****

**De La Salle University • College of Computer Studies**

**Maestro**

Context-Free Grammar and Parser

Name (last name first) : Fernandez, Ryan Austin

Poblete, Clarisse Felicia M.

San Pedro, Marc Dominic

Tan, Johansson E.

Section : G01

Date of Submission : February 4, 2016

**Context-Free Grammar**

MAESTRO = (V,T,P,PROG)

V = {PROG,NL,ELEM,SUBELEM,VAR,ELEMS,SEQ,SYNC,PLAY,NOTE,REST,TIME,DEC,SUBBODY}

T = {nl,+,++,-,--,<,<<,>,>>,\*,varname,[,num,],~,seq,sync,play,pitch,(,<comma>,),rest,time,.,->,{,}}

P is defined by

PROG -> ELEMS nl PLAY nl ELEMS | PLAY nl ELEMS | ELEMS nl PLAY | PLAY | NL PROG NL

NL -> nl | 

ELEM -> SYNC | SEQ | NOTE | REST | VAR | TIMES | SYNC SUBELEM | SEQ SUBELEM | NOTE SUBELEM | REST SUBELEM

| VAR SUBELEM

SUBELEM -> + num SUBELEM | ++ SUBELEM | - num SUBELEM | -- SUBELEM | < num SUBELEM | << SUBELEM

| > num SUBELEM | >> SUBELEM | \* num | 

VAR -> varname | varname[num] | varname[num~num]

ELEMS -> ELEM nl ELEMS | ELEM | DEC | DEC nl ELEMS

SEQ -> seq SUBBODY | seq nl SUBBODY

SYNC -> sync SUBBODY | sync nl SUBBODY

PLAY -> play SUBBODY | play nl SUBBODY

NOTE -> pitch(num,TIME)

REST -> rest(TIME)

TIME -> time | time. | time.. | time...

DEC -> ELEM -> varname

SUBBODY -> { nl ELEMS nl }

**Parser**

The parser is an LL(1) top-down parser with slight modifications. Since the left-factored, left-recursion-less grammar is not LL(1) at one production, the parser looks ahead two tokens in that one special case.

The parser takes the parsing table as input in the form of a csv file.

**Driver.java**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.PrintWriter;

import java.util.Iterator;

import java.util.List;

public class Driver {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(

new FileReader(

new File(args[0])));

String code = "";

String s;

do {

s = br.readLine();

if( s != null ) {

code += s + "\n";

}

} while(s != null);

br.close();

if( code.endsWith("\n") ) {

code = code.substring(0,code.length() - 1);

}

Tokenizer t = new Tokenizer(code);

t.tokenize();

List<Token> tokens = t.getTokens();

Parser p = new Parser(tokens);

System.out.println((p.parse() ? "Valid" : "Invalid") + " code");

}

}

**Parser.java**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

public class Parser {

private List<Token> tokens;

private HashMap<String,HashMap<String,String>> parseTable;

private ArrayList<String> stack;

private PrintWriter pw;

public Parser(List<Token> tokens) {

this.tokens = tokens;

parseTable = new HashMap<String,HashMap<String,String>>();

stack = new ArrayList<String>();

ArrayList<String> terminals = new ArrayList<String>();

try {

BufferedReader br = new BufferedReader(

new FileReader(

new File("CFG.csv")));

String terms = br.readLine();

String[] termList = terms.split(",");

for(int i = 1; i < termList.length; i++) {

terminals.add(termList[i]);

}

String s = "";

while( true ) {

s = br.readLine();

if( s != null ) {

String[] prods = s.split(",");

parseTable.put(prods[0],new HashMap<String,String>());

HashMap<String,String> temp = parseTable.get(prods[0]);

for(int i = 1; i < prods.length; i++) {

if( prods[i].length() > 0 ) {

temp.put(terminals.get(i - 1),prods[i].replaceAll("\\|",","));

}

}

} else {

break;

}

}

br.close();

} catch(IOException ioe) {

ioe.printStackTrace();

}

}

public boolean parse() {

try {

pw = new PrintWriter(

new BufferedWriter(

new FileWriter(

new File("logs.txt"))));

push("PROG");

int i = 0;

boolean error = false;

while(i <= tokens.size() && stack.size() > 0) {

String top = peek();

Token currToken = i == tokens.size() ? new Token("EOF","EOF"

,tokens.get(tokens.size() - 1).lineNo())

: tokens.get(i);

if( currToken.type().equals("other") ) {

System.out.println("Unrecognized token " + currToken.token()

+ " at line " + currToken.lineNo() );

pw.println("Unrecognized token " + currToken.token()

+ " at line " + currToken.lineNo() );

}

if( isVariable(top) ) {

String prod = parseTable.get(top).get(currToken.type());

if( prod == null ) {

System.out.println("Error: unexpected " + currToken);

pw.println("Error: unexpected " + currToken);

i++;

error = true;

} else if( prod.equals("SYNCH") ) {

pop();

error = true;

} else {

if( prod.equals("newline ELEMS")) {

String next = i + 1 < tokens.size()

? tokens.get(i + 1).type()

: "EOF";

String[] change = new String[] {

"}","EOF","play"

};

boolean willChange = false;

for(String s: change) {

if( next.equals(s)) {

willChange = true;

break;

}

}

if( willChange ) {

prod = "e";

}

}

updateStack(prod);

}

} else if( currToken.type().equals(top)) {

i++;

pop();

} else {

if( currToken.type().equals("newline")) {

pop();

} else {

i++;

System.out.println("Error at token " + currToken + " expected " + top);

pw.println("Error at token " + currToken);

}

error = true;

}

}

pw.close();

return !error && stack.size() == 0 && i == tokens.size();

} catch(IOException ioe) {

ioe.printStackTrace();

return false;

}

}

public void printStack() {

if( stack.size() == 0) {

System.out.println("Stack empty");

pw.println("Stack empty");

} else {

for(int i = stack.size() - 1; i >= 0; i-- ) {

System.out.print(stack.get(i) + " ");

pw.print(stack.get(i) + " ");

}

System.out.println();

pw.println();

}

}

private String peek() {

if( stack.size() > 0 ) {

return stack.get(stack.size() - 1);

} else {

return "Nothing to display";

}

}

private String pop() {

if( stack.size() > 0 ) {

String top = stack.get(stack.size() - 1);

stack.remove(stack.size() - 1);

return top;

} else {

return "Nothing to display";

}

}

private void push(String s) {

stack.add(s);

}

private boolean isVariable(String s) {

return s.charAt(0) >= 'A' && s.charAt(0) <= 'Z';

}

private void updateStack(String production) {

String[] arr = production.split(" ");

pop();

for(int i = arr.length - 1; i >= 0; i--) {

if( arr[i].length() > 0) {

if(!arr[i].equals("e")) {

push(arr[i]);

}

}

}

}

}

**Appendix: Parsing Table CSV File**

Each row of the parsing table save for the first is a variable. Each column save for the first is a terminal. The cells contain the production to be applied.

,sync,seq,pitch,rest,varname,play,newline,+,++,-,--,<,<<,>,>>,\*,->,[,],~,time,{,.,),EOF

PROG,ELEMS newline PLAY SUBELEMS2 NL,ELEMS newline PLAY SUBELEMS2 NL,ELEMS newline PLAY SUBELEMS2 NL,ELEMS newline PLAY SUBELEMS2 NL,ELEMS newline PLAY SUBELEMS2 NL,PLAY SUBELEMS2 NL,newline PROG NL,,,,,,,,,,,,,,,,,,SYNCH

ELEM,SYNC SUBELEM,SEQ SUBELEM,NOTE SUBELEM,REST SUBELEM,VAR SUBELEM,,SYNCH,,,,,,,,,,SYNCH,,,,,,,,

SUBELEM,,,,,,,e,+ num SUBELEM,++ SUBELEM,- num SUBELEM,-- SUBELEM,< num SUBELEM,<< SUBELEM,> num SUBELEM,>> SUBELEM,\* num,e,,,,,,,,SYNCH

VAR,,,,,varname SUBVAR,,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,,,

SUBVAR,,,,,,,e,e,e,e,e,e,e,e,e,e,e,[ num SUB2VAR,,,,,,,

SUB2VAR,,,,,,,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,],~ num ],,,,,

ELEMS,ELEM SUBELEMS,ELEM SUBELEMS,ELEM SUBELEMS,ELEM SUBELEMS,ELEM SUBELEMS,,e,,,,,,,,,,e,,,,,,,,e

SUBELEMS,,,,,,,newline ELEMS,,,,,,,,,,-> varname SUBELEMS2,,,,,,,,e

SUBELEMS2,,,,,,,newline ELEMS,,,,,,,,,,,,,,,,,,e

SEQ,,seq SUBBODY,,,,,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,,,

SYNC,sync SUBBODY,,,,,,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,,,

PLAY,,,,,,play SUBBODY,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,,,,SYNCH

SUBBODY,,,,,,,newline { newline ELEMS newline },SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,{ newline ELEMS newline },,,SYNCH

NOTE,,,pitch ( num | TIME ),,,,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,,,

REST,,,,rest ( TIME ),,,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,SYNCH,,,,,,,,

TIME,,,,,,,,,,,,,,,,,,,,,time SUBTIME,,,,

SUBTIME,,,,,,,,,,,,,,,,,,,,,,,. SUB2TIME,e,

SUB2TIME,,,,,,,,,,,,,,,,,,,,,,,. SUB3TIME,e,

SUB3TIME,,,,,,,,,,,,,,,,,,,,,,,.,e,

NL,,,,,,,newline,,,,,,,,,,,,,,,,,,e