Univerzitet u Beogradu

Elektrotehnčki fakultet



Kompajler za Mikrojavu

Projekat iz predmta Programski prevodioci 1

Iva Perić 2019/0392

Beograd, avgust 2023.

**Kratak opis postavke zadatka:**

Cilj projektnog zadatka je realizacija kompajlera za programski jezik Mikrojavu. Kompajleromogućava prevodjenje sintaksno i semantički ispravnih Mikrojava programa u Mikrojava bajtkod koji se izvršava na virtuelnoj mašini za Mikrojavu. Kompajler za Mikrojavu ima četiri faze: leksičku analizu, sintaksnu analizu, semantičku analizu i generisanje koda.

**Leksički analizator** prepoznaje jezičke lekseme i vratća skup tokena izdvojenih izizvornog koda, koji se dalje razmatraju u okviru sintaksne analize. Ukoliko se tokom leksičke analize detektuje leksička greška, odnosno ako leksički analizator ne može da očita neki token, prijavljuje se leksička greška kao i linija na kojoj se dogodila.

**Sintaksni analizator** ima zadatak da utvrdi da li izdvojeni tokeni iz izvornog koda programa mogu formiraju gramatički ispravne sentence i da kreira apstraktno sintaksno stablo na osnovu gramatike jezika. Izlaz je kreirano apstraktno sintaksno stablo koje se dalje koristi u preostale dve faze projekta. Ukoliko sintaksni analizator detektuje sintaksnu gresku, ispisuje se adekvatno objašnjenje o detektovanoj grešci, izvršava se oporavak (vrste oporavka su date u nastavku tekstualnog fajla) i nastavlja se parsiranje. Ukoliko se desi greška nepredvidjena oporavkom, prijavljuje se fatalna greška i zaustavlja se dalja obrada. Nakon parsiranja sintaksno ispravnih Mikrojava programa obaveštava se korisnik o uspešnosti parsiranja.

**Semantički analizator** je faza obilaska apstraktnog sintaksnog stabla radi provere semantičke ispravnosti programa koristeći bottom-up parsiranje. Prilikom posete čvorova, proverava se poštovanje semantičkih pravila datih u tekstu projekta, a u slučaju nepoštovanja nekog pravila, generiše se greška sa opisom problema i na kojoj liniji je nastao. Na kraju svog izvršavanja, prikazuje se poruka o uspešnosti parsiranja i stanje generisane tabele simbola. Za izradu semantičkog analizatora koristi se symboltable-1.1.jar.

**Generator koda** prevodi sintaksno i semantički ispravne programe u izvršni oblik zaodabrano izvršno okruženje Mikrojava VM. Generisanje koda se implementira na sličan način kao i semantička analiza, implementacijom metoda koje posećuju čvorove. Prve tri faze projekta se izvršavaju svakako, nezavisno od grešaka u izvornom kodu, dok se faza za generisanje koga izvršava samo ukoliko je parsiranje uspešno i ukoliko nije došlo do greške u prethodnim fazama. Za njegovu izradu se koristi mj-runtime-1.1.jar.

**Pokretanje projekta:**

Za implementaciju kompajlera za Mikrojava jezik korišćeno je Eclipse razvojno okruženje. U svrhu automatizacije procesa i povezivanja i prevodjenja izvornih fajlova projekta napisana je build.xml ANT skripta..

Generisanje klase leksičkog analizatora Yylex.java:

Na osnovu specifikacionog fajla mjlexer.flex generiše se klasa Yylex.java koja predstavlja izvorni kod leksičkog analizatora. Klasa Yylex.java generiše se stavljanjem sledećeg targeta u build.xml fajl:

**<target** **name=***"lexerGen"* **depends=***"delete"***>**

**<java** **jar=***"lib/JFlex.jar"* **fork=***"true"***>**

**<arg** **value=***"-d"***/>**

**<arg** **value=***"./src/rs/ac/bg/etf/pp1"***/>**

**<arg** **value=***"spec/mjlexer.flex"***/>**

**</java>**

**</target>**

Generisanje klase LALR(1) parsera i klase za simbole:

Na osnovu specifikacionog fajla mjparser.cup generiše se klasa MJParser.java koja predstavlja izvorni kod parsera, u istom folderu generiše se i klasa sym.java. Kreiraće se i dodatna ast specifikacija u fajlu mjparser\_astbuild.cup.

Generisanje se pokreće dodavanjem sledećeg targeta u build.xml fajl:

**<target** **name=***"parserGen"* **depends=***"lexerGen"***>**

**<java** **jar=***"lib/cup\_v10k.jar"* **fork=***"true"***>**

**<arg** **value=***"-destdir"***/>**

**<arg** **value=***"src/rs/ac/bg/etf/pp1"***/>**

**<arg** **value=***"-ast"***/>**

**<arg** **value=***"src.rs.ac.bg.etf.pp1.ast"***/>**

**<arg** **value=***"-parser"***/>**

**<arg** **value=***"MJParser"***/>**

**<arg** **value=***"-dump\_states"***/>**

**<arg** **value=***"-buildtree"***/>**

**<arg** **value=***"spec/mjparser.cup"***/>**

**</java>**

**</target>**

Имплементација главног програма :

Implementacija glavnog programa nazvana je MJParserTest.java. Sastoji se od niza sukcesivnih koraka koji prvo pokreću leksičku analizu pa sintaksnu pa semantičku za koju se koristi klasa iz istog paketa SemanticPass.java. Za generisanje koda se koristi klasa CodeGenerator.java koja se poziva samo ukoliko je parsiranje uspešno nakon prve tri faze. Prvi argument main funkcije klase MJParserTest predstavlja fajl iz kojeg će se čitati mikrojava kod, a drugi argument predstavlja izlazni .obj fajl.

Disasembliranje:

Prevedeni kod se iz objektnog fajla može disasemblirati korišćenjem klase rs.etf.pp1.mj.runtime.disasm. Disasembliranje se pokreće dodavanjem i zasebnim pokretanjem sledećeg targeta u build.xml fajl:

**<target** **name=***"disasm"***>**

**<java** **classname=***"rs.etf.pp1.mj.runtime.disasm"***>**

**<arg** **value=***"test/program.obj"***/>**

**<classpath>**

**<pathelement** **location=***"lib/mj-runtime-1.1.jar"***/>**

**</classpath>**

**</java>**

**</target>**

Pokretanje prevedenog programa:

Prevedeni program se može pokrenuti na mikrojava virtuelnoj mašini korišćenjem klase rs.etf.pp1.mj.runtime.Run. Pokretanje mikrojava programa se vrši dodavanjem i zasebnim pokretanjem sledećeg targeta u buil.xml fajl:

**<target** **name=***"runObj"* **depends=***"disasm"* **>**

**<java** **classname=***"rs.etf.pp1.mj.runtime.Run"***>**

**<arg** **value=***"test/program.obj"***/>**

**<arg** **value=***"-debug"***/>**

**<redirector** **input=***"test/input.txt"* **/>**

**<classpath>**

**<pathelement** **location=***"lib/mj-runtime-1.1.jar"***/>**

**</classpath>**

**</java>**

**</target>**

Za čitljiviji ispis zadužena je biblioteka log4j.jar.

**Greške koje su definisane za oporavak u sintaksnom analizatoru:**

U AST-CUP specifikaciju gramatike dodate su smene i akcije za oporavak od grešaka za sledeće jezičke elemente:

– definicija globalne promenljive – ignorišu se karakteri do prvog znaka ";" ili sledećeg ","

– konstrukcija iskaza dodele – ignorišu se karakteri do ";"

– deklaracija formalnog parametra funkcije – ignorišu se znakovi do znaka "," ili ")"

– logički izraz unutar if konstrukcije – ignorišu se karakteri do prvog znaka ")"

**Opis priloženih test primera:**

• Test301.mj predstavlja javni test za nivo A za sve 4 faze.

• Тest302.mj predstavlja javni test za nivo B za prve 3 faze.

• program.mj testira metodu findAny

• test1.mj testira ispravnost svih funkcionalnosti za B nivo za prve 3 faze

• test1ParserRadi.mj sadrži skup instrukcija za testiranje ispravnosti sintaksnog analizatora

• test1Parser1.mj, test1Parser2.mj i test1Parser3.mj sadrže skupove grešaka za koje je predvidjen oporavak od greške u fazi sintaksne analize.

• test2SemantickaRadi.mj sadrži skup instrukcija za testiranje ispravnosti semantičkog analizatora.

• test2Semanticka1.mj, test2Semanticka2.mj i test2Semanticka3.mj sadrže skupove instrukcija koje prikazuju preciznost detektovanja grešaka u ulaznom fajlu.

• test3Code.mj sadrži test za generisanje koda u kojem proverava rad promenljivih sa svim operacijama.

• test3Codee.mj sadrži test za generisanje koda u kojem proverava rad sa nizovima i svim tipovima podataka.

**Opis novouvedenih klasa:**

Sym.java – automatski generisana klasa sa mjlexer.lex

Mjparser.cup –koristi CUP alat, sadrži definisanu gramatiku i generiše MJParser.java klasu

MJParser.java – automatski generisana klasa sa mjparser.cup

SemanticPass.java – služi za semantičku analizu tokom obilaska apstraktnog sintsksnog stabla

CodeGenerator.java – služi za generisanje Mikrojava bajtkoda tokom obilaska apstraktnog sintaksnog stabla

TabIv.java je klasa izvedena iz Tab.java i služi za dodavanje nove strukture boolType.

StructIv.java je klasa izvedena iz Tab.java i služi za dodavanje public static final int Bool = 5;

CounterVisitor.java je klasa izvedena iz VisitorAdaptor, ima u sebi dve unutrašnje klase koje služe za prebrojavanje formalnih parametara i promenljivih.