Θέμα 2

Απόδειξη πως η δοσμένη γραμματική είναι LL(1):

Ισχύουν τα εξής:

LOOKAHEAD(Y🡪a) = {a}

LOOKAHEAD(Y🡪b) = {b}

LOOKAHEAD(Y🡪G) = {(}

Συνεπώς:

LOOKAHEAD(Y🡪a) **∩** LOKAHEAD(Y🡪b) **∩** LOOKAHEAD(Y🡪G) = ∅

Καθώς και:

LOOKAHEAD(Z🡪\*M) = {\*}

LOOKAHEAD(Z🡪-M) = {-}

LOOKAHEAD(Z🡪+M) = {+}

LOOKAHEAD(Z🡪ε) = {)}

LOOKAHEAD(Z🡪\*M) **∩** LOOKAHEAD(Z🡪-M) **∩** LOOKAHEAD(Z🡪+M) **∩** LOOKAHEAD(Z🡪ε) = ∅

Άρα απεδείχθη ότι η δοσμένη γραμματική είναι LL(1).

Ακολουθεί ένας στοιχειώδης ψευδοκώδικας (όχι σε κάποια συγκεκριμένη γλώσσα προγραμματισμού) που επεξηγεί τον υλοποιηθέντα αλγόριθμο, όπως ζητήθηκε:

START

READ (EXPRESSION)

WHILE (EXPRESSION NOT TERMINAL):

READ (EXPRESSION)

IF (ANY SYMBOL FROM EXPRESSION NOT INCLUDED IN GIVEN GRAMMAR):

EXPRESSION NOT RECOGNISED

LEFT = LEFTMOST SYMBOL OF EXPRESSION

RIGHT = RIGHTMOST SYMBOL OF STACK

ADD “G” TO TREE AS ITS ROOT

WHILE (STACK NOT EMPTY) AND (EXPRESSION NOT EMPTY):

IF (RIGHT IS TERMINAL):

IF (LEFT == RIGHT) AND (EXPRESSION LENGTH > 1) AND (STACK LENGTH > 1):

REMOVE RIGHT FROM STACK

REMOVE LEFT FROM EXPRESSION

ELSE IF (LEFT == RIGHT) AND (EXPRESSION LENGTH == 1):

IF (STACK LENGTH == 1):

REMOVE RIGHT FROM STACK

REMOVE LEFT FROM EXPRESSION

EXPRESSION RECOGNISED

ELSE:

EXPRESSION NOT RECOGNISED

ELSE:

WHILE (RULE NOT FOUND):

SEEK RULE

IF (RULE FOUND):

ADD LEFT PART OF RULE TO TREE

IF (LEFT PART OF RULE NOT “ε”):

REVERSE LEFT PART OF RULE AND CHANGE RIGHT LETTER OF STACK WITH IT

IF (RULE WAS FOUND):

PRINT (“NO PRODUCTION RULE FOUND’)

BREAK

IF (EXPRESSION WAS RECOGNISED):

PRINT (“EXPRESSION RECOGNISED”)

PRINT (SYNTACTICAL TREE) IN PREORDER

ELSE:

PRINT (“EXPRESSION NOT RECOGNISED”)

END

Ανάλυση του κώδικα:

Πριν ξεκινήσουμε την τεκμηρίωση και την ανάλυση του κώδικα αυτού καθ' αυτού είναι απαραίτητο να αναφέρουμε τις παραδοχές και τις συμβάσεις που έχουμε κάνει, για την υλοποίηση του προγράμματος. Αυτές είναι: (α) Δεχόμαστε, αρχικά, πως οι συμβολοσειρές εισόδου που μπορεί να χειριστεί ο συντακτικός αναλυτής μας είναι τερματικές. (β) Δεχόμαστε επίσης, ότι το σύμβολο G είναι το αρχικό σύμβολο της γραμματικής μας. (γ) Έπειτα, δεχόμαστε και ότι ο χρήστης δεν βάζει στο τέλος της συμβολοσειράς εισόδου το σύμβολο $ (που διαφορετικά θα δήλωνε το πέρας της συμβολοσειράς). Αυτό πρακτικά σημαίνει ότι μια κατά τα άλλα έγκυρη συμβολοσειρά δεν θα αναγνωριστεί εάν περιέχει αυτό το σύμβολο. Οι βασικότεροι λόγοι που κάναμε αυτοί τη σύμβαση είναι επειδή, πρώτον, είναι πιο απλό για τον χρήστη και δεύτερον απλοποιεί ως έναν βαθμό το πρόγραμμα που φτιάξαμε για την υλοποίηση του αλγορίθμου του συντακτικού αναλυτή. Τελευταία σύμβαση είναι πως μέσα στο πρόγραμμα δηλώνουμε την κενή συμβολοσειρά «ε» με αγγλικό «e» αντί για ελληνικό, καθώς το ελληνικό σύμβολο θα μας δημιουργούσε πρόβλημα λόγω της διαφορετικής κωδικοποίησής του από τα αγγλικά σύμβολα. Αυτή η αλλαγή που κάναμε δεν επηρεάζει το πρόγραμμα από οποιαδήποτε άλλη άποψη και λειτουργεί το ίδιο.

Ξεκινώντας λοιπόν από την αρχή του προγράμματος έχουμε την υλοποίηση της συνάρτησης reverse, την οποία θα αναλύσουμε όταν φτάσουμε στο σημείο κλήσης της. Στην main συνάρτηση του προγράμματος έχουμε την δήλωση των απαραίτητων μεταβλητών του προγράμματος, με σημαντικότερες αυτών το array Grammar, την expr και την stack. Το array Grammar αποτελεί, ουσιαστικά, τον προβλέπον συντακτικό πίνακα της γραμματικής, με κάθε πρώτο στοιχείο του κάθε υπο-πίνακα να αποτελεί τη γραμμή του πίνακα ( δηλαδή το στοιχείο στο αριστερό κομμάτι κάθε κανόνα παραγωγής), το κάθε δεύτερο στοιχείο είναι το δεξί κομμάτι του αντίστοιχου κανόνα παραγωγής και το κάθε τρίτο στοιχείο να αποτελεί τη στήλη του πίνακα. Η μεταβλητή expr αποθηκεύει τη συμβολοσειρά εισόδου από το χρήστη και η μεταβλητή stack είναι οι στοίβα, που αρχικοποιείται με G ( εφόσον αυτό είναι το αρχικό σύμβολο, όπως αναφέραμε και στη παραδοχή (α) ). Αρχικά, ζητάμε από τον χρήστη τη συμβολοσειρά εισόδου και ταυτόχρονα ελέγχουμε αν αυτή είναι τερματική, υλοποιώντας έτσι την παραδοχή (β). Έπειτα, και πριν ξεκινήσουμε την αυστηρή υλοποίηση του αλγορίθμου, ελέγχουμε αν υπάρχουν τερματικά σύμβολα στη συμβολοσειρά εισόδου που δεν υπάρχουν στην γραμματική. Αν ναι τότε δεν χρησιμοποιούμε τον αλγόριθμο, καθώς θα ήταν περιττό και η συμβολοσειρά δεν αναγνωρίζεται. Για τους δύο παραπάνω ελέγχους χρησιμοποιούμε την θέση του κάθε στοιχείου της συμβολοσειράς στον πίνακα ASCII.

Στην υλοποίηση του αλγορίθμου της συντακτικής ανάλυσης top-bottom, έχουμε ένα εξωτερικό loop που συνεχίζει τη διαδικασία της περιπτωσιολογίας, όσο η στοίβα και η συμβολοσειρά εισόδου έχουν στοιχεία. «Κρατάμε» το δεξιότερο στοιχείο της στοίβας (right) και το αριστερότερο της συμβολοσειράς (left) και διακρίνουμε δυο περιπτώσεις, με την καθεμία τους να έχει ξεχωριστές υποπεριπτώσεις: To right να είναι τερματικό και από εκεί αν είναι ίδιο με το left και ταυτόχρονα η έκφραση δεν είναι άδεια χωρίς το left ( συνθήκη ισοδυναμεί με αυτή που στις διαφάνειες και το βιβλίο αναφέρεται ως: right = left και left **≠** $**)** ή αν τα right και left είναι ίδια και ταυτόχρονα το left είναι το τελευταίο στοιχείο της συμβολοσειράς ( συνθήκη ισοδυναμεί με αυτή που στις διαφάνειες και το βιβλίο αναφέρεται ως: right = left =$**)**. Για να λειτουργήσει σωστά η τελευταία υποπερίπτωση εισαγάγαμε άλλο ένα if statement που διαχωρίζει την περίπτωση που το right είναι το τελευταίο στοιχείο της στοίβας (όπου προφανώς η συμβολοσειρά αναγνωρίστηκε επιτυχώς και θέτουμε to flag ως true, για την εκτύπωση του συντακτικού δένδρου) και την περίπτωση που, ενώ η μεταβλητή expr είναι κενή, η μεταβλητή stack έχει και άλλα στοιχεία μέσα (εδώ θέτουμε απευθείας τις expr και stack ίσες με ένα άδειο string για να τερματίσουμε αναγκαστικά το loop). Με άλλα λόγια εδώ συμπεραίνουμε ότι εδώ η συμβολοσειρά δεν αναγνωρίστηκε. Αν το right δεν είναι τερματικό τότε αναζητάμε τον κατάλληλο κανόνα παραγωγής στο array Grammar και αν τον βρούμε τότε βάζουμε το δεύτερο στοιχείο του κατάλληλου υπο-πίνακα αντεστραμμένο στο stack, εκτός και αν φυσικά παράγει το ε, δηλαδή την κενή συμβολοσειρά. Η αντιστροφή γίνεται με την κλήση της συνάρτησης reverse που δημιουργήσαμε αποκλειστικά γι’ αυτόν τον σκοπό, αφού αντιστρέφει κάθε φορά το δεξιότερο στοιχείο της μεταβλητής με το αριστερότερο, μέχρι να φτάσει στη μέση της λέξης. Σε περίπτωση που δεν βρούμε κατάλληλο κανόνα παραγωγής κάνουμε κατευθείαν brake από το loop του αλγορίθμου (εκτυπώνοντας παράλληλα κατάλληλο μήνυμα), ο οποίος περιείχε την περιπτωσιολογία. Τέλος, και εκτός του loop, ελέγχουμε αν η συμβολοσειρά έχει τελικά αναγνωριστεί ή όχι και σε κάθε περίπτωση εκτυπώνουμε κατάλληλο μήνυμα. Η συνθήκη του if statement αυτού αποτελείται από μια και μοναδική boolean μεταβλητή, την flag, η οποία γίνεται τελικά true αν και μόνο αν η συμβολοσειρά αναγνωριστεί επιτυχώς, εντός της δεύτερης εκ των τεσσάρων υποπεριπτώσεων που αναφέραμε.

Όσον αφορά την εκτύπωση, για την κατασκευή του δέντρου χρησιμοποιείται ένα singly linked list που υλοποιείται με μια κλάση Node. Κάθε αντικείμενο της κλάσης αυτής έχει μια μεταβλητή string που έχει το data του κάθε κλειδιού, ένα pointer στο διπλανό του αδελφό και ένα άλλο σε ένα singly linked list που έχει αποθηκευμένα τα παιδιά του κλειδιού αυτού. Το δέντρο αρχικοποιείται με το node "G" στην ρίζα του και κάθε φορά που γίνεται αντικατάσταση στο stack, καλείται μια συνάρτηση IsNewTerminal, που παίρνει ως παράμετρο την ρίζα και επιστρέφει το node που περιέχει το αμέσως επόμενο μη τερματικό σύμβολο που δεν έχει παιδιά (δεν έχει αναλυθεί δηλαδή), το οποίο είναι και το στοιχείο που πρόκειται να αντικατασταθεί στο stack. Έπειτα προσθέτει για κάθε χαρακτήρα του string "Grammar[i][1]" προσθέτει ένα παιδί στην λίστα children του προαναφερόμενου node με data το κάθε ξεχωριστό χαρακτήρα. Στο τέλος, αναλόγως με το αν το flag είναι true ή false εκτυπώνεται το δέντρο σε προδιάταξη.

Ακολουθεί screenshot από την εκτέλεση το προγράμματος για τη συμβολοσειρά: ((a-b)\*(a+b))

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα