Εργασία Μαθήματος

Γλώσσες Προγραμματισμού και Μεταγλωττιστές

Αλέξανδρος Αθανασιάδης
ΑΕΜ: 10006
ΤΗΜΜΥ ΑΠΘ

3 Οκτωβρίου 2025

Εισαγωγή

Αυτή η εργασία πραγματεύεται την σχεδίαση και υλοποίηση ενος απλού μεταγλωττιστή που αποδέχεται μια γλώσσα παρόμοια της `C` και στοχέυει στον "μυθικό υπολογιστή" `MIX` που περιγράφει ο Donald E. Knuth[1] στην βραβευμένη σειρά βιβλίων του "Η Τέχνη του Προγραμματισμού". Ο `MIX` είναι ενας υπολογιστής με αρχιτεκτονική πολύ κοντά σε αυτή των υπολογιστών των δεκαετιών του ′60 και ′70, έχοντας όμως το προτέρημα ότι είναι "μυθικός" και επομένως δεν χρειάζεται να μπλέκει με τις πολυπλοκότητες του πραγματικού κόσμου (όπως η δυσκολία υλοποίησης ή το κόστος κατασκευής). Γι′ αυτό πιθανώς είναι και πιο "όμορφος" σύμφωνα με τον σχεδιαστή του.

Ο μεταγλωττιστής της εργασίας λαμβάνει είσοδο στην γλώσσα υψηλού επιπέδου που θα περιγραφεί παρακάτω και επιστρέφει έξοδο στην `MIXAL`, την γλώσσα συμβολομετάφρασης του `MIX`. Η `MIXAL` είναι αρκετά κοντά στην γλώσσα μηχανής του `MIX` όμως εμπεριέχει ορισμένες πολυπλοκότητες, όπως η παρουσία ψευδοεντολών, σταθερών περιεχομένων μνήμης, κ.α. - επομένως η μεταγλώττιση αυτής σε γλώσσα μηχανής ανατίθεται στο `MIX Development Kit (mdk)` του `GNU Project`, το οποίο πέραν του `MIX` assembler (`mixasm`) παρέχει και το κατάλληλο `MIX` virtual machine (`mixvm`) που τρέχει τις παραγόμενες εντολές μηχανής σε εναν προσομοιωτή του υπολογιστή `MIX`.

1 Η γλώσσα υψηλού επιπέδου

Όπως προαναφέρθηκε η γλώσσα υψηλού επιπέδου που θέλουμε να μεταγλωττίσουμε έχει συντακτικό κοντα σε αυτο της `C`, είναι όμως πολυ πιο περιορισμένη απο αυτήν.

Όσον αφορα το συντακτικό της γλώσσας, αυτη επιτρέπει

- την δημιουργία και κλήση συναρτήσεων
- την δημιουργία και χρήση μεταβλητών τύπου `int`
- βασικές αριθμητικές πράξεις (`+`,`-`,`*`,`/`) μεταξύ μεταβλητών και σταθερών
- πράξεις σύγκρισης (`<`,`>`,`==`,...) μεταξύ μεταβλητών και σταθερών
- έλεγχο ροής με `if-else` αλλα και επαναληπτικούς βρόγχους `while`

Η γλώσσα επιβάλλει στις μεταβλητές τοπικό scope στην διάρκεια μιας μεθόδου, ενώ αυτές πρέπει

[1] Ο Donald Knuth είναι θρυλικός επιστήμονας και συγγραφέας στον τομέα της επιστήμης υπολογιστών, και δημιουργός της μηχανής στοιχειοθεσίας TEX, με χρήση της οποίας γράφτηκε αυτή η αναφορά

να ορίζονται στην αρχή της μεθόδου.[2] Ο μόνος επιτρεπτός τύπος μεταβλητής είναι `int`, ενω δεν υπάρχει δυνατότητα δημιουργίας πινάκων (`array`), δεικτών (`pointer`) ή δομών δεδομένων (`struct`).

Επιτρέπεται η αναδρομική κλήση συναρτήσεων, αλλα η κλήση μιας συνάρτησης απο κάποια άλλη εξαρτάται απο την σειρά ορισμού τους, αφου δεν υποστηρίζεται "forward declaration" συναρτήσεων. Κάθε αρχείο πηγαίου κώδικα πρέπει να περιέχει συνάρτηση `main` οπου είναι το σημείο εισόδου στο πρόγραμμα. Η απουσία linker κάνει αδύνατη την συρραφή κώδικα πολλών αρχείων, και επομένως κάθε αρχείο αποτελεί πλήρες πρόγραμμα απο μόνο του, και περιέχει τον κώδικα για όλες τις απαραίτητες υπορουτίνες που μπορεί να χρειαστεί.

Δεν υπάρχει καμία υποστήριξη για Είσοδο/Έξοδο (`I/O`) παρόλο που ο `MIX` υποστηρίζει E/E για πολλές μορφές συσκευών. Με πρωτοβουλία του συγγραφέα, ο κώδικας `MIXAL` που δημιουργεί ο μεταγλωττιστής τυπώνει στο τέλος του προγράμματος την τιμή επιστροφής της συνάρτησης `main`.

2 Ο Μεταγλωττιστής

Με βάση τα παραπάνω το πρόβλημα που καλείται να υλοποιήσει ο μεταγλωττιστής λοιπόν, είναι η ανάλυση του πηγαίου κώδικα υψηλού επιπέδου του προγράμματος εισόδου και η επιστροφή κώδικα `MIXAL` που αντιστοιχεί στο πρόγραμα εισόδου και το υλοποιεί.

Η δουλειά χωρίζεται κατα κύριο λόγο σε 3 στάδια χωριστών "περασμάτων", κάθε ένα απο τα οποία πρέπει να λήξει πρίν ξεκινήσει το επόμενο 1. Γλωσσική ανάλυση και κατασκευή του Συντακτικού Δέντρου (AST) 2. Σημασιολογική ανάλυση και κατασκευή Πινάκων Συμβόλων (Symbol Tables) και 3. Παραγωγή κώδικα `MIXAL` βάση του AST. Καθένα απο αυτά τα στάδια θα αναλυθούν στις ενότητες που ακολουθούν.

2.1 Γλωσσική ανάλυση

Η Γλωσσική Ανάλυση στοχεύει στην ανάλυση της Λεξιλογικής και Γραμματικής δομής του προγράμματος εισόδου, και την κατασκευή του Συντακτικού Δέντρου.

Η λεξιλογική ανάλυση διαχωρίζει και ταξινομεί τα σύμβολα εισόδου σε "λέξεις" ή "tokens", όπως όνομα μεταβλητής (`x`, `foo`, `_temp`), κυριολεκτική αριθμητική τιμή (`0`, `42`, `3`), σύμβολο πράξης (`+`, `/`, `>=`) ή κωδική λέξη της γλώσσας (`if`, `int`, `return`).

Το σύνολο των λέξεων που μπορεί να απεικονίζει ένα `token` ορίζεται ως μια Κανονική Γλώσσα με χρήση Κανονικών Εκφράσεων και επομένως η ταξινόμηση μπορεί να γίνει με χρήση Πεπερασμένων Αυτόματων (FSA).

Απο την άλλη η γραμματική ανάλυση αποσκοπεί στην ανάλυση των γραμματικών σχέσεων μεταξύ `tokens`, δηλαδή ανωτέρου τύπου σχέσεις για την συντακτική του προγράμματος, όπως για παράδειγμα: `ADDEXPR` → `ADDEXPR ADDOP TERM` | `TERM`. Αυτές οι εκφράζεις ορίζουν γραμματικές χωρίς συμφραζόμενα (CFG) που μπορούν να γίνουν αποδεκτές απο Αυτόματα Στοίβας (PDA).[3]

Η γραμματική της γλώσσας είναι σχεδόν ίδια με αυτή που δίνεται στην εκφώνηση της εργασίας, με δύο σημαντικές αλλαγές. 1. Τη μεταφορά των `ADDOP`, `MULOP`, `RELOP` σε λεξιλογικά `tokens` αντί για γραμματικά μη τερματικά σύμβολα και 2. Τη δημιουργία ενος νέου μη τερματικού συμβόλου `UNARY`, ενα επίπεδο προτεραιότητας κάτω απο το `FACTOR`, και κωδικοποιεί τις πράξεις μιας μεταβλητής. Αυτό ερχεται να αντικαταστήσει το προαιρετικό `'-'` στον ορισμό του `token` αριθμού, που προκαλούσε

[2] Παρόμοια με τις εκδόσεις της `C` πριν την `C99`

[3] Συγκεκριμένα, συχνά χρησιμοποιείται ενα συγκεκριμένο είδος ντετερμινιστικού αυτόματου στοίβας που ονομάζεται LR parser. Αναλύουν μια ΓΧΣ σε γραμμικό χρόνο, ενώ οι πίνακες μετάβασης μπορούν να κατασκευαστούν μηχανικά απο την γραμματική της γλώσσας. Οι LR parsers δημιουργήθηκαν απο τον Donald Knuth το 1965 και κάποια μορφή τους χρησιμοποιείται σήμερα στους περισσότερους parser generators

προβλήματα σε περιπτώσεις οπου υπήρχαν εκφράσεις όπως η `a-3`, η οποία θα αναλύονταν στη μεταβλητή `a` και έπειτα στον αριθμό `-3` και θα δημιουργούσε συντακτικό λάθος. Επομένως πλέον οι αρνητικοί αριθμοί κωδικοποιούνται σαν γραμματικό χαρακτηριστικό αντι για λεξιλογικό.

Για παράδειγμα

`-10` → `UNARY(ADDOP{-} UNARY(FACTOR(NUMBER{10}))).`

Όμως

`x-10` → `ADDEXPR(ADDEXPR(...(LOCATION{x})) ADDOP{-} TERM(...(NUMBER{10}))).`

2.1.1 Δημιουργία Λεξιλογικού και Γραμματικού αναλυτή

Στον μοντέρνο κόσμο[4] η γλωσσική ανάλυση γίνεται με χρήση εργαλείων που ονομάζονται compiler-compilers ή parser generators που λαμβάνουν την υψηλού επιπέδου περιγραφή της γλώσσας (σε μορφή κανονικών εκφράσεων ή γραμματικών κανόνων) και δημιουργούν αντίστοιχο κώδικα που υλοποιεί τον λεξιλογικό και γραμματικό αναλυτή της συγκεκριμένης γλώσσας.

Στα πλαίσια της εργασίας χρησιμοποιήθηκαν τα εργαλεία `flex` και `bison` για την δημιουργία του λεξιλογικού και γραμματικού αναλυτή αντίστοιχα. Τα εργαλεία αυτά βρίσκονται σε στενή σύνεργασία μεταξύ τους, και μάλιστα στην τυπική λειτουργία, ο γραμματικός αναλυτής του `bison` καλεί τον λεξιλογικό αναλυτή του `flex` για να επιστρέψει το επόμενο `token` του προγράμματος εισόδου, και βάσει αυτού να αποφασίσει για την γραμματική της γλώσσας. Γιάυτο τον λόγο η λεξιλογική και η γραμματική ανάλυση δεν περιγράφονται ως δυο χωριστά στάδια –αν και θα μπορούσαν να λειτουργούν έτσι– αλλα ως μέρος της ίδιας διαδικασίας, της γλωσσικής ανάλυσης.

2.1.2 Κατασκευή Συντακτικού Δέντρου

Σκοπός της γλωσσικής ανάλυσης είναι η κατασκευή του συντακτικού δέντρου, το οποίο ορίζεται στον κώδικα του μεταγλωττιστή ως μια μεικτή δομή δέντρου και λίστας, που περιγράφει τις ιδιαιτερότητες της συγκεκριμένης γλώσσας. Το συντακτικο δέντρο με τον τρόπο που έχει οριστεί δεν αποσκοπεί στην άμεση αναπαράσταση της γραμματικής της γλώσσας με βάση τους γραμματικούς κανόνες, αλλα έχει σχεδιαστεί ωστε να εξυπηρετεί στην μετέπειτα επεξεργασία της σημασιολογικής ανάλυσης και της παραγωγής κώδικα.

Κάθε κόμβος του δέντρου έχει εναν τύπο (π.χ. `N_METHOD` αντιπροσωπευει το είδος κόμβου για ορισμό μεθόδου), τις απαραίτητες πληροφορίες για τον συγκεκριμένο κόμβο (π.χ. το όνομα της μεθόδου που ορίζει ο κόμβος) και δείκτες στους κόμβους-παιδιά του συγκεκριμένου κόμβου (π.χ. τις παραμέτρους της μεθόδου). Οι λίστες χρησιμοποιούνται όταν κάποιος γραμματικός κανόνας περιγράφει την επανάληψη ομοίων συμβόλων το ένα μετα το άλλο, για ευκολία στην υλοποίηση και αποφυγή "ατέρμονων" κλαδιών του δέντρου.

Για την κατασκευή του δέντρου χρησιμοποιείται η ίδια διαδικασία με την οποία ο `bison` αποδέχεται τον κατάλληλο γραμματικό κανόνα, και ακολουθείται απο ενα "action", το οποίο για κάθε κανόνα ορίζω ως την κατασκευή του κατάλληλου κόμβου για το συντακτικό δέντρο και την σύνδεση του με το ήδη κατασκευασμένο δέντρο.

Για μια αναλυτικότερη περιγραφή των ειδών κόμβων που υπάρχουν στο Συντακτικό Δέντρο και την χρήση τους ανατρέξτε στον πηγαίο κώδικα του μεταγλωττιστή.

[4] Δηλαδη περίπου απο τις αρχές της δεκαετίας του ′60. Συγκεκριμένα ο `yacc`, ο "επίσημος" compiler-compiler του `Unix` και της `C` δημιουργήθηκε στις αρχές του ′70 και εκδόθηκε πρώτη φορά το 1975. Ο `bison` που χρησιμοποιήθηκε για την δημιουργία του μεταγλωττιστή της εργασίας, είναι κατα κάποιο τρόπο η "ελεύθερη" έκδοση του `yacc` ως μέρος του `GNU Project`

2.2 Σημασιολογική Ανάλυση

Το δεύτερο στάδιο της επεξεργασίας του προγράμματος εισόδου είναι η σημασιολογική ανάλυση. Αυτή αποσκοπεί στην κατασκευή των Πινάκων Συμβόλων, αλλα και τον έλεγχο του κώδικα εισόδου για "σημασιολογικά" λάθη (semantic errors), δηλαδή για λάθη που είναι αφενός συντακτικά σωστά αλλα γίνονται λάθος όταν διαβάσεις τα συμφραζόμενα του προγράμματος εισόδου.[5] Τέτοια λάθη μπορεί να είναι η κλήση μιας συνάρτησης που δεν υπάρχει, η προσπέλαση κάποιας μεταβλητής που δεν έχει οριστεί, ο διπλός ορισμός καποιας συνάρτησης ή μεταβλητής, κ.ο.κ.

Η ανίχνευση σημασιολογικών λαθών λοιπόν απαιτεί μια γνώση των συμφραζόμενων ή "context" του προγράμματος. Γι΄ αυτό τον λογο κατασκευάζουμε τους πίνακες συμβολών, που περιέχουν τις συναρτήσεις και τις μεταβλητές του προγράμματος, καθώς και πληροφορίες για αυτές, χρήσιμες για την παραγωγή κώδικα αργότερα, όπως η θέση που ξεκινάει μια συνάρτηση, ο αριθμός των παραμέτρων της, κ.α.

Η δομή δεδομένων που χρησιμοποιείται για τους Πίνακες Συμβόλων είναι το Hash Table, όπου key είναι το όνομα της συνάρτησης ή μεταβλητής. Για την επίλυση προβλημάτων σχετικά με συγκρούσεις hash μεταξύ δυο κλειδιών χρησιμοποιώ σε κάθε θέση του Hash Table μια συνδεδεμένη λίστα, επομένως η προσπέλαση στοιχείου πρέπει να ελέγχει μεταξύ των στοιχείων στην λίστα, εως ότου να βρεί αυτό με το ίδιο κλειδί. Η συνολική δομή λοιπόν ονομάζεται Πίνακας Συμβόλων (ΠΣ) και παρέχει μεθόδους προσθήκης και αναζήτησης στοιχείου.

Η συντακτική ανάλυση επομένως προσπελάζει το Συντακτικό Δέντρο αναδρομικά, και όπου συναντάει ορισμό συνάρτησης ή μεταβλητής το προσθέτει στον αντίστοιχο ΠΣ, ενώ έπειτα σε κλήση συνάρτησης ή χρήση μεταβλητής ελέγχει οτι αυτό το σύμβολο υπάρχει στον ΠΣ.

Προκύπτει ενα επιπλέον ζήτημα, αυτό του scope των συμβόλων. Φυσικά οι συναρτήσεις έχουν scope ολόκληρο το πρόγραμμα (global), αλλα οι μεταβλητές έχουν scope μόνο την διάρκεια της μεθόδου στην οποία ανήκουν (local). Επομένως δεν μπορούμε να έχουμε εναν global πίνακα συμβόλων που να περιέχει όλα τα σύμβολα, αλλιώς οι μεταβλητές με ίδιο όνομα απο διαφορετικές συναρτήσεις θα δημιουργούσαν σύγκρουση, ή θα μπορούσαμε να προσπελάσουμε μια μεταβλητή που βρίσκεται στο scope κάποιας άλλης συνάρτησης. Γι΄ αυτόν τον λόγο κάθε μέθοδος έχει το δικό της τοπικό ΠΣ για τις παραμέτρους και τις μεταβλητές της, ο οποιός αποθηκεύεται στον global ΠΣ στο στοιχείο εισόδου της συγκεκριμένης μεθόδου.

Στο τέλος της σημασιολογικής ανάλυσης, αν δεν έχει βρεθεί κάποιο σημασιολογικό σφάλμα, τότε σημαίνει οτι το πρόγραμμα εισόδου είναι ενα σωστό πρόγραμμα που μπορεί να προχωρήσει στο στάδιο παραγωγής κώδικα χωρις περαιτέρω ελέγχους. Επομένως απαιτεί μια προσοχή ωστε να μπορεί να ανιχνεύσει όλα τα σφάλματα που θα μπορούσαν να δημιουργήσουν σοβαρό πρόβλημα στην παραγωγή κώδικα και να τερματίσει.

2.3 Παραγωγή Κώδικα

Το τελικό στάδιο της μεταγλώττισης είναι η παραγωγή κώδικα με βάση το Συντακτικό Δέντρο και τους Πίνακες Συμβόλων.

Οι μοντέρνοι compilers συνήθως χρησιμοποιούν μια "ενδιάμεση αναπαράσταση" (`IR`) σαν πρώτο στάδιο της παραγωγής κώδικα. Αυτή είναι συνήθως γλώσσα μηχανής για κάποια θεωρητική μηχανή (π.χ. `JVM Bytecode`, `LLVM IR`) που συνήθως παραπέμπει σε κάποιο εξιδανικευμένο γενικό μοντέλο υπολογισμού (Stack Machine, Register Machine).

Πάνω στην `IR` ο compiler θα εφαρμόσει βελτιστοποίηση, και στη συνέχεια είτε θα την χρησιμοποιήσει

[5] Φυσικά τότε η γλώσσα υψηλού επιπέδου δεν είναι μια Γλώσσα Χωρίς Συμφραζόμενα, αφού η αποδοχή της ως σωστή εξαρτάται απο συμφραζόμενα. Η γραμματική της γλώσσας παρ΄ ολ΄ αυτά μπορεί να θεωρηθεί ως μια ΓΧΣ και για αυτό η σημασιολογική ανάλυση γίνεται σε δεύτερο στάδιο

ως έχει σαν κώδικα μηχανής καποιου Virtual Machine, ή θα την κάνει περαιτέρω μεταγλώττιση στην γλώσσα μηχανής της αρχιτεκτονικής του πραγματικού υπολογιστή στον οποίο θα τρέξει το πρόγραμμα, σε κάθε περίπτωση προσπαθώντας να υλοποιήσει την υψηλού επιπέδου λειτουργία της θεωρητικής μηχανής στο αντίστοιχο υλικό.[6]

Ο μεταγλωττιστής της εργασίας δεν χρησιμοποιεί κάποια ενδιάμεση αναπαράσταση, παράγει κατευθείαν κώδικα `MIXAL`.[7] Όμως παρόλο που δεν αναπαραστάται, η μεταγλώττιση παρόλαυτα ακολουθεί ενα θεωρητικό μοντέλο υπολογισμού, συγκεκριμένα κάτι κοντά σε Stack Machine, μόνο που οι εντολές γράφονται κατευθείαν σε `MIXAL` αντί να περάσουν πρώτα κάποιο ενδιάμεσο στάδιο.

2.3.1 Η θεωρητική μηχανή υπολογισμών

Θα ακολουθήσει μια σύντομη περιγραφή της θεωρητικής μηχανής υπολογισμού που χρησιμοποιεί ο μεταγλωττιστής για να κατασκευάσει το πρόγραμμα εισόδου.

Η μηχανή είναι αρκετά κοντά σε μια ιδανική μηχανή στοίβας (Stack Machine), με εντολές για είσοδο (`push`) ή έξοδο (`pop`) στοιχείων στη κορυφή της στοίβας, τυχαία προσπέλαση σε μια περιοχή που ονομάζεται `frame`, καθώς και μια σειρά απο πράξεις (`add`, `sub`, `neg`, ...) οπου λειτουργούν σαν τελεστές στοίβας, δηλαδή κάνουν `pop` τις εισόδους τους, και `push` το αποτέλεσμα της πράξης.

Υπάρχει φυσικά δυνατότητα ελέγχου ροής, συγκεκριμένα η μηχανή κάνει `pop` κάποια τιμή απο τη στοίβα και μπορεί να μεταπηδήσει σε αντίστοιχο σημείο του κώδικα ανάλογα (αν η τιμή που έλαβε είναι `0` ή οχι).

Μεγάλη σημασία έχει η λεγόμενη "σύμβαση κλήσης" των υπορουτίνων, απο όπου προκύπτει και η έννοια του `stack frame`.

Θα θέλαμε μια μέθοδος στην γλώσσα υψηλού επιπέδου να αντιστοιχεί σε μια υπορουτίνα στη γλώσσα μηχανής. Κάθε μέθοδος έχει τις αντίστοιχες παραμέτρους και τοπικές μεταβλητές της, καθώς πρέπει να αποθηκευέι κάπου και την θέση μνήμης απο την οποία κλήθηκε η υπορουτίνα ωστε να επιστρέψει στην κανονική ροή ελέγχου μετα την τερμάτωση της. Αν η γλώσσα υψηλού επιπέδου δεν επέτρεπε αναδρομική κλήση των συναρτήσεων, θα μπορούσαμε να έχουμε ένα σταθερό τμήμα μνήμης που θα αποθηκεύονταν οι τιμές των μεταβλητών κάθε υπορουτίνας για την διάρκεια της, και έπειτα θα αχρηστευόταν αφού θα αρχικοποιούνταν εκ νέου στην επόμενη κλήση της.

Όμως απο τη στιγμή που επιτρέπεται η αναδρομή, μια κλήση της υπορουτίνας απο τον εαυτό της, θα επικάλυπτε τις προηγούμενες τιμές των μεταβλητών με τις νέες, έτσι όταν επέστρεφε στην προηγούμενη κλίση η υπορουτίνα θα έβρισκε μεταλλαγμένες τις τιμές των μεταβλητών της. Ακόμα θα επικαλύπτονταν και η θέση μνήμης επιστροφής απο την θέση που γίνεται η κλίση εντός της υπορουτίνας, επομένως η "επιστροφή" θα έβαζε τον υπολογιστή σε εναν ατέρμονο βρόγχο, οπου θα επέστρεφε πάντα στην ίδια υπορουτίνα.

Επομένως θα θέλαμε κάθε κλήση της υπορουτίνας να έχει την δικιά της περιοχή μνήμης, ωστε οι αναδρομικές κλήσεις μιας συνάρτησης να μην επικαλύπτονται. Έτσι εισάγεται η έννοια του `stack`

[6] Η Virtual Machine είναι ενα πρόγραμμα που διαβάζει τις εντολές της `IR` σε πραγματικό-χρόνο και τις χρησιμοποιεί για την αλλαγή της εσωτερικής του κατάστασης. Σε επίπεδο γλώσσας μηχανής, οι εντολές της `IR` δεν μεταφράζονται συνήθως ποτέ σε "νέο" κώδικα μηχανής, αλλα ανατρέχουν σε υπάρχουσες ρουτίνες που υλοποιούν την υψηλού επιπέδου λειτουργία της εντολής στην αναπαράσταση που έχει επιλέξει ο σχεδιαστής της `VM`. Απο την άλλη όταν εφαρμόζεται μεταγλώττιση, οι εντολές της `IR` μεταφράζονται μία-προς-μία σε εντολές μηχανής της αρχιτεκτονικής στόχου, στο επίπεδο του συγκεκριμένου προγράμματος. Πάλι όμως θα στοχεύει στην υλοποίηση της υψηλού επιπέδου λειτουργίας της κάθε εντολής στο υλικό, παρόμοια με την `VM`. Η ειδοποιός διαφορά είναι πως σε αντίθεση με την `VM` παράγεται νέος κώδικας μηχανής ο οποίος υλοποιεί μόνο τις συγκεκριμένες εντολές του προγράμματος. Θα μπορούσε να θεωρηθεί σαν μια `VM` μόνο του συγκεκριμένου προγράμματος, που ακολουθεί τη συγκεκριμένη πορεία υπολογισμού αυτού.

[7] Βέβαια και ο ίδιος ο `MIX` είναι μια "εξιδανικευμένη" μηχανή που τρέχει πάνω σε `VM`. Όμως δέν είναι τόσο εξιδανικευμένος ωστε να μπορούμε να χρησιμοποιήσουμε την `MIXAL` ως "υψηλού επιπέδου" γλώσσα μηχανής.

`frame` ωστε να επιλύσει αυτό το πρόβλημα. Πρακτικά η κάθε κλήση της συνάρτησης δημιουργεί στην στοίβα μια δικιά της περιοχή μνήμης, καθώς και εναν δείκτη σε ενα σταθερό σημείο του `frame` (`frame pointer`, `FP`), γύρω απο το οποίο αποθηκεύονται οι τιμές των παραμέτρων, των τοπικών μεταβλητών, της θέσης επιστροφής (`return address`, `RA`) καθώς και του προηγούμενου `FP` ωστε μετά την επιστροφή απο την υπορουτίνα να μπορεί να επιστρέψει την κατάσταση της μηχανής σε αυτή ακριβώς που είχε πρίν την κλήση.

Συγκεκριμένα, όταν κάποιο τμήμα του κώδικα θέλει να κάνει κλήση σε κάποια συνάρτηση, προσθέτει στη στοίβα της παραμέτρους εισόδου, απο την τελευταία προς την πρώτη,[8] και μεταπηδάει στην θέση μνήμης που ξεκινάει η υπορουτίνα, αποθηκεύοντας την επόμενη θέση μνήμης σε εναν ειδικό `register`. Στη συνέχεια η υπορουτίνα είναι υπεύθυνη ωστε να αποθηκεύσει στη στοίβα την `RA`, την προηγούμενη τιμή του `FP`, να θέσει τον `FP` στην νέα τιμή του, και να αυξήσει τον `stack pointer` (`SP`) ωστε να δεσμεύσει χώρο για τις τοπικές μεταβλητές της υπορουτίνας. Κάθε παράμετρος ή μεταβλητή της υπορουτίνας βρίσκεται σε σχέση με τον `FP`, ως μια σταθερή μετατόπιση (`offset`) ως προς αυτόν, θετική ή αρνητική ανάλογα με το είδος της μεταβλητής (τοπική ή παράμετρος αντίστοιχα). Στο τέλος της υπορουτίνας, αυτή είναι υπεύθυνη για την αποδέσμευση της μνημης ολόκληρου του `frame` (ακόμα και των παραμέτρων που έκανε `push` η συνάρτηση κλήσης), επιστρέφει την `FP` στην προηγούμενη τιμή της, κάνει `push` την τιμή επιστροφής της συνάρτησης (`return value`) και επιστρέφει την ροή του προγράμματος στην `RA`.

Με αυτόν τον τρόπο, η κάθε συνάρτηση λειτουργεί απο μόνη της σαν ένας τελεστής στοίβας. Απο την οπτική της συνάρτησης κλήσης (`caller`), κάνει `push` τις παραμέτρους εισόδου, καλεί την συνάρτηση/τελεστή, και βρίσκει στην στοίβα αντί των παραμέτρων την τιμή εξόδου της συνάρτησης.

Για παράδειγμα έστω η συνάρτηση `int foo(int a, int b)` που χρησιμοποιεί τις τοπικές μεταβλητές `int x, y, z`. Το `stack frame` κάποιας κλήσης της συνάρτησης θα μοιάζει ώς εξής

	...	
`offset -3`	parameter `b`	
`offset -2`	parameter `a`	
	return address (`RA`)	
	previous `FP`	⇐ `FP`
`offset +1`	variable `x`	
`offset +2`	variable `y`	
`offset +3`	variable `z`	⇐ `SP`
	...	

Και έστω η κλήση της `foo(5,8)` τότε η σειρά των βημάτων απο πλευράς του `caller` θα πρέπει να είναι:

1	`push 8`
2	`push 5`
3	`call foo`
4	`pop foo(5,8)`

2.3.2 Αναπαράσταση θεωρητικής μηχανής στον `MIX`

Η θεωρητική μηχανή που αναλύθηκε παραπάνω, για να έχει ουσία σαν υπολογιστική μηχανή, πρέπει να έχει αναπαράσταση στην πραγματική μηχανή που εκτελεί τις πράξεις, δηλαδή στον `MIX`. Αυτό δεν ειναι ενα ιδιαίτερα σύνθετο ζήτημα δεδομένου οτι η αρχιτεκτονική του `MIX` δεν διαφέρει πάρα πολύ απο αυτή της θεωρητικής μηχανή. Σε εναν βαθμό δηλαδή, έχει ήδη τις λειτουργίες που θέλουμε να υλοποιήσουμε, μόνο που δεν λαμβάνουν την είσοδο τους απο κάποια στοίβα, αλλα απο συγκεκριμένες

[8] Εξαιτίας του τρόπου που υπολογίζονται τα `offsets` στην σημασιολογική ανάλυση

θέσεις μνήμης. Αρκεί λοιπόν να σχεδιαστεί μια αναπαράσταση της στοίβας στη μνήμη, ωστε να είναι δυνατό να χρησιμοποιήσουμε τις εντολές αυτές με τον τρόπο που αναλύθηκε παραπάνω.

Η Στοίβα φυσικά δεν είναι τίποτα παρα μια περιοχή μνήμης, με περιορισμό στην είσοδο και έξοδο δεδομένων. Συγκεκριμένα ο λεγόμενος δείκτης στοίβας (SP) δείχνει πάντα στην κεφαλή της μνήμης, και η εγγραφή ή διαγραφή των δεδομένων γίνεται πάντα μέσω αυτού. Ο SP δεν είναι παρά ένας register του MIX (αυθαίρετα επιλέχθηκε ο rI6) οπου χρησιμοποιείται σε συνδιασμό με την θέση αρχης της στοίβας για την προσπέλαση της θέσης μνήμης που αντιστοιχεί στην κορυφή της στοίβας. Έστω λοιπόν οτι η αρχική θέση μνήμης της στοίβας είναι STACK. Τότε αν στο πεδίο διεύθυνσης κάποιας εντολής χρησιμοποιήσω την μορφή STACK,6 αυτό αντιστοιχεί στη θέση μνήμης STACK[SP].

Συγκεκριμένα, για να εισάγω ενα στοιχείο στη στοίβα (push), πρώτα αυξάνω τον SP κατα 1, και μετά γράφω στην θέση μνήμης που δείχνει πλέον αυτός. Για να εξάγω στοιχείο (pop) διαβάζω απο την θέση μνήμης που δείχνει ο SP και μετά τον μειώνω κατα 1.

Ο frame pointer (FP) είναι επίσης ένας register του MIX (πάλι αυθαίρετα επιλέχθηκε ο rI5) ο οποίος χρησιμοποιείται με τον ίδιο τρόπο που χρησιμοποιείται ο SP, μόνο που δεν αλλάζει παρα μόνο στην είσοδο ή έξοδο απο υπορουτίνα, και χρησιμοποιείται πάντα σε συνδιασμό με κάποιο offset που αντιστοιχεί στην τοπική μεταβλητή που θέλουμε να προσπελάσουμε. Για παράδειγμα αν στο offset -2 βρίσκεται η παράμετρος a, τότε η θέση μνήμης της είναι STACK[FP-2] και αυτό αντιστοιχεί στο πεδίο διεύθυνσης STACK-2,5.

Με αυτόν τον τρόπο γίνεται εύκολη η προσπέλαση των χρήσιμων για εμάς θέσεων μνήμης, και απαιτεί μονάχα προσοχή ωστε σε κάθε εντολή να τηρούνται οι συμβάσεις που έχουμε επιλέξει.

Όσον αφορά τους τελεστές στοίβας, θα εξετάσουμε ενα παράδειγμα, αυτό του τελεστή πρόσθεσης (add) και πώς αυτός μπορεί να υλοποιηθεί σε γλώσσα MIXAL. Αυτός ο τελεστής της θεωτηρικής μηχανής εξάγει 2 τιμές απο τη στοίβα, τις προσθέτει μεταξύ τους, και εισάγει το αποτέλεσμα στη στοίβα. Στον MIX με τις συμβάσεις που έχουμε κάνει αυτό αντιστοιχεί στις παρακάτω εντολές

Line #	MIXAL instr.		Comment
1	LDA	STACK,6	rA ← STACK[SP]
2	DEC6	1	SP ← SP − 1
3	ADD	STACK,6	rA ← rA + STACK[SP]
4	DEC6	1	SP ← SP − 1
5	INC6	1	SP ← SP + 1
6	STA	STACK,6	STACK[SP] ← rA

Η παραπάνω σειρά εντολών είναι χωρισμένη σε 3 λογικά τμήματα. Οι γραμμές 1-2 υλοποιούν την πρώτη ενέργεια pop. Διαβάζει απο την κορυφή της στοίβας στον register rA και μειώνει τον SP. Αντίστοιχα οι γραμμές 3-4 κάνουν pop απο τη στοίβα, μόνο που αντί για να αποθηκεύεται το αποτέλεσμα στον rA, προστίθεται σε αυτόν με την εντολή MIXAL ADD. Τέλος οι γραμμές 5-6 κάνουν push το αποτέλεσμα που βρίσκεται στον rA στη στοίβα με αντιστοιχο τρόπο.[9] Έτσι αυτή η

[9]Εδώ η σειρά 4 μειώνει τον SP και η σειρα 5 άμεσα τον ξανα-αυξάνει. Φυσικά αυτή η αλληλουχία αντίστροφων εντολών είναι ανούσια και για αυτό στον πραγματικό κώδικα παραλλείπεται. Εδώ το δείχνω διότι αυτές οι δυο εντολές είναι μέρος του λογικού συνόλου pop και push των αντίστοιχων εντολών. Στο επίπεδο υλοποίησης κάποιας εντολής υψηλού επιπέδου σε MIXAL μπορούμε να κάνουμε τέτοιες βελτιώσεις, όμως δεν μπορούμε όταν κάτι τέτοιο προκύπτει μεταξύ εντολών. Για παράδειγμα, δύο εντολές add η μία μετά την άλλη θα οδηγήσουν στην εγγραφή του rA στην κορυφή της στοίβας, και έπειτα στην ανάγνωση του rA απο την κορυφή της στοίβας. Αυτές είναι πάλι δυο αντίστροφες εντολές που δεν έχουν ουσία για το πρόγραμμα, όμως δεν μπορούμε να τις αφαιρέσουμε απο τον κώδικα μηχανής της εντολής, δίοτι δεν μπορούμε φυσικά να υποθέσουμε πως κάθε εντολή add θα ακολουθείται απο κάποια άλλη εντολή add, και έτσι αναγκαστικά το τελικό πρόγραμμα MIXAL θα σπαταλάει κάποιους κύκλους μηχανής για τις ανακρίβειες της μεταγλώττισης. Τέτοιου είδους ανακρίβειες θα μπορούσαν να μειωθούν με ένα στάδιο βελτιστοποίησης του μεταγλωττισμένου κώδικα MIXAL (ή της IR αν υπήρχε), όμως αυτό δεν υλοποιείται στα πλαίσια της εργασίας.

αλληλουχία εντολών MIXAL αντιπροσωπεύει την εντολή add της θεωρητικής μηχανής.

Οι υπόλοιποι τελεστές στοίβας λειτουργούν με αντίστοιχο τρόπο. Για την συγκεκριμένη υλοποίηση τους ανατρέξτε στον πηγαίο κώδικα του μεταγλωττιστή.

2.3.3 Μεταγώττιση του Συντακτικού Δέντρου σε κώδικα MIXAL

Απομένει λοιπόν η ανάγνωση του Συντακτικού Δέντρου και η παραγωγή του κώδικα MIXAL που υλοποιεί το πρόγραμμα υψηλού επιπέδου. Η ανάγνωση του AST γίνεται αναδρομικά, και ανάλογα με το είδος κόμβου που συναντηθεί παράγεται ο αντίστοιχος κώδικας πρίν ή μετά την ανάγνωση τον βαθύτερων κόμβων.

Για παράδειγμα όταν συναντηθεί κάποιος τερματικός κόμβος NUMBER ή LOCATION η τιμή του εισέρχεται στη στοίβα. Όταν συναντάται πράξη δύο τελεστέων, πρώτα γίνεται η ανάγνωση των κόμβων παιδιών, και έπειτα εκτελείται η αντίστοιχη πράξη.[10] Αντίστοιχα για πράξεις ενός τελεστέου. Έτσι ήδη, αφού το ΣΔ σέβεται λόγω της γραμματικής την προτεραιότητα των πράξεων, μπορούμε να υπολογίσουμε το αποτέλεσμα όποιασδήποτε αλγεβρικής παράστασης.

Όταν συναντάται κλήση συνάρτησης CALL, πρώτα γίνεται η ανάγνωση των κόμβων όλων των παραμέτρων απο τον τελευταίο προς τον πρώτο, και έπειτα γίνεται μεταπήδηση στην υπορουτίνα. Να σημειωθεί πως εφόσον το αποτέλεσμα κάθε αλγεβρικής πράξης είναι η τοποθέτηση του αποτελέσματος στην κορυφή της στοίβας, μπορούν να χρησιμοποιηθούν αλγεβρικές πράξεις ως παράμετροι συναρτήσεων και κλήσεις συναρτήσεων ως μέρος αλγεβρικών πράξεων.

Ο κόμβος ASSIGN πρώτα κάνει ανάγνωση της παράστασης δεξιά του =, εξάγει το αποτέλεσμα απο τη κορυφή της στοίβας και το αποθηκεύει στην θέση μνήμης που αντιστοιχεί στην μεταβλητή αριστερά του =.

Οταν συναντάται ορισμός μεθόδου, προστίθεται το τμήμα κώδικα που αποθηκέυει το RA και FP και δεσμεύει χώρο μνήμης για τις τοπικές μεταβλητές. Έπειτα γίνεται η ανάνγωση κάθε εντολής υψηλού επιπέδου στο "body" της μεθόδου. Στο τέλος προστίθεται το τμήμα της αποδέσμευσης της μνήμης της μεθόδου και η επιστροφή του SP στη προηγούμενη τιμή του. Κάποιος κόμβος RETURN στο εσωτερικό της μεθόδου, εισάγει το αποτέλεσμα της παράστασης που ακολουθεί την κωδική λέξη στη κορυφή της στοίβας, και μεταπηδά στο τελικό στάδιο της συνάρτησης που περιγράφτηκε νωρίτερα.

Με παρόμοιο τρόπο γίνεται η ανάγνωση όλων των κόμβων και η μετατροπή τους σε αντίστοιχο τμήμα κώδικα συμβολομετάφρασης MIXAL.

Τέλος, στην αρχή και στο τέλος του προγράμματος προστίθεται ενα τμήμα που ορίζει τις θέσεις μνήμης, τις σταθερές, τις αρχικές τιμές των πινάκων και την αρχή εκτέλεσης του προγράμματος. Απο επιλογή του συγγραφέα, η τιμή που επιστρέφει η συνάρτηση main (η οποία βρίσκεται στην κορυφή της στοίβας όταν επιστρέψουμε στο τμήμα κώδικα MIXAL που την κάλεσε) τυπώνεται με ένα μύνημα ωστε να μπορούμε να ελέγχουμε τα αποτελέσματα των προγραμμάτων χωρίς να χρειάζεται να κοιτάμε την μνήμη του VM.

Για μια αναλυτικότερη περιγραφή της αντιμετώπισης του κάθε κόμβου ανατρέξτε τον πηγαίο κώδικα του μεταγλωττιστή.

[10] Εδώ για ορισμένες πράξεις έχει σημασία η σειρά των τελεστέων. Για παράδειγμα $a - b \neq b - a$, οπότε έχει σημασία η σειρά ανάγνωσης των δεξιών και αριστερών κόμβων. Εφόσον ο πρώτος τελεστέος γίνεται πρώτος pop απο τη στοίβα, πρέπει πρίν φτάσουμε στην πράξη sub (ή οποια άλλη) να βρίσκεται στην κορυφή αυτής. Επομένως πρέπει πρώτα να γίνεται η ανάγνωση του δεξιού κόμβου, και έπειτα του αριστερού, έτσι ωστε ο αριστερός τελεστέος να βρίσκεται "απο πάνω" στην στοίβα

2.4 Αποτίμιση / Περιορισμοί

Με όλα τα παραπάνω, ο μεταγλωττιστής πλέον είναι πλήρης και λειτουργικός. Παρ΄ ολ΄ αυτά υπάρχουν πολλά σημεία στη σχεδίαση του που θα μπορούσαν να βελτιωθούν.

Πρωτ΄ απ΄ όλα η ίδια η γλώσσα υψηλού επιπέδου, λόγω της περιορισμένης έκτασης της αφήνει αρκετά κενά στην πληρότητα της γλώσσας σαν υπολογιστική μηχανή.[11] Αυτό βέβαια δεν είναι έλλειμμα του μεταγλωττιστή, αλλα της ίδιας της δομής της γλώσσας υψηλού επιπέδου.

Ο μεταγλωττιστής απο πλευράς του έχει και αυτός αρκετά προβλήματα. Το μάλλον κρισιμότερο είναι η διαρύθμιση της μνήμης. Στον χρόνο της εγγραφής αυτής της αναφοράς, ο μεταγλωττιστής διανέμει την μνήμη ως εξής. Οι θέσεις μνήμης 0-2999 είναι η περιοχή της στοίβας, 3000-3899 είναι δεσμευμένο για τον κώδικα μηχανής του μεταγλωττισμένου προγράμματος, 3900-3999 αρχικές τιμές πινάκων και σταθερών. Αυτή είναι μια κατανομή η οποία λειτουργεί στα πλαίσια του φυσιολογικού και δεν έχει δημιουργήσει προβλήματα εως τώρα. Όμως αν είχα κάποιο πρόγραμμα με πολύ μεγάλο αριθμό μεταφρασμένων εντολών, ή ενα πρόγραμμα με πολύ βαθιά αναδρομή, δεν είναι απίθανο οι αντίστοιχες περιοχές να μη φτάνουν για την εξυπηρέτηση των αναγκών του προγράματος. Μια δυναμική κατανομή μνήμης, που λαμβάνει υπόψη το μέγεθος του προγράμματος και κατανέμει την μνήμη ανάλογα θα ήταν καλύτερη λύση, όμως αυτό δεν έχει υλοποιηθεί.

Επιπλέον δεν υπάρχει κανένας έλεγχος για υπερχείλιση ή υποχείλιση στοίβας. Αυτό σημαίνει πως αν κάποιο πρόγραμμα χρειαστεί περισσότερο χώρο στοίβας απ΄ ότι έχει διατεθεί, αντί για να τερματίσει με εκτύπωση κάποιου μυνήματος σφάλματος, θα αρχίσει να μεταλλάσσει εσφαλμένα τις εντολές του προγράμματος μετά την θέση 3000. Αυτό είναι μεγάλο πρόβλημα, και θα λυνόταν με τον κατάλληλο έλεγχο του μεγέθους της στοίβας κάθε φορά που αυξάνω ή μειώνω των `SP`.

Τέλος, ο πηγαίος κώδικας του μεταγλωττιστή επιφέρει σίγουρα βελτιώσεις. Σέ ενα βαθμό ο κώδικας γράφτηκε με προσοχή, ωστε η λειτουργία του να είναι σκιαγραφημένη πρίν την υλοποίηση του, ωστε να μην δημιουργηθούν σενάρια ασυμβατότητας του μελλοντικού σταδίου με κάποιο προηγούμενο. Παρ΄ ολ΄ αυτά τέτοιες περιπτώσεις υπήρξαν, και το αποτέλεσμα είναι πως ορισμένα τμήματα του κώδικα έχουν διαφορετικές συμβάσεις. Απο την άλλη ο έλεγχος λαθών είναι περιορισμένος σε ορισμένα τμήματα του κώδικα και ελλειπές.

[11] Με μία γρήγορη, και καθόλου αυστηρή εκτίμιση, μπορούμε να πούμε οτι η γλώσσα μπορεί να υπολογίσει την κλάση συναρτήσεων των πρωτόγονων αναδρομικών συναρτήσεων. Όσον αφορά τις γενικές αναδρομικές συναρτήσεις, θέλει περισσότερη ανάλυση. Πάντως ανεξάρτητα εξαιτίας της ικανότητας τυχαίας προσπέλασης περιοχών μνήμης, είναι ανίκανη να προσομοιώσει κάποια Μηχανή Turing. Σε θεωρητικό επίπεδο, αν δεν υπήρχαν περιορισμοί στο μέγεθος των μεταβλητών θα μπορούσε κανείς εύκολα να κατασκευάσει μια απλή Μηχανή Minsky με 2 Counters, η οποία είναι γνωστό πως μπορεί να προσομοιώσει μηχανή Turing. Αυτό το γεγονός όμως βασίζεται στην ικανότητα των counters να αυξηθούν απεριόριστα. Απο την στιγμή που είναι περιορισμένοι στον τύπο `int`, που αντιστοιχεί σε μια λέξη `MIX` (περίπου 30 `bits`), η ταινία της Μηχανής Turing θα είναι περιορισμένη σε 30 θέσεις εγγραφής με αλφάβητο $\{0, 1\}$. Μακριά απο την θεωρητικά άπειρη ταινία.