ΚΕΦΑΛΑΙΟ 6 ΣΥΝΤΑΚΤΙΚΑ ΚΑΤΕΥΘΥΝΟΜΕΝΗ ΜΕΤΑΦΡΑΣΗ

Στόχος

Στόχος του Κεφαλαίου αυτού είναι να μελετήσουμε την δημιουργία ενδιάμεσου κώδικα, με την βοήθεια του σχήματος Συντακτικά Κατευθυνόμενης Μετάφρασης, για τις διάφορες συντακτικές δομές μιας γλώσσας προγραμματισμού.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει το Κεφάλαιο αυτό θα μπορείτε να

- Εξηγήσετε τι είναι ένα Συντακτικά Κατευθυνόμενο Μεταφραστικό σχήμα,
- Περιγράψετε τις τρεις διαφορετικές μορφές ενδιάμεσου κώδικα,
- Περιγράψετε την κατασκευή κώδικα σε μορφή Postfix, συντακτικού δένδρου και τετράδων,
- Κατασκευάσετε μεταφραστικό σχήμα για αριθμητικές εκφράσεις με απλούς και μεικτούς τύπους δεδομένων,
- Περιγράψετε τους δυο διαφορετικούς τρόπους αναπαράστασης των Boolean εκφράσεων,
- Κατασκευάσετε μεταφραστικό σχήμα για δομημένες εντολές τύπου for, while, if-then-else κλπ για ένα Bottom up αναλυτή,
- Κατασκευάσετε μεταφραστικό σχήμα για δομημένες εντολές τύπου for, while, if-then-else κλπ για ένα Top Down αναλυτή,

Έννοιες-Κλειδιά

Συντακτικά Κατευθυνόμενη Μετάφραση, μεταφραστικά πεδία, σημασιολογική δράση, κώδικας τριών διευθύνσεων, τετράδες, τριάδες, postfix, συντακτικό δένδρο, οπισθομπάλωση κώδικα, μαρκαδόροι.

Εισαγωγικές παρατηρήσεις

Για τη δημιουργία ενδιάμεσου κώδικα χρησιμοποιείται μία τεχνική που καλείται "σχήμα για συντακτικά κατευθυνόμενη μετάφραση" (ΣΚΜ) και η οποία επιτρέπει να επισυνάπτονται ρουτίνες (σημασιολογικές δράσεις) στους κανόνες μιας context-free γραμματικής. Η χρησιμότητα του ευρίσκεται στο ότι επιτρέπει στον σχεδιαστή ενός μεταγλωττιστή να εκφράζει την δημιουργία του ενδιάμεσου κώδικα κατευθείαν από την συντακτική δομή της γλώσσας.

Θα δώσουμε έμφαση σε μία μορφή ενδιάμεσου κώδικα που καλείται "κώδικας τριών διευθύνσεων" και σε μία συγκεκριμένη υλοποίηση αυτού που καλείται "τετράδες" (guadruples).

Θα μελετήσουμε την σχεδίαση και υλοποίηση σημασιολογικών δράσεων στα πλαίσια της Συντακτικά Κατευθυνόμενης Μετάφρασης για την μετάφραση γλωσσικών δομών

όπως αριθμητικές εκφράσεις, λογικές εκφράσεις, εκχωρήσεις και εντολές σε ενδιάμεσο κώδικα τριών διευθύνσεων τόσο για Bottom up αναλυτές όσο και για Top Down αναλυτές.

ΕΝΟΤΗΤΑ 6.1 ΣΗΜΑΣΙΟΛΟΓΙΚΕΣ ΡΟΥΤΙΝΕΣ

Μια Συντακτικά Κατευθυνόμενη Μετάφραση (ΣΚΜ) είναι μία γραμματική χωρίς συμφραζόμενα (context-free) στην οποία κάποια ρουτίνα ή κομμάτι προγράμματος συσχετίζεται με κάθε γραμματικό κανόνα. Π.χ. αν "α" είναι κάποια δράση (action ή semantic action) που συσχετίζεται με τον κανόνα A -> XYZ, τότε η δράση "α" εκτελείται κάθε φορά που ο συντακτικός αναλυτής αναγνωρίζει στην είσοδό του μια συμβολοσειρά w η οποία έχει μία παραγωγή της μορφής A => XYZ =>* w. Σ' ένα bottom-up αναλυτή, η δράση εφαρμόζεται όταν το XYZ μειώνεται σε A. Σ' ένα top-down αναλυτή μία δράση εφαρμόζεται όταν τα A, X, Y ή Z αναπτύσσονται.

Μια δράση μπορεί να περιλαμβάνει τον υπολογισμό τιμών μεταβλητών που ανήκουν στον μεταγλωττιστή, την δημιουργία ενδιάμεσου κώδικα, την εκτύπωση ενός διαγνωστικού, την εισαγωγή μιας τιμής σε κάποιο πίνακα κ.λ.π., και συνήθως συσχετίζεται με κάποιο κόμβο ενός δένδρου ανίχνευσης.

Μια μεταβλητή η οποία συσχετίζεται μ' ένα γραμματικό σύμβολο ονομάζεται "μετάφραση" (translation) του συμβόλου αυτού. Η μετάφραση αυτή μπορεί να είναι μία δομή με πεδία διαφόρων τύπων. Οι δε κανόνες για τον υπολογισμό της τιμής μιας μετάφρασης μπορούν να είναι όσο πολύπλοκοι επιθυμούμε.

Θα συμβολίζουμε τα "μεταφραστικά πεδία" ενός γραμματικού συμβόλου X με ονόματα X.VAL, X.TRUE, κ.λ.π. Όταν έχουμε ένα κανόνα με αρκετές εμφανίσεις του ίδιου συμβόλου στα δεξιά, θα διακρίνουμε τα σύμβολα με άνω δείκτες. Π.χ.

$E \rightarrow E^{(1)} + E^{(2)}$ {E.VAL:= $E^{(1)}$.VAL + $E^{(2)}$.VAL}

όπου οι δράσεις εμφανίζονται μετά τους κανόνες και μέσα σε αγκύλες. Η παραπάνω μετάφραση ταιριάζει βέβαια σ' ένα desk calculator και όχι σ' ένα μεταγλωττιστή.

6.1.1 ΥΛΟΠΟΙΗΣΗ ΤΩΝ ΣΥΝΤΑΚΤΙΚΑ ΚΑΤΕΥΘΥΝΟΜΕΝΩΝ ΜΕΤΑΦΡΑΣΕΩΝ

Ένα σχήμα Συντακτικά Κατευθυνόμενης Μετάφρασης (ΣΚΜ) είναι μία βολική περιγραφή του τι θα θέλαμε να κάνουμε. Η έξοδος που ορίζεται είναι ανεξάρτητη από το είδος του αναλυτή που χρησιμοποιείται ή το είδος του μηχανισμού που χρησιμοποιείται για τον υπολογισμό των μεταφράσεων.

Ένα άλλο προσόν της μεθόδου είναι η δυνατότητα για εύκολες τροποποιήσεις. Καινούριοι κανόνες και αντίστοιχες ρουτίνες δράσεις μπορούν να προστεθούν χωρίς παρενέργειες στο υπάρχον σύστημα.

Έχοντας γράψει ένα σχήμα για ΣΚΜ, η επόμενη ενέργεια είναι να το μετατρέψουμε σε πρόγραμμα αυτόματα (με κάποιον γεννήτορα αναλυτών) ή όχι. Θα εξετάσουμε όμως πρώτα τι είδους μηχανισμοί μπορούν να χρησιμοποιηθούν για τη υλοποίηση ενός ΣΚΜ.

Για να υπολογίσουμε την μετάφραση σ' ένα κόμβο Α που σχετίζεται με κάποιο κανόνα A -> XYZ, χρειαζόμαστε μόνο τις τιμές των μεταφράσεων που σχετίζονται με τους κόμβους X, Y, και Z. Οι δε κόμβοι αυτοί θα είναι ρίζες υποδένδρων στο δάσος που παριστάνει το μερικώς συμπληρωμένο δένδρο ανίχνευσης (για παραδείγματα και πολύ μεγαλύτερη ανάλυση του κεφαλαίου αυτού κοίταξε το βιβλίο: A. Aho, R. Sethi, J. D. Ullman, Compilers: Principles techniques, and Tools, Addison-Wesley, 1986).

Ένας τρόπος υλοποίησης ενός ΣΚΜ είναι να χρησιμοποιήσουμε και άλλα πεδία στη στοίβα του αναλυτή με τα γραμματικά σύμβολα. Τα πεδία αυτά περιέχουν τις τιμές των αντίστοιχων μεταφράσεων. Ας υποθέσουμε ότι η στοίβα υλοποιείται με δύο arrays STATE και VAL. Κάθε θέση του STATE είναι ένας δείκτης στον πίνακα ανίχνευσης (parsing Table) της γραμματικής. Αν το STATE[i] δείχνει στο σύμβολο Ε, τότε το VAL[i] περιέχει την τιμή της μετάφρασης E.VAL που σχετίζεται με το κόμβο Ε του δένδρου ανίχνευσης. TOP είναι ένας δείκτης για την κορυφή της στοίβας.

Υποθέτουμε ότι οι σημασιολογικές ρουτίνες εκτελούνται πριν από κάθε μείωση. Πριν το XYZ μειωθεί σε Α, η τιμή της μετάφρασης του Z είναι στο VAL[TOP], αυτή του Y στο VAL[TOP+1] και αυτή του X στο VAL{TOP+2]. Μετά την μείωση του XYZ σε Α η τιμή του A.VAL εμφανίζεται στο VAL[TOP].

Παράδειγμα 1 / Κεφ. 6

Θα δούμε πως ένα ΣΚΜ που περιγράφει ένα desk calculator μπορεί να υλοποιηθεί από ένα bottom-up αναλυτή ο οποίος καλεί ρουτίνες για τον υπολογισμό των σημασιολογικών ρουτινών. Ο calculator θα υπολογίζει εκφράσεις ακεραίων αριθμών και των τελεστών + και *. Υποτίθεται ότι η είσοδος τελειώνει με $, και η έξοδος είναι η αριθμητική τιμή της έκφρασης η οποία τυπώνεται μέσω της δράσης {print E.VAL}του Σχήματος 6.1. Για ένα τέτοιο μεταφραστή γράφουμε κατ' αρχήν την γραμματική που περιγράφει την είσοδο.

Οι κανόνες της γραμματικής είναι:

S -> E$
E -> E+E
E -> E*E
E -> (E)
E -> I
I -> I digit
I -> digit

Υποθέτουμε τις γνωστές προτεραιότητες και προσεταιριστικότητες των τελεστών + και *. Τερματικά σύμβολα είναι τα $,+,*,(,), και digit (0,1,…,9). Πρέπει τώρα να προσθέσουμε σημασιολογικές δράσεις στους παραπάνω κανόνες. Με τα μη-τερματικά Ε και Ι συσχετίζουμε μία μετάφραση ακέραιης τιμής E.VAL και I.VAL αντίστοιχα, που προσδιορίζει την αριθμητική τιμή της έκφρασης ή του ακέραιου που παριστάνεται μ' ένα

κόμβο που επιγράφεται E ή I στο δένδρο ανίχνευσης. Με το τερματικό σύμβολο digit συσχετίζουμε την μετάφραση LEXVAL, η οποία υποθέτουμε ότι είναι το δεύτερο μέρος του ζεύγους (digit, LEXVAL) το οποίο επιστρέφεται από τον λεξικό αναλυτή όταν έχει βρεθεί ένα token τύπου digit. Ένα σύνολο από σημασιολογικές δράσεις για την γραμματική του desk calculator είναι αυτή του Σχήματος 6.1

Κανόνας		σημασιολογική δράση
(1)	S -> E$	{print E.VAL}
(2)	E -> $E^{(1)}$+$E^{(2)}$	{E.VAL:=$E^{(1)}$.VAL+$E^{(2)}$.VAL}
(3)	E -> $E^{(1)}$*$E^{(2)}$	{E.VAL:=$E^{(1)}$.VAL*$E^{(2)}$.VAL}
(4)	E -> ($E^{(1)}$)	{E.VAL:=$E^{(1)}$.VAL}
(5)	E -> I	{E.VAL:=I.VAL}
(6)	I -> $I^{(1)}$ digit	{I.VAL:=10*$I^{(1)}$.VAL+LEXVAL}
(7)	I -> digit	{I.VAL:=LEXVAL}

Σχήμα 6.1 Σημασιολογικές δράσεις για την γραμματική του desk calculator

Άσκηση αυτοαξιολόγησης 1 / Κεφ. 6

Σας δίνεται το παραπάνω σχήμα συντακτικά κατευθυνόμενης μετάφρασης και η είσοδος 23*5+4$. Μπορείτε να κατασκευάσετε το δένδρο ανίχνευσης διακοσμημένο στους αντίστοιχους κόμβους με τις αντίστοιχες μεταφράσεις των E.VAL, I.VAL και LEXVAL;

Για την υλοποίηση του σχήματος συντακτικά κατευθυνόμενης μετάφρασης (ΣΚΜ) του desk calculator, χρειάζεται να κατασκευάσουμε ένα λεκτικό αναλυτή και ένα bottom-up συντακτικό αναλυτή τον οποίο θα υποχρεώσουμε να εκτελεί τις αντίστοιχες δράσεις πριν από κάθε μείωση. Υπενθυμίζοντας τις παρατηρήσεις υλοποίησης στο τέλος της υποενότητας 6.1.1 και ότι τον δείκτη TOP που δείχνει στη στοίβα του αναλυτή τον χειρίζεται ο αναλυτής και όχι οι σημασιολογικές δράσεις, παρουσιάζουμε στο Σχήμα 6.2 το ΣΚΜ για τον desk calculator.

Κανόνες	Σημασιολογικές δράσεις
(1) S -> E$	{print VAL [TOP]}
(2) E -> E+E	{VAL[TOP}:=VAL[TOP]+VAL[TOP-2]}
(3) E -> E*E	{VAL[TOP]:=VAL[TOP]*VAL[TOP-2]}
(4) E -> (E)	{VAL [TOP]:=VAL [TOP-1]}
(5) E -> I	{τίποτα}
(6) I -> I d	{VAL[TOP]:=10*VAL[TOP]+LEXVAL}
(7) I -> d	{VAL [TOP]:=LEXVAL}

Σχήμα 6.2 Υλοποίηση του ΣΚΜ για τον desk calculator.

ΕΝΟΤΗΤΑ 6.2 ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

Στόχος

Στόχος της Ενότητας αυτής είναι να μελετήσουμε τις πιο συνηθισμένες μορφές ενδιάμεσου κώδικα που χρησιμοποιούνται σε μεταγλωττιστές και διερμηνευτές.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα αυτή θα μπορείτε να

- Περιγράψετε τις τρεις μορφές ενδιάμεσου κώδικα
- Περιγράψετε πως γίνεται ο υπολογισμός των postfix εκφράσεων
- Περιγράψετε πως δημιουργούμε postfix κώδικα και συντακτικά δένδρα με τη βοήθεια απλών συντακτικά κατευθυνόμενων μεταφραστικών σχημάτων.
- Δώσετε παραδείγματα εντολών τριών διευθύνσεων σε αναπαράσταση τριάδων και τετράδων

Έννοιες-Κλειδιά

Κώδικας τριών διευθύνσεων, τετράδες, τριάδες, postfix, συντακτικό δένδρο.

Εισαγωγικές παρατηρήσεις

Η βασική διαφορά μεταξύ ενδιάμεσου κώδικα και κώδικα Assembly (η μηχανής) είναι ότι στον ενδιάμεσο κώδικα ο καταχωρητής στον οποίον αναφέρεται κάποια πράξη δεν καθορίζεται. Τρία είδη ενδιάμεσου κώδικα συναντώνται στους διάφορους μεταγλωττιστές

- postfix κώδικας,
- συντακτικά δένδρα,
- κώδικας τριών διευθύνσεων (τετράδες, τριάδες)

Θα δούμε μορφές αριθμητικών εκφράσεων και εντολών στα τρία αυτά είδη ενδιάμεσου κώδικα Ακόμη θα δούμε πώς γίνεται ο υπολογισμός των postfix εκφράσεων, και πώς δημιουργούμε postfix κώδικα και συντακτικά δένδρα με τη βοήθεια συντακτικά κατευθυνόμενων μεταφραστικών σχημάτων. Στη συνέχεια θα δούμε την γενική μορφή και τις πιο κοινές μορφές κώδικα τριών διευθύνσεων και πώς αναπαριστώνται υπό την μορφή τετράδων και τριάδων.

6.2.1 POSTFIX ΚΩΔΙΚΑΣ

Αν e_1 και e_2 είναι εκφράσεις σε postfix και $\underline{op}$ κάποιος δυαδικός τελεστής η έκφραση $e_1 \underline{op} e_2$ σε postfix μορφή είναι $e_1 e_2 \underline{op}$. Στις postfix εκφράσεις δεν χρησιμοποιούνται παρενθέσεις διότι η θέση και το πλήθος των έντελων των διαφόρων τελεστών επιτρέπουν μία και μοναδική αποκωδικοποίηση των εκφράσεων αυτών.

Παράδειγμα 2 / Κεφ. 6

(1) η (a+b)*c σε postfix γράφεται ab+c*

(2) η a*(b+c) γράφεται abc+*, και η

(3) (a+b) * (c+d) γράφεται ab+cd+*

(4) Ένας τριαδικός τελεστής είναι η υπό συνθήκη έκφραση if e then x else y. Αν συμβολίσουμε με ? τον τελεστή αυτόν η υπό συνθήκη έκφραση γράφεται: exy?

Άσκηση αυτοαξιολόγησης 2 / Κεφ. 6

Μπορείτε να δώσετε την postfix μορφή της παρακάτω έκφρασης η οποία σας δίνεται σε infix μορφή;

if a then if c-d then a+c else a*c else a+b

Υπολογισμός των Postfix εκφράσεων

Γίνεται χρήση μιας στοίβας στην οποία τοποθετείται κάθε έντελο που συναντάμε καθώς ανιχνεύουμε την postfix έκφραση από αριστερά προς τα δεξιά. Αν συναντήσουμε ένα τελεστή των k έντελων (επιδρά σε k έντελα), το αριστερότερο έντελο είναι k-1 θέσεις κάτω από την κορυφή της στοίβας, και το δεξιότερο (k-στο) στην κορυφή της στοίβας και εν γένει το i-στο ευρίσκεται k-i θέσεις κάτω από την κορυφή της στοίβας. Εφαρμόζουμε τον τελεστή στις k κορυφαίες τιμές της στοίβας. Οι τιμές αυτές εξάγονται από τη στοίβα και το αποτέλεσμα της πράξης του τελεστή τοποθετείται στην κορυφή της στοίβας.

Συντακτικά Κατευθυνόμενη Μετάφραση σε Postfix κώδικα.

Η δημιουργία postfix ενδιάμεσου κώδικα είναι απλή για εκφράσεις και φαίνεται στο παράδειγμα του Σχήματος 6.3. Το E.CODE είναι μετάφραση που παίρνει τιμή string. Η τιμή της μετάφρασης E.CODE για τον πρώτο κανόνα είναι η σύζευξη (concatenation) των δύο μεταφράσεων $E^{(1)}$.CODE και $E^{(2)}$.CODE. Η σύζευξη υποδηλώνεται με το "||".

κανόνας	δράση
E -> E(1) op E(2)	{E.CODE:=E(1).CODE||E(2).CODE||"op"}
E -> (E(1))	{E.CODE:=E(1).CODE}
E -> id	{E.CODE:=id}

Σχήμα 6.3 Δημιουργία postfix ενδιάμεσου κώδικα (γενικό σχήμα)

Η μετάφραση του μη-τερματικού στα αριστερά κάθε κανόνα είναι η σύζευξη των μεταφράσεων των μη-τερματικών στα δεξιά στην ίδια σειρά όπως στον κανόνα, ακολουθούμενη από κάποια επιπλέον συμβολοσειρά (ενδεχομένως την κενή συμβολοσειρά). Η υλοποίηση ενός τέτοιου "απλού postfix" μεταφραστικού σχήματος μπορεί να γίνει χωρίς μεταφραστική στοίβα.

Κανόνας	δράση
E -> E(1) op E(2)	{βγάλε op}
E -> (E(1))	{ }
E -> id	{βγάλε id}

Σχήμα 6.4. Η υλοποίηση ενός "απλού postfix" μεταφραστικού σχήματος

Τα κομμάτια προγράμματος (δράσεις) του Σχήματος 6.4 μπορούν να χρησιμοποιηθούν για το παραπάνω σχήμα. Έτσι όταν μειώνουμε με τον κανόνα E -> id, εξάγουμε τον identifier. Στην μείωση με το E -> (E) δεν εξάγουμε τίποτα, και όταν μειώνουμε με το E -> E op E εξάγουμε τον op. Ενεργώντας έτσι δημιουργούμε το postfix ισοδύναμο μιας infix έκφρασης.

Παράδειγμα 3 / Κεφ 6

Παραδείγματος χάριν, στην επεξεργασία της συμβολοσειράς εισόδου a+b*c, ένας συντακτικά κατευθυνόμενος infix-σε-postfix μεταφραστής, θα έκανε την ακολουθία βημάτων του Σχήματος 6.5**.** Στο παράδειγμα αυτό θεωρούμε τα a, b, c, +, και x σαν λεκτικές τιμές (ανάλογες του LEXVAL) που σχετίζονται με id και op.

1. μετακίνησε το a
2. μείωσε σε E -> id και τύπωσε a
3. μετακίνησε +
4. μετακίνησε b
5. μείωσε E -> id και τύπωσε b
6. μετακίνησε *
7. μετακίνησε c
8. μείωσε E -> id και τύπωσε c
9. μείωσε E -> E op E και τύπωσε *.
10. μείωσε E -> E op E και τύπωσε +.

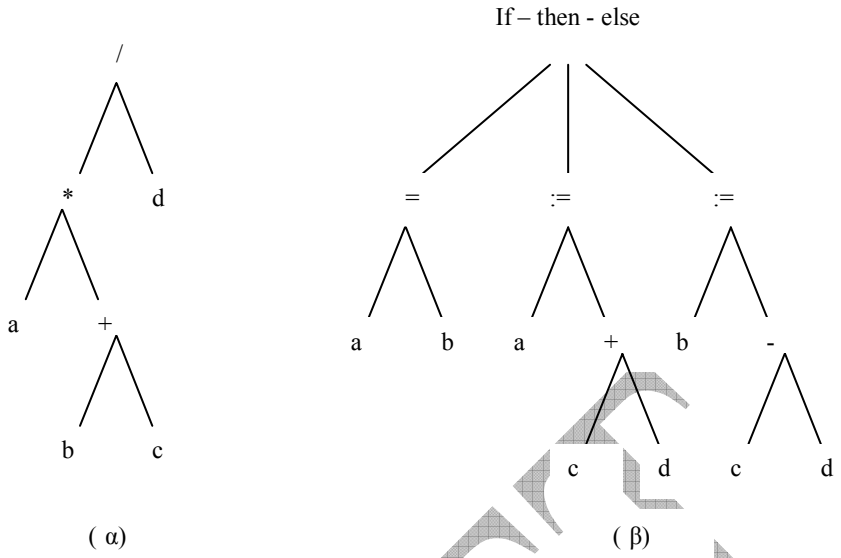
Σχήμα 6.5 Ακολουθία βημάτων infix-σε-postfix μετάφρασης

6.2.2 ΔΕΝΔΡΑ ΑΝΙΧΝΕΥΣΗΣ ΚΑΙ ΣΥΝΤΑΚΤΙΚΑ ΔΕΝΔΡΑ

Ένα Δένδρο Ανίχνευσης συχνά περιέχει πλεονάζουσα πληροφορία η οποία μπορεί να απαλειφθεί δημιουργώντας έτσι μία πιο οικονομική αναπαράσταση του πηγαίου προγράμματος. Μία τέτοια μορφή καλείται Συντακτικό Δένδρο και είναι ένα δένδρο στο οποίο κάθε φύλλο παριστάνει ένα έντελο και κάθε εσωτερικός κόμβος ένα τελεστή.

Παράδειγμα 4 / Κεφ. 6

Το συντακτικό δένδρο της έκφρασης a*(b+c)/d φαίνεται στο Σχήμα 6.6(α) και εκείνο της εντολής: if a=b then a:=c+d else b:=c-d φαίνεται στο 6.6(β).



Σχήμα 6.6 Συντακτικά Δένδρα.

Συντακτικά Κατευθυνόμενη Δημιουργία Συντακτικών Δένδρων

Όπως και στην περίπτωση του postfix, είναι εύκολο να καθοριστεί ένα δένδρο ανίχνευσης ή ένα συντακτικό δένδρο από ένα ΣΚΜ σχήμα. Το Σχήμα 6.7 καθορίζει εκφράσεις. E.VAL είναι μια μετάφραση της οποίας η τιμή είναι ένας δείκτης σ' ένα κόμβο του συντακτικού δένδρου.

κανόνας	δράση
(1) E -> E(1)opE(2)	{E.VAL:=NODE(op,E(1).VAL,E(2).VAL)}
(2) E -> (E(1))	{E.VAL:=E(1).VAL}
(3) E -> -E(1)	{E.VAL:=UNARY(-, E(1).VAL}
(4) E -> id	{E.VAL:=LEAF(id)}

Σχήμα 6.7 Δημιουργία συντακτικού δένδρου

Η συνάρτηση NODE δημιουργεί ένα καινούριο κόμβο (op) και κάνει την δεύτερη και τρίτη παράμετρο αριστερό και δεξιό "παιδί" του καινούριου κόμβου, επιστρέφοντας ένα δείκτη για τον δημιουργηθέντα κόμβο. Η συνάρτηση UNARY (OP, CHILD) δημιουργεί ένα καινούριο κόμβο που περιγράφεται OP και δημιουργεί CHILD σαν παιδί του και φυσικά επιστρέφει ένα δείκτη στον κόμβο. Η συνάρτηση LEAF (ID) δημιουργεί ένα κόμβο που επιγράφεται ID και επιστρέφει ένα δείκτη στον κόμβο. Στην πράξη η επιγραφή του κόμβου ID μπορεί να είναι ένας δείκτης στο Symbol Table.

6.2.3 ΚΩΔΙΚΑΣ ΤΡΙΩΝ ΔΙΕΥΘΥΝΣΕΩΝ ΚΑΙ ΥΛΟΠΟΙΗΣΕΙΣ ΤΟΥ.

Ο κώδικας τριών διευθύνσεων είναι μία ακολουθία εντολών της γενικής μορφής A:=B op C, όπου A, B και C είναι μεταβλητές του προγράμματος, σταθερές ή μεταβλητές που δημιούργησε ο Μεταγλωττιστής και op είναι κάποιος τελεστής. Είναι φανερό ότι ο κώδικας τριών διευθύνσεων δεν επιτρέπει πολύπλοκες αριθμητικές εκφράσεις. Οι πιο κοινές εντολές τριών διευθύνσεων είναι οι παρακάτω:

a) Εκχωρήσεις A:=B op C όπου op δυαδικός αριθμητικός ή λογικός τελεστής.
b) Εκχωρήσεις τύπου A:=op B όπου op = μοναδικός τελεστής π.χ. μείον, λογική άρνηση, τελεστής μετατόπισης (shift) κ.λ.π. Μία ειδική περίπτωση op είναι η ταυτοτική συνάρτηση όπου A:=B που σημαίνει ότι η τιμή του B εκχωρείται στο A.
c) goto L, που σημαίνει εκτέλεση της εντολής τριών διευθύνσεων με Label.
d) if a relop B goto L. Οπου relop είναι τελεστής συσχέτισης (<, =, >, κ.λ.π.)
e) param A και call P, n. Οι εντολές αυτές χρησιμοποιούνται στην υλοποίηση υποπρογραμμάτων. Π.χ. η ακολουθία:

   param A_1
   param A_2
   ...
   param A_n
   call P, n

   δημιουργείται σαν μέρος της κλήσης τους ρουτίνας $P(A_1,A_2,...A_n)$.
f) Δεικτοδοτημένες εκχωρήσεις της μορφής A:=B[I] και A[I]:=B. Τα A, B και I υποτίθεται ότι αναφέρονται σε δεδομένα και παριστάνονται με δείκτες στον πίνακα συμβόλων.
g) Εκχωρήσεις διευθύνσεων (addresses) και δεικτών (pointers) του τύπου: A:=addr B, A=*B και *A=B. Η πρώτη θέτει στην A την διεύθυνση της B.

Οι εντολές τριών διευθύνσεων είναι μία αφηρημένη μορφή ενδιάμεσου κώδικα. Σε έναν μεταγλωττιστή οι εντολές αυτές μπορούν να υλοποιηθούν με έναν από τους παρακάτω τρόπους.

Τετράδες (quadruples)

Η αναπαράσταση αυτή των εντολών τριών διευθύνσεων αποτελείται από μία δομή τεσσάρων πεδίων που τα συμβολίζουμε με ΚΩΔ, ΠΑΡ1, ΠΑΡ2, και ΑΠΟΤΕΛ. ΚΩΔ περιέχει ένα εσωτερικό κώδικα για τον τελεστή. Κάνουμε την παραδοχή ότι εντολές με unary (μοναδικούς) τελεστές - π.χ. A:=-B ή A:=B - δεν χρησιμοποιούν το ΠΑΡ2. Τελεστές όπως param δεν χρησιμοποιούν ούτε ΠΑΡ2 ούτε ΑΠΟΤΕΛ.

Παράδειγμα 5 / Κεφ. 6

Η εντολή εκχώρησης A:= -B * (C+D) μπορεί να μεταφρασθεί στις παρακάτω εντολές τριών διευθύνσεων (τα T1, T2, T3 είναι ονόματα προσωρινών μεταβλητών)

T1 := - B
T2 := C+D

T3 := T1 * T2

A := T3

οι οποίες αναπαριστάνονται από τις παρακάτω τετράδες του Σχήματος 6.8.

	ΚΩΔ	ΠΑΡ1	ΠΑΡ2	ΑΠΟΤΕΛ
(0)	Uminus	B	-	T1
(1)	+	C	D	T2
(2)	*	T1	T2	T3
(3)	:=	T3	-	A

Σχήμα 6.8 Οι τετράδες της A:= -B * (C+D)

Τα περιεχόμενα των πεδίων ΠΑΡ1, ΠΑΡ2, και ΑΠΟΤΕΛ είναι συνήθως δείκτες στις θέσεις του πίνακα συμβόλων. Στην περίπτωση αυτή τα προσωρινά ονόματα (π.χ. T1, T2, T3) πρέπει να εισάγονται στον πίνακα συμβόλων καθώς δημιουργούνται. Στη βιβλιογραφία (π.χ. στο προτεινόμενο βιβλίο [2]) μπορείτε να βρείτε τρόπους με τους οποίους τα προσωρινά ονόματα μπορούν να επαναχρησιμοποιούνται για να μην δημιουργείται συνωστισμός συμβόλων στον πίνακα.

Τριάδες (Triples)

Για να αποφύγουμε την εισαγωγή προσωρινών ονομάτων στον πίνακα συμβόλων μπορούμε να επιτρέψουμε η εντολή που υπολογίζει κάποια τιμή ενός προσωρινού ονόματος να "παριστάνει" την τιμή αυτή. Κάνοντας αυτό, οι εντολές τριών διευθύνσεων αναπαριστάνονται με μία δομή τριών πεδίων (τριάδες) ΚΩΔ, ΠΑΡ1, ΠΑΡ2, όπου τα ΠΑΡ1 και ΠΑΡ2 είναι είτε δείκτες στον πίνακα συμβόλων είτε δείκτες στην δομή των τριάδων. Χρησιμοποιούμε αριθμούς σε παρενθέσεις για να παραστήσουμε δείκτες στην δομή των τριάδων, ενώ οι δείκτες του πίνακα συμβόλων παριστάνονται από αυτά τα ίδια τα ονόματα.

Παράδειγμα 6 / Κεφ. 6

Ο κώδικας τριών διευθύνσεων του παραδείγματος 5 μπορεί να υλοποιηθεί με τη μορφή τριάδων του Σχήματος 6.9.

	ΚΩΔ	ΠΑΡ1	ΠΑΡ2
(0)	Uminus	B	-
(1)	+	C	D
(2)	*	(0)	(1)
(3)	:=	A	(2)

Σχήμα 6.9 Οι τριάδες της A:= -B * (C+D)

Ένας τριαδικός τελεστής όπως ο A[I]:=B χρειάζεται δύο τριάδες και παριστάνεται όπως δείχνεται στο Σχήμα 6.10 (α). Επίσης δυο τριάδες χρειάζεται και ο A:=B[I] όπως δείχνεται στο 6.10(β).

	ΚΩΔ	ΠΑΡ1	ΠΑΡ2
(0)	[]=	A	I
(1)	:=	(0)	B

(α)

	ΚΩΔ	ΠΑΡ1	ΠΑΡ2
(0)	=[]	B	I
(1)	:=	A	(0)

(β)

Σχήμα 6.10 Οι τριάδες του A[I]:=B και του A:=B[I]

Τόσο οι τετράδες όσο και οι τριάδες κάνουν σπατάλη χώρου. Όταν υπάρχει έλλειψη χώρου μπορούμε να χρησιμοποιήσουμε ένα μονοδιάστατο πίνακα μία και ο κάθε τελεστής καθορίζει ποια πεδία χρησιμοποιούνται κάθε φορά. Έτσι οι τετράδες του Σχήματος 6.8 αναπαριστάνονται γραμμικά ως εξής:

uminus, B, T1, +, C, D, T2, *, T1, T2, T3, :=, T3, A

Η μέθοδος μειονεκτεί όταν προσπαθήσουμε να εξετάσουμε τις εντολές με αντίστροφη σειρά. Δεν γνωρίζουμε κατά πόσο μία λέξη παριστάνει τελεστή ή έντελο. Η αντίστροφη εξέταση είναι πρακτικό πρόβλημα στην διευκόλυνση δημιουργίας κώδικα (τελικού) object.

Σύνοψη Ενότητας

Στην ενότητα αυτή μελετήσαμε τις τρεις βασικές μορφές ενδιάμεσου κώδικα με περισσότερη έμφαση στον postfix και τον κώδικα τριών διευθύνσεων. Στις επόμενες ενότητες θα υιοθετήσουμε σαν ενδιάμεσο κώδικα τον κώδικα τριών διευθύνσεων με την μορφή τετράδων και θα προτείνουμε μεταφραστικά σχήματα δημιουργίας τέτοιου κώδικα για εκφράσεις και διάφορους τύπους εντολών.

ΕΝΟΤΗΤΑ 6.3 ΜΕΤΑΦΡΑΣΗ ΕΝΤΟΛΩΝ ΕΚΧΩΡΗΣΗΣ

Στόχος

Στόχος της ενότητας αυτής είναι να μελετήσουμε την δημιουργία ενδιάμεσου κώδικα για εντολές εκχώρησης και αριθμητικές εκφράσεις με ακέραιους μόνο αλλά και με μικτούς τύπους δεδομένων.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα αυτή θα μπορείτε να

- Κατασκευάσετε ένα μεταφραστικό σχήμα για εντολές εκχώρησης,
- Κατασκευάσετε μεταφραστικά σχήματα για αριθμητικές εκφράσεις με έντελα ενός μόνο τύπου,
- Κατασκευάσετε μεταφραστικά σχήματα για αριθμητικές εκφράσεις με έντελα μικτού τύπου.

Έννοιες-Κλειδιά

Εντολές εκχώρησης, μίξη τύπων

Εισαγωγικές παρατηρήσεις

Εδώ θα περιγράψουμε ένα Συντακτικά Κατευθυνόμενο Μεταφραστικό σχήμα για απλές εντολές εκχώρησης και αριθμητικές εκφράσεις. Ακόμη για απλούστευση του προβλήματος υποθέτουμε ότι όλοι οι identifiers υποδηλώνουν πρωτογενείς τύπους δεδομένων.

Ξεκινάμε υποθέτοντας ότι έχουμε εντολές εκχώρησης και αριθμητικές εκφράσεις με ακέραιους μόνο τύπους δεδομένων. Η παρακάτω γραμματική (1) περιγράφει την μορφή των εκχωρήσεων αυτού του τύπου.

A -> id := E	(1)
E -> E + E | E * E| -E | (E) | id	

Προφανώς ο κανόνας Α δηλώνει εκχώρηση. Θα δείξουμε μόνο τους δύο δυαδικούς τελεστές + και * σαν παραδείγματα ολόκληρου του συνόλου των τελεστών που εμφανίζονται σε μία τυπική γλώσσα προγραμματισμού. Εν γένει, ο κώδικας τριών διευθύνσεων για το id:= Ε αποτελείται από κώδικα (εντολές) για τον υπολογισμό του Ε σε κάποιο "όνομα" Τ, ακολουθούμενο από την εκχώρηση Χ:=Τ, όπου Χ είναι το όνομα που αντιστοιχεί στον id στα αριστερά της εκχώρησης.

Για τον υπολογισμό μιας έκφρασης της μορφής Ε+Ε, για παράδειγμα, πρώτα υπολογίζουμε την έκφραση στα αριστερά, κατόπιν την άλλη στα δεξιά του +, χρησιμοποιώντας δύο προσωρινά ονόματα, έστω Τ1 και Τ2, που κρατούν τις τιμές των εκφράσεων. Κατόπιν προσθέτουμε τα Τ1 και Τ2 και δίνουμε στο αποτέλεσμα ένα καινούριο προσωρινό όνομα Τ3.

6.3.1 ΜΕΤΑΦΡΑΣΤΙΚΟ ΣΧΗΜΑ

Για την δημιουργία καινούριων προσωρινών ονομάτων όπου χρειάζεται θα κάνουμε χρήση της συνάρτησης NEWTEMP(), η οποία επιστρέφει ένα καινούριο προσωρινό όνομα με κάθε κλήση της.

Θα παρουσιάσουμε τώρα το μεταφραστικό σχήμα δημιουργίας ενδιάμεσου κώδικα σε μορφή τετράδων για την γραμματική (1). Κατ' αρχήν, E.PLACE και id.PLACE μπορούν να θεωρούνται σαν δείκτες στον πίνακα συμβόλων και ακόμη οι δείκτες αυτοί μπορούν να κρατούνται στην στοίβα ενός bottom-up αναλυτή μαζί με τα αντίστοιχα γραμματικά τους σύμβολα.

Για ευκολία στους συμβολισμούς μας, θα κάνουμε χρήση μιας procedure GEN(A := B op C) η οποία θα εισάγει τον τελεστή op και τις τιμές των Α, Β και C στο διάνυσμα των τετράδων.

Μπορούμε τώρα να δείξουμε στο Σχήμα 6.11 το μεταφραστικό σχήμα για τη γραμματική (1).

(1)	A->id := E	{ GEN(id.PLACE := E.PLACE) }
(2)	E->E(1)+E(2)	{ T:=NEWTEMP(); E.PLACE:=T; GEN(E.PLACE := E(1).PLACE + E(2).PLACE) }
(3)	E-> E(1)*E(2)	{ T:=NEWTEMP(); E.PLACE:=T; GEN(E.PLACE := E(1).PLACE * E(2).PLACE) }
(4)	E->-E(1)	{ T:=NEWTEMP(); E.PLACE:=T; GEN(E.PLACE := - E(1).PLACE) }
(5)	E->(E(1))	{όχι κλήση}
(6)	E->id	{όχι κλήση}

Σχήμα 6.11 Μεταφραστικό σχήμα για τη γραμματική (1)

Παράδειγμα 7 / Κεφ. 6

Η δράση ενός bottom-up shift-reduce αναλυτή για την εκχώρηση A := -B * (C+D) φαίνεται στο Σχήμα 6.12. Το πεδίο PLACE φυλάσσεται μαζί με τα γραμματικά σύμβολα στην ίδια στοίβα. Στο σχήμα, το PLACE δείχνεται σε δική του στοίβα στην οποία η ένδειξη "-" υποδηλώνει κενή θέση για καλύτερη κατανόηση της αντιστοίχησης με την στοίβα του αναλυτή. Εντολές τριών διευθύνσεων δείχνονται στο βήμα πριν από την πραγματική δημιουργία τους.

Είσοδος	Stack	PLACE	Ενδιάμ. Κώδικας
A:=- B*(C+D)			
:=- B*(C+D)	id	A	
- B*(C+D)	id :=	A -	
B*(C+D)	id := -	A -	
*(C+D)	id := - id	A- -B	
*(C+D)	id := - E	A- -B	T1 := - B
*(C+D)	id := E	A-T1	
(C+D)	id := E*	A-T1-	
(C+D)	id := E*(	A-T1--	
+D)	id := E*(id	A-T1--C	
+D)	id := E*(E	A-T1--C	
D)	id := E*(E+	A-T1--C-	
)	id := E*(E+id	A-T1--C-D	
)	id := E*(E+E	A-T1--C-D	T2 := C+D
)	id := E*(E	A-T1--T2	
	id := E*(E)	A-T1--T2	
	id := E*E	A-T1-T2	T3 := T1 * T2
	id := E	A-T3	A := T3
	A	A	

Σχήμα 6.12 Ιχνηλάτηση μιας συντακτικά κατευθυνόμενης μετάφρασης.

6.3.2 ΕΚΦΡΑΣΕΙΣ ΜΕ ΜΙΞΗ ΤΥΠΩΝ

Στο προηγούμενο παράδειγμα υποθέσαμε ότι όλα τα id είναι του ίδιου τύπου. Στην πράξη βέβαια χρησιμοποιούνται μεταβλητές και σταθερές διαφόρων τύπων και ο μεταγλωττιστής πρέπει είτε να απορρίψει ορισμένες πράξεις μικτού τύπου είτε να φροντίσει για την δημιουργία κατάλληλων εντολών μετατροπής.

Έστω η ίδια γραμματική (1) που χρησιμοποιήθηκε παραπάνω. Υποθέτουμε τώρα ότι υπάρχουν δύο τύποι-real και integer και ότι οι integers μετατρέπονται σε reals όπου χρειάζεται. Μπορούμε να εισάγουμε ένα καινούριο πεδίο στην μετάφραση του E, το E. MODE, του οποίου η τιμή είναι είτε REAL είτε INTEGER.

Το μέρος της σημασιολογικής δράσης που αφορά το E.MODE και που σχετίζεται με τον γενικότερο κανόνα E -> E(1) op E(2) είναι:

```
{  if E(1).MODE = INTEGER and  E(2).MODE = INTEGER
        then E.MODE := INTEGER
        else E.MODE := REAL  }
```

Ολόκληρος ο σημασιολογικός κανόνας για τον κανόνα E->E(1)opE(2) και οι περισσότεροι άλλοι κανόνες πρέπει να τροποποιηθούν, όπου χρειάζεται, για να δημιουργηθούν εντολές τριών διευθύνσεων της μορφής A:= inttoreal B, των οποίων το αποτέλεσμα είναι η μετατροπή του ακέραιου B σ' ένα real της ίδιας τιμής, καλούμενο A. Ακόμη πρέπει να συμπεριλάβουμε μαζί με τον κώδικα του τελεστή και μία ένδειξη για το κατά πόσον υπάρχει πρόθεση για fixed ή floating-point αριθμητική. Για παράδειγμα, για είσοδο X: = Y + I * J, υποθέτοντας ότι τα X και Y είναι τύπου REAL και I και J έχουν τύπο INTEGER, η έξοδος θα είναι ως εξής:

```
T1 := I int* J
T2 := inttoreal T1
T3 := Y real+ T2
X := T3
```

Δραστηριότητα 1 / Κεφ. 6

Λαμβάνοντας υπόψη τις προηγούμενες παρατηρήσεις προσπαθήστε να γράψετε την πλήρη σημασιολογική δράση για τον κανόνα E->E(1)opE(2) δημιουργώντας κώδικα ο οποίος κάνει τις όποιες μετατροπές τύπων είναι απαραίτητες. Προσπαθήστε χωρίς να κοιτάξετε την απάντηση που δίνεται στη συνέχεια

Η σημασιολογική δράση περιγράφεται στο Σχήμα 6.13 και χρησιμοποιεί δύο μεταφραστικά πεδία E.PLACE και E.MODE για το μη-τερματικό σύμβολο E. Ο αναλυτής επομένως θα χρειάζεται δύο πεδία στη στοίβα μετάφρασης για την υλοποίηση αυτού του μεταφραστικού σχήματος. Καθώς ο αριθμός των τύπων (modes) αυξάνει, αυξάνουν κατά δύναμη του δύο και οι διαφορετικές περιπτώσεις (ή και χειρότερα αν

υπάρχουν τελεστές με περισσότερα από δύο έντελα). Χρειάζεται λοιπόν προσεκτική κωδικοποίηση των σημασιολογικών κανόνων όταν αυξάνει ο αριθμός των τύπων (modes).

```
{  T:= NEWTEMP( ) ;
   if E(1).MODE = INTEGER and E(2).MODE = INTEGER
   then begin
                GEN(T:=E(1).PLACE int op E(2).PLACE) ;
                E.MODE:= INTEGER
        end
   else if E(1).MODE= REAL and E(2).MODE = REAL
        then begin
                GEN(T:=E(1).PLACE real op E(2).PLACE) ;
                E.MODE := REAL
             end
         else if E(1).MODE = INTEGER    /* and E(2).MODE=REAL*/
              then begin
             U:=NEWTEMP( ) ;
             GEN(U:=inttoreal E(1).PLACE) ;
             GEN(T:=U real op E(2). PLACE) ;
             E.MODE:=REAL
                   end
              else  /* E(1).MODE = REAL and E(2).MODE = INTEGER*/
              begin
                U:=NEWTEMP( ) ;
                GEN(U:=inttoreal E(2).PLACE) ;
                GEN(T:=E(1).PLACE real op U) ;
                E.MODE := REAL
              end;
E. PLACE:=T     }
```

Σχήμα 6.13 Σημασιολογική δράση για τον κανόνα E -> E(1) op E(2).

Σύνοψη ενότητας

Στην ενότητα αυτή είδαμε την μορφή που πρέπει να έχουν τα μεταφραστικά σχήματα που φτιάχνουμε για να δημιουργήσουμε κώδικα τετράδων τόσο για εντολές εκχώρησης, όσο και για αριθμητικές εκφράσεις οι οποίες περιλαμβάνουν ένα μόνο ή και περισσότερους τύπους δεδομένων.

ΕΝΟΤΗΤΑ 6.4 ΜΕΤΑΦΡΑΣΗ ΛΟΓΙΚΩΝ ΕΚΦΡΑΣΕΩΝ

Στόχος

Στόχος της ενότητας αυτής είναι να μελετήσουμε τρόπους αναπαράστασης των λογικών εκφράσεων και μεταφραστικά σχήματα δημιουργίας ενδιάμεσου κώδικα αυτών

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα αυτή θα μπορείτε να

- Περιγράψετε τις δύο βασικές μεθόδους αναπαράστασης των λογικών εκφράσεων,

- Κατασκευάσετε μεταφραστικά σχήματα δημιουργίας κώδικα αριθμητικής αναπαράστασης των λογικών εκφράσεων,
- Κατασκευάσετε μεταφραστικά σχήματα δημιουργίας κώδικα των λογικών εκφράσεων κάνοντας χρήση της αναπαράστασης ελέγχου ροής,

Έννοιες-Κλειδιά

Αριθμητική αναπαράσταση, αναπαράσταση ελέγχου ροής, σημασιολογική δράση, οπισθομπάλωμα, παραγοντοποίηση συντακτικών κανόνων, μαρκαδόροι (σηματοδότες).

Εισαγωγικές παρατηρήσεις

Ας υποθέσουμε ότι μία περιορισμένη μορφή Boolean εκφράσεων δίνεται από την γραμματική:

E -> E or E |E and E | not E | (E) | id | id relop id

όπου relop είναι ένα από τα <, ≤, =, #, > ή ≥, και ακόμη υποθέτουμε ότι or και and είναι αριστερά-προσεταιριστικοί και ότι or έχει μικρότερη προτεραιότητα, κατόπιν and και κατόπιν not.

Υπάρχουν δύο βασικές μέθοδοι για την αναπαράσταση της τιμής μίας boolean έκφρασης. Η πρώτη μέθοδος είναι να κωδικοποιηθούν τα true και false αριθμητικά και να υπολογίζεται μια boolean έκφραση κατά τρόπο ανάλογο με τις αριθμητικές εκφράσεις. Συχνά χρησιμοποιείται το 1 για true και το 0 για false ενώ θα μπορούσε κανείς ακόμη να χρησιμοποιήσει οποιαδήποτε μη αρνητική ποσότητα για true και οποιοδήποτε αρνητικό αριθμό για false.

Η δεύτερη μέθοδος υλοποίησης των boolean εκφράσεων είναι μέσω ελέγχου ροής (by flow of control) δηλαδή, αναπαριστώντας την τιμή μιας boolean έκφρασης με κάποια σχετική θέση σ' ένα πρόγραμμα. Η δεύτερη μέθοδος είναι ιδιαίτερα χρήσιμη στην υλοποίηση εντολών τύπου if-then-else και while-do.

Στην συνέχεια θα συζητήσουμε και τις δύο μεθόδους για την μετάφραση boolean εκφράσεων σε κώδικα τριών διευθύνσεων. Πέρα από τις εντολές τριών διευθύνσεων που είδαμε μέχρι τώρα, θα χρησιμοποιήσουμε και τις εντολές αλλαγής ροής:

goto L,

if A goto L,

if A relop B goto L,

Όπου A και B είναι απλές μεταβλητές ή σταθερές, L είναι η επιγραφή (label) μιας τετράδας (quadruple) και relop είναι τελεστής συσχέτισης όπως παραπάνω (υποθέτουμε ότι φτιάχνουμε τετράδες σαν κώδικα τριών διευθύνσεων).

6.4.1 ΑΡΙΘΜΗΤΙΚΗ ΑΝΑΠΑΡΑΣΤΑΣΗ

Έστω ότι χρησιμοποιούμε 1 για true και 0 για false. Οι εκφράσεις θα υπολογίζονται από αριστερά προς τα δεξιά, παρόμοια με τις αριθμητικές. Π.χ. η μετάφραση του

A or B and C είναι η παρακάτω ακολουθία εντολών τριών διευθύνσεων:

T1 := B and C

T2 := A or T1

Υποθέτουμε ότι and και or τελικά υλοποιούνται με αντίστοιχους λογικούς τελεστές του στοχευόμενου υπολογιστή.

Μια relational expression όπως η A<B είναι πρακτικά ισοδύναμη με την υπό συνθήκη εντολή if A < B then 1 else 0, η οποία μπορεί να μεταφρασθεί στην παρακάτω ακολουθία του σχήματος 6.14 (α) στην οποία το T είναι προσωρινή μεταβλητή.

```
1. if A<B goto (4)
2. T := 0
3. goto (5)
4. T : = 1
5. ...
```

Σχήμα 6.14 (α)

Έτσι η boolean έκφραση A < B or C μπορεί να μεταφρασθεί όπως στο Σχήμα 6.14 (β).

```
1. if A < B goto (4)
2. T1 := 0
3. goto (5)
4. T1 := 1
5. T2 := T1 or C
```

Σχήμα 6.14 (β)

Η σημασιολογική δράση για τον κανόνα E -> E or E είναι η παρακάτω:

```
E -> E(1) or E(2)      {  T:=NEWTEMP( ) ; E.PLACE:=T;
                          GEN(T:=E(1).PLACE or E(2).PLACE  }
```

Άσκηση αυτοαξιολόγησης 3 / Κεφ. 6

Λαμβάνοντας υπόψη την μορφή του κώδικα που δείχνεται στο Σχήμα 6.14(α), γράψτε το ΣΚΜ σχήμα του κανόνα E -> id relop id

Παρόμοιες δράσεις μπορούν να ορισθούν και για τους άλλους κανόνες. Στους κανόνες αυτούς υποθέτουμε ότι δημιουργούνται τετράδες και ότι NEXTQUAD δηλώνει την επόμενη κενή θέση στο διάνυσμα των τετράδων. Η GEN αυξάνει το NEXTQUAD μετά την δημιουργία κάθε τετράδας.

6.4.2 ΑΝΑΠΑΡΑΣΤΑΣΗ ΕΛΕΓΧΟΥ ΡΟΗΣ (Control Flow)

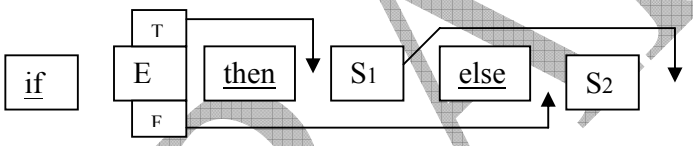
Στο Σχήμα 6.14 η τιμή του T1 κατά κάποιο τρόπο πλεονάζει μια και μπορούμε να γνωρίζουμε την τιμή του T1 από το κατά πόσον έχουμε φτάσει στην εντολή (2) ή στην εντολή (4). Αυτό υποδηλώνει ότι μπορούμε να χρησιμοποιήσουμε την θέση που φτάσαμε στο πρόγραμμα για να παραστήσουμε την τιμή μιας boolean έκφρασης. Αν υπολογίζουμε εκφράσεις με θέσεις στο πρόγραμμα, μπορούμε να αποφύγουμε τον υπολογισμό ολόκληρης της έκφρασης. Για παράδειγμα, δοθείσης της έκφρασης A or B, αν προσδιορίσουμε ότι το A είναι true, τότε μπορούμε να υποθέσουμε ότι όλη η έκφραση είναι true χωρίς να υπολογίσουμε και το B.

Στην συνέχεια θα θεωρήσουμε την μετάφραση των boolean εκφράσεων στο πλαίσιο των υπό συνθήκη εντολών όπως:

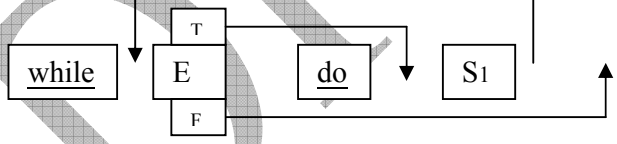
if E then S(1) else S(2), και

while E do S(1)

Στα πλαίσια αυτά μπορούμε να συσχετίσουμε δύο είδη εξόδων (exits) με την boolean έκφραση E, μια έξοδο true στην εντολή TRUE και μια false στην εντολή FALSE. Η γενική μορφή των μεταφράσεων για τις εντολές αυτές δείχνονται στο Σχήμα 6.15(α) και (β).



(α)



(β)

Σχήμα 6.15 Σχηματικός κώδικας για δομές που χρησιμοποιούν boolean εκφράσεις.

Παράδειγμα 8 / Κεφ. 6

Έτσι, με βάση τα παραπάνω, αν έχουμε την εντολή Fortran:

IF(A.LT.B.OR.C.LT.D) X = Y + Z

μπορούμε να την μεταφράσουμε στην παρακάτω ακολουθία εντολών τριών διευθύνσεων:

```
1. if A < B  goto  (4)          Εδώ  (4) είναι η TRUE και
2. if C < D  goto  (4)                (6) είναι η FALSE έξοδος.
3. goto 6
4. T := Y + Z
5. X := T
```

6.**...**

Μια έκφραση E θα μεταφρασθεί σε μια ακολουθία εντολών τριών διευθύνσεων που "υπολογίζουν" το E. Η μετάφραση αυτή είναι μια ακολουθία από εντολές αλλαγής ροής (υπό συνθήκη ή χωρίς συνθήκη) οι οποίες κατευθύνονται σε μια από δύο διευθύνσεις. Η μια από τις διευθύνσεις είναι εκείνη στην οποία πηγαίνει το πρόγραμμα όταν η τιμή της έκφρασης είναι true, και η άλλη είναι εκείνη στην οποία πηγαίνει το πρόγραμμα όταν η τιμή της έκφρασης είναι false.

Έστω μια έκφραση της μορφής E(1) or E(2). Αν E(1) είναι true, τότε γνωρίζουμε αμέσως ότι το E είναι true και μπορούμε να κάνουμε την θέση TRUE της E(1) να είναι η ίδια με την TRUE της E. Αν E(1) είναι false, τότε πρέπει να υπολογίσουμε το E(2), και ως εκ τούτου κάνουμε την FALSE της E(1) να δείχνει στην πρώτη εντολή του κώδικα για το E(2). Οι έξοδοι true και false του E(2) μπορούν να γίνουν οι ίδιες με τις εξόδους true και false του E αντίστοιχα.

Ανάλογες σκέψεις ισχύουν για την μετάφραση του E(1) and E(2). Η μετάφραση μιας έκφρασης E της μορφής notE(1) είναι πιο εύκολη. Απλώς εναλλάσσουμε τις εξόδους true και false του E(1) για να πάρουμε τις true και false του E.

Ας θεωρήσουμε τώρα ένα σχήμα ΣΚΜ που δημιουργεί τετράδες για boolean εκφράσεις όπως περιγράφηκε παραπάνω.

Ένα πρόβλημα που εμφανίζεται όταν δημιουργούμε κώδικα bottom-up είναι ότι μπορεί να μην έχουμε δημιουργήσει τις πραγματικές τετράδες στις οποίες μεταφέρεται ο έλεγχος (γίνονται τα jumps) κατά την στιγμή που δημιουργούνται οι εντολές jump. Ο κώδικας λοιπόν που δημιουργούμε είναι μια ακολουθία από εντολές jump με τους στόχους (διευθύνσεις) των jumps να είναι προσωρινά ακαθόριστοι. Κάθε τέτοια τετράδα θα ανήκει σε κάποια λίστα τετράδων που πρόκειται να συμπληρωθούν όταν οι στόχοι (διευθύνσεις) έχουν προσδιορισθεί. Αυτό το εκ των υστέρων συμπλήρωμα των τετράδων καλείται backpatching (οπισθομπάλωμα) και έχουμε αναφερθεί σ'αυτό στο κεφάλαιο 1.
Για τον χειρισμό των τετράδων χρησιμοποιούμε τις παρακάτω ρουτίνες.

1. MAKELIST (i) δημιουργεί μια καινούρια λίστα η οποία περιέχει μόνο το i, ένα δείκτη στο διάνυσμα των τετράδων που δημιουργούνται. Η ρουτίνα επιστρέφει ένα δείκτη στην λίστα που έφτιαξε.
2. MERGE (P1, P2), παίρνει τις δύο λίστες που δείχνουν οι δείκτες P1 και P2 τις συνενώνει και επιστρέφει ένα δείκτη στην ενωμένη λίστα.
3. BACKPATCH (p, i), συμπληρώνει με i (στοχευόμενος αριθμός τετράδες) κάθε ελλιπή τετράδα της λίστας που δείχνει ο δείκτης p.

Μπορούμε τώρα να περιγράψουμε τις δύο μεταφράσεις του E που χρειάζονται για την δημιουργία του κώδικα τριών διευθύνσεων. Οι δύο αυτές μεταφράσεις E.TRUE και E.FALSE είναι δείκτες σε λίστες τετράδων που έχουν δημιουργηθεί και οι οποίες πρέπει να συμπληρωθούν με τον στοχευόμενο αριθμό τετράδες, εφ' όσον το E είναι true και false αντίστοιχα.

Οι σημασιολογικές δράσεις αντανακλούν τις παραπάνω σκέψεις.

Έστω ο κανόνας E -> E(1) and E(2). Αν E(1) είναι false, τότε E είναι false και επομένως οι τετράδες της λίστας E. FALSE μπορούν να συμπληρωθούν με την θέση (αριθμό τετράδας) που ακολουθεί την μεγαλύτερη έκφραση E στην περίπτωση που το E είναι false. Δηλαδή, η σημασιολογική ρουτίνα για το E -> E(1) and E(2), μεταξύ άλλων, θα κάνει το E(1).FALSE να είναι μέρος της λίστας E.FALSE. Αν όμως το E(1) είναι true, τότε πρέπει να εξετάσουμε το E(2). Έτσι ο στόχος για τις τετράδες της λίστας E.TRUE πρέπει να είναι η αρχή του κώδικα που δημιουργείται για το E(2).

Αν όμως περιμένουμε μέχρις ότου φτάσουμε στο σημείο να μειώσουμε το E(1) and E(2) σε E, θα είναι πολύ αργά για να πισωμπαλώσουμε την λίστα E(1).TRUE, μια και δεν θα έχουμε πια τον αριθμό της πρώτης τετράδας του E(2).CODE.

Μια λύση είναι να παραγοντοποιήσουμε το E -> E(1) and E(2), έτσι ώστε η κατάλληλη σημασιολογική δράση, πισωμπάλωμα του E(1).TRUE, να μπορεί να γίνει ευθύς αμέσως την δημιουργία του κώδικα του E(1).

Εκείνη τη στιγμή ένας δείκτης NEXTQUAD, ο οποίος κρατάει τον αριθμό της επόμενης τετράδας που ακολουθεί δίνει την τιμή με την οποία πρέπει να συμπληρωθούν οι τετράδες της λίστας E(1).TRUE.

Δηλαδή, μπορούμε να αντικαταστήσουμε το E -> E and E με τους:

E -> EAND E

EAND -> E and

και να κάνουμε την κλήση BACKPATCH (E(1).TRUE, NEXTQUAD) να είναι μέρος της σημασιολογικής ρουτίνας για το EAND -> E and.

Μια άλλη λύση είναι να εισάγουμε ένα κανόνα M ->ε, σαν μαρκαδόρο, με σημασιολογική δράση:

M ->ε {M.QUAD := NEXTQUAD}

Ο αποκλειστικός σκοπός αυτού του κανόνα είναι μια σημασιολογική δράση θα συλλάβει την τιμή του NEXTQUAD την κατάλληλη στιγμή και θα τη διατηρήσει για μετέπειτα χρήση. Η τροποποιημένη γραμματική είναι:

```
1.  E ->  E(1) or M E(2)
2.        | E(1) and M E(2)
3.        | not E(1)
4.        | (E(1))
5.        | id
6.        | id(1) relop id(2)
```

```
7.  M -> ε
```

Το ΣΚΜ σχήμα για τους κανόνες 1, 2, 3, 5 και 7 δίνεται παρακάτω.

```
1. E -> E(1) or M E(2)
        {  BACKPATCH(E(1).FALSE, M.QUAD);
           E.TRUE:= MERGE(E(1).TRUE, E(2).TRUE);
           E.FALSE:= E(2).FALSE  }
```

```
2. E -> E(1) and M E(2)
        {BACKPATCH(E(1).TRUE, M.QUAD);
           E.TRUE:= E(2).TRUE;
           E.FALSE:= MERGE(E(1).FALSE, E(2).FALSE)}
```

```
3. E -> not E(1)
        {E.TRUE:= E(1).FALSE; E.FALSE:=E(1).TRUE}
```

```
5. E -> id
        {E.TRUE:= MAKELIST(NEXTQUAD);
           E.FALSE:= MAKELIST(NEXTQUAD+1);
           GEN(if id.PLACE goto -);
           GEN(goto -)}
```

```
7. M -> ∈
        {M.QUAD:= NEXTQUAD}
```

Η σημασιολογική δράση (5) δημιουργεί δύο τετράδες, ένα goto υπό συνθήκη και ένα χωρίς συνθήκη. Και στα δύο λείπουν οι στόχοι. Ο δείκτης της πρώτης δημιουργούμενης τετράδας γίνεται μια λίστα στην οποία δείχνει ο δείκτης E.TRUE. Η δεύτερη δημιουργούμενη τετράδα, "goto -", γίνεται επίσης λίστα και δίνεται στο E.FALSE.

Άσκηση αυτοαξιολόγησης 4 / Κεφ. 6

Γράψτε το ΣΚΜ σχήμα για τους κανόνες (4) και (6) της παραπάνω γραμματικής.

Σύνοψη ενότητας

Στην ενότητα αυτή εξετάσαμε τους δύο βασικούς τρόπους αναπαράστασης της τιμής μίας λογικής έκφρασης, την αριθμητική αναπαράσταση και την αναπαράσταση ελέγχου ροής. Είδαμε την μορφή των σημασιολογικών δράσεων που αντιστοιχούν στους συντακτικούς κανόνες των λογικών εκφράσεων, και τις βοηθητικές ρουτίνες MAKELIST, MERGE, και BACKPATCH που μας βοηθούν στην κατασκευή των δράσεων. Ακόμη είδαμε τις δύο λύσεις στο πρόβλημα της οπισθομπάλωσης δηλαδή, την παραγοντοποίηση των συντακτικών κανόνων και την εισαγωγή μαρκαδόρων (σηματοδοτών) σε αυτούς.

ΕΝΟΤΗΤΑ 6.5 ΜΕΤΑΦΡΑΣΗ ΕΝΤΟΛΩΝ ΡΟΗΣ

Όταν ένας μεταγλωττιστής συναντά μια εντολή της μορφής goto L, ψάχνει για την επιγραφή L στον πίνακα συμβόλων. Αν η ροή είναι προς τα πίσω που σημαίνει ότι το L έχει ήδη συναντηθεί προηγούμενα, τότε ο πίνακας συμβόλων περιέχει το L και τον αριθμό της πρώτης τετράδας που δημιουργήθηκε για την εντολή με επιγραφή L. Στην περίπτωση αυτή δημιουργούμε μια εντολή goto με στόχο τον αριθμό αυτής της τετράδας. Στην περίπτωση που η ροή είναι προς τα εμπρός, τότε μπορεί να είναι η πρώτη φορά που συναντάμε το L και επομένως το εισάγουμε στον πίνακα συμβόλων. Εφ' όσον η εντολή με επιγραφή L δεν έχει συναντηθεί ακόμη, δημιουργούμε μια "goto -" τετράδα και προσθέτουμε τον αριθμό της τετράδας σε μια λίστα τετράδων των οποίων στόχος είναι το L. Το L στον πίνακα συμβόλων περιέχει και ένα δείκτη στην αρχή αυτής της λίστας.

Μπορούμε να περιγράψουμε την σύνταξη των επιγραφόμενων εντολών με κανόνες όπως τα

S -> LABEL : S
LABEL -> id

Η δε σημασιολογική δράση για το LABEL -> id περιλαμβάνει:

1. Τοποθέτηση του identifier στον πίνακα συμβόλων εφ' όσον δεν είναι ήδη μέσα,
2. Καταγραφή του ότι η τετράδα στην οποία αναφέρεται αυτή η επιγραφή είναι η τρέχουσα τιμή του NEXTQUAD, και τέλος
3. Πισομπάλωσε την λίστα των goto's των οποίων στόχος είναι η επιγραφή που μόλις ευρέθηκε.

ΕΝΟΤΗΤΑ 6.6 ΜΕΤΑΦΡΑΣΗ ΔΟΜΗΜΕΝΩΝ ΕΝΤΟΛΩΝ

Στόχος

Στόχος της ενότητας αυτής είναι να μελετήσουμε και να σχεδιάσουμε μεταφραστικά σχήματα δημιουργίας ενδιάμεσου κώδικα για τις πιο κοινές δομημένες εντολές των γλωσσών προγραμματισμού.

Προσδοκώμενα αποτελέσματα

Όταν θα έχετε μελετήσει την ενότητα αυτή θα μπορείτε να

- Κατασκευάσετε μεταφραστικά σχήματα για τις διάφορες μορφές της εντολής FOR,
- Κατασκευάσετε μεταφραστικά σχήματα και να γράψετε δράσεις για εντολές τύπου WHILE, IF-THEN-ELSE όπως και σύνθετες εντολές της μορφής BEGIN-END κάνοντας είτε χρήση μαρκαδόρων είτε παραγοντοποίηση των κανόνων.

Έννοιες-Κλειδιά

Δομημένες εντολές, εντολές ροής

Εισαγωγικές παρατηρήσεις

Θα εξετάσουμε τα μεταφραστικά σχήματα που χρειάζονται για τους παρακάτω κανόνες οι οποίοι περιγράφουν δομημένες εντολές. Θα εφαρμόσουμε παραγοντοποίηση

των κανόνων αλλά και χρήση μαρκαδόρων (εναλλακτικά) για να λύσουμε τα προβλήματα της οπισθομπάλωσης στις σημασιολογικές δράσεις.

S ->	for V := E(1) step E(2) until E(3) do S(1)	(1)
|	if E then S(1) else S(2)	(2)
|	while E do S(1)	(3)
|	begin L end	(4)
|	A	(5)
L->	L(1) **;** S	(6)
|	S	(7)

Σχήμα 6.16 Δομημένες εντολές

όπου S, E, A και L υποδηλώνουν αντίστοιχα statement, expression, assignment και statement list.

6.6.1 ΜΕΤΑΦΡΑΣΗ ΤΗΣ ΕΝΤΟΛΗΣ FOR

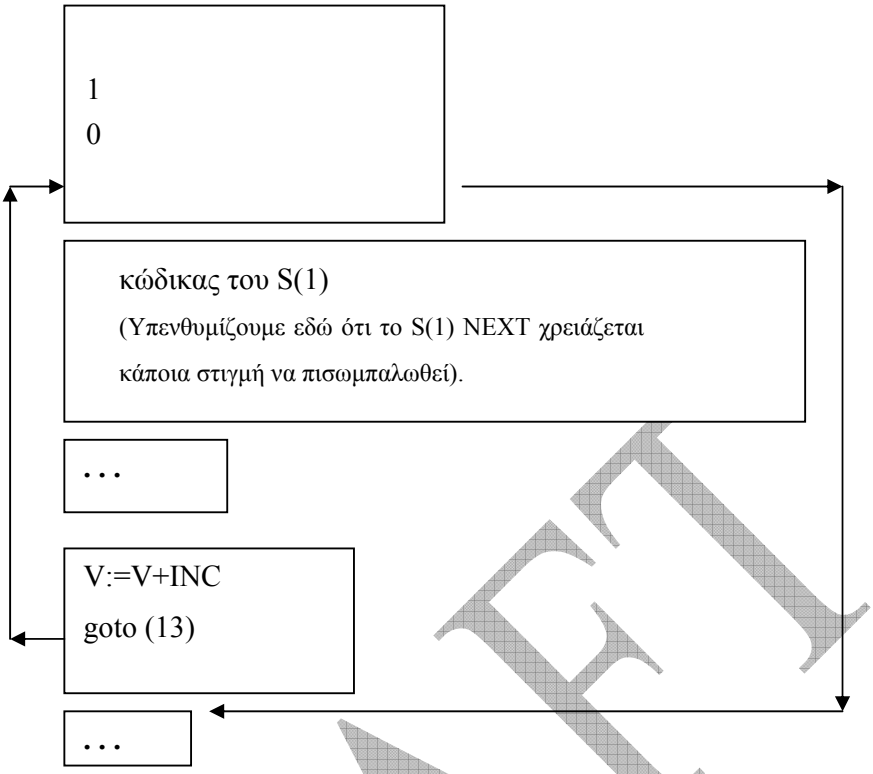
Θα ξεκινήσουμε με τον κανόνα (1) του Σχήματος 6.16 όπου υποθέτουμε ότι V είναι μια μεταβλητή, η μεταβλητή (index), της εντολής for. Ακόμη υποθέτουμε ότι οι E(1), E(2) και E(3) υπολογίζονται μόνο μια φορά και μάλιστα πριν από το loop. Στην περίπτωση της εντολής αυτής ο κώδικας που πρέπει να παραχθεί μπορεί να είναι αυτός που δείχνεται διαγραμματικά στο Σχήμα 6.17.

Για να είμαστε σε θέση να γνωρίζουμε την τιμή της INC και την τετράδα επιστροφής όταν δημιουργούμε τον κώδικα
V := V + INC; goto 13, πρέπει να κάνουμε παραγοντοποίηση του κανόνα (1) στον :

T -> for V := E(1) step E(2) until E(3) do	
S -> T S(1)	(8)

και να χρησιμοποιήσουμε το T.QUAD για να φυλάξουμε την τιμή της τετράδας (if V > FIN goto ___).
Ενώ στο πεδίο T.INC φυλάμε την INC.
Οι τετράδες (10) - (13) μπορούν να κατασκευαστούν από την δράση του κανόνα T. Εδώ παρατηρούμε ότι στην (13) λείπει ο στόχος για την επόμενη εντολή στην σειρά εκτέλεσης όταν V > FIN (ήτοι το τέλος του loop).
Η τετράδα λοιπόν (13) θα πρέπει να τοποθετηθεί στην T. NEXT αλυσίδα. Ο στόχος όμως που λείπει στην (13) είναι τελικά ο ίδιος με εκείνον της S -> TS(1). Δηλαδή σαν μέρος της δράσης της S -> TS(1) θα πρέπει να είναι και το:

S.NEXT: = T.NEXT



Σχήμα 6.17 Ο κώδικας της for διαγραμματικά.

Έχουμε λοιπόν τις εξής δράσεις (Σχήμα 6.18):

```
T -> for V = E(1) step E(2) until E(3) do
                    {   INC:= NEWTEMP( );
                        FIN:= NEWTEMP( );
                        GEN (V.PLACE:= E(1).PLACE);
                        GEN (INC:= E(2).PLACE);
                        GEN (FIN:= E(3).PLACE);
                        T.INC := INC
                        T.QUAD:= NEXTQUAD;
                        T.NEXT:= MAKELIST(NEXTQUAD);
                        GEN(if V.PLACE > FIN goto ___ )
                        T.PLACE:=V.PLACE        }
```

```
S -> T S(1)         {   BACKPATCH(S(1).NEXT, NEXTQUAD);
                          GEN (T. PLACE:= T.PLACE + T. INC);
                        GEN(T.PLACE:=T.PLACE+T.INC);
                        GEN(goto T. QUAD)
                        S.NEXT := T.NEXT        }
```

Σχήμα 6.18 Η σημασιολογική δράση της for

Στην παραπάνω συζήτηση έχει γίνει η υπόθεση ότι η τιμή της έκφρασης E(1) είναι πάντοτε θετική. Ακόμη, ότι πρώτα εξετάζουμε την συνθήκη για τέλος του loop και κατόπιν εκτελούμε τις εντολές του S(1) αν η συνθήκη δεν ικανοποιείται.

Άσκηση αυτοαξιολόγησης 5 / Κεφ. 6

Στο προηγούμενο ΣΚΜ σχήμα για την εντολή for είχε γίνει η υπόθεση ότι οι εκφράσεις E(2) και E(3) δεν χρειάζεται να υπολογίζονται σε κάθε επανάληψη της ανακύκλωσης. Σχεδιάστε ένα παρόμοιο ΣΚΜ σχήμα για την περίπτωση που οι τιμές των E(2) και E(3) μπορεί να μεταβάλλονται σε κάθε επανάληψη.

Λάβετε υπόψη σας ότι η αρχική γραμματική θα πρέπει να μετασχηματισθεί από

S -> for V := E(1) step E(2) until E(3) do S(1)

στην:

F1 -> for V := E(1)
F2 -> F1 step E(2)
F3 -> F2 until E(3)
S -> F3 do S(1)

Στην προκειμένη περίπτωση, όπως και προηγούμενα, έχουμε παραγοντοποίησει τον αρχικό κανόνα για να μπορούμε να πισομπαλώσουμε τα κενά.

6.6.2 ΜΕΤΑΦΡΑΣΗ ΔΟΜΗΜΕΝΩΝ ΕΝΤΟΛΩΝ

Είδαμε πως μπορούμε να φτιάξουμε κώδικα για την εντολή for του Σχήματος 6.16. Θα μελετήσουμε τώρα την δημιουργία κώδικα για τις υπόλοιπες εντολές του Σχήματος 6.16.

Όπως γνωρίζουμε το E έχει δυο μεταφράσεις E.T και E.F. Μεταφράσεις χρειάζονται και τα L και S. Αυτές είναι οι L.NEXT και S.NEXT. S.NEXT είναι δείκτης σε μια λίστα που περιέχει όλα τα υπό συνθήκη ή χωρίς συνθήκη άλματα στην τετράδα η οποία ακολουθεί την εντολή S στην σειρά εκτέλεσης. Παρόμοια ισχύουν για την L.NEXT. Η ανάγκη για ένα τέτοιο μεταφραστικό πεδίο φαίνεται όταν εξετάσουμε τον κώδικα που θέλουμε να παράγουμε για τον κανόνα

while E do S(1)

Ο κώδικας πρέπει να είναι τέτοιος ώστε να εκτελείται το S(1) όσο η τιμή της E είναι true. Στη συνέχεια επιστρέφει πίσω στο σημείο που γίνεται ο έλεγχος της τιμής της E. Αν και έχουμε ήδη συναντήσει την τετράδα αυτή (κώδικας για τον έλεγχο της E), θα πρέπει να λάβουμε υπόψη ότι το S(1) ενδεχομένως είναι και αυτό μια άλλη εντολή τύπου while και ενδεχομένως με πολλά επίπεδα βάθους. Είναι λοιπόν σημαντικό να μπορούμε να διακρίνουμε σε ποιά έκφραση E επιστρέφει ποιό S(1).

Εισάγουμε λοιπόν ένα καινούριο κανόνα M ο οποίος αναλαμβάνει ρόλο μαρκαδόρου και η δουλειά του είναι να κρατήσει την θέση της αρχής του κώδικα του E και του S(1) αντίστοιχα. Έτσι, ξαναγράφουμε το while E do S(1) ως εξής

S-> while M(1) E do M(2) S(1)

M-> ε

Κάθε M έχει μια μετάφραση M.QUAD η οποία είναι η θέση (ο αριθμός) της επόμενης τετράδας που πρόκειται να δημιουργηθεί στο διάνυσμα των τετράδων. Δηλαδή,

M-> ε { M.QUAD:=NEXTQUAD }

Όταν μειώνουμε το while M(1) E do M(2) S(1) σε S, πισομπαλώνουμε την λίστα S(1).NEXT για να κάνουμε όλους τους κενούς στόχους στην λίστα αυτή να δείχνουν στο M(1).QUAD. Ακόμη, προσθέτουμε και ένα άλμα προς την αρχή του κώδικα για το E. Μπορούμε τώρα να πισομπαλώσουμε την λίστα E.TRUE να πηγαίνει στην αρχή του κώδικα του S(1), δηλαδή γράφουμε

BACKPATCH(E.TRUE,M(2).QUAD)

Έχουμε λοιπόν,

```
S-> while M(1) E do M(2) S(1)  {   BACKPATCH(S(1).NEXT, M(1).QUAD);
                                   BACKPATCH(E.TRUE,M(2).QUAD);
                                   S.NEXT:=E.FALSE;
                                   GEN(goto M(1).QUAD)  }
```

Ας δούμε τώρα τον κανόνα

S-> if E then S(1) else S(2)

Όταν τελειώνουμε με την εκτέλεση του S(1), δεν γνωρίζουμε που είναι η επόμενη τετράδα μέχρι να τελειώσουμε και με την δημιουργία του κώδικα για το S(2). Χρειαζόμαστε επομένως ένα μαρκαδόρο N.

S-> if E then S(1) N else S(2)

Όταν το παραπάνω μειωθεί σε S πρέπει να γνωρίζει τις αρχές του κώδικα για τα S(1) και S(2) για να πισωμπαλώσει τις λίστες E.TRUE και E.FALSE της E αντίστοιχα. Χρειαζόμαστε επομένως δυο ακόμη μαρκαδόρους M(1) και M(2), οπότε ο κανόνας γίνεται

S-> if E then M(1) S(1) N else M(2) S(2)

Σημειώνουμε ότι ο μαρκαδόρος N έχει κανόνα και δράση τα:

```
N-> ε    {     N.NEXT:=MAKELIST(NEXTQUAD);
               GEN (goto ---)      }
```

Ο λόγος που χρησιμοποιούμε λίστα (N.NEXT) αντί για κάποιο ακέραιο είναι διότι το "goto ---" που δημιουργήθηκε πρέπει να δείχνει στην επόμενη εντολή μετά το S και να είναι ο ίδιος στόχος που πρέπει να έχουν και τα S(1).NEXT και S(2).NEXT, και που φυσικά είναι το ίδιο με το S.NEXT, άρα:

S.NEXT:=MERGE(S(1).NEXT, N.NEXT, S(2).NEXT).

Έτσι, η πλήρης δράση για τον κανόνα

S-> if E then M(1) S(1) N else M(2) S(2)

Είναι η

```
{ BACKPATCH(E.TRUE, M(1).QUAD);
  BACKPATCH(E.FALSE, M(2).QUAD);
  S.NEXT:=MERGE(S(1).NEXT, N.NEXT, S(2).NEXT) }
```

Δραστηριότητα 2 / Κεφ. 6

Σκεπτόμενοι όπως προηγούμενα, προσπαθήστε να γράψετε τις σημασιολογικές δράσεις για τους παρακάτω κανόνες χωρίς να κοιτάξετε την συνέχεια του κειμένου.

(1) S->begin L end

(2) S->A

(3) L->L(1) ; M S

(4) L->S

(1)	S->begin L end	{ S.NEXT:=L.NEXT }
(2)	S->A	{ S.NEXT:= MAKELIST() }

Στην (2) που το A δηλώνει εντολή καταχώρησης, απλώς αρχικοποιούμε την λίστα S.NEXT.

(3)	L->L(1) ; M S	{ BACKPATCH(L(1).NEXT, M.QUAD) L.NEXT:=S.NEXT }

Στην (3) γίνεται χρήση του μαρκαδόρου M διότι χρειαζόμαστε να πισομπαλώσουμε το L(1).NEXT να δείχνει στην αρχή του κώδικα του S.

Παρατηρώντας όλες τις παραπάνω δράσεις βλέπουμε ότι κώδικας δημιουργείται μόνο στους κανόνες N->ε και S-> while M(1) E do M(2) S(1). Ο ενδιάμεσος κώδικας που χρειάζεται για κάθε περίπτωση θα έχει δημιουργηθεί από τις δράσεις των εντολών καταχώρησης, των Boolean εκφράσεων και των υπολοίπων εντολών. Η βασική επομένως λειτουργία των παραπάνω δράσεων είναι να εξασφαλίσει το σωστό πισωμπάλωμα.

Άσκηση αυτοαξιολόγησης 6 / Κεφ. 6

Επαναλάβετε την άσκηση αυτοαξιολόγησης 5 κάνοντας τώρα χρήση μαρκαδόρων αντί παραγοντοποίησης του κανόνα.

Σύνοψη ενότητας

Στην ενότητα αυτή μελετήσαμε μεταφραστικά σχήματα δημιουργίας κώδικα για τους πιο κοινούς τύπους δομημένων εντολών που συναντώνται στις περισσότερες γλώσσες προγραμματισμού. Μια κεντρική παρατήρηση είναι ότι η βασική λειτουργία των δράσεων για τις εντολές αυτές είναι να εξασφαλίσουν το σωστό πισομπάλωμα.

ΕΝΟΤΗΤΑ 6.7 ΜΕΤΑΦΡΑΣΗ ΣΕ TOP DOWN ΑΝΙΧΝΕΥΣΗ

Θα συζητήσουμε το πως να εφαρμόσουμε τις ιδέες αυτού του κεφαλαίου σε Top-down αναλυτές όπως οι Recursive Descent. Στην πραγματικότητα όταν κάνουμε top-down ανίχνευση υπάρχουν περιπτώσεις που η χρήση μαρκαδόρων όπως το M για να κρατάμε τον αριθμό της τετράδας, ή η παραγοντοποίηση των κανόνων μπορεί να αποφεύγεται.

Το προτέρημα ενός top-down αναλυτή είναι ότι σημασιολογικές ρουτίνες μπορούν να καλούνται στο μέσον των κανόνων. Έτσι, αν ψάχνουμε για ένα A, και αναπτύσσουμε το A σύμφωνα με το κανόνα A->BCD, μπορούμε να καλέσουμε μια ρουτίνα αμέσως, μια άλλη αμέσως μετά την αναγνώριση του B, μια άλλη αμέσως μετά την αναγνώριση του C και μια άλλη μετά την αναγνώριση του D. Την συμπεριφορά αυτή μπορούμε να την επιτύχουμε και σ' ένα bottom-up περιβάλλον. Ένας απλός τρόπος για να αποκτήσουμε τον έλεγχο στο μέσον ενός κανόνα είναι να εισάγουμε μαρκαδόρους ως εξής:

A -> M1 B M2 C M3 D
M1 -> ε
M2 -> ε
M3 -> ε

Τότε, η ρουτίνα που καλείται από ένα top-down αναλυτή αμέσως μετά την ανάπτυξη του A, μπορεί να καλείται από τον bottom-up αναλυτή κατά την μείωση του M1, η ρουτίνα που καλείται μετά την αναγνώριση του B μπορεί να καλείται μετά την μείωση του M2 κλπ.

Ας δούμε τώρα πως μπορούμε να εισάγουμε σημασιολογικές δράσεις σ' ένα Recursive Descent αναλυτή.

Έστω η παρακάτω γραμματική για εντολές ελέγχου.

1. S ->	if E then S(1)	S __ εντολή
2.	| if E then S(1) else S(2)	E __ έκφραση με τιμή boolean
3.	| while E do S(1)	
4.	| begin L end	L __ λίστα εντολών
5.	| A	A __ εντολή εκχώρησης
6. L ->	L(1) ; S	
7.	| S	
8. A ->	id := E	

Δραστηριότητα 3 / Κεφ. 6

Στα πλαίσια της δραστηριότητας αυτής θέλουμε να εισάγουμε σημασιολογικές δράσεις στις ρουτίνες του Recursive Descent αναλυτή που αντιστοιχεί στην παραπάνω γραμματική. Παρατηρήστε ότι για να κάνουμε την γραμματική αυτή κατάλληλη για Recursive Descent αναλυτή πρέπει να κάνουμε μερικές αλλαγές (ουσιαστικά πρέπει να

την κάνουμε LL(1)) στους κανόνες (1), (2), (6) και (7). Ακόμη, για κάθε μη τερματικό σύμβολο θεωρούμε μια ρουτίνα η οποία ψάχνει την είσοδο και καταναλίσκει μια συμβολοσειρά που παράγεται από αυτό το μη-τερματικό. Η ρουτίνα θα δημιουργήσει ορισμένες τετράδες και θα επιστρέψει στην ρουτίνα από την οποία είχε κληθεί. Προσπαθήστε τώρα να μετασχηματίσετε την γραμματική και να γράψετε τις ρουτίνες και τις αντίστοιχες σημασιολογικές δράσεις χωρίς να κοιτάξετε το παρακάτω κείμενο.

Αριστερή παραγοντοποίηση του if στο S δίνει:

S -> if E then S TAIL

TAIL -> else S | ε

Κατόπιν απαλείφουμε την αριστερή αναδρομή στους κανόνες του L και έχουμε το

L -> L ; S | S

να γινεται

L -> S L'

L' -> ; S L' | ε

Παρακάτω δίνεται η ρουτίνα S. Για απλοποίηση της ρουτίνας έχουν παραλειφθεί εντολές οι οποίες αφαιρούν σύμβολα όπως if ή while από τη συμβολοσειρά εισόδου.

```
Procedure S ( ) ;
Switch (first input token)
   begin
      case  'if' :
            begin ( E.TRUE, E.FALSE):= E( ) ;
               /* E recognizes a Boolean expression and
               returns E.TRUE and E.FALSE */
            BACKPATCH(E.TRUE, NEXTQUAD) ;
            /* note that no marker nonterminal is needed, we can
               backpatch as soon as the expression is found. */
            if next input symbol is 'then' then S(1).NEXT:=S( )
            /* S returns S.NEXT */
            else ERROR( ) ;
            if next input symbol is 'else' then
               begin
                        GEN (goto -):   /* jump after S(2) */
                        BACKPATCH (E.FALSE, NEXTQUAD);
                        /* NEXTQUAD is now the beginning of the
               statement following the code for S(1) */
                        S(1).NEXT:=MERGE(S(1).NEXT,
                                 MAKELIST(NEXTQUAD-1));
                        S(2). NEXT:= S( );
                        /* S returns S.NEXT */
                        return MERGE (S(1).NEXT, S(2).NEXT)
                        /* S returns its own S.NEXT list */
```

```
                end
                else        /* no else part is present */
                            return MERGE (S(1).NEXT, E.FALSE)
        end /* 'if' case */
case 'while':
    begin
            QUAD:= NEXTQUAD;
            /* To remember the beginning quadraple for the expression */
            (E.TRUE, E.FALSE):= E ( ) ;
            BACKPATCH(E.TRUE, NEXTQUAD);
            if next input symbol is 'do' then
             begin
                 S(1).NEXT:= S ( ) ;
                 BACKPATCH(S(1).NEXT, QUAD);
                 GEN (goto QUAD);
                 /* S jumps to the code for E after it finishes */
                 return E.FALSE
                 /* S. NEXT for the while-statement is the list E. FALSE */
             end
            else ERROR ( )
    end /* 'while' case */
case 'begin':
    begin
       L.NEXT:= L ( )
       /* L Returns L.NEXT */
       return L.NEXT
    end    /* 'begin' case */
    case 'id' :    /* do not remove id from input string */
       begin
            A( ) ;
            /* search for assignment statement */
            return MAKELIST ( )
            /* return empty list for S. NEXT */
       end    /* 'id' case */
    default:    ERROR ( )
end         /* Switch ends here */
```

ΣΥΝΟΨΗ ΚΕΦΑΛΑΙΟΥ

Στο κεφάλαιο αυτό μελετήσαμε την Συντακτικά Κατευθυνόμενη Μετάφραση σαν μια σχεδιαστική μεθοδολογία δημιουργίας ενδιάμεσου κώδικα για ένα Μεταγλωττιστή ή ένα Διερμηνευτή. Είδαμε τις δυνατότητες της μεθόδου και την ευελιξία της στην δημιουργία κώδικα.

Μελετήσαμε τις διάφορες μορφές ενδιάμεσου κώδικα σαν κώδικα τριών διευθύνσεων, postfix και συντακτικά δένδρα. Είδαμε τους διάφορους τύπους κώδικα τριών διευθύνσεων (πχ A:=BopC, if ArelopB goto L, A:=B[I]) και πως αναπαρίστανται μέσω τετράδων και τριάδων, και τελικά επικεντρώσαμε την προσοχή μας στις τετράδες για την συνέχεια του κεφαλαίου.

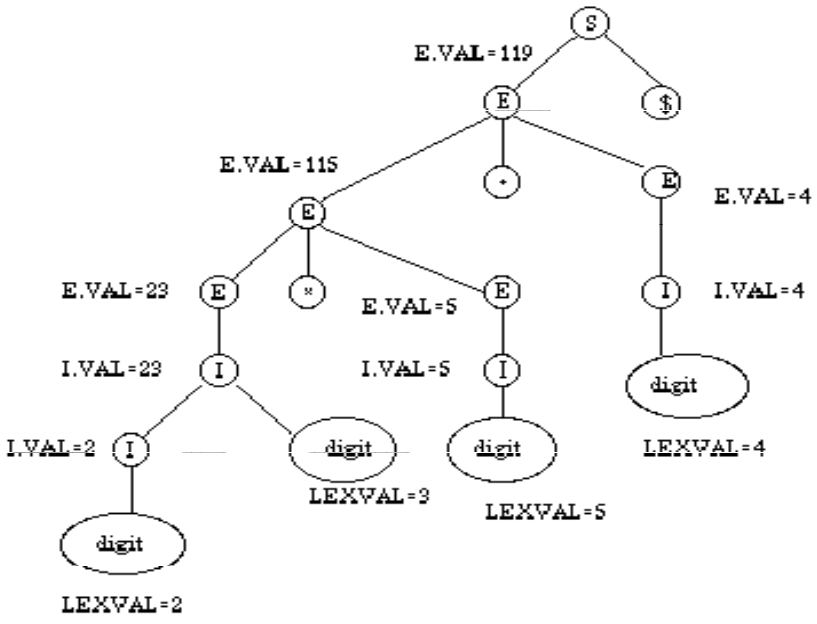
Μελετήσαμε την κατασκευή τετράδων για αριθμητικές και λογικές εκφράσεις, και εντολές καταχώρησης. Είδαμε τις παρεμβάσεις που πρέπει να γίνουν στον κώδικα όταν οι εκφράσεις περιέχουν περισσότερους από ένα τύπους δεδομένων και δώσαμε έμφαση στην αναπαράσταση των λογικών εκφράσεων μέσω ελέγχου ροής.

Τέλος, μελετήσαμε την μετάφραση (δημιουργία κώδικα) δομημένων εντολών τόσο για Bottom up (shift-reduce) αναλυτές (όπως είναι και οι LR(1)), όσο και για Top Down χωρίς οπισθοανίχνευση αναλυτές (όπως είναι οι Recursive Descent και οι LL(1)).

ΑΠΑΝΤΗΣΕΙΣ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ

Απάντηση άσκησης 1/ Κεφ. 6

Το δένδρο ανίχνευσης διακοσμημένο στους αντίστοιχους κόμβους με τις αντίστοιχες μεταφράσεις των E.VAL, I.VAL και LEXVAL δίνεται παρακάτω.



Αν φτιάξατε αυτό το δένδρο μπράβο σας. Αν όχι, δεν πειράζει, μελετήστε καλά αυτό το δένδρο και αφού ξαναδοκιμάσετε και το πετύχετε, δοκιμάστε και με κάποια άλλη είσοδο όπως η 7+31*2$.

Απάντηση άσκησης 2 / Κεφ. 6

Η έκφραση αυτή είναι μια φωλιασμένη μορφή εντολής if-then-else όπου το εσωτερικό if-then-else δείχνεται μέσα σε πλαίσιο.

if a then if c-d then a+c else a*c else a+b

Η postfix μορφή του εσωτερικού if-then-else είναι η :

cd- ac+ ac* ?

Και αντίστοιχα η postfix μορφή για ολόκληρη την έκφραση είναι η:

a cd-ac+ac*?ab+ ?

Απάντηση άσκησης 3 / Κεφ. 6

Παρατηρώντας προσεκτικά την μορφή του κώδικα που δείχνεται στο Σχήμα 6.14(α), μπορούμε να γράψουμε το παρακάτω ΣΚΜ σχήμα του κανόνα E -> id relop id το οποίο παράγει αυτόν τον κώδικα.

```
E -> id(1) relop id(2)      {  T:=NEWTEMP( ); E.PLACE:=T;
                               GEN (if id(1).PLACE relop id (2).PLACE
                                     goto NEXTQUAD+3);
                               GEN (T:=0);
                               GEN (goto NEXTQUAD+2);
                               GEN (T:=1)  }
```

Αν η απάντηση σας είναι ίδια με την παραπάνω συγχαρητήρια έχετε καταλάβει τον βασικό τρόπο δημιουργίας ενδιάμεσου κώδικα. Αν δεν τα καταφέρατε, μελετήστε την απάντηση και ξαναδοκιμάστε.

Απάντηση άσκησης 4 / Κεφ. 6

Οι Σημασιολογικές δράσεις για τους κανόνες 4 και 6 είναι οι παρακάτω

```
4. E -> E(1)
            {   E.TRUE:= E(1).TRUE; E.FALSE:= E(1).FALSE   }
```

```
6. E -> id(1) relop id(2)
            {   E.TRUE:= MAKELIST(NEXTQUAD);
                E.FALSE:= MAKELIST(NEXTQUAD + 1);
                GEN(if id(1).PLACE relop id(2).PLACE goto-)
                GEN(goto-)   }
```

Αν η απάντηση σας είναι ίδια με την παραπάνω συγχαρητήρια έχετε καταλάβει πως χρησιμοποιούμε τις λίστες E.TRUE και E.FALSE. Αν δεν τα καταφέρατε, δεν πειράζει μελετήστε πάλι την υποενότητα 6.4.2 και ξαναδοκιμάστε.

Απάντηση άσκησης 5 / Κεφ. 6

Υπόδειξη: Θεωρήστε ότι ο κώδικας που θα παραχθεί στην περίπτωση αυτή είναι της μορφής:

```
            V := E(1)
            goto ___ [L1}
[L2:]   V := V + E(2)
[L1:]   if  V > E(3) goto ___ {LOOP EXIT}
            { κώδικας του S(1) }
            . . .
```

goto - [L2]

{LOOP EXIT}**...**

Ακόμη θυμηθείτε ότι η αρχική γραμματική θα πρέπει να μετασχηματισθεί από :

S -> for V := E(1) step E(2) until E(3) do S(1)

στην:

F1 -> for V := E(1)
F2 -> F1 step E(2)
F3 -> F2 until E(3)
S -> F3 do S(1)

Απάντηση άσκησης 6 / Κεφ. 6

Υπόδειξη: Θεωρήστε και πάλι ότι ο κώδικας που θα παραχθεί στην περίπτωση αυτή είναι της μορφής:

V := E(1)

goto ___ [L1}

[L2:] V := V + E(2)

[L1:] if V > E(3) goto ___ {LOOP EXIT}

{ κώδικας του S(1) }

. . .

goto - [L2]

{LOOP EXIT}**...**

Τώρα θα πρέπει να εισάγετε μαρκαδόρους στον κανόνα

S -> for V := E(1) step E(2) until E(3) do S(1)

οι οποίοι πρέπει να κρατήσουν τις θέσεις της αρχής του κώδικα των E(2), E(3) και S(1).

Διαγραμματικά οι θέσεις των μαρκαδόρων και τα άλματα ροής δείχνονται στο παρακάτω διάγραμμα.