισμό γλωσσών χωρίς συμφραζόμενα. Χρησιμοποιήθηκαν από τον Wirth για την περιγραφή της σύνταξης της γλώσσας προγραμματισμού Pascal[13]. Σύμφωνα με αυτό το συμβολισμό, τα μη τερματικά πλαίσια γράφονται μέσα σε ορθογώνια, ενώ τα τερματικά σύμβολα γράφονται μέσα σε σχήματα με στρογγυλεμένες γωνίες. Η διαδοχή των συμβόλων παριστάνεται με βέλη.

Στο παρακάτω παράδειγμα, δίνεται μια γραμματική που περιγράφει τη σύνταξη των αριθμών χωρίς πρόσημο, όπως αυτή ορίζεται στη γλώσσα προγραμματισμού Pascal και μετά δίνεται η γραμματική αυτή με τους συμβολισμούς BNF, EBNF και των συντακτικών διαγραμμάτων (σχήμα 4.1), αντίστοιχα.

Η γραμματική ως τετράδα: G = (VN,VN,P,S) όπου

VN = {unsigned-number, integer-part, decimal-fraction, exponent-part, digit, S}

VT = {., +, -, E, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

P = {

S -> unsigned-number

unsigned-number -> integer-part decimal-fraction exponent-part

integer-part -> digit integer-part

integer-part -> digit

decimal-fraction -> .integer-part

decimal-fraction -> ε

exponent-part -> E sign integer-part

exponent-part -> ε

sign -> +

sign -> -

sign -> ε

digit -> 0

digit -> 1

.

digit -> 9 }

Η γραμματική σε συμβολισμό BNF:

<unsigned number> ::=<integer part><decimal fraction><exponent part>

<integer part> ::=<digit><integer part>|<digit>

<decimal fraction>::=.<integer part>|ε

<exponent part> ::=E<sign><integer part>|ε

<sign> ::=+ | - | ε

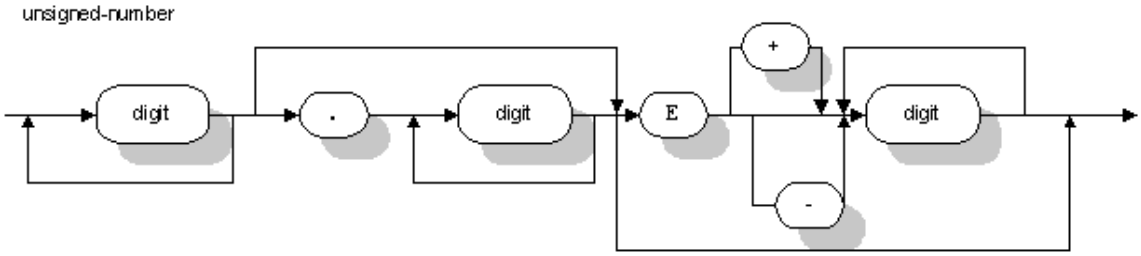
<digit> ::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Η γραμματική σε συμβολισμό EBNF:

unsigned number ::=digit^{+}['.'digit^{+}]['E'['+'|'-']digit^{+}]

digit::='0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'.

Η γραμματική σε συμβολισμό συντακτικών διαγραμμάτων:



Σχήμα 4.1 Συντακτικό διάγραμμα

4.3.2 Αυτόματα στοίβας

Τα αυτόματα στοίβας (ΑΣ) είναι αφηρημένες μηχανές που χρησιμοποιούνται ως αναγνωριστές για γλώσσες χωρίς συμφραζόμενα. Τα ΑΣ, όπως και τα πεπερασμένα αυτόματα, έχουν μια εσωτερική κατάσταση η οποία επηρεάζει τη λειτουργία τους. Επιπλέον όμως, κάθε ΑΣ διαθέτει και μια *στοίβα συμβόλων* και η λειτουργία του επηρεάζεται επιπρόσθετα από το σύμβολο που βρίσκεται στην κορυφή αυτής της στοίβας.

Ένα *αυτόματο στοίβας* (push-down automaton) είναι μια διατεταγμένη επτάδα της μορφής $M = (A, Q, H, \delta, q_o, h_o, F)$ όπου:

- A είναι το *αλφάβητο της συμβολοσειράς εισόδου.*
- Q είναι ένα μη κενό και πεπερασμένο *σύνολο καταστάσεων.*
- H είναι ένα δεύτερο αλφάβητο που ονομάζεται *αλφάβητο της στοίβας.*
- $\delta: Q \times H \times (A \cup \{\epsilon\}) \rightarrow P(H^* \times Q)$ είναι η *συνάρτηση μετάβασης.*
- $q_o \in Q$ είναι μια ειδική κατάσταση που ονομάζεται *αρχική κατάσταση.*

- $h_o \in H$ είναι ένα ειδικό σύμβολο του αλφάβητου της στοίβας που ονομάζεται *αρχικό σύμβολο της στοίβας* και
- $F \in Q$ είναι ένα υποσύνολο του συνόλου των καταστάσεων, τα στοιχεία του οποίου ονομάζονται *τελικές καταστάσεις*.

Τα αυτόματα στοίβας και κατ' επέκταση οι γλώσσες που αυτά χαρακτηρίζουν, κατατάσσονται σε διάφορες κλάσεις με ιδιαίτερα ενδιαφέροντα χαρακτηριστικά από πλευράς πολυπλοκότητας αναγνώρισης. Ένα ΑΣ ονομάζεται *πραγματικού χρόνου* (real-time) αν δεν έχει ϵ-μεταβάσεις. Ένα ΑΣ πραγματικού χρόνου ονομάζεται *απλό* αν έχει μόνο μια κατάσταση.

Ένα ΑΣ ονομάζεται *ντετερμινιστικό* (ΝΑΣ) αν η συνάρτηση μετάβασης δ πληρεί τις ακόλουθες προϋποθέσεις:

- Για κάθε q $\in Q$, $h \in H$ *και* $\chi \in A \cup \{\epsilon\}$ *το σύνολο* $\delta(q, h, \chi)$ έχει το πολύ ένα στοιχείο.
- Για κάθε $q \in Q$ και h $\in H$, αν $\delta(q, h, \epsilon) \neq 0$ τότε $\delta(q, h,$ a$) = 0$ για κάθε a $\in A$.

Διαφορετικά, το ΑΣ ονομάζεται *μη ντετερμινιστικό* (ΜΑΣ). Είναι φανερό ότι, για μια δεδομένη συμβολοσειρά εισόδου, ένα ΝΑΣ μπορεί να ακολουθήσει μια και μοναδική εκτέλεση, η οποία είτε αναγνωρίζει είτε απορρίπτει τη συμβολοσειρά.

Στους μεταγλωττιστές γενικά περιοριζόμαστε στη χρήση ντετερμινιστικών γλωσσών χωρίς συμφραζόμενα. Οι γλώσσες αυτές αναγνωρίζονται από ΝΑΣ τα οποία μπορούν να υλοποιηθούν άμεσα χωρίς οπισθοδρόμηση. Επιπλέον, είναι φανερό από τον ορισμό του ΝΑΣ, ότι ο αλγόριθμος προσομοίωσης εκτελείται σε χρόνο γραμμικό ως προς το μέγεθος της συμβολοσειράς εισόδου. Ειδικότερα, στους μεταγλωττιστές χρησιμοποιούνται συνήθως γραμματικές ειδικών κατηγοριών, όπως οι γραμματικές LL(1) και LR(l). Αυτές οι γραμματικές όχι μόνο αντιστοιχούν σε ΝΑΣ, αλλά υπάρχουν και (σχετικά) εύκολοι μηχανικοί τρόποι για την κατασκευή αυτών των ΝΑΣ. Οι γλώσσες που περιγράφουν είναι υποσύνολα των ντετερμινιστικών γλωσσών χωρίς συμφραζόμενα.

4.4 ΚΑΤΗΓΟΡΙΚΕΣ ΓΡΑΜΜΑΤΙΚΕΣ

Αντί να αντιμετωπίσουμε το πρόβλημα των γλωσσών με συμφραζόμενα και των

γραμμικά περιορισμένων μηχανών Turing σε όλο του το εύρος, στην κατασκευή μεταγλωττιστών περιοριζόμαστε σε μια συντηρητικότερη προσέγγιση. Επεκτείνουμε τις γραμματικές χωρίς συμφραζόμενα με την εισαγωγή *κατηγορημάτων* που περιγράφουν τα σύμβολα της γραμματικής. Κατ' αυτό τον τρόπο, προκύπτουν οι *κατηγορικές γραμματικές* οι οποίες επιπλέον της ικανότητάς τους να αναγνωρίζουν γλώσσες χωρίς συμφραζόμενα, μπορούν να χρησιμοποιηθούν και για την εκτέλεση υπολογισμών σχετικών με αυτές.

Μια *κατηγορική γραμματική* (attribute grammar) ορίζεται ως μια διατεταγμένη τριάδα K = *(G, V, F)* όπου:

- *G = (T, N, P, S)* είναι μια γραμματική χωρίς συμφραζόμενα.
- *V* είναι ένα πεπερασμένο σύνολο συμβόλων που ονομάζονται *κατηγορήματα* (attributes) και
- *F* είναι ένα σύνολο *περιορισμών* (assertions).

Κάθε κατηγόρημα υ ε *V* αντιστοιχεί σε ένα μοναδικό τερματικό ή μη τερματικό σύμβολο της γραμματικής $\chi \in T \cup N$. Τα κατηγορήματα μπορούν να θεωρηθούν ως στατικές τιμές που περιγράφουν τις εμφανίσεις των συμβόλων της γραμματικής σε ένα συντακτικό δέντρο. Το σύνολο των κατηγορημάτων που αντιστοιχούν σε ένα σύμβολο της γραμματικής είναι συγκεκριμένο, καθώς και ο τύπος των τιμών που μπορεί να πάρει το καθένα. Οι τιμές τους όμως είναι διαφορετικές για κάθε εμφάνιση ενός συμβόλου σε ένα συντακτικό δέντρο. Στην κατασκευή ενός μεταγλωττιστή χρησιμοποιείται κατ' αρχήν μια απλή γραμματική χωρίς συμφραζόμενα για την περιγραφή της σύνταξης της γλώσσας. Στη συνέχεια, η γραμματική αυτή επεκτείνεται σε μια κατάλληλη κατηγορική γραμματική που αποσκοπεί στο να υλοποιήσει ένα μεγάλο μέρος της συνολικής δουλειάς του μεταγλωττιστή.

ΚΕΦΑΛΑΙΟ 5

ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

5.1 ΛΕΚΤΙΚΕΣ ΜΟΝΑΔΕΣ

Στη φάση της *λεκτικής ανάλυσης* (lexical analysis), ο μεταγλωττιστής δέχεται ως είσοδο ένα αρχικό πρόγραμμα με τη μορφή μιας συμβολοσειράς χαρακτήρων και δίνει ως έξοδο το ίδιο πρόγραμμα με τη μορφή μιας συμβολοσειράς *λεκτικών μονάδων* (tokens). Οι λεκτικές αυτές μονάδες είναι τα τερματικά σύμβολα της γραμματικής, που περιγράφει τη σύνταξη της αρχικής γλώσσας προγραμματισμού, σε κάθε φάση της μεταγλώττισης, δηλαδή στη συντακτική ανάλυση. Έτσι, μπορεί κανείς να θεωρήσει ότι η λεκτική ανάλυση είναι κατ' ουσία μέρος της συντακτικής ανάλυσης. Ο λόγος που στην πράξη υλοποιείται ως ξεχωριστή φάση είναι γιατί αυτό διευκολύνει τη σχεδίαση και την υλοποίηση του μεταγλωττιστή.

Το τμήμα του μεταγλωττιστή που υλοποιεί τη φάση της λεκτικής ανάλυσης, λέγεται *λεκτικός αναλυτής* (lexical analyser, scanner). Εκτός από την εξαγωγή των λεκτικών μονάδων από το αρχικό πρόγραμμα, ο λεκτικός αναλυτής (ΛΑ) είναι δυνατόν να κάνει και άλλες λειτουργίες, ανάλογα με τη φύση της γλώσσας προγραμματισμού. Έτσι, μπορεί να χρησιμοποιεί τον πίνακα συμβόλων για να αποθηκεύει ειδικές κατηγορίες λεκτικών μονάδων, όπως τα ονόματα και οι σταθερές ή για να διαπιστώνει το είδος των λεκτικών μονάδων. Επίσης, σε περίπτωση διαπίστωσης λεκτικών σφαλμάτων μπορεί να καλεί το διαχειριστή σφαλμάτων για την εκτύπωση κατάλληλων διαγνωστικών μηνυμάτων.

Δεν υπάρχει κανόνας που να προσδιορίζει ποιες είναι οι λεκτικές μονάδες στη γενική περίπτωση[18]. Οι *λέξεις κλειδιά* (keywords), τα *ονόματα* (identifiers), οι *σταθερές* (constants), οι *τελεστές* (operators) και οι *διαχωριστές* (delimiters), θεωρούνται συνήθως λεκτικές μονάδες. Σε πολλές γλώσσες προγραμματισμού, όπως π.χ. στην Pascal ή στην ANSI C, οι λεκτικές μονάδες καθορίζονται στον ορισμό της γλώσσας.

5.2 ΑΝΑΓΝΩΡΙΣΗ ΛΕΚΤΙΚΩΝ ΜΟΝΑΔΩΝ

Οι λεκτικές μονάδες μιας γλώσσας προγραμματισμού κατατάσσονται σε κατηγορίες, κάθε μια από τις οποίες μπορεί να περιγραφεί από μια κανονική έκφραση και έτσι να αναγνωρισθεί από ένα πεπερασμένο αυτόματο. Επομένως, το πρόβλημα της κατασκευής του λεκτικού αναλυτή, δηλαδή αυτό της αναγνώρισης των λεκτικών μονάδων, φαίνεται κατ' αρχήν να συμπίπτει με το πρόβλημα της σχεδίασης τόσων πεπερασμένων αυτομάτων όσες είναι οι κατηγορίες των προς αναγνώριση λεκτικών μονάδων.

Όμως, το πρόβλημα είναι στην πραγματικότητα λίγο πιο πολύπλοκο. Οι συμβολοσειρές που απαρτίζουν τις λεκτικές μονάδες δεν είναι ξεχωριστές, αλλά αποτελούν τμήματα της συμβολοσειράς εισόδου. Έτσι, κατά τη διάρκεια της λειτουργίας του, ο ΛΑ καλείται να αναγνωρίσει μια λεκτική μονάδα που είναι πρόθεμα του εναπομείναντος τμήματος της συμβολοσειράς εισόδου, χωρίς όμως να είναι γνωστό το τέλος της. Το γεγονός ότι δεν είναι γνωστό το τέλος των λεκτικών μονάδων, δημιουργεί πρόβλημα για την αναγνώρισή τους και άρα για την κατασκευή του ΛΑ.

Το συνολικό πρόβλημα της κατασκευής ενός ΛΑ λύνεται με τη χρησιμοποίηση κατάλληλα τροποποιημένων ΝΠΑ, τα οποία:

- Διαβάζουν ενδεχομένως περισσότερους χαρακτήρες από όσους έχει μια λεκτική μονάδα.
- Οπισθοδρομούν αν αυτό χρειαστεί, και
- Διαθέτουν έξοδο, στην οποία προκύπτει κάθε φορά η λεκτική μονάδα που αναγνωρίζεται.

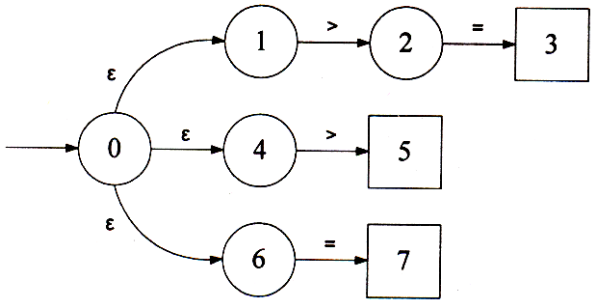
Για την παράσταση των τροποποιημένων ΝΠΑ, χρησιμοποιείται ένα διάγραμμα ειδικής μορφής, που ονομάζεται *διάγραμμα μετάβασης* (transition diagram). Τα διαγράμματα μετάβασης (ΔΜ), από τα οποία προκύπτει άμεσα ο γράφος μετάβασης (ΓΜ) των αντίστοιχων απλών ΝΠΑ, είναι πολύ χρήσιμα γιατί απεικονίζουν εύγλωττα τις λειτουργίες του ΛΑ.

Η κατασκευή του ΔΜ ενός λεκτικού αναλυτή γίνεται σχετικά εύκολα με εμπειρικό τρόπο και για το λόγο αυτό, δε θα περιγραφεί συστηματική μέθοδος. Παρακάτω περιγράφονται δυο τρόποι αναγνώρισης λεκτικών μονάδων. Ο πρώτος είναι γενικός και στηρίζεται σε τροποποιημένα ΜΠΑ με τη μορφή ΔΜ. Ο δεύτερος είναι ειδικός και στηρίζεται

στα ΔΜ.

5.2.1 Μια γενική μέθοδος

Για κάθε ξεχωριστή λεκτική μονάδα ή κατηγορία λεκτικών μονάδων καταγράφεται κατ' αρχήν η αντίστοιχη κανονική έκφραση. Στη συνέχεια, για κάθε τέτοια κανονική έκφραση κατασκευάζεται το ΜΠΑ-ϵ που την αναγνωρίζει.



Σχήμα 5.1. Αναγνώριση τελεστών "=", ">", ">="

Στο σχήμα 5.1 βλέπουμε το ΔΜ για την αναγνώριση των τελεστών "=", ">", ">=".

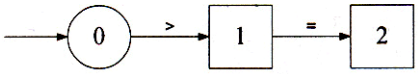
Στην πράξη οι ΛΑ δεν υλοποιούνται κατ' αυτό τον τρόπο. Το βασικό μειονέκτημα ενός ΛΑ που κατασκευάζεται με βάση τα παραπάνω είναι η ταχύτητά του. Με τη μέθοδο που περιγράφεται στην επόμενη ενότητα μπορούμε, χρησιμοποιώντας μια λιγότερο συστηματική και περισσότερο εμπειρική μέθοδο, να καταλήξουμε σε έναν πολύ ταχύτερο ΛΑ.

5.2.2 Μια ειδική μέθοδος με βάση τα διαγράμματα μετάβασης

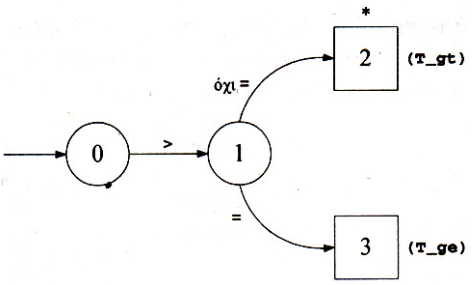
Η παρακάτω μέθοδος είναι ειδική και εφαρμόζεται εύκολα όταν τα ΠΑ που αντιστοιχούν στις ομάδες λεκτικών μονάδων μιας γλώσσας προγραμματισμού είναι ΝΠΑ και η ενοποίησή τους σε ένα αυτόματο μπορεί επίσης να γίνει με ΝΠΑ. Ακολουθεί μια σύντομη και άτυπη περιγραφή της μεθόδου αυτής.

Για κάθε ομάδα λεκτικών μονάδων μιας γλώσσας προγραμματισμού κατασκευάζουμε το αντίστοιχο ΝΠΑ και σχηματίζουμε το ΓΜ του. Μετά μετατρέπουμε κάθε ΓΜ σε ένα ΔΜ. Αυτή η μετατροπή γίνεται εύκολα, αρκεί να ληφθούν υπόψη οι διαφορές μεταξύ ΓΜ και ΔΜ. Στη συνέχεια, ενοποιούμε τα επιμέρους ΔΜ σε ένα γενικό ΔΜ το οποίο πρέπει

να είναι ντετερμινιστικό. Αυτό το γενικό ΔΜ περιγράφει εύ-γλωττα πως μπορούν να εξαχθούν οι λεκτικές μονάδες της συγκεκριμένης γλώσσας προγραμματισμού.



Σχήμα 5.2(α). Γράφος μετάβασης



Σχήμα 5.2(β). Διάγραμμα μετάβασης

Στο σχήμα 5.2(α) βλέπουμε το γράφο μετάβασης για την αναγνώριση των λεκτικών μονάδων "=", ">" και ">=" ενώ στο σχήμα 5.2(β) βλέπουμε το αντίστοιχο διάγραμμα μετάβασης.

5.3 ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ ΜΕ ΒΑΣΗ ΤΑ ΔΙΑΓΡΑΜΜΑΤΑ ΜΕΤΑΒΑΣΗΣ

Στο πρώτο βήμα της σχεδίασης του ΛΑ, γίνεται η καταγραφή των χαρακτήρων που αποτελούν το αλφάβητο της αρχικής γλώσσας. Για το σκοπό αυτό εξετάζονται οι λεκτικές μονάδες της αρχικής γλώσσας και καταγράφεται το σύνολο των χαρακτήρων που τις απαρτίζουν. Πολλές φορές χρησιμοποιείται ως αλφάβητο της αρχικής γλώσσας, ένα υπερσύνολο αυτού του συνόλου, όπως π.χ. το σύνολο των χαρακτήρων ASCII. Επίσης, είναι δυνατόν να προστεθεί στο αλφάβητο της αρχικής γλώσσας και ένας επιπλέον χαρακτήρας που αντιστοιχεί στο τέλος της συμβολοσειράς εισόδου. Ο χαρακτήρας αυτός συμβολίζεται συνήθως με EOF. Προκειμένου να μειωθεί το μέγεθος του αλφάβητου εισόδου Σ των αυτομάτων που αναγνωρίζουν τις λεκτικές μονάδες, συχνά οι χαρακτήρες ταξινομούνται σε ομάδες που έχουν την ίδια λειτουργική θέση.

Το δεύτερο βήμα στη σχεδίαση ενός ΛΑ, είναι η καταγραφή των λεκτικών μονάδων της γλώσσας προγραμματισμού για την οποία σχεδιάζεται ο ΛΑ. Στην καταγραφή αυτή συνηθίζεται να ταξινομούνται οι λεκτικές μονάδες σε ομάδες που παρουσιάζουν λειτουργικές ομοιότητες. Για παράδειγμα, τα ονόματα και οι αριθμητικές σταθερές αποτελούν τέ-

τοιες ομάδες λεκτικών μονάδων. Επίσης, συχνά παρουσιάζεται η ανάγκη να εξαιρεθούν κάποιες μορφές λεκτικών μονάδων από γενικότερες ομάδες. Παράδειγμα τέτοιας περίπτωσης στις περισσότερες γλώσσες προγραμματισμού είναι οι λέξεις κλειδιά, οι οποίες αποτελούν εξαίρεση στην ομάδα των ονομάτων.

Σε κάθε λεκτική μονάδα ή ομάδα λεκτικών μονάδων αντιστοιχεί ένας μοναδικός κωδικός, που συνήθως αποτελείται από τα εξής:

- Έναν κωδικό αριθμό που συνήθως είναι φυσικός αριθμός αλλά για να διευκολύνεται η ανάγνωσή του παριστάνεται με ένα κατάλληλο συμβολικό όνομα.
- Την ακολουθία των χαρακτήρων που αντιστοιχεί στη λεκτική μονάδα. Η συμβολοσειρά αυτή συνήθως ονομάζεται *lexeme* και χρησιμεύει για το διαχωρισμό λεκτικών μονάδων που ανήκουν σε μια ομάδα με κοινό κωδικό αριθμό.

Σε αυτό το βήμα της σχεδίασης του ΛΑ, εκτός από την ίδια την περιγραφή των λεκτικών μονάδων πρέπει επίσης να εξετασθούν τα ακόλουθα θέματα.

- Ο *τρόπος διαχωρισμού* των λεκτικών μονάδων. Συνήθως επιτρέπεται ο διαχωρισμός τους με κενούς χαρακτήρες.
- Τα *σχόλια* (comments), δηλαδή τμήματα του αρχικού προγράμματος τα οποία αγνοούνται.
- Η ύπαρξη διάκρισης *πεζών* και *κεφαλαίων γραμμάτων*.

Όπως είναι φανερό από τη φύση των ΔΜ, ο ΛΑ πρέπει να μπορεί να οπισθοδρομεί. Αυτό είναι αναγκαίο κυρίως στην περίπτωση που το τέλος μιας λεκτικής μονάδας δεν μπορεί να αναγνωριστεί παρά μόνο αν διαβαστούν χαρακτήρες που ακολουθούν.

Στο επόμενο βήμα, για κάθε ομάδα λεκτικών μονάδων κατασκευάζεται ένα ΝΠΑ, το οποίο στη συνέχεια μετατρέπεται σε ΔΜ. Ύστερα, όλα τα ΔΜ που αντιστοιχούν σε όλες τις λεκτικές μονάδες της αρχικής γλώσσας ενοποιούνται σε ένα γενικό ΔΜ, που περιγράφει τη λειτουργία του ΛΑ.

Τελικά ένας ΛΑ μπορεί να υλοποιηθεί ως *ρουτίνα* του συντακτικού αναλυτή. Μετά τη σχεδίαση του ενοποιημένου ΔΜ ανοίγονται δύο δρόμοι για την υλοποίηση του ΛΑ:

- Περιγράφουμε με κώδικα τις μεταβάσεις του ΔΜ για κάθε κατάσταση χωριστά αρχίζοντας από την αρχική κατάσταση. Ο τρόπος αυτός είναι περισσότερο εμπειρικός.
- Χρησιμοποιούμε έναν πίνακα δ που μας δίνει την επόμενη μετάβαση του ΛΑ. Ο πίνακας αυτός έχει μια γραμμή για κάθε μη τελική κατάσταση του ΔΜ και μια στήλη

για κάθε ομάδα χαρακτήρων εισόδου. Επιπλέον διατηρούμε τους χαρακτήρες της υπό αναγνώριση συμβολοσειράς σε μια ενδιάμεση μνήμη που λέγεται *lexeme.*

5.4 ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ ΜΕ ΤΟ ΜΕΤΑΕΡΓΑΛΕΙΟ FLEX

Το μεταεργαλείο flex είναι ένας γεννήτορας λεκτικών αναλυτών. Αποτελεί βελτίωση του μεταεργαλείου lex[19], που κατασκευάστηκε από την AT&T στη δεκαετία του 1970 ως ένα από τα συνοδευτικά προγράμματα του λειτουργικού συστήματος unix. Σήμερα το flex αναπτύσσεται από το GNU. Το GNU ξεκίνησε το 1984, με σκοπό την ανάπτυξη ενός ολοκληρωμένου λειτουργικού συστήματος ως ελεύθερο λογισμικό. Το επίσημο Manual του Flex μπορείτε να το βρείτε στη σελίδα της GNU στη διεύθυνση www.gnu.org/software/flex/manual/ και μπορείτε να το χρησιμοποιήσετε ελεύθερα με βάση τους κανόνες του GNU. Τα δυο εργαλεία lex και flex είναι συμβατά σε αρκετά μεγάλο βαθμό και είναι σήμερα διαθέσιμα σε εκδόσεις και για προσωπικούς υπολογιστές.

Το flex δέχεται ως είσοδο ένα μεταπρόγραμμα που περιγράφει τις προς αναγνώριση λεκτικές μονάδες, καθώς και τις ενέργειες που πρέπει να γίνουν όταν αυτές αναγνωρισθούν. Η έξοδός του είναι ένα πρόγραμμα γραμμένο σε C, το οποίο περιέχει τη συνάρτηση *yylex* που υλοποιεί το λεκτικό αναλυτή. Η συνάρτηση αυτή αναγνωρίζει την επόμενη λεκτική μονάδα και επιστρέφει έναν ακέραιο αριθμό, που αντιστοιχεί στον κωδικό της. Σε περίπτωση τέλους της συμβολοσειράς εισόδου, η yylex επιστρέφει την τιμή 0. Η ακολουθία των χαρακτήρων που απαρτίζουν την αναγνωρισθείσα λεκτική μονάδα είναι διαθέσιμη στη μεταβλητή *yytext* η οποία είναι δείκτης σε χαρακτήρα.

Το μεταπρόγραμμα που δίνεται ως είσοδος στο flex περιέχει τρία μέρη, τα οποία χωρίζονται με την ακολουθία χαρακτήρων %%. Η γενική μορφή ενός μεταπρογράμματος είναι:

Μέρος Α
%%
Μέρος Β
%%
Μέρος Γ

Και τα τρία μέρη μπορούν να είναι κενά αν και, όπως θα δούμε στη συνέχεια, ένα κενό μεταπρόγραμμα δεν είναι χρήσιμο. Ακολουθεί μια περιγραφή των περιεχομένων των τριών μερών:

- Στο μέρος Α περιλαμβάνονται τα εξής:
  - *Σχόλια,* τα οποία ακολουθούν τη σύμβαση της C /* This is a comment */
  - *Κώδικας* C, που περικλείεται από % { και %}. Ο κώδικας αυτός περιέχει δηλώσεις μακροεντολών, τύπων δεδομένων και μεταβλητών που χρησιμοποιούνται από το λεκτικό αναλυτή, π.χ.

```
% {
        #define MAX_LEXEME 256
        #define T_eof  0
        typedef struct {
                char lexeme [MAX_LEXEME];
                int lineNumber, charPosition;
        } tokenInfo;
        int currentLine = 1, currentChar = 1;
% }
```

  - *Μνημονικά ονόματα* που χρησιμοποιούνται στο μέρος Β ως συντομογραφίες για κανονικές εκφράσεις. Για παράδειγμα, οι παρακάτω γραμμές ορίζουν τα μνημονικά ονόματα letter και digit, που παριστάνουν αντίστοιχα γράμματα και ψηφία:

```
letter [A-Za-z]
digit [0-9]
```

  - Δηλώσεις *αρχικών καταστάσεων*, οι οποίες περιγράφονται παρακάτω.

- Το μέρος Β περιέχει κανόνες, κάθε ένας από τους οποίους περιγράφει μια ομάδα λεκτικών μονάδων και τις ενέργειες που θα γίνουν μετά την αναγνώρισή της. Η μορφή του μέρους αυτού είναι γενικά η εξής:

$$\text{κανονική έκφραση}_1 \ \text{ενέργεια}_1$$
$$\ldots$$
$$\text{κανονική έκφραση}_n \ \text{ενέργεια}_n$$

Η ακριβής μορφή των κανονικών εκφράσεων περιγράφεται στην επόμενη ενότητα

5.4.1 ενώ κάθε ενέργεια είναι μια εντολή της C, και συνηθέστερα μια ομάδα εντολών που περικλείεται από άγκιστρα. Συνήθως η ενέργεια αυτή επιστρέφει τον κωδικό της λεκτικής μονάδας που αναγνωρίσθηκε.

Η συνάρτηση *yylex* που παράγεται από το flex, συμπεριφέρεται κατά τον παρακάτω τρόπο: όταν καλείται, αρχίζει να διαβάζει χαρακτήρες από το αρχείο εισόδου έως ότου αναγνωρίσει το μακρύτερο πρόθεμα που περιγράφεται από μία από τις κανονικές εκφράσεις του μέρους Β. Τότε, προβαίνει στην ενέργεια που αντιστοιχεί σε αυτή την κανονική έκφραση. Αν το πρόθεμα αυτό περιγράφεται από περισσότερες της μιας κανονικές εκφράσεις, τότε χρησιμοποιείται η πρώτη.

- Στο μέρος Γ περιλαμβάνεται κώδικας C. Συνήθως σε αυτό το μέρος υλοποιούνται οι βοηθητικές συναρτήσεις που καλούνται από το μέρος Β και, αν αυτό είναι επιθυμητό, συμπεριλαμβάνεται η αρχική συνάρτηση main().

5.4.1 Περιγραφή κανονικών εκφράσεων στο flex

Στον πίνακα 5.1 δίνονται οι διάφορες μορφές κανονικών εκφράσεων του flex μέσω μιας σειράς παραδειγμάτων, και εξηγείται η σημασία τους. Οι παρακάτω χαρακτήρες έχουν ει-δική σημασία, είναι δηλαδή μετασύμβολα.

" \ [ ] { I ^ * ? . + | $ / % < >

Σε περίπτωση που κάποιος από αυτούς τους χαρακτήρες πρέπει να χρησιμοποιηθεί όχι ως μετασύμβολο, πρέπει να κλεισθεί ανάμεσα σε εισαγωγικά ή να προηγηθεί αυτού ο χαρακτήρας \ (backslash).

Πίνακας 5.1 Πίνακας κανονικών εκφράσεων του flex

Έκφραση	Περιγραφή.
a	Ο χαρακτήρας a.
.	Οποιοσδήποτε χαρακτήρας εκτός από το "τέλος γραμμής".
\χ	Αν ο χαρακτήρας είναι ένας από τους χαρακτήρες a, b, f, n, r, t, v, ή 0, τότε αυτή η ακολουθία ερμηνεύεται όπως η ANSI C. Διαφορετικά, ο ίδιος χαρακτήρας.
\123	Ο χαρακτήρας ASCII με οκταδική τιμή 123.

\x3f	Ο χαρακτήρας ASCII με δεκαεξαδική τιμή 3F.
"abc"	Ορίζεται μια απλή συμβολοσειρά, συγκεκριμένα η abc.
[abc]	Ένας από τους χαρακτήρες a-z.
[a-z]	Ένας οποιοσδήποτε από τους χαρακτήρες a-z.
[^a-z]	Ένας από τους χαρακτήρες ASCII εκτός από τους a-z.
{let}	Αντικαθίσταται από την κανονική έκφραση που συμβολίζει το let, και έχει δηλωθεί στο ΜΕΡΟΣ Α.
rs	Η παράθεση των κανονικών εκφράσεων r και s.
r|s	Η διάζευξη των κανονικών εκφράσεων r και s.
r/s	Η κανονική έκφραση r αλλά μόνο αν ακολουθεί η κανονική έκφραση s.
(r)	Η κανονική έκφραση r. Οι παρενθέσεις χρησιμοποιούνται για ομαδοποίηση.
r*	Η έκφραση r επαναλαμβάνεται από 0 έως άπειρες φορές.
r+	Η έκφραση r επαναλαμβάνεται από 1 έως άπειρες φορές.
r?	Η έκφραση r επαναλαμβάνεται 0 ή 1 φορά.
r(7)	Η έκφραση r επαναλαμβάνεται ακριβώς 7 φορές.
r{3, 5}	Η έκφραση r επαναλαμβάνεται από 3 έως 5 φορές.
r{3, }	Η έκφραση r επαναλαμβάνεται 4 φορές ή περισσότερες.
^r	Η έκφραση r πρέπει να βρίσκεται στην αρχή της γραμμής.
r$	Η έκφραση r πρέπει να βρίσκεται στην τέλος της γραμμής.
$<S_1, S_2>r$	Αρχική κατάσταση είναι μία από τις S_1, S_2.

5.4.2 Ένα παράδειγμα χρήσης του flex

Με τη βοήθεια του flex μπορούμε να υλοποιήσουμε εύκολα έναν απλό λεκτικό αναλυτή ο οποίος καταμετρά τον αριθμό των γραμμών και των λέξεων χαρακτήρων ενός αρχείου εισόδου.

```
#include <stdio.h>
    int num_lines = 0, num_chars = 0;
%%
“\n” {++num_lines; ++num_chars;}
“.” {++num_chars;}
```

```
%%
main(){
      yylex();
      printf( "# of lines = %d, # of chars = %d\n", num_lines, num_chars );
}
```

Στο μέρος Α του μεταπρογράμματος, κατ' αρχήν ορίζουμε με num_lines τον αριθμό των γραμμών και με num_chars τον αριθμό των χαρακτήρων. Στο μέρος Β ορίζουμε ότι αν αναγνωριστεί ο χαρακτήρας νέας γραμμής, να αυξηθεί και ο αριθμός των γραμμών ++num_lines; και ο αριθμός των χαρακτήρων ++num_chars; ενώ όταν αναγνωριστεί ο οποιοσδήποτε χαρακτήρας εκτός του χαρακτήρα αλλαγής γραμμής (**.**), να αυξηθεί ο αριθμός των χαρακτήρων ++num_chars;. Τέλος, στο μέρος Γ καλούμε την συνάρτηση yylex() και εμφανίζουμε τον αριθμό των γραμμών και χαρακτήρων που υπολογίσθηκε. Το αρχείο κειμένου που θα δημιουργήσουμε πρέπει να έχει επέκταση .l. Π.χ. calc.l. Στη συνέχεια από γραμμή εντολών, πρέπει να δώσουμε την εντολή >flex <όνομα_αρχείου.l> π.χ. >flex calc.l. Θα δημιουργηθεί το αρχείο lex.yy.c στο οποίο υπάρχει η συνάρτηση yylex(). Το αρχείο αυτό θα το χρησιμοποιήσουμε αργότερα με το bison για την παραγωγή του compiler.

5.4.3 Το περιβάλλον εκτέλεσης του flex

Το περιβάλλον του flex παρέχει έτοιμες τις παρακάτω συναρτήσεις οι οποίες είναι δυνατόν να επανορισθούν από τον προγραμματιστή.

- *int yylex ();* Η συνάρτηση αυτή υλοποιεί το ΛΑ. Η τιμή επιστροφής είναι 0 αν έχει βρεθεί το τέλος του αρχείου εισόδου, διαφορετικά ο κωδικός της λεκτικής μονάδας που αναγνωρίσθηκε.
- *void yymore ();* Η συνάρτηση αυτή επιτρέπει στο ΛΑ να κρατήσει την ήδη αναγνωρισμένη συμβολοσειρά και όποια αναγνωριστεί σαν επόμενη. Αυτό βοηθά όταν ο προγραμματιστής γνωρίσει ότι σε κάποιο σημείο, το αυτόματο έχει αναγνωρίσει λιγότερη συμβολοσειρά από όση έπρεπε, οπότε πρέπει η παρούσα συμβολοσειρά να μην σβηστεί από το εσωτερικό buffer yytext[].
- *void yyless (int η);* Η συνάρτηση αυτή οπισθοδρομεί *n* χαρακτήρες στη

συμβολοσειρά εισόδου.

- *int yywrap ();* Η συνάρτηση αυτή καλείται αυτόματα όταν βρεθεί το τέλος του αρχείου εισόδου. Αν η τιμή που θα επιστρέψει η *yywrap* είναι 1 τότε η λειτουργία του ΛΑ τερματίζεται, διαφορετικά συνεχίζει. Με αυτό τον τρόπο μπορεί να επιτευχθεί η διαδοχική επεξεργασία περισσότερων αρχείων εισόδου.

Οι μεταβλητές που αποτελούν το περιβάλλον του flex είναι οι εξής:

- *char * yytext;* Η μεταβλητή αυτή είναι ένας προσωρινός χώρος αποθήκευσης, στον οποίο αποθηκεύεται η συμβολοσειρά που αντιστοιχεί στην αναγνωρισθείσα λεκτική μονάδα. Είναι πολύ χρήσιμη στην περίπτωση λεκτικών μονάδων όπως τα ονόματα ή οι σταθερές.
- *int yyleng;* Περιέχει το μήκος της αναγνωρισθείσας λεκτικής μονάδας.
- *FILE * yyin;* Η μεταβλητή αυτή περιέχει ένα δείκτη στο αρχείο εισόδου.
- *YYSTYPE yylval;* Μέσω της μεταβλητής αυτής επικοινωνεί ο ΛΑ με τον συντακτικό αναλυτή. Ο τύπος YYSTYPE πρέπει να ορισθεί από τον προγραμματιστή στο μέρος Α (αρχικά είναι int). Κάθε φορά που ο ΛΑ αναγνωρίζει μια λεκτική μονάδα και την επιστρέφει στο συντακτικό αναλυτή, η τιμή *yylval* είναι η σημασιολογική τιμή που αντιστοιχεί σε αυτό το τερματικό σύμβολο και πρέπει να ενημερώνεται κατάλληλα στις ενέργειες του μέρους Β.

ΚΕΦΑΛΑΙΟ 6

ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

6.1 ΕΙΣΑΓΩΓΗ

Στη φάση της *συντακτικής ανάλυσης* (syntactic analysis, parsing), ελέγχεται αν το αρχικό πρόγραμμα ανήκει στη γλώσσα της οποίας η σύνταξη ορίζεται από μια δεδομένη γραμματική χωρίς συμφραζόμενα. Κατά τη διάρκεια αυτής της φάσης κατασκευάζεται το συντακτικό δέντρο που αντιστοιχεί στο αρχικό πρόγραμμα. Αν το δέντρο δεν είναι απαραίτητο για τη συνέχεια της μεταγλώττισης, κατασκευάζεται μόνο έμμεσα και δεν αποθηκεύεται. Το τμήμα του μεταγλωττιστή που κάνει αυτή τη δουλειά ονομάζεται συντακτικός αναλυτής (syntactic analyser, parser).

Η είσοδος ενός συντακτικού αναλυτή (ΣΑ) είναι το αρχικό πρόγραμμα με τη μορφή μιας ακολουθίας λεκτικών μονάδων. Η έξοδός του είναι το συντακτικό δέντρο που αντιστοιχεί σε αυτό το πρόγραμμα ή μια ένδειξη ότι το αρχικό πρόγραμμα δεν είναι συντακτικά ορθό. Στην πράξη όμως, ο ΣΑ συνεργάζεται στενά με τις επόμενες φάσεις της μεταγλώττισης και το συντακτικό δέντρο δεν είναι ορατό ως έξοδος αυτού του τμήματος. Οι κόμβοι του συντακτικού δέντρου περιέχουν τα σύμβολα μιας γραμματικής. Συγκεκριμένα, η ρίζα περιέχει το αρχικό σύμβολο, οι εσωτερικοί κόμβοι περιέχουν μη τερματικά σύμβολα, ενώ τα φύλλα περιέχουν τερματικά σύμβολα. Επιπλέον, οι απόγονοι κάθε εσωτερικού κόμβου ακολουθούν τους κανόνες παραγωγής της γραμματικής.

Υπάρχουν δυο βασικοί τρόποι κατασκευής ενός συντακτικού δέντρου δεδομένης της συμβολοσειράς εισόδου. Σύμφωνα με τον πρώτο τρόπο, το δέντρο κατασκευάζεται ξεκινώντας από τη ρίζα και προχωρώντας προς τα φύλλα. Οι ΣΑ που βασίζονται σε αυτό τον τρόπο ονομάζονται συντακτικοί αναλυτές *από πάνω προς τα κάτω* (top-down parsers). Σύμφωνα με το δεύτερο τρόπο, το δέντρο κατασκευάζεται ξεκινώντας από τα φύλλα και προχωρώντας προς τη ρίζα. Οι ΣΑ που βασίζονται στο δεύτερο τρόπο ονομάζονται συντακτικοί αναλυτές *από κάτω προς τα πάνω* (bottom-up parsers). Στους μεταγλωττιστές χρησιμοποιούνται συνήθως ΣΑ αυτών των δυο κατηγοριών που επιπλέον δεν

οπισθοδρομούν. Αυτοί οι ΣΑ ονομάζονται ντετερμινιστικοί.

6.1.1 Βοηθητικές έννοιες: FIRST και FOLLOW

Οι δυο έννοιες που παρουσιάζονται σε αυτή την ενότητα είναι κοινές και στις δύο κατηγορίες ΣΑ.

Έστω μια γραμματική $G = (T, N, P, S)$. Για κάθε συμβολοσειρά α Ε $(T \cup N)^*$ ορίζεται το σύνολο FIRST(a) $\subseteq T \cup \{\epsilon\}$ ως εξής:

- Αν υπάρχει κάποια δυνατή παραγωγή της μορφής α =>* αβ, όπου a $\in T$ και $\beta \in (T \cup N)^*$, τότε a ∈ FIRST(a).
- Αν υπάρχει μια δυνατή παραγωγή της μορφής α =>* ϵ, τότε ϵ ∈ FIRST(a).

Με άλλα λόγια, το σύνολο FIRST(a) περιέχει όλα τα τερματικά σύμβολα με τα οποία είναι δυνατόν να αρχίζουν οι συμβολοσειρές που παράγονται από την α. Αν η α παράγει την κενή συμβολοσειρά, τότε το σύνολο FIRST(a) περιέχει επίσης και το ϵ.

Για κάθε μη τερματικό σύμβολο $A \in N$ της ίδιας γραμματικής G, ορίζεται επίσης το σύνολο FOLLOW(A) ⊆ T ∪ {EOF} όπου με EOF συμβολίζεται η λεκτική μονάδα που δηλώνει το τέλος της συμβολοσειράς εισόδου:

- Αν υπάρχει κάποια δυνατή παραγωγή της μορφής S =>* αAaβ, όπου α, $\beta \in (N \cup T)^*$ και a $\in T$, τότε a ∈ FOLLOW(A).
- Αν υπάρχει κάποια δυνατή παραγωγή της μορφής S =>* $\alpha A a \beta$ όπου α $\in (N \cup T)^*$, τότε EOF ∈ FOLLOW(A).

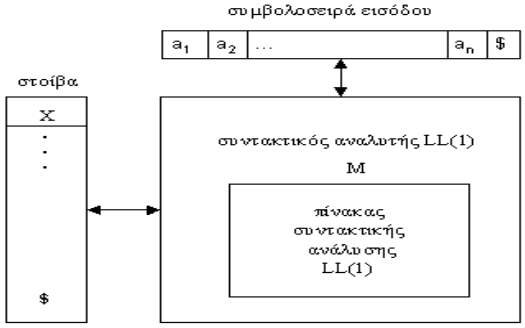
Με άλλα λόγια, το σύνολο FOLLOW(A) περιέχει όλα τα τερματικά σύμβολα που μπορούν να ακολουθούν το A σε έναν προτασιακό τύπο. Επίσης, αν το A μπορεί να είναι το τελευταίο σύμβολο σε έναν προτασιακό τύπο, τότε το σύνολο FOLLOW(A) περιέχει επίσης και το EOF. Ο υπολογισμός των συνόλων FOLLOW για όλα τα μη τερματικά σύμβολα μιας γραμματικής γίνεται συγχρόνως.

6.2 ΣΥΝΤΑΚΤΙΚΟΙ ΑΝΑΛΥΤΕΣ ΑΠΟ ΠΑΝΩ ΠΡΟΣ ΤΑ ΚΑΤΩ

Ένας ΣΑ από πάνω προς τα κάτω ξεκινά την κατασκευή του συντακτικού δέντρου από τη ρίζα, η οποία είναι γνωστό ότι περιέχει το αρχικό σύμβολο S της γραμματικής. Στη συνέχεια, ο ΣΑ επιλέγει τον κανόνα παραγωγής που θα χρησιμοποιηθεί για την αντικατάσταση του S, έστω π.χ. τον κανόνα S → α, τοποθετεί τα σύμβολα της α ως παιδιά του κόμβου S και επαναλαμβάνει για κάθε νέο κόμβο την ίδια διαδικασία. Η κατασκευή του δέντρου ολοκληρώνεται όταν όλα τα φύλλα είναι τερματικά σύμβολα της γραμματικής. Στη διαδικασία που περιγράφηκε πιο πάνω, ο ΣΑ καλείται να αποφασίσει για τα εξής:

- Ποιος κανόνας παραγωγής θα χρησιμοποιηθεί για την αντικατάσταση του συμβόλου $A \in N$ που περιέχεται στον τρέχοντα κόμβο;
- Αφού γίνει η επιλογή του κανόνα, με ποια σειρά θα γίνει η επεξεργασία των νέων κόμβων που προκύπτουν;

Όσον αφορά στη δεύτερη ερώτηση, οι περισσότεροι ΣΑ από πάνω προς τα κάτω διενεργούν την επεξεργασία των νέων κόμβων από αριστερά προς τα δεξιά, επιλέγοντας δηλαδή τη σειρά με την οποία αυτοί εμφανίζονται στο δεξιό μέλος του κανόνα παραγωγής. Η πρώτη ερώτηση είναι αρκετά πιο δύσκολη. Προκειμένου να αποφασίσει ποιον κανόνα παραγωγής θα χρησιμοποιήσει, ο ΣΑ είναι συνήθως αναγκασμένος να διαβάσει έναν αριθμό λεκτικών μονάδων από τη συμβολοσειρά εισόδου. Αν αρκούν *k* λεκτικές μονάδες προκειμένου να ληφθεί αυτή η απόφαση, τότε ο ΣΑ ονομάζεται LL(k). Τα δυο L σε αυτή την ονομασία υποδεικνύουν ότι η ανάγνωση της συμβολοσειράς εισόδου γίνεται από αριστερά προς τα δεξιά (left-to-right) και ότι η διαδικασία της κατασκευής του συντακτικού δέντρου αντιστοιχεί στην αριστερότερη παραγωγή (leftmost derivation).



Σχήμα 6.1 Συντακτικός αναλυτής LL(1)

Ο αριθμός *k* αντιστοιχεί όπως προαναφέρθηκε, στον αριθμό των συμβόλων που πρέπει να διαβαστούν.

Παράδειγμα ΣΑ από πάνω προς τα κάτω είναι οι *συντακτικοί αναλυτές LL(1)* και συνήθως υλοποιούνται με αυτόματο τρόπο, χρησιμοποιώντας κατάλληλα μεταεργαλεία. Η μορφή του αυτόματου στοίβας που υλοποιεί ένα ΣΑ LL(1) φαίνεται στο σχήμα 6.1. Ο ΣΑ αποτελείται από:

- Μια ενδιάμεση μνήμη απ' όπου διαβάζεται η συμβολοσειρά εισόδου, μια λεκτική μονάδα κάθε φορά.
- Μια στοίβα όπου τοποθετούνται τερματικά και μη τερματικά σύμβολα της γραμματικής.
- Ένα δισδιάστατο πίνακα ελέγχου που ονομάζεται *πίνακας συντακτικής ανάλυσης LL(1)*. Ο πίνακας αυτός περιέχει μια γραμμή για κάθε μη τερματικό σύμβολο *A* της γραμματικής και μια στήλη για κάθε τερματικό σύμβολο a. Επιπλέον, περιέχει μια ακόμα στήλη για το ειδικό σύμβολο EOF, που αντιστοιχεί στο τέλος της συμβολοσειράς εισόδου. Κάθε στοιχείο *M(A,* a) μπορεί να περιέχει έναν κανόνα παραγωγής της μορφής $A \rightarrow \alpha$, ή να είναι κενό.

6.2.1 Γραμματικές LL(1)

Μια γραμματική χωρίς συμφραζόμενα ονομάζεται LL(1), όταν μπορεί να αναγνωρισθεί από έναν ΣΑ LL(1). Μπορεί να αποδειχθεί ότι για να συμβαίνει αυτό πρέπει να πληρούνται οι ακόλουθες προϋποθέσεις:

- Αν $A \rightarrow \alpha$ και $A \rightarrow \beta$ είναι δυο διαφορετικοί κανόνες της γραμματικής, τότε πρέπει FIRST(α) ∩ FIRST(β) = 0. Δηλαδή τα σύνολα FIRST για όλους τους εναλλακτικούς κανόνες ενός μη τερματικού συμβόλου πρέπει να είναι ξένα μεταξύ τους.
- Αν $A \rightarrow \varepsilon$ είναι κανόνας της γραμματικής, τότε πρέπει FIRST(A) ∩ FOLLOW(A) = 0. Δηλαδή αν υπάρχει μηδενική παραγωγή για ένα μη τερματικό σύμβολο, τότε τα σύνολα FIRST και FOLLOW αυτού του συμβόλου πρέπει να είναι ξένα μεταξύ τους.

Σε αρκετές όμως περιπτώσεις, μια γραμματική που δεν είναι LL(1) μπορεί να μετατραπεί σε μια άλλη ισοδύναμη γραμματική που να είναι LL(1). Τα τρία είδη μετασχηματισμών που περιγράφονται στις επόμενες ενότητες είναι ιδιαίτερα χρήσιμα για αυτό το σκο-

πό.

6.2.1.1 Αντικατάσταση

Με το μετασχηματισμό αυτό αντικαθιστούμε ένα μη τερματικό σύμβολο *A,* που βρίσκεται στο δεξιό μέλος ενός κανόνα παραγωγής για το μη τερματικό σύμβολο *B,* με όλα τα εναλλακτικά δεξιά μέλη κανόνων για το *A.* Αν υπάρχουν *n* τέτοια εναλλακτικά μέλη, ο κανόνας για το *B* επαναλαμβάνεται *n* φορές και σε κάθε έναν αντικαθιστούμε μια από τις εναλλακτικές περιπτώσεις για το *A.*

6.2.1.2 Αριστερή παραγοντοποίηση

Έστω μια γραμματική στην οποία δύο ή περισσότεροι εναλλακτικοί κανόνες παραγωγής αρχίζουν με το ίδιο πρόθεμα. Οι κανόνες αυτοί μπορούν να μετασχηματιστούν αφαιρώντας τον "κοινό παράγοντα" και χρησιμοποιώντας ένα νέο μη τερματικό σύμβολο. Αν το κοινό πρόθεμα α υπάρχει μεν αλλά δεν είναι εμφανές στους κανόνες παραγωγής του *A,* μπορεί συνήθως να χρησιμοποιηθεί η τεχνική της αντικατάστασης έτσι ώστε να εμφανισθεί η περίπτωση που μόλις εξετάστηκε.

6.2.1.3 Απαλοιφή αριστερής αναδρομής

Αριστερή αναδρομή εμφανίζεται γενικά όταν υπάρχει ένας κανόνας της μορφής $A \rightarrow \alpha$, όπου είναι δυνατή η παραγωγή $\alpha =>^{*} A\beta$. Οι δυο συνηθέστερες μορφές αριστερής αναδρομής είναι οι εξής:

- Η *άμεση,* δηλαδή η ύπαρξη ενός κανόνα της μορφής $A \rightarrow A\alpha$.
- Η *έμμεση,* δηλαδή η ύπαρξη μιας σειράς κανόνων της μορφής $A_1 \rightarrow A_2\alpha_1$, $A_2 \rightarrow A_3\alpha_2$, ..., $A_n \rightarrow A_1\alpha_n$.

6.3 ΣΥΝΤΑΚΤΙΚΟΙ ΑΝΑΛΥΤΕΣ ΑΠΟ ΚΑΤΩ ΠΡΟΣ ΤΑ ΠΑΝΩ

Ένας ΣΑ από κάτω προς τα πάνω, ξεκινά την κατασκευή του συντακτικού δέντρου από τα φύλλα, δηλαδή τα τερματικά σύμβολα που περιέχονται στη συμβολοσειρά εισόδου. Στη συνέχεια, προσπαθεί να βρει τον αριστερότερο κόμβο του συντακτικού δέντρου, που δεν έχει ακόμα κατασκευαστεί, ενώ όλα τα παιδιά του έχουν κατασκευαστεί. Η διαδικασία αυτή συνεχίζεται μέχρι να κατασκευαστεί η ρίζα του δέντρου, που περιέχει το αρχικό σύμβολο *S* της γραμματικής. Επομένως, κάθε στιγμή ο ΣΑ καλείται να επιλέξει ποιους από τους ήδη κατασκευασμένους κόμβους θα χρησιμοποιήσει ως παιδιά του νέου κόμβου

που θα κατασκευάσει. Η διαδικασία αυτή ονομάζεται *ελάττωση* (reducing).

Μια γενική κατηγορία συντακτικών αναλυτών από κάτω προς τα πάνω, είναι η κατηγορία των ΣΑ *ολίσθησης - ελάττωσης* (shift-reduce parsers). Οι αναλυτές αυτοί χρησιμοποιούν μια στοίβα στην οποία τοποθετούν τερματικά και μη τερματικά σύμβολα της γραμματικής. Η στοίβα είναι αρχικά κενή. Κατά τη λειτουργία τους, εκτελούν δυο ειδών πράξεις:

- Η πράξη της *ολίσθησης* (shift), αφαιρεί ένα σύμβολο από την αρχή της συμβολοσειράς εισόδου και το τοποθετεί στην κορυφή της στοίβας.
- Η πράξη της *ελάττωσης* (reduce), δεν επηρεάζει τη συμβολοσειρά εισόδου, αλλά μπορεί να εφαρμοστεί μόνο όταν στην κορυφή της στοίβας έχει εμφανισθεί το δεξιό μέλος κάποιου συντακτικού κανόνα. Τότε, τα σύμβολα που αποτελούν το δεξιό μέλος του κανόνα αφαιρούνται από τη στοίβα και στην κορυφή αυτής προστίθεται το αριστερό μέλος του κανόνα.

Η διαδικασία αυτή τερματίζεται όταν η στοίβα περιέχει μόνο το αρχικό μη τερματικό σύμβολο της γραμματικής και έχει εξαντληθεί η συμβολοσειρά εισόδου. Στην περίπτωση αυτή, η συμβολοσειρά εισόδου αναγνωρίζεται. Διαφορετικά, αν η διαδικασία καταλήξει σε αδιέξοδο, η συμβολοσειρά εισόδου απορρίπτεται.

Για να μη χρειάζεται ένας ΣΑ ολίσθησης-ελάττωσης να οπισθοδρομεί, κάτι που είναι ιδιαίτερα δαπανηρό τόσο σε ταχύτητα εκτέλεσης όσο και σε κατανάλωση μνήμης, πρέπει να μπορεί να επιλύει σωστά τις ενδεχόμενες συγκρούσεις. Ένας τέτοιος ΣΑ ονομάζεται ντετερμινιστικός. Οι ΣΑ που ανήκουν σε αυτή την κατηγορία, διαβάζουν τη συμβολοσειρά εισόδου από αριστερά προς τα δεξιά (left-to-right), κατασκευάζουν το συντακτικό δέντρο ακολουθώντας τη δεξιότερη παραγωγή (ή rightmost derivation) σε αντίστροφη σειρά και αν ο αριθμός των προπορευόμενων συμβόλων (look-ahead symbols) που διαβάζουν είναι ίσος με *k* ονομάζονται ΣΑ *LR(k)*. Στην πράξη, χρησιμοποιούνται πιο συχνά οι ΣΑ *LR(1)*, καθώς είναι ιδιαίτερα δύσκολο να υλοποιηθούν ΣΑ *LR(k)* με $k > 1$. Οι γραμματικές που παράγουν γλώσσες οι οποίες αναγνωρίζονται από κάποιο ΣΑ *LR(1)* ονομάζονται γραμματικές *LR(1)*.

6.3.1 Γραμματικές LR(1)

Μια γραμματική ονομάζεται *LR(1)* όταν μπορεί να αναγνωρισθεί από έναν ΣΑ LR(1). Μπορεί να αποδειχθεί ότι για να συμβαίνει αυτό πρέπει να πληρούνται οι ακόλουθες προϋποθέσεις:

- Start =>*aAw=>*abw
- Start=>*gBx=>*aby
- FIRST(w)=FIRST(y)

6.4 ΣΥΝΤΑΚΤΙΚΟΙ ΑΝΑΛΥΤΕΣ LALR(L) ΜΕ ΤΟ ΜΕΤΑΕΡΓΑΛΕΙΟ BISON

Το μεταεργαλείο bison[20] είναι ένας γεννήτορας συντακτικών αναλυτών για γλώσσες και γραμματικές τύπου LALR(1). Αποτελεί βελτίωση του μεταεργαλείου yacc που, όπως και το flex, κατασκευάστηκε από την AT&T στη δεκαετία του 1970 ως ένα από τα συνοδευτικά προγράμματα του λειτουργικού συστήματος unix. Σήμερα το bison όπως και το flex αναπτύσσεται από το GNU. Το επίσημο Manual του bison μπορείτε να το βρείτε στη σελίδα της GNU στη διεύθυνση www.gnu.org/software/bison/manual/ και να το χρησιμοποιήσετε ελεύθερα με βάση τους κανόνες του GNU. Το bison είναι συμβατό σε αρκετά μεγάλο βαθμό με το yacc. Και τα δύο αυτά εργαλεία είναι σήμερα διαθέσιμα σε εκδόσεις και για προσωπικούς υπολογιστές.

Το bison δέχεται ως είσοδο ένα μεταπρόγραμμα που περιγράφει τη σύνταξη της αρχικής γλώσσας προγραμματισμού, καθώς και τις ενέργειες που πρέπει να γίνονται κατά την αναγνώριση των συμβολοσειρών εισόδου. Η έξοδός του είναι ένα πρόγραμμα γραμμένο σε C, το οποίο περιέχει τη συνάρτηση *yyparse* που υλοποιεί το συντακτικό αναλυτή. Η συνάρτηση αυτή επιχειρεί να αναγνωρίσει τη συμβολοσειρά εισόδου και, παράλληλα με την κατασκευή του συντακτικού δέντρου, εκτελεί τις ενέργειες που περιγράφονται στο μεταπρόγραμμα.

Το μεταπρόγραμμα που δίνεται ως είσοδος στο bison, περιέχει τρία μέρη τα οποία χωρίζονται με την ακολουθία χαρακτήρων %%. Η γενική μορφή ενός μεταπρογράμματος είναι:

Μέρος Α

%%

Μέρος Β

%%

Μέρος Γ

Η λειτουργία των τριών μερών μοιάζει πάρα πολύ με την αντίστοιχη για το μεταεργαλείο flex. Και τα τρία μέρη μπορούν να είναι κενά αν και, όπως θα δούμε στη συνέχεια και σε αυτή την περίπτωση ένα κενό μεταπρόγραμμα δεν είναι χρήσιμο. Ακολουθεί μια περιγραφή των περιεχομένων των τριών μερών:

- Στο μέρος Α περιλαμβάνονται τα εξής:
  - *Σχόλια,* τα οποία ακολουθούν τη σύμβαση της C π.χ *1** This is a comrnent **1*
  - *Κώδικας* C, που περικλείεται από %{ και %}. Ο κώδικας αυτός συνήθως περιέχει δηλώσεις μακροεντολών, τύπων δεδομένων και μεταβλητών που χρησιμοποιούνται από το συντακτικό αναλυτή.
  - *Δηλώσεις των λεκτικών μονάδων,* αν για αυτές χρησιμοποιούνται συμβολικά ονόματα, π.χ.

    %token T_id T_int_const T_float_const

    %token T_if T_then T_else
  - *Δηλώσεις των τελεστών της γλώσσας,* και συγκεκριμένα της *προτεραιότητας* (precedence) και της *προσεταιριστικότητας* (associativity) αυτών. Η σειρά προτεραιότητας δηλώνεται κατά αύξουσα σειρά, ενώ η προσεταιριστικότητα με τις εντολές %left και %right π.χ.

    %left '+' '-'

    %left '*' '/' T_div T_mod
  - *Δήλωση του συνόλου των σημασιολογικών τιμών.* Ο τύπος που χρησιμοποιείται για την παράσταση των σημασιολογικών τιμών έχει το όνομα *YYSTYPE* και είναι συνήθως μια ένωση (*union*). Η δήλωση του γίνεται όπως στο παρακάτω παράδειγμα:

    %union{

    int i;

    double f;

}

όπου ορίζεται ότι οι σημασιολογικές τιμές μπορούν να είναι ακέραιες ή πραγματικές.

- *Δήλωση του τύπου της σημασιολογικής τιμής κάθε συμβόλου.* Με αυτές τις δηλώσεις καθορίζεται ποιο πεδίο του τύπου ένωσης *YYSTYPE* θα χρησιμοποιούν τα τερματικά αλλά και τα μη τερματικά σύμβολα της γραμματικής για τις σημασιολογικές τιμές τους, π.χ.

```
%token<str> T_id
%token<i> T_int_const
%token<f> T_float_const
%type<f> expression
```

όπου ορίζεται αφενός ότι τα τερματικά σύμβολα T_id, T_int_const και T_float_const θα χρησιμοποιούν τα πεδία str, i και f, αντίστοιχα, για τις σημασιολογικές τους τιμές, αφετέρου ότι το μη τερματικό σύμβολο expression θα χρησιμοποιεί το πεδίο f.

- Στο μέρος Β ορίζονται οι κανόνες παραγωγής της γραμματικής, διατυπωμένοι σε μορφή BNF. Επίσης, ορίζονται οι σημασιολογικές ρουτίνες που εκτελούνται κατά την κατασκευή του συντακτικού δέντρου, όταν χρησιμοποιούνται οι κανόνες παραγωγής.
- Στο μέρος Γ περιλαμβάνεται κώδικας C. Συνήθως σε αυτό το μέρος υλοποιούνται οι συναρτήσεις που καλούνται από το μέρος Β και αν αυτό είναι επιθυμητό, συμπεριλαμβάνεται η αρχική συνάρτηση main.

Το μεταπρόγραμμα που δίνεται ως είσοδος στο bison αναλαμβάνει συνήθως τον κεντρικό έλεγχο του μεταγλωττιστή, που επιτυγχάνεται με τη συνεργασία των παρακάτω:

- του λεκτικού αναλυτή, που πολλές φορές κατασκευάζεται αυτόματα με χρήση του μεταεργαλείου flex,
- του συντακτικού αναλυτή, που περιγράφεται στους κανόνες παραγωγής του μέρους Β,
- του πίνακα συμβόλων, η ενημέρωση του οποίου υλοποιείται με κατάλληλες κλήσεις συναρτήσεων στις σημασιολογικές ρουτίνες των κανόνων παραγωγής στο μέρος Β,
- του σημασιολογικού αναλυτή, που υλοποιείται στις σημασιολογικές ρουτίνες των

κανόνων παραγωγής, στο μέρος B, και

- του γεννήτορα ενδιάμεσου κώδικα, που και αυτός υλοποιείται στις σημασιολογικές ρουτίνες των κανόνων παραγωγής.

6.4.1 Περιγραφή κανόνων παραγωγής στο bison

Η γραμματική LALR(l) περιγράφεται στο μέρος B του μεταπρογράμματος του bison με μια ακολουθία κανόνων παραγωγής της μορφής:

$$A : \begin{matrix} \chi_1^1 \chi_2^1 \cdots \chi_{m1}^1 \\ \chi_1^2 \chi_2^2 \cdots \chi_{m2}^2 \\ \cdots \\ \chi_1^n \chi_2^n \cdots \chi_{mn}^n \end{matrix}$$

όπου $n \geq 0$, και $m_i \geq 0$ για κάθε i τέτοιο ώστε $1 \leq i \leq n$. Το σύμβολο *A* στο αριστερό μέλος του κανόνα, είναι ένα μη τερματικό σύμβολο της γραμματικής. Τα σύμβολα χ_j^i στο δεξί μέλος του κανόνα είναι σύμβολα της γραμματικής (τερματικά ή μη τερματικά) ή σημασιολογικές ρουτίνες. Κάθε σημασιολογική ρουτίνα αποτελείται από εντολές C που περικλείονται από άγκιστρα. Στο δεξιό μέλος κανόνων επιτρέπονται επίσης σχόλια, που ακολουθούν τη σύμβαση της C. Αρχικό σύμβολο της γραμματικής θεωρείται αυτό που βρίσκεται στο αριστερό μέλος του πρώτου κανόνα παραγωγής.

Στη συνέχεια, γράφουμε μερικά παραδείγματα κανόνων παραγωγής στο μέρος B

```
expression : term                    { $$ = $1;       }
           | expression '+' term     { $$ = $1 + $3; }
```

Στους κανόνες αυτούς υπάρχουν σημασιολογικές ρουτίνες που εκτελούν τις πράξεις και υπολογίζουν τις σημασιολογικές τιμές των συμβόλων. Με *$$* συμβολίζεται η σημασιολογική τιμή του αριστερού μέλους ενός κανόνα, ενώ με *$n*, όπου $n > 0$, συμβολίζεται η σημασιολογική τιμή του n-οστού συμβόλου στο δεξιό μέλος. Για παράδειγμα, η εντολή $$ = $1 + $3; στο δεύτερο κανόνα παραγωγής για το σύμβολο expression ορίζει ότι η σημασιολογική τιμή της εμφάνισης αυτού του συμβόλου στο αριστερό μέλος του κανόνα είναι ίση με το άθροισμα των σημασιολογικών τιμών του πρώτου και του τρίτου συμβόλου στο δεξιό μέλος, δηλαδή των εμφανίσεων των συμβόλων expression και terrn. Με τον τρόπο αυτό είναι δυνατή στο μεταεργαλείο bison η περιγραφή κατηγορικών γραμματικών.

6.4.2 Χειρισμός συντακτικών σφαλμάτων στο bison

Συντακτικό σφάλμα παράγεται στο bison όταν ο λεκτικός αναλυτής επιστρέψει μια λεκτική μονάδα που δεν ανήκει στο σύνολο των αναμενόμενων σε αυτή την κατάσταση του αυτομάτου ή όταν κληθεί η συνάρτηση *yyerror()* μέσα σε μια σημασιολογική ρουτίνα. Ο ΣΑ που κατασκευάζεται από το bison μπορεί επίσης να οπισθοδρομεί όταν εντοπίσει συντακτικό σφάλμα, ώσπου να φτάσει σε μια προηγούμενη κατάσταση όπου το σφάλμα να προβλέπεται. Αυτό γίνεται με τη χρήση του ειδικού συμβόλου *error* στους κανόνες παραγωγής, η οποία επιτρέπει την καθοδήγηση του ΣΑ για την αποτελεσματική ανάνηψη από σφάλματα.

6.5 ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΕΚΤΕΛΕΣΗΣ ΤΟΥ BISON

Τα μεταεργαλεία bison και flex (όπως αντίστοιχα τα yacc και lex) έχουν κατακευαστεί για να συνεργάζονται στενά. Ως εκ τούτου, είναι φυσικό να μοιράζονται ένα μέρος από το περιβάλλον εκτέλεσής τους.

Οι συναρτήσεις που αποτελούν το περιβάλλον εκτέλεσης του bison είναι οι εξής:

- *int yyparse ();* Η συνάρτηση αυτή υλοποιεί το ΣΑ. Επιστρέφει την τιμή 1 αν αναγνωρισθεί η συμβολοσειρά εισόδου, ή την τιμή 0 σε περίπτωση συντακτικού σφάλματος.
- *int yylex ();* Αυτή είναι η συνάρτηση που υλοποιεί τον λεκτικό αναλυτή. Καλείται από την yyparse κάθε φορά που πρέπει να διαβαστεί μια νέα λεκτική μονάδα από τη συμβολοσειρά εισόδου. Η τιμή επιστροφής της είναι 0 αν έχει βρεθεί το τέλος του αρχείου εισόδου, διαφορετικά ο κωδικός της λεκτικής μονάδας που αναγνωρίσθηκε.
- *void yyerror (const char * message);* Η συνάρτηση καλείται αυτόματα όταν εντοπιστεί κάποιο συντακτικό σφάλμα για να εκτυπώσει κατάλληλο διαγνωστικό μήνυμα ή όταν στον κώδικα του χρήστη παρεμβληθεί η εντολή YYERROR.

Οι μεταβλητές που αποτελούν το περιβάλλον του bison είναι οι ακόλουθες:

- *YYSTYPE yylval;* Μέσω της μεταβλητής αυτής επικοινωνεί ο ΣΑ με το λεκτικό αναλυτή. Ο τύπος YYSTYPE πρέπει να ορισθεί από τον προγραμματιστή στο μέρος Α (αρχικά είναι int). Η μεταβλητή yylval περιέχει τη σημασιολογική τιμή της λεκτικής

μονάδας που επιστρέφεται από το λεκτικό αναλυτή.

- *int yydebug;* Όταν αυτή η μεταβλητή έχει μη μηδενική τιμή, ο ΣΑ εκτυπώνει πληροφορίες σχετικά με την πρόοδο της συντακτικής ανάλυσης, π.χ. ποια λεκτική μονάδα διάβασε και ποιον κανόνα παραγωγής εφαρμόζει. Έτσι είναι δυνατή η παρακολούθηση του ΣΑ και διευκολύνεται η εύρεση σφαλμάτων στη γραμματική.

Ένα αρχείο εισόδου στο bison πρέπει να έχει επέκταση .y. Από την γραμμή εντολών δίνουμε την εντολή >bison –d <όνομα_αρχείου.y>. Θα δημιουργηθεί ένα αρχείο όνομα_αρχείου.tab.c. Το αρχείο αυτό με τη βοήθεια του αρχείου που δημιουργήθηκε από το flex θα δημιουργήσουν τον compiler. Π.χ. έστω ότι έχουμε ένα αρχείο το calc.l. Θα πρέπει να δώσουμε τις εξής εντολές:

>bison –d calc.y

>flex calc.l

Τα ερχεία lex.yy.c και clac.tab.c περιέχουν κώδικα της γλώσσας C και πρέπει να μεταγλωττιστούν με βάση τον τρόπο που η συγκεκριμένη γλώσσα ορίζει:

>gcc lex.yy.c clac.tab.c –o parser –lfl

Με αυτόν τον τρόπο μπορούμε να παράγουμε τον μεταγλωττιστή και με την εντολή >parser όνομα_αρχείου μπορούμε να μεταγλωττίσουμε το αρχείο.

Τα flex και bison, αν χρησιμοποιείτε το λειτουργικό σύστημα windows μπορείτε να τα εγκαταστήσετε με βάση τις οδηγίες που θα βρείτε στο GNU. Αν χρησιμοποιείται το λειτουργικό σύστημα UNIX ή λειτουργικά προερχόμενα από το UNIX όπως το LINUX τα flex και bison είναι ενσωματωμένα και μπορείτε να τα επιλέξετε για εγκατάσταση κατά την εγκατάσταση του λειτουργικού.

ΚΕΦΑΛΑΙΟ 7

Η ΕΦΑΡΜΟΓΗ

7.1 Η ΓΛΩΣΣΑ ΕΦΑΡΜΟΓΗΣ

Η γλώσσα που υλοποιήθηκε, είναι μια απλή γλώσσα με την οποία μπορούμε να κάνουμε αναθέσεις τιμών σε μεταβλητές, καθώς και μαθηματικές πράξεις. Μπορούμε να πούμε ότι λειτουργεί σαν αριθμομηχανή. Μπορούμε να ορίσουμε μεταβλητές. Οι μεταβλητές μπορούν να είναι λέξεις οι οποίες αποτελούνται από γράμματα πεζά ή κεφαλαία. Στις μεταβλητές μπορούμε να αναθέσουμε ακέραιες ή πραγματικές τιμές, θετικές ή αρνητικές. Οι πράξεις που μπορούμε να κάνουμε είναι οι τέσσερις βασικές πράξεις δηλαδή η πρόσθεση, η αφαίρεση, ο πολλαπλασιασμός και η διαίρεση.

Για παράδειγμα η παράσταση 1.1+3.5 δίνει σαν αποτέλεσμα 4.6 ενώ οι παραστάσεις:
x = 3.1
y = 4.6
x+y δίνουν σαν αποτέλεσμα 7.7.

7.2 ΤΟ ΛΕΞΙΚΟ ΚΑΙ ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ FLEX

Το αρχείο εισόδου στο flex έχει δομή, την τυπική δομή όλων των αρχείων εισόδου στο flex. Αρχικά στο χώρο των ορισμών συμπεριλαμβάνονται μακροεντολές, βιβλιοθήκες και αρχεία που είναι απαραίτητα για την μεταγλώττιση. Στη συνέχεια ορίζουμε τις κανονικές εκφράσεις NUM και WORD. Στο χώρο των κανόνων ορίζονται κανόνες που αντιστοιχούν σε λεκτικές μονάδες και τις ενέργειες που θα πρέπει να γίνουν όταν οι μονάδες αυτές αναγνωριστούν.

Οι λεκτικές μονάδες της γλώσσας χωρίζονται στις παρακάτω κατηγορίες:

- Ονόματα μεταβλητών τα οποία μπορούν να αποτελούνται από ένα ή περισσότερα γράμματα πεζά ή κεφαλαία.
- Χαρακτήρες διαφυγής: \t, \n.

- Τελεστές +, -, *, /, =.
- Τους διαχωριστές (,), [,].
- Κενοί χαρακτήρες ή χαρακτήρες στηλοθέτησης.

Η γλώσσα υποστηρίζει τους τύπους:

- char: χαρακτήρες.
- double: πραγματικοί διπλής ακρίβειας.

Στο τρίτο μέρος υπάρχουν συναρτήσεις όπως οι putsym και getsym, που χρησιμοποιούνται στο χώρο των κανόνων. Η συνάρτηση main δεν υλοποιείται σε αυτό το αρχείο, αλλά στο αρχείο εισόδου στο bison.

7.3 Η ΓΡΑΜΜΑΤΙΚΗ ΚΑΙ ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ BISON

Όπως το αρχείο λεκτικής αναγνώρισης, το αρχείο συντακτικής αναγνώρισης ακολουθεί την τυπική δομή των αρχείων εισόδου στο bison. Στο πρώτο μέρος υπάρχει κώδικας σε C όπου συμπεριλαμβάνονται βιβλιοθήκες και αρχεία απαραίτητα για την μεταγλώττιση των προγραμμάτων. Στη συνέχεια με την ένωση %union ορίζονται οι σημασιολογικές τιμές της γραμματικής που μπορούν να είναι char ή double. Η ένωση αυτή είναι η παράσταση του τύπου YYSTYPE που περιέχει τη σημασιολογική τιμή της λεκτικής μονάδας που επιστρέφει ο λεκτικός αναλυτής. Μετά ορίζονται οι τύποι των τερματικών και μη τερματικών συμβόλων. Τερματικά σύμβολα είναι οι NUMBER και VAR, τα σύμβολα ‘+’, ‘-‘, ‘*’, ‘/’, ‘=’ είναι αριστερής προτεραιότητας, ενώ μη τερματικό σύμβολο είναι το exp. Στο τελικό τμήμα του πρώτου μέρους τα τερματικά σύμβολα NUMBER και VAR θα χρησιμοποιούν τα πεδία num και vptr της ένωσης union, ενώ το μη τερματικό σύμβολο exp θα χρησιμοποιεί το πεδίο num.

Στο δεύτερο μέρος ορίζονται οι κανόνες παραγωγής της γραμματικής διατυπωμένοι σε BNF.

Η γραμματική σε μορφή BNF:.

```
input:
        | input line
        ;
```

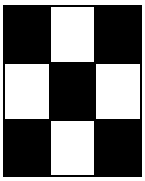
```
line: '\n'
        | exp '\n'        {   printf("H timn eivai %f\n", $1);    }
        | error '\n'      {   yyerror("LATHOS");  }
        ;

exp:    NUMBER    {       $$ = $1;        }
                | VAR                   {       $$ = $1->prize; }
                | VAR '=' exp           {       $$ = $3;  $1->prize = $3; }
                | exp '+' exp           {       $$ = $1 + $3;   }
                | exp '-' exp           {       $$ = $1 - $3;   }
                | exp '*' exp           {       $$ = $1 * $3;   }
                | exp '/' exp           {       $$ = $1 / $3;   }
                | '+' exp               {       $$ = +$2;  }
                | '(' exp ')'           {       $$ = $2;   }
                | '[' exp ']'           {       $$ = $2;   }
                | '-' exp               {       $$ = -$2;  }
                ;
```

Στο τρίτο μέρος έχει τοποθετηθεί κώδικας σε C. Στο τμήμα αυτό καλείται η συνάρτηση main και άλλες βοηθητικές συναρτήσεις. Στην εφαρμογή η συνάρτηση main έχει τοποθετηθεί στο αρχείο εισόδου στο bison και όχι στο αρχείο εισόδου στο flex. Στη main καλείται η συνάρτηση yyparse στην οποία είναι υλοποιημένος ο συντακτικός αναλυτής.

7.4 ΨΗΦΙΟΠΟΙΗΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ

Η διαδικασία ψηφιοποίησης των προγραμμάτων στην πράξη δεν είναι μια απλή διαδικασία. Απαιτούνται πολλές γνώσεις, ειδικά από τον χώρο της επεξεργασίας εικόνας. Επειδή όμως είναι απαραίτητη για την παρούσα εργασία, η ψηφιακή μορφή (σε μορφή δηλαδή 0 και 1) κάποιων αρχείων χρησιμοποιήσαμε μια πολύ απλή τεχνική για την παραγωγή αρχείων σε αυτή τη μορφή. Δεν χρησιμοποιήσαμε κάποιο scanner ή κάποια άλλη τεχνική λήψης, αλλά έτοιμα “ψηφιοποιημένα” αρχεία. Για την παράσταση του κάθε

χαρακτήρα χρησιμοποιούνται εννιά ψηφία 0 ή 1. Για παράδειγμα το γράμμα x μορφή εικόνας bitmap και σε δυαδική μορφή φαίνεται στο παρακάτω σχήμα 7.1(α) – (β) :



Σχήμα 7.1(α) Εικόνα bitmap

1	0	1
0	1	0
1	0	1

Σχήμα 7.1(α) Δυαδική μορφή

Το αντίστοιχο πρότυπο εισόδου που προκύπτει είναι 101010101.

7.5 ΤΟ ΤΕΧΝΗΤΟ ΝΕΥΡΩΝΙΚΟ ΔΙΚΤΥΟ

Το τεχνητό νευρωνικό δίκτυο που αναπτύχθηκε, βασίζεται στον τύπο των πολυεπίπεδων Perceptron. Η κατηγορία αυτή νευρωνικών δικτύων, είναι η ευρύτερα χρησιμοποιούμενη κατηγορία τεχνητών νευρωνικών δικτύων. Για τα δίκτυα αυτά υπάρχει πλούσια βιβλιογραφία αλλά και μεγάλη εμπειρία στην κατασκευή τους. Έχουν σαν βασικό χαρακτηριστικό το γεγονός ότι είναι ο ισχυρότερος τύπος τεχνητού νευρωνικού δικτύου. Από την άλλη έχουν σαν μεγάλο μειονέκτημα ότι είναι πολύ ‘αργή’ η εξαγωγή των αποτελεσμάτων.

Το δίκτυο αποτελείται από τρία επίπεδα: το επίπεδο εισόδου, ένα κρυφό επίπεδο και το επίπεδο εξόδου. Υπάρχει η δυνατότητα οι αριθμοί αυτοί να αλλάζουν κάθε φορά ανάλογα με την εφαρμογή. Οι νευρώνες κάθε επιπέδου συνδέονται με βάρη με όλους τους νευρώνες του επόμενου επιπέδου. Η εκπαίδευση του δικτύου στηρίζεται στον αλγόριθμο οπισθοδρομικής διάδοσης σφάλματος (back error propagation). Η διόρθωση των τιμών των βαρών γίνεται μετά από κάθε εποχή. Σαν εποχή ορίζεται ένα πέρασμα όλων των προτύπων εκπαίδευσης μία φορά. Μετά από αυτό το πέρασμα τα βάρη διορθώνονται κατά την αντίστροφη φορά, με βάση τον αλγόριθμο οπισθοδρομικής διάδοσης σφάλματος. Για τον τερματισμό της εκπαίδευσης επιλέχθηκε το εξής κριτήριο: θεωρούμε ότι το δίκτυο συγκλίνει, όταν η μέση τιμή των σφαλμάτων για κάθε εποχή G_{av} (1.4), γίνει μικρότερη από ένα ικανοποιητικά μικρό κατώφλι τ. Αν για παράδειγμα, ορίσουμε το κατώφλι αυτό σε τ = 0.01, όταν η τιμή του G_{av} γίνει μικρότερη από 0.01 τότε θεωρούμε ότι το δίκτυο συγκλίνει. (Βλέπε Πειραματικά Αποτελέσματα).

7.6 ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Για τον έλεγχο της απόδοσης της εφαρμογής, δημιουργήσαμε 10 ζεύγη από πρότυπα και τα αντίστοιχα διανύσματα εξόδου. Σε 4 από αυτά υπάρχουν συντακτικά λάθη, τα οποία πρέπει η εφαρμογή να ανακαλύψει και να τα παρουσιάσει. Ακολουθούν παραδείγματα προτύπων (ένα χωρίς συντακτικά λάθη και ένα με συντακτικά λάθη) και του αναμενόμε-νου αποτελέσματος. Δίπλα στα πρότυπα εισόδου υπάρχουν και οι εντολές στις οποίες αντιστοιχούν.

Α) Το επόμενο πρότυπο δεν έχει συντακτικά λάθη:

11101010100000000011000011000001000000010010010 (x = 1)

10101110111100011010101010101011110010010010111 (x=x+1)

Τα αναμενόμενα αποτελέσματα είναι:

1

2

Β) Το επόμενο πρότυπο έχει συντακτικά λάθη:

11101010100000000011000011000001000000010010010 (x = 1)

11100011010101010101011110010010010111 (=x+1)

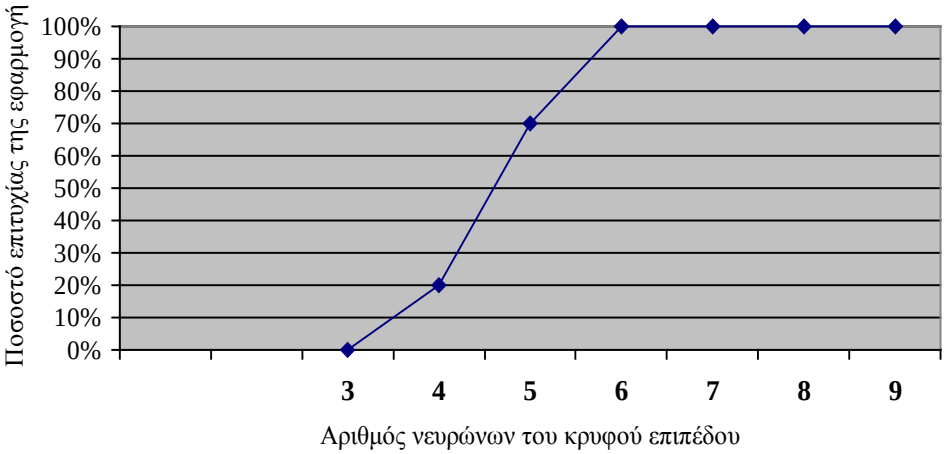
Τα αναμενόμενα αποτελέσματα είναι:

1

parser error.

Όλα τα πρότυπα χρησιμοποιήθηκαν σε δοκιμές μειώνοντας διαδοχικά τον αριθμό των νευρώνων του κρυφού επιπέδου. Όταν οι νευρώνες του κρυφού επιπέδου είναι από 6 έως 9, η εφαρμογή έδωσε τα προσδοκώμενα αποτελέσματα και μεταγλώττισε σωστά και τα 10 πρότυπα. Όταν οι νευρώνες έγιναν 5, ο compiler μεταγλώττισε σωστά τα 7 από τα 10 πρότυπα, ενώ όταν οι νευρώνες έγιναν 4 ο compiler μεταγλώττισε σωστά 2 από τα 10 πρότυπα.

Όταν οι νευρώνες έγιναν 3 ο compiler δεν μεταγλώττισε σωστά κανένα από το 10 πρό-τυπα. Επιπλέον δοκιμές με 2 ή 1 νευρώνα στο κρυφό επίπεδο δεν έγιναν. Στο παρακάτω διάγραμμα βλέπουμε το ποσοστό επιτυχίας συναρτήσει του αριθμού νευρώνων του κρυ-φού επιπέδου.



Παρατηρούμε ότι με 5 έως 7 νευρώνες στο κρυφό επίπεδο επιτυγχάνουμε υψηλά επίπεδα καλής λειτουργίας.

Σε κάθε φάση μείωσης των νευρώνων του κρυφού επιπέδου, έγιναν δοκιμές μείωσης του μέσου σφάλματος G_{av}. Σε κάθε φάση έγιναν δοκιμές με τιμές αρχικά 0,01 και στη συνέχεια 0,001 και 0,0001. Σε όλες τις περιπτώσεις δεν παρουσιάστηκε διαφορά στα αποτελέσματα.

ΠΑΡΑΡΤΗΜΑ

Α.1 ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ FLEX

Σε αυτή την παράγραφο παρουσιάζονται πληροφορίες σχετικά με το αρχείο εισόδου στο flex. Το αρχείο αυτό ονομάζεται calc.l.

Ακολουθεί ο κώδικας του calc.l.

```
%{
        #include <stdio.h>
        #include <math.h>
        #include "calc.h"
        #include "calc.tab.h"
%}

NUM    [0-9]
WORD [a-zA-Z]
%%

{NUM}+"."{NUM}*          { yylval.num = atof(yytext);  return NUMBER; }
{NUM}+                   { yylval.num = atof(yytext);  return NUMBER; }
"+"                      { return '+'; }
"-"                      { return '-'; }
"*"                      { return '*'; }
"/"                      { return '/'; }
"="                      { return '='; }
[\[\]\(\)]+              { return (int) yytext[0]; }
{WORD}+       {
                                variable * s  = (variable *)0;
                                s = getsym(yytext);
                                if (s == 0)
                                {       s = putsym(yytext);       }
                                yylval.vptr = s;
                                return VAR;
```

```
                                        }
"\n"                            { return '\n'; }
[\t' ']

%%

extern variable * tablevar = (variable *)0;
variable * getsym (const char * var_name)
{
        variable * ptr;
        for (ptr = tablevar; ptr != (variable *) 0; ptr = (variable *) ptr->next )
        {
                if (strcmp(ptr->name,var_name) == 0){
                        return ptr;
                }
        }
        return 0;
}

variable * putsym (const char * var_name)
{
        variable *ptr = (variable *) malloc (sizeof (variable));
        ptr->name = (char *) malloc (strlen (var_name) + 1);
        strcpy (ptr->name,var_name);
        ptr->next = (variable *) tablevar;
        tablevar = ptr;
        return ptr;
}
```

Στο πρώτο μέρος συμπεριλαμβάνονται οι βιβλιοθήκες της C stdio.h και math.h καθώς και τα αρχεία calc.y και calc.tab.h. Το αρχείο calc.y είναι το αρχείο εισόδου στο bison και το αρχείο calc.tab.h είναι το αρχείο που παράγεται από το bison. Στο δεύτερο μέρος υπάρχουν οι κανόνες που καθορίζουν ποιες ενέργειες θα γίνουν αν αναγνωριστούν κάποιες λεκτικές μονάδες. Στο τρίτο μέρος του αρχείου, είναι γραμμένες σε κώδικα C οι συναρτη-

σεις getsym(const char * var_name) και putsym (const char * var_name) που χρησιμοποιούνται στον χώρο των κανόνων.

Α.2 ΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΣΤΟ BISON

Σε αυτή την παράγραφο παρουσιάζονται πληροφορίες σχετικά με το αρχείο εισόδου στο bison. Το αρχείο αυτό ονομάζεται calc.y.

Ακολουθεί ο κώδικας του calc.y.

```
%{
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
%}

%union
{
        double num;
        variable * vptr;
}

%token <num> NUMBER
%token <vptr> VAR
%type <num> exp

%left '+'
%left '-'
%left '*'
%left '/'
%left '='

%%
```

```
input:
        | input line
        ;
line: '\n'
        | exp '\n'          {   printf("H timn eivai %f\n", $1);    }
        | exp ' ' '\n'      {   printf("H timn eivai %f\n", $1);    }
        | error '\n'        {   yyerror("LATHOS");  }
        ;
exp:    NUMBER      { $$ = $1;          }
                | VAR                   {       $$ = $1->prize; }
                | VAR '=' exp           {       $$ = $3;  $1->prize = $3; }
                | exp '+' exp           {       $$ = $1 + $3;     }
                | exp '-' exp           {       $$ = $1 - $3;     }
                | exp '*' exp           {       $$ = $1 * $3;     }
                | exp '/' exp           {       $$ = $1 / $3;     }
                | '+' exp               {       $$ = +$2;  }
                | '(' exp ')'           {       $$ = $2;   }
                | '[' exp ']'           {       $$ = $2;   }
                | '-' exp               {       $$ = -$2;  }
                ;
%%
int yyerror(const char * s)
{
        printf("%s\n", s);
}

extern FILE * yyin;

int main(int argc, char * argv[])
{
        variable * tablevar = (variable *)0;
        if (argc > 1)
        {
                yyin = fopen(argv[1], "r");
```

```
        }
        else
        {
                yyin = stdin;
        }
        return yyparse();
}
```

Στο πρώτο μέρος συμπεριλαμβάνονται οι βιβλιοθήκες της C stdlib.h, ctype.h, stdio.h, και math.h καθώς και το αρχείο calc.h. Το αρχείο calc.h περιέχει τη δομή δεδομένων variable που χρησιμοποιείται για την δημιουργία των λεκτικών μονάδων. Στο τρίτο μέρος έχει γραφεί σε C ο κώδικας της συνάρτησης main().

Α.3 ΤΟ ΝΕΥΡΩΝΙΚΟ ΔΙΚΤΥΟ

Σε αυτή την ενότητα παρουσιάζεται ο κώδικας του νευρωνικού δικτύου.

```
#include <math.h>
#include "libexample.h"
#include "headBp.h"

main(int argc, char * argv[])
{
    int c=0;
    int d[numofcat];  //Το επιθυμητό διάνυσμα εξόδου
    float e[numofcat]; //Το διάνυσμα σφάλματος
    float dd1[numofn];  //Ο πίνακας αποκλίσεων πρώτου επιπέδου
    float dd2[numofcat];  //Ο πίνακας αποκλίσεων του τελευταίου επιπέδου
    int i=0,j=0,k=0,l=0,sc=0;
    float ath=0.0,gav=0.0,n=0.1;
    float y1[numofn];  //Διάνυσμα εισόδου
    float y2[numofn];  //Διάνυσμα εξόδων του πρώτου επιπέδου
    float y3[numofcat];  //Ο πίνακας τιμών των εξόδων των νευρώνων
```

```
int num_of_edarray = numofpat; //sizeof(seira) / sizeof(int); //Αριθμός εκπαιδευτικών διανυσμάτων
//num_of_edarray είναι ο αριθμός των εκπαιδευτικών διανυσμάτων
int * seira = (int *)calloc(numofpat,sizeof(int));//Περιέχει την τυχαία σειρά με την οποία θα καλούμε κάθε φορά τα διανύσματα εκπαίδευσης.
float edar[numofn]={0.0,1.0,0.0,1.0,1.0,1.0,0.0,1.0,0.0};
float * w = (float *)calloc((numofn)*(numofn+1),sizeof(float));
rand_weight(w,(numofn)*(numofn+1));
float * w1 = (float *)calloc((numofcat)*(numofn+1),sizeof(float));
rand_weight(w,(numofcat)*(numofn+1));
FILE * fop = fopen(argv[1],"r");
FILE * fop1 = fopen(argv[2],"w");

if(argc == 1)
{
    printf("Δεν έχετε ορίσει αρχείο εισόδου και αρχείο εγγραφής των αποτελεσμάτων \n");
    return;
}
else if(argc == 2)
{
    printf("Δεν έχετε ορίσει αρχείο εγγραφής των αποτελεσμάτων \n");
    return;
}
else if(fop == NULL)
{
    printf("Δεν μπορώ να ανοίξω το αρχείο %s\n",argv[1]);
    return;
}
else if(fop1 == NULL)
{
    printf("Δεν μπορώ να ανοίξω το αρχείο %s\n",argv[2]);
    return;
}
else
```

```
{
    for(i=0; i<=numofpat; i++)
    {
        seira[i] = -1;
    }
    do
    {
        rand1(seira,(numofpat-1)); //Δημιουργούμε την τυχαία σειρά με την οποία θα κα-
    λέσουμε τα διανύσματα εκπαίδευσης. Θα περιέχει αριθμούς από το 0 έως το
    num_of_edarray-1 με τυχαία σειρά
        gav = 0.0;
        for(l=0; l <= numofpat-1; l++)//Περνάει μια εποχή
        {
            copy(y1,&ed_array[0][0],seira[l],numofn);
            //Αντιγράφουμε στο y1 το εκπαιδευτικό διάνυσμα
            front(w,w1,y1,y2,y3,numofn,numofcat);
            //Προσωτροφοδότηση του δικτύου
            patarray(d,ed_array[seira[l]][numofn],numofcat);
            //Αντιγράφουμε στο d το το επιθυμητό πρότυπο εξόδου για το
        εκπαιδευτικό διάνυσμα που χρησιμοποιούμε.
            diafora(d,y3,e,numofcat);
            //Υπολογίζουμε το σφάλμα στην έξοδο του εκπαιδευτικού διανύσματος
            for(i=0; i<=numofcat-1; i++)
            {
                gav=gav + (e[i]*e[i]);
            }
            //Υπολογίζουμε το άθροισμα των γινομένων e[i]*e[i] όπου e[i] η διαφορά
        d-y3 στην έξοδο κάθε νευρώνα του επιπέδου εξόδου.
            local_edd(dd1,dd2,w1,y2,e,y3,numofn,numofcat);
            //Υπολογίζουμε τις τοπικές αποκλίσεις σε οπισθόδρομη πορεία του
        δικτύου
            back(w,w1,dd1,dd2,y1,y1,numofn,numofcat,n);
            //Διορθώνουμε τα βάρη που συνδέουν τους νευρώνες με τους νευρώνες
        των προηγούμενων επιπέδων.
```

```
        }
        //Ολοκληρώνεται μια εποχή
        gav = (1.0/(2.0*( (float)numofpat) ))*gav;    //Υπολογίζουμε τη μέση τιμή σφάλματος μετά το πέρασμα μιας εποχής.
}
while(gav > 0.001);
i=0;
while( (c=getc(fop)) != EOF )
{
        if( (c == '\n') && (i == numofn) )
        {
                j= find(w,w1,&edar[0],&y2[0],&y3[0],numofn,numofcat);
                putc(j,fop1);
                putc('\n',fop1);
                i=0;
                continue;
        }
        else if( (c == '\n') && (i == 0) )
        {
                putc('\n',fop1);
                continue;
        }
        if( i < numofn )
        {
                edar[i] = (float)c - (float)'0';
                i++;
        }
        else
        {
                j= find(w,w1,&edar[0],&y2[0],&y3[0],numofn,numofcat);
                putc(j,fop1);
                i=0;
                edar[i] = (float)c - (float)'0';
                i++;
```

```
                }
        }
        if( i == 9 )
        {
                j= find(w,w1,&edar[0],&y2[0],&y3[0],numofn,numofcat);
                putc(j,fop1);
                fputc('\n',fop1);
        }
    }

    free(w);
    free(w1);
    free(seira);
    fclose(fop);
    fclose(fop1);
    printf("Τέλος\n");
    return;
}
```

Στο αρχείο συμπεριλαμβάνονται τα αρχεία libbp.h και headBp.h. Το αρχείο libbp.h περιέχει συναρτήσεις που είναι απαραίτητες για τη λειτουργία του νευρωνικού. Οι συναρτήσεις αυτές είναι:

```
extern void rand1(int *,int );
extern void  front(float *,float *,float *,float *,float *,int,int);
extern void copy(float *,int *,int,int);
extern void local_edd(float *,float *,float *,float *,float *,float *,int,int);
extern void back(float *,float *,float *,float *,float *,float *,int,int,float);
extern void diafora(int *,float *,float *,int);
extern void patarray(int *,int,int);
extern int find(float *,float *,float *,float *,float *,int,int);
```

Από τις συναρτήσεις αυτές δημιουργήσαμε μία βιβλιοθήκη την libbp.a, την οποία χρησιμοποιήσαμε κατά την διάρκεια της μεταγλώττισης.

Το αρχείο headBp.h περιέχει δεδομένα και πληροφορίες για την λειτουργία του νευρωνικού δικτύου, όπως τα διανύσματα εκπαίδευσης, ο αριθμός των νευρώνων κ.λ.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Ρίζος Γ., ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΘΕΩΡΙΑ ΚΑΙ ΕΦΑΡΜΟΓΕΣ, Εκδόσεις Νέων Τεχνολογιών, 1996

2. Παπασπύρου Ν., Σκορδαλάκης Ε., ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ, Εκδόσεις Συμμετρία, 2002

3. Kernighan B., Ritchie D., Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C, Prentice Hall Software Series, Εκδόσεις Κλειδάριθμος, 1990.

4. Welsh M., Dalheimer M., Kaufman L., Ο ΟΔΗΓΟΣ ΤΟΥ LINUX, Εκδόσεις Κλειδάριθμος.

5. Μπαλτής Σ., ΜΕΘΟΔΟΛΟΓΙΑ ΚΑΙ ΤΕΧΝΙΚΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ, Τυπογραφείο Πανεπιστημίου Ιωαννίνων, 1993.

6. Λύκας, Πανεπιστημιακές παραδόσεις του μαθήματος ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ, Τυπογραφείο Πανεπιστημίου Ιωαννίνων.

ΔΙΕΥΘΥΝΣΕΙΣ INTERNET

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

www.spinellis.gr/comp/

www.cs.uoi.gr/~pitoura/cs462-sping00.html

www.softlab.ntua.gr/~nickie/Books/Compilers/

http://courses.softlab.ntua.gr/compilers/

http://www.combo.org/lex_yacc_page/

http://www.intelligence.tuc.gr/~compilers/index.html

http://www.gnu.org/home.el.html

http://www.gnu.org/software/flex/manual/

http://www.gnu.org/software/flex/flex.html

http://www.gnu.org/software/bison/bison.html

http://gcc.gnu.org/

ΝΕΥΡΩΝΙΚΑ

http://engine.ieee.org/nnc/

http://www.mbfys.kun.nl/snn/

http://www.supelec-rennes.fr/acth/welcome.htm

http://ai.iit.nrc.ca/cscsi_point.html

http://www.neuronet.ph.kcl.ac.uk/neuronet/organizations/enns.html

http://cns-web.bu.edu/inns/

http://www-dsi.ing.unifi.it/neural/siren/sirenEN.html

http://jnns-www.okabe.rcast.u-tokyo.ac.jp/home.html

http://www.kcl.ac.uk/neuronet/about/roadmap/nninocr.html

http://www.nd.com/neurosolutions/products/ns/whatisNN.html

http://www.ccs.neu.edu/home/feneric/charrecnn.html

http://citeseer.nj.nec.com/garris99neural.html

http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html

http://aetos.it.teithe.gr/~kdiamant/Neural_main.htm

http://egnatia.ee.auth.gr/~imatz/ann_index.html

http://www.ntua.gr/archtech/course/left/ADPS-pg.html

Επιστημονικά Περιοδικά

http://alife.santafe.edu/alife/

http://www.csu.edu.au/ci/

http://www.ieee.org/nnc/newsletter/

http://www.ieee.org/nnc/pubs/transactions.html

http://www.elsevier.nl/inca/publications/store/5/0/5/6/2/8/

Προγράμματα

http://www.clients.globalweb.co.uk/nctt/

http://www.mbfys.kun.nl/snn/siena/

http://dmiwww.cs.tut.fi/nfs/Welcome_uk.html

http://www.neuronet.ph.kcl.ac.uk/

http://www.mech.gla.ac.uk/~nactftp/nact.html

http://www.dice.ucl.ac.be/neural-nets/ELENA/ELENA.html

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

1. McCulloch W.S., Pitts W., A logical calculus of the ideas immanent in nervous activity, σελ 115-133, 1943.

2. Hebb D.O., The organisation of behaviour, σελ. 60-78 NY Wiley, 1949.

3. Rosenblatt F., The perceptron, Psychological rev 65 σελ. 386-408, 1958.

4. Widrow B., Hoff M.E., Adaptive switching circuits, IRE WESCON Convention Record part 4 σελ. 96-104, 1960.

5. Minsky M.L,PapertS., Perceprtons p1-20 MIT Press Cambridge MA, 1969.

6. Grossberg S., Adaptive pattern classification and universal recoding, boil cybernetics 23 p121-134, 1976.

7. Kohonen T., Self-organised formation of topologically correct feature maps, Biological cybernetics 43 p59-69, 1982.

8. Widrow B., Hoff M.E., Adaptive switching circuits, IRE WESCON Convention Record, 1960.,

9. Rumelhart D.E., Hinton G.E., Williams R.J., Learning internal representations by error propagation, in Parallel distributed processing vol.1 σελ 318-362, 1986.

10. Sejnowski T.J., Rosenberg C.R., Parallel networks that learn to pronounce English text, Complex systems vol.1 σελ 145-168, 1987.

11. Rosenberg C.R., Revealing the structure of NETtalk' s internal representations, Proc of 9th annual conf of the cognitive science society, σελ 537-554, 1987.

12. Gorman R.P., Sejnowski T.J., Analysis of hidden units in a layered network trained to classify sonar targets, Neural Networks vol.1 σελ 75-89, 1988.

13. Glover D.E., Automated inspection via NN classification of optically derived 2-D Fourier feature signatures, Instrument society of America ISA/88 conf (Oct), 1988.

14. Waibel A., Hanazawa T., et al, Phoneme recognition using time delay neural networks, IEEE trans on ASSP vol.37 no.3 σελ 328-338, 1989.

15. Burr D.J., Experiments with a connectionist text reader, Proc of the 1st IEEE Intl conf on Neural Networks vol.4 σελ 717-724, 1987.

16. Dayhoff R.E., Oayhoff J.E., Neural networks for medical image processing, IEEE SCAMC Proceedings σελ 271-275, 1988.

17. Παπασπύρου Νικ. και Σκορδαλάκης Ε., Μεταγλωτιστές, σελ. 30-32.

18. A. I. Holub, Compiler Design in C, Prentice-Hall International, 1990.

19. M. E. Lesk and E. Schmidt, Lex: A Lexical Analyzer Generator, Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, Nj, 1975.

20. V. R. Volkman, Bison: A GNU Breed of Yacc, C Users Journal, σελ. 117, August 1989.