Univerzitet u Beogradu Elektrotehnicki fakultet

MJProjekat

Projekat iz Programskih prevodioca 1

Nikola Subaric 2020/0271 Datum:17.1.2024.

Contents

[O projektu 3](#_Toc156325301)

[Faze projekta 4](#_Toc156325302)

[Leksicka analiza 4](#_Toc156325303)

[Sintaksna analiza 4](#_Toc156325304)

[Semanticka analiza 4](#_Toc156325305)

[Generisanje koda 4](#_Toc156325306)

[Korisceni alati 5](#_Toc156325307)

[Test primeri 7](#_Toc156325308)

[Novouvedene klase 13](#_Toc156325309)

[Kraj 13](#_Toc156325310)

# O projektu

Cilj samog projekta je bio da se upoznamo sa fazama kroz koje se prolazi da bi se doslo do izvrsnog koda. Za to nam je posluzio jezik MikroJava ciji source kod se prevodi do bytecode-a koji se izvrsava na MikroJava virtuelnoj masini.

Faze kroz koje se prolazilo su sledece:

1. Leksicka analiza
2. Sintaksna analiza
3. Semanticka analiza
4. Generisanje koda

# Faze projekta

## Leksicka analiza

U procesu leksicke analize sam source kod se tokenizovao pomocu Lex alata. Tako dobijeni tokeni su se dalje prosledjivali na obradu parseru.

## Sintaksna analiza

U procesu sintaksne analize proveravala se gramaticka ispravnost na osnovu gramatike koja je bila specificirana u .cup fajlu. Ova faza funkcionise tako sto se jedan po jedan token dovlaci pomocu leksera I ispituje kojem delu gramatike (ne) odgovara.

## Semanticka analiza

U procesu semanticke analize se sama analiza podize na visi nivo. Sada vise nije dovoljno da tokeni budu slozeni po gramatici, vec se uvodi I logicki smisao redosleda tokena. Na primer na mestu gde se ocekuje naziv promenljive nije moguce da se naidje na tip promenljive I slicno.

## Generisanje koda

U procesu generisanja koda sve staticke stvari su proverene. Sada jos preostaje da sam kod spustimo na nivo bytecoda koji je moguce izvrsiti na MikroJava virtuelnoj masini. To jem oguce uciniti pomocu rs.etf.pp1.mj.runtime.jar arhive.

# Korisceni alati

Radi lakseg manipulisanja, kako delovima projekta, tako I celokupnim projektom, koriscen je Ant alat. U nastavku je prilozen sadrzaj istog:

<?xml version="1.0" encoding="UTF-8"?>

<project name="MJCompiler" default="compile" basedir=".">

<target name="delete">

<delete>

<fileset dir="src/rs/ac/bg/etf/pp1">

<exclude name="sym\_old.java"/>

<exclude name="sym.java"/>

<exclude name="util/Log4JUtils.java"/>

<exclude name="RuleVisitor.java"/>

<exclude name="Yylex.java"/>

<exclude name="SemanticPass.java"/>

<exclude name="CodeGenerator.java"/>

<exclude name="CounterVisitor.java"/>

<exclude name="nTab.java"/>

</fileset>

</delete>

<delete>

<fileset dir="spec">

<exclude name="mjlexer.lex"/>

<exclude name="mjparser.cup"/>

</fileset>

</delete>

</target>

<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.lex"/>

</java>

</target>

<target name="parserGen" depends="delete">

<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>

<target name="repackage" depends="parserGen">

<!-- Replaces all of the references to the old package name in files in the "src" directory -->

<replace dir="src" value="rs.ac.bg.etf.pp1.ast" token="src.rs.ac.bg.etf.pp1.ast" summary="true"/>

</target>

<target name="compile" depends="repackage">

<javac srcdir="src/rs/ac/bg/etf/pp1" includeantruntime="false">

<classpath>

<pathelement path="lib/JFlex.jar"/>

<pathelement path="lib/cup\_v10k.jar"/>

<pathelement path="lib/log4j-1.2.17.jar"/>

<pathelement path="lib/symboltable.jar"/>

<pathelement path="lib/mj-runtime.jar"/>

</classpath>

</javac>

</target>

<target name="disasm">

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

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

<classpath>

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

</classpath>

</java>

</target>

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

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

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

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

<classpath>

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

</classpath>

</java>

</target>

</project>

# Test primeri

1. Nadogradjen test primer 301 sa bool tipovima

// Test301

program test301

namespace Konstante{

const int nula = 0;

const int jedan = 1;

const int pet = 5;

{}

}

namespace Nizovi{

int niz[];

char nizch[];

bool boolNiz[];

{}

}

int a;

{

bool verify() {

print('v');

return false;

}

void main()

int bodovi;

bool bt;

int citanje;

bool nizBool[];

{

bt = false;

print(bt);

nizBool = new bool[3];

nizBool[1] = true;

print(nizBool[1]);

nizBool[1] = false;

print(nizBool[1]);

print('A');

Nizovi::boolNiz = new bool[4];

Nizovi::boolNiz[2] = true;

print(Nizovi::boolNiz[2]);

bodovi = 0;

bodovi++;

bodovi = bodovi + Konstante::jedan;

bodovi = bodovi \* Konstante::pet;

bodovi--;

print(bodovi);

Nizovi::niz = new int[3];

Nizovi::niz[Konstante::nula] = Konstante::jedan;

Nizovi::niz[1] = 2;

Nizovi::niz[Nizovi::niz[Konstante::jedan]] = Nizovi::niz[Nizovi::niz[0]] \* 3;

bodovi = Nizovi::niz[2]/Nizovi::niz[0];

print(bodovi);

print(Nizovi::niz[2]);

Nizovi::nizch = new char[3];

Nizovi::nizch[0] = 'a';

Nizovi::nizch[Konstante::jedan] = 'b';

Nizovi::nizch[Konstante::pet - 3] = 'c';

print(Nizovi::nizch[1]);

print(Nizovi::nizch[Konstante::jedan \* 2]);

a=1;

read(bodovi);

bodovi = -bodovi + (Konstante::pet \* a / 2 - Konstante::jedan) \* bodovi - (3 % 2 + 3 \* 2 - 3);

print(bodovi);

//read(citanje);

}

}

1. Poziv metode sa praznim telom test

// Test301

program test301

namespace Konstante{

const int nula = 0;

const int jedan = 1;

const int pet = 5;

{}

}

namespace Nizovi{

int niz[];

char nizch[];

bool boolNiz[];

{}

}

int a;

{

bool verify() {

print('v');

return false;

}

void main()

{

verify();

}

}

1. Poziv globalne metode sa prosledjenim argumentom test

// Test301

program test301

namespace Konstante{

const int nula = 0;

const int jedan = 1;

const int pet = 5;

{}

}

namespace Nizovi{

int niz[];

char nizch[];

bool boolNiz[];

{}

}

int a;

{

bool verify() {

print('v');

return false;

}

void saJednimParametrom(char x){

print(x);

}

void main()

{

verify();

saJednimParametrom('s');

}

}

1. Test sa sirinama za print

// Test301

program test301

namespace Konstante{

const int nula = 0;

const int jedan = 1;

const int pet = 5;

{}

}

namespace Nizovi{

int niz[];

char nizch[];

bool boolNiz[];

{}

}

int a;

{

bool verify() {

print('v');

return false;

}

void saJednimParametrom(char x){

print(x, 4);

}

void main()

{

verify();

saJednimParametrom('s');

print(-1, 4);

}

}

# Novouvedene klase

Radi lakseg rada I povecanja citljivosti za neke od faza uvedene su nove klase. Za samu semanticku analizu uvedena je klasa SemanticPass koja je imala za cilj laksi prolazak kroz samo AST stablo. S obzirom da je izvedena iz klase VisitorAdaptor koja je generisana pomocu CUP alata, lako se mogu obici pojedinacni cvorovi. Pored nje, uvedena je I klasa CodeGenerator koja je takodje izvedena iz klase VisitorAdaptor takodje iz razloga lakseg pristupa Samim cvorovima AST. Tu je bilo moguce pripremati samo stanje steak, dodavanje instrukcija ukoliko je za tim bilo potrebe na odredjeni mestima.

# Kraj

Sam projekat je osim velikog truda I zalaganja iziskivao I strpljenje I smrenost da bi bio realizovan. Iako na prvi pogled obiman projekat, zaista je bilo zanimljivo raditi na stvaranju neceg sa cim se svakodnevno susrecem.