



**合肥工业大学**  
HEFEI UNIVERSITY OF TECHNOLOGY

## **编译原理实验报告**

学生姓名：林天岳

学号：2017217893

班级：计算机科学与技术 2017-5

完成日期：2019 年 10 月 22 日

# 实验 1：词法分析设计

## 1.数据结构及算法描述

```
1. String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"; //字母
2. String number = "0123456789"; //数字
3. String keyword[] = {"auto", "break", "case", "char", "const", "continue", "default", "do",
4. "double", "else", "enum", "extern", "float", "for", "goto", "if", "int",
5. "long", "register", "return", "short", "signed", "sizeof", "static", "struct", "switch",
6. "typedef", "unsigned", "union", "void", "volatile", "while"}; //关键字
7. String operator[] = {"<=", ">=", "&&", "||", "<", "=", "*", "^", "=", "+", "--", "/", "-",
8. "=", "+=", "%=", "!=", ">=", "[", "]", "!", "%", "(", ")", "*", "+", "-", "/", "<", "=", ">"}; //运算符
9. String arithmeticOperator[] = {"+", "--", "+", "-", "*", "/", "%"}; //算术运算符
10. String relationalOperator[] = {"<=", "<", ">=", ">", "=", "!="}; //关系运算符
11. String logicalOperator[] = {"&&", "||", "!", "&"}; //逻辑运算符
12. String delimiter[] = {";", ",", "(", ")", "[", "]", " "}; //分界符
13. String assignmentOperator[] = {"=", "+=", "-=", "*=", "/=", "%=", "<=", ">=", "%=", "^=", "|="}; //赋值运算符
14. Map<String, String> opS; // <单个运算符, 种类名>
15. Map<String, String[]> KindToArray; // <种类名, 对应的运算符数组>
16.
17. List<Result> result = new ArrayList<>(); //结果 保存后显示为表格

1. GUI 包含一个 Solution
2. 分析时 在 textArea 中输入需要的分析的代码
3. 或者直接打开文件读取到 textArea
4.
5. 分析 则使用 Solution.Solve 返回分析结果
6. 显示在界面上
7. Solution 包含一个 Analyzer 分析器
8. 调用 Solve 方法 传入 String 数组 返回分析结果
9. for(String line:传入的 String 数组){
10.     result.addAll(Analyzer.LineAnalyse(line)); Analyzer.LineAnalyse(line)
11. }
12. return result
13. LineAnalyse 方法
14. if(当前分析的是 null, //, \n, 或者 Length == 0){
15.     则直接结束
16. }
17. else {
18.     if(字母表含有当前头部的 string){
19.         while(是字母或者是数字){
20.             继续取出之后的部分
21.         }
22.         得到了一个 String
23.         if(单词是关键字){
24.             标记为 关键字
25.         }
26.         else{
27.             标记为 标识符
28.         }
29.     }
30.     else if(数字表含有当前头部的 string){
31.         while(是数字或者是小数点){
32.             继续取出之后的部分
33.         }
34.         if(数字之后直接追加字母){
35.             标记错误
36.             break
37.         }
38.         标记为 常数 //常数的标记使用一个静态方法调用方法返回当前的数目+1 ERROR 使用同样的方法编号
39.     }
40.     else{
41.         if(匹配到了符号){ //运算符经过按照长度排序 确保长度较长的先匹配到 比如 ++ 会优先于+匹配
42.             标记 运算符
43.         }
44.         else{
45.             标记 错误
46.         }
47.     }
48.     递归处理之后的 String
49.     return result.addAll(递归的结果);
50. }
```

## 2.算法流程图

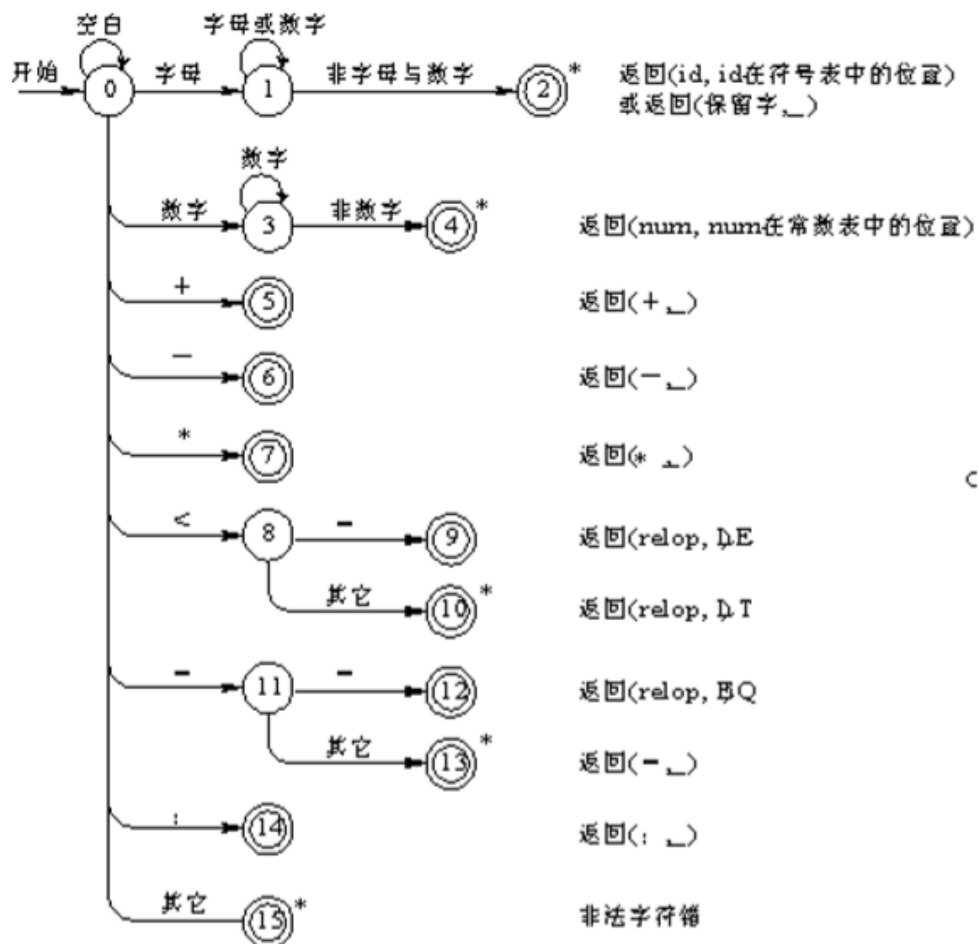


表1 关键字表

指针	关键字
0	do
1	end
2	for
3	if
4	printf
5	scanf
6	then
7	while

表2 分界符表

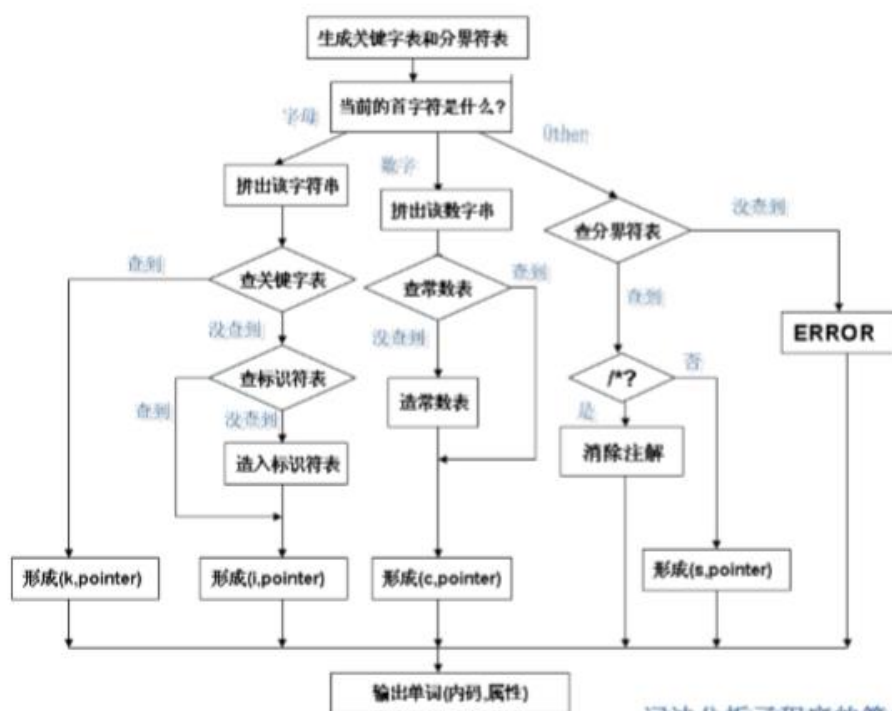
指针	分界符
0	,
1	;
2	(
3	)
4	[
5	]

表3 算术运算符

i值	算术运算符
10H	+
11H	-
20H	*
21H	/

表4 关系运算符

i值	关系运算符
00H	<
01H	<=
02H	=
03H	>
04H	>=
05H	<>



词法分析子程序的简化框图

### 3. 源码及测试结果

#### Main.java :

```
1. package 实验一——词法分析设计;
2.
3.
4. public class Main{
5.     public static void main(String[] args) {
6.         Windows windows = new Windows();
7.     }
8. }
```

#### Solution.java

```
1. package 实验一——词法分析设计;
2.
3. import java.util.*;
4.
5. class Solution{
6.     Analyzer ana = new Analyzer();
7.     public List<Result> Solve(String[] lines) {
8.         List<Result> res = new ArrayList<>();
9.         if(lines==null|| lines.length==0)
10.            return res;
11.         List<String> text = new ArrayList<>();
12.         for(String line:lines){
13.             line = line.replaceAll("\t"," ");
14.             if(line.length()>0)
15.                 text.add(line);
16.         }
17.         int l = 1;
18.         for(String str:text){
19.             if(str.length()>2 && str.substring(0,2).equals("//"))
20.                 continue;
21.             ana.result.clear();
22.             res.addAll(ana.LineAnalyse(str+"\n",l,1));
23.             l++;
24.         }
25.         return res;
26.     }
27.     public void manuallySetKP(String k[] , String p[] ){
28.         ana.setKP(k,p);
29.     }
30. }
31. class Analyzer{//在这里是用 C 的标准了
32.     String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";//字母
33.     String number = "0123456789";//数字
34.
35.     String keyword[] = {"auto","break","case","char","const ","continue","default","do ",
36.         "double ","else ","enum ","extern","float","for","goto","if","int",
37.         "long","register","return","short","signed","sizeof","static","struct","switch"
38.     },
39.         "typedef","unsigned","union","void","volatile","while";//关键字
40.     String operator[] = {"<=",">=","&&","||","<=","|=","*=","^=","=","+","-","/","=","-","
41.         "+","%","!=", ">=", "[", "]", "!", "%", "(", ")", "*", "+", "-", "/", "<", "=", ">";//运算符
42.     String arithmeticOperator[] = {"+", "-", "+", "-", "*", "/"};//算术运算符
43.     String relationalOperator[] = {"<=", "<", ">=", ">", "=", "!="};//关系运算符
44.     String logicalOperator[] = {"&&","||","!"};//逻辑运算符
45.     String delimiter[] = {";", ",", "(", ")", "[", ""];//分界符
46.     String assignmentOperator[] = {"=", "+=", "-=", "*=", "/=", "%=", "<=", ">=", "%=", "^=", "|="};//赋值运算符
47.
48.     Map<String,String> opS;//<单个运算符,种类名>
49.     Map<String,String[]> KindtoArray;//<种类名,对应的运算符数组>
50.
51.     List<Result> result = new ArrayList<>();
52.
53.     public Analyzer(){
54.         KindtoArray = new HashMap<>();
55.         KindtoArray.put("算术运算符",arithmeticOperator);
56.         KindtoArray.put("关系运算符",relationalOperator);
57.         KindtoArray.put("逻辑运算符",logicalOperator);
58.         KindtoArray.put("分界符",delimiter);
59.         KindtoArray.put("赋值运算符",assignmentOperator);
60.     }
```

```

58.         opS = new HashMap<>();
59.         KindtoArray.keySet().forEach(KindStr->Arrays.asList(KindtoArray.get(KindStr)).for
Each(str->opS.put(str,KindStr)));
60.     }
61.
62.     public void setKP(String k[] , String p[] ){
63.         this.keyword = k;
64.         this.operator = p;
65.     }
66.     public List<Result> LineAnalyse(String line,int L,int C){//当前行 行数 列数
67.         //System.out.print("当前分析 :"+line+" ");
68.         if(line == null || line.length()==0 || line.equals("\n") || (line.length())>=2 && li
ne.substring(0,2).equals("//")){
69.             return null;//行空 长度为0 回车 注释 行结束
70.         }
71.         if(line.substring(0,1).equals(" ")){//是空格 跳过当前单词
72.             LineAnalyse(line.substring(1),L,C);
73.             return result;
74.         }
75.         Result res = new Result();
76.         String head = line.substring(0,1);
77.         int i = 0;
78.         if(alphabet.contains(head)){//匹配到字母
79.             while(i!=line.length() && (alphabet+number).contains(line.substring(i,i+1))){
80.                 i++;
81.             }
82.             String wordGet = line.substring(0,i);
83.             Boolean ketWordMatch = false;
84.             int count = 0;
85.             for(String str:keyword){
86.                 if(wordGet.equals(str)){//是关键字
87.                     ketWordMatch = true;
88.                     res.setKind("关键字");
89.                     res.setSequence("(" + count + ", " + wordGet + ")");
90.                     break;
91.                 }
92.                 count++;
93.             }
94.             if(!ketWordMatch){//是标识符
95.                 res.setKind("标识符");
96.                 res.setSequence("(" + DataList.getID(wordGet) + ", " + wordGet + ")");
97.             }
98.             res.setWord(wordGet);
99.         }
100.         else if(number.contains(head)){//匹配到数字考虑小数，但小数不会以"."开头
101.             while (i!=line.length() && (number+ ".").contains(line.substring(i,i+1))){
102.                 i++;
103.             }
104.             if(alphabet.contains(line.substring(i,i+1))){//数字之后直接追加字母 非法输入
105.                 while(i!=line.length() && (alphabet+number).contains(line.substring(i,i+1)
)){
106.                     i++;
107.                 }
108.                 res.setWord(line.substring(0,i+1));
109.                 res.setKind("ERROR");
110.                 res.setSequence("ERROR"+DataList.getERROR(line.substring(0,i+1)));
111.             }
112.             else{
113.                 String number = line.substring(0,i);
114.                 res.setWord(number);
115.                 res.setKind("常数");
116.                 res.setSequence("(" + DataList.getCI(line.substring(0,i)) + ", " + line.substring
(0,i) + ")");
117.             }
118.         }
119.         }
120.         else{
121.             Boolean match = false;
122.             for(String str:operator){//用运算符来匹配而不是去匹配运算符 避免 ++ 匹配出 +*2
123.                 if(str.length() > line.length()){
124.                     continue;//符号是在尾部 且不会匹配成功 则直接跳过
125.                     //System.out.println(str+"匹配"+line.substring(0,str.length()));
126.                     if(str.equals( line.substring(0,str.length()) )){//是运算符
127.                         res.setWord(str);
128.                         res.setKind(opS.get(str));
129.                         int count = Arrays.asList(KindtoArray.get(opS.get(str))).indexOf(str);
130.                         res.setSequence("(" + count + ", " + str + ")");
131.                         match = true;
132.                         i+=str.length();
133.                         break;
134.                     }
135.                 }
136.             }
137.             if(!match){//没有匹配到
138.                 res.setWord(line.substring(0,1));
139.                 res.setKind("ERROR");

```

```

139.         res.setSequence("ERROR"+DataList.getERROR(line.substring(0,1)));
140.         i++;
141.     }
142. }
143.     line = line.substring(i);
144.     res.setLocation("(" + L + ", " + C + ")");
145.     result.add(res);
146.     LineAnalyse(line, L, ++C);
147.     return result;
148. }
149. }
150. class Result{
151.     private String word; //单词
152.     private String binarySequence; //二原序列
153.     private String kind; //类型
154.     private String location; //位置
155.     public Result(){
156.         this.word = "Null";
157.         this.binarySequence = "Null";
158.         this.kind = "Null";
159.         this.location = "Null";
160.     }
161.     public void setWord(String word){
162.         this.word = word;
163.     }
164.     public void setSequence(String Sequence){
165.         this.binarySequence = Sequence;
166.     }
167.     public void setKind(String kind){
168.         this.kind = kind;
169.     }
170.     public void setLocation(String location){
171.         this.location = location;
172.     }
173.     public String[] toStringArray(){
174.         String strS[] = {" " + word, " " + binarySequence, " " + kind, " " + location};
175.         return strS;
176.     }
177.     @Override
178.     public String toString(){
179.         String strS[] = {word, binarySequence, kind, location};
180.         StringBuffer toString = new StringBuffer();
181.         for(String str:strS){
182.             str = String.format("%-20s", str);
183.             toString.append(str);
184.         }
185.         return toString.toString();
186.     }
187. }
188. class DataList{
189.     static List<String> id = new ArrayList<>(), ci = new ArrayList<>(), ERROR = new ArrayList
    <>(); //标识符 常数
190.     public static int getID(String str){ //获取标识符位置 存在则返回地址 不存在则存入 返回最后位
    置
191.         if(id.contains(str)){
192.             return id.indexOf(str);
193.         }
194.         else{
195.             id.add(str);
196.             return id.size()-1;
197.         }
198.     }
199.     public static int getCI(String str){ //获取常数位置
200.         if(ci.contains(str)){
201.             return ci.indexOf(str);
202.         }
203.         else{
204.             ci.add(str);
205.             return ci.size()-1;
206.         }
207.     }
208.     public static int getERROR(String str){ //获取错误代码
209.         if(ERROR.contains(str)){
210.             return ERROR.indexOf(str);
211.         }
212.         else{
213.             ERROR.add(str);
214.             return ERROR.size()-1;
215.         }
216.     }
217. }

```

GUI.java

```

1. package 实验一——词法分析设计;
2.
3. import javax.swing.*;
4. import javax.swing.table.AbstractTableModel;
5. import java.awt.*;
6. import java.io.BufferedReader;
7. import java.io.File;
8. import java.io.FileReader;
9. import java.io.IOException;
10. import java.awt.event.ActionEvent;
11. import java.awt.event.ActionListener;
12. import java.util.List;
13. import java.util.Vector;
14.
15. class Windows extends JFrame{
16.     JMenuBar bar;
17.     JMenu menu;
18.     JMenuItem file;
19.     JMenuItem manuallySet;
20.     JMenuItem exit;
21.     JTextArea TA;
22.     JButton clear;
23.     JButton analyse;
24.     String[] text;
25.     JTable table;
26.     Vector<String[]> vecRes = new Vector<>();
27.     TableDataModel tableDataModel;
28.     JScrollPane restableScrollPane;
29.     Solution sl = new Solution();
30.     public Windows(){
31.         try{
32.             setIconImage(new ImageIcon("bilibili.PNG").getImage());
33.             Font f = new Font("Yahei Consolas Hybrid",Font.PLAIN,16);
34.             String names[]={ "MenuBar","Menu","MenuItem", "TextArea", "Button", "ScrollPa
ne", "Table"};
35.             for (String item : names) {
36.                 UIManager.put(item+ ".font",f);
37.             }
38.             UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
39.         }catch(Exception e){}
40.         init();
41.
42.         setSize(600,800); //初始大小
43.         setLocation(640,100); //初始位置
44.         setVisible(true); //是否可视
45.         setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //X 退出
46.     }
47.     public void init(){
48.         setTitle("词法分析器");
49.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50.         setVisible(true);
51.         setResizable(false);
52.         setLayout(null);
53.         setBounds(10, 10, 300, 400);
54.         initMenu(); //初始化菜单
55.         initTextArea(); //初始化输入文本
56.         initButton(); //初始化按钮
57.         initResultTable(); //初始化结果区域
58.     }
59.     private void initMenu(){
60.         class fileListen implements ActionListener{
61.             @Override
62.             public void actionPerformed(ActionEvent e){
63.                 JFileChooser fileChooser = new JFileChooser("D:\\工作\\programs");
64.                 fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
65.                 fileChooser.showOpenDialog(null);
66.                 File file = fileChooser.getSelectedFile();
67.                 if(file!=null){
68.                     try{
69.                         BufferedReader read = new BufferedReader(new FileReader( file ));
70.                         Object[] lines = read.lines().toArray();
71.                         StringBuffer bufferTA = new StringBuffer();
72.                         for(Object line:lines){
73.                             bufferTA.append(line.toString()+"\n");
74.                         }
75.                         TA.setText(bufferTA.toString());
76.                     }
77.                     catch (IOException err){
78.                     }
79.                 }
80.             }
81.         }
82.         class manuallySet implements ActionListener {
83.             @Override
84.             public void actionPerformed(ActionEvent e) {
85.                 String [] k ={};

```



```

86.         String [] p ={};
87.         String kString,pString;
88.         Boolean changed = true;
89.         do{
90.             kString = JOptionPane.showInputDialog(null,"请输入 K[ ]: \n","自定义
K,P",JOptionPane.PLAIN_MESSAGE);
91.             if(kString==null){
92.                 changed = false;
93.                 break;
94.             }
95.         }while (kString.length()<2);
96.         if(changed){
97.             boolean allchanged = true;
98.             do{
99.                 pString = JOptionPane.showInputDialog(null,"请输入 P[ ]: \n","自定义
K,P",JOptionPane.PLAIN_MESSAGE);
100.                 if(pString == null){
101.                     allchanged = false;
102.                     break;
103.                 }
104.             }while (pString.length()<2);
105.             if(allchanged){
106.                 kString = kString.substring(1,kString.length()-1);
107.                 pString = pString.substring(1,pString.length()-1);
108.                 k = kString.split(" ");
109.                 p = pString.split(" ");
110.                 if(k.length<2 || p.length<2){//如果输入不规范 警告 不修改 kp
111.                     JOptionPane.showMessageDialog(null, "格式输入错误
", "Error !", JOptionPane.ERROR_MESSAGE);
112.                 }
113.                 else
114.                     sl.manullySetKP(k,p);
115.             }
116.         }
117.     }
118. }
119. class exitlisten implements ActionListener {
120.     @Override
121.     public void actionPerformed(ActionEvent e) {
122.         dispose();
123.     }
124. }
125.
126. bar = new JMenuBar();
127. setJMenuBar(bar);
128.
129. menu = new JMenu("选项");
130. bar.add(menu);
131.
132. file = new JMenuItem("选择文件");
133. file.addActionListener(new fileListen());//读取文件到 TA 里
134.
135. manuallySet = new JMenuItem("手动设定");
136. manuallySet.addActionListener(new manuallySet());
137.
138. exit = new JMenuItem("退出");
139. exit.addActionListener(new exitListen());
140.
141. menu.add(file);
142. menu.add(manuallySet);
143. menu.add(exit);
144. }
145. private void initTextArea(){
146.     TA = new JTextArea();
147.     JScrollPane SP = new JScrollPane(TA);
148.     TA.setLineWrap(true); // 设置自动换行
149.     SP.setBounds(10, 10, 565, 300);
150.     add(SP);
151. }
152. private void initButton(){
153.     class clearListen implements ActionListener{
154.         @Override
155.         public void actionPerformed(ActionEvent e){
156.             TA.setText("");
157.             vecRes.clear();
158.             table.validate();
159.             table.updateUI();
160.             restablescrollPane.updateUI();
161.         }
162.     }
163.     class analyselisten implements ActionListener{
164.         @Override
165.         public void actionPerformed(ActionEvent e){
166.             text = TA.getText().split("\n");//这样分割后的 String 没有\n
167.             //for(String str:text) System.out.println(str);
168.             vecRes.clear();
169.             List<Result> resS = sl.Solve(text);

```

```

170.         for(Result result:resS){
171.             vecRes.add(result.toStringArray());
172.         }
173.         //vecRes.forEach(Strings -> {for(String str:Strings) System.out.print(str+"
");System.out.println();});
174.         table.validate();
175.         table.updateUI();
176.         restablescrollPane.updateUI();
177.     }
178. }
179. clear = new JButton("清空");
180. clear.addActionListener(new clearListen());
181.
182. analyse = new JButton("分析");
183. analyse.addActionListener(new analyseListen());
184.
185. clear.setBounds(400,320,70,35);
186. analyse.setBounds(500,320,70,35);
187. add(clear);
188. add(analyse);
189. }
190. private void initResultTable(){
191.     tableDataModel = new TableDataModel(vecRes);
192.     table = new JTable(tableDataModel);
193.     table.setVisible(true);
194.     table.setPreferredScrollableViewportSize(new Dimension(550, 100));
195.     table.setRowHeight(24);
196.     restablescrollPane = new JScrollPane(table);
197.     restablescrollPane.setBounds(10, 367, 565, 363);
198.     add(restablescrollPane);
199.     pack();
200. }
201. }
202.
203. class TableDataModel extends AbstractTableModel{
204.     private Vector<String[]> TableData; //用来存放表格数据的线性表
205.     private Vector<String> TableTitle; //表格的 列标题
206.     public TableDataModel(Vector data){
207.         String Names[] = {"单词","二元序列","类 型","位置 (行, 列)"};
208.         Vector Namessss = new Vector();
209.         for(String str:Names){
210.             Namessss.add(str);
211.         }
212.         TableTitle = Namessss;
213.         TableData = data;
214.     }
215.
216.     @Override
217.     public int getRowCount(){
218.         return TableData.size();
219.     }
220.     public int getColumnCount(){
221.         return TableTitle.size();
222.     }
223.     @Override
224.     public String getColumnName(int colum){
225.         return TableTitle.get(colum);
226.     }
227.     public Object getValueAt(int rowIndex, int columnIndex){
228.         String LineTemp[] = this.TableData.get(rowIndex);
229.         return LineTemp[columnIndex];
230.     }
231.     @Override
232.     public boolean isCellEditable(int rowIndex, int columnIndex){ //不允许编辑
233.         return false;
234.     }
235. }

```

运行结果：

词法分析器

选项

```
#include <iostream>
#include <time.h>
#define true 1
#define false 0
#define MAXLENGTH 100
#define coefficient 0.9 //参数 决定了AI优先关注玩家不赢还是自己赢
using namespace std;

int weight_1(Board B, int i, int j, int kind) {
    char yes, no;
    int a, b, count, comprehensiveweight = 0;
    int weight;
    if (kind == 1) {
```

清空 分析

单词	二元序列	类 型	位置 (行, 列)
#	ERROR1	ERROR	(1,1)
include	(0,include)	标识符	(1,2)
<	(1,<)	关系运算符	(1,3)
iostream	(1,iostream)	标识符	(1,4)
>	(3,>)	关系运算符	(1,5)
#	ERROR1	ERROR	(2,1)
include	(0,include)	标识符	(2,2)
<	(1,<)	关系运算符	(2,3)
time	(2,time)	标识符	(2,4)
.	ERROR2	ERROR	(2,5)
h	(3,h)	标识符	(2,6)
>	(3,>)	关系运算符	(2,7)
#	ERROR1	ERROR	(3,1)
define	(4,define)	标识符	(3,2)

词法分析器

选项

```
#include <iostream>
#include <time.h>
#define true 1
#define false 0
#define MAXLENGTH 100
#define coefficient 0.9 //参数 决定了AI优先关注玩家不赢还是自己赢
using namespace std;

int weight_1(Board B, int i, int j, int kind) {
    char yes, no;
    int a, b, count, comprehensiveweight = 0;
    int weight;
    if (kind == 1) {
```

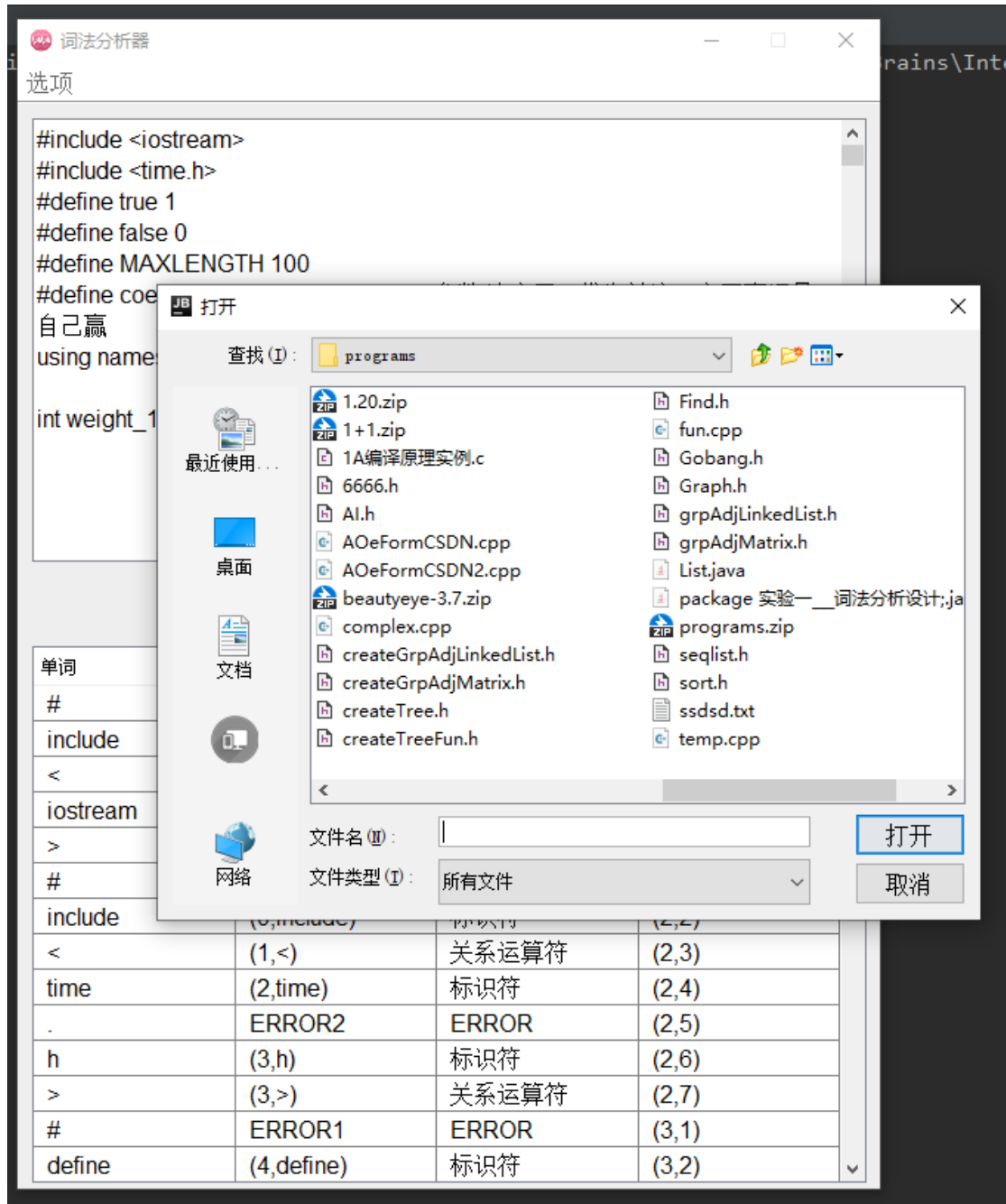
分析

自定义K,P

请输入K[]:

确定 取消

单词	二元序列		
#	ERROR1	ERROR	(1,1)
include	(0,include)	标识符	(1,2)
<	(1,<)	关系运算符	(1,3)
iostream	(1,iostream)	标识符	(1,4)
>	(3,>)	关系运算符	(1,5)
#	ERROR1	ERROR	(2,1)
include	(0,include)	标识符	(2,2)
<	(1,<)	关系运算符	(2,3)
time	(2,time)	标识符	(2,4)
.	ERROR2	ERROR	(2,5)
h	(3,h)	标识符	(2,6)
>	(3,>)	关系运算符	(2,7)
#	ERROR1	ERROR	(3,1)
define	(4,define)	标识符	(3,2)



## 4. 实验收获

本次试验算法部分较为简单，核心部分为递归行分析中的使用每个符号去匹配字符串的头部，根据匹配结果得出分析结果，然后将剩余的部分递归处理。大部分时间都用于学习设计界面 UI，初步掌握了 UI 的设计方法，有了一套自己的设计思路。

# 实验 2: LL(1)分析法

## 1. 数据结构及算法描述

```
2. Set<String> noTerminal = new HashSet<>(); //非终结符
3. Set<String> terminal = new HashSet<>(); //终结符
4. Map<String, Set<String>> First = new HashMap<>(); //First 集
5. Map<String, Set<String>> Follow = new HashMap<>(); //Follow 集
6. Map<String, Set<String>> select = new HashMap<>(); //产生式的 select 集
7. List<String[]> Symbol_Gram = new ArrayList<>(); // { { 符号, 存在的文法 } *n }
8. //使用 set 保存数据, 确保无重复元素 在计算的时候无序手动排除重复元素
```

```
1. 传入文法 G
2.
3. G 按照\n 和 "->" 以及 "\\|" 分割为单元
4.
5. if(成功){
6.     保存并更新语法
7. }
8. else{
9.     弹出语法错误警告
10. }
11. if(存在左递归){
12.     弹出左递归警告
13. }
14.
15. 计算 First 集(){
16.     for(String symbol: 终结符){
17.         First(symbol) = {symbol}
18.     } //终结符的 First 集是本身
19.     while(First 集的大小还在变化){
20.         for(String 左边->右边 : 所有文法){
21.             取出右边下标为 0 的符号
22.             if(当前符号为非终结符){ //当前符号的 First 集除了空都加入到左边符号的 First 集中
23.                 First(左边).addAll(First(当前符号).except("ε")); //
24.                 if(当前符号的 First 集合含有空){
25.                     看向下一个符号 //即下标+1 递归处理
26.                 }
27.             }
28.             else if(当前符号是终结符 或者 "ε"){
29.                 把当前符号加入到左边符号的 First 集中
30.             }
31.             else if(是 "\\0"){
32.                 停止
33.             }
34.             else{
35.                 停止
36.             }
37.         }
38.     }
39. }
40.
41. 计算 Follow 集(){
42.     在开始符号的 Follow 集中加入 "#"
43.     for(String 当前符号: 非终结符){
44.         for(语法 当前语法: 所有的含有当前计算 Follow 集符号的语法){
45.             String 紧跟符号 = 当前语法中, 当前符号之后的一个符号
46.             if(紧跟符号为终结符){
47.                 把紧跟符号加到当前符号的 Follow 集中
48.             }
49.             else if(紧跟符号为非终结符){
50.                 把紧跟符号的 First 集 - "ε" 加入到当前符号的 Follow 集中
51.                 if(当前符号可以的 First 集合含有空)
52.                     看向下一个符号 //也是递归求解
53.             }
54.             else if(当前符号是 "\\0"){
55.                 把空加入到当前符号的 Follow 集
56.             }
57.             else{
58.                 报错 停止
59.             }
60.         }
61.     }
62. }
63.
64. 计算 Select 集(){
```

```

65.     for(String 当前文法(左边->右边):所有文法){
66.         String 当前符号 = 右边的第一个符号
67.         if(当前符号是终结符){
68.             把当前符号加入 Select(当前文法)
69.         }
70.         else if(当前符号是"ε"或者是"\0"){
71.             把 Follow(左边)加入到 Select(当前文法)
72.         }
73.         else if(当前符号是非终结符){
74.             把 First(当前符号).except("ε")加入到 Select(当前文法)中
75.             if(当前符号的 First 集含有"ε")
76.                 看向下一个符号 递归求解
77.         }
78.         else{
79.             报错 停止
80.         }
81.     }
82. }
83.
84. 计算M表(){
85.     for(String 当前文法:select 集){
86.         for(String 当前符号:Select(当前文法)){
87.             M(当前文法 的 左边,当前符号) = 当前文法
88.         }
89.     }
90. }
91.
92. 分析过程(String 输入的内容){
93.     初始化分析栈
94.     初始化输入栈
95.     while(结束标记为未结束){
96.         if(存在文法){
97.             if(M(x,a) == "ε"){
98.                 分析栈出栈
99.             }
100.            else{
101.                分析栈.push(M(分析栈.pop(),a))
102.            }
103.        }
104.        else if(匹配到了 且没有结束){
105.            输入栈.pop()
106.            分析栈.pop()
107.        }
108.        else if(匹配成功, 是#){
109.            结束标记修改为结束
110.        }
111.        else{
112.            报错
113.            结束标记修改为结束
114.        }
115.    }
116. }

```



## 2.算法流程图



LL(1)预测分析程序流程

### 3. 源码及测试结果

#### Main.java :

```
1. package 实验二__LL1分析法;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         Windows windows = new Windows();
6.     }
7. }
```

#### Solution.java

```
1. package 实验二__LL1分析法;
2. import java.util.*;
3. import java.util.List;
4. import java.util.stream.Collectors;
5.
6. class Solution {
7.     Map<String,String> AnaTable = new HashMap<>();
8.     Map<String,Set<String>> select = new HashMap<>();//产生式的 select 集
9.     List<String[]> Symbol_Gram = new ArrayList<>();// { { 符号 , 存在的文法 } *n }
10.    String x;
11.    String a;
12.    Set<String> noTerminal = new HashSet<>();
13.    Set<String> terminal = new HashSet<>();
14.    Set<String> allG = new HashSet<>();
15.    Map<String,Set<String>> First = new HashMap<>();
16.    Map<String,Set<String>> Follow = new HashMap<>();
17.    String GStart = "null";
18.    Solution(){
19.        setG("E -> TG \n" +
20.            "G -> +TG | -TG \n" +
21.            "G -> ε \n" +
22.            "T -> FS \n" +
23.            "S -> *FS | /FS \n" +
24.            "S -> ε \n" +
25.            "F -> (E) \n" +
26.            "F->i \n");
27.    }
28.    String setG(String G){
29.        Map<String,String> AnaTable = new HashMap<>();
30.        Symbol_Gram.clear();
31.        G = G.replaceAll(" ", "");
32.        String []Gs = G.split("\n");
33.        if(Gs.length<1){
34.            return "输入错误";
35.        }
36.        GStart = Gs[0].substring(0,1) ;
37.        for (String gLine : Gs) {
38.            if( gLine.split("->").length!=2){
39.                return "格式错误";
40.            }
41.            String split0 = gLine.split("->")[0];
42.            for (String str : gLine.split("->")[1].split("\\|")) {
43.                String [] SingleG = {split0,str};
44.                Symbol_Gram.add(SingleG);
45.            }
46.        }
47.        GStart = Gs[0].substring(0,1) ;
48.        MyStack stack = new MyStack();
49.        for (String[] strings : Symbol_Gram) {
50.            if(strings[0].equals(strings[1].substring(0,1))){
51.                return "左递归";
52.            }
53.        }
54.        getFF();//计算 First Follow 集
55.        for (String[] strings : Symbol_Gram) {
56.            select.put(strings[0]+strings[1],new HashSet<>());
57.        }
58.        //例如 s->ab strings[0] -> strings[1]
59.        for (String[] strings : Symbol_Gram) {
```

```

60.         setSelect(strings,0);    //创建 select 集  key = 产生式 value = [] Set
61.     }
62.     AnaTable.clear();//清空
63.     terminal.add("#");//在终结符内加入# 以达到
64.     for (String s0 : select.keySet()) {
65.         for (String s1 : select.get(s0)) {
66.             AnaTable.put(s0.substring(0,1)+s1,s0.substring(1));
67.         }
68.     }
69.     this.AnaTable = AnaTable;
70.     return null;
71. }
72. Vector<String[]> Solve(String text){
73.     Vector<String[]> procesList = new Vector();
74.     int textLength = text.length();
75.     MyStack AnaStack = new MyStack();//分析栈
76.     MyStack inputString = new MyStack();//输入串
77.     AnaStack.push("#","E","S");
78.     inputString.push( new StringBuffer(text).reverse().toString().split("") );
79.     Boolean flag = true;
80.     Boolean matched = true;
81.     int linenumber = 0;
82.     while (flag){
83.         String[] INFO = new String[5];
84.         INFO[0] = String.valueOf(linenumber++);
85.         INFO[1] = AnaStack.toString();
86.         StringBuffer inputsb= new StringBuffer(inputString.toString());
87.         for(int sblength = inputsb.length() ; sblength < textLength+1 ; sblength++){
88.             inputsb.append(" ");
89.         }
90.         INFO[2] = inputsb.reverse().toString();
91.         x = AnaStack.getTop();//获取分析栈顶
92.         a = inputString.getTop();//第一个符号读到 a
93.         if(M(x,a)!=null){//存在对应的文法
94.             if(M(x,a)[0].equals("ε")){//为空
95.                 INFO[3] = x+" -> ε";
96.                 INFO[4] = "POP";
97.                 AnaStack.pop();
98.             }
99.             else {
100.                 AnaStack.push(M(AnaStack.pop(), a));
101.                 StringBuffer sb = new StringBuffer();
102.                 for(String string:M(x,a)){
103.                     sb.append(string);
104.                 }
105.                 INFO[3] = x+" -> " + sb.reverse();
106.                 INFO[4] = "POP,PUSH("+sb.reverse()+")";
107.                 //System.out.println("存在文法
108.                 ["+x+", "+a+"] -> ["+sb + " ] STACK:" + AnaStack);
109.             }
110.             else if( !x.equals("#") && x.equals(a) ){//匹配到了
111.                 //System.out.println("匹配到了"+x+" "+a);
112.                 INFO[4] = "POP,GETNEXT(i)";
113.                 inputString.pop();
114.                 AnaStack.pop();
115.             }
116.             else if(x.equals("#") && x.equals(a)){//结束了
117.                 flag=false;
118.             }
119.             else{//报错
120.                 flag=false;
121.                 matched=false;
122.             }
123.             for(int i = 0 ; i < 5 ; i++){
124.                 if(INFO[i]!=null)
125.                     INFO[i] = " "+INFO[i];
126.                 else
127.                     INFO[i] = " ";
128.             }
129.             procesList.add(INFO);
130.         }
131.     }
132.     if(matched){
133.         //System.out.println("匹配成功");
134.     }
135.     else{
136.         //System.out.println("匹配失败");
137.         String ss[] ={"ERROR","ERROR","ERROR","ERROR","ERROR"};
138.         procesList.add(ss);
139.     }
140. }
141. // for (String s : Grammer) {
142. //     System.out.print(s+" ");
143. // }
144. return procesList;
145. }

```

```

146. String[] M(String Line ,String column){
147.     //倒序 分割
148.     //System.out.println("查询 " + Line + "<->" + column);
149.     if(AnaTable.get(Line+column) == null) {
150.         if ( AnaTable.get(Line+"#")!=null) {
151.             // System.out.println("返回空");
152.             String ss[] = {"ε"};
153.             return ss;
154.         }
155.         else {
156.             return null;
157.         }
158.     }
159.     else{
160.         return new StringBuffer(AnaTable.get(Line+column)).reverse().toString().split("
");
161.     }
162. }
163. Map[] getFF(){
164.     noTerminal.clear();
165.     First.clear();
166.     Follow.clear();
167.     allG.clear();
168.     terminal.clear();
169.     for (String[] strings : Symbol_Gram) {
170.         noTerminal.add(strings[0]);
171.     }//非终结符
172.     for (String[] strings : Symbol_Gram) {
173.         allG.add(strings[0]);
174.         for (String s : strings[1].split("")) {
175.             allG.add(s);
176.         }
177.     }//所有符
178.     allG.remove("ε");
179.     rs.toSet());terminal.addAll(allG.stream().filter(S-> !noTerminal.contains(S)).collect(Collectors.toSet()));//终结符 = 所有符号 - 非终结符
180.     //System.out.println(Grammer+"\n"+EndG);
181.     for (String s1 : allG) {
182.         First.put(s1,new HashSet<>());
183.         Follow.put(s1,new HashSet<>());
184.     }//终结符的 First 集是本身终结符的 First 集是本身
185.     for (String s1 : terminal) {
186.         First.get(s1).add(s1);
187.     }
188.     int FirstSize = 0;
189.     do{
190.         FirstSize = 0;
191.         for (String s1 : First.keySet()) {
192.             FirstSize+=First.get(s1).size();
193.         }
194.         for (String[] strings : Symbol_Gram) {
195.             String lam = strings[1];
196.             String G = strings[0];
197.             setFirst(lam,G);
198.         }
199.         for (String s1 : First.keySet()) {
200.             FirstSize-=First.get(s1).size();
201.         }
202.     }while (FirstSize != 0);
203.     Follow.get(GStart).add("#");//文法开始符号 Follow 加入#
204.
205.     int FollowSize = 0;
206.     do{
207.         FollowSize = 0;
208.         for (String s1 : Follow.keySet()) {
209.             FollowSize+=Follow.get(s1).size();
210.         }
211.         for (String[] strings : Symbol_Gram) {
212.             String lam = strings[1];
213.             String G = strings[0];
214.             for (String s2 : noTerminal) {
215.                 setFollow(lam,G,s2);
216.             }
217.         }
218.
219.
220.         for (String s1 : Follow.keySet()) {
221.             FollowSize-=Follow.get(s1).size();
222.         }
223.     }while (FollowSize != 0);
224.     for (String s1 : terminal) {
225.         Follow.remove(s1);
226.     }
227.     Map[] res = new Map[2];
228.     res[0] = First;
229.     res[1] = Follow;
230.     return res;

```

```

231.     }
232.     private void setFirst(String lam,String G){
233.         String first = lam.substring(0,1);
234.         if(terminal.contains(first)){//终结符
235.             First.get(G).add(first);
236.         }
237.         else if(first.equals("ε")){//符号空
238.             First.get(G).add("ε");
239.         }
240.         else if(noTerminal.contains(first)){//非终结符
241.             First.get(G).addAll(First.get(first).stream().filter(S->!S.equals("ε")).collect
242. (Collectors.toSet()));
243.             if(M(first,"ε")!=null){//是否可以推出空
244.                 setFirst(lam.substring(1),G);//扫描下一个
245.             }
246.         }
247.         else {
248.         }
249.     }
250.     private void setFollow(String lam,String G,String sym){//产生式 ->左边的符号 当先所求的
非终结符
251.         if(!lam.contains(sym)){
252.             return;
253.         }
254.         int index = lam.indexOf(sym);//位置
255.         if(index == lam.length()-1){//是\0
256.             Follow.get(sym).add("#");
257.             Follow.get(sym).addAll(Follow.get(G));//把产生式左边的 FOLLOW 加入到其的 FOLLOW 集
中
258.             return;
259.         }
260.         else if(index < lam.length()-1){
261.             String next = lam.substring(index+1,index+2);//右边的符号
262.             if(terminal.contains(next)){//是终结符
263.                 Follow.get(sym).add(next);
264.             }
265.             else if(noTerminal.contains(next)){//非终结符
266.                 Follow.get(sym).addAll(First.get(next).stream().filter(S->!S.equals("ε")).c
ollect(Collectors.toSet()));//把他的 First 集-ε 加入到当前分析的 Follow 集中
267.                 if(M(next,"ε")!=null){//检查可否推出空
268.                     //扫描下一个符号
269.                     String changedLam = new StringBuffer(lam) ;
270.                     changedLam.deleteCharAt(index+1);//删除 达到左移的效果\
271.                     setFollow(changedLam.toString(),G,sym);
272.                 }
273.             }
274.         }
275.     }
276. }
277. }
278. private void setSelect(String[] strings,int index){
279.     if(index == strings[1].length()){//是空
280.         select.get(strings[0]+strings[1]).addAll(Follow.get(strings[0]));
281.     }
282.     else {
283.         String firstSym = strings[1].substring(index,index+1);
284.         if(terminal.contains(firstSym)){//如果是终结符
285.             select.get(strings[0]+strings[1]).add(strings[1].substring(0,1));
286.         }
287.         else if(firstSym.equals("ε")){//是空
288.             select.get(strings[0]+strings[1]).addAll(Follow.get(strings[0]));
289.         }
290.         else if(noTerminal.contains(firstSym)){//是非终结符
291.             select.get(strings[0]+strings[1]).addAll(First.get(firstSym).stream().filter(S->!S.equals("ε")).collect(Collectors.toSet()));
292.             if(M(firstSym,"ε") != null){//可以推空 则扫描下一个
293.                 setSelect(strings,index+1);
294.             }
295.         }
296.     }
297. }
298. }
299. class MyStack{
300.     List<String> s;
301.     MyStack(){
302.         s = new LinkedList<>();
303.     }
304.     void push(String value){
305.         s.add(value);
306.     }
307.     void push(String...values){
308.         for(String value:values){
309.             push(value);
310.         }
311.     }
312.     String pop(){

```

```

313.         return s.remove(s.size()-1);
314.     }
315.     String getTop(){
316.         return s.get(s.size()-1);
317.     }
318.     @Override
319.     public String toString(){
320.         StringBuffer sb = new StringBuffer();
321.         for(String value:s){
322.             sb.append(value);
323.         }
324.         return sb.toString();
325.     }
326.     public Boolean isEmpty(){
327.         return s.size()==0;
328.     }
329. }

```

## GUI.java

```

1.  package 实验二__LL1分析法;
2.
3.  import javax.swing.*;
4.  import javax.swing.table.AbstractTableModel;
5.  import java.awt.*;
6.  import java.awt.event.ActionEvent;
7.  import java.awt.event.ActionListener;
8.  import java.util.*;
9.
10. class Windows extends JFrame {
11.     JButton clear,confirm,setG,FF;
12.     JTextArea textArea;
13.     JTabbedPane tabbedPane;
14.     Solution sol;
15.     Windows(){
16.         setVisible(false);
17.         try{
18.             setIconImage(new ImageIcon("bilibili.PNG").getImage());
19.             Font f = new Font("Yahei Consolas Hybrid",Font.PLAIN,16);
20.             String names[]={ "MenuBar","Menu","MenuItem", "TextArea", "Button", "ScrollPa
ne", "Table", "TabbedPane"};
21.             for (String item : names) {
22.                 UIManager.put(item+ ".font",f);
23.             }
24.             UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
25.         }catch(Exception e){}
26.         init();
27.         setSize(800,600);//初始大小
28.         setLocation(300,200);//初始位置
29.         setVisible(true);//是否可视
30.         setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);//X退出
31.     }
32.     void init(){
33.         setTitle("LL(1)分析法");
34.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
35.         setVisible(true);
36.         setResizable(false);
37.         setLayout(null);
38.         sol = new Solution();
39.         initButton();
40.         initText();
41.         initResult();
42.     }
43.
44.     void initButton(){
45.         class clearListen implements ActionListener{
46.             @Override
47.             public void actionPerformed(ActionEvent e){
48.                 textArea.setText("");
49.                 tabbedPane.removeAll();
50.                 tabbedPane.updateUI();
51.             }
52.         }
53.         class confirmListen implements ActionListener{
54.             @Override
55.             public void actionPerformed(ActionEvent e){
56.                 tabbedPane.removeAll();
57.                 for(String text:textArea.getText().split("\n")){

```

```

58.         if(text.length()<2 || !text.substring(text.length()-1).equals("#") ){
59.             JOptionPane.showMessageDialog(null, "格式输入错误
", "Error !", JOptionPane.ERROR_MESSAGE);
60.             break;
61.         }
62.         addTable(text,sol.Solve(text));
63.         tabbedPane.updateUI();
64.     }
65.
66.
67.     }
68. }
69. class MyDialog extends JDialog implements ActionListener{
70.     JTextArea input;
71.     JButton confirm,cancel;
72.     String title;
73.     MyDialog(Windows f){
74.         setLayout(null);
75.         setResizable(false);
76.         setIconImage(new ImageIcon("bilibili.PNG").getImage());
77.         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
78.         setTitle("请输入语法");
79.         input=new JTextArea("E -> TG \n" +
80.             "G -> +TG | -TG \n" +
81.             "G -> ε \n" +
82.             "T -> FS \n" +
83.             "S -> *FS | /FS \n" +
84.             "S -> ε \n" +
85.             "F -> (E) \n" +
86.             "F->i \n");
87.         JScrollPane jScrollPane = new JScrollPane(input);
88.         jScrollPane.setBounds(10,10,265,200);
89.         add(jScrollPane);
90.
91.         class confirmListener implements ActionListener{
92.             @Override
93.             public void actionPerformed(ActionEvent e){
94.                 String getInput = input.getText();
95.                 String setRes = sol.setG(getInput);
96.                 if(setRes!=null){
97.                     JOptionPane.showMessageDialog(null, setRes, "Error !", JOptionP
ane.ERROR_MESSAGE);
98.                 }
99.                 else {
100.                     setVisible(false);
101.                 }
102.             }
103.         }
104.         confirm=new JButton("确定");
105.         confirm.addActionListener(new confirmListener());
106.         confirm.setBounds(195,220,80,30);
107.         add(confirm);
108.
109.         class cancelListener implements ActionListener{
110.             @Override
111.             public void actionPerformed(ActionEvent e){
112.                 input.setText("E -> TG \n" +
113.                     "G -> +TG | -TG \n" +
114.                     "G -> ε \n" +
115.                     "T -> FS \n" +
116.                     "S -> *FS | /FS \n" +
117.                     "S -> ε \n" +
118.                     "F -> (E) \n" +
119.                     "F->i \n");
120.                 setVisible(false);
121.             }
122.         }
123.         cancel=new JButton("取消");
124.         cancel.addActionListener(new cancelListener());
125.         cancel.setBounds(105,220,80,30);
126.         add(cancel);
127.
128.
129.         setBounds(600,260,300,300);
130.         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
131.     }
132.     public void actionPerformed(ActionEvent e){
133.         setVisible(true);
134.     }
135. }
136. class FFListen extends JDialog implements ActionListener{
137.     JTabbedPane jTabbedPane;
138.     FFListen(){
139.         jTabbedPane = new JTabbedPane();
140.         setLayout(null);
141.         setResizable(false);
142.         setIconImage(new ImageIcon("bilibili.PNG").getImage());

```

```

143.         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
144.         setTitle("Fist Follow 集");
145.         setBounds(650,300,300,400);
146.         jTabbedPane.setBounds(10,7,267,350);
147.         add(jTabbedPane);
148.
149.     }
150.     @Override
151.     public void actionPerformed(ActionEvent e){
152.         setVisible(true);
153.         jTabbedPane.removeAll();
154.         JTable First,Follow;
155.         Map<String,Set<String>>[] res = sol.getFF();
156.
157.         Object[][] FirstData = new Object[res[0].size()][2];
158.         int index = 0;
159.         for (String s : res[0].keySet()) {
160.             FirstData[index][0] = s;
161.             FirstData[index][1] = res[0].get(s).toString();
162.             index++;
163.         }
164.         Object[] columnNames = {"", ""};
165.         First = new JTable(FirstData, columnNames);
166.         First.setRowHeight(24);
167.         First.getTableHeader().setVisible(false);
168.         JScrollPane FirstScrollable = new JScrollPane(First);
169.         FirstScrollable.setBorder(null);
170.         jTabbedPane.addTab("First 集",FirstScrollable);
171.
172.         Object[][] FollowData = new Object[res[1].size()][2];
173.         index = 0;
174.         for (String s : res[1].keySet()) {
175.             FollowData[index][0] = s;
176.             FollowData[index][1] = res[1].get(s).toString();
177.             index++;
178.         }
179.         Follow = new JTable(FollowData, columnNames);
180.         Follow.setRowHeight(24);
181.         Follow.getTableHeader().setVisible(false);
182.         JScrollPane FollowScrollable = new JScrollPane(Follow);
183.         FollowScrollable.setBorder(null);
184.         jTabbedPane.addTab("Follow 集",FollowScrollable);
185.
186.     }
187. }
188.
189.
190. clear = new JButton("清除");
191. clear.setBounds(600,160,80,30);
192. clear.addActionListener(new clearListen());
193.
194. confirm = new JButton("确认");
195. confirm.setBounds(695,160,80,30);
196. confirm.addActionListener(new confirmListen());
197.
198. setG = new JButton("自定义语法");
199. setG.setBounds(460,160,120,30);
200. setG.addActionListener(new MyDialog(this));
201.
202. FF = new JButton("Fist,Follow 集");
203. FF.setBounds(260,160,180,30);
204. FF.addActionListener(new FFListen());
205.
206. class selectListen extends JDialog implements ActionListener{
207.     JTabbedPane jTabbedPane;
208.     selectListen(){
209.         jTabbedPane = new JTabbedPane();
210.         setLayout(null);
211.         setResizable(false);
212.         setIconImage(new ImageIcon("bilibili.PNG").getImage());
213.         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
214.         setTitle("select 集");
215.         setBounds(650,300,500,300);
216.         jTabbedPane.setBounds(10,7,467,250);
217.         add(jTabbedPane);
218.
219.     }
220.     @Override
221.     public void actionPerformed(ActionEvent e){
222.         setVisible(true);
223.         jTabbedPane.removeAll();
224.         JTable First,Follow;
225.         Map<String,Set<String>> res = sol.select;
226.
227.         Object[][] FirstData = new Object[res.size()][2];
228.         int index = 0;
229.         for (String s : res.keySet()) {

```



```

230.         FirstData[index][0] = s;
231.         FirstData[index][1] = res.get(s).toString();
232.         index++;
233.     }
234.     Object[] columnNames = {"", ""};
235.     First = new JTable(FirstData, columnNames);
236.     First.setRowHeight(24);
237.     First.getTableHeader().setVisible(false);
238.     JScrollPane FirstScrollable = new JScrollPane(First);
239.     FirstScrollable.setBorder(null);
240.     jTabbedPane.addTab("select 集", FirstScrollable);
241.
242.     {
243.         Map<String,Integer> Grammap = new HashMap<>();
244.         Map<String,Integer> endGmap = new HashMap<>();
245.
246.         int a = 0;
247.         for (String s1 : sol.terminal) {
248.             //System.out.println(" "+s1);
249.             endGmap.put(s1,a);
250.             a++;
251.         }
252.         int b = 0;
253.         for (String s : sol.noTerminal) {
254.             Grammap.put(s,b);
255.             b++;
256.         }
257.         String[][] map = new String[Grammap.size()][endGmap.size()];
258.         //System.out.println();
259.         for (String s1 : sol.noTerminal) {
260.             //System.out.println(s1+": ");
261.             for (String s2 : sol.terminal) {
262.                 String resss = sol.AnaTable.get(s1+s2);
263.                 if(resss == null){
264.                     //System.out.println(" ");
265.                     map[Grammap.get(s1)][endGmap.get(s2)] = " ";
266.                 }
267.                 else {
268.                     //System.out.println(resss+" ");
269.                     map[Grammap.get(s1)][endGmap.get(s2)] = resss;
270.                 }
271.             }
272.             //System.out.println();
273.         }
274.         //System.out.println("=====");
275.
276.
277.
278.
279.         //
280.         //
281.         //
282.         //
283.         //
284.         //
285.         for (String[] strings : map) {
286.             for (String string : strings) {
287.                 System.out.print(string+" ");
288.             }
289.             System.out.println();
290.         }
291.
292.         String [] colum = new String[sol.noTerminal.size()+1];
293.         int counter = 0;
294.         for (String s : sol.noTerminal) {
295.             colum[counter] = s;
296.             counter++;
297.         }
298.         String[][] data = new String[map.length][map[0].length+1];
299.         for (int j = 0; j < data.length; j++) {
300.             data[j][0] = colum[j];
301.             for (int i = 1; i < data[0].length; i++) {
302.                 data[j][i] = map[j][i-1];
303.             }
304.         }
305.         String[] name = new String[sol.terminal.size()+1];
306.         int i = 1;
307.         name[0] = " ";
308.         for (String s : sol.terminal) {
309.             name[i] = s;
310.             i++;
311.         }
312.         JTable mmm = new JTable(data, name);
313.         mmm.setRowHeight(30);
314.         JScrollPane secsa = new JScrollPane(mmm);
315.         secsa.setBorder(null);
316.         jTabbedPane.addTab("分析表",secsa);
317.     }
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.

```

```

317.
318.     }
319. }
320.
321.
322.     JButton select;
323.     select = new JButton("secect 集");
324.     select.setBounds(10,160,180,30);
325.     select.addActionListener(new selectListen());
326.     add(select);
327.
328.     add(clear);
329.     add(confirm);
330.     add(setG);
331.     add(FF);
332. }
333. void initText(){
334.     textArea = new JTextArea("i+i*i#\ni*i*i#\ni*(i+i)#\ni^i#");
335.     JScrollPane textAreaRollPane = new JScrollPane(textArea);
336.     textAreaRollPane.setBounds(10,10,765,140);
337.     add(textAreaRollPane);
338. }
339. void initResult(){
340.     tabbedPane = new JTabbedPane();
341.     tabbedPane.setBounds(10, 200, 765, 350);
342.     add(tabbedPane);
343. }
344. void addTable(String title,Vector vec ){
345.     TableDataModel tableDataModel = new TableDataModel(vec);
346.     JTable table = new JTable(tableDataModel);
347.     table.setVisible(true);
348.     table.setPreferredScrollableViewportSize(new Dimension(550, 100));
349.     table.setRowHeight(24);
350.     JScrollPane tablePane = new JScrollPane(table);
351.     tablePane.setBounds(10, 200, 765, 350);
352.     tabbedPane.addTab(title,tablePane);
353. }
354. }
355.
356. class TableDataModel extends AbstractTableModel {
357.     private Vector<String[]> TableData; //用来存放表格数据的线性表
358.     private Vector<String> TableTitle; //表格的 列标题
359.     public TableDataModel(Vector data){
360.         String Names[] = {"步骤", "分析栈", "剩余输入栈", "所用产生式", "动作"};
361.         Vector Namessss = new Vector();
362.         for(String str:Names){
363.             Namessss.add(str);
364.         }
365.         TableTitle = Namessss;
366.         TableData = data;
367.     }
368.
369.     @Override
370.     public int getRowCount(){
371.         return TableData.size();
372.     }
373.     public int getColumnCount(){
374.         return TableTitle.size();
375.     }
376.     @Override
377.     public String getColumnName(int colum){
378.         return TableTitle.get(colum);
379.     }
380.     public Object getValueAt(int rowIndex, int columnIndex){
381.         String LineTemp[] = this.TableData.get(rowIndex);
382.         return LineTemp[columnIndex];
383.     }
384.     @Override
385.     public boolean isCellEditable(int rowIndex, int columnIndex){ //不允许编辑
386.         return false;
387.     }
388. }

```

运行结果：

LL(1)分析法

i+i\*i#

i\*i\*i#

i\*(i+i)#

i^i#

secect集

Fist,Follow集

自定义语法

清除

确认

i+i\*i#

i\*i\*i#

i\*(i+i)#

i^i#

步骤	分析栈	剩余输入栈	所用产生式	动作
6	#G	+i*i#	G -> +TG	POP,PUSH(GT+)
7	#GT+	+i*i#		POP,GETNEXT(i)
8	#GT	i*i#	T -> FS	POP,PUSH(SF)
9	#GSF	i*i#	F -> i	POP,PUSH(i)
10	#GSi	i*i#		POP,GETNEXT(i)
11	#GS	*i#	S -> *FS	POP,PUSH(SF*)
12	#GSF*	*i#		POP,GETNEXT(i)
13	#GSF	i#	F -> i	POP,PUSH(i)
14	#GSi	i#		POP,GETNEXT(i)
15	#GS	#	S -> ε	POP
16	#G	#	G -> ε	POP
17	#	#		

LL(1)分析法

$i+i\#$   
 $i*i\#$   
 $i*(i+i)\#$   
 $i^i\#$

secect集

$i+i\#$   $i*i\#$   $i*(i+i)\#$   $i^i\#$

步骤	分析栈	剩余输入	动作
6	#G	$i+i\#$	POP,PUSH(GT+)
7	#GT+	$i+i\#$	POP,GETNEXT(i)
8	#GT	$i+i\#$	POP,PUSH(SF)
9	#GSF	$i+i\#$	POP,PUSH(i)
10	#GSi	$i+i\#$	POP,GETNEXT(i)
11	#GS	$i+i\#$	POP,PUSH(SF*)
12	#GSF*	$i+i\#$	POP,GETNEXT(i)
13	#GSF	$i+i\#$	POP,PUSH(i)
14	#GSi	$i+i\#$	POP,GETNEXT(i)
15	#GS	$i+i\#$	POP
16	#G	$i+i\#$	POP
17	#	$i+i\#$	

请输入语法

$E \rightarrow TG$   
 $G \rightarrow +TG \mid -TG$   
 $G \rightarrow \epsilon$   
 $T \rightarrow FS$   
 $S \rightarrow *FS \mid /FS$   
 $S \rightarrow \epsilon$   
 $F \rightarrow (E)$   
 $F \rightarrow i$

取消 确定

清除 确认

LL(1)分析法

$i+i\#$   
 $i*i\#$   
 $i*(i+i)\#$   
 $i^i\#$

secect集

$i+i\#$   $i*i\#$   $i*(i+i)\#$   $i^i\#$

步骤	分析栈	剩余输入	动作
6	#G	$i+i\#$	POP,PUSH(GT+)
7	#GT+	$i+i\#$	POP,GETNEXT(i)
8	#GT	$i+i\#$	POP,PUSH(SF)
9	#GSF	$i+i\#$	POP,PUSH(i)
10	#GSi	$i+i\#$	POP,GETNEXT(i)
11	#GS	$i+i\#$	POP,PUSH(SF*)
12	#GSF*	$i+i\#$	POP,GETNEXT(i)
13	#GSF	$i+i\#$	POP,PUSH(i)
14	#GSi	$i+i\#$	POP,GETNEXT(i)
15	#GS	$i+i\#$	POP
16	#G	$i+i\#$	POP
17	#	$i+i\#$	

Fist Follow 集

First集 Follow集

S	[ $\epsilon$ , *, /]
T	[(, i]
E	[(, i]
F	[(, i]
G	[ $\epsilon$ , +, -]
(	[()
)	[]]
i	[i]
*	[*]
+	[+]
-	[-]
/	[/]

清除 确认

LL(1)分析法

$i+i\#$   
 $i*i\#$   
 $i*(i+i)\#$   
 $i^\wedge\#$

select集      First, Follow集

步骤	分析栈	剩余输入	动作
6	#G	+i*i#	
7	#GT+	+i*i#	
8	#GT	i*i#	
9	#GSF	i*i#	
10	#GSi	i*i#	
11	#GS	*i#	
12	#GSF*	*i#	
13	#GSF	i#	
14	#GSi	i#	
15	#GS	#	S → ε      POP
16	#G	#	G → ε      POP
17	#	#	

First Follow 集

First集	Follow集
S	[#, ), +, -]
T	[#, ), +, -]
E	[#, )]
F	[#, ), *, +, -, /]
G	[#, )]

P,PUSH(GT+)  
 P,GETNEXT(i)  
 P,PUSH(SF)  
 P,PUSH(i)  
 P,GETNEXT(i)  
 P,PUSH(SF\*)  
 P,GETNEXT(i)  
 P,PUSH(i)  
 POP,GETNEXT(i)

删除      确认

LL(1)分析法

$i+i\#$   
 $i*i\#$   
 $i*(i+i)\#$   
 $i^\wedge\#$

select集      First, Follow集

步骤	分析栈	剩余输入	动作
6	#G	+i*i#	
7	#GT+	+i*i#	
8	#GT	i*i#	
9	#GSF	i*i#	
10	#GSi	i*i#	
11	#GS	*i#	
12	#GSF*	*i#	
13	#GSF	i#	
14	#GSi	i#	
15	#GS	#	S → ε      POP
16	#G	#	G → ε      POP
17	#	#	

First Follow 集

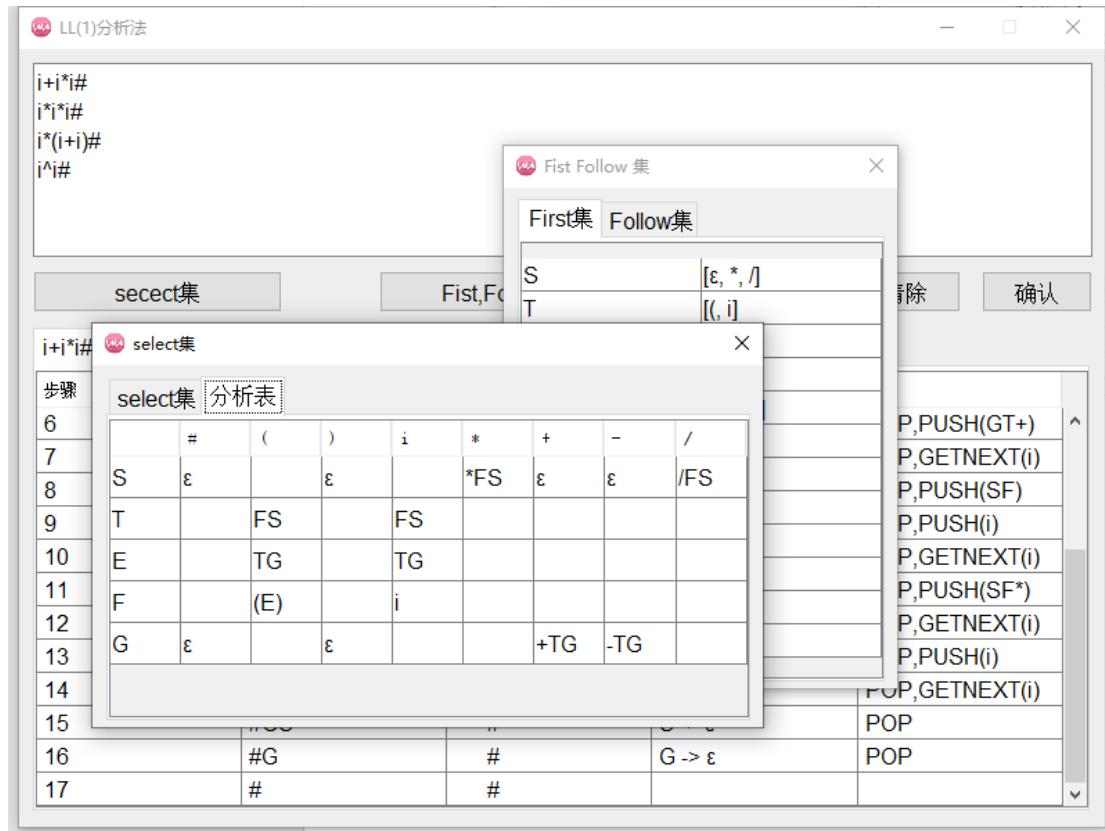
First集	Follow集
S	[#, ), +, -]
T	[#, ), +, -]

select集 分析表

select集	分析表
TFS	[(, i]
S*FS	[*]
Sε	[#, +, -]
Fi	[i]
F(E)	[(]
G+TG	[+]
ETG	[(, i]
G-TG	[-]

P,PUSH(GT+)  
 P,GETNEXT(i)  
 P,PUSH(SF)  
 P,PUSH(i)  
 P,GETNEXT(i)  
 P,PUSH(SF\*)  
 P,GETNEXT(i)  
 P,PUSH(i)  
 POP,GETNEXT(i)

删除      确认



## 4.实验收获

本次实验设计的程序层次分明，程序界面与处理程序低耦合高内聚，分析器使用文法作为参数，对传入的字符串进行分析，并返回分析结果，此外通过调用类中的方法可以返回对应的 First 集 Follow 集等，便于显示。LL1 对分析法有了更好的理解和掌握。

# 实验三 LR(1)分析法

## 9. 数据结构及算法描述

```
a) public String grammarText;
b) public List<String[]> grammar = new ArrayList<>(); //语法格式:(T->SF)= {T,SF}
c) public Set<String> nonTerminal = new HashSet<>(); //非终结符
d) public Set<String> terminal = new HashSet<>(); //终结符
e) public Set<String> allSymbol = new HashSet<>(); //全部符号
f) public Map<String,Set<String>> First = new HashMap<>(); //First 集
g) public List<project> projectList = new ArrayList<>(); //所有项目
h) public List<projectSet> cProjectSets = new ArrayList<>(); //项目集 C
i) public Map<point,String> ActionTable = new HashMap<>(); //action 表
j) public Map<point,String> GOTO = new HashMap<>(); //goto 表
k) public String[][] ActionAndGoTo; //Action 和 GOTO 表
l) public String startSymbol = "S"; //文法开始符号

1. 初始化文法
2. 读取文法,计算 First 集(使用实验 2 的算法即可)
3. 初始化项目 List(){
4.     加入(S'->.S),#和(S'-.>S.,#) //开始文法 特殊情况 手动添加
5.     for(文法 A->BC:所有文法){
6.         for( index : 所有可以插入点的位置){
7.             for(文法 当前文法:所有文法){
8.                 if(当前文法的右边含有 A){
9.                     把(A->BC,index,First(点后部的字符串))加入到项目集中
10.                }
11.            }
12.        }
13.    }
14. }
15. 初始化项目集 C(){
16.     初始化栈
17.     创建一个项目集
18.     把开始文法的项目放入其中
19.     计算它的闭包
20.     把这个项目放入栈中
21.     while(栈非空){
22.         当前项目集 = 栈.pop()
23.         for(String Symbol : 每个符号){
24.             创建一个新的项目集
25.             for(项目 : 当前项目集中的所有项目){
26.                 if(是形如 A->...•X... 的项目){
27.                     点向后移动一位创建新的项目,加入到新的项目集中
28.                 }
29.             }
30.             计算闭包(新的项目集)
31.             if(如果新的项目集为空){
32.                 标记当前项目集通过 Symbol 跳转到-1
33.             }
34.             else if(新的项目集已经存在){
35.                 标记当前项目集通过 Symbol 跳转到 项目集 List.indexOf(新的项目集)
36.             }
37.             else{
38.                 把新的项目集加入到项目集 List 中
39.                 标记当前项目集通过 Symbol 跳转到 项目集 List.indexOf(新的项目集)
40.                 栈.push(新的项目集)
41.             }
42.         }
43.     }
44. }
45. 计算闭包(项目集){
46.     新建栈
47.     项目集.forEach(栈::push)
48.     while(栈非空){
49.         当前项目 = 栈.pop()
50.         项目.add(当前项目);
51.         点后符号 = 当前项目的点后的第一个符号
52.         if(点后符号是非终结符){
53.             for(当前产生式:每个左边是 B 的产生式){
54.                 for(symbol:First(B 之后的部分,当前项目的展望符)){
55.                     new 新项目(当前产生式,0,symbol);
56.                     if(项目原本不存在项目集中){
57.                         栈.push(新项目)
58.                     }
59.                 }
60.             }
61.         }
62.     }
63. }
```

```

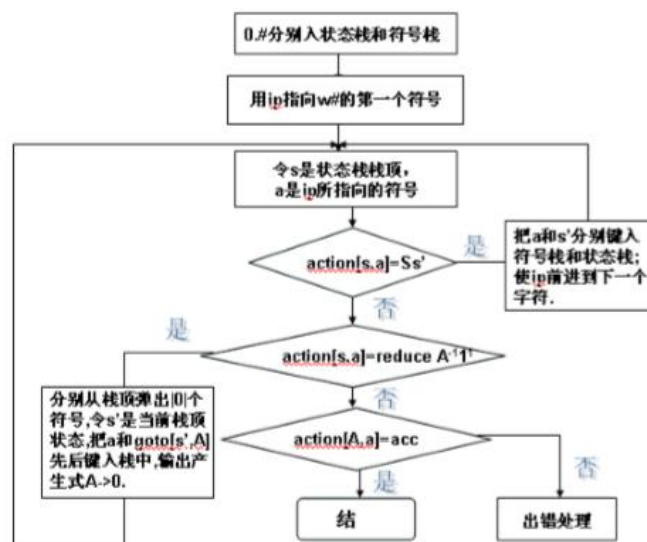
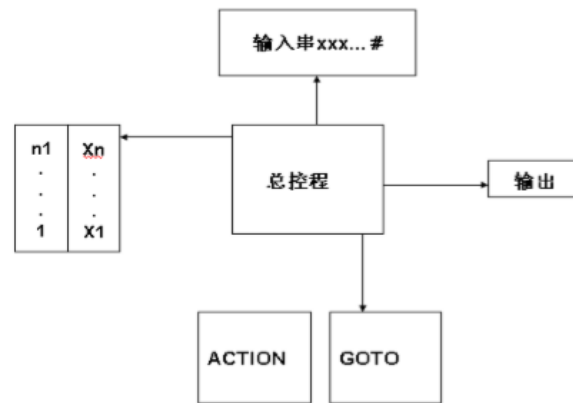
62.     }
63. }
64. Go(项目集,符号){
65.     if(项目集通过(符号跳转) != -1){
66.         return 项目集通过(符号跳转)的项目集
67.     }
68.     else
69.         return 空集
70. }
71. 创建 Action 和 Goto 表(){
72.     for(当前项目集:所有项目集){
73.         for( 当前项目 : 所有项目 ){
74.             当前项目 例如 [A->...•a...,b]
75.             if(a是终结符){
76.                 ActionTable(当前项目.index,a) = S + Go(当前项目集,a).index
77.             }
78.             if(a是""){
79.                 ActionTable(当前项目.index,b) = R + 当前项目集.indexOf 语法
80.             }
81.         }
82.     }
83.     ActionTable(初始项目.index,#) = acc
84.     for(当前项目集:所有项目集){
85.         for(当前符号:所有符号){
86.             Goto(当前项目集,当前符号) = Go(当前项目集,当前符号)
87.         }
88.     }
89.     其他位置标记为 err
90. }
91. 主控函数(){
92.     初始化输入串栈
93.     初始化符号栈
94.     初始化状态栈
95.     while(结束标记不为结束){
96.         String i = 状态栈顶
97.         String a = 输入串栈顶
98.         String action = ActionTable.(i,a)
99.         if(action() == null 或者 er ){
100.             报错
101.             标记结束
102.         }
103.         else if(action == acc ){
104.             成功
105.             标记结束
106.         }
107.         else if(action == Si){
108.             i 入状态栈
109.             a 到文法符号栈
110.         }
111.         else if(action == Ri){
112.             用 index 产生式规约
113.             符号栈中取出文法的右侧的长度的字符
114.             再压栈文法左边的符号
115.             状态栈.push( GOTO ( 状态栈.top() , 符号栈.top() ) )
116.         }
117.         else{
118.             报错
119.             标记结束
120.         }
121.     }
122. }

```

## 2.算法流程图



### LR分析器结构:



## 3. 源码及测试结果

Main.java :

```

1. package 实验三_LR1 分析法;
2.
  
```

```

3. public class Main {
4.     public static void main(String[] args) {
5.         new Thread(() -> new Windows()).start();
6.     }
7. }

```

## Solution.java

```

1. package 实验三_LR1 分析法;
2.
3.
4. import java.util.*;
5. import java.util.stream.Collectors;
6.
7. class Solution {
8.     public String grammarText;
9.     public List<String[]> grammar = new ArrayList<>(); //(T->SF)= {T,SF} forAll
10.    public Set<String> nonTerminal = new HashSet<>(); //非终结符
11.    public Set<String> terminal = new HashSet<>(); //终结符
12.    public Set<String> allSymbol = new HashSet<>(); //全部符号
13.    public Map<String,Set<String>> First = new HashMap<>();
14.    public List<project> projectList = new ArrayList<>(); //所有项目
15.    public List<projectSet> cProjectSets = new ArrayList<>(); //项目集C
16.    public Map<point,String> ActionTable = new HashMap<>(); //action 表
17.    public Map<point,String> GOTO = new HashMap<>(); //goto 表
18.    public String[][] ActionAndGoTo; //Action 和 GOTO 表
19.    public String startSymbol = "S";
20.    class project{//项目 A->αBβ,a grammar: A->αBβ index: location of(.) extSymbol: a
21.        private int indexOfGrammar;
22.        private int indexOfNode;
23.        private String extSymbol;
24.        project(int indexOfGrammar,int index,String extSymbol){
25.            this.indexOfGrammar = indexOfGrammar;
26.            this.indexOfNode = index;
27.            this.extSymbol = extSymbol;
28.        }
29.        public String[] getGrammar() { //获取产生式
30.            return grammar.get(indexOfGrammar);
31.        }
32.        public int getIndex() { //获取所用产生式的编号
33.            return indexOfNode;
34.        }
35.        public String getHead() { //获取产生式左边
36.            return this.getGrammar()[0];
37.        }
38.        public String getExtSymbol() {
39.            return extSymbol;
40.        }
41.        public String getRight() { //获取产生式右边点后面的部分
42.            return this.getGrammar()[1].substring(indexOfNode);
43.        }
44.        public String getFirstSymbolAfterNode() { //获取右侧字符串的首字符
45.            if(this.getRight().length()<1){
46.                return "";
47.            }
48.            else {
49.                return this.getRight().substring(0,1);
50.            }
51.        }
52.        public String getRestStringAfterFirst() { //获取右侧首个字符的之后的字符串
53.            if(this.getRight().length()<2){
54.                return "";
55.            }
56.            else {
57.                return this.getRight().substring(1);
58.            }
59.        }
60.        @Override
61.        public String toString() {
62.            StringBuffer str = new StringBuffer(grammar.get(indexOfGrammar)[1]);
63.            str.insert(indexOfNode,".");
64.            str.insert(0,"["+grammar.get(indexOfGrammar)[0]+"->");
65.            str.append(", "+extSymbol+"]");
66.            return str.toString();
67.        }
68.        @Override
69.        public boolean equals(Object obj) {
70.            if(!obj.getClass().equals(this.getClass())){
71.                return false;
72.            }
73.            project cmp = ((project)obj);

```

```

74.         if( (cmp.indexOfGrammar == this.indexOfGrammar) && (cmp.indexOfNode==this.in
dexOfNode) && (cmp.extSymbol.equals(this.extSymbol)) ){
75.             return true;
76.         }
77.         return false;
78.     }
79.     @Override
80.     public int hashCode() {
81.         String hash = indexOfGrammar+","+indexOfNode+","+extSymbol;
82.         return hash.hashCode();
83.     }
84. } //项目
85. class projectSet { //项目集(闭包) 重写了 equals 和 hashCode 再加上是 set 存储 判断是不是生成了
重复的对象
86.     Set<project> projects; //集合
87.     Set<Integer> indexOfProjects; //
88.     Map<String,Integer> sons; //孩子 S 即 通过 String 可以跳转到 Integer 下标的另一个集合
89.     public projectSet(){
90.         projects = new HashSet<>();
91.         indexOfProjects = new HashSet<>();
92.         sons = new HashMap<>();
93.     }
94.     public boolean add(project project){
95.         if(projectList.contains(project)){
96.             indexOfProjects.add(projectList.indexOf(project));
97.             return projects.add(project);
98.         }
99.         else {
100.             return false;
101.         }
102.     }
103.     public Set<project> getSet() {
104.         return projects;
105.     }
106.     private String toCompare(){
107.         StringBuffer sb = new StringBuffer();
108.         projects.forEach(S->sb.append(S.toString()));
109.         return sb.toString(); //比较用 set 相同即可认为是相同的集合
110.     }
111.     @Override
112.     public String toString() {
113.         return cProjectSets.indexOf(this)+" ";
114.     }
115.     @Override
116.     public boolean equals(Object obj) {
117.         if(!obj.getClass().equals(this.getClass())){
118.             return false;
119.         }
120.         return this.toCompare().equals(((projectSet)obj).toCompare()) ;
121.     }
122.     @Override
123.     public int hashCode() {
124.         return this.toCompare().hashCode();
125.     }
126. }
127. public Solution(String text){
128.     grammarText = text;
129.     setGrammar(text);
130.     setFirst();
131.     setProjectList();
132.     setCanonicalCollection();
133.     setActionAndGOTOtable();
134.     System.out.println("跳转表");
135.     for (projectSet projectSet : cProjectSets) {
136.         System.out.println(cProjectSets.indexOf(projectSet));
137.         projectSet.getSet().forEach(System.out::print);
138.         System.out.println("\n");
139.         projectSet.sons.keySet().forEach(S-> System.out.print("[ "+S+"=>"+projectSet.son
s.get(S)+" "+ " ]"));
140.         System.out.println("\n\n");
141.     }
142. }
143. private void setGrammar(String text){
144.     for (String s : text.replaceAll(" ", "").split("\n")) {
145.         for (String s1 : s.split("->")[1].split("\\|")) {
146.             String [] gram = {s.split("->")[0],s1};
147.             grammar.add(gram);
148.         }
149.     }
150.     for (String[] strings : grammar) {
151.         System.out.println(strings[0]+"->"+strings[1]);
152.     } //读取文法
153.     System.out.println();
154.     for (String[] strings : grammar) {
155.         nonTerminal.add(strings[0]);
156.         terminal.addAll(Arrays.asList(strings[1].split(" ")));
157.     }

```

```

158.         nonTerminal.forEach(S->terminal.remove(S));
159.
160.         allSymbol.addAll(terminal);
161.         allSymbol.addAll(nonTerminal);
162.         System.out.println("非终结符"+nonTerminal);
163.         System.out.println("终结符"+terminal);
164.         System.out.println();
165.     }
166.     private void setFirst(){
167.         nonTerminal.forEach(S->First.put(S,new HashSet<>()));
168.         terminal.forEach(S->First.put(S,new HashSet<>()));
169.         terminal.forEach(S->First.get(S).add(S));//终结符的 First 集是本身
170.         int FirstSize = 0;
171.         do{
172.             FirstSize = 0;
173.             for (String s1 : First.keySet()) {
174.                 FirstSize+=First.get(s1).size();
175.             }//记录原本大小
176.             for (String[] strings : grammar) {
177.                 String lam = strings[1];
178.                 String G = strings[0];
179.                 setSingleFirst(lam,G);//计算 First 集过程
180.             }
181.             for (String s1 : First.keySet()) {
182.                 FirstSize-=First.get(s1).size();
183.             }//计算修改后大小
184.         }while (FirstSize != 0);//如果大小不在变化 则停下
185.         System.out.println("First");
186.         for (String s : First.keySet()) {
187.             System.out.println(s+" "+First.get(s));
188.         }
189.     }
190.     private void setSingleFirst(String lam,String G){
191.         String first = lam.substring(0,1);
192.         if(terminal.contains(first)){//终结符
193.             First.get(G).add(first);
194.         }
195.         else if(first.equals("ε")){//符号空
196.             First.get(G).add("ε");
197.         }
198.         else if(nonTerminal.contains(first)){//非终结符
199.             First.get(G).addAll(First.get(first).stream().filter(S->!S.equals("ε")).collect
200. (Collectors.toSet()));
201.             if(First.get(first).contains("ε")){//是否可以推出空
202.                 setSingleFirst(lam.substring(1),G);//扫描下一个
203.             }
204.         }
205.         else {
206.             System.out.println("ERROR");
207.         }
208.     }
209.     private void setProjectList(){
210.         projectList.add(new project(0,0,"#"));
211.         projectList.add(new project(0,1,"#"));//开始符号 特殊 手动添加
212.         for (int indexOfGrammar = 0; indexOfGrammar < grammar.size(); indexOfGrammar++) {
213.             for (int indexOfNode = 0; indexOfNode <= grammar.get(indexOfGrammar)[1].length(
214. ) ; indexOfNode++) {
215.                 String A = grammar.get(indexOfGrammar)[0];//A->BC A
216.                 Set<String> a = new HashSet<>();
217.                 for (String[] strings : grammar) {
218.                     if(strings[1].contains(A)){
219.                         int index = strings[1].indexOf(A)+1;
220.                         String sub = strings[1].substring(index);
221.                         a.addAll(First(sub));
222.                     }
223.                 }
224.                 for (String s : a) {
225.                     projectList.add( new project(indexOfGrammar,indexOfNode,s) );
226.                 }
227.             }
228.             System.out.println("项目 s");
229.             for (int i = 0; i < projectList.size(); i++) {
230.                 System.out.println(i+ " : " +projectList.get(i));
231.             }
232.         }//读取项目
233.     }
234.     private projectSet extendSingleClosure(projectSet closure) {
235.         Stack<project> stack = new Stack<>();
236.         closure.getSet().forEach(stack::push);
237.         while(!stack.empty()){
238.             project top = stack.pop();
239.             String B = top.getFirstSymbolAfterNode();
240.             closure.projects.add(top);
241.             if(nonTerminal.contains(B)){//如果是 A->...•B...
242.                 List<String[]> GsHeadIsB = grammar.stream().filter(G->G[0].equals(B)).coll
243. ect(Collectors.toList());//每个左边是 B 的产生式
244.                 String beta = top.getRestStringAfterFirst();//获得 B 之后的部分

```

[illegible]

```

325.         for (project project : projectSet.getSet()) {
326.             if (project.getFirstSymbolAfterNode().equals("")) {
327.                 ActionTable.put(new point(projectSet.toString(), project.getExtSymbol())
, "r" + project.indexOfGrammar);
328.             }
329.         }
330.     } // <2>
331.     for (project project : projectList) {
332.         if (project.getHead().equals(startSymbol) && project.getFirstSymbolAfterNode().e
quals("")) {
333.             for (projectSet projectSet : cProjectSets) {
334.                 if (projectSet.getSet().contains(project)) {
335.                     int k = cProjectSets.indexOf(projectSet);
336.                     ActionTable.put(new point(String.valueOf(k), "#"), "acc");
337.                 }
338.             }
339.         }
340.     } // <3>
341.     for (String A : nonTerminal) {
342.         for (int k = 0; k < cProjectSets.size(); k++) {
343.             int j = cProjectSets.indexOf(GO(cProjectSets.get(k), A));
344.             if (j != -1) {
345.                 GOTO.put(new point(String.valueOf(k), A), String.valueOf(j));
346.             }
347.         }
348.     } // <4>
349.     int len = cProjectSets.size(); // C 集的 SIZE
350.     for (int i = 0; i < len; i++) {
351.         for (String s : nonTerminal) {
352.             if (!ActionTable.containsKey(new point(String.valueOf(i), s))) {
353.                 ActionTable.put(new point(String.valueOf(i), s), "err");
354.             }
355.         }
356.         for (String s : terminal) {
357.             if (!ActionTable.containsKey(new point(String.valueOf(i), s))) {
358.                 ActionTable.put(new point(String.valueOf(i), s), "err");
359.             }
360.         }
361.         if (!ActionTable.containsKey(new point(String.valueOf(i), "#"))) {
362.             ActionTable.put(new point(String.valueOf(i), "#"), "err");
363.         }
364.     } // 空位置打上 err
365.     Map<point, String> adder = new HashMap<>(); // Action 和 GOTO 合并为一个 方便显示
366.     adder.putAll(ActionTable);
367.     adder.putAll(GOTO);
368.     List<String> tableSymbol = new ArrayList<>();
369.     tableSymbol.addAll(terminal);
370.     tableSymbol.add("#"); // 加上 #
371.     tableSymbol.addAll(nonTerminal);
372.     tableSymbol.remove(startSymbol); // 删掉 S`
373.     ActionAndGoTo = new String[len][tableSymbol.size()+1];
374.     for (int i = 0; i < len; i++) {
375.         ActionAndGoTo[i][0] = String.valueOf(i);
376.         for (int j = 0; j < tableSymbol.size(); j++) {
377.             ActionAndGoTo[i][j+1] = adder.get(new point(String.valueOf(i), tableSymbol.g
et(j)));
378.         }
379.     }
380.     System.out.print("\t");
381.     for (String s : tableSymbol) {
382.         System.out.print(s + "\t");
383.     }
384.     System.out.println();
385.     for (String[] strings : ActionAndGoTo) {
386.         for (String string : strings) {
387.             System.out.print(string + "\t");
388.         }
389.         System.out.println();
390.     }
391. } // 创建 Action 和 GOTO 表
392. public String[][] getActionAndGoTo() { // 返回分析表
393.     return ActionAndGoTo;
394. }
395. public String[] getHeader() {
396.     List<String> tableSymbol = new ArrayList<>();
397.     tableSymbol.addAll(terminal);
398.     tableSymbol.add("#"); // 加上 #
399.     tableSymbol.addAll(nonTerminal);
400.     tableSymbol.remove(startSymbol); // 删掉 S`
401.     String[] header = new String[tableSymbol.size()+1];
402.     header[0] = "";
403.     for (int i = 0; i < tableSymbol.size(); i++) {
404.         header[i+1] = tableSymbol.get(i);
405.     }
406.     return header;
407. }
408. public Vector<String[]> analyse(String text) {

```

```

409.     Vector<String[]> processRecord = new Vector<>();
410.     MyStack inputStack = new MyStack();//输入串
411.     MyStack symbolStack = new MyStack();//符号栈
412.     MyStack statusStack = new MyStack();//状态栈
413.     inputStack.push("#");
414.     inputStack.push(new StringBuffer(text).reverse().toString().split(""));
415.     symbolStack.push("#");
416.     statusStack.push("0");
417.     Boolean iFlag = true;
418.     int count = 0;
419.     System.out.println("分析开始=====");
420.     while (iFlag){
421.         String[] currentStep = new String[5];
422.         currentStep[1] = statusStack.toString()+"\t";
423.         currentStep[2] = symbolStack.toString()+"\t";
424.         currentStep[3] = new StringBuffer(inputStack.toString()).reverse()+"\t";
425.         processRecord.add(currentStep);
426.         String i = statusStack.getTop();//状态栈
427.         String a = inputStack.getTop();//输入串
428.         System.out.println("状态栈顶:["+i+"]");
429.         System.out.println("输入栈顶:["+a+"]");
430.         String action = ActionTable.get(new point(i,a));
431.         if(action == null){
432.             System.out.println("nullActErr");
433.             currentStep[4] = "nullActErr";
434.             iFlag = false;
435.         }
436.         else if(action.equals("err")){
437.             System.out.println("EqualsErrn");
438.             currentStep[4] = "EqualsError";
439.             iFlag = false;
440.         }
441.         else if(action.equals("acc")){
442.             System.out.println("成功!");
443.             currentStep[4] = "成功";
444.             iFlag = false;
445.         }
446.         else if(action.substring(0,1).equals("s")){//状态入栈
447.             Integer j = Integer.valueOf(action.substring(1));
448.             System.out.println("当前 S"+j);
449.             statusStack.push(String.valueOf(j)); //j 入状态栈
450.             symbolStack.push(inputStack.pop()); //a 到文法符号栈
451.             currentStep[4] = "Action["+i+", "+a+"]="+action+", 状态"+a+"入栈";
452.         }
453.         else if(action.substring(0,1).equals("r")){//规约 然后 GOTO 入栈
454.             Integer index = Integer.valueOf(action.substring(1)); //用 index 产生式规约
455.             for (int times = 0; times < grammar.get(index)[1].length(); times++) {
456.                 symbolStack.pop();
457.                 statusStack.pop();
458.             } //符号栈中取出文法的右侧的长度的字符
459.             symbolStack.push(grammar.get(index)[0]); //再压栈文法左边的符号
460.
461.             String nextSta = GOTO.get(new point(statusStack.getTop(),symbolStack.getTop
462.             currentStep[4] = action+"."+grammar.get(index)[0]+"->" +grammar.get(index)[1
463.             ]+"归约,GOTO("+statusStack.getTop()+", "+statusStack.getTop()+")="+nextSta+"入栈";
464.             statusStack.push(nextSta);
465.         }
466.         else {
467.             System.out.println("err");
468.             currentStep[4] = "ERROR";
469.             iFlag = false;
470.         }
471.         currentStep[0] = count+"\t";
472.         count++;
473.
474.         System.out.println("=====");
475.     }
476.     return processRecord;
477. }
478. Set<String> First(String beta,String a){
479.     Set<String> res = new HashSet<>();
480.     if(beta.length() == 0){
481.         res.add(a);
482.         return res;
483.     }
484.     else {
485.         res.addAll(First(beta));
486.         if(First(beta).contains("ε")){
487.             res.add(a);
488.             res.remove("ε");
489.         }
490.     }
491.     return res;
492. }
493. Set<String> First(String s){//单个字串的 First 集

```

```

494.         Set<String> res = new HashSet<>();
495.         if(s.length() == 0){
496.             res.add("#");
497.             return res;
498.         }
499.         for (String symbol : s.split("")) {
500.             if(First.containsKey(symbol)){
501.                 res.addAll(First.get(symbol));
502.                 if(!First.get(symbol).contains("ε")){//如果 当前的字 可以推出空 看向字串的下一
个字
503.                     break;
504.                 }
505.             }
506.         }
507.         Boolean elicitNull = true;
508.         for (String symbol : s.split("")){
509.             if(First.containsKey(symbol) && !First.get(symbol).contains("ε")){
510.                 elicitNull = false;
511.                 break;
512.             }
513.         }
514.         if(!elicitNull){
515.             res.remove("ε");
516.             //只有 所有的字 都能推出空 这个字串才可以推出空
517.             return res;
518.         }
519.     }
520.
521.     class point {
522.         String head, tail;
523.
524.         point(String head, String tail) {
525.             this.head = head;
526.             this.tail = tail;
527.         }
528.
529.         public String getHead() {
530.             return head;
531.         }
532.
533.         public String getTail() {
534.             return tail;
535.         }
536.
537.         @Override
538.         public int hashCode() {
539.             return (head + "->" + tail).hashCode();
540.         }
541.
542.         @Override
543.         public boolean equals(Object o) {
544.             if (o.getClass() != this.getClass()) {
545.                 return false;
546.             } else {
547.                 return ((point) o).getHead().equals(this.head) && ((point) o).getTail().equals(
this.tail);
548.             }
549.         }
550.
551.     }
552.
553.     class MyStack {
554.         List<String> s;
555.
556.         MyStack() {
557.             s = new LinkedList<>();
558.         }
559.
560.         void push(String value) {
561.             s.add(value);
562.         }
563.
564.         void push(String... values) {
565.             for (String value : values) {
566.                 push(value);
567.             }
568.         }
569.
570.         String pop() {
571.             return s.remove(s.size() - 1);
572.         }
573.
574.         String getTop() {
575.             return s.get(s.size() - 1);
576.         }
577.
578.         @Override

```



```

579.     public String toString() {
580.         StringBuffer sb = new StringBuffer();
581.         for (String value : s) {
582.             sb.append(value);
583.         }
584.         return sb.toString();
585.     }
586.
587.     public Boolean isEmpty() {
588.         return s.size() == 0;
589.     }
590. }

```

## GUI.java

```

1.  package 实验三_LR1 分析法;
2.
3.  import javax.swing.*;
4.  import java.awt.*;
5.  import java.awt.event.ActionEvent;
6.  import java.awt.event.ActionListener;
7.  import java.util.Vector;
8.  import javax.swing.table.JTableHeader;
9.  import javax.swing.table.TableColumn;
10. import java.util.*;
11.
12. class Windows extends JFrame {
13.     Solution sol;
14.     JButton clear,confirm;
15.     JTextArea grammarTextArea,inputTextArea,projectArea;
16.     JScrollPane tablePane;
17.     JTabbedPane resultTable;
18.     JScrollPane projectList;
19.
20.     Windows(){
21.         setVisible(false);
22.         try{
23.             setIconImage(new ImageIcon("bilibili.PNG").getImage());
24.             Font f = new Font("Yahei Consolas Hybrid",Font.PLAIN,16);
25.             String names[]={ "MenuBar","Menu","MenuItem", "TextArea", "Button", "ScrollPa
26. ne", "Table", "TabbedPane"};
27.             for (String item : names) {
28.                 UIManager.put(item+ ".font",f);
29.             }
30.             UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
31.
32.         }catch(Exception e){}
33.         sol = new Solution("S`->E\n"+
34.             "E->E+T\n" +
35.             "E->T\n" +
36.             "T->T*F\n" +
37.             "T->F\n" +
38.             "F->(E)\n" +
39.             "F->i");
40.         init();
41.         setSize(1280,720);//初始大小
42.         setLocation(100,80);//初始位置
43.         setVisible(true);//是否可视
44.         setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);//X 退出
45.     }
46.     void init(){
47.         setTitle("LR(1)分析法");
48.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49.         setVisible(true);
50.         setResizable(false);
51.         setLayout(null);
52.         initGrammarText();
53.         initActTable();
54.         initButton();
55.         initInputArea();
56.         initResultTable();
57.         initProjectPane();
58.     }
59.     void initGrammarText(){
60.         grammarTextArea = new JTextArea("当前使用文法:\n"+sol.grammarText);
61.         grammarTextArea.setEditable(false);
62.         JScrollPane textAreaRollPane = new JScrollPane(grammarTextArea);
63.         textAreaRollPane.setBounds(10,10,150,200);
64.         add(textAreaRollPane);

```

```

63.     }
64.     private void initActTable(){
65.         tablePane = new JScrollPane();
66.         tablePane.setBounds(10,220,500,450);
67.         add(tablePane);
68.         updateActTable();
69.     }
70.     private void initButton(){
71.         clear = new JButton("清空");
72.         clear.setBounds(1120,100,110,50);
73.         add(clear);
74.         clear.addActionListener(actionEvent -> {
75.             inputTextArea.setText("");
76.             resultTable.removeAll();
77.         });
78.         confirm = new JButton("确认");
79.         confirm.setBounds(1120,160,110,50);
80.         add(confirm);
81.         confirm.addActionListener(actionEvent -> {
82.             resultTable.removeAll();
83.             for (String inputText : inputTextArea.getText().split("\n")) {
84.                 Vector<String[]> result = sol.analyse(inputText);
85.                 String[] head = {"步骤", "状态栈", "符号栈", "输入串", "动作说明"};
86.                 String[][] data = new String[result.size()][5];
87.                 int i = 0;
88.                 for (String[] strings : result) {
89.                     data[i] = strings;
90.                     i++;
91.                 }
92.                 JTable singleResult = new JTable(data,head);
93.                 FitTableColumns(singleResult);
94.                 singleResult.setRowHeight(30);
95.                 JScrollPane resultTablePane = new JScrollPane(singleResult);
96.                 resultTable.addTab(" "+inputText+" ",resultTablePane);
97.             }
98.         });
99.         JButton G1 = new JButton("G1");
100.        G1.setBounds(410,10,100,35);
101.        add(G1);
102.        G1.addActionListener(actionEvent -> {
103.            updateGrammar("S' ->E\nE->E+T\nE->T\nT->T*F\nT->F\nF->(E)\nF->i");
104.        });
105.        JButton G2 = new JButton("G2");
106.        G2.setBounds(410,65,100,35);
107.        add(G2);
108.        G2.addActionListener(actionEvent -> {
109.            updateGrammar("S' ->E\nE->E+T | T\nT->T*F | F\nF->P+T | P\nP->(E) | i\n");
110.        });
111.        JButton G3 = new JButton("G3");
112.        G3.setBounds(410,120,100,35);
113.        add(G3);
114.        G3.addActionListener(actionEvent -> {
115.            updateGrammar("S' ->S\nS->aAd\nS->bAc\nS->aec\nS->bed\nA->e");
116.        });
117.        JButton more = new JButton("手动输入");
118.        more.setBounds(410,175,100,35);
119.        add(more);
120.        class MyDialog extends JDialog implements ActionListener{
121.            JTextArea input;
122.            JButton confirm, cancel;
123.            String title;
124.            MyDialog(){
125.                setLayout(null);
126.                setResizable(false);
127.                setIconImage(new ImageIcon("bilibili.PNG").getImage());
128.                setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
129.                setTitle("输入语法");
130.                input=new JTextArea();
131.                JScrollPane jScrollPane = new JScrollPane(input);
132.                jScrollPane.setBounds(10,10,265,200);
133.                add(jScrollPane);
134.                class confirmListener implements ActionListener{
135.                    @Override
136.                    public void actionPerformed(ActionEvent e){
137.                        updateGrammar(input.getText());
138.                        setVisible(false);
139.                    }
140.                }
141.                confirm=new JButton("确定");
142.                confirm.addActionListener(new confirmListener());
143.                confirm.setBounds(195,220,80,30);
144.                add(confirm);
145.                class cancellistener implements ActionListener{
146.                    @Override
147.                    public void actionPerformed(ActionEvent e){
148.                        setVisible(false);
149.                    }

```

```

150.         }
151.         cancel=new JButton("取消");
152.         cancel.addActionListener(new cancelListener());
153.         cancel.setBounds(105,220,80,30);
154.         add(cancel);
155.         setBounds(600,260,300,300);
156.         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
157.     }
158.     public void actionPerformed(ActionEvent e){
159.         setVisible(true);
160.     }
161. }
162. more.addActionListener(new MyDialog());
163. }
164. private void initInputArea(){
165.     inputTextArea = new JTextArea();
166.     inputTextArea.setLineWrap(true);
167.     JScrollPane inputAreaPane = new JScrollPane(inputTextArea);
168.     inputAreaPane.setBounds(520,10,580,200);
169.     add(inputAreaPane);
170. }
171. private void initResultTable(){
172.     resultTable = new JTabbedPane();
173.     resultTable.setBounds(520,220,735,450);
174.     add(resultTable);
175. }
176. private void initProjectPane(){
177.     projectArea = new JTextArea();
178.     projectList = new JScrollPane(projectArea);
179.     projectList.setBounds(170,10,230,200);
180.     add(projectList);
181.     updateProjectPane();
182. }
183. }
184. private void updateProjectPane(){
185.     StringBuffer sb = new StringBuffer();
186.     int count = 0;
187.     for (Solution.projectSet projectSet : sol.cProjectSets) {
188.         sb.append( (count++ )+" : ");
189.         for (Solution.project project : projectSet.getSet()) {
190.             sb.append(project+" ");
191.         }
192.         sb.append("\n");
193.     }
194.     count = 0;
195.     for (Solution.projectSet projectSet : sol.cProjectSets) {
196.         sb.append( (count++ )+" : ");
197.         for (String s : sol.allSymbol) {
198.             int index = projectSet.sons.get(s);
199.             if(index !=-1){
200.                 sb.append(s+"->" +index);
201.             }
202.         }
203.         sb.append("\n");
204.     }
205.     projectArea.setText(sb.toString());
206.     projectList.updateUI();
207. }
208. private void updateActTable(){
209.     String[] head =sol.getHeader();
210.     String [][] data = sol.getActionAndGoTo();
211.     for (int i = 0; i < data.length; i++) {
212.         for (int j = 0; j < data[i].length; j++) {
213.             if(data[i][j].equals("errn")){
214.                 data[i][j] = "";
215.             }
216.         }
217.     }
218.     JTable actGoTable;
219.     actGoTable = new JTable(data,head);
220.     actGoTable.setRowHeight(30);
221.     tablePane.setViewportViewView(actGoTable);
222.     tablePane.updateUI();
223. }//更新 ActGo 表
224. public void updateGrammar(String s){
225.     sol = new Solution(s);
226.     grammarTextArea.setText("当前使用文法:\n"+sol.grammarText);
227.     updateActTable();
228.     updateProjectPane();
229. }//更新语法
230. public void FitTableColumns(JTable myTable) {
231.     JTableHeader header = myTable.getTableHeader();
232.     int rowCount = myTable.getRowCount();
233.     Enumeration columns = myTable.getColumnModel().getColumns();
234.     while (columns.hasMoreElements()) {
235.         TableColumn column = (TableColumn) columns.nextElement();
236.         int col = header.getColumnModel().getColumnIndex(column.getIdentifier());

```

```

237.         int width = (int) myTable.getTableHeader().getDefaultRenderer()
238.             .getTableCellRendererComponent(myTable, column.getIdentifier()
239.                 , false, false, -1, col).getPreferredSize().getWidth();
240.         for (int row = 0; row < rowCount; row++) {
241.             int preferredWidth = (int) myTable.getCellRenderer(row, col).getTableCellRen
242. dererComponent(myTable,
243.                 myTable.getValueAt(row, col), false, false, row, col).getPreferredS
244. ize().getWidth();
245.             width = Math.max(width, preferredWidth);
246.         }
247.         header.setResizingColumn(column);
248.         column.setWidth(width + myTable.getInterCellSpacing().width+10);
249.     }

```

运行结果：

LR(1)分析法

当前使用文法:

```

S -> E
E -> E+T
E -> T
T -> T*F
T -> F
F -> (E)
F -> i

```

21: [T->T\*F, #] [T->T\*F, +] [T->^

0: T->1E->2F->3(>4i->5

1: \*->20

2: +->18

3:

4: T->6E->7F->8(>9i->10

5:

6: \*->15

7: ^->17, ^->12

8:

9:

10:

11:

12:

13:

14:

15:

16:

17:

18:

19:

20:

21:

22:

23:

24:

25:

26:

27:

28:

29:

30:

31:

32:

33:

34:

35:

36:

37:

38:

39:

40:

41:

42:

43:

44:

45:

46:

47:

48:

49:

50:

51:

52:

53:

54:

55:

56:

57:

58:

59:

60:

61:

62:

63:

64:

65:

66:

67:

68:

69:

70:

71:

72:

73:

74:

75:

76:

77:

78:

79:

80:

81:

82:

83:

84:

85:

86:

87:

88:

89:

90:

91:

92:

93:

94:

95:

96:

97:

98:

99:

100:

101:

102:

103:

104:

105:

106:

107:

108:

109:

110:

111:

112:

113:

114:

115:

116:

117:

118:

119:

120:

121:

122:

123:

124:

125:

126:

127:

128:

129:

130:

131:

132:

133:

134:

135:

136:

137:

138:

139:

140:

141:

142:

143:

144:

145:

146:

147:

148:

149:

150:

151:

152:

153:

154:

155:

156:

157:

158:

159:

160:

161:

162:

163:

164:

165:

166:

167:

168:

169:

170:

171:

172:

173:

174:

175:

176:

177:

178:

179:

180:

181:

182:

183:

184:

185:

186:

187:

188:

189:

190:

191:

192:

193:

194:

195:

196:

197:

198:

199:

200:

201:

202:

203:

204:

205:

206:

207:

208:

209:

210:

211:

212:

213:

214:

215:

216:

217:

218:

219:

220:

221:

222:

223:

224:

225:

226:

227:

228:

229:

230:

231:

232:

233:

234:

235:

236:

237:

238:

239:

240:

241:

242:

243:

244:

245:

246:

247:

248:

249:

250:

251:

252:

253:

254:

255:

256:

257:

258:

259:

260:

261:

262:

263:

264:

265:

266:

267:

268:

269:

270:

271:

272:

273:

274:

275:

276:

277:

278:

279:

280:

281:

282:

283:

284:

285:

286:

287:

288:

289:

290:

291:

292:

293:

294:

295:

296:

297:

298:

299:

300:

301:

302:

303:

304:

305:

306:

307:

308:

309:

310:

311:

312:

313:

314:

315:

316:

317:

318:

319:

320:

321:

322:

323:

324:

325:

326:

327:

328:

329:

330:

331:

332:

333:

334:

335:

336:

337:

338:

339:

340:

341:

342:

343:

344:

345:

346:

347:

348:

349:

350:

351:

352:

353:

354:

355:

356:

357:

358:

359:

360:

361:

362:

363:

364:

365:

366:

367:

368:

369:

370:

371:

372:

373:

374:

375:

376:

377:

378:

379:

380:

381:

382:

383:

384:

385:

386:

387:

388:

389:

390:

391:

392:

393:

394:

395:

396:

397:

398:

399:

400:

401:

402:

403:

404:

405:

406:

407:

408:

409:

410:

411:

412:

413:

414:

415:

416:

417:

418:

419:

420:

421:

422:

423:

424:

425:

426:

427:

428:

429:

430:

431:

432:

433:

434:

435:

436:

437:

438:

439:

440:

441:

442:

443:

444:

445:

446:

447:

448:

449:

450:

451:

452:

453:

454:

455:

456:

457:

458:

459:

460:

461:

462:

463:

464:

465:

466:

467:

468:

469:

470:

471:

472:

473:

474:

475:

476:

477:

478:

479:

480:

481:

482:

483:

484:

485:

486:

487:

488:

489:

490:

491:

492:

493:

494:

495:

496:

497:

498:

499:

500:

501:

502:

503:

504:

505:

506:

507:

508:

509:

510:

511:

512:

513:

514:

515:

516:

517:

518:

519:

520:

521:

522:

523:

524:

525:

526:

527:

528:

529:

530:

531:

532:

533:

534:

535:

536:

537:

538:

539:

540:

541:

542:

543:

544:

545:

546:

547:

548:

549:

550:

551:

552:

553:

554:

555:

556:

557:

558:

559:

560:

561:

562:

563:

564:

565:

566:

567:

568:

569:

570:

571:

572:

573:

574:

575:

576:

577:

578:

579:

580:

581:

582:

583:

584:

585:

586:

587:

588:

589:

590:

591:

592:

593:

594:

595:

596:

597:

598:

599:

600:

601:

602:

603:

604:

605:

606:

607:

608:

609:

610:

611:

612:

613:

614:

615:

616:

617:

618:

619:

620:

621:

622:

623:

624:

625:

626:

627:

628:

629:

630:

631:

632:

633:

634:

635:

636:

637:

638:

639:

640:

641:

642:

643:

644:

645:

646:

647:

648:

649:

650:

651:

652:

653:

654:

655:

656:

657:

658:

659:

660:

661:

662:

663:

664:

665:

666:

667:

668:

669:

670:

671:

672:

673:

674:

675:

676:

677:

678:

679:

680:

681:

682:

683:

684:

685:

686:

687:

688:

689:

690:

691:

692:

693:

694:

695:

696:

697:

698:

699:

700:

701:

702:

703:

704:

705:

706:

707:

708:

709:

710:

711:

712:

713:

714:

715:

716:

717:

718:

719:

720:

721:

722:

723:

724:

725:

726:

727:

728:

729:

730:

731:

732:

733:

734:

735:

736:

737:

738:

739:

740:

741:

742:

743:

744:

745:

746:

747:

748:

749:

750:

751:

752:

753:

754:

755:

756:

757:

758:

759:

760:

761:

762:

763:

764:

765:

766:

767:

768:

769:

770:

771:

772:

773:

774:

775:

776:

777:

778:

779:

780:

781:

782:

783:

784:

785:

786:

787:

788:

789:

790:

791:

792:

793:

794:

795:

796:

797:

798:

799:

800:

801:

802:

803:

804:

805:

806:

807:

808:

809:

810:

811:

812:

813:

814:

815:

816:

817:

818:

819:

820:

821:

822:

823:

824:

825:

826:

827:

828:

829:

830:

831:

832:

833:

834:

835:

836:

837:

838:

839:

840:

841:

842:

843:

844:

845:

846:

847:

848:

849:

850:

851:

852:

853:

854:

855:

856:

857:

858:

859:

860:

861:

862:

863:

864:

865:

866:

867:

868:

869:

870:

871:

872:

873:

874:

875:

876:

877:

878:

879:

880:

881:

882:

883:

884:

885:

886:

887:

888:

889:

890:

891:

892:

893:

894:

895:

896:

897:

898:

899:

900:

901:

902:

903:

904:

905:

906:

907:

908:

909:

910:

911:

912:

913:

914:

915:

916:

917:

918:

919:

920:

921:

922:

923:

924:

925:

926:

927:

928:

929:

930:

931:

932:

933:

934:

935:

936:

937:

938:

939:

940:

941:

942:

943:

944:

945:

946:

947:

948:

949:

950:

951:

952:

953:

954:

955:

956:

957:

958:

959:

960:

961:

962:

963:

964:

965:

966:

967:

968:

969:

970:

971:

972:

973:

974:

975:

976:

977:

978:

979:

980:

981:

982:

983:

984:

985:

986:

987:

988:

989:

990:

991:

992:

993:

994:

995:

996:

997:

998:

999:

1000:

1001:

1002:

1003:

1004:

1005:

1006:

1007:

1008:

1009:

1010:

1011:

1012:

1013:

1014:

1015:

1016:

1017:

1018:

1019:

1020:

1021:

1022:

1023:

1024:

1025:

1026:

1027:

1028:

1029:

1030:

1031:

1032:

1033:

1034:

1035:

1036:

1037:

1038:

1039:

1040:

1041:

1042:

1043:

1044:

1045:

1046:

1047:

1048:

1049:

1050:

1051:

1052:

1053:

1054:

1055:

1056:

1057:

1058:

1059:

1060:

1061:

1062:

1063:

1064:

1065:

1066:

1067:

1068:

1069:

1070:

1071:

1072:

1073:

1074:

1075:

1076:

1077:

1078:

1079:

1080:

1081:

1082:

1083:

1084:

1085:

1086:

1087:

1088:

1089:

1090:

1091:

1092:

1093:

1094:

1095:

1096:

1097:

1098:

1099:

1100:

1101:

1102:

1103:

1104:

1105:

1106:

1107:

1108:

1109:

1110:

1111:

1112:

1113:

1114:

1115:

1116:

1117:

1118:

1119:

1120:

1121:

1122:

1123:

1124:

1125:

1126:

1127:

1128:

1129:

1130:

1131:

1132:

1133:

1134:

1135:

1136:

1137:

1138:

1139:

1140:

1141:

1142:

1143:

1144:

1145:

1146:

1147:

1148:

1149:

1150:

1151:

1152:

1153:

1154:

1155:

1156:

1157:

1158:

1159:

1160:

1161:

1162:

1163:

1164:

1165:

1166:

1167:

1168:

1169:

1170:

1171:

1172:

1173:

1174:

1175:

1176:

1177:

1178:

1179:

1180:

1181:

1182:

1183:

1184:

1185:

1186:

1187:

1188:

1189:

1190:

1191:

1192:

1193:

1194:

1195:

1196:

1197:

1198:

1199:

1200:

1201:

1202:

1203:

1204:

1205:

1206:

1207:

1208:

1209:

1210:

1211:

1212:

1213:

1214:

1215:

1216:

1217:

1218:

1219:

1220:

1221:

1222:

1223:

1224:

1225:

1226:

1227:

1228:

1229:

1230:

1231:

1232:

1233:

1234:

1235:

1236:

1237:

1238:

1239:

1240:

1241:

1242:

1243:

1244:

1245:

1246:

1247:

1248:

1249:

1250:

1251:

1252:

1253:

1254:

1255:

1256:

1257:

1258:

1259:

1260:

1261:

1262:

1263:

1264:

1265:

1266:

1267:

1268:

1269:

1270:

1271:

1272:

1273:

1274:

1275:

1276:

1277:

1278:

1279:

1280:

1281:

1282:

1283:

1284:

1285:

1286:

1287:

1288:

1289:

1290:

1291:

1292:

1293:

1294:

1295:

1296

LR(1)分析法

当前使用文法:  
S->E  
E->E+T | T  
T->T\*F | F  
F->P|F | P  
P->(E)|i

20: P->F->Z1(->T1->T2  
21:  
22: P->1T->23F->4(->5i->6  
23: \*->24  
24: P->1F->25(->5i->6  
25:  
26: P->1F->27(->5i->6  
27:

G1  
G2  
G3  
手动输入

i+(i\*(i+i))\*(i+(i\*(i+i)))+(i+(i\*(i+i)))\*i+(i\*(i+i))

清空  
确认

	↑	(	)	i	*	+	=	P	T	E	F	
0			s5		s6				1	2	3	4
1		s26			r6	r6	r6					
2					s24	r2	r2					
3						s22	acc					
4					r4	r4	r4					
5			s11		s12				7	8	9	10
6		r8			r8	r8	r8					
7		s20		r6		r6	r6					
8				r2		s17	r2					
9					s19		s15					
10				r4		r4	r4					
11			s11		s12				7	8	13	10
12		r8			r8	r8	r8					
13				s14		s15						

步骤	状态栈	符号栈	输入串	动作说明
131	0322581711131510	#E+(T*(E+F))#	)#	r4.T->F归约, GOTO(15,15)=16入栈
132	0322581711131516	#E+(T*(E+T))#	)#	r1.E->E+T归约, GOTO(11,11)=13入栈
133	032258171113	#E+(T*(E))#	)#	Action[13,]=s14,状态入栈
134	03225817111314	#E+(T*(E))#	)#	r7.P->(E)归约, GOTO(17,17)=7入栈
135	032258177	#E+(T*P)	)#	r6.F->P归约, GOTO(17,17)=18入栈
136	0322581718	#E+(T*F)	)#	r3.T->T*F归约, GOTO(5,5)=8入栈
137	032258	#E+(T)	)#	r2.E->T归约, GOTO(5,5)=9入栈
138	032259	#E+(E)	)#	Action[9,]=s19,状态入栈
139	03225919	#E+(E)	#	r7.P->(E)归约, GOTO(22,22)=1入栈
140	03221	#E+P	#	r6.F->P归约, GOTO(22,22)=4入栈
141	03224	#E+F	#	r4.T->F归约, GOTO(22,22)=23入栈
142	032223	#E+T	#	r1.E->E+T归约, GOTO(0,0)=3入栈
143	03	#E	#	成功

LR(1)分析法

当前使用文法:  
S->E  
E->E+T | T  
T->T\*F | F  
F->P|F | P  
P->(E)|i

20: P->F->Z1(->T1->T2  
21:  
22: P->1T->23F->4(->5i->6  
23: \*->24  
24: P->1F->25(->5i->6  
25:  
26: P->1F->27(->5i->6  
27:

G1  
G2  
G3  
手动输入

i+(i\*(i+i))\*(i+(i\*(i+i)))+(i+(i\*(i+i)))\*i+(i\*(i+i))

清空  
确认

	↑	(	)	i	*	+	=	P	T	E	F	
0			s5		s6				1	2	3	4
1		s26			r6	r6	r6					
2					s24	r2	r2					
3						s22	acc					
4					r4	r4	r4					
5			s11		s12				7	8	9	10
6		r8			r8	r8	r8					
7		s20		r6		r6	r6					
8				r2		s17	r2					
9					s19		s15					
10				r4		r4	r4					
11			s11		s12				7	8	13	10
12		r8			r8	r8	r8					
13				s14		s15						

步骤	状态栈	符号栈	输入串	动作说明
134	03225817111314	#E+(T*(E))#	)#	r7.P->(E)归约, GOTO(17,17)=7入栈
135	032258177	#E+(T*P)	)#	r6.F->P归约, GOTO(17,17)=18入栈
136	0322581718	#E+(T*F)	)#	r3.T->T*F归约, GOTO(5,5)=8入栈
137	032258	#E+(T)	)#	r2.E->T归约, GOTO(5,5)=9入栈
138	032259	#E+(E)	)#	Action[9,]=s19,状态入栈
139	03225919	#E+(E)	#	r7.P->(E)归约, GOTO(22,22)=1入栈
140	03221	#E+P	#	r6.F->P归约, GOTO(22,22)=4入栈
141	03224	#E+F	#	r4.T->F归约, GOTO(22,22)=23入栈
142	032223	#E+T	#	r1.E->E+T归约, GOTO(0,0)=3入栈
143	03	#E	#	成功

输入语法

S->S  
S->aAd  
S->bAc  
S->aec  
S->bed  
A->e

取消  
确定

## 4. 实验收获

相比较与前几次的实验，本次实验的算法相对复杂抽象，但是当理解算法并设计了合适的数据结构去存储后，各个部分功能明确，总体设计起来也就比较容易。项目集和项目都使用了重写了 `hashCode()` 和 `equals()` 方法并使用 `set` 存储来更加方便的判断重复，在计算项目集闭包和展望符跳转的时候更加便利。此外，相比较于上次实验，本次分析器直接将主控程序与语法绑定在一起，在修改语法时就不需要额外的更新语法的方法，只需创建一个新的分析器即可，更新界面上的语法相关信息也更加方便。