Compilers Report

Στοιχεία Μελών Ομάδας

Ονοματεπώνυμο: Θοδωρής Δήμας
Αριθμός Μητρώου: 2682

Ονοματεπώνυμο: Ελευθέριος Παναγιώτης Κωστάκης
Αριθμός Μητρώου: 2741

Πίνακας Περιεχομένων

Λεκτικός Αναλυτής

Στη φάση της λεκτικής ανάλυσης ο μεταγλωττιστής δέχεται στην είσοδο ένα αρχικό πρόγραμμα ως μια συμβολοσειρά χαρακτήρων και δίνει στην έξοδο το ίδιο πρόγραμμα αλλά ως μια συμβολοσειρά **λεκτικών μονάδων (tokens)**. Οι λεκτικές αυτές μονάδες είναι τα τερματικά σύμβολα της γραμματικής που περιγράφει τη σύνταξη του αρχικού προγράμματος στην επόμενη φάση του μεταγλωττιστή που είναι η φάση της συντακτικής ανάλυσης.

Οι λέξεις κλειδιά (**keywords**), τα ονόματα (**identifiers**), οι σταθερές (**constants**), οι τελεστές(**operators**) και οι διαχωριστές (**delimiters**) θεωρούνται συνήθως λεκτικές μονάδες. Η επιλογή των λεκτικών μονάδων είναι θέμα του σχεδιαστή ενός μεταγλωττιστή

Βέβαια, η φάση της λεκτικής ανάλυσης μπορεί να ενσωματωθεί στη φάση της συντακτικής ανάλυσης. Ο λόγος που στην πράξη υπάρχει η λεκτική ανάλυση ως ξεχωριστή φάση είναι γιατί κάνει τη σχεδίαση και την υλοποίηση του μεταγλωττιστή ευκολότερη.

Το πρόγραμμα που υλοποιεί τη φάση της λεκτικής ανάλυσης λέγεται "Λεκτικός Αναλυτής"(ΛΑ) (**Lexical analyzer**). Εκτός από την εξαγωγή των λεκτικών μονάδων από το αρχικό πρόγραμμα, ο λεκτικός αναλυτής μπορεί σε περίπτωση διαπίστωσης λάθους να καλεί το κομμάτι του μεταγλωττιστή που διαχειρίζεται τα λάθη, προκειμένου να αναφέρει ένα λεκτικό λάθος στον προγραμματιστή.

Έστω ότι έχουμε μια συνηθισμένη γλώσσα προγραμματισμού που περιλαμβάνει τις λεκτικές μονάδες >= , > , = . Τότε περιμένουμε η συμβολοσειρά χαρακτήρων ">=" , ως πρόθεμα στη συμβολοσειρά εισόδου, να μας δώσει τη λεκτική μονάδα που αντιστοιχεί στο >= . Όμως, από την παραπάνω συμβολοσειρά χαρακτήρων θα μπορούσαμε επίσης να πάρουμε μία λεκτική μονάδα για το > και μια για το = . Δηλαδή, από αυτή τη συμβολοσειρά μπορούν να αναγνωρισθούν τρεις διαφορετικές λεκτικές μονάδες ενώ στην πράξη απαιτείται να αναγνωρίζεται μια. Για την επίλυση του προβλήματος αυτού κρατάμε την μεγαλύτερη δυνατή αποδεκτή ποσότητα που μπορούμε.

Το τέλος μιας λεκτικής μονάδας μπορεί να αναγνωρισθεί με την επισκόπηση επιπλέον χαρακτήρων. Σ' αυτή την περίπτωση απαιτείται να ενημερώνεται ο δείκτης στη συμβολοσειρά εισόδου ώστε όταν αναγνωρισθεί μια λεκτική μονάδα αυτός να δείχνει στην αρχή της επόμενης. Αυτό βέβαια γίνεται με κατάλληλη **οπισθοδρόμηση** που σημαίνει ότι

κατά την εξαγωγή των λεκτικών μονάδων ορισμένοι χαρακτήρες στη συμβολοσειρά εισόδου επισκέπτονται περισσότερο από μία φορά.

Για την παράσταση των μεταβάσεων χρησιμοποιείται ένα διάγραμμα ειδικής μορφής που ονομάζεται "διάγραμμα μετάβασης" (**transition diagram**).

Ως παράδειγμα, αν θεωρήσουμε ότι θέλουμε να αναγνωρίσουμε τις επόμενες **λεκτικές μονάδες** στη **συμβολοσειρά εισόδου**:

ΠΑΡΑΔΕΙΓΜΑ

```
if (i == j)
        z = 0 ;
else
        z = 1 ;
```

Η **μορφή εισόδου** του κειμένου στον συντακτικό αναλυτή είναι η παρακάτω:

\t|if (i==j)\n\t\tz = 0;\n\telse\n\t\tz = 1 ;

Ο **διαχωρισμός** που οφείλει να γίνει είναι ο εξής:

\t|if|(|i|==|j|)|\n\t\t|z| |=| 0|;|\n\t|else|\n\t\t|z| |=| |1| ;|

Λίγα σχετικά **λάθη εντοπίζονται** κατά τη φάση της λεκτικής ανάλυσης και αυτό συμβαίνει επειδή ο ΛΑ έχει μόνο μια περιορισμένη τοπικά άποψη του αρχικού κώδικα. Τα λάθη αυτά δεν είναι σε θέση να προσδιορίσουν σημαντικά σφάλματα που κάνει ο προγραμματιστής όπως θα δούμε στην συνέχεια. Σε επόμενες φάσεις θα έχουμε την δυνατότητα κάποια ουσιαστικά λάθη να είμαστε σε θέση να του τα υποδείξουμε με μεγάλη ακρίβεια.

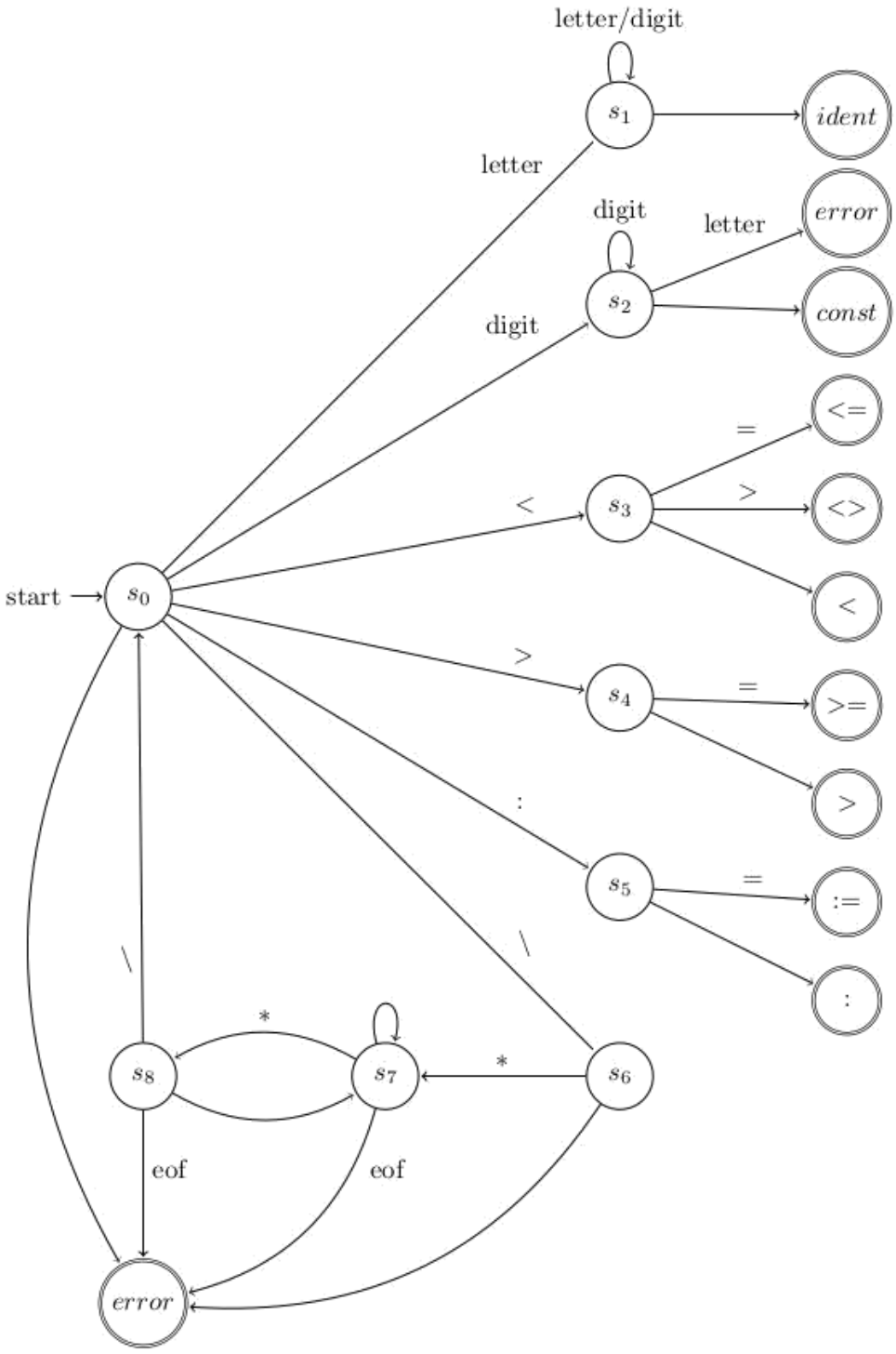
Το **παράδειγμα** που ακολουθεί διευκρινίζει το ζήτημα αυτό. Ας υποθέσουμε ότι η συμβολοσειρά if συναντάται σε ένα πρόγραμμα σε C για πρώτη φορά στην έκφραση:

fi (a == f (x))...

Ο ΛΑ δεν μπορεί να διακρίνει αν η συμβολοσειρά fi είναι τυπογραφικό λάθος της λέξης κλειδί if ή αν είναι αναγνωριστικό κάποιας συνάρτησης (που δεν έχει δηλωθεί). Επομένως, αφού fi είναι έγκυρο αναγνωριστικό, ο ΛΑ πρέπει να επιστρέψει τη λεκτική μονάδα που αντιστοιχεί σε αναγνωριστικό, αφήνοντας κάποια άλλη φάση της μεταγλώττισης να εντοπίσει και να χειριστεί το λάθος.

Εάν προκύψει μια κατάσταση στην οποία ο **ΛΑ αδυνατεί να προχωρήσει** γιατί καμία από τις μορφές που έχουν οι λεκτικές μονάδες δεν ταιριάζει με ένα πρόθεμα της συμβολοσειράς που έχει απομείνει στην είσοδο τότε η **στρατηγική** που ακολουθούμε είναι να διακόψουμε την ροή του και να εμφανίσουμε κατάλληλο μήνυμα λάθους.

Ένας καλός λεκτικός αναλυτής χρειάζεται μόνο ένα πέρασμα προκειμένου να εκπληρώσει την λειτουργία του και όχι κάποιες περιττές επαναλήψει


letter/digit
s_1
ident
letter
digit
letter
error
s_2
const
digit
=
<=
<
s_3
>
<>
<
start
s_0
>
s_4
=
>=
>
:
s_5
=
:=
:
\
\
*
*
s_8
s_7
s_6
eof
eof
error

Συντακτικός Αναλυτής

Στη φάση της συντακτικής ανάλυσης ελέγχεται αν ένα αρχικό πρόγραμμα ανήκει στη γλώσσα της οποίας η σύνταξη ορίζεται από μία ορισμένη γραμματική. Το κομμάτι του μεταγλωττιστή που κάνει αυτή τη δουλειά ονομάζεται συντακτικός αναλυτής(**parser**).

Η **είσοδος** ενός συντακτικού αναλυτή είναι ένα αρχικό πρόγραμμα (ως μία συμβολοσειρά λεκτικών μονάδων) και η **έξοδος** του ένα συντακτικό δένδρο ή μία ένδειξη ότι το αρχικό πρόγραμμα δεν είναι συντακτικά ορθό.

Λειτουργία του συντακτικού αναλυτή **LL(1)**

Το πρώτο βήμα είναι να θέσουμε το συντακτικό αναλυτή LL(1) στην αρχική του κατάσταση. Αυτό σημαίνει ότι (1) στη στοίβα βάζουμε $S όπου $ συμβολίζει το τέλος της συμβολοσειράς εισόδου και S το αρχικό σύμβολο της γραμματικής και (2) ο δείκτης της συμβολοσειράς εισόδου να δείχνει στο πρώτο σύμβολό της.

Μετά ο συντακτικός αναλυτής LL(1) εκτελεί βήματα μέχρι να αναγνωρίσει τη συμβολοσειρά εισόδου ή να βρει λάθος.

LL(1) Γραμματικές

LL(1) γραμματική: Αναγνώσιμη από ένα LL(1) ΣΑ (αναδρομικής κατάβασης ή ΑΣ). Υποσύνολο των γραμματικών χωρίς συμφραζόμενα, αρκετά πλούσιο όμως για να περιγράψει τις περισσότερες γλώσσες.

Προϋποθέσεις:

- Για κάθε ζεύγος κανόνων παραγωγής $A \to \alpha$ και $A \to \beta$ πρέπει να ισχύει $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$.
- Αν το ε ανήκει στο FIRST(A) (για παράδειγμα υπάρχει κανόνας $A \to \varepsilon$) πρέπει να ισχύει $FIRST(A) \cap FOLLOW(A) = \emptyset$.

Αποκλείονται οι γραμματικές που είναι:

- Αριστερά αναδρομικές
- Έχουν δύο εναλλακτικούς κανόνες με δεξιά μέλη που ξεκινούν με το ίδιο σύμβολο.
- Έχουν δύο εναλλακτικούς κανόνες τα δεξιά μέλη των οποίων παράγουν την κενή συμβολοσειρά.

Ένας πολύ εύκολος τρόπος για να διαπιστώσει κάποιος απευθείας αν η γραμματική μας είναι LL(1) είναι να ελέγξει αν είναι διφορούμενη. Αν είναι , δηλαδή για μία συμβολοσειρά υπάρχουν 2 ή και παραπάνω συντακτικές παραγωγές τότε διαπιστώνουμε πολύ εύκολα ότι η γραμματική μας δεν είναι LL(1).

Για να επιτύχουμε η γραμματική μας να είναι LL(1) κάνουμε τις εξής ενέργειες:

- Απαλοιφή αριστερής αναδρομής
- Αντικατάσταση
- Αριστερή παραγοντοποίηση

Ενδιάμεσος Κώδικας

Στη φάση αυτή ο μεταγλωττιστής **μετατρέπει** το αρχικό πρόγραμμα σε ένα άλλο ισοδύναμο πρόγραμμα που είναι γραμμένο σε μια ενδιάμεση γλώσσα. Οι λόγοι που συνηγορούν στην ύπαρξη της φάσης αυτής είναι:

1. Διευκολύνετε το έργο της μετάφρασης
2. Η βελτιστοποίηση του παραγόμενου κώδικα είναι αποτελεσματικότερη όταν γίνεται στον ενδιάμεσο κώδικα
3. Ένα μεγάλο κομμάτι του μεταγλωττιστή γίνεται ανεξάρτητο από τον υπολογιστή για τον οποίο παράγει τον τελικό κώδικα, με αποτέλεσμα να μπορούν να υλοποιηθούν γρηγορότερα και ευκολότερα μεταγλωττιστές για την ίδια γλώσσα προγραμματισμού που να τρέχουν σε διαφορετικούς υπολογιστές.

Με την τεχνική αυτή μπορούν να μεταφρασθούν σε μια ενδιάμεση γλώσσα οι βασικές δομές των γλωσσών προγραμματισμού, όπως είναι: οι αριθμητικές παραστάσεις, οι λογικές παραστάσεις, οι εντολές αντικατάστασης, οι εντολές ελέγχου κλπ. Ως **ενδιάμεση γλώσσα** θα χρησιμοποιηθεί εκείνη που ονομάζεται "**τετράδες**".

Κάθε τετράδα έχει την εξής μορφή: **label: op,op1,op2,op3**

Πιο συγκεκριμένα ο σχηματισμός αυτών θα είναι ως εξής:

jump,_,_,label
relop,s1,s2,label /* relop = { > , < , <>, <= , >=, = }
begin_block name,_,_
end_block name,_,_

Τα begin_block και end_block **δεν επιτρέπεται** να είναι εμφωλευμένα.

Halt,_,_,_ /* Σταματάει την εκτέλεση του προγράμματος μας
par,x,mode,_ /* mode = {CV,CP,REF}
call,name,_,_ /* Για την κλήση της συνάρτησης με όνομα name
ret,x,_,_ /* Αποτέλεσμα επιστροφής συνάρτησης

Κάποιες **βοηθητικές συναρτήσεις** που θα συνεισφέρουν στην παραγωγή του ενδιάμεσου κώδικα είναι οι παρακάτω:

nextquad():
Μας επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.

genquad(op,x,y,z):
Για την δημιουργία / παραγωγή τετράδας με τα δοθέντα ορίσματα.

newTemp():
Επιστρέφει το όνομα της επόμενης προσωρινής μεταβλητής. Χρησιμοποιείται κυρίως για την διάσπαση μια σύνθετης έκφρασης σε πιο απλές. Να τονίσουμε εδώ ότι η σειρά της προτεραιότητα των πράξεων μας καθορίζεται αυστηρά από την γραμματική μας.

emptyList():
Δημιουργία μίας άδειας λίστας.

makelist(x):
Δημιουργία μιας νέας λίστα που θα περιλαμβάνει μόνο την ετικέτα x.

mergelist(x,y):

Συνενώνει τις 2 λίστες σε 1. Έπειτα το πέρας της θα υπάρχει μόνο 1 λίστα. Στις περιπτώσεις που α αναλύσουμε αργότερα θα διαπιστώσουμε ότι η ροή/μεταβάσεις για κάποια διαφορετικά σημεία του κώδικα μας θα είναι κοινή.

backpatch(list,label):

Δέχεται μία λίστα και μία ετικέτα και συμπληρώνει το τελευταίο πεδίο κάθε τετράδας της λίστας με την ετικέτα label. Η χρησιμότητα της είναι τεράστια καθώς σε διάφορα σημεία του κώδικα δεν θα είμαστε σε θέση να προσδιορίσουμε άμεσα την μετάβαση από μία τετράδα σε μία άλλη. Η μέθοδος αυτή θα μας προσφέρει την δυνατότητα συμπλήρωσης σε κάποιο μεταγενέστερο χρονικό σημείο.

Είναι φανερό ότι με το πλαίσιο αυτό η φάση της **συντακτικής ανάλυσης** και η φάση της **παραγωγής του ενδιάμεσου κώδικα** γίνονται **παράλληλα**.

Σε κάθε σύμβολο της γραμματικής μπορούμε να αντιστοιχίσουμε μία ή περισσότερες μεταβλητές που αντιπροσωπεύουν ιδιότητες αυτού του συμβόλου και θα λέγονται "μεταβλητές ιδιοτήτων". Οι μεταβλητές ιδιοτήτων του συμβόλου X θα συμβολίζονται ως X.PLACE, X.FALSE κλπ. Οι μεταβλητές ιδιοτήτων είναι πολύ χρήσιμες, όπως θα φανεί παρακάτω, και χρησιμοποιούνται από τις σημαντικές ρουτίνες.

Παραγωγή Ενδιάμεσου Κώδικα για Γλωσσικές Δομές

$E \rightarrow \mathbf{T^{(1)} (+ T^{(2)} \{P1\})^* \{P2\}}$

{P1} : {w = newTemp()
genquad('+', $T^{(1)}$.PLACE,$T^{(2)}$.PLACE,w);
$T^{(1)}$.PLACE = w}

{P2} E.PLACE = $T^{(1)}$.PLACE /* Σημαντικό για την σωστή ανανέωση της μεταβλητής

Αντίστοιχα ακριβώς το ίδιο αν είχαμε '*'.

F → (E) {P1}
{P1} : {F.PLACE = E.PLACE}

F → (id) {P1}
{P1} : {F.PLACE = id.PLACE}

Σε αντίθεση με τις αριθμητικές παραστάσεις που φρόντιζαν για την σωστή απόδοση τιμών, οι αριθμητικές παραστάσεις καθορίζουν τις σωστή μετάβαση/ροή στο πρόγραμμα μας.

Για τον σκοπό αυτό θα κάνουμε συχνή χρήση των μεταβλητών ιδιοτήτων FALSE και TRUE που είχαμε προαναφέρει.

B→ $Q^{(1)}$ {P1} (or {P2} $Q^{(2)}$ {P3})*

{P1} : {BTrue = $Q^{(1)}$True
Bfalse = $Q^{(1)}$False}

{P2} : {backpatch(BFalse , nextquad())}
/* Κάθε false μας υποδεικνύει να πάμε στον επόμενο

έλεγχο {P3} : {BFalse = $Q^{(2)}$False

/* Ανανέωση λίστας για να μην χαθεί
Btrue = merge(BTrue,$Q^{(2)}$True}
/* Τα true δεν γνωρίζουμε ακόμα σε ποιο σημείο κώδικα θα μας οδηγήσουν, επίσης είναι αυτά που ακόμα και 1 να ισχύει μας οδηγούν έξω.

B→ $Q^{(1)}$ {P1} (and {P2} $Q^{(2)}$ {P3})*

{P1} : {BTrue = $Q^{(1)}$True
Bfalse = $Q^{(1)}$False}

{P2} : {backpatch(BTrue , nextquad())}
/* Κάθε True μας υποδεικνύει να πάμε στον επόμενο έλεγχο

{P3} : {BTrue = $Q^{(2)}$True
/* Ανανέωση λίστας για να μην χαθεί
BTrue = merge(BFalse,$Q^{(2)}$False}
/* Τα false δεν γνωρίζουμε ακόμα σε ποιο σημείο κώδικα θα μας οδηγήσουν, επίσης είναι αυτά που ακόμα και 1 να είναι έτσι μας οδηγούν έξω.

R → (B) {P1}

{P1}: {Rfalse = Bfalse
Rtrue = Btrue}

R →not (B) {P1}

{P1} {Rtrue = Bfalse
Rfalse = Btrue}

R →$E^{(1)}$ exp $E^{(2)}$ {P1}

{P1} : {Rtrue = makelist(nextquad())
genquad(exp,$E^{(1)}$.PLACE,$E^{(2)}$.PLACE,_)
Rfalse = makelist(nextquad())
genquad(jump,_,_,_)

S →while {P0} (cond)
{P1} <sequence> {P2}
endwhile

Ένα while γνωρίζει που θα κάνει jump , ανάλογα με το αν ισχύει η δεν ισχύ η συνθήκη ελέγχου του. Επίσης όταν φτάσει στο τέλος το while ξανακάνει jump πίσω ώστε να ξαναγίνει έλεγχος της συνθήκης.

{P0}:{firstquad = nextquad() }
/* Σημειώνω την επόμενη που θα δημιουργηθεί ώστε να γνωρίζω που βρίσκεται

{P1}{backpatch(condTrue,nextquad())}

{P2}: {genquad(jump,_,_,firstquad)
backpatch(condfalse,nextquad())

S → if B then {P1} $S^{(1)}$ {P2} TAIL {P3}
TAIL →else $S^{(2)}$ | ε

{P1}backpatch(Btrue,nextquad())

{P2}:{iflist = makelist(nextquad())
genquad(jump,_,_,_)
backpatch(Bfalse,nextquad())

{P3} backpatch(iflist,nextquad())
/* Υπάρχει ανάγκη κάτι να με βγάλει εκτός

S → repeat {P1} S until (cond) {P2}

{P1} fquad nextquad()

{P2} : {backpatch(condfalse,fquad)
backpatch(condtrue,nextquad())

/* Στην περίπτωση που έχουμε κενό εσωτερικά και δημιουργηθεί ατέρμονος βρόχος είναι απόλυτα σωστό

S → id := e {P1}
genquad(:=,E.PLACE,_,id)

S → return e {P1}
genquad(retv,_,_e.PLACE)

S → input id
genquad(inp,_,_id)

S → print e
genquad(out,_,_e.PLACE)

{P0}switch_id(id$^{(1)}$)
{
(case id$^{(1)}$ {P1}: S$^{(1)}$ {P2} break)*
default: S$^{(2)}$ {P3}
}

{P0} exitlist = emptylist()

{P1} tlist = makelist(nextquad())
genquad('<>',id.PLACE,E.PLACE,_)

{P2} t = makelist(nextquad())
genquad(jump,_,_,_)
merge(exitlist,t)
backpatch(tlist,nextquad())

{P3} backpatch(exitlist(nextquad())

{P0}switch_cond
{
((cond): {P1} S$^{(1)}$ break {P2})*
default: S$^{(2)}$ {P3}
}

{P0} exitlist = emptylist()

{P1} backpatch(condTrue,nextquad())

{P2} t = makelist(nextquad())
genquad(jump,_,_,_)
merge(exitlist,t)
backpatch(condFalse,nextquad())

{P3} backpatch(exitlist(nextquad())

{P0} incase ((cond) {P1} $S^{(1)}$ {P2})*
default {P3} $S^{(2)}$

Πάντα φτάνουμε στο cond το επόμενο και δεν ξέρουμε αν έχει ισχύσει ή όχι.
Πρέπει την πληροφορία για τον αν ισχύει ή όχι να την κρατάμε πάνω στο cond σε χρόνο εκτέλεσης και όχι μετάφρασης.

{P0} : t= newTemp()
flagquad = nextquad()
genquad(:=,0,_,t)

{P1} : backpatch(condTrue,nextquad())
/* Αφού έχω φτάσει μέχρι εδώ σημαίνει ότι ισχύει 1 τουλάχιστον συνθήκη

genquad(:=,1,_,t)

{P2} backpatch(condfalse,nextquad())

{P3} genquad(:=,1,t,flagquad)
/* Jump για πίσω

dowhile{P1}
<statements>
enddowhile(<condition>){P2}

Αυτή η περίπτωση έρχεται σε αντιστοιχία με την repeat-until που έχουμε προαναφέρει.

Εφόσον είναι True η condition πρέπει να επιστρέψουμε στην αρχή αλλιώς οδηγούμαστε στην έξοδο.

{P1} firstquad = nextquad()

{P2} backpatch(conditionTrue,firstquad())
backpatch(conditionFalse,nextquad())

incase{P1}
(when (<condition>) : {P2} <statements>{P3})*
{P4} endincase

{P1} w = newTemp()
firstquad = nextquad()
genquad(:=,0,_,w)

{P2} backpatch(condTrue,nextquad())
genquad(:=,1,_,w)

{P3}backpatch(condFalse,nextquad())

{P4}genquad(=,1,w,firstquad)
forecase {P1}
(when(<condition>) : {P2} <statements> {P3})*
default : <statements> {P4} enddefault
endforcase {P5}

{P1} exitlist = emptylist()
firstquad = nextquad()

{P2} backpatch(condTrue,nextquad())

{P3} exit = makelist(nextquad())
genquad(jump,_,_,_)
backpatch(condFalse,nextquad())
exitlist = merge(exitlist,exit)

{P4} genquad(jump,_,_,firstquad)

{P5} backpatch(exitlist,nextquad())

{P1} loop <statements {P3} endloop {P4}
<statements> →exit{P2}

Η έξοδος από το βρόχο γίνεται όταν κληθεί η εντολή exit.
Μπορεί να μην υπάρχει exit. Τότε πρέπει να υπάρξει ατέρμον βρόχος που να πιστοποιεί την σωστή λειτουργία του κώδικα μας.

{P1} firstquad = nextquad()
exitlist = emptylist()

{P2} exit = meakelist(nextquad())
genquad(jump,_,_,_)
merge(exitlist,exit)

{P3} genquad(jump,_,_,firstquad)

{P4} backpatach(exitlist,nextquad())

Πίνακας Συμβόλων

Ένας μεταγλωττιστής χρειάζεται να συγκεντρώνει και να χρησιμοποιεί πληροφορίες για τα ονόματα που εμφανίζονται στο αρχικό πρόγραμμα. Οι πληροφορίες αυτές τοποθετούνται στον πίνακα συμβόλων (**symbol table**).

Κάθε στοιχείο του πίνακα συμβόλων είναι ένα ζευγάρι της μορφής (όνομα, πληροφορίες). Κάθε φορά που συναντιέται ένα όνομα, ο πίνακας συμβόλων προσπελαύνεται για να εξακριβωθεί αν το όνομα είναι καινούργιο - οπότε και εισέρχεται στον πίνακα - ή αν έχει ήδη συναντηθεί και τοποθετηθεί μέσα στον πίνακα. Οι πληροφορίες για τα ονόματα τοποθετούνται κατά τη συντακτική ανάλυση.

Οι πληροφορίες που υπάρχουν στον πίνακα συμβόλων χρησιμοποιούνται σε διάφορα στάδια κατά τη διάρκεια της μετάφρασης. Για παράδειγμα κατά τη σημασιολογική ανάλυση, όπου ελέγχεται αν η χρήση των ονομάτων συμφωνεί με τον τύπο τους (άμεσα ή έμμεσα δηλωμένο). Ακόμη χρησιμοποιείται κατά την παραγωγή κώδικα, όπου πρέπει να είναι γνωστό πόση και τι είδους μνήμη πρέπει να κρατηθεί για ένα όνομα.

Ο πίνακας συμβόλων αποτελείται από:

- **Entities** (Μεταβλητή, Σταθερά , Συνάρτηση)
- **Arguments** (Ορίσματα Συναρτήσεων)
- **Scopes** (Δηλώνουν την εμβέλεια και είναι σε θέση να υποδείξουν αν ένα αντικείμενο (συνάρτηση, μεταβλητή κοκ) μπορεί αν έχει πρόσβαση κάπου αλλού)

Ο πίνακας συμβόλων είναι ένας πίνακας με δύο πεδία : το πεδίο ονόματος και το πεδίο πληροφοριών. Πρέπει να είναι κατασκευασμένος έτσι ώστε να μπορούμε να :

- αποφασίζουμε αν κάποιο όνομα είναι στον πίνακα
- προσθέτουμε ένα καινούργιο όνομα στον πίνακα
- παίρνουμε τις πληροφορίες που σχετίζονται με κάποιο όνομα
- προσθέτουμε πληροφορίες για κάποιο όνομα
- διαγράφουμε ένα όνομα ή μια ομάδα από ονόματα από τον πίνακα.

Οι πληροφορίες που αφορούν ένα όνομα και πρέπει να βρίσκονται μέσα στον πίνακα συμβόλων είναι :

1. Η συμβολοσειρά χαρακτήρων του ονόματος. Αν το ίδιο όνομα μπορεί να χρησιμοποιηθεί και σε άλλα τμήματα του προγράμματος τότε πρέπει να υπάρχει κάποια ένδειξη για το τμήμα του προγράμματος στο οποίο ανήκει το όνομα αυτό.

2. Τα χαρακτηριστικά του ονόματος όπως π.χ. ο τύπος του (ακέραιος, πραγματικός κ.λπ.) και το είδος του (παράμετρος, μεταβλητή, υπό-πρόγραμμα, ετικέτα, κ.λπ.).

3. Μια απόκλιση (offset) που δίνει τη σχετική θέση ενός ονόματος ως προς μια αρχή.

Οι πληροφορίες αυτές τοποθετούνται στον πίνακα σε διάφορες στιγμές. Τα χαρακτηριστικά π.χ. των ονομάτων τοποθετούνται όταν βρεθούν οι δηλώσεις τους (declarations). Κάθε όνομα πρέπει να διατηρείται μέσα στον πίνακα συμβόλων μέχρι να μην είναι δυνατόν να γίνει άλλη αναφορά σ ‘αυτό.

Σημασιολογική Ανάλυση

Ένας μεταγλωττιστής πρέπει να ελέγχει ότι το αρχικό πρόγραμμα ακολουθεί τόσο τις συντακτικές όσο και τις σημασιολογικές συμβάσεις της αρχικής γλώσσας. Αυτός ο έλεγχος, ο οποίος λέγεται στατικός έλεγχος (static checking), διασφαλίζει ότι θα διαγνωστούν και θα αναφερθούν συγκεκριμένα είδη προγραμματιστικών λαθών. Αντίθετα, ο δυναμικός έλεγχος (dynamic checking) γίνεται κατά το χρόνο εκτέλεσης και με αυτόν γίνεται διάγνωση λαθών τα οποία δεν είναι δυνατόν να εντοπισθούν κατά τη μετάφραση..

Στατικοί έλεγχοι αποτελούν οι ακόλουθοι:

1. Έλεγχοι Τύπων (type checks): Ο μεταγλωττιστής θα πρέπει να αναφέρει ένα λάθος, όποτε ένας τελεστής εφαρμόζεται σε μη επιτρεπόμενα τελούμενα. Για παράδειγμα, όταν μία μεταβλητή-πίνακας προστίθεται σε μια μεταβλητή-συνάρτηση.

2. Έλεγχοι Μοναδικότητας : Υπάρχουν περιπτώσεις στις οποίες ένα αντικείμενο πρέπει να ορίζεται ακριβώς μια φορά και να χρησιμοποιείται ακριβώς με τον τρόπο που ορίστηκε (ως συνάρτηση, μεταβλητή , κοκ).

3 Έλεγχοι Ονομάτων : Μερικές φορές το ίδιο όνομα δεν πρέπει να εμφανίζεται σε δύο ή περισσότερες θέσεις ίδιας εμβέλειας.

4.Έλεγχοι Επιστροφής : Κάθε συνάρτηση θα πρέπει να είναι σε θέση να επιστρέφει μία τιμή, αλλιώς θα την ονομάζουμε διαδικασία(δεν υπάρχει στον μεταγλωττιστή μας).

5.Έλεγχοι ορισμάτων συναρτήσεων: Η κλήση κάθε συνάρτησης οφείλει να συνοδεύεται από τον ίδιο αριθμό ορισμάτων καθώς και τον ίδιο τρόπο περάσματος με την δήλωση, διαφορετικά είναι λάθος.

Οι Έλεγχοι Ροής και Ονομάτων μπορούν να ενσωματωθούν στη Συντακτική Ανάλυση. Οι Έλεγχοι Μοναδικότητας γίνονται με τη βοήθεια του **Πίνακα Συμβόλων**.

Τελικός Κώδικας

Η σπουδαιότερη και δυσκολότερη φάση της μετάφρασης ενός προγράμματος είναι η φάση της παραγωγής του τελικού κώδικα. Η **είσοδος** στο γεννήτορα τελικού κώδικα είναι το πρόγραμμα σε μορφή ενδιάμεσου κώδικα και η **έξοδος** είναι ένα ισοδύναμο πρόγραμμα σε μορφή τελικού κώδικα.

Τα τοπικά δεδομένα ενός υπό-προγράμματος αποθηκεύονται σε ένα κομμάτι μνήμης που ονομάζεται "**εγγράφημα δραστηριοποίησης**" (ΕΔ) . Το ΕΔ χρειάζεται κατά τη διάρκεια της εκτέλεσης του υπό-προγράμματος.

Ένα ΕΔ έχει πεδία για την αποθήκευση παραμέτρων, αποτελεσμάτων, τοπικών μεταβλητών, προσωρινών μεταβλητών κλπ.

Το μέγεθος και η μορφή ενός ΕΔ καθορίζεται από τις πληροφορίες για τα ονόματα που βρίσκονται στον πίνακα συμβόλων. Στη συνέχεια θα θεωρείται ότι οι πληροφορίες που θα αποθηκεύονται σε ένα ΕΔ είναι οι εξής: στις πρώτες θέσεις οι πραγματικές παράμετροι της αντίστοιχης κλήσης του υπό-προγράμματος. Στη συνέχεια, σε ένα σταθερό τμήμα, με την έννοια ότι είναι κοινό για όλες τις κλήσεις υπό-προγραμμάτων, αποθηκεύονται η διεύθυνση του αποτελέσματος, ο σύνδεσμος προσπέλασης ή η τιμή του προηγούμενου δείκτη, στον πίνακα δεικτών και η διεύθυνση επιστροφής. Τέλος, αποθηκεύονται οι τοπικές μεταβλητές και οι προσωρινές μεταβλητές.

Η μνήμη του υπολογιστή, κατά το χρόνο εκτέλεσης, χωρίζεται σε δύο περιοχές. Η πρώτη περιοχή χρησιμοποιείται για την αποθήκευση του τελικού κώδικα που αντιστοιχεί στους τελεστές του αρχικού προγράμματος. Για ένα ορισμένο πρόγραμμα, το μέγεθος της περιοχής αυτής είναι σταθερό και γνωστό κατά τη μετάφραση. Η δεύτερη περιοχή χρησιμοποιείται για την αποθήκευση των ΕΔ, δηλαδή των τελούμενων του αρχικού προγράμματος. Τα ΕΔ τοποθετούνται σ' αυτή την περιοχή κατά την εκτέλεση του προγράμματος. Πρώτα τοποθετείται το ΕΔ του κυρίου προγράμματος. Μετά, κάθε φορά που καλείται ένα υπό-πρόγραμμα , τοποθετείται το αντίστοιχο ΕΔ στον ελεύθερο χώρο που υπάρχει σ’ αυτή την περιοχή και παραμένει εκεί έως ότου τελειώσει η εκτέλεση του υπό-προγράμματος αυτού οπότε και το αντίστοιχο ΕΔ απομακρύνεται από αυτή την περιοχή. Το μήκος αυτής της περιοχής αυξομειώνεται κατά την εκτέλεση σύμφωνα με τις δραστηριοποιήσεις των υπό-προγραμμάτων, γι' αυτό οργανώνεται ως μια στοίβα που λέγεται "στοίβα εκτέλεσης".

Η θέση του ΕΔ ενός υπό-προγράμματος δεν είναι γνωστή παρά μόνο κατά την εκτέλεση. Η αρχική διεύθυνση του ΕΔ αποθηκεύεται σε έναν καταχωρητή, ώστε οι θέσεις μέσα στο ΕΔ να μπορούν να προσπελασθούν σχετικά με την τιμή αυτή του καταχωρητή.

Όταν καλείται ένα υπό-πρόγραμμα, το υπό-πρόγραμμα που καλεί (caller) σπρώχνει (push) στη στοίβα τις πραγματικές παραμέτρους και τα στοιχεία του σταθερού τμήματος του ΕΔ μέχρι τη διεύθυνση επιστροφής και μεταφέρει τον έλεγχο στο καλούμενο υπό-πρόγραμμα (called). Όταν ο έλεγχος επιστρέφει, το υπό-πρόγραμμα που κάλεσε αφαιρεί από τη στοίβα εκτέλεσης το τμήμα του ΕΔ του καλούμενου υπό=προγράμματος που αυτό δημιούργησε. Δηλαδή, η κλήση ενός υπό-προγράμματος τοποθετεί δεδομένα στη στοίβα εκτέλεσης και μεταφέρει τον έλεγχο στο καλούμενο υπό-πρόγραμμα.

Πέρασμα παραμέτρων

Όταν ένα υπό-πρόγραμμα καλεί ένα άλλο, οι τρόποι που επικοινωνούν μεταξύ τους είναι συνήθως δύο (α) δια μέσου μη-τοπικών μεταβλητών και (β) δια μέσου παραμέτρων. Οι παράμετροι ενός υπό-προγράμματος λέγονται "τυπικές παράμετροι"(formal). Οι παράμετροι που αντικαθιστούν τις τυπικές κατά την κλήση λέγονται "πραγματικές παράμετροι" (actual).

Οι πιο συνηθισμένοι τρόποι αντιστοιχίας μεταξύ των πραγματικών και των τυπικών παραμέτρων είναι : κλήση-με-τιμή (call-by-value), κλήση-με- αναφορά (call-by-reference), κλήση-με-όνομα (call-by-name), αντιγραφή (copy)

Παρακάτω περιγράφονται μόνο οι μέθοδοι (α) κλήση-με-τιμή και (β) κλήση- με-αναφορά. (γ) κλήση -με- αντιγραφή.

Α. Κλήση-με-τιμή

Αυτός ο τρόπος αντιστοιχίας μεταξύ τυπικών και πραγματικών παραμέτρων είναι ίσως ο απλούστερος. Σύμφωνα με αυτόν, οι πραγματικές παράμετροι υπολογίζονται και η τιμή τους περνά στο καλούμενο υπό-πρόγραμμα. Στην στοίβα της συνάρτησης που δημιουργείται έχουμε αντιγράψει τις τιμές των πραγματικών παραμέτρων (μεταβλητών). Έπειτα οποιαδήποτε αλλαγή πραγματοποιείται εντός της συνάρτησης αλλάζει τις τιμές των μεταβλητών στην στοίβα της συνάρτησης και έτσι οι μεταβλητές θα υπόκεινται σε αλλαγές μόνο εντός της συνάρτησης. Εκτός της συνάρτησης οι πραγματικές παράμετροι δεν έχουν υποστεί κάποια αλλαγή.

Β. Κλήση-με-αναφορά

Στην κλήση-με-αναφορά το υπό-πρόγραμμα που καλεί περνά στο καλούμενο για κάθε πραγματική παράμετρο τη διεύθυνσή της. Η διεύθυνση αυτή τοποθετείται στο ΕΔ στη θέση της αντίστοιχης παραμέτρου. Η προσπέλαση , λοιπόν, στην τιμή μιας τέτοιας παραμέτρου ή η αποθήκευση σ’ αυτήν μιας τιμής πρέπει να γίνεται με έμμεσο τρόπο. Αυτό σημαίνει ότι σε αντίθεση με πριν δεν θα αντιγράψουμε και αποθηκεύσουμε στην στοίβα τις τιμές αλλά τις διευθύνσεις των πραγματικών παραμέτρων. Έτσι κάθε σε κάθε αλλαγή που θα κάνουμε για κάθε συνάρτηση που το πέρασμα της είναι με αυτό τον τρόπο θα έχουμε πρόσβαση στην διεύθυνση της και από εκεί έπειτα στο περιεχόμενο της ώστε να αποθηκεύουμε την οποιαδήποτε αλλαγή.

Γ. Κλήση-με-αντιγραφή

Στην κλήση-με-αντιγραφή οι τιμές των παραμέτρων αρχικά αντιγράφονται (όπως στην κλήση με τιμή) κατά την εκτέλεση του υπό-προγράμματος λειτουργούν ως τοπικές μεταβλητές και στο τέλος αυτού αποθηκεύονται στις πραγματικές παραμέτρους ως οι νέες τιμές που παρέμειναν στο τέλος.

Πριν ξεκινήσει να γίνει η κλήση της μεθόδου. Συγκεκριμένα από την οπτική γωνία του ενδιάμεσου κώδικα – σημείο στο οποίο οι μεταβλητές που θα αποτελούν τις πραγματικές παραμέτρους για την κλήση της συνάρτησης πρέπει οι μεταβλητές που περνιούνται με αντιγραφή να διαχειριστούν σαν να περνούν με τιμή. Δηλαδή στην στοίβα της συνάρτησης μπορεί να πραγματοποιηθούν αλλαγές σε αυτές τις μεταβλητές . Οι αλλαγές αυτές όπως και στο πέρασμα με τιμή αποθηκεύονται στον αντίστοιχο χώρο που έχει δεσμευτεί γι'αυτές τις μεταβλητές στην στοίβα. Όταν διαπιστώσουμε πως η συνάρτηση ολοκληρώνεται , σημείο στον ενδιάμεσο κώδικα που γίνετε το call και πριν μετακινήσουμε τον δείκτη στοίβας πριν την καλούσα οφείλουμε για κάθε μία από αυτές τις μεταβλητές που περάστηκαν με αντιγραφή να κάνουμε τις εξής ενέργειες. Για κάθε μία τυπική παράμετρο να αναζητήσουμε στην στοίβα της συνάρτησης που ήταν να φορτώσουμε σε έναν καταχωρητή το περιεχόμενο που διατηρεί αυτή στην στοίβα μας (προσωρινή μεταβλητή). Έπειτα να αποθηκεύσουμε αυτή την τιμή στην διεύθυνση μνήμης όπου είναι η πραγματική παράμετρος. Ενδεικτικά σε assembly mips θα πρέπει σε εκείνο το σημείο για κάθε μία εξ αυτών των μεταβλητών να έχουμε τα παρακάτω:

lw $t0,-offset($fp)
/* Ευρύτερα **loadvr(v_typical_par,$t0)** – Εύρεση τιμής στην στοίβα */
storerv($t0,v_actual_par)
/* Αποθήκευση πίσω στην θέση μνήμης της πραγματικής παραμέτρου. */

Βοηθητικές Συναρτήσεις για την υλοποίηση:

gnlvcode(a)
Παίρνει σαν είσοδο το όνομα α μιας μεταβλητής και μεταφέρει στον $t0 το όνομα μιας μη τοπικής μεταβλητής.

Χρήσιμη θα είναι όταν θα ελέγχουμε αν ένα βάθος φωλιάσματος είναι ή όχι ίσο με το τρέχον. Σε περίπτωση που δεν είναι, είναι απαραίτητη η χρήση της, γιατί πλέον δεν θα κάνουμε αναζήτηση στην στοίβα μας.

loadvr(v,r)

Μεταφέρει δεδομένα στον καταχωρητή r. Η μεταφορά μπορεί σε αυτή την περίπτωση να γίνει από τη μνήμη(στοίβα).Διακρίνουμε περιπτώσεις;

Αν v είναι σταθερά
li $tr,v

Αν v είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα
lw $tr,-offset($s0)

Αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή (Βρίσκεται στην στοίβα μας)
lw $tr,-offset($sp)

Αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον (Βρίσκεται στην στοίβα μας)
lw $t0,-offset($sp)
lw $tr,($t0)

Αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον.
gnlvcode()
lw $tr,($t0)

Αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον.
gnlvcode()
lw $t0,($t0)
lw $tr,($t0)

storerv(r,v)

Γίνετε μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v). Διακρίνουμε περιπτώσεις:

Αν v είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα
sw $tr,-offset($s0)

Αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή
sw $tr,-offset($sp)

Αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον

lw $t0,-offset($sp)
sw $tr,($t0)

Αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον.
gnlvcode(v)
sw $tr,($t0)

Αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον.
gnlvcode(v)
lw $t0,($t0)
sw $tr,($t0)

Εντολές Αλμάτων

jump, “_”, “_”, label
b label

relop(?),x,y,z
loadvr(x,$t1)
loadvr(y, $t2)
branch(?),$t1,$t2,z όπου branch(?) : beq,bne,bgt,blt,bge,ble

Εκχώρηση

:=, x, “_”, z
loadvr(x,$t1)
storerv($t1,z)

Εντολές Αριθμητικών Πράξεων

op x,y,z
loadvr(x, $t1)
loadvr(y, $t2)
op $t1,$t1,$t2 op: add,sub,mul,div
storerv($t1,z)

Εντολές Εισόδου - Εξόδου

out “_”, “_”, x
li$v0,1
loadvr(x,$a0)
syscall

in “_”, “_”, x
li$v0,5
syscall
storerv($v0,x)

Επιστροφή τιμής συνάρτησης

retv“_”, “_”, x
loadvr(x, $t1)
lw$t0,-8($sp)
sw$t1,($t0)

αποθηκεύεται ο x στη διεύθυνση που είναι αποθηκευμένη στην 3ηθέση του εγγραφήματος δραστηριοποίησης

Παράμετροι Συνάρτησης

Πριν κάνουμε τις ενέργειες ώστε να γίνει το **πέρασμα της πρώτης παραμέτρου**, τοποθετούμε τον $fp να δείχνει στην στοίβα της συνάρτησης που θα δημιουργηθεί
addi $fp,$sp,framelength .Στη συνέχεια, **για κάθε παράμετρο** και ανάλογα με το αν περνά με τιμή ή αναφορά ή αντιγραφή κάνουμε τις εξής ενέργειες:

par,x,CV, _ & par,x,CP, _
loadvr(x, $t0)
sw$t0, -(12+4i)($fp) , όπου i ο αύξων αριθμός της παραμέτρου

par,x,REF, _
Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το **ίδιο βάθος** φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με **τιμή ή αντιγραφή**
addi$t0,$sp,-offset
sw$t0,-(12+4i)($fp)

par,x,REF, _
Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το **ίδιο βάθος** φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με **αναφορά**
lw$t0,-offset($sp)
sw$t0,-(12+4i)($fp)

par,x,REF, _
Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν **διαφορετικό βάθος** φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με **τιμή ή αντιγραφή**
gnlvcode(x)
sw$t0,-(12+4i)($fp)

par,x,REF, _
Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν **διαφορετικό βάθος** φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με **αναφορά**
gnlvcode(x)
lw$t0,($t0)
sw$t0,-(12+4i)($fp)

par,x,RET, _
Γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα **επιστραφεί** η τιμή
addi$t0,$sp,-offset
sw$t0,-8($fp)

Κλήση Συνάρτησης

call, _, _, f
Αρχικά γεμίζουμε το 2ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης, **τον σύνδεσμο προσπέλασης**, με την διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα της, ώστε η κληθείσα να γνωρίζει που να κοιτάξει αν χρειαστεί να προσπελάσει μία μεταβλητή την οποία έχει δικαίωμα να προσπελάσει, αλλά δεν της ανήκει.

Αν καλούσα και κληθείσα έχουν το **ίδιο βάθος** φωλιάσματος, τότε έχουν τον ίδιο γονέα
lw$t0,-4($sp)
sw$t0,-4($fp)

Αν καλούσα και κληθείσα έχουν **διαφορετικό βάθος** φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας
sw$sp,-4($fp)

στη συνέχεια **μεταφέρουμε τον δείκτη στοίβας στην κληθείσα**
addi$sp,$sp,framelength

καλούμε τη συνάρτηση
jal f

και **όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας** στην καλούσα
addi$sp,$sp,-framelength

Μέσα στην κληθείσα

Στην αρχή κάθε συνάρτησης αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την **διεύθυνση επιστροφής** της την οποία έχει τοποθετήσει στον $ra η jal.

sw$ra,($sp)

Στο **τέλος** κάθε συνάρτησης κάνουμε το αντίστροφο, παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την **διεύθυνση επιστροφής** της συνάρτησης και την βάζουμε πάλι στον $ra. Μέσω του $raεπιστρέφουμε στην καλούσα
lw$ra,($sp)
jr$ra

Το κυρίως πρόγραμμα δεν είναι το πρώτο πράγμα που μεταφράζεται, οπότε στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος
j Lmain

φυσικά η **j Lmain** πρέπει να δημιουργηθεί όταν ξεκινά η μετάφραση της main
στη συνέχεια πρέπει να **κατεβάσουμε τον $sp κατά framelength της main**

addi$sp,$sp,framelength

και να σημειώσουμε στον **$s0 το εγγράφημα δραστηριοποίησης της main** ώστε να έχουμε εύκολη πρόσβαση στις global μεταβλητές

move $s0,$sp