COMPILERS

2Η ΦΑΣΗ 2017

Βασίλειος Δρέττας – 1115201300042

Δημήτριος Κωνσταντάκης – 1115201300079

Η εργασία έχει υλοποιηθεί σε Java.

Abstract Syntax Tree

‘Εχει υλοποιηθεί η μετατροπή του Concrete Syntax Tree σε Abstract Syntax Tree .

Για την δημιουργία του Abstract Syntax Tree έχουμε διατηρήσει σχεδόν την ίδια γραμματική της πρώτης φάσης με ελάχιστες αλλαγές. Τα στοιχεία που έχουμε προσθέσει είναι μία ακόμη παραγωγή **expr_list** (λίστα από expressions), καθώς και το token **erroneous_number** για τον εντοπισμό λάθους σε πράξεις (mod, div) όταν ο αριθμός δεν χωρίζεται με κενό με το token της πράξης.Τέλος, έχουμε επεκτείνει τις πράξεις pos και neg (εφαρμογή προσήμου), οι οποίες τώρα εφραμόζονται σε παραγωγή expr_par και όχι term (προκειμένου να δέχεται την πράξη ---3).

Το traverse εφαρμόζεται στις ρουτίνες in και out του απλοποιημένου δέντρου AST. Γενικά για το AST, οι παραγωγές expressions παράγονται κατευθείαν από τις παραγωγές expr, expr_par, factor, term του CST και δημιουργούνται οι κόμβοι της αντίστοιχης πράξης. Έχουν προστεθεί ακόμη στην παραγωγή expression, οι παραγωγές function call και lvalue, προκειμένου να υπάρχει μια γενικότερη απλοποιημένη μορφή στο δέντρο. Το ίδιο συμβάνει και με τα conditions και τις αντίστοιχες παραγωγές τους. Από τις παραγωγές έχουμε αφαιρέσει τα tokens που δεν χρησιμοποιούνται (σύμβολα πράξεων, brackets, τελεστές,…) .

Semantic Analysis

Ο επιθυμητός στόχος είναι ο καλύτερος έλεγχος του προγράμματος .Η σημασιολογική ανάλυση πρέπει να παρέχει στο οπίσθιο μέρος του μεταγλωττιστή ότι πληροφορία χρειάζεται για να παράγει τον τελικό κώδικα. Για την υλοποίηση του Πίνακα Συμβόλων (SymbolTable) δημιουργούμε μία λίστα από πίνακες της δομής Hashtable. Κάθε κόμβος της λίστας αναφέρεται σε μία διαφορετική εμβέλεια. Έχουν υλοποιηθεί οι ρουτίνες :
enter(): δημιουργία καινούργιου πίνακα. Καλείται στην caseAFunDefinition, μετά την κλήση της in.
insert(): προσθήκη στον τρέχον πίνακα. Καλείται κάθε φορά που θέλουμε να προσθέσουμε ένα αντικείμενο τύπου Record στον πίνακα.
lookup(): αναζήτηση σε κάθε πίνακα και επιστρέφει την πρώτη εμφάνιση του αντικειμένου.
exit(): καταστροφή του τρέχοντος πίνακα. Καλείται επίσης στην caseAFunDefinition, μετά την κλήση της out και την δημιουργία του ενδιάμεσου κώδικα.

Record Classes:

Τα αντικείμενα που δημιουργούμε και προσθέτουμε στους πίνακες είναι τύπου Record. Ως Record έχουμε ορίσει μία γενική κλάση η οποία περιλαμβάνει ένα όνομα και ένα τύπο. Για κάθε αντικείμενο που θέλουμε να προσθέσουμε καποιές ακόμη ιδιότητες χρησιμοποιούμε subclasses .Για παράδειγμα αν το αντικείμενο αυτό είναι μία συνάρτηση τότε δημιουργούμε μία δομή RecordFunction και την αρχικοποιούμε με τα επιπρόσθετα χαρακτηριστικά όπως την λίστα με τις παραμέτρους. Με την ίδια λογική έχουμε υλοποιήσει τις δομές RecordArray, RecordParam, RecordParamArray. Με casting μπορούμε το αποτέλεσμα του lookup που είναι τύπου Record να το μετατρέψουμε στον τύπο της υποκλάσης που είναι instance of και να έχουμε πρόσβαση σε όλα τα δεδομένα της.

Error Class:

Για κάθε τύπο λάθους έχουμε υλοποιήσει μία αντίστοιχη συνάρτηση στην κλάση Error η οποία εκτυπώνει το λάθος που εντοπίστηκε κατά την σημασιολογική ανάλυση καθως και τη γραμμή που βρίσκεται στον κώδικα.

RecType Class:

Είναι βοηθητική κλάση για τον σημασιολογικό έλεγχο. Για κάθε μεταβλητή ή σταθερά που χρησιμοποιείται στον κώδικα φτιάχνουμε ένα αντικείμενο τύπου RecType και το τοποθετούμε στη στοίβα typeStack. Αυτό περιλαμβάνει το όνομα, τον τύπο και τις διαστάσεις του. Όταν έχουμε μία αριθμητική ή λογική πράξη , assignment,… και χρειάζεται να ελέγξουμε αν οι τύποι των μεταβλητών είναι οι σωστοί και οι αναμενόμενοι ,τότε ανάλογα με την περίπτωση κάνουμε pop όσα αντικείμενα χρειάζονται από την στοίβα και συγκρίνουμε τους τύπους τους. Αφού

γίνει η σύγκριση εμείς επιλέξαμε να βάζουμε στη στοίβα ανεξαρτήτως αποτελέσματος τον αναμενόμενο παραγόμενο τύπο αφενός για να μην εμφανιστούν αλυσιδωτά errors, αφετέρου πράξεις μεταξύ διαφορετικών τύπων (πχ. int + char) δεν υφίστανται στην γλώσσα.

Με τον ίδιο τρόπο έχουμε υλοποιήσει και μία στοίβα με τους τύπους επιστροφής των συναρτήσεων για να τσεκάρουμε αν κάθε φορά που χρησιμοποιείται τον statement return μέσα στην συνάρτηση επιστρέφει αντικείμενο ίδιου τύπου με τον τύπο ορισμού της συνάρτησης.

Τις συναρτήσεις της standard library της Γλώσσας Grace τις εισάγωγουμε χειροκίνητα στο SymbolTable κατά την δημιουργία του.

Intermidiate Code:

Για την παραγωγή του ενδιάμεσου Κώδικα, θα πρέπει να μην έχει εντοπιστεί κάποιο Error στον σημασιολογικό έλεγχο. Μόλις ολοκληρωθεί ο σημασιολογικός έλεγχος και δεν υπάρχει error τότε καλούμε την caseAFunDefinition του VisitorIR για τον κόμβο του δέντρου στον οποίο μόλις καλέσαμε την outAFunDefintion της κλάσης SymbolTable. Ο ενδιάμεσος κώδικας παράγεται δηλαδή πριν βγούμε από το caseAFunDefintion που βρισκόμαστε. Πρακτικά αυτό που γίνεται είναι να ξανακάνουμε traverse στον υπόδεντρο που ήμασταν , αυτή τη φορά όμως γνωρίζουμε ότι δεν υπάρχει σημασιολογικό λάθος και προχωράμε στην υλοποίηση των quads.

Quad Class:

Αποτελείται από ένα label, τον operator και τα opt1, opt2 και opt3.

QuadList Class:

Περιλαμβάνει τη λίστα που αποθηκεύονται τα quads και τη λίστα στην οποία αποθηκεύουμε τις προσωρινές μεταβλητές που χρησιμοποιούμε (TmpVar). Οι προσωρινές μεταβλητές έχουν όνομα και τύπο. Οι βασικότερες ρουτίνες που έχουμε υλοποιήσει είναι οι NextQuad() που επιστρέφει τον αύξοντα αριθμό για το νέο quad και GenQuad() που δημιουργεί ένα νέο quad και το βάζει στην λίστα (quadlist).

Condition Class:

Περιλαμβάνει τις δύο λίστες true και false. Σε αυτές εισάγουμε τα labels των quads. Σε μια λογική συνθήκη δημιουργούμε ένα προσωρινό αντικείμενο extrCond τύπου Contition και στη συνέχεια ανάλογα τη συνθήκη εφαρμόζουμε την κατάλληλη ρουτίνα mergeLists ή swapLists . Αφού ολοκληρωθεί η συνθήκη, καλείται η συνάρτηση backPatch η οποία τοποθετεί στα “*” των quads της αντίστοιχης λίστα (true,false) στο πεδίο opt3 το κατάλληλο label.

RecordLValue/ StringLiteral:

Χρησιμοποιείται με τον ίδιο τρόπο όπως και η κλάση RecType, σε συνδυασμό με τη στοίβα stackLValue.Την δομή StringLiteral την χρησιμοποιούμε καθώς δεν ορίζεται εγγραφή για τα string literal. Τα string literal αρχικά τα κάνουμε ανάθεση σε tmpVar. Την δομή RecordLValue την χρησιμοποιούμε όταν αναγνωρίζεται ένα lvalue.

Η εργασία βρίσκεται στο παρακάτω link :

https://github.com/billDrett/GraceCompiler