Πανεπιστήμιο Θεσσαλίας
Τμήμα Πληροφορικής

Μεταγλωττιστές

Παραδείγματα Ενοτήτων 3-6

Ενότητα 3: Λεκτική ανάλυση

Άσκηση 3-1:

Να δώσετε το ενοποιημένο διάγραμμα μετάβασης που αναγνωρίζει τις παρακάτω λεκτικές μονάδες μιας γλώσσας:

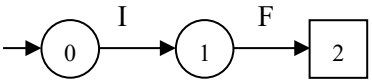
T_IF : *"IF"*
T_INT : *"INT"*
T_INTEGER : *"INTEGER"*
T_ELSE : *"ELSE"*
T_ID : *[A-Z][A-Z0-9]**
T_ICONST : *0\|[1-9][0-9]**

όπου δίπλα σε κάθε λεκτική μονάδα δίνεται η κανονική έκφραση που την περιγράφει. Υποθέστε ότι το αλφάβητο στο οποίο ορίζονται οι λεκτικές μονάδες είναι το αλφάβητο των εκτυπώσιμων ASCII χαρακτήρων.

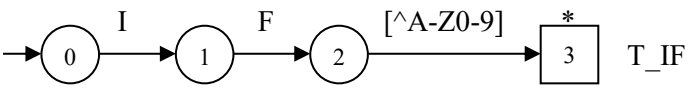
Απάντηση

Για να κατασκευάσουμε το ενοποιημένο διάγραμμα μετάβασης (ΔΜ) της γλώσσας, ξεκινάμε από τα ΠΑ που αναγνωρίζουν τις λεκτικές μονάδες (ΛΜ) της γλώσσας. Στη συνέχεια, είτε μετατρέπουμε τα ΠΑ σε ΔΜ και τα ενοποιούμε, είτε ενοποιούμε τα ΠΑ και μετατρέπουμε το προκύπτον ΜΠΑ-ε σε ΔΜ. Προς το παρόν θα ακολουθήσουμε την πρώτη μέθοδο.

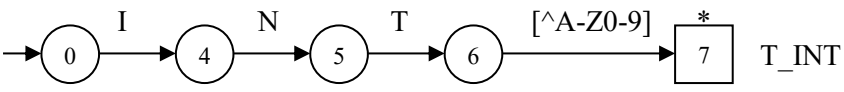
Πρώτα θα κατασκευάσουμε τα ΠΑ και τα αντίστοιχα ΔΜ που αναγνωρίζουν τις τέσσερεις λέξεις-κλειδιά της γλώσσας. Έτσι, το ΝΠΑ για την αναγνώριση της ΛΜ T_IF θα είναι:



Παρατηρήστε ότι το πιο πάνω ΠΑ είναι και ΔΜ χωρίς οπισθοδρόμηση και η τελική κατάστασή του επιστρέφει τη ΛΜ T_IF, αν δεν λάβουμε υπόψη ότι η λέξη μπορεί να συνεχιστεί ως άλλο όνομα. Λαμβάνοντας όμως υπόψη ότι στη συνέχεια θα συμπεριλάβουμε στο τελικό ΔΜ και όλα τα ονόματα της γλώσσας, μπορούμε από τώρα να επεκτείνουμε το πιο πάνω ΔΜ στο ακόλουθο σωστό:



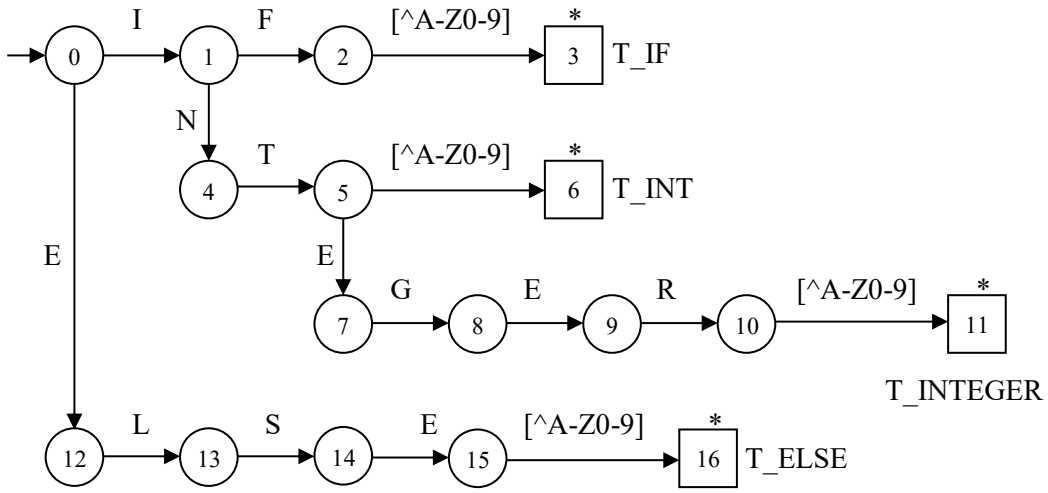
Όμοια σκεπτόμενοι, κατασκευάζουμε το ΔΜ που αντιστοιχεί στη ΛΜ T_INT:



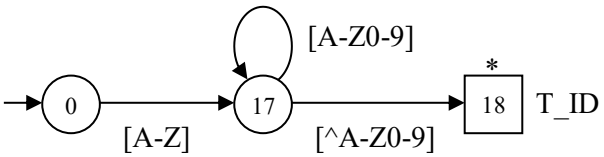
Η ενοποίηση των δύο παραπάνω διαγραμμάτων μπορεί να γίνει με ενοποίηση της κατάστασης 0 και επακόλουθη εφαρμογή του αλγόριθμου μετατροπής ενός ΜΠΑ σε ΝΠΑ, ή απ' ευ-

θείας, με ενοποίηση των δύο πρώτων καταστάσεων των ΔΜ. Με οποιαδήποτε μέθοδο κι αν δουλέψουμε θα πάρουμε το ίδιο αποτέλεσμα.

Με τον ίδιο τρόπο βρίσκουμε και ενσωματώνουμε σε ενιαίο διάγραμμα τα ΔΜ για τις επόμενες δύο ΛΜ της γλώσσας. Το τελικό ΔΜ θα είναι το εξής:



Το ΔΜ που αναγνωρίζει τη ΛΜ T_ID είναι το εξής:

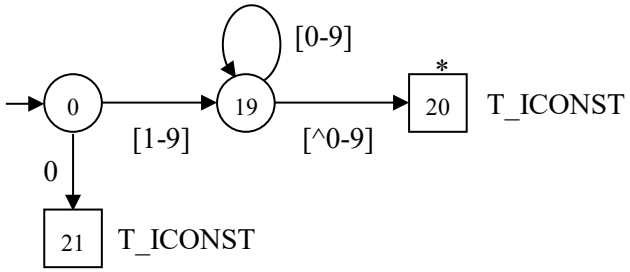


Η ενοποίηση αυτού στο προηγούμενο διάγραμμα απαιτεί τροποποίηση της πρώτης μετάβασης, ώστε να συμβαίνει για τους χαρακτήρες [A-DF-HJ-Z], επειδή οι υπόλοιποι δύο αλφαβητικοί χαρακτήρες συμμετέχουν στις μεταβάσεις μεταξύ της αρχικής κατάστασης 0 και των καταστάσεων 1 και 12. Ταυτόχρονα, πρέπει να συνδέσουμε όλες τις μη τελικές καταστάσεις εκτός της αρχικής με την 17, για όλες τις περιπτώσεις αλφαριθμητικών χαρακτήρων που απορρίπτουν την αντίστοιχη λέξη-κλειδί, και με την 18, για όλες τις περιπτώσεις *μη* αλφαριθμητικών χαρακτήρων. Αυτό συμβαίνει επειδή το ζητούμενο ΔΜ οφείλει να είναι ντετερμινιστικό. Και πάλι, αν απλά ενοποιούσαμε την κατάσταση 0 των διαγραμμάτων και εφαρμόζαμε τον αλγόριθμο μετατροπής ενός ΜΠΑ σε ΝΠΑ, θα παίρναμε το ίδιο αποτέλεσμα.
Δε θα δείξουμε τη γραφική αναπαράσταση του νέου διαγράμματος, αλλά θα δώσουμε τη συνάρτηση μετάβασης για όλες τις νέες μεταβάσεις που προαναφέραμε:

$\delta(0, [\text{A-DF-HJ-Z}]) = \delta(1, [\text{A-EG-MO-Z0-9}]) = \delta(2, [\text{A-Z0-9}]) = \delta(4, [\text{A-SU-Z0-9}]) =$
$= \delta(5, [\text{A-DF-Z0-9}]) = \delta(7, [\text{A-FH-Z0-9}]) = \delta(8, [\text{A-DF-Z0-9}]) = \delta(9, [\text{A-QS-Z0-9}]) =$
$= \delta(10, [\text{A-Z0-9}]) = \delta(12, [\text{A-KM-Z0-9}]) = \delta(13, [\text{A-RT-Z0-9}]) = \delta(14, [\text{A-DF-Z0-9}]) =$
$= \delta(15, [\text{A-Z0-9}]) = 17$

$\delta(1, [\text{\^{}A-Z0-9}]) = \delta(4, [\text{\^{}A-Z0-9}]) = \delta(7, [\text{\^{}A-Z0-9}]) = \delta(8, [\text{\^{}A-Z0-9}]) = \delta(9, [\text{\^{}A-Z0-9}]) =$
$= \delta(12, [\text{\^{}A-Z0-9}]) = \delta(13, [\text{\^{}A-Z0-9}]) = \delta(14, [\text{\^{}A-Z0-9}]) = 18$

Τέλος, το ΔΜ για τη ΛΜ T_ICONST δίνεται πιο κάτω. Η ενσωμάτωση αυτού στο προηγούμενο ΔΜ απαιτεί απλά την ενοποίηση της κατάστασης 0 των δύο διαγραμμάτων, αφού δεν έχουμε άλλη μετάβαση για αριθμητικό χαρακτήρα από την κατάσταση αυτή.



Εναλλακτικά, όπως προαναφέραμε, μπορούμε να κατασκευάσουμε το ζητούμενο ΔΜ απ' ευθείας από το ΝΠΑ που αναγνωρίζει τις ΛΜ της γλώσσας. Κατασκευάζουμε δηλαδή ένα ΠΑ για κάθε ΛΜ, ενοποιούμε τα επιμέρους ΠΑ συνδέοντάς τα με ε-μεταβάσεις από μία νέα κοινή αρχική κατάσταση, μετατρέπουμε το ΜΠΑ-ε που προκύπτει σε ΝΠΑ, και τέλος μετατρέπουμε το ΝΠΑ σε ΔΜ.

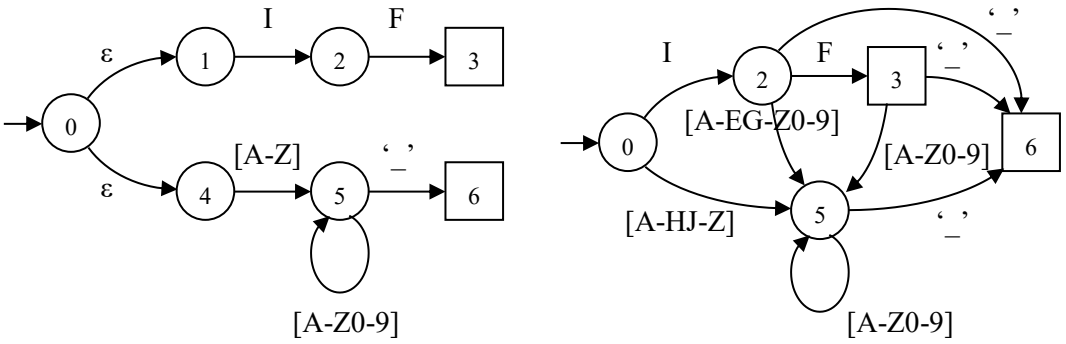
Ειδικότερα, για τη μετατροπή του ΝΠΑ σε ΔΜ, κάνουμε τα εξής:

- Οι τελικές καταστάσεις του ΝΠΑ που *δεν* έχουν μεταβάσεις να ξεκινάνε από αυτές παραμένουν τελικές καταστάσεις του ΔΜ. Αυτές οι τελικές καταστάσεις του ΔΜ θα αναγνωρίζουν τις αντίστοιχες ΛΜ, τις οποίες και αναγράφουμε δίπλα σε αυτές, και δεν θα έχουν οπισθοδρόμηση.
- Κάθε τελική κατάσταση F του ΝΠΑ που έχει τουλάχιστον μία μετάβαση να ξεκινάει από αυτή, θα μετατραπεί σε μη τελική κατάσταση του ΔΜ. Μια ή περισσότερες νέες καταστάσεις θα προστεθούν στο ΔΜ, δημιουργώντας μερικά αντίγραφα του ΝΠΑ για όλες τις μεταβάσεις που ξεκινούν από την F και καταλήγουν με ένα ή περισσότερα βήματα σε κάποια τελική κατάσταση. Πιο συγκεκριμένα, ξεκινώντας από την F: Για κάθε μη τελική κατάσταση A που συναντάμε, εξετάζουμε αν η A είναι προσπελάσιμη μόνο μέσω της F. Αν ναι, δημιουργούμε μια νέα τελική κατάσταση, στην οποία μεταβαίνουμε από την A για όλους τους χαρακτήρες του αλφαβήτου που δεν έχουν μετάβαση από την A στο ΝΠΑ. Η νέα τελική κατάσταση του ΔΜ θα αναγνωρίζει την ίδια ΛΜ που αναγνωρίζει η F, με τόσους χαρακτήρες οπισθοδρόμησης, όσα είναι τα βήματα που κάνουμε από την F για να φτάσουμε σε αυτή. Αν η A *δεν* είναι προσπελάσιμη μόνο μέσω της F – ή αν είναι η αρχική κατάσταση, δημιουργούμε ένα αντίγραφο της A, έστω A', και μετατρέπουμε τη μετάβαση από την F στην A σε μετάβαση από την F στην A' για τον ίδιο χαρακτήρα. Με τη δημιουργία της A', αντιγράφουμε και όλες τις μεταβάσεις από την A σε μεταβάσεις από την A'. Επιπλέον, δημιουργούμε μια νέα τελική κατάσταση στην οποία θα γίνεται μετάβαση από την A', όπως ακριβώς κάναμε και παραπάνω. Επαναλαμβάνουμε την παραπάνω διαδικασία για κάθε κατάσταση που συναντάμε ακολουθώντας τις μεταβάσεις της νέας κατάστασης A', μέχρι να εξαντλήσουμε τις μη τελικές καταστάσεις που εμφανίζονται. Τέλος, δημιουργούμε μια νέα τελική κατάσταση με μετάβαση από την F προς αυτή, για όλους τους χαρακτήρες του αλφαβήτου που δεν έχουν μετάβαση από την F στο ΝΠΑ. Η νέα τελική κατάσταση του ΔΜ θα αναγνωρίζει την ίδια ΛΜ που αναγνωρίζει η F, με ένα χαρακτήρα οπισθοδρόμησης. Σε κάθε τελική κατάσταση που δημιουργήσαμε με την παραπάνω διαδικασία αναγράφουμε τόσο τη ΛΜ που αναγνωρίζεται, όσο και τον αριθμό χαρακτήρων οπισθοδρόμησης, που όπως είπαμε είναι το πλήθος βημάτων που απαιτούνται για να φτάσουμε σε αυτήν από την F.

Η μερική αντιγραφή του ΝΠΑ που κάνουμε παραπάνω είναι απαραίτητη, ώστε το ΔΜ να θυμάται ότι έχει ήδη περάσει από την F, κι επομένως μπορεί να κάνει αναγνώριση. Με την αντιγραφή εξασφαλίζουμε ότι στις καταστάσεις που δημιουργούμε φτάνουμε μόνο μέσω της F.

Αξίζει να σημειωθεί ότι στη διαδικασία που περιγράψαμε είναι πιθανό να βρούμε τελικές καταστάσεις με μη σταθερό αριθμό χαρακτήρων οπισθοδρόμησης. Κάτι τέτοιο θα συμβεί αν στο ΝΠΑ εμφανίζεται κύκλος μεταβάσεων μεταξύ μη τελικών καταστάσεων που συναντάμε κατά τη μετατροπή. Τότε, ο αριθμός χαρακτήρων οπισθοδρόμησης περιορίζεται μόνο από το μήκος της συμβολοσειράς εισόδου κι επομένως δεν είναι άνω φραγμένος.

Εφαρμόζοντας την παραπάνω μέθοδο για την κατασκευή του ΔΜ της άσκησης, καταλήγουμε στο ΔΜ που βρήκαμε νωρίτερα. Ειδικότερα, κατά τη μετατροπή του ΜΠΑ-ε σε ΝΠΑ, όλες οι καταστάσεις του ΝΠΑ γίνονται τελικές καταστάσεις, κι έτσι η εφαρμογή των παραπάνω κανόνων είναι απλή. Για καλύτερη κατανόηση της περίπτωσης μεταβάσεων από μη τελικές καταστάσεις του ΝΠΑ, ας δούμε μια παραλλαγή της άσκησης, όπου η λεκτική μονάδα T_ID περιγράφεται από την κανονική έκφραση [A-Z][A-Z0-9]*‘_’ και η μόνη άλλη λεκτική μονάδα είναι η T_IF. Ένα πιθανό ΜΠΑ-ε και το ισοδύναμο ΝΠΑ θα είναι τα παρακάτω:

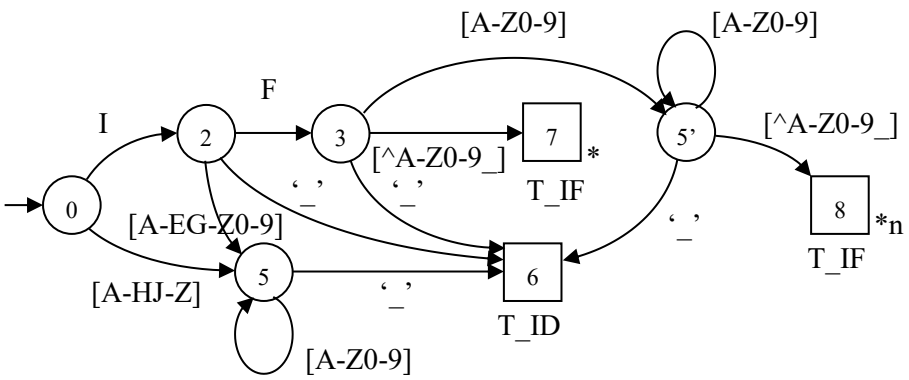


Παρατηρούμε ότι η τελική κατάσταση 6 του ΝΠΑ δεν έχει μεταβάσεις προς άλλες καταστάσεις, κι επομένως στο αντίστοιχο ΔΜ η κατάσταση 6 παραμένει όπως είναι, αναγνωρίζοντας τη λεκτική μονάδα T_ID, χωρίς να χρειαστεί οπισθοδρόμηση.

Όσο αφορά την τελική κατάσταση 3 όμως, παρατηρούμε ότι έχει μετάβαση τόσο προς την κατάσταση 5, που είναι μη τελική, όσο και προς την κατάσταση 6. Έτσι, η κατάσταση 3 του ΝΠΑ θα γίνει μη τελική κατάσταση στο ΔΜ, ενώ θα προσθέσουμε στο ΔΜ μια νέα τελική κατάσταση, έστω 7, στην οποία θα γίνεται μετάβαση από την 3, για όλους τους χαρακτήρες του αλφαβήτου για τους οποίους δεν ορίζεται μετάβαση στο ΝΠΑ, δηλαδή για τους χαρακτήρες που περιγράφονται ως [^A-Z0-9_], αφού για τους υπόλοιπους ορίζεται μετάβαση προς τις καταστάσεις 5 και 6. Η κατάσταση 7 θα αναγνωρίζει τη ΛΜ που αναγνωρίζει η κατάσταση 3 στο ΝΠΑ, δηλαδή την T_IF, και θα υποδεικνύει οπισθοδρόμηση ενός χαρακτήρα, εφ’ όσον διαβάζεται ένας επιπλέον χαρακτήρας για να αποφασιστεί η μετάβαση στην κατάσταση 7 του ΔΜ.

Επειδή η κατάσταση 5 του ΝΠΑ προς την οποία υπάρχει μετάβαση από την 3 είναι προσπελάσιμη και από την αρχική κατάσταση 0, θα δημιουργήσουμε ένα αντίγραφο αυτής, έστω την κατάσταση 5’, στην οποία θα μεταβαίνουμε από την 3 για τους χαρακτήρες [A-Z0-9]. Η 5’ θα έχει τις ίδιες μεταβάσεις με την 5. Επίσης, θα δημιουργήσουμε μια νέα τελική κατάσταση, έστω 8, στην οποία θα γίνεται μετάβαση από την 5’, για τους χαρακτήρες που περιγράφονται ως [^A-Z0-9_], αφού για τους υπόλοιπους ορίζεται μετάβαση. Η κατάσταση 8 θα αναγνωρίζει και αυτή τη ΛΜ T_IF, μόνο που δε θα μπορεί να υποδεικνύει σταθερό αριθμό χαρακτήρων οπισθοδρόμησης, μια που λόγω της κυκλικής μετάβασης στην 5’, δεν είναι δυνατό να ξέρουμε εκ των προτέρων των αριθμό των βημάτων που έγιναν για να φτάσουμε σε αυτή.

Το ΔΜ που προκύπτει με την παραπάνω διαδικασία θα είναι το ακόλουθο:



όπου η ένδειξη *n υποδηλώνει ότι η οπισθοδρόμηση μπορεί να είναι της τάξης του μεγέθους της συμβολοσειράς εισόδου.

Για να υλοποιηθεί ΛΑ με μεταβλητό αριθμό χαρακτήρων οπισθοδρόμησης, θα πρέπει να γίνεται καταμέτρηση των χαρακτήρων που ακολουθούν την πιο πρόσφατη μη τελική κατάσταση του ΔΜ που όμως είναι τελική στο ΝΠΑ. Έτσι, με είσοδο τη συμβολοσειρά "IF10AB2+", ο ΛΑ πρέπει να έχει μετρήσει τους χαρακτήρες που ακολουθούν την υποσυμβολοσειρά "IF", δηλαδή όσους διαβάστηκαν μετά από την κατάσταση 3 που είναι η πιο πρόσφατη τελική κατάσταση του αντίστοιχου ΝΠΑ, ώστε φτάνοντας στην κατάσταση 8 με το χαρακτήρα '+', να επιστρέψει στην είσοδο ακριβώς 6 χαρακτήρες. Η επόμενη κλήση του ΛΑ θα ξεκινήσει από το χαρακτήρα '1'.

Γενικά, τα ΔΜ μπορούν να εμπλουτιστούν με πρόσθετες πληροφορίες που αφορούν τη λεκτική ανάλυση της συμβολοσειράς εισόδου, διότι στην πραγματικότητα αποτελούν *εργαλεία προγραμματισμού* του ΛΑ, και δεν είναι αυστηρά ορισμένα, όπως είναι από την άλλη μεριά τα ΠΑ. Τέτοιες πληροφορίες μπορούν να προστεθούν είτε πάνω στους κόμβους των ΔΜ είτε μέσω καθολικών μεταβλητών. Έτσι, για παράδειγμα, αντί να δημιουργούμε αντίγραφα κόμβων, μπορούμε να σημειώνουμε τις τελικές καταστάσεις του ΝΠΑ που έχουμε περάσει, ώστε να μην κάνουμε αναγνώριση αν φτάσουμε σε τελική κατάσταση του ΔΜ χωρίς να έχουμε περάσει από κάποια τελική κατάσταση του ΝΠΑ, ή ώστε να ξέρουμε ποια ΛΜ αναγνωρίστηκε, σε περίπτωση που σε κάποια τελική κατάσταση του ΔΜ καταλήγουν μονοπάτια που περνάνε από πολλαπλές τελικές καταστάσεις του ΝΠΑ.

Άσκηση 3-2:

Να δώσετε το αρχείο εισόδου στο μετα-εργαλείο flex για την επεξεργασία ενός αρχείου ASCII χαρακτήρων, έτσι ώστε:
(α) να αφαιρούνται από αυτό όλοι οι χαρακτήρες κενού ' ' και '\t',
(β) να αφαιρούνται από αυτό γραμμές σχολίων, οι οποίες αρχίζουν με το χαρακτήρα '$' στην αρχή της γραμμής, και
(γ) να ενώνονται δύο διαδοχικές γραμμές, όταν η πρώτη έχει το χαρακτήρα '\' στο τέλος της, πιθανά πριν από έναν ή περισσότερους από τους πιο πάνω χαρακτήρες κενού, οπότε ο χαρακτήρας '\', οι χαρακτήρες κενού και ο χαρακτήρας αλλαγής γραμμής θα πρέπει να αφαιρούνται. Στην τελευταία περίπτωση δεν επιτρέπεται η επόμενη γραμμή να είναι γραμμή σχολίου, κάτι που πρέπει να ανιχνεύεται σα σφάλμα.
Θεωρήστε ότι η ανάγνωση και η εκτύπωση ASCII χαρακτήρων γίνεται μέσω των αρχείων standard input, standard output και standard error της C.

Απάντηση

Το μετα-εργαλείο flex κατασκευάζει ένα πρόγραμμα C, το οποίο, καθώς διαβάζει το αρχείο χαρακτήρων, εκτελεί τις ενέργειες που αντιστοιχούν σε κάθε κανονική έκφραση (ΚΕ) που αναγνωρίζεται. Οι ενέργειες αυτές καθορίζονται στο β' μέρος του αρχείου εισόδου του flex, δίπλα στην αντίστοιχη ΚΕ.
Οι ΚΕ που μας ενδιαφέρουν για την επεξεργασία του αρχείου χαρακτήρων περιγράφουν (α) τους χαρακτήρες κενού, (β) τις γραμμές σχολίων, (γ) τους χαρακτήρες συνέχισης γραμμής και (δ) τους υπόλοιπους χαρακτήρες. Άλλες βοηθητικές ΚΕ που θα χρειαστούμε στο αρχείο εισόδου του flex, θα προκύψουν στη συνέχεια. Οι ενέργειες – από την άλλη μεριά – που αντιστοιχούν στις πιο πάνω ΚΕ θα είναι είτε αντιγραφή της λέξης στην έξοδο του προγράμματος, είτε τίποτα, κάτι που ισοδυναμεί με αφαίρεση της λέξης από το αρχείο χαρακτήρων. Πιο συγκεκριμένα, αντιγραφή θα γίνεται μόνο στην ΚΕ (δ), αφού για όλες τις άλλες το ζητούμενο είναι η αφαίρεση της αντίστοιχης λέξης. Στο flex, αντιγραφή από την είσοδο στην έξοδο επιτυγχάνεται με τη βοήθεια της ειδικής ενέργειας ECHO.

Για διευκόλυνσή μας αρχίζουμε ορίζοντας στο α' μέρος τους χαρακτήρες κενού ' ' και '\t' με κατάλληλο μνημονικό όνομα:

```
space [ \t]
```

Έτσι, διαδοχικοί χαρακτήρες κενού θα αναγράφονται ως ΚΕ του β' μέρους με τον ακόλουθο τρόπο:

```
{space}+
```

υπονοώντας *τουλάχιστον ένα* χαρακτήρα κενού.
Μια γραμμή σχολίων μπορεί να αναγραφεί ως ΚΕ με τη βοήθεια του μετασυμβόλου '^', ως εξής:

```
^$.*\n
```

Η ΚΕ για τους χαρακτήρες συνέχισης γραμμής θα δίνεται ως:

```
\\{space}*\n
```

δηλώνοντας ότι μετά από το χαρακτήρα '\' και πριν από το χαρακτήρα '\n' θα βρίσκονται *μηδέν ή περισσότεροι* χαρακτήρες κενού.
Για να μπορέσουμε να διαχωρίσουμε τη μη επιτρεπόμενη εμφάνιση του χαρακτήρα '$' αμέσως μετά από χαρακτήρες συνέχισης γραμμής από μια νόμιμη γραμμή σχολίων, αντί να χρησιμοποιήσουμε την παραπάνω ΚΕ για τη γραμμή σχολίων, θα χρησιμοποιήσουμε μια ΚΕ, η οποία να ανιχνεύει απλά το χαρακτήρα '$' στην αρχή της γραμμής, και δύο αρχικές καταστάσεις. Η πρώτη αρχική κατάσταση, έστω CONT, είναι *κοινή* και χρησιμοποιείται όταν έχουμε ανιχνεύσει τους χαρακτήρες συνέχισης γραμμής, και εξετάζουμε να δούμε (α) αν ακολουθεί ο χαρακτήρας '$' και (β) αν φτάνουμε σε τέλος γραμμής χωρίς να έχουμε νέα συνέχισή της. Η δεύτερη αρχική κατάσταση, έστω COMM, είναι *αποκλειστική* και χρησιμοποιείται όταν έχουμε ανιχνεύσει νόμιμη εμφάνιση του χαρακτήρα '$' στην αρχή γραμμής, δηλαδή μια έναρξη σχολίου, και διαβάζουμε τους υπόλοιπους χαρακτήρες του σχολίου.

Σύμφωνα με τα παραπάνω, η ενέργεια που απαιτείται στην ανίχνευση των χαρακτήρων συνέχισης γραμμής θα είναι απλά η μετάβαση στην κατάσταση CONT. Δε χρειάζεται άλλη ενέργεια, αφού οι χαρακτήρες συνέχισης γραμμής πρέπει να αφαιρεθούν. Όσο το πρόγραμμα θα βρίσκεται στην κατάσταση CONT, μπορεί να αναγνωρίσει όλες τις ΚΕ που την κατονομάζουν, καθώς και όλες τις ΚΕ που δεν κατονομάζουν καμία αρχική κατάσταση. Για να ελέγχουμε τις δύο υποπεριπτώσεις που αναφέραμε, αρκεί να προσθέσουμε ενέργειες στις δύο αντίστοιχες ΚΕ, και να τις υποχρεώσουμε να κατονομάζουν την CONT. Η μία ΚΕ θα είναι η:

```
^\$
```

όπου το μετασύμβολο '\' είναι απαραίτητο, αφού ο χαρακτήρας '$' στο τέλος μιας ΚΕ λειτουργεί επίσης ως μετασύμβολο! Η ενέργεια αυτής της ΚΕ θα είναι η διαπίστωση του σφάλματος, με κατάλληλο μήνυμα στο αρχείο εξόδου standard error, και στη συνέχεια η επιστροφή από τη συνάρτηση yylex(). Η δεύτερη ΚΕ θα είναι απλά η:

```
\n
```

με ενέργεια την εκτύπωση της λέξης "\n" και την επάνοδο στην κατάσταση INITIAL. Οι υπόλοιποι χαρακτήρες που πιθανά υπάρχουν πριν από το χαρακτήρα '\n' ανιχνεύονται με άλλες ΚΕ. Αξίζει να σημειωθεί ότι ενδεχόμενη νέα συνέχιση γραμμής θα ανιχνευτεί από την ίδια ΚΕ που ανίχνευσε και την πρώτη συνέχιση.
Αν δε βρισκόμαστε στην κατάσταση CONT, η πρώτη από τις δύο πιο πάνω ΚΕ ανιχνεύει τη νόμιμη έναρξη σχολίου, οπότε απαιτείται ως ενέργεια η μετάβαση στην κατάσταση COMM.

Όταν το πρόγραμμα βρίσκεται στην κατάσταση COMM, οι ΚΕ που αναγνωρίζονται είναι μόνο όσες την κατονομάζουν. Για να αναγνωρίσουμε όλο το υπόλοιπο σχόλιο μέχρι και το τέλος της γραμμής, μπορούμε να χρησιμοποιήσουμε μία μοναδική ΚΕ, την

```
.*\n
```

αλλά προτιμάμε να χρησιμοποιήσουμε δύο ΚΕ, μία για οποιονδήποτε χαρακτήρα πλην του '\n', χρησιμοποιώντας για το σκοπό αυτό το μετασύμβολο '.', και μία για το χαρακτήρα '\n'. Με τον τρόπο αυτό μπορούμε να ανιχνεύσουμε την εμφάνιση του ειδικού συμβόλου EOF μέσα σε σχόλιο, όπως θα δούμε πιο κάτω. Αν χρησιμοποιήσουμε μόνο μία ΚΕ για όλο το σχόλιο μετά το '$', δε θα μπορούσαμε να ανιχνεύσουμε το EOF, κάτι που θα οδηγούσε σε αποτυχία την όλη επεξεργασία.
Με την υλοποίηση των δύο ΚΕ, η πρώτη δε θα έχει καμία ενέργεια, ενώ η δεύτερη θα έχει ως μόνη ενέργεια την επάνοδο στην κατάσταση INITIAL.

Αυτό που μας έχει απομείνει είναι (α) η αναγνώριση και αφαίρεση των υπόλοιπων κενών, και (β) η αντιγραφή των υπόλοιπων χαρακτήρων στην έξοδο. Για την πρώτη λειτουργία, αρκεί να προσθέσουμε την ΚΕ των κενών που γράψαμε νωρίτερα, χωρίς καμία ενέργεια. Για τη δεύτερη λειτουργία, θα χρησιμοποιήσουμε την ΚΕ:

```
.|\n
```

με ενέργεια την εκτύπωση της λέξης που ανιχνεύει.
Για κανονική επιστροφή από τη συνάρτηση yylex() θα εισάγουμε την ειδική ΚΕ του χαρακτήρα EOF, με ενέργεια την επιστροφή τιμής 0. Για να ενεργοποιούμε την έκφραση αυτή από κάθε κατάσταση, είτε κοινή είτε αποκλειστική, τοποθετούμε πριν από αυτή ως συνθήκη κατάστασης την <*>, που σημαίνει οποιαδήποτε κατάσταση.

Με βάση όσα αναφέραμε μέχρι τώρα, το ζητούμενο αρχείο εισόδου του flex θα είναι το παρακάτω:

```
space [ \t]

%option noyywrap
%s CONT
%x COMM

%%

\\{space}*\n { BEGIN(CONT); }
<CONT>{
^\$ {
      fprintf(stderr,"Illegal '$' at a continuation!\n");
      return 1;
    }
\n { ECHO; BEGIN(INITIAL); }
}

^\$ { BEGIN(COMM); }
<COMM>{
.* { }
\n { BEGIN(INITIAL); }
}

<*><<EOF>> { return 0; }

{space}+ { }

.|\n { ECHO; }

%%

int main() {
   return yylex();
}
```

όπου η επιλογή noyywrap υποδηλώνει ότι η επεξεργασία ολοκληρώνεται με το τέλος του τρέχοντος αρχείου εισόδου.

Τελειώνοντας, θα πρέπει να παρατηρήσουμε ότι η τοποθέτηση των ΚΕ στο β' μέρος δε γίνεται με τυχαία σειρά. Ειδικότερα, επειδή η κατάσταση CONT είναι κοινή, η ΚΕ

```
^\$
```

μπορεί να αναγνωριστεί από αυτή σε δύο σημεία. Προκειμένου να ανιχνευτεί η περίπτωση σφάλματος, η ΚΕ που κατονομάζει την CONT τοποθετείται πρώτη. Παρόμοια, η ΚΕ

```
\n
```

που κατονομάζει την CONT τοποθετείται πριν από την ΚΕ

```
.|\n
```

ώστε να είναι δυνατή η επάνοδος από την κατάσταση CONT στην κατάσταση INITIAL. Η ΚΕ

```
.|\n
```

τοποθετείται στο τέλος, αφού το μετασύμβολο '.' αναγνωρίζει όλους τους χαρακτήρες πλην του '\n', κάτι που θέλουμε να συμβαίνει μόνο αν *δεν* γίνεται αναγνώριση από άλλη ΚΕ!

<u>Ενότητα 4:</u> Συντακτική ανάλυση

Άσκηση 4-1:

Να δώσετε μια μέθοδο κατασκευής ΑΣ για την αναγνώριση συμβολοσειρών γλωσσών χωρίς συμφραζόμενα με βάση τη συντακτική ανάλυση από πάνω προς τα κάτω. Εφαρμόστε τη μέθοδό σας σε κάποια γραμματική LL(1). Στη συνέχεια, τροποποιήστε τη μέθοδο που δώσατε, ώστε να κατασκευάζει ΝΑΣ από γραμματικές LL(1), και εφαρμόστε την στην ίδια γραμματική.

Απάντηση

Κάθε γλώσσα χωρίς συμφραζόμενα περιγράφεται από τουλάχιστον μία γραμματική χωρίς συμφραζόμενα, στην οποία κάθε κανόνας έχει στο αριστερό του μέλος ένα μη τερματικό σύμβολο της γραμματικής. Για μια τέτοια γραμματική, η συντακτική ανάλυση από πάνω προς τα κάτω ακολουθεί παρόμοια βήματα με τη διαδικασία παραγωγής, και, ξεκινώντας από το αρχικό σύμβολο της γραμματικής προχωρά σε διαδοχικές αντικαταστάσεις μη τερματικών συμβόλων με αντίστοιχα δεξιά μέλη κανόνων, μέχρι να καταλήξει στη συμβολοσειρά εισόδου, ή μέχρι να απορρίψει τη συμβολοσειρά εισόδου. Αν η διαδικασία αυτή αντικαθιστά τα μη τερματικά σύμβολα από αριστερά προς τα δεξιά, και η συμβολοσειρά εισόδου διαβάζεται αντίστοιχα από αριστερά προς τα δεξιά, το αναγνωρισμένο πρόθεμα των προτασιακών τύπων που δημιουργούνται κατά τις διαδοχικές αντικαταστάσεις μπορεί να διαγραφεί, και το υπόλοιπο να αποθηκευτεί σε μια στοίβα, ώστε να θυμόμαστε ποια μη τερματικά σύμβολα δεν έχουν ακόμα αντικατασταθεί, αλλά και ποια τερματικά σύμβολα από τη συμβολοσειρά εισόδου απομένουν να αναγνωριστούν.
Στη γενική περίπτωση, μια γραμματική χωρίς συμφραζόμενα έχει εναλλακτικά δεξιά μέλη κανόνων για τουλάχιστον ένα μη τερματικό σύμβολό της. Έτσι, η αντικατάσταση αυτού του συμβόλου κατά την παραπάνω διαδικασία είτε είναι μη ντετερμινιστική, είτε *γίνεται* ντετερμινιστική με την προσθήκη κάποιων κριτηρίων που σχετίζονται με σύμβολα από την είσοδο που δεν έχουν ακόμα αναγνωριστεί.
Η μη ντετερμινιστική διαδικασία επιλογής μας οδηγεί σε μη ντετερμινιστικά ΑΣ, στα οποία μέσα σε μια μοναδική μη τελική κατάσταση, έχουμε:
(α) πιθανά πολλαπλές ε-μεταβάσεις για μη τερματικά σύμβολα στην κορυφή της στοίβας, στις οποίες απλά αντικαθιστούμε το σύμβολο στην κορυφή της στοίβας με κάποιο από πιθανά πολλαπλά εναλλακτικά δεξιά μέλη κανόνων, με την αντίστροφη όμως σειρά,
(β) μεταβάσεις ανάγνωσης της εισόδου για τερματικά σύμβολα στην κορυφή της στοίβας, και για ανάγνωση του ίδιου συμβόλου από την είσοδο, οπότε έχουμε και αφαίρεση του συμβόλου από την κορυφή της στοίβας, και
(γ) ε-μεταβάσεις προς μια μοναδική τελική κατάσταση, που ορίζονται όταν η στοίβα περιέχει το αρχικό σύμβολό της, και οδηγούν σε αναγνώριση, μόνο αν η συμβολοσειρά εισόδου έχει εξαντληθεί.
Έτσι, τα μη ντετερμινιστικά ΑΣ για αναγνώριση τυχαίας γλώσσας χωρίς συμφραζόμενα μέσα από μια αντίστοιχη γραμματική χωρίς συμφραζόμενα θα έχουν αλφάβητο εισόδου το σύνολο των τερματικών συμβόλων της γραμματικής, αλφάβητο στοίβας το σύνολο όλων των συμβόλων της γραμματικής και ενός αρχικού συμβόλου X, δύο καταστάσεις, από τις οποίες η μία είναι η αρχική και η άλλη είναι τελική, και μεταβάσεις σύμφωνα με τα παραπάνω. Η πρώτη κίνηση ενός τέτοιου ΑΣ θα είναι μια ε-μετάβαση, με την οποία εισάγεται στη στοίβα το αρχικό σύμβολο της γραμματικής.

Ας θεωρήσουμε τώρα τη γλώσσα που περιγράφεται από την πιο κάτω γραμματική LL(1):

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Το ΑΣ που κατασκευάζεται σύμφωνα με τα παραπάνω θα είναι το $M \equiv (A, Q, H, \delta, q_0, h_0, F)$, όπου:

$A \equiv \{$‘id’, ‘+’, ‘*’, ‘(’, ‘)’$\}$

$Q \equiv \{S, Z\}$

$H \equiv \{X, E, E', T, T', F,$ ‘id’, ‘+’, ‘*’, ‘(’, ‘)’$\}$

$$\delta \equiv \begin{cases} \delta(S, X, \varepsilon) = & \{(\text{“}XE\text{”}, S), (\text{“}X\text{”}, Z)\}, \\ \delta(S, E, \varepsilon) = & \{(\text{“}E'T\text{”}, S)\}, \\ \delta(S, E', \varepsilon) = & \{(\text{“}E'T+\text{”}, S), (\text{“”}, S)\}, \\ \delta(S, T, \varepsilon) = & \{(\text{“}T'F\text{”}, S)\}, \\ \delta(S, T', \varepsilon) = & \{(\text{“}T'F*\text{”}, S), (\text{“”}, S)\}, \\ \delta(S, F, \varepsilon) = & \{(\text{“})E(\text{”}, S), (\text{“id”}, S)\}, \\ \delta(S, \text{‘id’}, \text{‘id’}) = & \{(\text{“”}, S)\}, \\ \delta(S, \text{‘+’}, \text{‘+’}) = & \{(\text{“”}, S)\}, \\ \delta(S, \text{‘*’}, \text{‘*’}) = & \{(\text{“”}, S)\}, \\ \delta(S, \text{‘(’}, \text{‘(’}) = & \{(\text{“”}, S)\}, \\ \delta(S, \text{‘)’}, \text{‘)’}) = & \{(\text{“”}, S)\} \end{cases}$$

$q_0 \equiv S$

$h_0 \equiv X$

$F \equiv \{Z\}$

Για να κάνουμε τη διαδικασία επιλογής εναλλακτικών δεξιών μελών κανόνων ντετερμινιστική, εξετάζουμε ένα ή περισσότερα προπορευόμενα σύμβολα από την είσοδο, *χωρίς όμως να τα καταναλώσουμε*. Η κατανάλωση ενός συμβόλου από την είσοδο στο ΣΑ από πάνω προς τα κάτω γίνεται μόνο όταν είμαστε έτοιμοι να αναγνωρίσουμε το σύμβολο αυτό. Στους αναλυτές της οικογένειας LL(1) κάτι τέτοιο συμβαίνει όταν μετά από κάποια αντικατάσταση μη τερματικού ή προηγούμενη αναγνώριση τερματικού συμβόλου, φτάσουμε σε κάποιο τερματικό σύμβολο από το δεξί μέλος του κανόνα που εξετάζουμε. Η αναγνώριση του συμβόλου θα γίνει, αν βέβαια το σύμβολο που διαβάζουμε από την είσοδο ταυτίζεται με το σύμβολο που θέλουμε να αναγνωρίσουμε, διαφορετικά θα έχουμε απόρριψη της συμβολοσειράς.

Έτσι, αν η γραμματική που αναλύουμε είναι LL(1), μας αρκεί το ένα προπορευόμενο σύμβολο του ΣΑ LL(1), ώστε να μπορούμε να επιλέγουμε τον κανόνα που θα χρησιμοποιήσουμε σε οποιαδήποτε αντικατάσταση. Για να μπορέσουμε τώρα να κατασκευάσουμε ένα ΝΑΣ που να αντιστοιχεί σε κάποιον ΣΑ LL(1), θα πρέπει κατ’ αρχήν να κατανοήσουμε ότι σε κάθε βήμα της λειτουργίας ενός ΝΑΣ μπορούμε να έχουμε είτε μια μετάβαση ανάγνωσης είτε μια ε-μετάβαση. Δε μπορούμε να έχουμε ε-μετάβαση ανάγνωσης, κάποια μετάβαση δηλαδή που, ενώ διαβάζει το επόμενο σύμβολο, δεν το καταναλώνει. Ο ΣΑ LL(1) από την άλλη μεριά, μπορεί να κάνει πολλές κινήσεις με βάση τον προπορευόμενο χαρακτήρα, το χαρακτήρα δηλαδή που, ενώ τον έχει διαβάσει, δεν τον έχει ακόμα καταναλώσει.

Επομένως, για να δούμε πώς ο δεύτερος λειτουργεί ως ΝΑΣ, θα πρέπει να δούμε πώς υλοποιείται μια τέτοια κίνηση σε ένα ΝΑΣ. Επειδή θέλουμε σε κάθε τέτοια περίπτωση να *θυμόμαστε* το επόμενο σύμβολο εισόδου, αρκεί να το διαβάσουμε μια φορά – οπότε και καταναλώνεται, και να ορίσουμε μια κατάλληλη κατάσταση, η οποία θα το *θυμάται*. Έτσι, χρειαζόμαστε τόσες τέτοιες καταστάσεις, όσα είναι τα τερματικά σύμβολα της γραμματικής, μαζί με το ειδικό σύμβολο ‘EOF’ με το οποίο *διαβάζεται* το τέλος της εισόδου. Οι καταστάσεις του ΝΑΣ που υλοποιεί τη ΣΑ LL(1) θα είναι συνολικά οι παραπάνω, μαζί με την αρχική και την τελική κατάσταση που είχαμε και στο μη ντετερμινιστικό ΑΣ.

Οι κινήσεις του ΝΑΣ στηρίζονται στον πίνακα ΣΑ LL(1). Έτσι, κάθε κατάσταση με ένα μη τερματικό σύμβολο στην κορυφή της στοίβας ορίζει μια ε-μετάβαση, στην οποία αφαιρείται το σύμβολο αυτό, και αντικαθίσταται σύμφωνα με τον πίνακα. Τέλος, κάθε κατάσταση με τερματικό σύμβολο στην κορυφή της στοίβας ορίζει μια μετάβαση ανάγνωσης μόνο για το σύμβολο που *θυμάται* η κατάσταση. Αν δηλαδή το πρώτο δεν ταυτίζεται με το δεύτερο, δεν ορίζεται η κίνηση κι έχουμε συντακτικό σφάλμα και απόρριψη της συμβολοσειράς εισόδου. Διαφορετικά, αφαιρούμε το σύμβολο από την κορυφή της στοίβας και μεταβαίνουμε σε μια

νέα κατάσταση, με βάση το επόμενο σύμβολο από την είσοδο. Μετάβαση στην τελική κατάσταση επιτρέπεται μόνο αν έχει διαβαστεί το σύμβολο 'EOF' και η στοίβα περιέχει το αρχικό σύμβολό της.

Τα παραπάνω θα γίνουν πιο κατανοητά, αν τα εφαρμόσουμε στην παραπάνω γραμματική, για να κατασκευάσουμε το ΝΑΣ $N \equiv (A, Q, H, \delta, q_0, h_0, F)$.
Το αλφάβητο εισόδου A θα είναι το σύνολο των τερματικών συμβόλων της γραμματικής, μαζί με το σύμβολο 'EOF':

$A \equiv$ {'id', '+', '*', '(', ')', 'EOF'}

Για καθένα από τα 6 σύμβολα του A, το N έχει μία κατάσταση που μαζί με την αρχική και την τελική δίνουν σύνολο 8 καταστάσεων:

$Q \equiv \{S, S_{id}, S_+, S_*, S_(, S_), S_{EOF}, Z\}$

Εφ' όσον στη στοίβα του ΣΑ εισάγονται όλα τα σύμβολα της γραμματικής, θα έχουμε:

$H \equiv$ {X, E, E', T, T', F, 'id', '+', '*', '(', ')'}

όπου X είναι το αρχικό σύμβολο στοίβας.
Ακόμα:

$q_0 \equiv S$

$h_0 \equiv X$

$F \equiv \{Z\}$

Η συνάρτηση μετάβασης δ του N θα δίνεται από τον παρακάτω πίνακα. Με την πρώτη κίνηση εισάγεται στη στοίβα το αρχικό σύμβολο της γραμματικής και διαβάζεται το πρώτο σύμβολο από την είσοδο. Η κίνηση "read" σημαίνει ανάγνωση ενός συμβόλου από την είσοδο και μετάβαση στην αντίστοιχη κατάσταση, είναι δηλαδή το σύνολο κινήσεων "read('id'), move(S_{id}); read('+'), move(S_+); read('*'), move(S_*); read('('), move($S_($); read(')'), move($S_)$); read('EOF'), move(S_{EOF})". Οι συμβολοσειρές που δίνονται αναφέρονται σε κινήσεις στοίβας και συγκεκριμένα αντικατάσταση του συμβόλου της κορυφής με τη συμβολοσειρά.

	X	E	E'	T	T'	F	id	+	*	(	)	⊣
S	read, push(E)											
S_{id}		keep, "E'T"		keep, "T'F"		keep, "id"	read, pop					
S_+			keep, "E'T+"		keep, pop			read, pop				
S_*					keep, "T'F*"				read, pop			
$S_($		keep, "E'T"		keep, "T'F"		keep, ")E("				read, pop		
$S_)$			keep, pop		keep, pop						read, pop	
S_{EOF}	keep, move(Z)		keep, pop		keep, pop							
Z												accept

Για παράδειγμα, η αναγνώριση της συμβολοσειράς "id + id * id EOF" γίνεται ως εξής:

βήμα	στοίβα	είσοδος	κατάσταση	περιγραφή κίνησης
0	X	id + id * id EOF	S	$\delta(S, X, id) =$ ("XE", S_{id})
1	X E	+ id * id EOF	S_{id}	$\delta(S_{id}, E, \varepsilon) =$ ("E'T", S_{id})
2	X E' T	+ id * id EOF	S_{id}	$\delta(S_{id}, T, \varepsilon) =$ ("T'F", S_{id})
3	X E' T' F	+ id * id EOF	S_{id}	$\delta(S_{id}, F, \varepsilon) =$ ("id", S_{id})
4	X E' T' id	+ id * id EOF	S_{id}	$\delta(S_{id}, id, +) =$ ("", S_+)
5	X E' T'	id * id EOF	S_+	$\delta(S_+, T', \varepsilon) =$ ("", S_+)
6	X E'	id * id EOF	S_+	$\delta(S_+, E', \varepsilon) =$ ("E'T+", S_+)
7	X E' T +	id * id EOF	S_+	$\delta(S_+, +, id) =$ ("", S_{id})

8	X E' T	* id EOF	S_{id}	$\delta(S_{id}, T, \varepsilon) = (\text{“T'F”}, S_{id})$
9	X E' T' F	* id EOF	S_{id}	$\delta(S_{id}, F, \varepsilon) = (\text{“id”}, S_{id})$
10	X E' T' id	* id EOF	S_{id}	$\delta(S_{id}, id, *) = (\text{“”}, S_{*})$
11	X E' T'	id EOF	S_{*}	$\delta(S_{*}, T', \varepsilon) = (\text{“T'F*”}, S_{*})$
12	X E' T' F *	id EOF	S_{*}	$\delta(S_{*}, *, id) = (\text{“”}, S_{id})$
13	X E' T' F	EOF	S_{id}	$\delta(S_{id}, F, \varepsilon) = (\text{“id”}, S_{id})$
14	X E' T' id	EOF	S_{id}	$\delta(S_{id}, id, EOF) = (\text{“”}, S_{EOF})$
15	X E' T'	ε	S_{EOF}	$\delta(S_{EOF}, T', \varepsilon) = (\text{“”}, S_{EOF})$
16	X E'	ε	S_{EOF}	$\delta(S_{EOF}, E', \varepsilon) = (\text{“”}, S_{EOF})$
17	X	ε	S_{EOF}	$\delta(S_{EOF}, X, \varepsilon) = (\text{“X”}, Z)$
18	X	ε	Z	Αναγνώριση

Άσκηση 4-2:

Θεωρήστε τη γραμματική χωρίς συμφραζόμενα με κανόνες:

$S \rightarrow Aa \mid bAc \mid bda$
$A \rightarrow d$

Να δείξετε ότι η γραμματική αυτή είναι LR(1) αλλά όχι SLR(1).

Απάντηση

Για να *μην* είναι μια γραμματική SLR(1), πρέπει μια κατάσταση της ΣΑ SLR(1) αυτής να εμφανίζει σύγκρουση. Αυτό θα συμβεί όταν δύο στοιχεία της κατάστασης έχουν:
(1) κοινά σύμβολα στα αντίστοιχα σύνολα FOLLOW αν είναι στοιχεία ελάττωσης (σύγκρουση ελάττωσης/ελάττωσης), ή
(2) κάποιο σύμβολο ολίσθησης (τερματικό σύμβολο μετά από τελεία) στο ένα από αυτά να ανήκει στο σύνολο FOLLOW του άλλου, όταν αυτό είναι στοιχείο ελάττωσης (σύγκρουση ολίσθησης/ελάττωσης).

Στην παραπάνω γραμματική μπορούμε εύκολα να δούμε ότι:
FOLLOW(S) = {'EOF'}
FOLLOW(A) = {'a','c'}

Ας επιλέξουμε να διερευνήσουμε συγκρούσεις ολίσθησης/ελάττωσης. Οποιαδήποτε ελάττωση για το S (αποδοχή της συμβολοσειράς εισόδου) γίνεται μόνο με προπορευόμενο το 'EOF', το οποίο δε μπορεί να είναι σύμβολο ολίσθησης, επομένως δε μπορεί να δημιουργήσει σύγκρουση ολίσθησης/ελάττωσης. Αντίθετα, ελάττωση για το A γίνεται με προπορευόμενα τα 'a' και 'c', τα οποία μπορούν να είναι στοιχεία ολίσθησης στη γραμματική μας.
Παρατηρούμε ότι ελάττωση για το A συμβαίνει μετά από ανάγνωση του συμβόλου 'd' του δεξιού μέλους για το A. Ο μόνος άλλος κανόνας που περιλαμβάνει το τερματικό σύμβολο 'd', και επομένως θα μπορούσε να έχει στοιχείο στην ίδια κατάσταση, είναι ο κανόνας S → bda. Ειδικότερα, θα ήταν δυνατό να έχουμε σύγκρουση ολίσθησης/ελάττωσης, αν έχουμε στην ίδια κατάσταση ενός ΣΑ SLR(1) τα στοιχεία

S → bd•a
A → d•

επειδή δημιουργούν σύγκρουση για το σύμβολο ολίσθησης 'a' που όπως είδαμε ανήκει στο FOLLOW του A.

Θα εξετάσουμε τώρα (α) αν τα δύο αυτά στοιχεία περιέχονται όντως σε κάποια κατάσταση του ΣΑ SLR(1) για την παραπάνω γραμματική, και (β) αν οι καταστάσεις του ΣΑ LR(1) για την ίδια γραμματική περιέχουν σύγκρουση.

Η αρχική κατάσταση του ΣΑ SLR(1) είναι η S_0:

S_0 = {S → •Aa, S → •bAc, S → •bda, A → •d}
από την οποία με το σύμβολο 'b' παίρνουμε την S_1:
S_1 = {S → b•Ac, S → b•da, A → •d}
Η S_1 με το σύμβολο 'd' μας δίνει την S_2:
S_2 = {S → bd•a, A → d•}
η οποία είναι ακριβώς η κατάσταση που ψάχνουμε, δηλαδή μια κατάσταση με σύγκρουση που εμφανίζεται όταν επόμενο σύμβολο είναι το 'a'.

Επομένως η παραπάνω γραμματική δεν είναι SLR(1).

Επειδή η προηγούμενη ακολουθία καταστάσεων είναι παρούσα σε κάθε αλγόριθμο ΣΑ LR – αγνοώντας όποια προπορευόμενα σύμβολα, για να είναι η γραμματική μας LR(1), θα πρέπει να δούμε αν τα παραπάνω δύο στοιχεία μπορούν να συνοδεύονται από κοινά προπορευόμενα σύμβολα, ενώ θα πρέπει να εξετάσουμε για συγκρούσεις και όλες τις υπόλοιπες καταστάσεις του ΣΑ. Για το σκοπό αυτό, κατασκευάζουμε τις καταστάσεις LR(1) ξεκινώντας με την παραπάνω S_0, στην οποία προσθέτουμε το σύνολο {'EOF'} σα σύνολο προπορευόμενων συμβόλων για τους κανόνες με αριστερό μέλος το σύμβολο S:
S_0 = {S → •Aa{'EOF'}, S → •bAc{'EOF'}, S → •bda{'EOF'}, A → •d{'a'}}
όπου το σύνολο {'a'} προστέθηκε σα σύνολο προπορευόμενων συμβόλων στο στοιχείο
A → •d
επειδή το στοιχείο αυτό προέρχεται από τη συνάρτηση CLOSURE του πρώτου στοιχείου που είναι της μορφής S → •Aα, κι επομένως το σύμβολο A μπορεί να ακολουθείται μόνο από το σύνολο FIRST(α) που στην προκειμένη περίπτωση είναι το FIRST("a"), δηλαδή το {'a'}.
Από την κατάσταση S_0 παίρνουμε την S_1 για το σύμβολο 'b':
S_1 = {S → b•Ac{'EOF'}, S → b•da{'EOF'}, A → •d{'c'}}
Παρατηρούμε ότι το στοιχείο
A → •d
βρίσκεται και στην κατάσταση S_1, αλλά σύμφωνα με τον αλγόριθμο ΣΑ LR(1), το σύνολο προπορευόμενων συμβόλων που τώρα του αποδίδεται θα είναι το FIRST("c"), που είναι το {'c'}.
Από την S_1 παίρνουμε την S_2 για το σύμβολο 'd':
S_2 = {S → bd•a{'EOF'}, A → d•{'c'}}
όπου είναι εμφανές ότι δεν υπάρχει η σύγκρουση που είχαμε νωρίτερα, επειδή η ελάττωση στον κανόνα A → d μπορεί να γίνει μόνο αν το επόμενο σύμβολο είναι το 'c', ενώ η ολίσθηση στο πρώτο στοιχείο της S_2 γίνεται στο σύμβολο 'a'.
Στο σημείο αυτό επαληθεύουμε ότι ο αλγόριθμος LR(1) είναι πιο ισχυρός από τον SLR(1), επειδή υπολογίζει *επακριβώς* σε κάθε κατάσταση το σύνολο των συμβόλων που μπορούν να ακολουθούν το αριστερό μέλος κάθε στοιχείου της.

Συμπληρώνουμε την ανάλυσή μας με την εύρεση των υπόλοιπων καταστάσεων του ΣΑ LR(1) για τη γραμματική μας. Έτσι, από την S_0 με το σύμβολο 'd' βρίσκουμε την S_3:
S_3 = {A → d•{'a'}}
ενώ με το σύμβολο A βρίσκουμε την S_4:
S_4 = {S → A•a{'EOF'}}
Από την S_1 με το σύμβολο A βρίσκουμε την S_5:
S_5 = {S → bA•c{'EOF'}}
και από την S_2 παίρνουμε την S_6 με το σύμβολο 'a':
S_6 = {S → bda•{'EOF'}}
Από την S_4 παίρνουμε την S_7 με το σύμβολο 'a':
S_7 = {S → Aa•{'EOF'}}
και από την S_5 παίρνουμε την S_8 με το σύμβολο 'c':
S_8 = {S → bAc•{'EOF'}}
Επειδή σε καμιά από τις πιο πάνω καταστάσεις δεν υπάρχει σύγκρουση εφόσον περιέχουν ένα μόνο στοιχείο, συμπεραίνουμε ότι η γραμματική μας είναι LR(1).

Ολοκληρώνοντας την άσκηση, μπορούμε να παρατηρήσουμε ότι αν στην αρχή επιλέγαμε να εξετάσουμε τη γραμματική για συγκρούσεις ελάττωσης/ελάττωσης, δεν θα βρίσκαμε καμία, επειδή οι κανόνες δεν μπορούν να οδηγήσουν σε τέτοιες συγκρούσεις. Ειδικότερα, δύο κανόνες μπορούν να οδηγήσουν σε σύγκρουση ελάττωσης/ελάττωσης, μόνο αν το ένα από τα δύο δεξιά μέλη είναι επίθεμα στο άλλο. Διαφορετικά τα στοιχεία ελάττωσής τους δε μπορούν να βρεθούν στην ίδια κατάσταση. Η αναγκαία αυτή συνθήκη δεν ικανοποιείται από κανένα ζεύγος κανόνων στη γραμματική μας.

Άσκηση 4-3:

Να μελετήσετε τη συντακτική ανάλυση της γραμματικής με τους ακόλουθους κανόνες:

$S \rightarrow Az \mid Bx$
$A \rightarrow xAy \mid \varepsilon$
$B \rightarrow yB \mid y$

όπου 'x', 'y' και 'z' τα τερματικά σύμβολα της γραμματικής. Πιο συγκεκριμένα:

Α. Να εξετάσετε αν η παραπάνω γραμματική είναι LL(1). Αν η γραμματική παρουσιάζει εμφανή χαρακτηριστικά που την κάνουν να μην είναι LL(1), βρείτε μια ισοδύναμη γραμματική που να μην εμφανίζει αυτά τα χαρακτηριστικά, και εξετάστε αν εκείνη είναι LL(1). Αν ναι, κατασκευάστε τον πίνακα ΣΑ LL(1) και δείξτε τα βήματα για την αναγνώριση της συμβολοσειράς "xxyyz".

Β. Να εξετάσετε διαδοχικά αν η παραπάνω γραμματική είναι LR(0), SLR(1), LR(1) και LALR(1). Για κάθε περίπτωση, αν η απάντηση είναι καταφατική, κατασκευάστε τον αντίστοιχο πίνακα ΣΑ και δείξτε τα βήματα για την αναγνώριση της πιο πάνω συμβολοσειράς.

Απάντηση

Α. Τρία είναι τα εμφανή χαρακτηριστικά που κάνουν μία γραμματική να μην είναι LL(1): Η αριστερή αναδρομή, το κοινό πρόθεμα στα εναλλακτικά δεξιά μέλη κανόνων και η παραγωγή της κενής συμβολοσειράς από εναλλακτικά δεξιά μέλη κανόνων. Κατ' αρχήν, είναι εύκολο να δούμε ότι η γραμματική που μας δίνεται δεν έχει αριστερή αναδρομή. Παρόλο που έχει άμεση αναδρομή, τόσο στο σύμβολο A, όσο και στο σύμβολο B, σε καμία περίπτωση η αναδρομή δεν είναι αριστερή. Το τρίτο χαρακτηριστικό επίσης δεν εμφανίζεται στη γραμματική. Εφ' όσον τα δεξιά μέλη που περιέχουν τερματικά σύμβολα δεν είναι ποτέ δυνατό να παράγουν την κενή συμβολοσειρά, μένει μόνο μία περίπτωση που την παράγει, κάτι που όμως δεν αρκεί. Όσο για το δεύτερο χαρακτηριστικό, παρατηρούμε ότι αυτό όντως εμφανίζεται στη γραμματική μας, και συγκεκριμένα στους εναλλακτικούς κανόνες για το σύμβολο B, όπου το τερματικό σύμβολο 'y' αποτελεί το κοινό πρόθεμα. Για να κατανοήσουμε καλύτερα γιατί το χαρακτηριστικό αυτό κάνει τη γραμματική να μην είναι LL(1), αρκεί να παρατηρήσουμε ότι λόγω του κοινού προθέματος:

FIRST("yB") = FIRST("y") = {'y'}

ότι δηλαδή τα δύο εναλλακτικά δεξιά μέλη έχουν το ίδιο σύνολο FIRST, κι επομένως παρουσιάζουν σύγκρουση FIRST/FIRST.

Αφού το κοινό πρόθεμα είναι το μόνο εμφανές χαρακτηριστικό που κάνει τη γραμματική να μην είναι LL(1), θα χρησιμοποιήσουμε την τεχνική της αριστερής παραγοντοποίησης, ώστε να βρούμε μια ισοδύναμη γραμματική χωρίς αυτό το χαρακτηριστικό. Γι' αυτό το σκοπό θα χρησιμοποιήσουμε ένα νέο μη τερματικό σύμβολο, έστω C, ως εξής:

S → Az | Bx
A → xAy | ε
B → yC
C → B | ε

Η νέα γραμματική δεν παρουσιάζει πια το δεύτερο από τα πιο πάνω χαρακτηριστικά, αφού το δεύτερο εναλλακτικό δεξί μέλος από εκείνα που εμφάνιζαν το χαρακτηριστικό αυτό μετα-

τράπηκε στην κενή συμβολοσειρά. Μπορεί ακόμα εύκολα να αποδειχτεί ότι με την αριστερή παραγοντοποίηση που κάναμε δεν προέκυψε ούτε κάποιο από τα άλλα δύο χαρακτηριστικά. Αυτό όμως δε σημαίνει ότι η νέα γραμματική είναι LL(1), αφού τα χαρακτηριστικά αυτά αποτελούν ικανή συνθήκη ώστε μια γραμματική να *μην* είναι LL(1), και η έλλειψη αυτών δε συνεπάγεται το αντίθετο.

Για να εξετάσουμε λοιπόν αν η νέα γραμματική είναι LL(1), θα πρέπει να εξετάσουμε τους κανόνες της για πιθανή εμφάνιση συγκρούσεων FIRST/FIRST και FIRST/FOLLOW.
Για το σύμβολο S θα εξετάσουμε τα σύνολα FIRST("Az") και FIRST("Bx"). Για να υπολογίσουμε όμως τα σύνολα αυτά θα χρειαστεί πρώτα να υπολογίσουμε τα σύνολα FIRST(A) και FIRST(B):

FIRST(A) = FIRST("xAy") $\cup$ {ε} = {'x', ε}
FIRST(B) = FIRST("yC") = {'y'}

οπότε:

FIRST("Az") = (FIRST(A) – {ε}) $\cup$ FIRST("z") = {'x', 'z'}
FIRST("Bx") = FIRST(B) = {'y'}

και επομένως FIRST("Az") $\cap$ FIRST("Bx") = ∅, δηλαδή οι εναλλακτικοί κανόνες για το S δεν εμφανίζουν σύγκρουση FIRST/FIRST. Επίσης, δεν εμφανίζουν ούτε και σύγκρουση FIRST/FOLLOW, αφού κανένα από τα δύο εναλλακτικά δεξιά μέλη δεν παράγει την κενή συμβολοσειρά.
Για το σύμβολο A παρατηρούμε ότι ε $\notin$ FIRST("xAy"), κι επομένως οι δύο εναλλακτικοί κανόνες του δεν εμφανίζουν σύγκρουση FIRST/FIRST. Επειδή όμως το δεύτερο εναλλακτικό δεξί μέλος είναι η κενή συμβολοσειρά, θα πρέπει να υπολογίσουμε το σύνολο FOLLOW(A):

FOLLOW(A) = FIRST("z") $\cup$ FIRST("y") = {'z', 'y'}

και αφού FIRST(A) $\cap$ FOLLOW(A) = {'x', ε} $\cap$ {'z', 'y'} = ∅, συμπεραίνουμε ότι οι κανόνες για το A δεν εμφανίζουν ούτε σύγκρουση FIRST/FOLLOW.
Για το σύμβολο B δε νοείται σύγκρουση FIRST/FIRST, αφού έχει μόνο έναν κανόνα. Επίσης, αφού το δεξί μέλος αυτού του κανόνα δεν είναι η κενή συμβολοσειρά, δεν υπάρχει ούτε σύγκρουση FIRST/FOLLOW.
Για το τελευταίο σύμβολο C έχουμε ήδη βρει ότι το σύνολο FIRST(B) δεν περιέχει την κενή συμβολοσειρά, κι επομένως οι δύο εναλλακτικοί κανόνες του δεν εμφανίζουν σύγκρουση FIRST/FIRST. Επειδή το δεύτερο εναλλακτικό δεξί μέλος είναι η κενή συμβολοσειρά, θα πρέπει να υπολογίσουμε τα σύνολα FIRST(C) και FOLLOW(C):

FIRST(C) = FIRST(B) $\cup$ {ε} = {'y', ε}
FOLLOW(C) = FOLLOW(B) = FIRST("x") = {'x'}

και εφ' όσον FIRST(C) $\cap$ FOLLOW(C) = {'y', ε} $\cap$ {'x'} = ∅, συμπεραίνουμε ότι ούτε οι κανόνες για το C δεν εμφανίζουν σύγκρουση FIRST/FOLLOW.
Με βάση τα παραπάνω, μπορούμε να συμπεράνουμε με ασφάλεια ότι η νέα γραμματική που βρήκαμε είναι LL(1).

Ο πίνακας ΣΑ LL(1) για την καινούργια γραμματική κατασκευάζεται με βάση τα σύνολα FIRST όλων των εναλλακτικών κανόνων της, ενώ για τα σύμβολα A και C που παράγουν και την κενή συμβολοσειρά, κατασκευάζεται και με βάση τα αντίστοιχα σύνολα FOLLOW. Όλα τα απαραίτητα σύνολα είναι ήδη υπολογισμένα, οπότε ο ζητούμενος πίνακας θα είναι ο ακόλουθος:

	'x'	**'y'**	**'z'**	**EOF**
S	S → Az	S → Bx	S → Az	
A	A → xAy	A → ε	A → ε	
B		B → yC		
C	C → ε	C → B		

και η αναγνώριση της συμβολοσειράς "xxyyz" θα γίνει με τα εξής βήματα:

βήμα	στοίβα	είσοδος	περιγραφή κίνησης
0	S	x x y y z $	κανόνας S → Az
1	z A	x x y y z $	κανόνας A → xAy
2	z y A x	x x y y z $	απορρόφηση συμβόλου 'x'
3	z y A	x y y z $	κανόνας A → xAy
4	z y y A x	x y y z $	απορρόφηση συμβόλου 'x'
5	z y y A	y y z $	κανόνας A → ε
6	z y y	y y z $	απορρόφηση συμβόλου 'y'
7	z y	y z $	απορρόφηση συμβόλου 'y'
8	z	z $	απορρόφηση συμβόλου 'z'
9	ε	$	αναγνώριση

Β. Στη ΣΑ από κάτω προς τα επάνω εξετάζουμε τη γραμματική για συγκρούσεις ολίσθησης/ελάττωσης και ελάττωσης/ελάττωσης. Οι συγκρούσεις αυτές εμφανίζονται σε καταστάσεις του ΣΑ, κι επομένως η εξέταση για συγκρούσεις μπορεί να γίνει ταυτόχρονα με την εύρεση των καταστάσεων του ΣΑ.
Για το ΣΑ LR(0), όπως και για τους υπόλοιπους ΣΑ της οικογένειας LR, η εύρεση των καταστάσεων αρχίζει με την κατάσταση που περιέχει τα στοιχεία που ξεκινούν τους εναλλακτικούς κανόνες για το αρχικό σύμβολο της γραμματικής, καθώς και όλα τα στοιχεία που προστίθενται με διαδοχική εφαρμογή της συνάρτησης CLOSURE. Παρατηρούμε ότι εφ' όσον το αρχικό σύμβολο της γραμματικής δε βρίσκεται στο δεξί μέλος κάποιου κανόνα, δε χρειάζεται να διευρύνουμε τη γραμματική. Έτσι, η αρχική κατάσταση, έστω I_0, θα είναι:

I_0 ≡ CLOSURE({S → •Az, S → •Bx}) =
={S → •Az, S → •Bx, A → •xAy, A → •, B → •yB, B → •y}

όπου μετά τα δύο πρώτα στοιχεία, τα δύο επόμενα προκύπτουν με την εφαρμογή της συνάρτησης CLOSURE στο πρώτο, και τα δύο τελευταία με την εφαρμογή της συνάρτησης CLOSURE στο δεύτερο στοιχείο. Βλέπουμε ότι η κατάσταση I_0 περιέχει ένα στοιχείο ελάττωσης, το στοιχείο A → •, και τρία στοιχεία ολίσθησης, τα A → •xAy, B → •yB και B → •y, κι επομένως ήδη από την I_0 ξέρουμε ότι η γραμματική μας δεν είναι LR(0), αφού εμφανίζει τουλάχιστον μία σύγκρουση ολίσθησης/ελάττωσης. Από την πρώτη κιόλας στιγμή, ένας ΣΑ LR(0) της γραμματικής που μας δίνεται δε μπορεί να αποφασίσει αν θα ξεκινήσει με ολίσθηση ή με ελάττωση.
Για να εξετάσουμε αν η γραμματική είναι SLR(1), πρέπει να εξετάσουμε τις καταστάσεις LR(0) στις οποίες εμφανίζονται συγκρούσεις, και να δούμε αν οι συγκρούσεις αυτές επιλύονται με βάση τα σύνολα FOLLOW των συμβόλων του αριστερού μέλους των στοιχείων ελάττωσης. Να εξετάσουμε δηλαδή αν για κάθε σύγκρουση ολίσθησης/ελάττωσης το σύμβολο ολίσθησης *δεν* περιέχεται στο σύνολο FOLLOW του συμβόλου του αριστερού μέλους του στοιχείου ελάττωσης, και αν για κάθε σύγκρουση ελάττωσης/ελάττωσης τα σύνολα FOLLOW των αντίστοιχων συμβόλων αριστερού μέλους *δεν* έχουν κοινά στοιχεία. Αν έστω και για μία περίπτωση κάτι τέτοιο δεν επαληθεύεται, τότε η γραμματική δεν είναι ούτε SLR(1). Τα σύνολα FOLLOW της γραμματικής μας προκύπτουν ως εξής:

FOLLOW(S) = {'EOF'}
FOLLOW(A) = FIRST("z") ∪ FIRST("y") = {'z', 'y'}
FOLLOW(B) = FIRST("x") ∪ FOLLOW(B) = {'x'}

Έτσι, στην περίπτωση της σύγκρουσης που εντοπίσαμε παραπάνω, το σύμβολο 'y', στο οποίο γίνεται ολίσθηση στα δύο τελευταία στοιχεία της κατάστασης, περιέχεται στο σύνολο FOLLOW του A, στο οποίο γίνεται ελάττωση. Επομένως, η σύγκρουση δεν επιλύεται, γεγονός που μας οδηγεί στο συμπέρασμα ότι η γραμματική δεν είναι SLR(1).

Για το ΣΑ LR(1) τώρα, ενεργούμε από την αρχή, κατασκευάζοντας εκ νέου τις καταστάσεις του, ώστε να συμπεριλάβουμε τα επιτρεπόμενα προπορευόμενα σύμβολα κάθε στοιχείου:

I_0 ≡ CLOSURE({S → •Az{$}, S → •Bx{$}}) =

$= \{S \rightarrow \bullet Az\{\$\}, S \rightarrow \bullet Bx\{\$\}, A \rightarrow \bullet xAy\{\text{‘z’}\}, A \rightarrow \bullet\{\text{‘z’}\}, B \rightarrow \bullet yB\{\text{‘x’}\},$
$B \rightarrow \bullet y\{\text{‘x’}\}\}$

όπου το μοναδικό προπορευόμενο σύμβολο για το αρχικό σύμβολο της γραμματικής είναι το σύμβολο τέλους συμβολοσειράς, ενώ τα προπορευόμενα σύμβολα στα υπόλοιπα τέσσερα στοιχεία προκύπτουν από την εφαρμογή της συνάρτησης CLOSURE. Τα προπορευόμενα σύμβολα ορίζονται για κάθε στοιχείο του ΣΑ, όμως χρησιμοποιούνται μόνο από τα στοιχεία ελάττωσης, ως τα σύμβολα για τα οποία επιτρέπεται η συγκεκριμένη κίνηση. Έτσι, στην παραπάνω κατάσταση όπου υπάρχει ένα στοιχείο ελάττωσης, βλέπουμε ότι κίνηση ελάττωσης επιτρέπεται μόνο για το σύμβολο ‘z’, στο οποίο δεν γίνεται ολίσθηση σε κανένα άλλο στοιχείο της κατάστασης. Ενώ στην αντίστοιχη κατάσταση του ΣΑ SLR(1) επιτρέπαμε ελάττωση και για τα δύο σύμβολα από το σύνολο FOLLOW(A), οπότε με το σύμβολο ‘y’ προέκυπτε σύγκρουση, εδώ η σύγκρουση έχει επιλυθεί.

Φυσικά, η επίλυσης της παραπάνω σύγκρουσης δε σημαίνει ότι τελειώσαμε. Ίσα-ίσα, τώρα ξεκινάμε ουσιαστικά τη μελέτη της γραμματικής, αφού θα πρέπει να εξετάσουμε όλες τις καταστάσεις του ΣΑ LR(1) για τυχόν συγκρούσεις. Έτσι, η διαδοχική εφαρμογή των συναρτήσεων GOTO και CLOSURE θα δώσει:

$I_1 \equiv \text{CLOSURE}(\text{GOTO}(I_0, A)) = \text{CLOSURE}(\{S \rightarrow A\bullet z\{\$\}\}) = \{S \rightarrow A\bullet z\{\$\}\}$
$I_2 \equiv \text{CLOSURE}(\text{GOTO}(I_0, B)) = \text{CLOSURE}(\{S \rightarrow B\bullet x\{\$\}\}) = \{S \rightarrow B\bullet x\{\$\}\}$
$I_3 \equiv \text{CLOSURE}(\text{GOTO}(I_0, \text{‘x’})) = \text{CLOSURE}(\{A \rightarrow x\bullet Ay\{\text{‘z’}\}\}) =$
$= \{A \rightarrow x\bullet Ay\{\text{‘z’}\}, A \rightarrow \bullet xAy\{\text{‘y’}\}, A \rightarrow \bullet\{\text{‘y’}\}\}$
$I_4 \equiv \text{CLOSURE}(\text{GOTO}(I_0, \text{‘y’})) = \text{CLOSURE}(\{B \rightarrow y\bullet B\{\text{‘x’}\}, B \rightarrow y\bullet\{\text{‘x’}\}\}) =$
$= \{B \rightarrow y\bullet B\{\text{‘x’}\}, B \rightarrow y\bullet\{\text{‘x’}\}, B \rightarrow \bullet yB\{\text{‘x’}\}, B \rightarrow \bullet y\{\text{‘x’}\}\}$
$I_5 \equiv \text{CLOSURE}(\text{GOTO}(I_1, \text{‘z’})) = \text{CLOSURE}(\{S \rightarrow Az\bullet\{\$\}\}) = \{S \rightarrow Az\bullet\{\$\}\}$
$I_6 \equiv \text{CLOSURE}(\text{GOTO}(I_2, \text{‘x’})) = \text{CLOSURE}(\{S \rightarrow Bx\bullet\{\$\}\}) = \{S \rightarrow Bx\bullet\{\$\}\}$
$I_7 \equiv \text{CLOSURE}(\text{GOTO}(I_3, A)) = \text{CLOSURE}(\{A \rightarrow xA\bullet y\{\text{‘z’}\}\}) =$
$= \{A \rightarrow xA\bullet y\{\text{‘z’}\}\}$
$I_8 \equiv \text{CLOSURE}(\text{GOTO}(I_3, \text{‘x’})) = \text{CLOSURE}(\{A \rightarrow x\bullet Ay\{\text{‘y’}\}\}) =$
$= \{A \rightarrow x\bullet Ay\{\text{‘y’}\}, A \rightarrow \bullet xAy\{\text{‘y’}\}, A \rightarrow \bullet\{\text{‘y’}\}\}$
$I_9 \equiv \text{CLOSURE}(\text{GOTO}(I_4, B)) = \text{CLOSURE}(\{B \rightarrow yB\bullet\{\text{‘x’}\}\}) = \{B \rightarrow yB\bullet\{\text{‘x’}\}\}$
$I_{10} \equiv \text{CLOSURE}(\text{GOTO}(I_7, \text{‘y’})) = \text{CLOSURE}(\{A \rightarrow xAy\bullet\{\text{‘z’}\}\}) =$
$= \{A \rightarrow xAy\bullet\{\text{‘z’}\}\}$
$I_{11} \equiv \text{CLOSURE}(\text{GOTO}(I_8, A)) = \text{CLOSURE}(\{A \rightarrow xA\bullet y\{\text{‘y’}\}\}) =$
$= \{A \rightarrow xA\bullet y\{\text{‘y’}\}\}$
$I_{12} \equiv \text{CLOSURE}(\text{GOTO}(I_{11}, \text{‘y’})) = \text{CLOSURE}(\{A \rightarrow xAy\bullet\{\text{‘y’}\}\}) =$
$= \{A \rightarrow xAy\bullet\{\text{‘y’}\}\}$

καθώς και:

$\text{CLOSURE}(\text{GOTO}(I_4, \text{‘y’})) = \text{CLOSURE}(\{B \rightarrow y\bullet B\{\text{‘x’}\}, B \rightarrow y\bullet\{\text{‘x’}\}\}) = I_4$
$\text{CLOSURE}(\text{GOTO}(I_8, \text{‘x’})) = \text{CLOSURE}(\{A \rightarrow x\bullet Ay\{\text{‘y’}\}\}) = I_8$

Οι καταστάσεις I_1 και I_2 δεν περιέχουν κανένα στοιχείο ελάττωσης, άρα δε μπορούν να εμφανίσουν σύγκρουση. Η κατάσταση I_3 περιέχει το στοιχείο ελάττωσης $A \rightarrow \bullet$ και το στοιχείο ολίσθησης $A \rightarrow \bullet xAy$, αλλά επειδή η ελάττωση επιτρέπεται μόνο για το προπορευόμενο σύμβολο ‘y’, το οποίο διαφέρει από το σύμβολο ‘x’ με το οποίο γίνεται η ολίσθηση, σύγκρουση δεν υπάρχει. Παρόμοια και για την επόμενη κατάσταση I_4, όπου υπάρχει ένα στοιχείο ελάττωσης, το $B \rightarrow y\bullet$, και δύο στοιχεία ολίσθησης, τα $B \rightarrow \bullet yB$ και $B \rightarrow \bullet y$, αλλά και εδώ σύγκρουση δεν υπάρχει, αφού η ελάττωση επιτρέπεται μόνο για το σύμβολο ‘x’, ενώ η ολίσθηση γίνεται με το σύμβολο ‘y’. Οι καταστάσεις I_5, I_6 και I_7 περιέχουν ένα μόνο στοιχείο, οπότε σε αυτές δε νοείται σύγκρουση. Η κατάσταση I_8 περιέχει τα ίδια στοιχεία με την I_3 και διαφέρει από αυτήν σε ένα προπορευόμενο σύμβολο, όχι όμως σε εκείνο του στοιχείου ελάττωσης, επομένως δεν παρουσιάζει σύγκρουση. Οι υπόλοιπες καταστάσεις περιέχουν ένα μόνο στοιχείο, οπότε ούτε αυτές έχουν σύγκρουση.

Αφού λοιπόν η μελέτη των καταστάσεων LR(1) της γραμματικής δεν έδειξε σύγκρουση, συμπεραίνουμε ότι αυτή είναι LR(1), οπότε μπορούμε να προχωρήσουμε στην κατασκευή του

αντίστοιχου πίνακα ΣΑ. Αριθμούμε τους κανόνες με τη σειρά που δίνονται, από το 1 έως το 6, οπότε καταλήγουμε στον ακόλουθο πίνακα:

	'x'	**'y'**	**'z'**	**EOF**	**A**	**B**
0	s3	s4	r4		1	2
1			s5			
2	s6					
3	s8	r4			7	
4	r6	s4				9
5				r1(acc)		
6				r2(acc)		
7		s10				
8	s8	r4			11	
9	r5					
10			r3			
11		s12				
12		r3				

Με βάση τον πίνακα αυτόν, η αναγνώριση της συμβολοσειράς "xxyyz" θα γίνει με τα εξής βήματα:

βήμα	**στοίβα**	**είσοδος**	**περιγραφή κίνησης**
0	0	x x y y z $	ολίσθηση 'x' και μετάβαση στην I_3
1	0 x 3	x y y z $	ολίσθηση 'x' και μετάβαση στην I_8
2	0 x 3 x 8	y y z $	ελάττωση με τον κανόνα A → ε και μετάβαση στην I_{11}
3	0 x 3 x 8 A 11	y y z $	ολίσθηση 'y' και μετάβαση στην I_{12}
4	0 x 3 x 8 A 11 y 12	y z $	ελάττωση με τον κανόνα A → xAy και μετάβαση στην I_7
5	0 x 3 A 7	y z $	ολίσθηση 'y' και μετάβαση στην I_{10}
6	0 x 3 A 7 y 10	z $	ελάττωση με τον κανόνα A → xAy και μετάβαση στην I_1
7	0 A 1	z $	ολίσθηση 'z' και μετάβαση στην I_5
8	0 A 1 z 5	$	αναγνώριση

Για το ΣΑ LALR(1), τέλος, χρησιμοποιούμε τις καταστάσεις LR(1) για να κατασκευάσουμε το σύνολο καταστάσεών του, συμπτύσσοντας τις καταστάσεις που έχουν ίδια στοιχεία, ενώνοντας τα αντίστοιχα σύνολα προπορευόμενων συμβόλων. Οι καταστάσεις στις οποίες γίνεται σύμπτυξη είναι οι I_3 και I_8, οι I_7 και I_{11}, καθώς και οι I_{10} και I_{12}. Οι νέες καταστάσεις, έστω $I_{3\text{-}8}$, $I_{7\text{-}11}$ και $I_{10\text{-}12}$, αντίστοιχα, θα είναι οι ακόλουθες:

$I_{3\text{-}8}$ ≡ { A → x•Ay{'z', 'y'}, A → •xAy{'y'}, A → •{'y'} }
$I_{7\text{-}11}$ ≡ { A → xA•y{'z', 'y'} }
$I_{10\text{-}12}$≡ { A → xAy•{'z', 'y'} }

Οι μεταβάσεις που σχετίζονταν με τις καταστάσεις I_3, I_7, I_8, I_{10}, I_{11} και I_{12} τώρα γίνονται για τις νέες καταστάσεις.
Εφ' όσον για την εύρεση των καταστάσεων LALR(1) συμπτύσσονται καταστάσεις LR(1) με τα ίδια στοιχεία, οι μεταβάσεις από συμπτυγμένες καταστάσεις θα είναι οι ίδιες με αυτές των αρχικών καταστάσεων, με τη διαφορά ότι οι καταστάσεις στις οποίες γίνεται μετάβαση μπορεί να είναι κι αυτές συμπτυγμένες. Για παράδειγμα, στη γραμματική που μελετάμε η μετάβαση από την κατάσταση $I_{3\text{-}8}$ για το σύμβολο 'x' θα γίνεται προς τον εαυτό της, ενώ για το

σύμβολο Α θα γίνεται προς τη νέα κατάσταση $I_{7\text{-}11}$. Επίσης, η μετάβαση από την $I_{7\text{-}11}$ για το σύμβολο 'y' θα γίνεται προς την κατάσταση $I_{10\text{-}12}$.
Η ένωση των συνόλων προπορευόμενων συμβόλων επηρεάζει μόνο τις κινήσεις ελάττωσης, αφού τα προπορευόμενα σύμβολα χρησιμοποιούνται μόνο σε αυτές. Έτσι, μπορούμε να δούμε ότι με τη σύμπτυξη που κάναμε στη γραμματική μας, η ελάττωση των καταστάσεων LR(1) I_3 και I_8 παραμένει ως έχει στην κατάσταση $I_{3\text{-}8}$, για το ίδιο δηλαδή προπορευόμενο σύμβολο, και συγκεκριμένα το 'y'. Από την άλλη όμως μεριά, η ελάττωση των καταστάσεων LR(1) I_{10} και I_{12} έχει αλλάξει, αφού στη νέα κατάσταση $I_{10\text{-}12}$ η ελάττωση γίνεται τόσο για το σύμβολο 'z', όσο και για το σύμβολο 'y'. Η αλλαγή αυτή δε δημιουργεί κάποια σύγκρουση, δεδομένου ότι το στοιχείο ελάττωσης είναι το μόνο στοιχείο της κατάστασης $I_{10\text{-}12}$.
Η διαφορά στην ελάττωση του στοιχείου A → xAy• είναι και η μόνη διαφορά μεταξύ του ΣΑ LALR(1) από τον LR(1) για τη γραμματική που μας δίνεται. Επειδή δε η διαφορά αυτή δεν είναι αιτία σύγκρουσης, η γραμματική μας είναι και LALR(1). Γενικά οι συμπτύξεις καταστάσεων LR(1) είναι πιθανό να *δημιουργήσουν* συγκρούσεις, επειδή διευρύνουν τα σύνολα προπορευόμενων συμβόλων που επιτρέπονται για ελάττωση, αλλά δεν *επιλύουν* συγκρούσεις. Έτσι, ένας ΣΑ LALR(1) αποτελεί ουσιαστικά μια προσέγγιση προς τα κάτω του αντίστοιχου LR(1), έχοντας το μικρότερο δυνατό αριθμό καταστάσεων, αλλά δεν είναι πάντα ελεύθερος συγκρούσεων, ακόμα κι αν ο LR(1) είναι.
Ο πίνακας ΣΑ LALR(1) για τη γραμματική μας μπορεί να προκύψει από σύμπτυξη του πιο πάνω πίνακα LR(1), ή με εξ αρχής κατασκευή του, και θα είναι ο εξής:

	'x'	**'y'**	**'z'**	**EOF**	**A**	**B**
0	s3-8	s4	r4		1	2
1			s5			
2	s6					
3-8	s3-8	r4			7-11	
4	r6	s4				9
5				r1(acc)		
6				r2(acc)		
7-11		s10-12				
9	r5					
10-12		r3	r3			

οπότε η αναγνώριση της συμβολοσειράς "xxyyz" θα γίνει με τα εξής βήματα, τα ίδια με τα βήματα του ΣΑ LR(1):

βήμα	**στοίβα**	**είσοδος**	**περιγραφή κίνησης**
0	0	x x y y z $	ολίσθηση 'x' και μετάβαση στην $I_{3\text{-}8}$
1	0 x 3-8	x y y z $	ολίσθηση 'x' και μετάβαση στην $I_{3\text{-}8}$
2	0 x 3-8 x 3-8	y y z $	ελάττωση με τον κανόνα A → ε και μετάβαση στην $I_{7\text{-}11}$
3	0 x 3-8 x 3-8 A 7-11	y y z $	ολίσθηση 'y' και μετάβαση στην $I_{10\text{-}12}$
4	0 x 3-8 x 3-8 A 7-11 y 10-12	y z $	ελάττωση με τον κανόνα A → xAy και μετάβαση στην $I_{7\text{-}11}$
5	0 x 3-8 A 7-11	y z $	ολίσθηση 'y' και μετάβαση στην $I_{10\text{-}12}$
6	0 x 3-8 A 7-11 y 10-12	z $	ελάττωση με τον κανόνα A → xAy και μετάβαση στην I_1
7	0 A 1	z $	ολίσθηση 'z' και μετάβαση στην I_5
8	0 A 1 z 5	$	αναγνώριση

Οι καταστάσεις του ΣΑ LALR(1) είναι οι ίδιες με τις καταστάσεις του LR(0), κι άρα με τις καταστάσεις του SLR(1), με την εξαίρεση των προπορευόμενων συμβόλων. Η διαφορά του LALR(1) από τον SLR(1) για τη γραμματική μας εντοπίζεται στις καταστάσεις I_0 και $I_{3\text{-}8}$, όπου οι αντίστοιχες ελαττώσεις γίνονται για υποσύνολο του συνόλου FOLLOW(A). Παρ' ό,τι

στην $I_{3\text{-}8}$ δε θα μας πείραζε αν γινόταν ελάττωση και για το σύμβολο ‘z’, στην κατάσταση I_0 δε μπορεί να γίνει ελάττωση για το ‘y’. Έτσι, ενώ ο ΣΑ SLR(1) έχει σύγκρουση στην I_0, ο ΣΑ LALR(1) επιλύει τη σύγκρουση όπως ακριβώς και ο LR(1). Γενικά, επειδή τα σύνολα προπορευόμενων συμβόλων που προκύπτουν από τις συμπτύξεις καταστάσεων στον LALR(1) δε μπορούν να είναι μεγαλύτερα από τα σύνολα FOLLOW των αντίστοιχων αριστερών μελών, ο ΣΑ LALR(1) δε μπορεί να είναι λιγότερο ακριβής από τον αντίστοιχο SLR(1). Μπορούμε έτσι να καταλάβουμε γιατί ο ΣΑ LALR(1) αποτελεί καλύτερη προσέγγιση του LR(1) απ’ ό,τι ο SLR(1), γι’ αυτό και προτιμάται σε υλοποιήσεις μεταγλωττιστών.

Αξίζει τέλος να σημειώσουμε ότι το γεγονός ότι οι ΣΑ LALR(1) και SLR(1) αποτελούν προσέγγιση του LR(1) σημαίνει ότι μπορεί να οδηγήσουν σε *λανθασμένες ελαττώσεις*, ελαττώσεις δηλαδή που επιτρέπονται για τερματικά σύμβολα που δε θα έπρεπε κανονικά να βρίσκονται στο συγκεκριμένο σημείο της εισόδου. Κάτι τέτοιο δεν οδηγεί σε εσφαλμένες αναγνωρίσεις ή απορρίψεις συμβολοσειρών, όμως οδηγεί σε καθυστερημένη ανίχνευση του σφάλματος της συμβολοσειράς. Για παράδειγμα, αν στους πιο πάνω ΣΑ που κατασκευάσαμε δώσουμε ως είσοδο τη συμβολοσειρά “xyy”, ο LR(1) δε θα επιτρέψει ελάττωση στην κατάσταση I_{10}, ενώ αντίθετα ο LALR(1) θα επιτρέψει ελάττωση στη συμπτυγμένη κατάσταση $I_{10\text{-}12}$, αλλά δε θα μπορεί να συνεχίσει μόλις βρεθεί στην I_1 και στην είσοδο ακολουθεί ‘y’. Ακόμα και αν η κατάσταση I_1 όριζε κίνηση για το σύμβολο ‘y’, ο ΣΑ LALR(1) θα σταματούσε σε άλλο σημείο.

Ενότητες 5-6: Ονόματα, τύποι, σημασιολογική ανάλυση και πίνακες συμβολων

Άσκηση 5-1:

Να δώσετε ένα απλό σύνολο σημασιολογικών κανόνων για τις ενέργειες που πρέπει να γίνονται για τη διαχείριση πολλαπλών εμβελειών στη μεταγλώττιση μιας γλώσσας προγραμματισμού που επιτρέπει φωλιασμένες εμβέλειες.

Απάντηση

Η διαχείριση των πολλαπλών εμβελειών μπορεί να γίνεται με τη βοήθεια μιας καθολικής μεταβλητής range_level η οποία περιέχει το τρέχον επίπεδο εμβέλειας. Ξεκινώντας με τιμή 0, η μεταβλητή αυξάνει τιμή σε κάθε είσοδο εμβέλειας και ελαττώνει σε κάθε έξοδο εμβέλειας. Δεδομένου ότι οι εμβέλειες είναι τέλεια φωλιασμένες, ώστε να μην έχουμε έξοδο από μια εμβέλεια πριν τις εξόδους των εσωτερικότερων εμβελειών, η χρήση της μεταβλητής range_level αρκεί για να πληροφορεί το σημασιολογικό αναλυτή σε ποιο επίπεδο εμβέλειας βρίσκεται.

Ειδικότερα, όσο αφορά τις αναφορές σε αναγνωριστικά, ο διαχωρισμός αυτών με τις πολλαπλές εμβέλειες επιτυγχάνεται με αποθήκευση του επιπέδου εμβέλειας στην πληροφορία που τα συνοδεύει στον πίνακα συμβόλων. Έτσι για παράδειγμα, η αναφορά σε μια μεταβλητή x της εμβέλειας 0 διαχωρίζεται από την αναφορά σε μια μεταβλητή x της εμβέλειας 2.
Πιο συγκεκριμένα, κατά τη δήλωση ενός αναγνωριστικού αποθηκεύουμε την τιμή του επιπέδου εμβέλειας σε κατάλληλο πεδίο της εγγραφής που φτιάχνουμε γι' αυτό στον πίνακα συμβόλων, ελέγχοντας ταυτόχρονα αν υπάρχει ήδη αναγνωριστικό με το ίδιο όνομα στην ίδια εμβέλεια. Σε μια τέτοια περίπτωση έχουμε σημασιολογικό σφάλμα. Πιθανή ύπαρξη του ίδιου ονόματος σε χαμηλότερο επίπεδο εμβέλειας δε μας ενοχλεί, ενώ την παρούσα στιγμή δεν είναι δυνατό να έχουμε ενεργή εμβέλεια σε ψηλότερο επίπεδο.
Κατά τη χρήση ενός αναγνωριστικού από την άλλη μεριά, ελέγχουμε την ύπαρξη δήλωσής του. Για το σκοπό αυτό, διαπερνάμε τον πίνακα συμβόλων με τέτοιον τρόπο, ώστε να ελέγξουμε πρώτα την τρέχουσα ενεργή εμβέλεια, και αν δε βρούμε δήλωση σε αυτήν, να συνεχίσουμε με χαμηλότερα επίπεδα εμβελειών με αναζήτηση σε φθίνουσα σειρά επιπέδων. Η δήλωση που θα βρούμε πρώτη θα είναι και η ζητούμενη, ενώ αν δε βρούμε καμία δήλωση, έχουμε σημασιολογικό σφάλμα.

Έτσι λοιπόν, κατά την είσοδο σε μια εμβέλεια δεν κάνουμε τίποτα παραπάνω από την αύξηση του επιπέδου εμβέλειας κατά 1:

```
Enter_range:
      range_level++;
```

Κατά την έξοδο όμως από μια εμβέλεια, πρέπει να σβήσουμε από τον πίνακα συμβόλων όλα τα αναγνωριστικά που εισάγαμε για την εμβέλεια αυτή. Σε μια μέθοδο μετάφρασης οδηγούμενης από τη σύνταξη έχουμε ολοκληρώσει την ανάλυση της εμβέλειας αυτής με την παραγωγή ενδιάμεσου κώδικα, κι έτσι τη στιγμή εξόδου τα αναγνωριστικά αυτά είναι πια άχρηστα για τον πίνακα συμβόλων. Μετά τη διαγραφή των αναγνωριστικών αυτών, μειώνουμε το επίπεδο εμβέλειας κατά 1. Στη συνέχεια με την επόμενη είσοδο εμβέλειας θα κατασκευάσουμε από την αρχή όποιες εγγραφές στον πίνακα συμβόλων χρειαστούμε για αναγνωριστικά της εμβέλειας αυτής. Έτσι:

```
Exit_range:
      Delete_symbol_table(range=range_level);
      range_level--;
```

Οι παραπάνω ενέργειες τοποθετούνται σε κατάλληλα σημεία των κανόνων της γλώσσας.

Άσκηση 6-1:

Θεωρήστε την υποθετική γλώσσα προγραμματισμού που είδαμε στην άσκηση 1-1. Η γλώσσα αυτή μπορεί να χρησιμοποιείται σε κομπιουτεράκια τσέπης, όπου τα προγράμματα δεν εκτελούνται μέσω τελικού κώδικα, αλλά μέσω διερμηνείας πάνω στη σημασιολογική ανάλυσή τους. Κάθε εντολή δηλαδή εκτελείται κατά την ανάλυσή της με βάση τους σημασιολογικούς κανόνες της γλώσσας και όχι με εκτέλεση κάποιου τελικού κώδικα.

Υποθέστε ότι οι δηλώσεις των μεταβλητών (βαθμωτών ή διανυσματικών) που χρησιμοποιούνται σε μια εντολή ανάθεσης γίνονται με ειδικές εντολές δήλωσης που περιγράφονται με άλλους κανόνες, και οι οποίες δεσμεύουν τον απαραίτητο χώρο μνήμης για κάθε μεταβλητή.

Για την αποτίμηση των τιμών που συμμετέχουν σε μια εντολή ανάθεσης αποδίδουμε στα σύμβολα της πιο πάνω γραμματικής τα ακόλουθα κατηγορήματα:

```
CONST : int val
ID    : int *lval
var   : int *lval
term  : int val
expr  : int val
paren : int val
```

όπου το κατηγόρημα val περιέχει μια ακέραια τιμή, ενώ το κατηγόρημα lval περιέχει ένα δείκτη σε μια ακέραια τιμή, δηλαδή μια διεύθυνση μνήμης. Τα κατηγορήματα των τερματικών συμβόλων παίρνουν τιμές από κάποιο λεκτικό αναλυτή.

Εάν (1) η διατιθέμενη μνήμη περιέχει MAXMEM θέσεις ακεραίων, που αριθμούνται από 0 έως MAXMEM-1 και αρχικά είναι μηδενισμένες, (2) η προσπέλαση κάποιας μεταβλητής δεν ελέγχει τον τύπο της (βαθμωτή ή διανυσματική) ούτε τα όρια του πιθανού δείκτη προσπέλασης, (3) η χρήση μη δηλωμένης μεταβλητής – γεγονός που αναγνωρίζεται με απόδοση τιμής MAXMEM στο αντίστοιχο κατηγόρημα – οδηγεί σε σφάλμα ανάλυσης, (4) η προσπέλαση διεύθυνσης εκτός ορίων μνήμης οδηγεί σε σφάλμα εκτέλεσης και (5) η διαίρεση με 0 οδηγεί σε σφάλμα εκτέλεσης:

Α. Δώστε ένα σύνολο σημασιολογικών κανόνων για την αποτίμηση των πιο πάνω κατηγορημάτων που να είναι σύμφωνοι με τους περιορισμούς που δίνονται.

Β. Δώστε τη σειρά εκτέλεσης των σημασιολογικών κανόνων που δώσατε, για την αποτίμηση των εκφράσεων που συμμετέχουν στην ανάθεση:

```
y2(i+1) = 101 / (x-33*j) + 2
```

Απάντηση

Α. Εφ' όσον χρησιμοποιούμε τη μέθοδο διερμηνείας πάνω στη σημασιολογική ανάλυση για την εκτέλεση των προγραμμάτων της υποθετικής γλώσσας, οι σημασιολογικοί κανόνες που ζητούνται πρέπει να εκτελούν απ' ευθείας τις πράξεις που περιγράφονται στους αντίστοιχους συντακτικούς κανόνες. Διαφορετικά, οι σημασιολογικοί κανόνες θα εκτελούσαν μόνο ελέγχους σημασιολογικής ορθότητας του προγράμματος, και μέσω κατάλληλων σημασιολογικών ρουτινών θα παρήγαγαν ενδιάμεσους ή τελικούς κώδικες για τους αντίστοιχους συντακτικούς κανόνες. Τότε, οι κώδικες αυτοί θα ήταν διαθέσιμοι για εκτέλεση σε κάποια επόμενη στιγμή[1].

Μπορούμε να σχεδιάσουμε τους ζητούμενους κανόνες σε έναν ψευδοκώδικα C ως εξής:

[1] Εναλλακτικά της διερμηνείας πάνω στη σημασιολογική ανάλυση, θα μπορούσαμε να υλοποιήσουμε διερμηνεία πάνω στον ενδιάμεσο κώδικα. Η δεύτερη μέθοδος διερμηνείας προϋποθέτει μεν την παραγωγή του ενδιάμεσου κώδικα, αλλά είναι πιο γρήγορη όταν έχουμε δομές βρόχων, μια που τότε η σημασιολογική ανάλυση γίνεται μόνο μία φορά και δεν επαναλαμβάνεται. Στη συνέχεια της άσκησης θα χρησιμοποιήσουμε ενδιάμεσο κώδικα για καλύτερη κατανόηση της διερμηνείας, αν και δεν κάνουμε διερμηνεία πάνω στον ενδιάμεσο κώδικα!

1. Για το συντακτικό κανόνα

   ```
   assign → var = expr
   ```

   ο αντίστοιχος σημασιολογικός κανόνας ελέγχει τη διεύθυνση μνήμης για την ανάθεση (κατηγόρημα lval του συμβόλου var) και εκτελεί την ανάθεση:

   ```
   {
    if (0 <= var.lval < MAXMEM) *var.lval = expr.val;
    else ERROR("Illegal memory access");
   }
   ```

2. Για το συντακτικό κανόνα

   ```
   var → ID paren
   ```

   ο αντίστοιχος σημασιολογικός κανόνας υπολογίζει μια διεύθυνση μνήμης (κατηγόρημα lval του συμβόλου var) προσθέτοντας στη διεύθυνση της μεταβλητής με όνομα ID (κατηγόρημα lval του συμβόλου ID) την τιμή ενός δείκτη, δηλαδή του κατηγορήματος val του συμβόλου paren:

   ```
   {
    if (ID.lval != MAXMEM) var.lval = ID.lval + paren.val;
    else ERROR("Undefined variable");
   }
   ```

   Παρατηρήστε ότι στον κανόνα αυτόν ελέγχεται μόνο η ύπαρξη δήλωσης της μεταβλητής. Δεν ελέγχεται ούτε ο τύπος της μεταβλητής ούτε η τιμή του δείκτη. Ο έλεγχος της τελικής διεύθυνσης που υπολογίζεται γίνεται σε άλλους κανόνες.

3. Για τους δύο εναλλακτικούς συντακτικούς κανόνες

   ```
   paren → ( expr ) | ε
   ```

   ο αντίστοιχος πρώτος σημασιολογικός κανόνας αντιγράφει την τιμή της παράστασης (κατηγόρημα val του συμβόλου expr) στο κατηγόρημα val του συμβόλου paren, ενώ ο δεύτερος μηδενίζει το ίδιο κατηγόρημα, εφ' όσον δεν υπάρχει παράσταση:

   ```
   {
    paren.val = expr.val;
   }

   {
    paren.val = 0;
   }
   ```

4. Για τους εναλλακτικούς συντακτικούς κανόνες

   ```
   expr → expr + term | expr − term | term
   ```

   οι αντίστοιχοι σημασιολογικοί εκτελούν τις όποιες πράξεις και είναι προφανείς:

   ```
   {
    expr0.val = expr1.val + term.val;
   }

   {
    expr0.val = expr1.val − term.val;
   }

   {
    expr.val = term.val;
   }
   ```

   όπου υποθέτουμε ότι δεικτοδοτούμε κατάλληλα τις διπλές εμφανίσεις του συμβόλου expr στους δύο πρώτους συντακτικούς κανόνες.

5. Τέλος, για τους εναλλακτικούς συντακτικούς κανόνες

   `term → term * var | CONST / term | var | CONST | ( expr )`

   οι αντίστοιχοι σημασιολογικοί πρέπει να είναι οι εξής:

```
{
 if (0 <= var.lval < MAXMEM)
    term0.val = term1.val * (*var.lval);
 else ERROR("Illegal memory access");
}

{
 if (term1.val != 0) term0.val = CONST.val / term1.val;
 else ERROR("Divide by zero");
}

{
 if (0 <= var.lval < MAXMEM) term.val = *var.lval;
 else ERROR("Illegal memory access");
}

{
 term.val = CONST.val;
}

{
 term.val = expr.val;
}
```

Παρατηρήστε ότι όπου υπάρχει αναφορά σε μεταβλητή μέσω του συμβόλου var, υπάρχει αντίστοιχος έλεγχος της διεύθυνσης της μεταβλητής, ενώ στην περίπτωση της πράξης της διαίρεσης ελέγχουμε την τιμή του διαιρέτη σύμφωνα με τους περιορισμούς που μας δίνονται.

Β. Παρ' ό,τι η διερμηνεία της εντολής που μας δίνεται γίνεται μέσα από τη σημασιολογική ανάλυσή της, μπορούμε να χρησιμοποιήσουμε το ΔΣΑ που δώσαμε στην άσκηση 1-1, για να κατανοήσουμε τη σειρά με την οποία εκτελούνται οι σημασιολογικοί κανόνες που εκτελούν την εντολή, και η οποία αντιστοιχεί σε μια διαπέραση του δέντρου από κάτω προς τα επάνω και από αριστερά προς τα δεξιά. Το κομπιουτεράκι δεν κατασκευάζει κανένα δέντρο, και η σειρά αυτή προκύπτει από τη σειρά με την οποία εφαρμόζονται οι συντακτικοί κανόνες κατά την ανάλυση της εντολής:

1. Εύρεση διεύθυνσης για τη μεταβλητή i:

```
{
 parenᵢ.val = 0;
 if (IDᵢ.lval != MAXMEM) varᵢ.lval = IDᵢ.lval + parenᵢ.val;
 else ERROR("Undefined variable");
}
```

2. Υπολογισμός του αριστερού μέλους της παράστασης "i+1":

```
{
 if (0 <= varᵢ.lval < MAXMEM) termᵢ.val = *varᵢ.lval;
 else ERROR("Illegal memory access");
 exprᵢ.val = termᵢ.val;
}
```

3. Υπολογισμός του δεξιού μέλους της παράστασης "i+1":

```
{
 term1.val = CONST1.val;
}
```

4. Αποτίμηση της παράστασης "i+1":

```
{
 exprᵢ₊₁.val = exprᵢ.val + term1.val;
}
```

5. Εύρεση διεύθυνσης για τη μεταβλητή y2(i+1):
```
{
 paren_y2.val = expr_i+1.val;
 if (ID_y2.lval != MAXMEM) var_y2.lval = ID_y2.lval + paren_y2.val;
 else ERROR("Undefined variable");
}
```
6. Εύρεση διεύθυνσης για τη μεταβλητή x:
```
{
 paren_x.val = 0;
 if (ID_x.lval != MAXMEM) var_x.lval = ID_x.lval + paren_x.val;
 else ERROR("Undefined variable");
}
```
7. Υπολογισμός του αριστερού μέλους της παράστασης "x-33*j":
```
{
 if (0 <= var_x.lval < MAXMEM) term_x.val = *var_x.lval;
 else ERROR("Illegal memory access");
 expr_x.val = term_x.val;
}
```
8. Υπολογισμός του αριστερού μέλους της παράστασης "33*j":
```
{
 term_33.val = CONST_33.val;
}
```
9. Εύρεση διεύθυνσης για τη μεταβλητή j:
```
{
 paren_j.val = 0;
 if (ID_j.lval != MAXMEM) var_j.lval = ID_j.lval + paren_j.val;
 else ERROR("Undefined variable");
}
```
10. Αποτίμηση της παράστασης "33*j":
```
{
 if (0 <= var_j.lval < MAXMEM)
    term_33*j.val = term_33.val * (*var_j.lval);
 else ERROR("Illegal memory access");
}
```
11. Αποτίμηση της παράστασης "x-33*j":
```
{
 expr_x-33*j.val = expr_x.val – term_33*j.val;
}
```
12. Υπολογισμός του δεξιού μέλους της παράστασης "101 / (x-33*j)":
```
{
 term_(x-33*j).val = expr_x-33*j.val;
}
```
13. Αποτίμηση της παράστασης "101 / (x-33*j)":
```
{
 if (term_(x-33*j).val != 0)
    term_101/(x-33*j).val = CONST_101.val / term_(x-33*j).val;
 else ERROR("Divide by zero");
}
```
14. Υπολογισμός του αριστερού μέλους της παράστασης "101 / (x-33*j) + 2":
```
{
 expr_101/(x-33*j).val = term_101/(x-33*j).val;
}
```
15. Υπολογισμός του δεξιού μέλους της παράστασης "101 / (x-33*j) + 2":
```
{
 term_2.val = CONST_2.val;
}
```
16. Αποτίμηση της παράστασης "101 / (x-33*j) + 2":
```
{
 expr_101/(x-33*j)+2.val = expr_101/(x-33*j).val + term_2.val;
}
```

17. Εκτέλεση της ανάθεσης στη μεταβλητή y2:

```
{
 if (0 <= var_y2.lval < MAXMEM) *var_y2.lval = expr_101/(x-33*j)+2.val;
 else ERROR("Illegal memory access");
}
```

όπου οι διαφορετικές εμφανίσεις του ίδιου συμβόλου σε έναν κανόνα έχουν δεικτοδοτηθεί κατάλληλα, ανάλογα με την υποέκφραση στην οποία αντιστοιχούν, ώστε να διαχωρίζονται οι τιμές των κατηγορημάτων τους και να ικανοποιούνται οι εξαρτήσεις μεταξύ αυτών.

Άσκηση 6-2:

Θεωρήστε τη γραμματική των δηλώσεων μεταβλητών κάποιας γλώσσας προγραμματισμού:

```
var_decl → type ID dims ';'
type → KEY_INT | KEY_FLOAT | ID
dims → dims '[' ICONST ']' | ε
```

όπου KEY_INT και KEY_FLOAT οι λεκτικές μονάδες των αντίστοιχων λέξεων-κλειδιά που παριστάνουν τους δύο βασικούς τύπους της γλώσσας, ακεραίων και πραγματικών αντίστοιχα, ID η λεκτική μονάδα των αναγνωριστικών, και ICONST η λεκτική μονάδα των ακέραιων σταθερών της γλώσσας.
Το σύστημα τύπων της γλώσσας περιλαμβάνει τους δύο βασικούς και έναν αριθμό από σύνθετους τύπους. Ανάμεσα στους σύνθετους τύπους ορίζεται και ένας τύπος πίνακα, πολυδιάστατος, με στοιχεία οποιουδήποτε τύπου. Οι σύνθετοι τύποι της γλώσσας μπορούν να ονομάζονται, με δήλωση άλλης μορφής που δε θα μας απασχολήσει εδώ. Ας υποθέσουμε ότι η σημασιολογική περιγραφή των δηλώσεων μεταβλητών της γλώσσας περιλαμβάνει τους εξής περιορισμούς:

(α) Αν χρησιμοποιείται αναγνωριστικό τύπου για τη δήλωση του τύπου της μεταβλητής ή των στοιχείων του πίνακα, αυτό πρέπει να έχει δηλωθεί νωρίτερα.
(β) Ο αριθμός διαστάσεων που δηλώνονται δε μπορεί να ξεπερνάει το 5.
(γ) Το μέγεθος του πίνακα σε κάθε διάσταση καθορίζεται με μια μη προσημασμένη ακέραια σταθερά που δεν πρέπει να έχει μηδενική τιμή.
(δ) Το όνομα της μεταβλητής δεν πρέπει να είναι ήδη δηλωμένο.

Για την αποθήκευση των τύπων των δηλωμένων ονομάτων υπάρχει ένας πίνακας συμβόλων (ΠΣ), κατάλληλα οργανωμένος, ώστε να περιέχει όση πληροφορία είναι αναγκαία για τη σημασιολογική ανάλυση. Έτσι, κάθε μεταβλητή που δηλώνεται με τον πιο πάνω τρόπο πρέπει να εισάγεται στον ΠΣ. Επίσης, κάθε όνομα τύπου πρέπει να είναι αποθηκευμένο στον ΠΣ, με κατάλληλη ένδειξη που να το διαχωρίζει από όνομα μεταβλητής.

Α. Ορίστε κατάλληλα κατηγορήματα στη γραμματική και δώστε ρουτίνες σημασιολογικής ανάλυσης που τα αποτιμούν, επαληθεύοντας τους παραπάνω σημασιολογικούς περιορισμούς, χρησιμοποιώντας όπου χρειάζεται τον ΠΣ.

Β. Δώστε τη σειρά αποτίμησης των κατηγορημάτων για τη συμβολοσειρά:

```
floatrec x [ 10 ] [ 100 ] [ 10 ]
```

Απάντηση

Α. Η γραμματική που μας δίνεται καλύπτει ένα μικρό μόνο μέρος της γραμματικής μιας γλώσσας προγραμματισμού υψηλού επιπέδου. Είναι αυτονόητο ότι σε πλήρη ανάπτυξη θα υπάρχουν πολύ περισσότεροι κανόνες, ενώ θα απαιτούνται πολύ περισσότερα κατηγορήματα από όσα θα ορίσουμε σε αυτή την άσκηση. Το ίδιο ισχύει και για τα πεδία των καταχωρήσεων του ΠΣ. Ας σκεφτούμε μόνο ότι η άσκηση δεν αναφέρεται καθόλου σε παραγωγή ενδιάμεσου κώδικα, και στην πράξη πολλά κατηγορήματα και πεδία του ΠΣ ορίζονται μόνο για

αυτήν. Έτσι, τόσο στα κατηγορήματα, όσο και στα πεδία των καταχωρήσεων του ΠΣ, θα επικεντρωθούμε μόνο στα δεδομένα και ζητούμενα της άσκησης.

Ας ορίσουμε κατ' αρχήν κάποια κατηγορήματα για τα σύμβολα της γραμματικής μας. Ο ορισμός αυτός δεν είναι μοναδικός, αλλά πρέπει να καλύπτει όλες τις ανάγκες της άσκησης.
Το μη τερματικό σύμβολο type απεικονίζει τους τύπους της γλώσσας, κι επομένως τα κατηγορήματά του θα πρέπει να κωδικοποιούν τους τύπους αυτούς. Επειδή το σύστημα τύπων περιλαμβάνει και ένα σύνθετο τύπο, δεν αρκεί ένας ακέραιος κωδικός για το σκοπό αυτό, αλλά απαιτείται και η κωδικοποίηση μέσω δομής τύπου. Ας υποθέσουμε ότι το σύμβολο type έχει δύο συντιθέμενα κατηγορήματα, το ένα – έστω t_code – θα δίνει τον κωδικό τύπου, και το άλλο – έστω t_struct – θα δίνει τη δομή τύπου. Έστω ότι οι κωδικοί των βασικών τύπων είναι οι σταθερές T_INT και T_FLOAT, ενώ ο κωδικός του σύνθετου τύπου πίνακα είναι η σταθερά T_ARRAY. Μια δομή αναπαράστασης τύπου για πίνακες θα πρέπει να περιέχει πληροφορίες για τον τύπο των στοιχείων και τις διαστάσεις του πίνακα, και να κατασκευάζεται με κατάλληλη συνάρτηση. Μια τέτοια συνάρτηση θα μπορούσε να είναι η create_array(), η οποία να δέχεται ως παραμέτρους τον κωδικό και τη δομή τύπου του στοιχείου, καθώς και το πλήθος και τα μεγέθη των διαστάσεων του πίνακα, και να επιστρέφει ένα δείκτη σε μια νέα δομή σύνθετου τύπου πίνακα. Σύμφωνα με τα παραπάνω, τα κατηγορήματα t_code και t_struct θα είναι ακέραιου τύπου και τύπου δείκτη σε δομή, αντίστοιχα.
Το μη τερματικό σύμβολο dims απεικονίζει τις διαστάσεις ενός τύπου πίνακα – παριστάνοντας βαθμωτούς τύπους με μηδέν διαστάσεις. Έτσι, θα πρέπει να έχει δύο κατηγορήματα, το πρώτο – έστω t_dims – θα δίνει το τρέχον πλήθος των διαστάσεων του πίνακα, και το δεύτερο – έστω t_size – θα δίνει το συνολικό πλήθος διαστάσεων και το μέγεθος ανά διάσταση. Επειδή το σύμβολο dims εμφανίζεται στη γραμματική με άμεση αριστερή αναδρομή, η τιμή του πρώτου κατηγορήματός του θα πρέπει να μεταφέρεται από το αριστερό μέλος προς το δεξί, ώστε πηγαίνοντας σε επόμενες διαστάσεις να έχουμε πάντα το σωστό πλήθος διαστάσεων. Επομένως το t_dims θα πρέπει να είναι κληρονομούμενο κατηγόρημα ακέραιου τύπου, ενώ το t_size θα πρέπει να είναι συντιθέμενο κατηγόρημα, κατάλληλου τύπου διανύσματος ακεραίων – έστω με τη μορφή δομής δύο πεδίων, με το πεδίο vector να είναι δείκτης σε μονοδιάστατο πίνακα ακεραίων και το πεδίο length να είναι το μήκος, δηλαδή το πλήθος των στοιχείων του διανύσματος.
Έστω ότι το τερματικό σύμβολο ICONST έχει ένα κατηγόρημα – έστω val – που δίνει την τιμή της σταθεράς, ενώ το τερματικό σύμβολο ID έχει ένα κατηγόρημα – έστω name – που δίνει το όνομα του αναγνωριστικού. Και τα δύο αυτά κατηγορήματα είναι συντιθέμενα που λαμβάνουν τιμή από κάποιο ΛΑ, ενώ ο τύπος τους θα είναι ακέραιος και ορμαθός χαρακτήρων (string), αντίστοιχα.
Το μη τερματικό σύμβολο var_decl δε χρειάζεται να έχει κατηγορήματα, αφού από τη μια δεν παρέχει τιμές σε κληρονομούμενα κατηγορήματα άλλων συμβόλων, ενώ από την άλλη το αποτέλεσμα της δήλωσης εισάγεται στον ΠΣ και δε μεταφέρεται ως συντιθέμενο κατηγόρημα σε άλλο σύμβολο. Σε μια τέτοια περίπτωση, και για να λάβουμε σωστή σειρά στην εκτέλεση των σημασιολογικών ρουτινών, θα θεωρήσουμε την ίδια τη σημασιολογική ρουτίνα που συνοδεύει το σύμβολο – αν υπάρχει – ως ανώνυμο συντιθέμενο κατηγόρημά της.

Όσον αφορά τον ΠΣ, κάθε καταχώρησή του θα περιέχει τουλάχιστον τρία πεδία που να αποθηκεύουν πληροφορία για τον τύπο του αναγνωριστικού, τα δύο από τα οποία για απλούστευση θα θεωρήσουμε ότι έχουν το ίδιο όνομα και τύπο με τα δύο κατηγορήματα του συμβόλου type, ενώ το τρίτο – έστω t_typename ακέραιου τύπου – θα δίνει την πληροφορία αν το αναγνωριστικό είναι όνομα τύπου ή όχι, με τιμές διάφορη του μηδενός ή μηδέν, αντίστοιχα. Θεωρήστε ότι διαθέτουμε κάποια συνάρτηση αναζήτησης του ΠΣ – έστω lookup() – που δεδομένου ενός ονόματος, μας επιστρέφει κάποιο δείκτη σε καταχώρηση του ΠΣ, μηδενικής τιμής αν το αναγνωριστικό δεν έχει οριστεί νωρίτερα. Ακόμα, διαθέτουμε και κάποια συνάρτηση εισαγωγής στον ΠΣ – έστω insert() – που δεδομένου κάποιου ονόματος, το εισάγει στον ΠΣ, επιστρέφοντας ένα δείκτη στην αντίστοιχη καταχώρηση.

Στη συνέχεια θα ξαναδώσουμε τους κανόνες της γραμματικής, δεικτοδοτώντας πολλαπλές εμφανίσεις του ίδιου συμβόλου στον ίδιο κανόνα, και σημειώνοντας σε αυτούς τα σημεία στα οποία θα πρέπει να εκτελείται κάποια ρουτίνα σημασιολογικής ανάλυσης:

```
var_decl → type ID (1) dims ';' (2)
type → KEY_INT (3) | KEY_FLOAT (4) | ID (5)
dims → (6) dims1 '[' ICONST ']' (7) | ε (8)
```

Η τοποθέτηση των ρουτινών στους κανόνες γίνεται ανάλογα με τα κατηγορήματα των συμβόλων που αποτιμώνται. Έτσι, οι ρουτίνες που αποτιμούν κληρονομούμενα κατηγορήματα είναι τοποθετημένες πριν από τα σύμβολα δεξιού μέλους που αφορούν, ενώ αυτές που αποτιμούν συντιθέμενα κατηγορήματα είναι τοποθετημένες στο τέλος των αντίστοιχων κανόνων, αφού αφορούν σύμβολα αριστερού μέλους. Πιο συγκεκριμένα, οι ρουτίνες (1) και (6) δίνουν τιμή στο κληρονομούμενο κατηγόρημα t_dims του συμβόλου dims, οι ρουτίνες (3), (4) και (5) δίνουν τιμή στα συντιθέμενα κατηγορήματα t_code και t_struct του συμβόλου type, ενώ οι ρουτίνες (7) και (8) δίνουν τιμή στο συντιθέμενο κατηγόρημα t_size του συμβόλου dims. Εκτός από αποτίμηση κατηγορημάτων, οι ρουτίνες αυτές εκτελούν και σημασιολογικό έλεγχο, καθώς και διαχείριση του ΠΣ. Ειδικότερα, η ρουτίνα (2) εκτελεί μόνο σημασιολογικό έλεγχο και διαχείριση του ΠΣ.
Η αρίθμηση που κάναμε στις ρουτίνες *δε* σχετίζεται με τη σειρά εκτέλεσής τους. Η σειρά εκτέλεσης καθορίζεται από τις εξαρτήσεις μεταξύ κατηγορημάτων, όπως αυτές προκύπτουν από τις ρουτίνες και εμφανίζονται στο δέντρο συντακτικής ανάλυσης (ΔΣΑ) για τη συγκεκριμένη συμβολοσειρά που αναλύεται. Αν η γραμματική μας είναι L-κατηγορική, οπότε η μεταφορά πληροφορίας για την αποτίμηση των κατηγορημάτων ενός κανόνα γίνεται πάντα από αριστερά προς τα δεξιά, τότε η εκτέλεση των ρουτινών μπορεί να γίνει κατά τη συντακτική ανάλυση, χωρίς να απαιτείται προηγούμενη κατασκευή του ΔΣΑ. Μ' άλλα λόγια, σε ένα ΣΑ που προχωράει από αριστερά προς τα δεξιά, όπως είναι όλοι οι ΣΑ που μελετήσαμε, η εκτέλεση των ρουτινών θα γίνεται όταν τις συναντούμε στους κανόνες που αναλύονται. Στη πιο γενική περίπτωση γραμματικής που δεν είναι L-κατηγορική, μπορεί να υπάρχει εξάρτηση κατηγορημάτων από δεξιά προς τα αριστερά, κι επομένως οι ρουτίνες δεν εκτελούνται πάντα με σειρά από αριστερά προς τα δεξιά. Έτσι, ένας ΣΑ που προχωράει από αριστερά προς τα δεξιά δε μπορεί να κάνει ταυτόχρονα και αποτίμηση κατηγορημάτων, και θα πρέπει πρώτα να κατασκευάζεται το ΔΣΑ για τη συμβολοσειρά που αναλύεται, και ύστερα να γίνεται η εκτέλεση των ρουτινών, με κατάλληλη διαπέραση του ΔΣΑ, με σειρά επίσκεψης κόμβων που καθορίζεται από τις εξαρτήσεις μεταξύ των κατηγορημάτων.
Στη γραμματική που έχουμε, μεταφορά πληροφορίας μέσα στον ίδιο κανόνα μπορεί να γίνει μόνο μεταξύ των ρουτινών (1) και (2), καθώς και μεταξύ των ρουτινών (6) και (7). Όλες οι υπόλοιπες μεταφορές πληροφορίας γίνονται είτε από πάνω προς τα κάτω για κληρονομούμενα, είτε από κάτω προς τα επάνω για συντιθέμενα κατηγορήματα. Σε καμία από τις δύο παραπάνω περιπτώσεις το κληρονομούμενο κατηγόρημα t_dims δεν εξαρτάται από κάποια τιμή κατηγορήματος που παράγεται δεξιότερα από τους αντίστοιχους κανόνες (1) και (6) που το αποτιμούν. Έτσι, η γραμματική μας *είναι* L-κατηγορική, οπότε μπορούμε να γράψουμε τις σημασιολογικές ρουτίνες μαζί με τους συντακτικούς κανόνες, όμως θα προτιμήσουμε να δείξουμε την υλοποίηση με αποτίμηση κατηγορημάτων μέσω του ΔΣΑ.

Ας δούμε τώρα τι πρέπει να κάνει κάθε σημασιολογική ρουτίνα. Επειδή το σύμβολο dims εμφανίζει όπως είδαμε αριστερή αναδρομή, η διαδικασία παραγωγής θα συναντήσει πρώτα την πρώτη διάσταση, μετά τη δεύτερη, κοκ. Έτσι, το κληρονομούμενο κατηγόρημα t_dims θα αρχικοποιείται στην τιμή 0 στη ρουτίνα (1), και στη συνέχεια, στη ρουτίνα (6), θα αυξάνεται κατά 1 όσο γίνεται αναδρομή, μέχρι να φτάσουμε στο τέλος της αναδρομής. Σε κάθε αύξηση θα πρέπει να ελέγχουμε το πλήθος των διαστάσεων σε σχέση με τη μέγιστη τιμή που μας δίνεται. Στο τέλος της αναδρομής, στη ρουτίνα (8), θα αρχικοποιήσουμε την τιμή του συντιθέμενου κατηγορήματος t_size. Επιστρέφοντας προς τα πίσω, στη ρουτίνα (7), θα αποθηκεύουμε το μέγεθος της κάθε διάστασης στον πίνακα t_size.vector. Όσο αφορά το σύμβολο type, οι ρουτίνες (3) και (4) δίνουν κατάλληλη τιμή στο κατηγόρημα t_code. Το κατηγόρημα t_struct δε λαμβάνει τιμή στις δύο αυτές ρουτίνες, μια που ο κωδικός τύπου για τους βασικούς τύπους

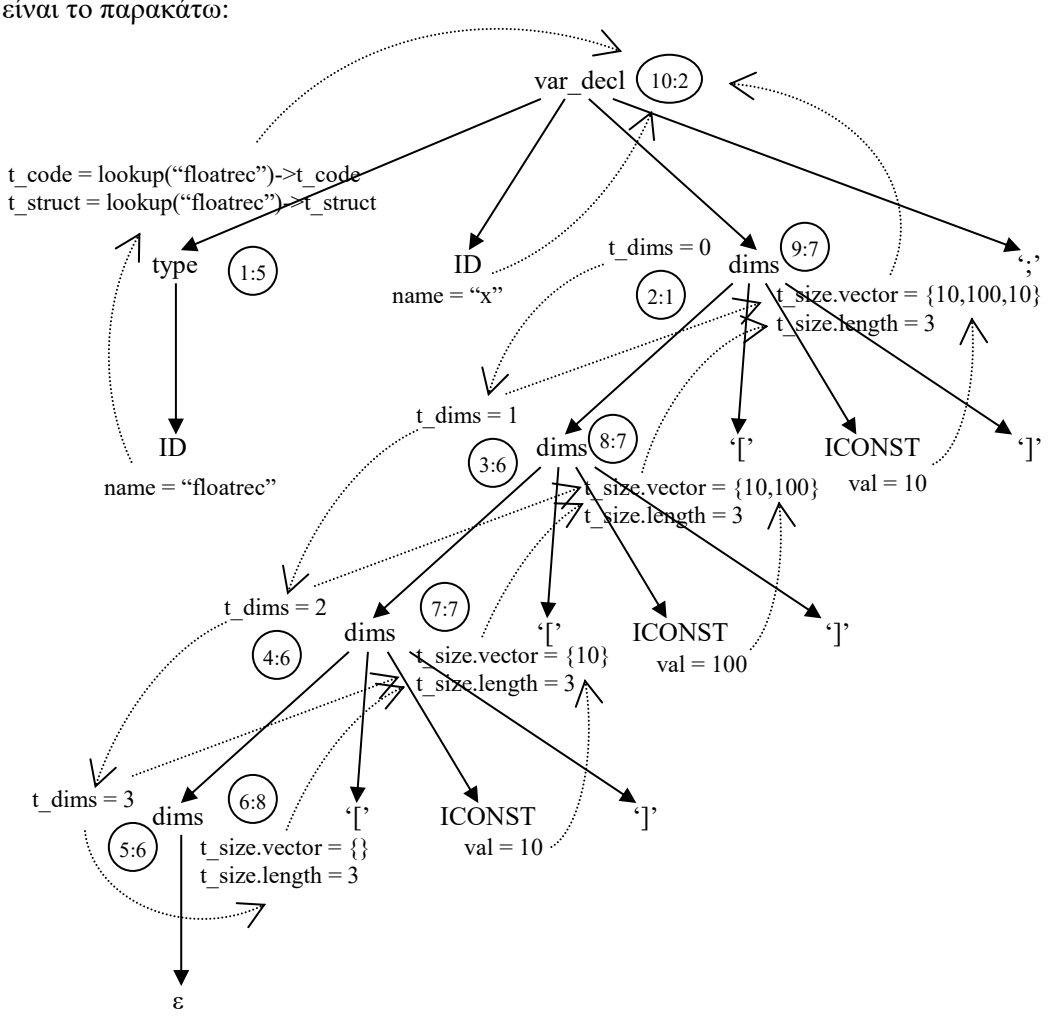
ακεραίων και πραγματικών αρκεί για τον προσδιορισμό τους. Όταν όμως ο τύπος δίνεται από αναγνωριστικό, τότε το κατηγόρημα t_struct θα λάβει τιμή από την αντίστοιχη καταχώρηση του ΠΣ. Έτσι, η ρουτίνα (5) δίνει τιμή τόσο στο κατηγόρημα t_code, όσο και στο κατηγόρημα t_struct, αφού πρώτα επαληθεύσει ότι υπάρχει η σχετική καταχώρηση στον ΠΣ, και μάλιστα ότι αντιστοιχεί σε όνομα τύπου. Τέλος, η ρουτίνα (2) δε συνδέεται με αποτίμηση κατηγορημάτων, αλλά διαχειρίζεται συνολικά τη δήλωση που αναλύθηκε. Έτσι, πρέπει πρώτα να επαληθεύσει ότι *δεν* υπάρχει προηγούμενη δήλωση του ίδιου αναγνωριστικού στον ΠΣ, και στη συνέχεια να δημιουργήσει νέα καταχώρηση σε αυτόν, στα πεδία της οποίας να αντιγράψει τις πληροφορίες που συγκεντρώθηκαν μέσω των συντιθέμενων κατηγορημάτων των συμβόλων type και dims. Ειδικότερα, αν το κατηγόρημα t_size του συμβόλου dims έχει πεδίο length με τιμή 0, τότε δεν έχουμε δήλωση πίνακα, αλλά βαθμωτής μεταβλητής, οπότε οι πληροφορίες τύπου των κατηγορημάτων του συμβόλου type θα αποδοθούν στη νέα μεταβλητή, διαφορετικά θα πρέπει να κατασκευάσουμε μια δομή τύπου πίνακα, με πληροφορίες μεγέθους να λαμβάνονται από τα κατηγορήματα του συμβόλου dims, ενώ οι πληροφορίες τύπου από το σύμβολο type θα αποδοθούν στον τύπο στοιχείου του πίνακα.

Σε μορφή ψευδοκώδικα C, οι σημασιολογικές ρουτίνες θα είναι οι εξής:

```
(1)   dims.t_dims = 0;

(2)   if (lookup(ID.name)) sem_error(1);
      ST_item_type * new_ST_item = insert(ID.name);
      if (dims.t_size.length > 0) {
         type_struct * new_struct = create_array(type.t_code,
            type.t_struct,dims.t_size.length,dims.t_size.vector);
         new_ST_item->t_code = T_ARRAY;
         new_ST_item->t_struct = new_struct;
      } else {
         new_ST_item->t_code = type.t_code;
         new_ST_item->t_struct = type.t_struct;
      }
      new_ST_item->t_typename = 0;

(3)   type.t_code = T_INT;

(4)   type.t_code = T_FLOAT;

(5)   ST_item_type * entry = lookup(ID.name);
      if (!entry) sem_error(2);
      if (!entry->t.typename) sem_error(3);
      type.t_code = entry->t_code;
      type.t_struct = entry->t_struct;

(6)   if (dims.t_dims == 5) sem_error(4);
      dims1.t_dims = dims.t_dims+1;

(7)   if (ICONST.val == 0) sem_error(5);
      dims.t_size.vector = dims1.t_size.vector;
      dims.t_size.vector[dims1.t_size.length-dims1.t_dims] =
         ICONST.val;
      dims.t_size.length = dims1.t_size.length;

(8)   dims.t_size.vector = create_vector(dims.t_dims);
      dims.t_size.length = dims.t_dims;
```

όπου: ST_item_type ο τύπος της καταχώρησης του ΠΣ, type_struct ο τύπος της δομής τύπου, create_vector() κατάλληλη συνάρτηση κατασκευής μονοδιάστατου πίνακα ακεραίων, και sem_error() κατάλληλη συνάρτηση διαχείρισης σημασιολογικών σφαλμάτων, η οποία δέχεται σαν παράμετρο κάποιον κωδικό σφάλματος και εκτυπώνει κατάλληλο μήνυμα, τερματίζοντας τη μετάφραση.

Β. Έχοντας δώσει τις σημασιολογικές ρουτίνες για τη γραμματική που μας δόθηκε, θα δούμε στη συνέχεια πώς εκτελούνται οι ρουτίνες αυτές μέσα από το ΔΣΑ μια συγκεκριμένης συμβολοσειράς. Για το σκοπό αυτό, το ΔΣΑ θα "διακοσμηθεί", ώστε σε κάθε κόμβο να φαίνονται τα κατηγορήματά του, καθώς και οι τιμές τους. Η σειρά εκτέλεσης των ρουτινών θα καθοριστεί από το γράφο εξαρτήσεων μεταξύ κατηγορημάτων, ο οποίος δημιουργείται συνδέοντας με διακεκομμένα κατευθυνόμενα βέλη τα κατηγορήματα μεταξύ των οποίων μεταφέρεται σημασιολογική πληροφορία.
Για τη συμβολοσειρά "floatrec x [10] [100] [10]", το αντίστοιχο διακοσμημένο ΔΣΑ θα είναι το παρακάτω:



Οι αριθμοί σε κύκλο δίνουν τη σειρά εκτέλεσης των σημασιολογικών ρουτινών και τον αριθμό της ρουτίνας που κάθε φορά εκτελείται. Οι ρουτίνες αποτίμησης κληρονομούμενων κατηγορημάτων δείχνονται δίπλα στο σύμβολο δεξιού μέλους που κληρονομεί την τιμή, ενώ οι ρουτίνες αποτίμησης συντιθέμενων κατηγορημάτων δείχνονται δίπλα στο σύμβολο αριστερού μέλους που συνθέτει την τιμή.
Οι εξαρτήσεις μεταξύ των κατηγορημάτων επιβάλλουν τη σειρά από 2-10, ενώ η ρουτίνα 5 που παρουσιάζεται πρώτη στη σειρά μπορεί στην πραγματικότητα να εκτελεστεί οποτεδήποτε πριν την 2, η οποία πρέπει να εκτελεστεί δέκατη. Για παράδειγμα, η ρουτίνα 7 της όγδοης εκτέλεσης χρησιμοποιεί το κληρονομούμενο κατηγόρημα της ρουτίνας 6 της τέταρτης εκτέλεσης, το συντιθέμενο κατηγόρημα της ρουτίνας 7 της έβδομης εκτέλεσης, και το συντιθέμενο κατηγόρημα του αντίστοιχου τερματικού συμβόλου που παρέχει ο ΛΑ. Η ρουτίνα 5 που δεν έχει κάποια εξάρτηση με τις παραπάνω θα μπορούσε να εκτελεστεί πριν από αυτές, μετά από αυτές, ή και ανάμεσά τους.