Μεταγλωττιστές

Άσκηση Μέρος 6

Παραγωγή Bytecode

Το έκτο και τελευταίο μέρος αναφέρεται στην παραγωγή JVM bytecode, με την βοηθεια της ASM βιβλιοθήκης. Η διαδικασία αυτή έχει στόχο την παραγωγή jvm bytecode κάνοντας ένα επιπλέον πέρασμα στο ήδη υπάρχον αφηρημένο συντακτικό δέντρο. Το πέρασμα αυτό πραγματοποιείται από την κλάση ByteCodeGeneratorASTVisitor.

Αντικαθιστούμε το προηγούμενο Visitor με τον bytecode generator, και αντί για τις κλάσεις που δημιουργήσαμε στο προηγούμενο βήμα χρησιμοποιούμε τις αντίστοιχες του ASM με λογική ανάλογη με το 3 address code και τρόπο που αρμόζει στο JVM. Είναι αναγκαίο δηλαδή να λαμβάνουμε υπ'όψη μας την λειτουργία των Stack Frames (που ένα αντιστοιχεί σε μία κλήση συνάρτησης), και του κάθε operand stack και local variables array που έχει καθ'ένα από αυτά. Στη λογική της παραγωγής μας ενδιαφέρει κυρίως να διαχειριζόμαστε σωστά το operand stack. Για την σύνταξη σωστών οδηγιών για τις προσωρινές μεταβλητές στο local variables array, απαραίτητη είναι η κλάση LocalIdexPool η οποία προσομοιώνει το local variables array κάθε συνάρτησης στο JVM. Συνεπώς, μετά από την ολοκλήρωση του SymTableBuilderASTVisitor, γίνεται ένα πέρασμα από τον LocalIndexBuilderASTVisitor. Στόχος αυτού του περάσματος είναι η εισαγωγή κάθε μεταβλητης ή αναφοράς σε ένα πίνακα ανάλογα με την συνάρτηση που βρίσκεται, παίρνοντας έναν αύξον αριθμό. Στην συνέχεια μπορούμε να επαναχρησιμοποιούμε εύκολα τα Indexes για τα Local variables που αποθηκεύουμε προσωρινά.

Ο ByteCodeGeneratorASTVisitor φτιάχνει μία κλάση (ClassNode σύμφωνα με το ASM) με ένα defult constructor (μια μέθοδο init) και στην συνέχεια προσθέτει την main μέθοδο και τις υπόλοιπες μεθόδους αν υπάρχουν (MethodNode).

Στο ClassNode οι global variables ως static fields. Επιπλέον ελέγχουμε το αν η καλούμενη μεταβλητή είναι local ή global ώστε να γίνεται η σωστή κλήση. Παράγεται Short-Circuit code για τις λογικές πράξεις and και or με αυτόν τον visitor, με λογική παρόμοια με του 3 address code, αλλά συμβατή με το JVM.Υλοποιούμε την τεχνική του Short-Circuit σε κάθε δομή ελέγχου. Αυτό σημαίνει πως δεν γίνονται όλοι οι έλεγχοι αν μπορεί να υπολογιστεί το τελικό αποτέλεσμα από τους ελέγχους που έχουν γίνει ήδη. Αυτό προσφέρει στον κώδικα προς εκτέλεση καλύτερη απόδοση. Πιο συγκεκριμένα, ελέγχουμε σε μια πράξη AND αν ο πρώτος όρος είναι false και σε μια OR αν ο πρώτος όρος είναι true. Ξέρουμε έτσι από την αρχή ποιο είναι το αποτέλεσμα της λογικής πράξης και αποφεύγουμε παραπάνω πράξεις.

Όλες οι πράξεις που έχουμε δηλώσει ότι μπορούν να λειτουργούν στα προηγούμενα βήματα, πραγματοποιούνται. Το αντικειμενο που παράγεται με την χρήση των κλάσεων ClassWriter και TraceClassVisitor (που χρησιμοποιείται κυρίως για

debugging) από το asm πακέτο, μπορεί να μετατραπεί στο τελικό εκτελέσιμο “.class” αρχείο, το οποίο κατόπιν εκτελείται.
Εκτελώντας τα samples και το MyInput.txt βλέπουμε πως ο compiler βρίσκει σωστά τα λάθη αν υπάρχουν και παράγει σωστό κώδικα όταν το input ακολουθεί τους κανόνες μας, όπως άλλωστε βλέπουμε και από τα αποτελέσματα.
Για ελέγχους στον κεντρικό κατάλογο του project:

```
mvn package
java -jar target/compiler-0.0.7.jar samples/sample0.c
java -jar target/compiler-0.0.7.jar samples/sample1.c
java -jar target/compiler-0.0.7.jar samples/sample2.c
java -jar target/compiler-0.0.7.jar samples/sample3.c
java -jar target/compiler-0.0.7.jar samples/sample4.c
java -jar target/compiler-0.0.7.jar samples/sample5.c
java -jar target/compiler-0.0.7.jar samples/error0.c
java -jar target/compiler-0.0.7.jar samples/error1.c
java -jar target/compiler-0.0.7.jar samples/error2.c
java -jar target/compiler-0.0.7.jar samples/error3.c
java -jar target/compiler-0.0.7.jar samples/error4.c
java -jar target/compiler-0.0.7.jar MyInput.txt
```