Лабораторная работа № 2.3 «Синтаксический анализатор на основе предсказывающего анализа»

18 апреля 2023 г.

Самохвалова Полина, ИУ9-62Б

Цель работы

Целью данной работы является изучение алгоритма построения таблиц предсказывающего анализатора.

Индивидуальный вариант

```
$AXIOM E
$NTERM E' T T' F
$TERM  "+"  "*"  "("  ")"  "n"

* правила грамматики
$RULE  E  = T E'
$RULE  E' = "+" T E'
$EPS
$RULE  T  = F T'
$RULE  T' = "*" F T'
$EPS
$RULE  F  = "n"
"(" E ")"
```

Реализация

Неформальное описание синтаксиса входного языка

Терминалы: “AxiomKeyword”, “NTermKeyword”, “TermKeyword”, “RuleKeyword”, “EpsKeyword”, “Nterm”, “Term”, “Equal”, “NewLine”

“AxiomKeyword”, “NTermKeyword”, “TermKeyword”, “RuleKeyword”, “EpsKeyword” - токены для ключевых слов

```
"AxiomKeyword" ::= "\\$AXIOM"
"NTermKeyword" ::= "\\$NTERM"
"TermKeyword" ::= "\\$TERM"
"RuleKeyword" ::= "\\$RULE"
"EpsKeyword" ::= "\\$EPS"
```

“Nterm” и “Term” - токены для нетерминалов и терминалов

```
"Nterm" ::= "[A-Z][A-Z_]*'|[A-Z][A-Z_]*[0-9]|[A-Z][A-Z_]*"
"Term" ::= "\"[a-zA-Z\\+\\*\\(\\)]+\""
```

“Equal” - токен для обозначения знака равенства

```
"Equal" ::= "="
```

Новая строка выделяется в отдельный токен “NewLine”, так как она служит для разделения альтернативами правых частей правил

```
"NewLine" ::= "\\n+"
```

Аксиома грамматики: S

Нетерминалы: NTERMS, TERMS_DEF, TERMS, RULES_DEF, RULES, RULE, R, R1, R2, R3

```
S ::= "AxiomKeyword" "Nterm" "NTermKeyword" "Nterm" NTERMS
TERMS_DEF RULES_DEF
```

NTERMS - последовательность нетерминалов

```
NTERMS ::= "Nterm" NTERMS | ε
```

TERMS_DEF - начинающаяся с ключевого слова последовательность терминалов

```
TERMS_DEF ::= "TermKeyword" "Term" TERMS
```

TERMS - последовательность терминалов

```
TERMS ::= "Term" TERMS | ε
```

RULES_DEF - непустая последовательсть правил

```
RULES_DEF ::= RULE RULES
```

RULES - последовательность правил

```
RULES ::= RULE RULES | ε
```

RULE - правило

```
RULE ::= "RuleKeyword" "Nterm" "Equal" R
```

R - правая часть правила

```
R ::= R1 R2
R1 ::= "Term" R3 | "Nterm" R3 | "EpsKeyword"
R3 ::= "Term" R3 | "Nterm" R3 | ε
R2 ::= "NewLine" R | ε
```

При получении нового токена токены “NewLine”, служащие для перехода на новую строку перед ключевыми словами $AXIOM, $NTERM, $TERM, %RULE будут пропускаться. То есть из токенов “Newline” учитываться будут только те, которые служат для разделения правых частей праил альтернативами

Лексическая структура

```
"AxiomKeyword" ::= "\\$AXIOM"
"NTermKeyword" ::= "\\$NTERM"
"TermKeyword" ::= "\\$TERM"
"RuleKeyword" ::= "\\$RULE"
"EpsKeyword" ::= "\\$EPS"
"Nterm" ::= "[A-Z][A-Z_]*'|[A-Z][A-Z_]*[0-9]|[A-Z][A-Z_]*"
"Term" ::= "\"[a-zA-Z\\+\\*\\(\\)]+\""
"Equal" ::= "="
"NewLine" ::= "\\n+"
```

Грамматика языка

```
S ::= "AxiomKeyword" "Nterm" "NTermKeyword" "Nterm" NTERMS
TERMS_DEF RULES_DEF
NTERMS ::= "Nterm" NTERMS | ε
TERMS_DEF ::= "TermKeyword" "Term" TERMS
TERMS ::= "Term" TERMS | ε
RULES_DEF ::= RULE RULES
RULES ::= RULE RULES | ε
RULE ::= "RuleKeyword" "Nterm" "Equal" R
R ::= R1 R2
R1 ::= "Term" R3 | "Nterm" R3 | "EpsKeyword"
R3 ::= "Term" R3 | "Nterm" R3 | ε
R2 ::= "NewLine" R | ε
```

Программная реализация

Fragment.java

```
public class Fragment {
    private String image;
    private Position start, follow;

    public Fragment(String image, Position start, Position
            follow) {
```

```
        this.image = image;
        this.start = start;
        this.follow = follow;
    }

    @Override
    public String toString(){
        return String.format("COMMENT %s-%s %s",
                start.toString(), follow.toString(), image);
    }
}
```

GrammarStructure.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedHashMap;

class GrammarParser extends Parser {
    GrammarParser() {
        super(GrammarStructure.termsList,
                GrammarStructure.nontermsList,
                GrammarStructure.axiom, GrammarStructure.q);
    }
}

class GrammarStructure {

    final static ArrayList<String> termsList = getTermTags();
    final static ArrayList<String> nontermsList =
            getNonTermTags();
    final static Nonterm axiom = new Nonterm("S");
    final static HashMap<String, String> tagsRegex =
            getTagRegex();
    final static HashMap<String, Rules> grammarList =
            getGrammar();
    final static RHS[][] q = analiseTable();

    private static ArrayList<String> getTermTags() {
        return new ArrayList<>(Arrays.asList(
                "AxiomKeyword", "NTermKeyword", "TermKeyword",
                "RuleKeyword", "EpsKeyword", "Term", "Nterm",
                "Equal", "NewLine", Term.EOF
        ));

    }
```

```
private static ArrayList<String> getNonTermTags() {
    return new ArrayList<>(Arrays.asList(
            "S", "NTERMS", "TERMS_DEF", "TERMS",
            "RULES_DEF", "RULES", "RULE", "R", "R1", "R2",
            "R3"
    ));
}

private static HashMap<String, String> getTagRegex() {
    LinkedHashMap<String, String> exprs =
            new LinkedHashMap<>();
    exprs.put("AxiomKeyword", "\\$AXIOM");
    exprs.put("NTermKeyword", "\\$NTERM");
    exprs.put("TermKeyword", "\\$TERM");
    exprs.put("RuleKeyword", "\\$RULE");
    exprs.put("EpsKeyword", "\\$EPS");
    exprs.put("Nterm",
    "[A-Z][A-Z_]*'|[A-Z][A-Z_]*[0-9]|[A-Z][A-Z_]*");
    exprs.put("Term", "\"[a-zA-Z\\+\\*\\(\\)]+\"");
    exprs.put("Equal", "=");
    exprs.put("NewLine", "\\n+");
    return exprs;
}

private static HashMap<String, Rules> getGrammar() {
    HashMap<String, Rules> rules = new HashMap<>();
    rules.put("S", new Rules(new RHS(
            new Term("AxiomKeyword"),
            new Term("Nterm"),
            new Term("NTermKeyword"),
            new Term("Nterm"),
            new Nonterm("NTERMS"),
            new Nonterm("TERMS_DEF"),
            new Nonterm("RULES_DEF")
    )));
    rules.put("NTERMS",
            new Rules(new RHS(
                    new Term("Nterm"),
                    new Nonterm("NTERMS")
            ),
                    new Epsilon())));

    rules.put("TERMS_DEF", new Rules(new RHS(
            new Term("TermKeyword"),
            new Term("Term"),
```

```
        new Nonterm("TERMS")
)));
rules.put("TERMS", new Rules(
        new RHS(
                new Term("Term"),
                new Nonterm("TERMS")
        ),
        new Epsilon()
));
rules.put("RULES_DEF", new Rules(new RHS(
        new Nonterm("RULE"),
        new Nonterm("RULES")
)));
rules.put("RULE", new Rules(new RHS(
        new Term("RuleKeyword"),
        new Term("Nterm"),
        new Term("Equal"),
        new Nonterm("R")
)));
rules.put("RULES", new Rules(new RHS(
        new Nonterm("RULE"),
        new Nonterm("RULES")
),
        new Epsilon()
));
rules.put("R", new Rules(
        new RHS(
                new Nonterm("R1"),
                new Nonterm("R2")
        )
));
rules.put("R1", new Rules(
        new RHS(
                new Term("Term"),
                new Nonterm("R3")
        ),
        new RHS(
                new Term("Nterm"),
                new Nonterm("R3")
        ),
        new RHS(
                new Term("EpsKeyword")
        )
));
rules.put("R3", new Rules(
        new RHS(
```

```
                        new Term("Term"),
                        new Nonterm("R3")
                ),
                new RHS(
                        new Term("Nterm"),
                        new Nonterm("R3")
                ),
                new Epsilon()
        ));
        rules.put("R2", new Rules(
                new RHS(
                        new Term("NewLine"),
                        new Nonterm("R")
                ),
                new Epsilon()
        ));

        return rules;
    }

    private static RHS[][] analiseTable() {
        ArrayList<String> T = termsList;
        ArrayList<String> N = nontermsList;
        HashMap<String, Rules> rules = grammarList;
        int m = N.size();
        int n = T.size();
        RHS[][] q = new RHS[m][n];
        for (RHS[] line: q) {
            Arrays.fill(line, new Error());
        }

        q[N.indexOf("S")][T.indexOf("AxiomKeyword")] =
                rules.get("S").get(0);

        q[N.indexOf("NTERMS")][T.indexOf("TermKeyword")] =
                rules.get("NTERMS").get(1);
        q[N.indexOf("NTERMS")][T.indexOf("Nterm")] =
                rules.get("NTERMS").get(0);

        q[N.indexOf("TERMS_DEF")][T.indexOf("TermKeyword")] =
                rules.get("TERMS_DEF").get(0);

        q[N.indexOf("TERMS")][T.indexOf("Term")] =
                rules.get("TERMS").get(0);
        q[N.indexOf("TERMS")][T.indexOf("RuleKeyword")] =
                rules.get("TERMS").get(1);
```

```
        q[N.indexOf("RULES_DEF")][T.indexOf("RuleKeyword")] =
                rules.get("RULES_DEF").get(0);

        q[N.indexOf("RULE")][T.indexOf("RuleKeyword")] =
                rules.get("RULE").get(0);

        q[N.indexOf("RULES")][T.indexOf("RuleKeyword")] =
                rules.get("RULES").get(0);
        q[N.indexOf("RULES")][T.indexOf(Term.EOF)] =
                rules.get("RULES").get(1);

        q[N.indexOf("R")][T.indexOf("Nterm")] =
                rules.get("R").get(0);
        q[N.indexOf("R")][T.indexOf("EpsKeyword")] =
                rules.get("R").get(0);
        q[N.indexOf("R")][T.indexOf("Term")] =
                rules.get("R").get(0);

        q[N.indexOf("R1")][T.indexOf("EpsKeyword")] =
                rules.get("R1").get(2);
        q[N.indexOf("R1")][T.indexOf("Nterm")] =
                rules.get("R1").get(1);
        q[N.indexOf("R1")][T.indexOf("Term")] =
                rules.get("R1").get(0);

        q[N.indexOf("R2")][T.indexOf("RuleKeyword")] =
                rules.get("R2").get(1);
        q[N.indexOf("R2")][T.indexOf("NewLine")] =
                rules.get("R2").get(0);
        q[N.indexOf("R2")][T.indexOf(Term.EOF)] =
                rules.get("R2").get(1);

        q[N.indexOf("R3")][T.indexOf("NewLine")] =
                rules.get("R3").get(2);
        q[N.indexOf("R3")][T.indexOf(Term.EOF)] =
                rules.get("R3").get(2);
        q[N.indexOf("R3")][T.indexOf("RuleKeyword")] =
                rules.get("R3").get(2);
        q[N.indexOf("R3")][T.indexOf("Term")] =
                rules.get("R3").get(0);
        q[N.indexOf("R3")][T.indexOf("Nterm")] =
                rules.get("R3").get(1);

        return q;
    }
```

```
}
```

Nonterm.java

```
class Nonterm extends Symbol {

    public Nonterm(String type) {
        super.name = type;
    }

    @Override
    public String toString() {
        return super.toString();
    }

    public String toDot() {
        return String.format("[label=\"%s\"][color=black]\n",
                name);
    }

}
```

Parser.java

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.Stack;

public class Parser {

    private final ArrayList<String> terms;
    private final ArrayList<String> nonterms;
    private final Nonterm axiom;
    private final RHS[][] table;
    private ParseTree parseTree = null;

    public Parser(ArrayList<String> terms, ArrayList<String>
            nonterms, Nonterm axiom, RHS[][] table) {
        this.terms    = terms;
        this.nonterms = nonterms;
        this.axiom    = axiom;
        this.table = table;
    }

    private RHS delta(Nonterm N, Term T) {
        return
```

```
    table[nonterms.indexOf(N.name)][terms.indexOf(T.name)];
}

private void printError(Token tok, Symbol expected) {
    System.out.println("ERROR: " + expected.toString() +
            " expected, got: " + tok.toString());
}

public ParseTree topDownParser(Scanner scanner) {
    Stack<Symbol> stack = new Stack<>();
    stack.push(new Term(Term.EOF));
    stack.push(axiom);
    parseTree = new ParseTree(axiom);
    Token tok = scanner.nextToken();
    do {
        Symbol X = stack.pop();
        if (X instanceof Term) {
            if (X.equals(tok)) {
                parseTree.setToken(tok);
                tok = scanner.nextToken();
            } else {
                printError(tok, X);
                return parseTree;
            }
        } else {
            RHS nextRule = delta((Nonterm)X, tok);
            if (nextRule instanceof Error) {
                printError(tok, X);
                return parseTree;
            } else {
                stack.addAll(nextRule.reverse());
                parseTree.add(nextRule);
            }
        }
    } while (!stack.empty());
    return parseTree;
}

public ParseTree getParseTree() {
    return parseTree;
}

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

    Scanner sc = new Scanner(args[0],
```

```
                GrammarStructure.tagsRegex);
        Parser parser = new GrammarParser();
        parser.topDownParser(sc);
        File dotfile = new File("graph.dot");
        Files.write(dotfile.toPath(),
                parser.getParseTree().toDot().getBytes());

        System.out.println(parser.getParseTree().toDot());
        for (Token tok = sc.nextToken();
             !tok.name.equals(Term.EOF); tok =
                     sc.nextToken()) {
            System.out.println(tok.toString());
        }

        System.out.println("COMMENTS");
        for(Fragment comm : sc.getComments()){
            System.out.print(comm.toString());
        }

    }
}
```

ParseTree.java

```
import javax.swing.event.TreeModelListener;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
import java.util.ArrayList;
import java.util.Enumeration;

class ParseNode implements TreeNode {
    private Symbol symb;
    private int number = 0;
    private final ArrayList<ParseNode> children =
            new ArrayList<>();
    private ParseNode parent = null;

    public void setNumber(int number) {
        this.number = number;
    }

    public ParseNode(Symbol symb) {
        this.symb = symb;
    }

    public ParseNode(Symbol symb, ParseNode parent) {
```

```
    this.symb = symb;
    this.parent = parent;
}

public void setToken(Token token) {
    symb = token;
}

public void addChildren(RHS nodes) {
    for (Symbol s: nodes) {
        children.add(new ParseNode(s, this));
    }
}

private boolean isTheMostRightChild() {
    return (parent.getIndex(this) ==
            (parent.children.size() - 1));
}

private boolean isRoot () {
    return parent == null;
}

public ParseNode successor() {
    if (isRoot()) return this;
    ParseNode res = this.parent;
    ParseNode prev = this;
    while(!res.isRoot() && prev.isTheMostRightChild()) {
        prev = res;
        res = res.parent;
    }
    if (res.isRoot() && prev.isTheMostRightChild()) {
        return this;
    }
    res = res.children.get(res.getIndex(prev) + 1);

    while (!res.isLeaf()) {
        res = res.children.get(0);
    }
    return res;
}

@Override
public TreeNode getChildAt(int childIndex) {
    return children.get(childIndex);
}
```

```
    @Override
    public int getChildCount() {
        return children.size();
    }

    @Override
    public TreeNode getParent() {
        return parent;
    }

    @Override
    public int getIndex(TreeNode node) {
        if (node instanceof ParseNode) {
            return children.indexOf(node);
        }
        else return -1;
    }

    @Override
    public boolean getAllowsChildren() {
        return !(symb instanceof Token);
    }

    @Override
    public boolean isLeaf() {
        return children.isEmpty();
    }

    @Override
    public Enumeration children() {
        return (Enumeration)children;
    }

    public String toDot() {
        StringBuilder res = new StringBuilder(number +
                " " + symb.toDot());
        for (ParseNode child: children) {
            res.append(child.toDot());
            res.append(number).append("->").append
                    (child.number).append("\n");
        }
        return res.toString();
    }
}
```

```
class ParseTree implements TreeModel {
    private final ParseNode root;
    private ParseNode current;
    private int currentNumber;

    public ParseTree(Nonterm axiom) {
        root = new ParseNode(axiom);
        current = root;
        currentNumber = 0;
        update();
    }

    private void update() {
        ++currentNumber;
        current.setNumber(currentNumber);
    }

    public void add(RHS rule) {
        if (!rule.isEmpty()) {
            current.addChildren(rule);
            current = (ParseNode)current.getChildAt(0);
        } else {
            current = current.successor();
        }
        update();
    }

    public void setToken(Token token) {
        current.setToken(token);
        current = current.successor();
        update();
    }

    public String toDot() {
        return "digraph {\n" + root.toDot() + "}\n";
    }

    @Override
    public Object getRoot()
    {
        return root;
    }

    public Object getChild(Object parent, int index)
    {
        return ((ParseNode)parent).getChildAt(index);
```

```
    }

    @Override
    public int getChildCount(Object parent) {
        return ((ParseNode)parent).getChildCount();
    }

    @Override
    public boolean isLeaf(Object node) {
        return ((ParseNode)node).isLeaf();
    }

    @Override
    public int getIndexOfChild(Object parent, Object child) {
        return ((ParseNode)parent).getIndex((TreeNode)child);
    }

    @Override
    public void valueForPathChanged(TreePath path,
                                    Object newValue) {

    }

    @Override
    public void addTreeModelListener(TreeModelListener l) {

    }

    @Override
    public void removeTreeModelListener(TreeModelListener l)
    {

    }
}
```

Position.java

```
public class Position {
    private final int row, column, pos;

    public int getPos() {
        return pos;
    }

    private Position(int row, int column, int pos) {
        this.row = row;
        this.column = column;
```

```
        this.pos = pos;
    }

    public static Position undefined() {
        return new Position(-1, -1, -1);
    }

    public static Position start() {
        return new Position(1, 1, 0);
    }

    public Position shift(int positions) {
        return new Position(row, column + positions,
                pos + positions);
    }

    public Position newline() {
        return new Position(row + 1, 1, pos + 1);
    }

    @Override
    public String toString() {
        return pos > -1 ? String.format("(%d, %d)",
                row, column) : "?";
    }

}
```

RHS.java

```
import java.util.ArrayList;
import java.util.Arrays;

class RHS extends ArrayList<Symbol> {
    public RHS(Symbol... symbols) {
        super(Arrays.asList(symbols));
    }

    public RHS() {
        super();
    }

    public RHS reverse() {
        RHS rule = new RHS();
        rule.ensureCapacity(this.size());
        for (int i = size() - 1; i >= 0; --i) {
            rule.add(get(i));
```

```
        }
        return rule;
    }
}

class Error extends RHS {
    @Override
    public String toString() {
        return "Error";
    }
}

class Epsilon extends RHS {

}

class Rules extends ArrayList<RHS> {
    public Rules(RHS... rules) {
        super(Arrays.asList(rules));
    }
}
```

Scanner.java

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Scanner {

    private HashMap<String, String> regexp = new HashMap<>();

    private String text = "";
    private Pattern p;
    private Matcher m;
    public Position coord;
    private String image = "";

    private int index;
    private ArrayList<Fragment> comments  = new ArrayList<>();
```

```
private ArrayList<Token> tokens_list  = new ArrayList<>();

public static String makeGroup(String name, String expr) {
    return "(?<" + name + ">(" + expr + "))";
}

public String setPattern() {
    StringBuilder res = new StringBuilder(makeGroup("tab",
            "[ \\t]+") + "|" + makeGroup("comments",
            "\\*[^\\*\\n]*\\n"));
    for (Map.Entry<String, String> e: regexp.entrySet()) {
        res.append("|").append(makeGroup(e.getKey(),
                e.getValue()));
    }
    return res.toString();
}
public Scanner(String filepath, HashMap<String,
        String> termsexpr) {
    File file = new File(filepath);
    try {
        text = new
                String(Files.readAllBytes(file.toPath()));
    } catch (IOException e) {
        System.err.printf("file %s cannot be read\n",
                file.toPath());
    }
    regexp = termsexpr;
    String pattern = setPattern();
    p = Pattern.compile(pattern, Pattern.DOTALL);
    m = p.matcher(text);
    coord = Position.start();
    Token t = getNextToken();
    tokens_list.add(t);
    while (!Objects.equals(t.name, "$")) {
        t = getNextToken();
        tokens_list.add(t);
    }
    index = 0;
}

private boolean isType(String type) {
    return (image = m.group(type)) != null;
}

private Token returnToken (String type) {
    Position last = coord;
```

```
    if (Objects.equals(type, "NewLine")) {
        for (int i = 0; i < image.length(); i++) {
            coord = coord.newline();
        }
    } else {
        coord = coord.shift(image.length());
    }
    return new Token(type, image, last, coord);
}

public Token getNextToken() {
    if (coord.getPos() >= text.length()) {
        return new Token(Term.EOF, coord);
    }
    String image;
    if (m.find()) {
        if (m.start() != coord.getPos()) {
            System.out.println
                    (String.format("ERROR: SYNTAX %d - %d",
                            m.start(), coord.getPos()) +
                            coord.toString());
            coord = coord.shift(m.start() - coord.getPos());
        }
        for (String s: regexp.keySet()) {
            if (isType(s)) {
                return returnToken(s);
            }
        }
        if ((image = m.group("tab")) != null) {
            coord = coord.shift(image.length());
            return getNextToken();
        }
        if ((image = m.group("comments")) != null) {
            Position last = coord;
            coord = coord.shift(image.length() - 1);
            comments.add(new Fragment(image, last, coord));
            coord = coord.newline();
            return getNextToken();
        } else {
            System.out.println("ERROR: " +
                    coord.toString() + " " +
                    text.substring(coord.getPos()));
            return getNextToken();
        }
    } else {
        return new Token(Term.EOF, coord);
```

```
        }
    }

    public Token nextToken() {
        if (index < tokens_list.size() - 1) {
            if ((Objects.equals(tokens_list.get
                    (index).name, "NewLine"))
                    && (Objects.equals(tokens_list.get
                    (index + 1).name, "AxiomKeyword")
                        || Objects.equals(tokens_list.get
                    (index + 1).name, "NTermKeyword")
                        || Objects.equals(tokens_list.get
                    (index + 1).name, "TermKeyword")
                        || Objects.equals(tokens_list.get
                    (index + 1).name, "RuleKeyword"))) {
                index++;
            }
            index++;
            return tokens_list.get(index - 1);
        } else if ((index == (tokens_list.size() - 1)) &&
                (!(Objects.equals(tokens_list.get(index).name,
                        "NewLine")))) {
            index++;
            return tokens_list.get(index - 1);
        } else {
            return new Token(Term.EOF, coord);
        }
    }

    public ArrayList<Fragment> getComments() {
        return this.comments;
    }
}
```

Symbol.java

```
class Symbol {

    protected String name;

    @Override
    public String toString() {
        return name;
    }

    @Override
    public boolean equals (Object o) {
```

```
        return ((o instanceof Symbol) &&
                name.equals(((Symbol)o).name))
                || ((o instanceof String) &&
                name.equals(o));
    }

    public String toDot() {
        return "";
    }
}

class Term extends Symbol {
    final static String EOF = "$";

    public Term(String type) {
        super.name = type;
    }

    public String toDot() {
        return String.format
                ("[label=\"%s\"][color=black]\n", name);
    }

}
```

Token.java

```
public class Token extends Term {
    public String image;
    private Position start, follow;

    public Token(String type, String image, Position start,
                 Position follow) {
        super(type);
        this.image = image;
        this.start = start;
        this.follow = follow;
    }

    public Token(String type, Position start) {
        super(type);
        this.image = "";
        this.start = start;
        this.follow = start;
    }

    @Override
```

```
    public String toString(){
        return String.format("Token %s %s-%s %s",
                super.toString(), start.toString(),
                follow.toString(), image);
    }

    @Override
    public String toDot() {
        return String.format("[label=\"%s\"][color=red]\n",
                toString().replaceAll("\"", "\\\\\""));
    }

}
```

Тестирование

Входные данные

```
$AXIOM E
$NTERM E' T T' F
$TERM  "+"  "*"  "("  ")"  "n"

* правила грамматики
$RULE  E  = T E'
$RULE  E' = "+" T E'
            $EPS
$RULE  T  = F T'
$RULE  T' = "*" F T'
            $EPS
$RULE  F  = "n"
            "(" E ")"
```

Вывод на `stdout`

```
digraph {
1 [label="S"][color=black]
2 [label="Token AxiomKeyword (1, 1)-(1, 7) $AXIOM"][color=red]
1->2
3 [label="Token Nterm (1, 8)-(1, 9) E"][color=red]
1->3
4 [label="Token NTermKeyword (2, 1)-(2, 7) $NTERM"][color=red]
1->4
5 [label="Token Nterm (2, 8)-(2, 10) E'"][color=red]
1->5
6 [label="NTERMS"][color=black]
7 [label="Token Nterm (2, 11)-(2, 12) T"][color=red]
```

```
6->7
8 [label="NTERMS"][color=black]
9 [label="Token Nterm (2, 13)-(2, 15) T'"][color=red]
8->9
10 [label="NTERMS"][color=black]
11 [label="Token Nterm (2, 16)-(2, 17) F"][color=red]
10->11
12 [label="NTERMS"][color=black]
10->12
8->10
6->8
1->6
13 [label="TERMS_DEF"][color=black]
14 [label="Token TermKeyword (3, 1)-(3, 6) $TERM"][color=red]
13->14
15 [label="Token Term (3, 8)-(3, 11) \"+\""][color=red]
13->15
16 [label="TERMS"][color=black]
17 [label="Token Term (3, 13)-(3, 16) \"*\""][color=red]
16->17
18 [label="TERMS"][color=black]
19 [label="Token Term (3, 18)-(3, 21) \"(\""][color=red]
18->19
20 [label="TERMS"][color=black]
21 [label="Token Term (3, 23)-(3, 26) \")\""][color=red]
20->21
22 [label="TERMS"][color=black]
23 [label="Token Term (3, 28)-(3, 31) \"n\""][color=red]
22->23
24 [label="TERMS"][color=black]
22->24
20->22
18->20
16->18
13->16
1->13
25 [label="RULES_DEF"][color=black]
26 [label="RULE"][color=black]
27 [label="Token RuleKeyword (6, 1)-(6, 6) $RULE"][color=red]
26->27
28 [label="Token Nterm (6, 8)-(6, 9) E"][color=red]
26->28
29 [label="Token Equal (6, 11)-(6, 12) ="][color=red]
26->29
30 [label="R"][color=black]
31 [label="R1"][color=black]
```

```
32 [label="Token Nterm (6, 13)-(6, 14) T"][color=red]
31->32
33 [label="R3"][color=black]
34 [label="Token Nterm (6, 15)-(6, 17) E'"][color=red]
33->34
35 [label="R3"][color=black]
33->35
31->33
30->31
36 [label="R2"][color=black]
30->36
26->30
25->26
37 [label="RULES"][color=black]
38 [label="RULE"][color=black]
39 [label="Token RuleKeyword (7, 1)-(7, 6) $RULE"][color=red]
38->39
40 [label="Token Nterm (7, 8)-(7, 10) E'"][color=red]
38->40
41 [label="Token Equal (7, 11)-(7, 12) ="][color=red]
38->41
42 [label="R"][color=black]
43 [label="R1"][color=black]
44 [label="Token Term (7, 13)-(7, 16) \"+\""][color=red]
43->44
45 [label="R3"][color=black]
46 [label="Token Nterm (7, 17)-(7, 18) T"][color=red]
45->46
47 [label="R3"][color=black]
48 [label="Token Nterm (7, 19)-(7, 21) E'"][color=red]
47->48
49 [label="R3"][color=black]
47->49
45->47
43->45
42->43
50 [label="R2"][color=black]
51 [label="Token NewLine (7, 21)-(8, 1)
"][color=red]
50->51
52 [label="R"][color=black]
53 [label="R1"][color=black]
54 [label="Token EpsKeyword (8, 13)-(8, 17) $EPS"][color=red]
53->54
52->53
55 [label="R2"][color=black]
```

```
52->55
50->52
42->50
38->42
37->38
56 [label="RULES"][color=black]
57 [label="RULE"][color=black]
58 [label="Token RuleKeyword (9, 1)-(9, 6) $RULE"][color=red]
57->58
59 [label="Token Nterm (9, 8)-(9, 9) T"][color=red]
57->59
60 [label="Token Equal (9, 11)-(9, 12) ="][color=red]
57->60
61 [label="R"][color=black]
62 [label="R1"][color=black]
63 [label="Token Nterm (9, 13)-(9, 14) F"][color=red]
62->63
64 [label="R3"][color=black]
65 [label="Token Nterm (9, 15)-(9, 17) T'"][color=red]
64->65
66 [label="R3"][color=black]
64->66
62->64
61->62
67 [label="R2"][color=black]
61->67
57->61
56->57
68 [label="RULES"][color=black]
69 [label="RULE"][color=black]
70 [label="Token RuleKeyword (10, 1)-(10, 6) $RULE"][color=red]
69->70
71 [label="Token Nterm (10, 8)-(10, 10) T'"][color=red]
69->71
72 [label="Token Equal (10, 11)-(10, 12) ="][color=red]
69->72
73 [label="R"][color=black]
74 [label="R1"][color=black]
75 [label="Token Term (10, 13)-(10, 16) \"*\""][color=red]
74->75
76 [label="R3"][color=black]
77 [label="Token Nterm (10, 17)-(10, 18) F"][color=red]
76->77
78 [label="R3"][color=black]
79 [label="Token Nterm (10, 19)-(10, 21) T'"][color=red]
78->79
```

```
80 [label="R3"][color=black]
78->80
76->78
74->76
73->74
81 [label="R2"][color=black]
82 [label="Token NewLine (10, 21)-(11, 1)
"][color=red]
81->82
83 [label="R"][color=black]
84 [label="R1"][color=black]
85 [label="Token EpsKeyword (11, 13)-(11, 17) $EPS"][color=red]
84->85
83->84
86 [label="R2"][color=black]
83->86
81->83
73->81
69->73
68->69
87 [label="RULES"][color=black]
88 [label="RULE"][color=black]
89 [label="Token RuleKeyword (12, 1)-(12, 6) $RULE"][color=red]
88->89
90 [label="Token Nterm (12, 8)-(12, 9) F"][color=red]
88->90
91 [label="Token Equal (12, 11)-(12, 12) ="][color=red]
88->91
92 [label="R"][color=black]
93 [label="R1"][color=black]
94 [label="Token Term (12, 13)-(12, 16) \"n\""][color=red]
93->94
95 [label="R3"][color=black]
93->95
92->93
96 [label="R2"][color=black]
97 [label="Token NewLine (12, 16)-(13, 1)
"][color=red]
96->97
98 [label="R"][color=black]
99 [label="R1"][color=black]
100 [label="Token Term (13, 13)-(13, 16) \"(\""][color=red]
99->100
101 [label="R3"][color=black]
102 [label="Token Nterm (13, 17)-(13, 18) E"][color=red]
101->102
```

```
103 [label="R3"][color=black]
104 [label="Token Term (13, 19)-(13, 22) \")\""][color=red]
103->104
105 [label="R3"][color=black]
103->105
101->103
99->101
98->99
106 [label="R2"][color=black]
98->106
96->98
92->96
88->92
87->88
109 [label="Token $ (13, 22)-(13, 22) "][color=red]
87->109
68->87
56->68
37->56
25->37
1->25
}

COMMENTS
COMMENT (5, 1)-(5, 21) * правила грамматики
```

Вывод

В результате выполнения лабораторной работы был изучен алгоритм построения таблиц предсказывающего анализатора. Было составлено описание лексической структуры и грамматики входного языка на основе примера из индивидуального варианта, разработан лексический анализатор для входного языка, составлена (вручную) таблица предсказывающего разбора для входного языка, разработан алгоритм предсказывающего разбора, работающий на основе порождённой таблицы.