Compilers 2017

Υλοποίηση μεταγλωττιστή για τη γλώσσα Grace

Συγγραφείς:
Κωστοπούλου Καλλιόπη sdi1200084
Τσόλκας Ανδρέας sdi1400212

- **Λεκτική Ανάλυση**
- **Συντακτική Ανάλυση**
- **Σημασιολογική Ανάλυση**
- **Ενδιάμεσος Κώδικας**
- **Τελικός Κώδικας (Assembly x86)**

Γενικά

Έχουν επανεξεταστεί όλες οι παρατηρήσεις που μας έγιναν στο πρώτο και δεύτερο μέρος του project και έχουν διορθωθεί όλα τα αντίστοιχα λάθη.

- Για την άμεση παραγωγή των έτοιμων αρχείων του sableCC (framework) καθώς και για τη μεταγλώττιση των .java έχει υλοποιηθεί ένα script.
  > ./myscript.sh
- Έχει υλοποιηθεί Makefile.
- Για την εκτέλεση του compiler έχει δημιουργηθεί ένα άλλο script. Συγκεκριμένα, για την μεταγλώττιση ενός προγράμματος grace:
  >./cgrace -o <output_file> <filename>

  ή

  >./cgrace <filename> -o <output_file>

- *Έχει υλοποιηθεί ένα mini_optimization για την αφαίρεση των άσκοπων jump μετά από κάθε condition.*

  Μετατρέπω το condition στην άρνηση του και σαν label στην τετράδα του βάζω το destination της jump από κάτω.

  Έτσι έχω τη δυνατότητα να αφαιρέσω την τετράδα της jump.
  Στο σώμα της mini_optimize(..) έχει δημιουργηθεί ο σκελετός για την υλοποίηση όλων των υπόλοιπων optimization που θα μπορούσαν να υλοποιηθούν.

  Επειδή η mini_optimize(..) παράχθηκε πρόχειρα, σε σύντομο χρονικό διάστημα, δίνουμε τη δυνατότητα στον χρήστη να μην τη χρησιμοποιήσει κατα την μεταγλώττιση του προγράμματός του, για την αποφυγή εμφάνισης κάποιου bug.

  Αυτό γίνεται με τη χρήση ενός επιπρόσθετου flag (-f) στην παραπάνω εντολή μεταγλώττισης.

  Με άλλα λόγια για να συμπεριληφθεί το optimization η εντολή μεταγλώττισης είναι:

  >./cgrace -o <output_file> <filename> -f

  ή

  >./cgrace <filename> -o <output_file> -f

- ***Για να εκτελέσετε τα bash scripts θα χρειαστεί να πειράξετε τα permissions.***

- Έχουμε συμπεριλάβει στο παραδοτέο το sableCC.jar για την επιτυχή εκτέλεση του script και συνεπώς για τη διευκόλυνση σας.

- Επειδή διαλέξαμε την υλοποίηση ενός symbol table (hashtable & stack) του οποίου οι “εμβέλειες’ καταστρέφονται την ώρα που βγαίνουμε, κατα το traversal, από αυτές, χωρίς να κρατώνται οι πληροφορίες των εμβελειών αυτών σε κάποια άλλη δομή, η χρήση παραπάνω visitor από έναν δεν ήταν εφικτή.

  Χρησιμοποιείται ένας και μοναδικός visitor για όλες τις φάσεις της μεταγλώττισης.

  Αυτό κανει τον κώδικά μας λιγάκι πιο δυσανάγνωστο, ωστόσο, έχει άλλα πλεονεκτήματα, όπως το ότι η υλοποίηση αυτή είναι πιο γρήγορη/αποδοτική.

  Η αδυναμία επιστροφής πληροφορίας από έναν κόμβο στο γονιό του μέσω του return value κάθε μεθόδου, εξαιτίας της δημιουργίας void μεθόδων από το sableCC, καθιστά απαραίτητη τη χρήση επιπρόσθετων members στον visitor, κάτι που επίσης καθιστά τον κώδικά μας λιγότερο ευανάγνωστο.

- Μεταξύ των δύο τρόπων υλοποίησης του μηχανισμού επικοινωνίας των παιδιών κόμβων με τους γονείς τους που συζητήθηκαν στο piazza, επιλέξαμε αυτόν κατά τον οποίο κάθε κόμβος αφήνει πληροφορία για τον γονέα του σε members του visitor.

- Για την επικοινωνία κόμβων - παιδιών με τους γονείς τους για την κατασκευή του symbol table χρησιμοποιήθηκαν τα members:

```
Boolean reference;
String datatype;
Key name;
Key funname;
LinkedList <Param> params;
LinkedList <Integer> arraylist;
LinkedList <Key> idlist;
String retvalue;
```

- Για την επικοινωνία κόμβων - παιδιών με τους γονείς τους για το type checking καθώς και για τους άλλους ελέγχους της σημασιολογικής ανάλυσης χρησιμοποιείται μια δομή stack.

```
LinkedList <TypeCheck> typeCheck;
```

  Κάθε κόμβος παίρνει την πληροφορία που χρειάζεται από τα παιδιά του κάνοντας pop ένα ή περισσότερα elements από τη στοίβα και στο τέλος κάνει push τη δική του πληροφορία σε αυτήν, έτσι ώστε να την εκμεταλλευτεί αργότερα ο δικός του γονέας.

  Τα elements της λίστας περιέχουν τόσο τον τύπο της έκφρασης/τιμής που “επιστρέφεται” όσο και άλλες διάφορες χρήσιμες πληροφορίες για τη σημασιολογική ανάλυση.

Γίνεται χρήση άλλης μιας δομής stack:

```
LinkedList <Node> funDefinition;
```

στην οποία αποθηκεύονται μόνο οι συναρτήσεις που έχουν ορισθεί με τη σειρά που ορίζονται. Αυτές αποθηκεύονται και στο symbol table. Η χρήσιμότητα της funcDefinition είναι για την ανίχνευση σημασιολογικών λαθών που αφορούν στην τιμή επιστροφής μιας συνάρτησης.
Κάθε φορά που φτάνουμε σε κάποιο κόμβο “return”, δηλαδή κάθε φορά που γίνεται return; / return expr; στο σώμα μιας συνάρτησης, λαμβάνεται (pop χωρίς remove) το τελευταίο element της στοιβας που αντιστοιχεί στη συνάρτηση, στης οποίας το σώμα βρισκόμαστε τώρα και γίνονται οι απαραίτητοι έλεγχοι/συγκρίσεις.
Ένα element αυτής της στοίβας γίνεται pop (με remove) κατά την έξοδο από τον ορισμό της συνάρτησης, δηλαδή στη

```
void outAFuncdefLocalDef(AFuncdefLocalDef node)
```

- Έχει δημιουργηθεί μια δομή (ScopeTemp) για την κράτηση των προσωρινών μεταβλητών της συνάρτησης στο σώμα της οποίας βρισκόμαστε κάθε φορά, για την διευκόλυνση της ανάπτυξης του τελικού κώδικα.

Τμήματα Project – Πορεία Υλοποίησης

➔ Λεκτική Ανάλυση
- Regular Expressions

➔ Συντακτική Ανάλυση
- Γραμματική (Productions section)
- Αφαίρεση ambiguities

➔ Μετατροπή CST -> AST

➔ Υλοποίηση Symbol Table

➔ Σημασιολογική Ανάλυση
- Type Checking
- Έλεγχοι Μοναδικότητας
. . . . . .

➔ Ενδιάμεσος Κώδικας
- Παραγωγή τετράδων
- Αποθήκευσή τους στη μνήμη (σε λίστα)

➔ Τελικός Κώδικας
- Παραγωγή αρχείου myAssembly.txt
- Υλοποίηση βοηθητικών συναρτήσεων
- Εκτέλεση προγραμμάτων σε grace

Λεκτική Ανάλυση

Έχουν ληφθεί υπόψη μας όλες οι αναφερθείσες παρατηρήσεις από την εξέταση του πρώτου μέρους της άσκησης – λεκτική/συντακτική ανάλυση και έχουν διορθωθεί !!

Πιο συγκεκριμένα, πλέον:

- Δεν γίνονται δεκτά strings που είναι multiline.
- Δεν γίνονται δεκτά unmatched multiline comments
  Διευκρύνιση: Το πρόβλημα ήταν ότι γινόταν δεκτό το single line comment που αποτελείται από ένα μόνο $,
  Δηλαδη: ***$***
- Δεν γίνονται δεκτά inputs τύπου: 5mod 12.
  Λύση: Δημιουργήσαμε άλλο ένα regular expression που αναγνωρίζει tokens τύπου ***digit+ letter+***
  τα οποία απορρίπτονται στη συνέχεια από τον συντακτικό αναλυτή.
- Δεν γίνονται δεκτά strings που περιέχουν backslash χωρίς να πρόκειται για escape character, πχ: “***\Foo” , “\xLL“***

Μετατροπή CST -> AST

Υλοποιήθηκε κανονικά το Abstract Syntax Tree section.
Έχουν αποκοπεί κόμβοι απο το cst που ήταν περιττοί για τις επόμενες φάσεις της μεταγλώττισης.
Ωστόσο,κάποιοι άλλοι, φαινομενικά περιττοί, κρατήθηκαν ή προστέθηκαν για τη δημιουργία ενός δέντρου που θα ευνοεί τις φάσεις της μεταγλώττισης όπως τις υλοποιήσαμε προγραμματιστικά εμείς.

Υλοποίηση Symbol Table

Για την υλοποίηση του symbol table χρησιμοποιείται μια δομή hashtable καθώς και μια δομή stack. Η συνολική δομή είναι όμοια με αυτή που προτάθηκε στο piazza, καθώς και με αυτή που υπάρχει στις διαφάνειες της θεωρίας του μαθήματος.
Έχουν υλοποιηθεί οι μέθοδοι:

- **enter()**
- **insert(<args>)**
- **lookup()**
- **exit()**

Για κάθε εισαγωγή στο symbol table μιας μεταβλητής/συνάρτησης κρατώνται οι παρακάτω πληροφορίες:

```
Key name;                       //Hashkey
String type;
int scope;
Boolean reference;
LinkedList <Param> params;
LinkedList <Integer> arraylist;
String retvalue;                //Type Of Returned Value
Boolean defined;                //Function Defined or Declared?
```

Some of the above fields concern only the function variables, while others concern only the simple variables. In case a field doesn’t concern a variable, it gets null as a value.

Συντακτική Ανάλυση

- Έχει διορθωθεί ένα λάθος στη γραμματική που αφορούσε την προτεραιότητα του “not” σε σχέση με τα “or” , “and” .
- Τροποποιήθηκαν ελάχιστα τα productions l_values και block της γραμματικής για λόγους που αφορούν στην παραγωγή ενός συντακτικού/αφηρημένου δέντρου που ευνοεί/εξυπηρετεί περισσότερο την κατασκευή του symbol table και τη σημασιολογική ανάλυση.

Σημασιολογική Ανάλυση

- Στη σημασιολογική ανάλυση λάβαμε υπόψη μας όσο το δυνατόν περισσότερες περιπτώσεις λαθών γινόταν.
  Δεν αποκλείεται φυσικά η πιθανότητα κάποια να μας ξέφυγαν.
- Το πρόγραμμά μας αναγνωρίζει σαν σημασιολογικά λάθη εκτός των άλλων και τις εξής δύο περιπτώσεις για τις οποίες ειπώθηκε στα μαθήματα πως δε θα γίνει έλεγχος από τους εξεταστές:
  - Πέρασμα σε συνάρτηση πίνακα διαφορετικών διαστάσεων απο αυτές που αυτή περιμένει. (Εξαιρείται, προφανώς, η πρώτη διάσταση μιας παραμέτρου – πίνακα όταν αυτή είναι κενή “[]“).
  - Index πίνακα out of bounds. Όταν ένας πίνακας έχει ορισθεί ως var x:int[10] για παράδειγμα, τότε τα r-values : x[-1], x[10], x[11] κοκ είναι σημασιολογικά λάθος.

    Φυσικά, τέτοιος έλεγχος μπορεί να διεξαχθεί μονάχα όταν το index του πίνακα είναι γνωστός αριθμός κι όχι μεταβλητή. Διαφορετικά το λάθος αυτό μπορεί να αναγνωριστεί μονάχα σε run-time φάση.
- Για την υλοποίηση της σημασιολογικής ανάλυσης, χρησιμοποιήθηκε όπως προαναφέρθηκε μια βασική στοίβα που χρησιμεύει στην επικοινωνία των παιδιών με τους γονείς τους, και κάποιες άλλες “μικρότερης σημασίας” στοίβες, βοηθητικές για διάφορες λειτουργίες.
- Προσπαθούμε να εμφανίζουμε λάθη όσο το δυνατόν πιο περιγραφικά του εκάστοτε προβλήματος, για την διευκόλυνση τόσο του debugging όσο και των εξεταστών.
- Έχει προστεθεί η εμφάνιση του νούμερου της εκάστοτε γραμμής όπου εμφανίζεται το λάθος.

Ενδιάμεσος Κώδικας

- Για τον ενδιάμεσο κώδικα έχει γίνει χρήση ξανά μιας καινούριας στοίβας για την επικοινωνία των κόμβων παιδιών με τους κόμβους γονείς.

  Σε αυτήν την στοίβα αποθηκεύονται τώρα πια πληροφορίες που αφορούν την παραγωγή ενδιάμεσου κώδικα (Irelement class).

- Όλες οι τετράδες που παράγονται αποθηκεύονται στη μνήμη, συγκεκριμένα σε μια λίστα, για τη διευκόλυνση της παραγωγής του τελικού κώδικα.

Τελικός Κώδικας

- Για την παραγωγή τελικού κώδικα έχουν γίνει ελάχιστες, μη ιδιαίτερης σημασίας τροποποιήσεις στο συμβολισμό κάποιων προσωρινών μεταβλητών στις τετράδες του ενδιάμεσου κώδικα.

- Επιλέχθηκε υλοποίηση σε assembly x86

- Όλα υλοποιήθηκαν σύμφωνα με τις διαφάνειες των διαλέξεων (θεωρίας/ φροντιστηρίου) καθώς και τις συζητήσεις στο piazza.

- *Οδηγίες για την μεταγλώττιση κι εκτέλεση προγραμμάτων grace δίνεται στο section "Γενικά" του documentation.*

- *Τα 'char' variables είναι 4-byte aligned*
  Τα 'int' variables είναι 1-byte char
  Οι παράμετροι είτε char είτε int είναι 4-byte aligned.