Πανεπιστήμιο Θεσσαλίας
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τμήμα Πληροφορικής

Μεταγλωττιστές

Σημασιολογική Ανάλυση και Παραγωγή Ενδιάμεσου Κώδικα

Στη μετάφραση οδηγούμενη από τη σύνταξη, τόσο η σημασιολογική ανάλυση, όσο και η παραγωγή ενδιάμεσου κώδικα, υλοποιούνται μέσα από τη συντακτική ανάλυση, και συγκεκριμένα επεκτείνοντας τη γραμματική της γλώσσας με εισαγωγή κατηγορημάτων στα σύμβολα και σημασιολογικών ρουτινών στους κανόνες της. Τα κατηγορήματα αποθηκεύουν την πληροφορία που συλλέγεται κατά την ανάλυση, ώστε να μπορεί αυτή να μεταφέρεται από σύμβολο σε σύμβολο. Οι σημασιολογικές ρουτίνες υλοποιούν τη μεταφορά της πληροφορίας, χρησιμοποιώντας όποια πληροφορία συλλέγεται από τις λεκτικές μονάδες που παρέχει ο λεκτικός αναλυτής, καθώς και την πληροφορία από την προηγούμενη συντακτική και σημασιολογική ανάλυση, και παράγοντας νέα.
Έτσι, για την υλοποίηση της σημασιολογικής ανάλυσης μέσα από το συντακτικό αναλυτή, πρέπει να επεκτείνουμε τη γραμματική της γλώσσας σε κατηγορική γραμματική. Το πρώτο που πρέπει να κάνουμε επομένως, είναι να απεικονίσουμε τα σημασιολογικά χαρακτηριστικά της γλώσσας σε κατάλληλα κατηγορήματα. Για παράδειγμα, οι βασικοί τύποι της γλώσσας μπορούν να παρασταθούν με κατάλληλη κωδικοποίηση μέσω κάποιου κατηγορήματος ακέραιου τύπου, το οποίο θα συνοδεύει όλα εκείνα τα σύμβολα της γραμματικής που έχουν τύπο, όπως είναι τα αναγνωριστικά, οι σταθερές και όλα τα σύμβολα παραστάσεων. Στη συνέχεια, θα πρέπει να μετατρέψουμε όλους τους σημασιολογικούς κανόνες και περιορισμούς της γλώσσας, που συνήθως μας δίνονται σε περιφραστική μορφή μέσα από κάποιο εγχειρίδιο, σε αλγεβρικές και λογικές παραστάσεις των αντίστοιχων κατηγορημάτων. Οι σημασιολογικοί κανόνες και περιορισμοί, στην τελική τους μορφή, συνοδεύουν τους συντακτικούς κανόνες της γλώσσας, και αποτιμώνται κάθε φορά που ο αντίστοιχος συντακτικός κανόνας χρησιμοποιείται στη συντακτική ανάλυση κάποιου προγράμματος.
Η παραγωγή ενδιάμεσου κώδικα γίνεται παράλληλα με τη σημασιολογική ανάλυση. Έτσι, σε κάθε σημείο της συντακτικής ανάλυσης που θέλουμε να παράγει ενδιάμεσο κώδικα, τοποθετούμε κατάλληλη βοηθητική ρουτίνα, η οποία να καλείται από το συντακτικό αναλυτή και να παράγει τον επιθυμητό ενδιάμεσο κώδικα σε κάποιο κατηγόρημα κατάλληλου τύπου, χρησιμοποιώντας τα αντίστοιχα κατηγορήματα από το δεξί μέλος του κανόνα. Ο ενδιάμεσος κώδικας κάθε μονάδας του προγράμματος εμφανίζεται στο τέλος της ανάλυσης της μονάδας, μέσα από κάποιο συντιθέμενο κατηγόρημα του συμβόλου στη ρίζα του αντίστοιχου συντακτικού δέντρου.

Ενσωμάτωση σημασιολογικών χαρακτηριστικών στη σύνταξη της γλώσσας

Πολλές φορές είναι επιθυμητό, κάποια σημασιολογικά χαρακτηριστικά μιας γλώσσας να ενσωματώνονται στη σύνταξη της γλώσσας. Αν και θεωρητικά όλα τα σημασιολογικά χαρακτηριστικά θα μπορούσαν να ενσωματωθούν στη σύνταξη, κάτι τέτοιο δε μπορεί να γίνει στην πράξη, διότι θα έκανε απαγορευτικό το μέγεθος και την πολυπλοκότητα της γραμματικής, που δε θα μπορούσε πια να είναι χωρίς συμφραζόμενα. Για παράδειγμα, επειδή το όνομα μιας μεταβλητής είναι σημασιολογικό χαρακτηριστικό της λεκτικής μονάδας του αναγνωριστικού, θα έπρεπε κάθε δυνατό όνομα να αποτελεί ανεξάρτητη λεκτική μονάδα στη γραμματική της γλώσσας. Επίσης, αν η χρήση μιας μεταβλητής πρέπει να ακολουθεί τη δήλωσή της σε κάποια μονάδα του προγράμματος, ο έλεγχος προηγούμενης δήλωσης στη χρήση μιας με-

ταβλητής απαιτεί η δήλωση να είναι συμφραζόμενο στον κανόνα στον οποίο ορίζεται η χρήση της μεταβλητής σε κάποια παράσταση.
Από την άλλη μεριά, η ενσωμάτωση κάποιων σημασιολογικών χαρακτηριστικών στους συντακτικούς κανόνες της γλώσσας μπορεί να απλοποιεί τη διαδικασία συντακτικής ανάλυσης, παρά την αύξηση σε αριθμό κανόνων. Για παράδειγμα, αν η γραμματική είναι διφορούμενη, κάποια σημασιολογικά χαρακτηριστικά μπορούν να τη μετατρέψουν σε ισοδύναμη μη διφορούμενη. Με τον τρόπο αυτό, η συντακτική ανάλυση γίνεται ντετερμινιστική, με συνέπεια τη σημαντική μείωση της χρονικής πολυπλοκότητάς της.
Την πλέον συνηθισμένη περίπτωση ενσωμάτωσης σημασιολογικών χαρακτηριστικών στη σύνταξη μιας γλώσσας αποτελούν οι ιδιότητες των τελεστών της γλώσσας. Ενώ οι εκφράσεις της γλώσσας μπορούν αρχικά να είναι διφορούμενες, η προτεραιότητα και η προσεταιριστικότητα των τελεστών μπορούν να τις μετατρέψουν σε μη διφορούμενες. Μάλιστα, τα συνηθισμένα μετα-εργαλεία συντακτικής ανάλυσης – όπως για παράδειγμα το Bison – δέχονται δηλώσεις προτεραιότητας και προσεταιριστικότητας, έτσι ώστε να κάνουν τη μετατροπή έμμεσα κατά τη διαδικασία κατασκευής των πινάκων ανάλυσης, χωρίς να αυξάνουν τους κανόνες της γραμματικής και – ακόμα περισσότερο – χωρίς να επιβάλλουν στον προγραμματιστή να κάνει τη μετατροπή[1].

Δημιουργία κατηγορικής γραμματικής

Η δημιουργία της κατηγορικής γραμματικής πάνω στην οποία θα στηριχθεί η σημασιολογική ανάλυση και η παραγωγή ενδιάμεσου κώδικα δεν είναι συστηματική διαδικασία που να μπορεί να περιγραφεί με κάποιον αλγόριθμο. Κάθε γλώσσα έχει ιδιαίτερα σημασιολογικά χαρακτηριστικά, τα οποία περιγράφονται στα εγχειρίδια της γλώσσας περιφραστικά μέσα από παραδείγματα, και όχι με κάποιον αυστηρό και γενικά τυποποιημένο τρόπο. Όσο υπάρχει τυποποίηση στην περιγραφή των συντακτικών κανόνων μιας γλώσσας, άλλο τόσο *δεν* υπάρχει τυποποίηση στην περιγραφή της σημασιολογίας της γλώσσας.
Και αν κάποια σημασιολογικά χαρακτηριστικά μπορούν να περιγραφούν μέσα από κάποιες αλγεβρικές παραστάσεις, όπως για παράδειγμα το σύστημα τύπων της γλώσσας, κάποια άλλα, όπως η σημασιολογία των δηλώσεων και των εντολών της γλώσσας, περιγράφονται κύρια μέσα από παραδείγματα. Έτσι, ενώ για την πρώτη περίπτωση η απεικόνιση των χαρακτηριστικών σε κατηγορήματα και σημασιολογικούς κανόνες και περιορισμούς είναι σχετικά εύκολη, για τη δεύτερη περίπτωση η απεικόνιση θα γίνει μετά από την άριστη κατανόηση της γλώσσας, που συνήθως αποκτάται μετά από αρχικές αποτυχημένες προσπάθειες σχεδίασης της σημασιολογικής ανάλυσης, και πολλές φορές με εκ των υστέρων πρόσθεση κάποιων κατηγορημάτων, αφού γίνει συνειδητή η ανάγκη υλοποίησής τους.
Για την περιγραφή του συστήματος τύπων για παράδειγμα, δύο είναι τα βασικά κατηγορήματα που απαιτούνται: Το ένα να παρέχει τον κωδικό τύπου, και το άλλο να παρέχει την πραγματική δομή του τύπου, αν είναι σύνθετος. Ο κωδικός τύπου μπορεί να είναι κάποιος ακέραιος, ενώ η δομή του τύπου είναι πιθανό να πρέπει να υλοποιηθεί με γράφημα, ιδιαίτερα αν η γλώσσα υποστηρίζει κυκλικές δομές τύπων, όπως η Pascal και η C που υποστηρίζουν δείκτες σε σύνθετους τύπους, οι οποίοι μπορούν να περιέχουν τους ίδιους δείκτες. Για βασικούς τύπους, ο κωδικός τύπου παρέχεται από τους κανόνες που περιέχουν τις αντίστοιχες λεκτικές μονάδες. Για σύνθετους τύπους, ο κωδικός και η δομή του τύπου παρέχονται από τους κανόνες που περιγράφουν τους κατασκευαστές αυτών των τύπων. Οι σημασιολογικές ρουτίνες στην περίπτωση του συστήματος τύπων θα περιλαμβάνουν τουλάχιστον κατασκευή των αντίστοιχων δομών αναπαράστασης – για σύνθετους τύπους, έλεγχο συμβατότητας τύπων για κάθε συντακτικό κανόνα όπου απαντώνται περισσότεροι από έναν τύπο, καθώς και πιθανή παραγωγή τύπου αποτελέσματος.

[1] Η παρέμβαση στην κατασκευή των πινάκων συντακτικής ανάλυσης μπορεί να οδηγεί σε ισοδύναμη γραμματική, αλλά μπορεί και όχι. Αν για παράδειγμα η γραμματική είναι εγγενώς διφορούμενη, καμία τέτοια παρέμβαση δε θα μπορούσε να οδηγήσει σε ισοδύναμη γραμματική με ντετερμινιστική συντακτική ανάλυση!

Δεδομένου ότι μαζί με τη σημασιολογική ανάλυση, οι σημασιολογικές ρουτίνες παράγουν και ενδιάμεσο κώδικα, ένα κατηγόρημα που μπορεί να συνοδεύει σύμβολα της γραμματικής που περιγράφουν εντολές ή παραστάσεις είναι κάποιο κατηγόρημα που απεικονίζει ενδιάμεσο κώδικα. Αν ο ενδιάμεσος κώδικας είναι σε μορφή αφηρημένου συντακτικού δέντρου, ένα βασικό κατηγόρημα θα είναι ένας δείκτης στη δομή κόμβου του ενδιάμεσου κώδικα, τυπικά κάποιου struct της C.
Ειδικά για σύμβολα λεκτικών μονάδων, η εύρεση κατηγορημάτων είναι πιο απλή, δεδομένου ότι οι λεκτικές μονάδες έχουν συγκεκριμένα χαρακτηριστικά που είναι εύκολα κατανοητά. Για παράδειγμα οι σταθερές μιας γλώσσας έχουν τύπο και τιμή, τα αναγνωριστικά της γλώσσας έχουν όνομα ή/και δείκτη στον πίνακα συμβόλων, ενώ οι τελεστές έχουν κάποιον κωδικό τελεστή, αν κάτι τέτοιο είναι απαραίτητο και δεν εννοείται από τη γραμματική της γλώσσας.

Εφαρμογή σημασιολογικών κανόνων και περιορισμών

Οι σημασιολογικοί κανόνες και περιορισμοί που συνδέουν τις τιμές των κατηγορημάτων μεταξύ τους και συνοδεύουν τους συντακτικούς κανόνες με τη μορφή ρουτινών, εφαρμόζονται με έναν από τους ακόλουθους δύο τρόπους:

1. Αφού κατασκευαστεί το δέντρο συντακτικής ανάλυσης για το δεδομένο πρόγραμμα, δημιουργείται πάνω σε αυτό ο γράφος εξαρτήσεων των κατηγορημάτων, ο οποίος και καθορίζει τη σειρά εφαρμογής.
2. Προκαθορίζοντας τη θέση των ρουτινών μέσα στους συντακτικούς κανόνες, κάνουμε την αποτίμησή τους τη στιγμή ακριβώς που η συντακτική ανάλυση φτάσει σε αυτή τη θέση.

Από τις δύο αυτές μεθόδους, η δεύτερη – που ονομάζεται και *σχήμα μετάφρασης* (translation scheme) – μπορεί να χρησιμοποιείται όταν η σειρά αποτίμησης ακολουθεί τη σειρά συντακτικής ανάλυσης των συμβόλων. Έτσι, για τις συνηθισμένες συντακτικές αναλύσεις που τα σύμβολα αναλύονται από αριστερά προς τα δεξιά, για να μπορεί να χρησιμοποιηθεί αυτή η μέθοδος, η αντίστοιχη κατηγορική γραμματική πρέπει να είναι L-κατηγορική. Στο μετα-εργαλείο Bison, που υποστηρίζει S-κατηγορικές γραμματικές – οι οποίες βέβαια είναι και L-κατηγορικές, χρησιμοποιείται αυτή η μέθοδος για τη σημασιολογική ανάλυση.
Αξίζει να σημειώσουμε ότι με τη δεύτερη μέθοδο, το δέντρο συντακτικής ανάλυσης δεν είναι απαραίτητο να κατασκευαστεί, μια που για την αποτίμηση των κατηγορημάτων μπορούμε να χρησιμοποιήσουμε τη στοίβα συντακτικής ανάλυσης, η οποία στην ουσία εξομοιώνει το δέντρο. Στη συνέχεια θα ασχοληθούμε μόνο με αυτή τη μέθοδο.

Κατηγορήματα και σημασιολογικές ρουτίνες με το μετα-εργαλείο Bison

Ας δούμε στη συνέχεια πώς υλοποιούνται οι κατηγορικές γραμματικές στο μετα-εργαλείο Bison. Το Bison υποστηρίζει ένα κατηγόρημα για κάθε σύμβολο της γραμματικής, και σημασιολογικές ρουτίνες με τη μορφή κώδικα C ανάμεσα σε άγκιστρα, οι οποίες τοποθετούνται σε κατάλληλες θέσεις στο δεξί μέλος ενός συντακτικού κανόνα. Κάθε ρουτίνα θεωρείται ειδικό σύμβολο της γραμματικής[2], γι' αυτό και προσμετράται όταν αριθμούνται τα σύμβολα του κανόνα για αναφορά στα κατηγορήματα αυτών!
Έτσι για παράδειγμα σε έναν κανόνα της μορφής:

```
A : a B c D
```

όπου a και c τερματικά σύμβολα, η προσθήκη σημασιολογικών ρουτινών μπορεί να δώσει ένα νέο κανόνα της μορφής:

```
A : a B { code1 } c D { code2 }
```

Η αντιστοίχηση κατηγορημάτων σε σύμβολα γίνεται ως εξής:

$1 : κατηγόρημα του a

[2] Επειδή η εισαγωγή νέων συμβόλων τροποποιεί τη γραμματική, πρέπει να ελέγχουμε ότι ο συντακτικός αναλυτής συνεχίζει να λειτουργεί σωστά μετά την εισαγωγή κάποιας σημασιολογικής ρουτίνας!

$2 : κατηγόρημα του B
$3 : κατηγόρημα του { code1 }
$4 : κατηγόρημα του c
$5 : κατηγόρημα του D
$6 : κατηγόρημα του { code2 }
$$: κατηγόρημα του A

Τα $1 και $4 που αναφέρονται σε τερματικά σύμβολα – αν χρησιμοποιούνται – πρέπει να έχουν λάβει τιμή από το λεκτικό αναλυτή (με τη βοήθεια της μεταβλητής yylval).
Τα $2 και $5 – αν χρησιμοποιούνται – πρέπει να λαμβάνουν τιμή από άλλους κανόνες που έχουν τα αντίστοιχα μη τερματικά σύμβολα B και D στο αριστερό μέλος τους.
Τα $3 και $6 μπορούν να χρησιμοποιηθούν σαν προσωρινές μεταβλητές.
Το $$ λαμβάνει τιμή μέσα από τις δύο σημασιολογικές ρουτίνες του κανόνα.
Επειδή το Bison υλοποιεί συντακτικούς αναλυτές της οικογένειας LR, η αναγνώριση των συμβόλων στο δεξί μέλος ενός κανόνα γίνεται από αριστερά προς τα δεξιά. Έτσι, αναφορά σε ένα κατηγόρημα από τα $1 - $6 δε μπορεί να γίνει πριν την αναγνώριση του αντίστοιχου συμβόλου. Για παράδειγμα, η ρουτίνα "code1" δε μπορεί να αναφερθεί στα κατηγορήματα $4 - $6, αφού κατά τη στιγμή εκτέλεσής της έχουν αναγνωριστεί μόνο τα σύμβολα a και B του κανόνα, το οποίο σημαίνει ότι μόνο τα $1 και $2 έχουν έγκυρες τιμές – αν βέβαια αυτές τους έχουν αποδοθεί! Η ρουτίνα "code1" μπορεί να δώσει τιμή στο κατηγόρημα $3, ώστε αυτό να χρησιμοποιηθεί από τη ρουτίνα "code2", όπως επίσης και στο κατηγόρημα $$, το οποίο μπορεί να χρησιμοποιηθεί από τη ρουτίνα "code2" ή να επιστραφεί με την ολοκλήρωση της αναγνώρισης του κανόνα (ελάττωση) σε κάποιον άλλο κανόνα που αναγνωρίζει το σύμβολο A στο δεξί μέλος του, όντας στην ίδια κατάσταση συντακτικής ανάλυσης με το στοιχείο ελάττωσης. Επειδή η μεταφορά πληροφορίας από σύμβολο σε σύμβολο και από κανόνα σε κανόνα μέσω των κατηγορημάτων επιτυγχάνεται με την εισαγωγή των τιμών αυτών στη στοίβα του συντακτικού αναλυτή, κάθε κατηγόρημα παραμένει ενεργό μέχρι την ολοκλήρωση της αναγνώρισης του αντίστοιχου κανόνα με την ελάττωσή του.
Η επεξεργασία που γίνεται σε κάθε σημασιολογική ρουτίνα περιλαμβάνει τουλάχιστον ένα από (α) έλεγχο σημασιολογικής ορθότητας του προγράμματος, (β) παραγωγή πληροφορίας που θα χρησιμοποιηθεί αργότερα, και (γ) παραγωγή ενδιάμεσου κώδικα.
Έστω για παράδειγμα μια γλώσσα προγραμματισμού, στην οποία η εφαρμογή ενός τελεστή X σε δύο τελούμενα γίνεται με βάση το συντακτικό κανόνα:

```
expression : expression X expression
```

και διέπεται από το σημασιολογικό κανόνα:
"Τα τελούμενα του τελεστή X πρέπει να είναι τύπου A ή B, και το αποτέλεσμα της εφαρμογής του είναι τύπου B, αν κάποιο τελούμενο είναι τύπου B, διαφορετικά είναι τύπου A."
Αν ο μεταγλωττιστής της γλώσσας αυτής κατασκευάζει ενδιάμεσο κώδικα στη μορφή αφηρημένου συντακτικού δέντρου και το σύμβολο expression έχει ως κατηγόρημα έναν κόμβο αυτού, ο συντακτικός κανόνας, για να υλοποιεί τον πιο πάνω σημασιολογικό περιορισμό, θα τροποποιηθεί ως εξής:

```
expression : expression X expression {
  if (($1->type != A) || ($1->type != B) ||
      ($3->type != A) || ($3->type != B))
     yyerror("Mh symbata teloumena tou telesth X");
  $$ = Create_Tree (opX, $1, $3);
  if (($1->type == B) || ($3->type == B)) $$->type = B;
  else $$->type = A;
}
```

όπου υποθέσαμε ότι ο κόμβος του αφηρημένου συντακτικού δέντρου έχει κάποιο πεδίο type που αποθηκεύει τον τύπο αυτού. Ο τελεστής X αποθηκεύεται σε κάποιο άλλο πεδίο του κόμβου με κατάλληλο κωδικό opX, μέσα από την κλήση της συνάρτησης Create_Tree().
Το πρώτο μέρος της πιο πάνω σημασιολογικής ρουτίνας υλοποιεί τον έλεγχο συμβατότητας των τελούμενων του X, το δεύτερο μέρος παράγει ενδιάμεσο κώδικα, κατασκευάζοντας έναν

κόμβο του αφηρημένου συντακτικού δέντρου, ενώ το τρίτο μέρος παράγει κάποια πληροφορία για τον κόμβο, που θα χρησιμοποιηθεί αργότερα σε κάποια άλλη εκτέλεση σημασιολογικής ρουτίνας (πιθανά της ίδιας!). Ο κόμβος που κατασκευάστηκε αποθηκεύεται στο κατηγόρημα $$, ώστε να μεταφερθεί σε άλλη αναγνώριση κανόνα.

Χρήση καθολικών μεταβλητών στο μεταγλωττιστή

Είδαμε πιο πάνω ότι η πληροφορία που συγκεντρώνεται κατά τη σημασιολογική ανάλυση αποθηκεύεται στα κατηγορήματα των συμβόλων της γραμματικής. Επειδή στην πράξη η σημασιολογική ανάλυση γίνεται μέσα από τα μετα-εργαλεία συντακτικής ανάλυσης, υπάρχουν περιορισμοί στη χρήση των κατηγορημάτων που προκύπτουν από τον τρόπο λειτουργίας αυτών των μετα-εργαλείων. Για παράδειγμα το Bison δεν υποστηρίζει κληρονομούμενα κατηγορήματα. Τέτοιοι περιορισμοί μας οδηγούν σε εναλλακτικές λύσεις, όπως είναι η χρήση καθολικών μεταβλητών στο μεταγλωττιστή για αποθήκευση πληροφορίας που σε ιδανικές συνθήκες θα αποθηκεύαμε σε κατηγορήματα.
Χαρακτηριστικό παράδειγμα καθολικής μεταβλητής που υλοποιείται σε κάθε μεταγλωττιστή είναι ο πίνακας συμβόλων. Λόγω της συχνής προσπέλασης του πίνακα συμβόλων, αν δεν τον υλοποιούσαμε ως καθολική μεταβλητή, θα έπρεπε να τον παρέχουμε ως κληρονομούμενο κατηγόρημα στα περισσότερα σύμβολα της γραμματικής, αλλά και να επιστρέφεται από αυτά ως συντιθέμενο κατηγόρημα. Τις πιο πολλές φορές, η μεταφορά του από σύμβολο σε σύμβολο θα γινόταν απλά για να φτάσει στον κανόνα στον οποίο πραγματικά προσπελαύνεται, κάτι που θα αύξανε άσκοπα το χρόνο μεταγλώττισης του προγράμματος. Ακόμα κι αν το μετα-εργαλείο μας παρείχε τη δυνατότητα υποστήριξης κληρονομούμενων κατηγορημάτων, θα συνεχίζαμε να προτιμάμε την υλοποίηση του πίνακα συμβόλων ως καθολική μεταβλητή.
Για παρόμοιους λόγους μπορούμε να ορίσουμε και να χρησιμοποιήσουμε και άλλες καθολικές μεταβλητές σε ένα μεταγλωττιστή, αλλά θα πρέπει να έχουμε υπ' όψη τα εξής:

1. Πρέπει να έχουμε καλή κατανόηση της λειτουργίας του μετα-εργαλείου που χρησιμοποιούμε, ώστε να είμαστε σίγουροι ότι οι καθολικές μεταβλητές προσπελαύνονται με τη σειρά που θέλουμε να προσπελαύνονται. Έτσι, για το Bison:
   α) Με τον τρόπο που ο συντακτικός αναλυτής εκτελεί τις κινήσεις του, μεταπηδάει συχνά από έναν κανόνα σε κάποιον άλλο, πριν ξαναγυρίσει στον πρώτο. Ένα κοινό λάθος είναι η χρήση μιας καθολικής μεταβλητής σε διαφορετικές θέσεις κάποιου κανόνα, νομίζοντας ότι η τιμή της δε θα αλλάξει, όσο δεν υπάρχει ανάθεση σε αυτήν μέσα στις σημασιολογικές ρουτίνες του κανόνα. Ίσως σε κάποιον άλλο κανόνα που χρησιμοποιείται ανάμεσα στα σύμβολα του δεξιού μέλους του πρώτου, είναι πιθανό να έχουμε γράψει κώδικα που να περιέχει ανάθεση στην ίδια μεταβλητή.
   β) Ένας συντακτικός αναλυτής που χρησιμοποιεί ένα προπορευόμενο σύμβολο στις κινήσεις του, προχωράει μια λεκτική μονάδα παρακάτω από το σημείο στο οποίο βρίσκεται. Επομένως, αν μια καθολική μεταβλητή προσπελαύνεται *και* από το λεκτικό αναλυτή, είναι πολύ πιθανό να προσπελαστεί με άλλη σειρά από την αναμενόμενη.
2. Οι πολλές καθολικές μεταβλητές κάνουν το μεταγλωττιστή δυσανάγνωστο, και δυσκολεύουν τη διόρθωση λαθών του.

Εντολές, δηλώσεις και παραγωγή ενδιάμεσου κώδικα

Γενικά στην παραγωγή του ενδιάμεσου κώδικα έχουμε δύο περιπτώσεις αρχικού κώδικα:
(α) Αρχικό κώδικα που δεν είναι εκτελέσιμος, όπως είναι οι δηλώσεις.
(β) Αρχικό κώδικα που είναι εκτελέσιμος, όπως είναι οι εντολές και οι αποτιμήσεις εκφράσεων.
Στην πρώτη περίπτωση, ο αρχικός κώδικας παρέχει πληροφορίες που αποθηκεύονται σε διάφορες δομές του μεταγλωττιστή, όπως στον πίνακα συμβόλων. Ο τελευταίος διατηρεί τις πληροφορίες του μόνο όσο διαρκεί η εμβέλεια στην οποία αυτές συγκεντρώθηκαν. Όποιες από αυτές χρειάζονται για το υπόλοιπο της μετάφρασης πρέπει να αποθηκεύονται και στον ενδιάμεσο κώδικα, όπως για παράδειγμα οι τύποι των μεταβλητών του προγράμματος. Κά-

ποιες πληροφορίες καταλήγουν και στον τελικό κώδικα, όπως για παράδειγμα οι αρχικές τιμές στατικών μεταβλητών.
Στη δεύτερη περίπτωση από την άλλη μεριά, είναι πιθανό, αλλά όχι απαραίτητο, ο αρχικός κώδικας να παρέχει πληροφορίες όπως οι πιο πάνω. Ο τύπος του αποτελέσματος της αποτίμησης μιας έκφρασης είναι παράδειγμα πληροφορίας που συγκεντρώνεται από εκτελέσιμο αρχικό κώδικα. Πέρα από όποιες πληροφορίες, ένας εκτελέσιμος αρχικός κώδικας παράγει τόσο ενδιάμεσο, όσο και τελικό κώδικα[3].

Δηλώσεις και Χώροι Δεδομένων

Μια από τις πιο βασικές πληροφορίες που χρειαζόμαστε για την παραγωγή κώδικα είναι οι πληροφορίες δέσμευσης χώρου στη μνήμη για κάθε δεδομένο του προγράμματος.
Κάθε μεταβλητή ενός προγράμματος αποτελεί ένα δεδομένο αυτού, που καταλαμβάνει κάποιο χώρο στη μνήμη. Κατά τη μετάφραση χρειαζόμαστε κάποιο "προσχέδιο" για την οργάνωση του χώρου δεδομένων, ώστε να μπορούμε να ξέρουμε κατά την εκτέλεση του κώδικα, πού βρίσκεται κάθε δεδομένο στη μνήμη. Για παράδειγμα, η εντολή C:

```
x = a+2;
```

πρέπει, όταν εκτελεστεί, να διαβάσει από το χώρο δεδομένων την τιμή της μεταβλητής a, να εκτελέσει την πρόσθεση με την ακέραια σταθερά 2, και να αποθηκεύσει το αποτέλεσμα πίσω στο χώρο δεδομένων, στη θέση που αντιστοιχεί στη μεταβλητή x. Σε κάθε μία από τις δύο προσπελάσεις του χώρου δεδομένων, ο κώδικας πρέπει να ξέρει ακριβώς ποια διεύθυνση θα προσπελάσει.
Σε ένα ορθό πρόγραμμα, μια μεταβλητή χρησιμοποιείται μετά τη δήλωσή της. Επομένως η δήλωση αυτής μπορεί να αποτελέσει το σημείο στο οποίο θα δεσμεύσουμε στο χώρο δεδομένων μια θέση γι' αυτή. Στη συνέχεια, σε κάθε αναφορά της, το μόνο που χρειαζόμαστε για την παραγωγή του κώδικα είναι (α) η θέση της και (β) ο τύπος της. Με κάθε δέσμευση αναγράφουμε τον αριθμό θέσης στον πίνακα συμβόλων, ώστε να συνδέσουμε τις επόμενες αναφορές στη μεταβλητή με τη θέση αυτή, όσο βέβαια διαρκεί η εμβέλεια στην οποία αυτή είναι δηλωμένη.
Όταν μπορούμε να έχουμε πολλούς ενεργούς χώρους δεδομένων, χρειαζόμαστε επιπλέον και την ένδειξη του χώρου δεδομένων στον οποίο αναφερόμαστε. Για παράδειγμα, σε φωλιασμένα υποπρογράμματα της Pascal μια αναφορά πρέπει να συνοδεύεται από το βάθος φωλιάσματος στο οποίο ορίζεται η αντίστοιχη μεταβλητή, ενώ σε οποιαδήποτε συνάρτηση της C μια αναφορά πρέπει να συνοδεύεται από την ένδειξη καθολικής ή τοπικής μεταβλητής.
Ας θεωρήσουμε το πιο κάτω πρόγραμμα C:

```
int x; float y[10];
int A(unsigned i,float z) {
    int x[100];
    static float a = -1.0;
    if (i > 10) {
        unsigned x;
        x = i;
        a += x+z;
        return 1;
    }
    a += z+y[i];
    return 0;
}
```

Με την πρώτη δήλωση "int x", κατ' αρχήν ενημερώνουμε τον πίνακα συμβόλων για τον τύπο του αναγνωριστικού "x", ώστε να μπορούμε να ανιχνεύσουμε σημασιολογικά σφάλματα (α) από δεύτερη δήλωση του ίδιου αναγνωριστικού στην ίδια εμβέλεια, και (β) από λανθασμένη χρήση του αναγνωριστικού. Εκτός από την ενημέρωση αυτή, πρέπει να δεσμεύσουμε κάποια

[3] Τελικός κώδικας δε θα παραχθεί, μόνο αν σε κάποια φάση βελτιστοποίησης ο ενδιάμεσος κώδικας για κάποιο λόγο απαλειφτεί.

θέση στο χώρο καθολικών δεδομένων του προγράμματος για τη μεταβλητή x, και να σημειώσουμε ότι η θέση αυτή είναι τύπου int. Για απλούστευση δεσμεύουμε θέσεις σειριακά, κι επομένως η μεταβλητή x θα αντιστοιχηθεί στην πρώτη θέση του καθολικού χώρου δεδομένων του προγράμματος. Για τη δέσμευση αυτή χρησιμοποιούμε ένα βοηθητικό πίνακα, τον πίνακα δέσμευσης του χώρου δεδομένων στον οποίο αναφερόμαστε.

Η σημείωση του τύπου στον πίνακα δέσμευσης είναι απαραίτητη, επειδή θα μας χρειαστεί όταν βγούμε από την τρέχουσα εμβέλεια, οπότε και θα χάσουμε όποιες πληροφορίες μας παρέχει ο πίνακας συμβόλων για τη μεταβλητή x[4]. Σε αντίθεση με τον πίνακα συμβόλων, όπου οι πληροφορίες αποθηκεύονται με βάση το όνομα μιας μεταβλητής, κι επομένως χάνονται με την έξοδο από την εμβέλεια στην οποία αυτή δηλώνεται, ο πίνακας δέσμευσης αποθηκεύει τις πληροφορίες με βάση τη θέση της μεταβλητής στο χώρο δεδομένων, κι έτσι τις διατηρεί μέχρι το τέλος της μετάφρασης της μονάδας, ώστε οι δεσμεύσεις να μεταφέρονται στον τελικό κώδικα με τη μορφή διευθύνσεων και μετατοπίσεων σε εντολές προσπέλασης μνήμης. Μάλιστα, επειδή ο ενδιάμεσος κώδικας δε γνωρίζει το μέγεθος των τύπων στην τελική γλώσσα μετάφρασης, ο πίνακας δέσμευσης χρησιμεύει και ως εργαλείο για τον υπολογισμό της επακριβούς λογικής διεύθυνσης και επακριβούς μετατόπισης σε ψηφιολέξεις κάθε μεταβλητής στο χώρο δεδομένων της. Εδώ για παράδειγμα, εφ' όσον δεν ξέρουμε τι αναπαράσταση έχει ο τύπος int στην τελική γλώσσα μετάφρασης, δηλαδή πόσες ψηφιολέξεις χρησιμοποιεί, δε μπορούμε να ξέρουμε τις μετατοπίσεις των επόμενων μεταβλητών στο χώρο δεδομένων πριν την παραγωγή τελικού κώδικα, οπότε και θα μπορέσουμε να τις υπολογίσουμε.

Με τη δήλωση "float y[10]" ενημερώνουμε τον πίνακα συμβόλων και δεσμεύουμε τις επόμενες δέκα θέσεις στον ίδιο χώρο δεδομένων, υποδεικνύοντας ότι αυτές είναι τύπου float. Ας σημειωθεί ότι σε γλώσσες με πολύπλοκες δομές, δε μας αρκεί η αναφορά του τύπου μιας μεταβλητής, αλλά και η οργάνωση του τύπου αυτού, ώστε να μπορούμε αργότερα να υπολογίσουμε τις διευθύνσεις και μετατοπίσεις των επιμέρους στοιχείων ή πεδίων τους. Κάτι τέτοιο είναι απαραίτητο, όχι μόνο για αναφορές σε μεμονωμένα στοιχεία ή πεδία των μεταβλητών αυτών, αλλά και για αναφορές σε ολόκληρες τις δομές, για γλώσσες που επιτρέπουν για παράδειγμα αναθέσεις ή συγκρίσεις σύνθετων τύπων. Τέλος, σε γλώσσες που το σύστημα τύπων επιβάλλει δυναμικό έλεγχο ορθότητας μιας αναφοράς, η οργάνωση ενός τύπου μεταφέρεται στον τελικό κώδικα, ο οποίος πρέπει να εκτελεί αυτόν τον έλεγχο. Έτσι, για την πιο πάνω δήλωση, θα πρέπει να σημειώσουμε ότι η δομή είναι μονοδιάστατη με 10 στοιχεία, ώστε να διαχωριστεί για παράδειγμα από τη δομή που θα αντιστοιχούσε στη δήλωση "float y[2][5]".

Μέσα στην εμβέλεια της συνάρτησης A, ο χώρος δεδομένων κατά την εκτέλεση του προγράμματος θα είναι ένα εγγράφημα δραστηριοποίησης, αφού η γλώσσα C χρησιμοποιεί εγγραφήματα δραστηριοποίησης για υποστήριξη αναδρομής στις κλήσεις συναρτήσεων. Η μορφή των εγγραφημάτων δραστηριοποίησης είναι σε μεγάλο βαθμό αυθαίρετη, και καθορίζεται από τους σχεδιαστές των μεταγλωττιστών. Εδώ θα ακολουθήσουμε μια μορφή, όπου στην αρχή του εγγραφήματος δεσμεύουμε χώρο που σχετίζεται με το μηχανισμό κλήσης της συνάρτησης και σύνδεσής της με το υπόλοιπο περιβάλλον[5], και μετά δεσμεύουμε χώρο για τις τοπικές μεταβλητές της συνάρτησης.

Έτσι, την πρώτη θέση του χώρου δεδομένων τη δεσμεύουμε για αποθήκευση της διεύθυνσης επανόδου, μια θέση που τη θεωρούμε τύπου void*. Τη δεύτερη θέση τη δεσμεύουμε για αποθήκευση της τιμής του αποτελέσματος της A, τύπου int. Κάθε εκτέλεση της εντολής "return 1;" θα αποθηκεύει την τιμή 1 στη θέση αυτή. Στη συνέχεια, δεσμεύουμε δύο θέσεις για τις δύο παραμέτρους της A, μια θέση τύπου unsigned και μια θέση τύπου float, και ενημερώνουμε τον πίνακα συμβόλων για τον τύπο των αντίστοιχων αναγνωριστικών "i" και "z", υποδεικνύοντας ταυτόχρονα ότι έχουμε δεσμεύσει γι' αυτά την τρίτη και την τέταρτη θέση του χώρου δεδομένων. Πριν από κάθε κλήση της A, το καλούν περιβάλλον τοποθετεί τις τιμές των

---

[4] Εδώ δε θα συμβεί κάτι τέτοιο, αφού βρισκόμαστε στην εξωτερική εμβέλεια. Θα συνέβαινε όμως, αν βρισκόμαστε σε εσωτερική εμβέλεια. Για ομοιομορφία επεξεργασίας, χρησιμοποιούμε πίνακες δέσμευσης για όλους τους χώρους δεδομένων.

[5] Για γλώσσες όπως η Pascal που υποστηρίζουν φωλιασμένες δηλώσεις υποπρογραμμάτων με στατικό δέσιμο, τουλάχιστον μια θέση πρέπει να δεσμεύεται για το σύνδεσμο προσπέλασης.

πραγματικών παραμέτρων με τη σειρά στις θέσεις αυτές. Εφαρμόζοντας κοινή τακτική για όλες τις κλήσεις συναρτήσεων, ξέρουμε πού πρέπει να τοποθετούμε τις τιμές των πραγματικών παραμέτρων πριν από μία κλήση, και πού θα βρίσκεται το αποτέλεσμα της συνάρτησης μετά την κλήση, ενώ η συνάρτηση που καλείται θα ξέρει σε ποια διεύθυνση να επιστρέψει[6].
Μετά το χώρο που δεσμεύσαμε για τις παραμέτρους της A, δεσμεύουμε χώρο για τις τοπικές μεταβλητές στοίβας. Έτσι, δεσμεύουμε χώρο 100 θέσεων τύπου int, και ενημερώνουμε κατάλληλα τον πίνακα συμβόλων για το αναγνωριστικό "x" της εμβέλειας της A.
Η μεταβλητή a είναι στατική, κι επομένως θα τοποθετηθεί στην 12η θέση του χώρου καθολικών δεδομένων. Αυτό δεν την καθιστά καθολική μεταβλητή, αφού δε θα είναι ορατή μόλις βγούμε από την εμβέλεια της συνάρτησης A, αλλά της δίνει διάρκεια ζωής ίδια με τη διάρκεια ζωής των καθολικών μεταβλητών, δηλαδή ίση με το χρόνο εκτέλεσης του προγράμματος. Ο τύπος της θέσης αυτής είναι float.
Όταν μπαίνουμε στη φωλιασμένη εμβέλεια, η δήλωση "unsigned x" δεσμεύει μια νέα θέση στο εγγράφημα δραστηριοποίησης της A, τύπου unsigned. Άσχετα με τις λειτουργίες του πίνακα συμβόλων για το χειρισμό φωλιασμένων εμβελειών, η μεταβλητή x θα τοποθετηθεί απλά στην επόμενη θέση του ίδιου εγγραφήματος δραστηριοποίησης, και όχι σε νέο εγγράφημα. Όσο βρισκόμαστε στην ίδια συνάρτηση, νέες δηλώσεις δεσμεύουν νέες θέσεις, αλλά πάντα στον ίδιο χώρο δεδομένων. Δυνατότητα συμπίεσης των δεσμεύσεων υπάρχει, αν υπάρχουν δηλώσεις μεταβλητών σε διαφορετικές εμβέλειες του ίδιου επιπέδου, για παράδειγμα στις δύο κατευθύνσεις μιας εντολής "if", οι οποίες δε θα μπορούσαν ποτέ να είναι ταυτόχρονα ενεργές, αλλά κάτι τέτοιο είναι αντικείμενο βελτιστοποίησης, και δε μπορεί να γίνει, πριν γίνουν γνωστά τα ακριβή μεγέθη σε ψηφιολέξεις των δεδομένων της συνάρτησης. Για το λόγο αυτό, η ύπαρξη φωλιασμένων εμβελειών μέσα στην ίδια συνάρτηση δεν αυξάνει την πολυπλοκότητα της παραγωγής ενδιάμεσου κώδικα για αυτή.

Με το παραπάνω προσχέδιο του εγγραφήματος δραστηριοποίησης της A, η αναφορά στην παράμετρο i γίνεται πάντα στην τρίτη θέση του χώρου δεδομένων, η αναφορά στη μεταβλητή x μέσα στη φωλιασμένη εμβέλεια γίνεται στην 105η θέση του χώρου δεδομένων, ενώ η αναφορά στην παράμετρο z γίνεται στην τέταρτη θέση του χώρου δεδομένων. Οι αναφορές στη στατική μεταβλητή a και στον πίνακα y γίνονται στο χώρο καθολικών δεδομένων, στις θέσεις 12 και 2-11, αντίστοιχα.
Συμπερασματικά:

- Σε κάθε δήλωση μεταβλητής δεσμεύουμε στον αντίστοιχο χώρο δεδομένων τις θέσεις που απαιτούνται, συμπληρώνοντας τον πίνακα δέσμευσης με τον τύπο και τη δομή της μεταβλητής. Ακόμα, αντιγράφουμε το δείκτη του πίνακα δέσμευσης στον πίνακα συμβόλων, για την επίλυση όλων των αναφορών στη μεταβλητή αυτή στην τρέχουσα εμβέλεια και στην ορατότητά της.
- Ο πίνακας δέσμευσης μιας συνάρτησης αποτελεί εν τέλει συντιθέμενο κατηγόρημα της δήλωσης αυτής, ώστε να παραμείνει προσπελάσιμος μετά την ολοκλήρωση της επεξεργασίας της, οπότε ο πίνακας συμβόλων καθαρίζεται από δηλώσεις της εμβέλειάς της.
- Ο πίνακας δέσμευσης αποτελεί ένα προσχέδιο του αντίστοιχου χώρου δεδομένων και αποτελεί τη “γέφυρα” του πίνακα συμβόλων με τον τελικό κώδικα. Μετά την παραγωγή ενδιάμεσου κώδικα δεν υπάρχει καμία δέσμευση ονομάτων στο πρόγραμμα. Όλες οι μεταβλητές έχουν αντιστοιχηθεί σε κάποια θέση στο συνολικό χώρο δεδομένων του προγράμματος, είτε είναι στατικές είτε είναι μεταβλητές στοίβας.

Πέρα από τις δηλώσεις μεταβλητών, οι δηλώσεις τύπων δε συνδέονται με κάποια λειτουργία όπως αυτή που περιγράψαμε, αλλά χρησιμεύουν για να αποδίδουν κάποια ονόματα σε τύπους, απαιτώντας την εισαγωγή τους στον πίνακα συμβόλων. Η περιγραφή της δομής κάθε τύπου που χρειάζεται ο πίνακας δέσμευσης για τις μεταβλητές αυτού αποθηκεύεται έτσι στον πίνακα συμβόλων. Με την έξοδο από μια εμβέλεια, οι δηλώσεις τύπων της εμβέλειας καταστρέφονται, και οι αντίστοιχες περιγραφές έχουν περάσει στους πίνακες δέσμευσης.

[6] Οι θέσεις αυτές ενδέχεται να μη χρησιμοποιηθούν στον τελικό κώδικα, ανάλογα με το πώς η τελική γλώσσα χρησιμοποιεί τους καταχωρητές της για την υλοποίηση κλήσεων συναρτήσεων.

Τέλος, οι δηλώσεις σταθερών έχουν σαν αποτέλεσμα την εισαγωγή των ονομάτων τους στον πίνακα συμβόλων, και την αντικατάσταση του ονόματος μιας σταθεράς με την τιμή της, κάθε φορά που αυτό χρησιμοποιείται. Εάν υποστηρίζονται σταθερές σύνθετου τύπου που μπορούν να περάσουν σαν παράμετροι σε υποπρογράμματα, απαιτείται δέσμευση στο χώρο δεδομένων γι' αυτές, ώστε να έχουν κάποια διεύθυνση. Τότε, οι σταθερές αυτές χρησιμοποιούνται σα στατικές μεταβλητές στην εμβέλεια στην οποία δηλώνονται, και τοποθετούνται στον εξωτερικό χώρο δεδομένων.

Αρχικοποιήσεις Στατικών Μεταβλητών

Οι αρχικοποιήσεις των στατικών μεταβλητών ενός προγράμματος με κάποιες τιμές πρέπει να γίνονται κατά τη μετάφρασή του. Αντί δηλαδή να παράγεται κώδικας που να αποθηκεύει αυτές τις τιμές στις αντίστοιχες θέσεις του χώρου δεδομένων, ο χώρος δεδομένων σχεδιάζεται με αυτές τις τιμές.
Έτσι, η δήλωση FORTRAN:

```
integer x(100)/100*6/
```

που αποδίδει την τιμή 6 σε όλα τα στοιχεία του πίνακα x, δεν είναι εκτελέσιμη εντολή, αλλά έχει σαν αποτέλεσμα την αρχικοποίηση του χώρου δεδομένων που δεσμεύεται για τον πίνακα x με τις αντίστοιχες τιμές. Παρόμοια για τη δήλωση C που είδαμε νωρίτερα:

```
static float a = -1.0;
```

η τιμή -1.0 αποθηκεύεται στο χώρο δεδομένων στη θέση της μεταβλητής a κατά τη σχεδίαση αυτού.
Γενικά, ένας μεταγλωττιστής παράγει ένα αρχείο, το οποίο περιέχει τον εκτελέσιμο κώδικα, μαζί με το χώρο δεδομένων που αντιστοιχεί σε αρχικοποιημένες στατικές μεταβλητές. Συνήθως ο χώρος δεδομένων τακτοποιείται, ώστε οι αρχικοποιημένες μεταβλητές να τοποθετούνται πρώτες, και οι υπόλοιπες να ακολουθούν, ώστε να μην απαιτείται αποθήκευση στο αρχείο εξόδου του μεταγλωττιστή ολόκληρου του χώρου δεδομένων, παρά μόνο αυτού που αντιστοιχεί στις αρχικοποιημένες μεταβλητές. Πληροφορία που αποθηκεύεται στο αρχείο ενημερώνει το φορτωτή (loader) του λειτουργικού συστήματος για το συνολικό μέγεθος του χώρου δεδομένων του προγράμματος που πρέπει να ενεργοποιηθεί για την εκτέλεσή του.
Έτσι, η δουλειά του μεταγλωττιστή στις αρχικοποιήσεις είναι να συνδέει τις δεσμευμένες θέσεις του χώρου δεδομένων με τις τιμές που δίνονται. Μ' άλλα λόγια, οι πίνακες δέσμευσης επεκτείνονται, ώστε να περιέχουν και τις αποδιδόμενες τιμές, οι οποίες συμπληρώνονται με κάθε τιμή που απαντάται στη λίστα αρχικών τιμών της αντίστοιχης δήλωσης. Στο τέλος της μετάφρασης, κατά την παραγωγή του αρχείου εξόδου του μεταγλωττιστή, αντιγράφονται σε αυτό οι τιμές από τους πίνακες δέσμευσης.
Ας σημειωθεί ότι η επέκταση των πινάκων δέσμευσης αφορά μόνο τους πίνακες όπου δεσμεύονται στατικές μεταβλητές. Στη FORTRAN όλες οι μεταβλητές είναι στατικές, ενώ στην PASCAL δεν υπάρχουν στατικές μεταβλητές. Στη C στατικές είναι όλες οι καθολικές μεταβλητές και όσες τοπικές μεταβλητές είναι δηλωμένες σα στατικές. Οι στατικές μεταβλητές της C δεσμεύονται στον εξωτερικό χώρο δεδομένων.

Παραγωγή Ενδιάμεσου Κώδικα Εκφράσεων

Ο ενδιάμεσος κώδικας σε μορφή αφηρημένου συντακτικού δέντρου που παράγεται για την αποτίμηση εκφράσεων είναι γενικά παρόμοιος με τα δέντρα εκφράσεων, αν και μετά από βελτιστοποιήσεις που βρίσκουν κοινές υποεκφράσεις μπορεί να λάβει μορφή κατευθυνόμενου ακυκλικού γράφου.
Χρησιμοποιώντας το Bison, και με την προϋπόθεση ότι οι παράμετροι προτεραιότητας και προσεταιριστικότητας των τελεστών της αρχικής γλώσσας έχουν δοθεί σωστά, η παραγωγή του ενδιάμεσου κώδικα σε εκφράσεις είναι πολύ εύκολη.
Παραδείγματα σημασιολογικών ρουτινών για την παραγωγή ενδιάμεσου κώδικα σε εκφράσεις είναι τα πιο κάτω:

```
E : E op E { $$ = Create_Tree($2, $1, $3); }
E : ( E ) { $$ = $2; }
```

```
E : id { $$ = Create_Leaf(ID, $1); }
E : const { $$ = Create_Leaf(CONST, $1); }
```

όπου οι συναρτήσεις Create_Tree() και Create_Leaf() δημιουργούν ένα νέο κόμβο του δέντρου[7], και αποδίδουν στα πεδία αυτού κατάλληλες τιμές από τις παραμέτρους που τους δίνονται.
Επιπλέον του παραπάνω κώδικα, οι ρουτίνες θα πρέπει να περιέχουν – αν αυτό είναι απαραίτητο – και τον κώδικα ελέγχου σημασιολογικής ορθότητας και παραγωγής πληροφορίας που προαναφέραμε.

Παραγωγή Ενδιάμεσου Κώδικα Εντολών

Η κύρια διαφορά του κώδικα εντολών με τον κώδικα εκφράσεων – πάντα στη μορφή αφηρημένου συντακτικού δέντρου – είναι ότι δεν έχει τόσο ομοιόμορφη δομή όπως ο παραπάνω. Κάθε εντολή έχει το δικό της τύπο κόμβου, αν και δεν είναι δύσκολο να βρεθεί ένας τύπος που να καλύπτει όλες τις εντολές της αρχικής γλώσσας.
Για παράδειγμα, οι περισσότερες εντολές αναφέρονται σε κάποια έκφραση: ανάθεση, εντολές βρόχου, εντολές διακλάδωσης, εντολές επιλογής. Έτσι, ο τύπος του κόμβου εντολής θα πρέπει να έχει πάντα ένα πεδίο δείκτη σε κώδικα έκφρασης. Παρόμοια, οι περισσότερες εντολές περιέχουν κάποια άλλη εντολή: εντολές διακλάδωσης, εντολές βρόχου, εντολές επανάληψης. Ο τύπος του κόμβου εντολής θα πρέπει επομένως να έχει κι ένα πεδίο δείκτη σε άλλον κώδικα εντολής. Τα δύο αυτά πεδία αρκούν για εντολές όπως η εντολή "while" της Pascal και της C, ή η εντολή "λογικού if" της FORTRAN. Πιο πολύπλοκες εντολές απαιτούν περισσότερα πεδία στους κόμβους τους.
Σε μια σύνθετη εντολή, η σύνδεση διαδοχικών εντολών στην ίδια εμβέλεια μπορεί να γίνει σε γραμμική λίστα, είτε σε δέντρο, με βάση το συνήθη κανόνα:

```
statements → statements statement | statement
```

Θα πρέπει εδώ να σημειωθεί ότι σε αρκετές από τις εντολές των συνηθισμένων γλωσσών προγραμματισμού, η ολοκλήρωση της μετάφρασης σε ενδιάμεσο κώδικα γι' αυτές δε μπορεί να γίνει, πριν την ολοκλήρωση της μετάφρασης κάποιου μπλοκ εντολών. Χαρακτηριστικό παράδειγμα αποτελούν οι εντολές goto, όταν αναφέρονται σε ετικέτες που δεν έχουν ακόμα εμφανιστεί, και οι οποίες θα πρέπει να υπάρχουν μέσα σε κάποιο συγκεκριμένο μπλοκ εντολών, σύμφωνα με τους κανόνες σημασιολογίας της αντίστοιχης γλώσσας. Τότε, εφαρμόζοντας τη μέθοδο του μπαλώματος (backpatching), τοποθετούμε τους κόμβους εντολών που δεν έχουν μεταφραστεί πλήρως σε κάποια λίστα, έναν-έναν καθώς τους παράγουμε, και (α) σε κάθε εμφάνιση ετικέτας διατρέχουμε τη λίστα, ολοκληρώνουμε τη μετάφραση των εντολών που αναφέρονται σε αυτή, και αφαιρούμε τους αντίστοιχους κόμβους, ενώ (β) όταν φτάσουμε στο τέλος του μπλοκ, ανιχνεύουμε σημασιολογικό σφάλμα αν η λίστα μας δεν είναι άδεια[8].

Παραδείγματα

Θα μελετήσουμε στη συνέχεια τρία παραδείγματα υλοποίησης σημασιολογικών ρουτινών σε μια πιο γενική μορφή από αυτή του μετα-εργαλείου Bison, με πολλαπλά κατηγορήματα ανά σύμβολο, αν και πάντα συντιθέμενα, και με αναφορά στα σύμβολα των κανόνων με το όνομά τους. Για μετατροπή σε μορφή κατάλληλη για το Bison, αρκεί κανείς να υλοποιήσει τα πολλαπλά κατηγορήματα ως πολλαπλά πεδία του ίδιου σύνθετου τύπου εγγραφής, χρησιμοποιώ-

[7] Ο τύπος του κόμβου του δέντρου είναι κάποια δομή struct με πεδία που καθορίζονται από τον προγραμματιστή, από τα οποία τουλάχιστον ένα καθορίζει τον τελεστή που αντιστοιχεί στον κόμβο, και δύο είναι δείκτες στα παιδιά του κόμβου. Το φύλλο του δέντρου είναι ίδιου τύπου, όμως οι δείκτες στα παιδιά είναι μηδενικοί (NULL).
[8] Η μέθοδος του μπαλώματος δεν εφαρμόζεται μόνο για εντολές, αλλά και για δηλώσεις. Για παράδειγμα, η επεξεργασία δηλώσεων υποπρογραμμάτων με προαναγγελία (Pascal), πρωτότυπο (C) ή απ' ευθείας κλήση τους (FORTRAN) γίνεται με αυτή τη μέθοδο.

ντας ως τελικό μοναδικό κατηγόρημα κάποιο δείκτη στον τύπο αυτόν, καθώς και να χρησιμοποιήσει το συμβολισμό και την αρίθμηση των κατηγορημάτων που ορίζει το Bison.

Παράδειγμα 1.

Ας θεωρήσουμε την παρακάτω γραμματική αριθμητικών εκφράσεων κάποιας γλώσσας προγραμματισμού:

```
expr → expr ADDOP expr | expr MULOP expr | '(' expr ')' | ID
```

όπου ADDOP η λεκτική μονάδα των τελεστών πρόσθεσης και αφαίρεσης, MULOP η λεκτική μονάδα των τελεστών πολλαπλασιασμού και διαίρεσης, και ID η λεκτική μονάδα των αναγνωριστικών της γλώσσας.
Το σύστημα τύπων της γλώσσας περιλαμβάνει δύο βασικούς αριθμητικούς τύπους, των ακεραίων και των πραγματικών, και κάποιους σύνθετους τύπους, μεταξύ των οποίων είναι και ένας τύπος δεικτών. Ας υποθέσουμε ότι η σημασιολογική περιγραφή των αριθμητικών εκφράσεων της γλώσσας περιλαμβάνει τα εξής:

(α) Οι τύποι που συμμετέχουν στις αριθμητικές εκφράσεις είναι είτε βασικοί, ακέραιοι ή πραγματικοί, είτε δείκτες.
(β) Τα τελούμενα των πράξεων πρέπει να είναι βασικού τύπου, εκτός από τις πράξεις πρόσθεσης και αφαίρεσης, όπου μπορεί το ένα τελούμενο να είναι τύπου δείκτη, αλλά τότε το άλλο πρέπει να είναι ακέραιου τύπου.
(γ) Αν σε μια πράξη μεταξύ βασικών τύπων συμμετέχει πραγματικός τύπος, το αποτέλεσμα είναι πραγματικού τύπου.
(δ) Αν σε μια πράξη συμμετέχει τύπος δείκτη, το αποτέλεσμα είναι του ίδιου τύπου δείκτη.
(ε) Τα αναγνωριστικά που συμμετέχουν σε μια έκφραση πρέπει να έχουν δηλωθεί νωρίτερα.
(στ) Οι τελεστές MULOP έχουν μεγαλύτερη προτεραιότητα από τους τελεστές ADDOP, ενώ όλοι έχουν αριστερή προσεταιριστικότητα.
(ζ) Οι πράξεις μέσα σε παρενθέσεις έχουν πάντα μεγαλύτερη προτεραιότητα από τις πράξεις που βρίσκονται έξω από αυτές.

Για τη σημασιολογική ανάλυση των αριθμητικών εκφράσεων, κατ' αρχήν θα ενσωματώσουμε τις ιδιότητες προτεραιότητας και προσεταιριστικότητας των τελεστών στη γραμματική της γλώσσας, ώστε να γίνει μη διφορούμενη. Για τη γενική περίπτωση, ανεξάρτητα δηλαδή του μετα-εργαλείου συντακτικής ανάλυσης που χρησιμοποιούμε, θα προχωρήσουμε σε μετασχηματισμό της γραμματικής. Σε ψηλότερα επίπεδα θα διατηρήσουμε τελεστές χαμηλότερης προτεραιότητας, ενώ με αριστερή αναδρομή θα εξασφαλίσουμε την αριστερή προσεταιριστικότητα. Έτσι, θα καταλήξουμε στη γραμματική:

```
expr → expr ADDOP term | term
term → term MULOP factor | factor
factor → '(' expr ')' | ID
```

Ας σημειωθεί ότι η μορφή της γραμματικής μας εξασφαλίζει αυτόματα τη σημασιολογία των παρενθέσεων, κι έτσι δε χρειάζεται καμία παρέμβαση γι' αυτό.
Στη συνέχεια, για κάθε σύμβολο της γραμματικής θα ορίσουμε τα απαιτούμενα κατηγορήματα. Έτσι, τα τρία μη τερματικά σύμβολα θα έχουν τουλάχιστον δύο κατηγορήματα, το ένα – έστω t_code – θα δίνει τον κωδικό τύπου, και το άλλο – έστω t_struct – θα δίνει τη δομή τύπου. Ας υποθέσουμε ότι οι δύο βασικοί τύποι που μας ενδιαφέρουν έχουν κωδικούς T_INT και T_REAL, ενώ ο τύπος των δεικτών έχει κωδικό T_POINT. Η μορφή του κατηγορήματος t_struct δε θα μας απασχολήσει εδώ.
Το τερματικό σύμβολο ID θα έχει ένα κατηγόρημα – έστω name – που θα δίνει το όνομα του αναγνωριστικού. Υποθέστε ότι διαθέτουμε κάποια συνάρτηση αναζήτησης – έστω lookup() – που δεδομένου ενός ονόματος, μας επιστρέφει κάποιο δείκτη στον πίνακα συμβόλων του μεταγλωττιστή. Αν το αναγνωριστικό δεν έχει οριστεί νωρίτερα, η συνάρτηση μας επιστρέφει μηδενική τιμή. Κάθε στοιχείο του πίνακα συμβόλων περιέχει τουλάχιστον δύο πεδία που αποθηκεύουν πληροφορία για τον τύπο του αναγνωριστικού, τα οποία για απλούστευση θα

θεωρήσουμε ότι έχουν το ίδιο όνομα με τα δύο κατηγορήματα των μη τερματικών συμβόλων που ορίσαμε νωρίτερα.
Ο σημασιολογικός αναλυτής θα πρέπει με βάση τα παραπάνω να ελέγχει την ορθότητα των τύπων που συμμετέχουν σε μια αριθμητική έκφραση, καθώς επίσης και να αποτιμά τα κατηγορήματα του αριστερού μέλους κάθε συντακτικού κανόνα, ώστε αυτά να μπορούν να χρησιμοποιηθούν σε ευρύτερες εκφράσεις. Σε μορφή σχήματος μετάφρασης, μια υλοποίηση σε ψευδοκώδικα C των σημασιολογικών ρουτινών που χρειαζόμαστε θα είναι:

```
expr0 → expr1 ADDOP term {
   if (expr1.t_code == T_POINT) {
      if (term.t_code != T_INT) sem_error(1);
      else {
         expr0.t_code = T_POINT;
         expr0.t_struct = expr1.t_struct;
      }
   } else if (term.t_code == T_POINT) {
      if (expr1.t_code != T_INT) sem_error(1);
      else {
         expr0.t_code = T_POINT;
         expr0.t_struct = term.t_struct;
      }
   } else if (((expr1.t_code != T_REAL) &&
               (expr1.t_code != T_INT)) ||
              ((term.t_code != T_REAL) &&
               (term.t_code != T_INT)))
      sem_error(2);
   else if ((expr1.t_code == T_REAL) ||
            (term.t_code == T_REAL))
      expr0.t_code = T_REAL;
   else expr0.t_code = T_INT;
}

expr → term {
   expr.t_code = term.t_code;
   expr.t_struct = term.t_struct;
}

term0 → term1 MULOP factor {
   if (((term1.t_code != T_REAL) && (term1.t_code != T_INT)) ||
       ((factor.t_code != T_REAL) && (factor.t_code != T_INT)))
      sem_error(3);
   else if ((term1.t_code == T_REAL) ||
            (factor.t_code == T_REAL))
      term0.t_code = T_REAL;
   else term0.t_code = T_INT;
}

term → factor {
   term.t_code = factor.t_code;
   term.t_struct = factor.t_struct;
}

factor → '(' expr ')' {
   factor.t_code = expr.t_code;
   factor.t_struct = expr.t_struct;
}

factor → ID {
   symbol_table_item * entry;
   if (!(entry = lookup(ID.name))) sem_error(4);
   else {
      factor.t_code = entry->t_code;
      factor.t_struct = entry->t_struct;
```

```
    }
}
```

όπου sem_error() κατάλληλη συνάρτηση διαχείρισης σημασιολογικών σφαλμάτων, που στην απλούστερη περίπτωση δέχεται σαν παράμετρο κάποιον κωδικό σφάλματος και εκτυπώνει κατάλληλο μήνυμα, τερματίζοντας τη μετάφραση, ενώ symbol_table_item είναι ο τύπος των στοιχείων του πίνακα συμβόλων του μεταγλωττιστή .
Παρατηρήστε ότι στην παραπάνω ανάλυση δε χρειάστηκε να διαχωρίσουμε τους τελεστές πρόσθεσης και αφαίρεσης, ή πολλαπλασιασμού και διαίρεσης, μεταξύ τους, εφ' όσον δε διαφέρουν σημασιολογικά, τουλάχιστον όσο αφορά τους τύπους των τελούμενών τους. Αν όμως προχωρούσαμε σε παραγωγή ενδιάμεσου κώδικα, θα έπρεπε να τους διαχωρίσουμε, και επομένως θα χρειαζόμασταν ένα κατάλληλο κατηγόρημα για τα τερματικά σύμβολα ADDOP και MULOP, το οποίο θα περνούσαμε στην αντίστοιχη σημασιολογική ρουτίνα.

Παράδειγμα 2.

Ας θεωρήσουμε τώρα την πιο κάτω γραμματική των εντολών διακλάδωσης κάποιας γλώσσας προγραμματισμού:

```
if_stmt → KEY_IF expr KEY_THEN stmt if_tail
if_tail → KEY_ELSE stmt | ε
```

όπου KEY_IF, KEY_THEN και KEY_ELSE οι λεκτικές μονάδες των αντίστοιχων λέξεων-κλειδιά, expr το μη τερματικό σύμβολο των εκφράσεων, και stmt το μη τερματικό σύμβολο των εντολών της γλώσσας.
Το σύστημα τύπων της γλώσσας περιλαμβάνει εκτός των άλλων ένα βασικό τύπο λογικών εκφράσεων. Ας υποθέσουμε ότι η σημασιολογική περιγραφή των εντολών διακλάδωσης της γλώσσας περιλαμβάνει τα εξής:

(α) Ο τύπος της έκφρασης στην εντολή διακλάδωσης πρέπει να είναι λογικός.
(β) Αν το αποτέλεσμα της αποτίμησης της έκφρασης είναι TRUE, εκτελείται η εντολή που ακολουθεί τη λέξη-κλειδί THEN, διαφορετικά, και αν υπάρχει λέξη-κλειδί ELSE, εκτελείται η εντολή που την ακολουθεί.

Ο έλεγχος ορθότητας μιας εντολής διακλάδωσης, όσο αφορά τον τύπο της έκφρασης, είναι πολύ απλός. Για την υλοποίησή του θέλουμε τουλάχιστον ένα κατηγόρημα για το σύμβολο expr, έστω t_code, το οποίο να μας δίνει τον κωδικό τύπου της έκφρασης. Έστω ότι ο κωδικός του λογικού τύπου είναι ο T_BOOL.
Ας υποθέσουμε ότι πέρα από τον έλεγχο ορθότητας, θέλουμε να παράγουμε και τον αντίστοιχο ενδιάμεσο κώδικα. Έτσι, θα χρειαστούμε ένα κατηγόρημα για όλα τα μη τερματικά σύμβολα της πιο πάνω γραμματικής, έστω i_code, το οποίο να είναι ένας δείκτης σε κάποια μορφή ενδιάμεσου κώδικα. Υποθέστε ότι διαθέτουμε μια συνάρτηση – έστω create_if_code(), η οποία παράγει τον ενδιάμεσο κώδικα μιας εντολής διακλάδωσης, δεδομένων τριών δεικτών στους ενδιάμεσους κώδικες της έκφρασης, της πρώτης και της δεύτερης εντολής.
Ο σημασιολογικός αναλυτής θα πρέπει επομένως να ελέγχει την ορθότητα του τύπου της έκφρασης, και αν αυτός είναι σωστός, να αποτιμά το κατηγόρημα του συμβόλου if_stmt. Έτσι, μια υλοποίηση των σημασιολογικών ρουτινών που χρειαζόμαστε, σε μορφή σχήματος μετάφρασης και σε ψευδοκώδικα C, θα είναι:

```
if_stmt → KEY_IF expr KEY_THEN stmt if_tail {
   if (expr.t_code != T_BOOL) sem_error(1);
   else if_stmt.i_code =
      create_if_code(expr.i_code, stmt.i_code, if_tail.i_code);
}

if_tail → KEY_ELSE stmt {
   if_tail.i_code = stmt.i_code;

}
if_tail → ε {
```

```
      if_tail.i_code = 0;
   }
```

όπου όπως και προηγουμένως, sem_error() είναι κατάλληλη ρουτίνα διαχείρισης σημασιολογικών σφαλμάτων.
Παρατηρήστε ότι ο πιο πάνω κώδικας καθυστερεί στην αναγνώριση σφάλματος στον τύπο της έκφρασης. Θα μπορούσαμε εναλλακτικά να υλοποιήσουμε τον πρώτο κανόνα ως εξής:

```
if_stmt → KEY_IF expr {
   if (expr.t_code != T_BOOL) sem_error(1);
} KEY_THEN stmt if_tail {
   if_stmt.i_code =
      create_if_code(expr.i_code, stmt.i_code, if_tail.i_code);
}
```

οπότε τυχόν σφάλμα ανιχνεύεται αμέσως μετά τη συντακτική αναγνώριση της έκφρασης, πριν προχωρήσουμε στη συντακτική αναγνώριση της μίας ή των δύο εντολών που συμπληρώνουν την εντολή διακλάδωσης.

Παράδειγμα 3.

Ας μελετήσουμε τέλος τη γραμματική των δηλώσεων μεταβλητών κάποιας γλώσσας προγραμματισμού:

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

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

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

Για τη σημασιολογική ανάλυση των δηλώσεων, θα υποθέσουμε ότι το μη τερματικό σύμβολο type έχει τα δύο κατηγορήματα που είδαμε νωρίτερα, t_code και t_struct. Οι κωδικοί των βασικών τύπων είναι T_INT και T_FLOAT, ενώ ο κωδικός T_ARRAY περιγράφει τύπο πίνακα. Μια δομή αναπαράστασης τύπου για πίνακες θα πρέπει να περιέχει πληροφορίες για τα μεγέθη των διαστάσεων και τον τύπο των στοιχείων του πίνακα, και να κατασκευάζεται με κατάλληλη συνάρτηση – έστω την create_array(), η οποία δέχεται ως παραμέτρους τον κωδικό και τη δομή τύπου του στοιχείου του πίνακα και επιστρέφει τη νέα δομή.
Το τερματικό σύμβολο ICONST θα έχει ένα κατηγόρημα – έστω val – που θα δίνει την τιμή της σταθεράς, ενώ το τερματικό σύμβολο ID θα έχει ένα κατηγόρημα – έστω name – που θα δίνει το όνομα του αναγνωριστικού. Θεωρήστε ότι διαθέτουμε κάποια συνάρτηση αναζήτησης – έστω lookup() – που δεδομένου ενός ονόματος, μας επιστρέφει κάποιο δείκτη στον πίνακα συμβόλων, μηδενικής τιμής αν το αναγνωριστικό δεν έχει οριστεί νωρίτερα. Ακόμα, διαθέτουμε και κάποια συνάρτηση εισαγωγής – έστω insert() – που δεδομένου κάποιου ονόματος, το εισάγει στον πίνακα συμβόλων του μεταγλωττιστή, επιστρέφοντας ένα δείκτη στο

αντίστοιχο στοιχείο. Κάθε στοιχείο του πίνακα συμβόλων περιέχει τουλάχιστον τέσσερα πεδία που αποθηκεύουν πληροφορία για τον τύπο του αναγνωριστικού, τα δύο από τα οποία για απλούστευση θα θεωρήσουμε ότι έχουν το ίδιο όνομα με τα δύο κατηγορήματα του συμβόλου type, το τρίτο – έστω t_type – θα δίνει την πληροφορία αν το αναγνωριστικό αντιστοιχεί σε όνομα τύπου ή όχι, ενώ το τέταρτο – έστω t_dims – θα δίνει τον αριθμό διαστάσεων του αναγνωριστικού.

Το μη τερματικό σύμβολο dims θα πρέπει να έχει ένα κληρονομούμενο κατηγόρημα, το οποίο να δείχνει στο στοιχείο του πίνακα συμβόλων που μόλις έχουμε εισάγει, ώστε με κάθε νέα διάσταση να προστίθεται σ' αυτό η νέα πληροφορία. Εναλλακτικά, μπορούμε να μετασχηματίσουμε τη γραμματική σε άλλη ισοδύναμή της, στην οποία να μην απαιτείται κληρονομούμενο κατηγόρημα[9]. Τέτοιος μετασχηματισμός δεν είναι πάντα εφικτός, και γι' αυτό θα προτιμήσουμε μια άλλη λύση, αυτή της χρήσης καθολικών μεταβλητών στο μεταγλωττιστή. Μια τέτοια λύση μπορεί να χρησιμοποιηθεί εδώ, διότι η γραμματική δεν επιτρέπει μια νέα δήλωση να αρχίσει, πριν ολοκληρωθεί η προηγούμενη, κι έτσι δεν υπάρχει κίνδυνος λάθους αποτίμησης στις όποιες καθολικές μεταβλητές χρησιμοποιήσουμε.

Έστω λοιπόν cur_item μια καθολική μεταβλητή δείκτη στον πίνακα συμβόλων, η οποία δείχνει στο πιο πρόσφατο στοιχείο που εισάγουμε σε αυτόν. Με τη βοήθεια της μεταβλητής αυτής θα μπορούμε να προσθέτουμε διαστάσεις σε ένα όνομα του πίνακα συμβόλων. Θα αποφύγουμε προγραμματιστικές λεπτομέρειες και θα θεωρήσουμε ότι η προσθήκη διάστασης γίνεται με κατάλληλη συνάρτηση – έστω add_dim(), η οποία δέχεται ως παραμέτρους ένα δείκτη σε δομή τύπου πίνακα και μια ακέραια σταθερά που αποτελεί το μέγεθος της νέας διάστασης.

Ο σημασιολογικός αναλυτής θα πρέπει με βάση τα παραπάνω να ελέγχει την ορθότητα των δηλώσεων, και να εισάγει στον πίνακα συμβόλων όλες τις πληροφορίες που προκύπτουν σε κάθε δήλωση. Σε μορφή σχήματος μετάφρασης, μια υλοποίηση σε ψευδοκώδικα C των σημασιολογικών ρουτινών που χρειαζόμαστε θα είναι:

```
var_decl → type ID {
   if (lookup(ID.name)) sem_error(1);
   cur_item = insert(ID.name);
   cur_item->t_code = type.t_code;
   cur_item->t_struct = type.t_struct;
   cur_item->t_type = 0;
   cur_item->t_dims = 0;
} dims ';'

type → KEY_INT {
   type.t_code = T_INT;
}

type → KEY_FLOAT {
   type.t_code = T_FLOAT;
}

type → ID {
   symbol_table_item * entry;
   if (!(entry = lookup(ID.name))) sem_error(2);
   else if (!entry->t.type) sem_error(3);
   else {
      type.t_code = entry->t_code;
      type.t_struct = entry->t_struct;
```

[9] Μια κατάλληλη ισοδύναμη γραμματική θα ήταν για παράδειγμα η ακόλουθη:

```
var_decl → var_decl_body ';'
var_decl_body → var_decl_body '[' ICONST ']' | type ID
type → INT | FLOAT | ID
```

όπου εξασφαλίζεται ότι το σύμβολο type θα βρίσκεται κάτω από όλες τις εμφανίσεις του συμβόλου var_decl_body στο δέντρο συντακτικής ανάλυσης, κι έτσι δε χρειαζόμαστε κάποιο κληρονομούμενο κατηγόρημα, για να περάσουμε τις πληροφορίες τύπου από το πρώτο στο δεύτερο σύμβολο.

```
   }
}
dims → dims '[' ICONST ']' {
   if (cur_item->t_dims == 5) sem_error(4);
   else if (ICONST.val == 0) sem_error(5);
   else if (cur_item->t_dims == 0) {
      type_struct_item * new_struct =
         create_array(cur_item->t_code, cur_item->t_struct);
      cur_item->t_code = T_ARRAY;
      cur_item->t_struct = new_struct;
      add_dim(cur_item->t_struct, ICONST.val);
      cur_item->t_dims = 1;
   }
   else {
      add_dim(cur_item->t_struct, ICONST.val);
      cur_item->t_dims++;
   }
}
dims → ε
```

όπου όπως και προηγουμένως, sem_error() είναι κατάλληλη ρουτίνα διαχείρισης σημασιολογικών σφαλμάτων και symbol_table_item είναι ο τύπος των στοιχείων του πίνακα συμβόλων, ενώ type_struct_item είναι ο τύπος των δομών αναπαράστασης τύπου του μεταγλωττιστή. Ο τελευταίος κανόνας δε χρειάζεται σημασιολογική ρουτίνα, αφού για δηλώσεις βαθμωτών μεταβλητών οι πληροφορίες τύπου έχουν ήδη αποδοθεί στο αναγνωριστικό και αποθηκευτεί στον πίνακα συμβόλων.