ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

PROGRAMSKI PREVODIOCI 1
KOMPAJLER ZA MIKROJAVU

–Projekat–

Lazar Vasić 2010/123

Nastavnik: dr Dragan Bojić, vanr. prof.
Asistenti: dipl. ing. Nemanja Kojić,
dipl. ing. Maja Vukasović

Beograd, Septembar 2017.

Sadržaj

1. OPIS PROJEKTA

Cilj projektnog zadatka je realizacija kompajlera za programski jezik Mikrojavu. Kompajler omogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji se izvršava na virtuelnoj mašini za Mikrojavu. Sintaksno i semantički ispravni Mikrojava programi su definisani specifikacijom.

Programski prevodilac za Mikrojavu ima četiri osnovne funkcionalnosti: leksičku analizu, sintaksnu analizu, semantičku analizu i generisanje koda.

Leksički analizator treba da prepoznaje jezičke lekseme i vrati skup tokena izdvojenih iz izvornog koda, koji se dalje razmatraju u okviru sintaksne analize. Ukoliko se tokom leksičke analize detektuje leksička greška, potrebno je ispisati odgovarajuću poruku na izlaz.

Sintaksni analizator ima zadatak da utvrdi da li izdvojeni tokeni iz izvornog koda programa mogu formiraju gramatički ispravne sentence. Tokom parsiranja Mikrojava programa potrebno je na odgovarajući način omogućiti i praćenje samog procesa parsiranja na način koji će biti u nastavku dokumenta detaljno opisan. Nakon parsiranja sintaksno ispravnih Mikrojava programa potrebno je obavestiti korisnika o uspešnosti parsiranja. Ukoliko izvorni kod ima sintaksne greške, potrebno je izdati adekvatno objašnjenje o detektovanoj sintaksnoj grešci, izvršiti oporavak i nastaviti parsiranje.

Semantički analizator se dobija proširenjem funkcionalnosti sintaksnog analizatora. Semantička analiza se vrši sprovodi kroz sintaksno-upravljano prevođenje. Osnovnoj gramatici, kojom je specificiran sintaksni analizator, dodaju se atributi i akcije (atributivno-translaciona gramatika).

Generator koda prevodi sintaksno i semantički ispravne programe u izvršni oblik za odabrano izvršno okruženje Mikrojava VM. Generisanje koda se implementira na sličan način kao i semantička analiza, kroz nadogradnju sintaksnog analizatora (sintaksno upravljano prevođenje).

2. IMPLEMENTACIJA

Projekat je implementiran u Javi, koristeći razne Java-ine bibliote:

1) Za leksičku koristi se Jflex.
2) Za sintaksnu analizu JavaCup
3) Symboltable – za semantičku analizu, i implementaciju tabele simbol
4) Generator koda – za generisanje koda u Java VM.

Pored .cup i .jflex fajla u projektu se nalazi pomoćna klasa „Singleton“, u kojoj se nalaze pomoćne funkcije. Ova klasa je napisana radi lakšeg debug-ovganja i korišćenja *intellisense* u Eclipse IDE.

U projektu je takodje ne menjajući postojeu implementaciju tabele simbola izvedena klasa MySymbolTableVisitor iz klase SymbolTableVisitor, ovo je bilo potrebno uraditi kako bi se dodale zahtevana funkcionalnosti projekta. (Dodavanje bool kao novi tip u *universe scope).*

Projekat koristi Log4j biblioteku za rad sa logerom.

3. POKRETANJE

3.0. Kompajliranje

javac -cp .;src;config;lib\java-cup-11a.jar;lib\mj-runtime-1.1.jar;lib\symboltable.jar;lib\log4j-1.2.17.jar src\pp1\vl130298\util*.java src\pp1\vl130298*.java

3.1. JFlex

Ovaj alat nam omogućuje da sa ulaza pokupimo znake i od njih napravimo tokene. On generise sym.java i Yzlex.java klasu. Za njegovo pokretanje potrebno je pokrenuti JFlex.Main klasu sa arguemntima

java -jar lib\JFlex.jar -d /test\pp1\vl130298\MJTest.java spec\mjlexer.flex

-d src\pp1\vl130298 spec\mjlexer.flex

Gde kao argumente navodimo -d destination path – putanja gde želimo da se izgenerišu klase. Zatim source path , putanja do .flex fajla iz koga čitamo naše tokene i kako se oni generišu.

Prikaz konzole:

```
Reading "spec\mjlexer.flex"
Constructing NFA : 216 states in NFA
Converting NFA to DFA :
..........................................................................
.....................................
128 states before minimization, 118 states in minimized DFA
Old file "src\pp1\vl130298\Yylex.java" saved as "src\pp1\vl130298\Yylex.java~"
Writing code to "src\pp1\vl130298\Yylex.java"
```

Lexer test:

java -cp .;src;config;lib\java-cup-11a.jar;lib\mj-runtime-1.1.jar;lib\symboltable.jar;lib\log4j-1.2.17.jar pp1.vl130298.MJTest test\ARRAYS.mj output\lexer.lex

3.2. JavaCup

JavaCup alat nam od specifikacije gramatike u .cup fajlu generisemo naš parser. Takođe ovde u zahtevanim slučajevima radimo oporavak od greške kao i prebrojavanje. Alat se pokreće iz java_cup.Main klase.

java -jar lib\java-cup-11a.jar -destdir src\pp1\vl130298 -parser MJParser spec\mjparser.cup

-destdir src\pp1\vl130298 -parser MJParser spec\mjparser.cup

Prvi argument nam predstavlja destinacioni diretkoriju u koji ce se generisati MJParser.java klasa, drugi arguemnt je source path našeg .cup fajla.

Prikaz konzole:

```
------- CUP v0.11a beta 20060608 Parser Generation Summary -------
  0 errors and 0 warnings
  51 terminals, 96 non-terminals, and 180 productions declared,
  producing 280 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "MJParser.java", and "sym.java".
---------------------------------------------------- (v0.11a beta 20060608)
```

Parser test:

java -cp .;src;config;lib\java-cup-11a.jar;lib\mj-runtime-1.1.jar;lib\symboltable.jar;lib\log4j-1.2.17.jar pp1.vl130298.MJParserTest test\functions.mj test\program.obj

java -cp .;src;config;lib\java-cup-11a.jar;lib\mj-runtime-1.1.jar;lib\symboltable.jar;lib\log4j-1.2.17.jar pp1.vl130298.MJParserTest test\functions.mj > test\functions.out 2> test\functions.err

3.3. Disasm

Alat za disembliranje. Pokreće ga klasa rs.etf.pp1.mj.runtime.disasm

test\program.obj

3.4. Run

Alat koji koristimo za generisanje i pokretanje koda na Java VM. Za pokretanje ovog alata koristi se klasa rs.etf.pp1.mj.runtime.Run. Kao argumen ova klasa treba da dobije objekti fajl koji je ranije izgenerisan. Ovu klasu takodje možemo da pokrenemo u debug režimu tako da možemo da vidimo svaku izvršenu instrukciju i ispis estack-a.

test\program.obj

java -cp lib\mj-runtime-1.1.jar rs.etf.pp1.mj.runtime.Run test\program.obj

4. TESTOVI

U paketu *test* nalaze se testovi.

Neki od osnovnih testova:

1) Javni testovi
2) *Operators.mj* – testira operatore
3) *If_for*.mj – za kontrolne strukture
4) *Functions*.mj – funkcije
5) *Vectors*.mj – nizove

...