

**编译原理实验报告**

学生姓名 ：林天岳

学 号 ：2017217893

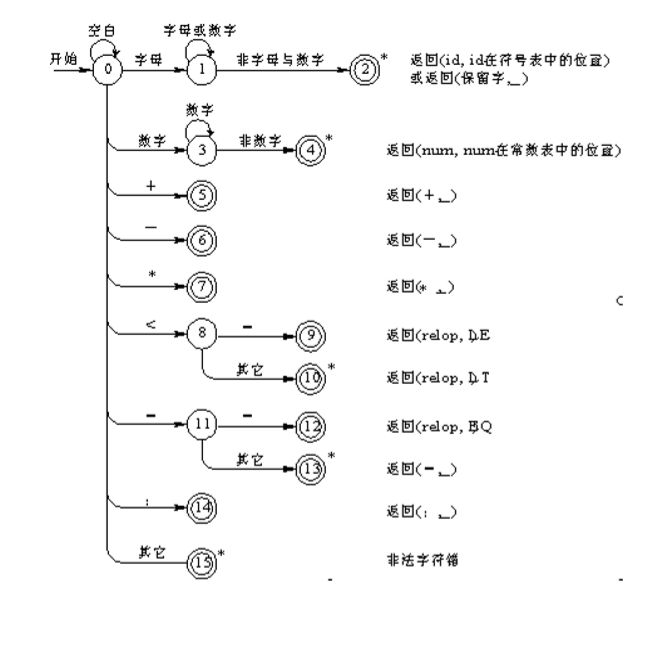
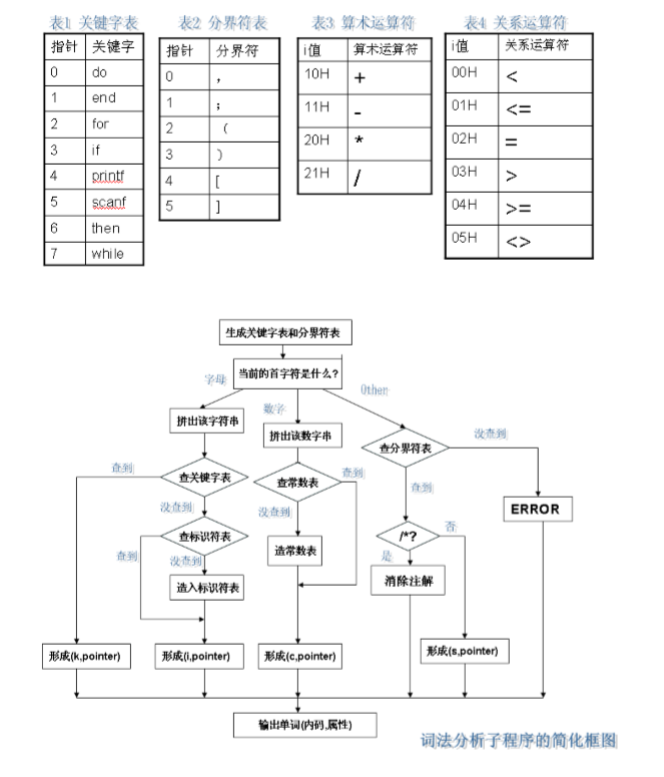
班 级 ：计算机科学与技术2017-5

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 = "ABCDEFGHIGKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";//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[] ={"<<=",">>=","&&","||","<=","|=","\*=","^=","==","++","--","/=","-=","+=","%=","!=",">=","[","]","!","%","(",")","\*","+",",","-","/",";","<","=",">"};//运算符
8. String arithmeticOperator[] = {"++","--","+","-","\*","/","%"};//算术运算符
9. String relationalOperator[] = {"<=","<",">=",">","==","!="};//关系运算符
10. String logicalOperator[] = {"&&","||","!"};//逻辑运算符
11. String delimiter[] = {";",",","(",")","[","]"};//分界符
12. String assignmentOperator[] ={"=","+=","-=","\*=","/=","%=","<<=",">>=","%=","^=","|="};//赋值运算符
14. Map<String,String> opS;//<单个运算符,种类名>
15. Map<String,String[]> KindtoArrary;//<种类名,对应的运算符数组>
17. List<Result> result = **new** ArrayList<>();//结果 保存后显示为表格
18. GUI包含一个Solution
19. 分析时 在textArea中输入需要的分析的代码
20. 或者直接打开文件读取到textArea
22. 分析 则使用Solution.Solve 返回分析结果
23. 显示在界面上
24. Solution 包含一个 Analyzer分析器
25. 调用Solve方法  传入String数组 返回分析结果
26. **for**(String line:传入的String数组){
27. result.addAll(Analyzer.LineAnalyse(line));Analyzer.LineAnalyse(line)
28. }
29. **return** result
30. LineAnalyse方法
31. **if**(当前分析的是 null,//,\n,或者Length == 0){
32. 则直接结束
33. }
34. **else** {
35. **if**(字母表含有当前头部的string){
36. **while**(是字母或者是数字){
37. 继续取出之后的部分
38. }
39. 得到了一个String
40. **if**(单词是关键字){
41. 标记为 关键字
42. }
43. **else**{
44. 标记为 标识符
45. }
46. }
47. **else** **if**(数字表含有当前头部的的string){
48. **while**(是数字或者小数点){
49. 继续取出之后的部分
50. }
51. **if**(数字之后直接追加字母){
52. 标记错误
53. brerak
54. }
55. 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
56. }
57. **else**{
58. **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
59. 标记 运算符
60. }
61. **else**{
62. 标记 错误
63. }
64. }
65. 递归处理之后的String
66. resturn  result.addAll(递归的结果);
67. }

2.算法流程图 

3.源码及测试结果

Main.java：

1. package 实验一\_\_\_词法分析设计;

4. **public** **class** Main{
5. **public** **static** **void** main(String[] args) {
6. Windows windows = **new** Windows();
7. }
8. }

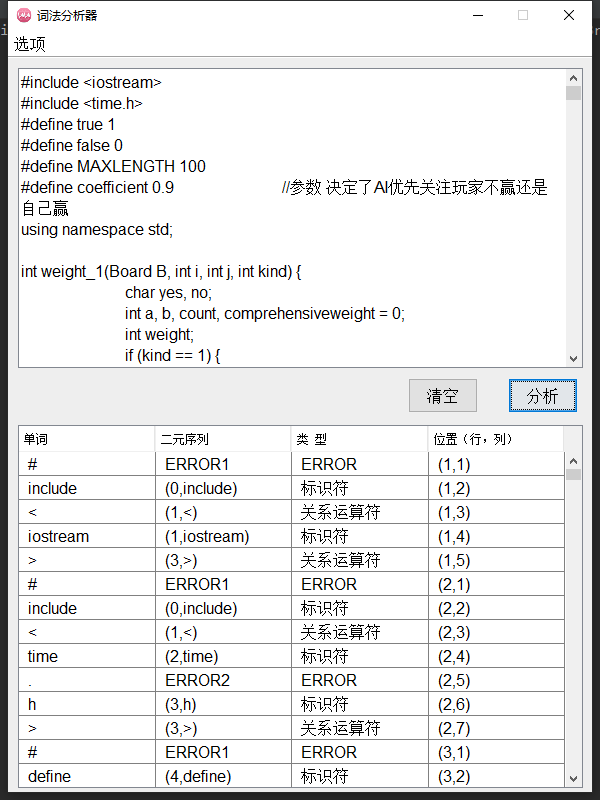
Solution.java

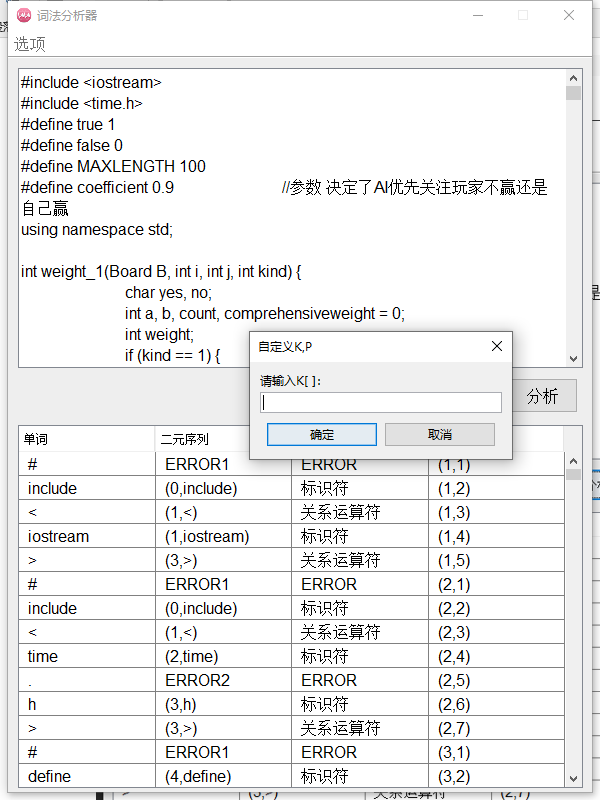
1. package 实验一\_\_\_词法分析设计;
3. import java.util.\*;
5. **class** Solution{
6. Analyzer ana = **new** Analyzer();
7. **public** List<Result> Solve(String[] lines) {
8. List<Result> res = **new** ArrayList<>();
9. **if**(lines==null|| lines.length==0)
10. **return** res;
11. List<String> text = **new** 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** manullySetKP(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 = "ABCDEFGHIGKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";//字母
33. String number = "0123456789";//数字
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. "typedef","unsigned","union","void","volatile","while"};//关键字
39. String operator[] ={"<<=",">>=","&&","||","<=","|=","\*=","^=","==","++","--","/=","-=","+=","%=","!=",">=","[","]","!","%","(",")","\*","+",",","-","/",";","<","=",">"};//运算符
40. String arithmeticOperator[] = {"++","--","+","-","\*","/","%"};//算术运算符
41. String relationalOperator[] = {"<=","<",">=",">","==","!="};//关系运算符
42. String logicalOperator[] = {"&&","||","!"};//逻辑运算符
43. String delimiter[] = {";",",","(",")","[","]"};//分界符
44. String assignmentOperator[] ={"=","+=","-=","\*=","/=","%=","<<=",">>=","%=","^=","|="};//赋值运算符
46. Map<String,String> opS;//<单个运算符,种类名>
47. Map<String,String[]> KindtoArrary;//<种类名,对应的运算符数组>
49. List<Result> result = **new** ArrayList<>();
51. **public** Analyzer(){
52. KindtoArrary = **new** HashMap<>();
53. KindtoArrary.put("算术运算符",arithmeticOperator);
54. KindtoArrary.put("关系运算符",relationalOperator);
55. KindtoArrary.put("逻辑运算符",logicalOperator);
56. KindtoArrary.put("分界符",delimiter);
57. KindtoArrary.put("赋值运算符",assignmentOperator);
58. opS = **new** HashMap<>();
59. KindtoArrary.keySet().forEach(KindStr->Arrays.asList(KindtoArrary.get(KindStr)).forEach(str->opS.put(str,KindStr)));
60. }
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 && line.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. }
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号  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(KindtoArrary.get(opS.get(str))).indexOf(str);
130. res.setSequence("("+count+","+str+")");
131. match = **true**;
132. i+=str.length();
133. **break**;
134. }
135. }
136. **if**(!match){//没有匹配到
137. res.setWord(line.substring(0,1));
138. 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[] toStringArrary(){
174. String stringS[] = {"  "+word,"  "+binarySequence,"  "+kind,"  "+location};
175. **return** stringS;
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. }

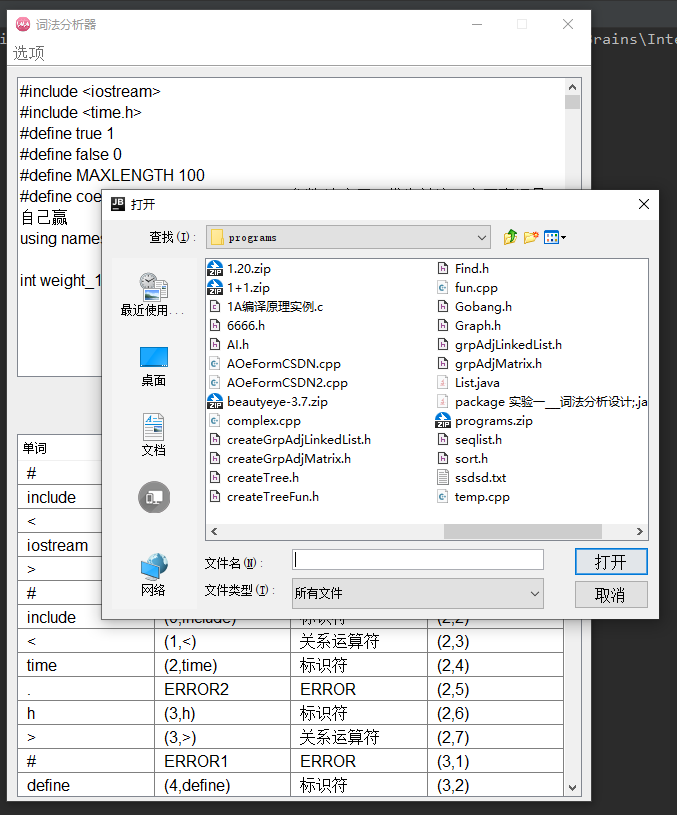
GUI.java

1. **package** 实验一\_\_\_词法分析设计;
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;
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", "ScrollPane", "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();
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. }
126. bar = **new** JMenuBar();
127. setJMenuBar(bar);
129. menu = **new** JMenu("选项");
130. bar.add(menu);
132. file = **new** JMenuItem("选择文件");
133. file.addActionListener(**new** fileListen());//读取文件到TA里
135. manuallySet = **new** JMenuItem("手动设定");
136. manuallySet.addActionListener(**new** manuallySet());
138. exit = **new** JMenuItem("退出");
139. exit.addActionListener(**new** exitListen());
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.toStringArrary());
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());
182. analyse = **new** JButton("分析");
183. analyse.addActionListener(**new** analyseListen());
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. }
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. }
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：







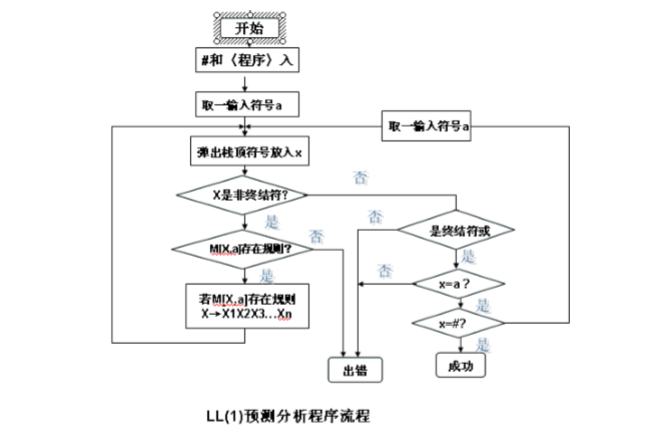
4.实验收获

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
9. 传入文法G
11. G按照\n和"->"以及"\\|"分割为单元
13. **if**(成功){
14. 保存并更新语法
15. }
16. **else**{
17. 弹出语法错误警告
18. }
19. **if**(存在左递归){
20. 弹出左递归警告
21. }
23. 计算First集(){
24. **for**(String symbol:终结符){
25. First(symbol) = [symbol]
26. }//终结符的First集是本身
27. **while**(First集的大小还在变化){
28. **for**(String 左边->右边 :所有文法){
29. 取出右边下标为0的符号
30. **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
31. First(左边).addAll(First(当前符号).except("ε"));//
32. **if**(当前符号的First集合含有空){
33. 看向下一个符号 //即下标+1 递归处理
34. }
35. }
36. **else** **if**(当前符号是终结符 或者 "ε"){
37. 把当前符号加入到左边符号的First集中
38. }
39. **else** **if**(是"\0"){
40. 停止
41. }
42. **else**{
43. 停止
44. }
45. }
46. }
47. }
49. 计算Follow集(){
50. 在开始符号的Follow集中加入"#"
51. **for**(String 当前符号:非终结符){
52. **for**(语法 当前语法:所有的含有当前计算Follow集符号的语法){
53. String 紧跟符号 = 当前语法中,当前符号之后的一个符号
54. **if**(紧跟符号为终结符){
55. 把紧跟符号加到当前符号的Follow集中
56. }
57. **else** **if**(紧跟符号为非终结符){
58. 把紧跟符号的First集-"ε"加入到当前符号的Follow集中
59. **if**(当前符号可以的First集含有空)
60. 看向下一个符号//也是递归求解
61. }
62. **else** **if**(当前符号是"\0"){
63. 把空加入到当前符号的Follow集
64. }
65. **else**{
66. 报错 停止
67. }
68. }
69. }
70. }
72. 计算Select集(){
73. **for**(String 当前文法(左边->右边) :所有文法){
74. String 当前符号 = 右边的第一个符号
75. **if**(当前符号是终结符){
76. 把当前符号加入Select(当前文法)
77. }
78. **else** **if**(当前符号是"ε"或者是"\0"){
79. 把Follow(左边)加入到Select(当前文法)
80. }
81. **else** **if**(当前符号是非终结符){
82. 把First(当前符号).except("ε")加入到Select(当前文法)中
83. **if**(当前符号的First集含有"ε)
84. 看向下一个符号 递归求解
85. }
86. **else**{
87. 报错 停止
88. }
89. }
90. }
92. 计算M表(){
93. **for**(String 当前文法:select集){
94. **for**(String 当前符号:Select(当前文法)){
95. M(当前文法 的 左边,当前符号) = 当前文法
96. }
97. }
98. }
100. 分析过程(String 输入的内容){
101. 初始化分析栈
102. 初始化输入栈
103. **while**(结束标记为未结束){
104. **if**(存在文法){
105. **if**(M(x,a) == "ε"){
106. 分析栈出栈
107. }
108. **else**{
109. 分析栈.push(M(分析栈.pop(),a))
110. }
111. }
112. **else** **if**(匹配到了 且没有结束){
113. 输入栈.pop()
114. 分析栈.pop()
115. }
116. **else** **if**(匹配成功 是#){
117. 结束标记修改为结束
118. }
119. **else**{
120. 报错
121. 结束标记修改为结束
122. }
123. }
124. }

2.算法流程图



3.源码及测试结果

Main.java：

1. package 实验二\_\_\_LL1分析法;
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;
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["+x+","+a+"] -> ["+sb + "]  STACK:" + AnaStack);
108. }
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. }
130. procesList.add(INFO);
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. }
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. 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#
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. }

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(Collectors.toSet()));
242. **if**(M(first,"ε")!=null){//是否可以推出空
243. setFirst(lam.substring(1),G);//扫描下一个
244. }
245. }
246. **else** {
247. }
248. }
249. **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
250. **if**(!lam.contains(sym)){
251. **return**;
252. }
253. **int** index = lam.indexOf(sym);//位置
254. **if**(index == lam.length()-1){// 是\0
255. Follow.get(sym).add("#");
256. Follow.get(sym).addAll(Follow.get(G));//把产生式左边的FOLLOW 加入到其的FOLLOW集中
257. **return**;
258. }
259. **else** **if**(index < lam.length()-1){
260. String next = lam.substring(index+1,index+2);//右边的符号
261. **if**(terminal.contains(next)){//是终结符
262. Follow.get(sym).add(next);
263. }
264. **else** **if**(noTerminal.contains(next)){//非终结符
265. Follow.get(sym).addAll(First.get(next).stream().filter(S->!S.equals("ε")).collect(Collectors.toSet()));//把他的Fist集-ε 加入到当前分析的Follow集中
266. **if**(M(next,"ε")!=null){//检查可否推出空
267. //扫描下一个符号
268. StringBuffer changedLam = **new** StringBuffer(lam) ;
269. changedLam.deleteCharAt(index+1);//删除 达到左移的效果\
270. setFollow(changedLam.toString(),G,sym);
271. }
272. }

275. }
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;
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.\*;
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", "ScrollPane", "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();
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. }

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);
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 !", JOptionPane.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);
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);

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);
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();
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);
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);
186. }
187. }

190. clear = **new** JButton("清除");
191. clear.setBounds(600,160,80,30);
192. clear.addActionListener(**new** clearListen());
194. confirm = **new** JButton("确认");
195. confirm.setBounds(695,160,80,30);
196. confirm.addActionListener(**new** confirmListen());
198. setG = **new** JButton("自定义语法");
199. setG.setBounds(460,160,120,30);
200. setG.addActionListener(**new** MyDialog(**this**));
202. FF = **new** JButton("Fist,Follow集");
203. FF.setBounds(260,160,180,30);
204. FF.addActionListener(**new** FFListen());
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);
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;
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);
242. {
243. Map<String,Integer> Grammap = **new** HashMap<>();
244. Map<String,Integer>  endGmap = **new** HashMap<>();

247. **int** a = 0;
248. **for** (String s1 : sol.terminal) {
249. //System.out.print("     "+s1);
250. endGmap.put(s1,a);
251. a++;
252. }
253. **int** b = 0;
254. **for** (String s : sol.noTerminal) {
255. Grammap.put(s,b);
256. b++;
257. }
258. String[][] map = **new** String[Grammap.size()][endGmap.size()];
259. //System.out.println();
260. **for** (String s1 : sol.noTerminal) {
261. //System.out.print(s1+":  ");
262. **for** (String s2 : sol.terminal) {
263. String resss = sol.AnaTable.get(s1+s2);
264. **if**(resss == null){
265. //System.out.print("      ");
266. map[Grammap.get(s1)][endGmap.get(s2)] = " ";
267. }
268. **else** {
269. //System.out.print(resss+"    ");
270. map[Grammap.get(s1)][endGmap.get(s2)] = resss;
271. }
272. }
273. //System.out.println();
274. }
275. //System.out.println("====================");


279. //                    for (String[] strings : map) {
280. //                        for (String string : strings) {
281. //                            System.out.print(string+" ");
282. //                        }
283. //                        System.out.println();
284. //                    }

287. String [] colum = **new** String[sol.noTerminal.size()+1];
288. **int** counter = 0;
289. **for** (String s : sol.noTerminal) {
290. colum[counter] = s;
291. counter++;
292. }
293. String[][] data = **new** String[map.length][map[0].length+1];
294. **for** (**int** j = 0; j < data.length; j++ ) {
295. data[j][0] = colum[j];
296. **for** (**int** i = 1; i < data[0].length; i++) {
297. data[j][i] = map[j][i-1];
298. }
299. }
300. String[] name = **new** String[sol.terminal.size()+1];
301. **int** i = 1;
302. name[0] = " ";
303. **for** (String s : sol.terminal) {
304. name[i] = s;
305. i++;
306. }
307. JTable  mmm = **new** JTable(data, name);
308. mmm.setRowHeight(30);
309. JScrollPane secsa = **new** JScrollPane(mmm);
310. secsa.setBorder(null);
311. jTabbedPane.addTab("分析表",secsa);
313. }

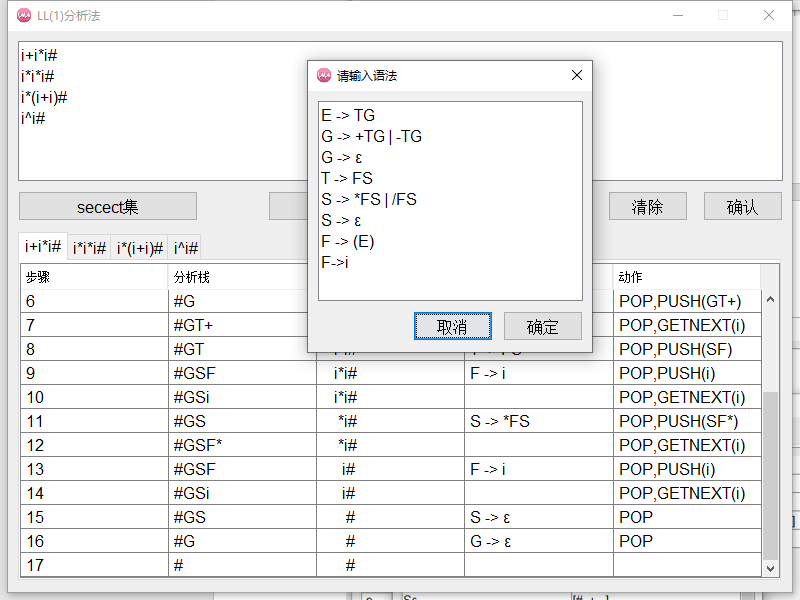


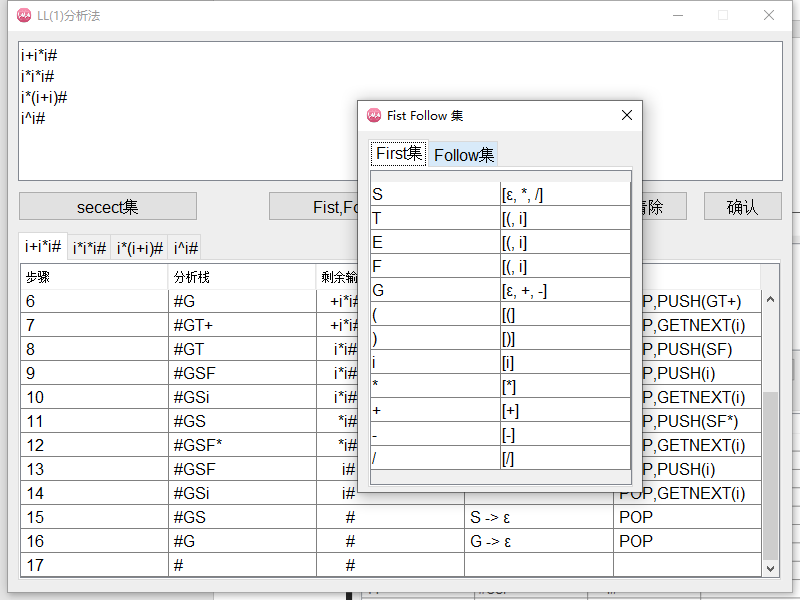
318. }
319. }

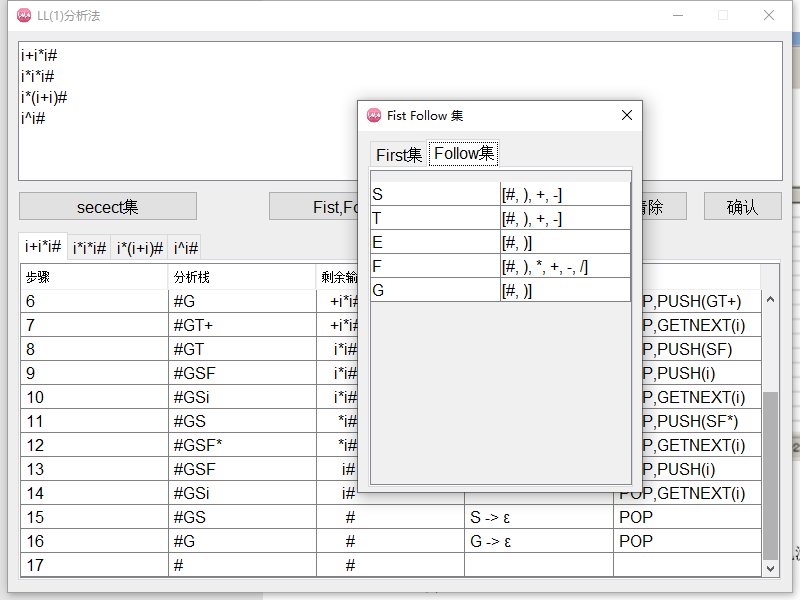
322. JButton select;
323. select = **new** JButton("secect集");
324. select.setBounds(10,160,180,30);
325. select.addActionListener(**new** selectListen());
326. add(select);
328. add(clear);
329. add(confirm);
330. add(setG);
331. add(FF);
332. }
333. **void** initText(){
334. textArea = **new** JTextArea("i+i\*i#\ni\*i\*i#\ni\*(i+i)#\ni^i#");
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. }
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. }
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. }

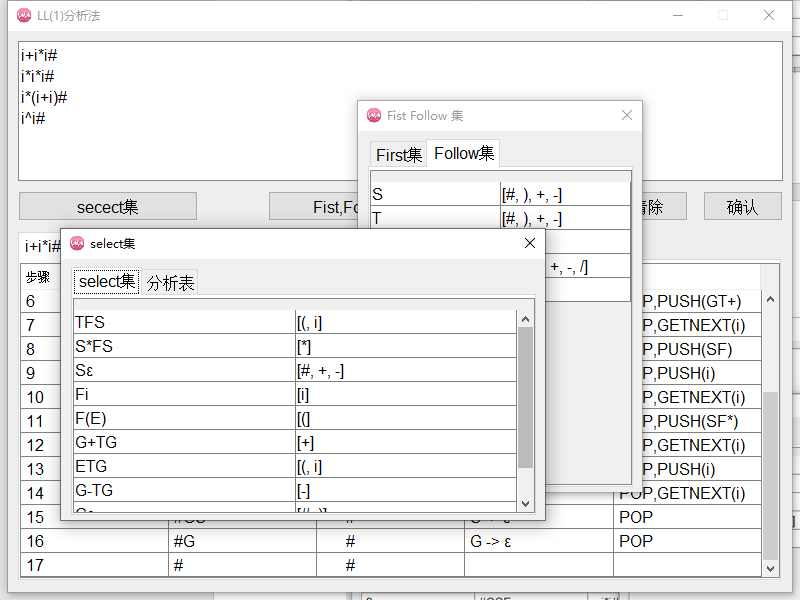
运行结果：

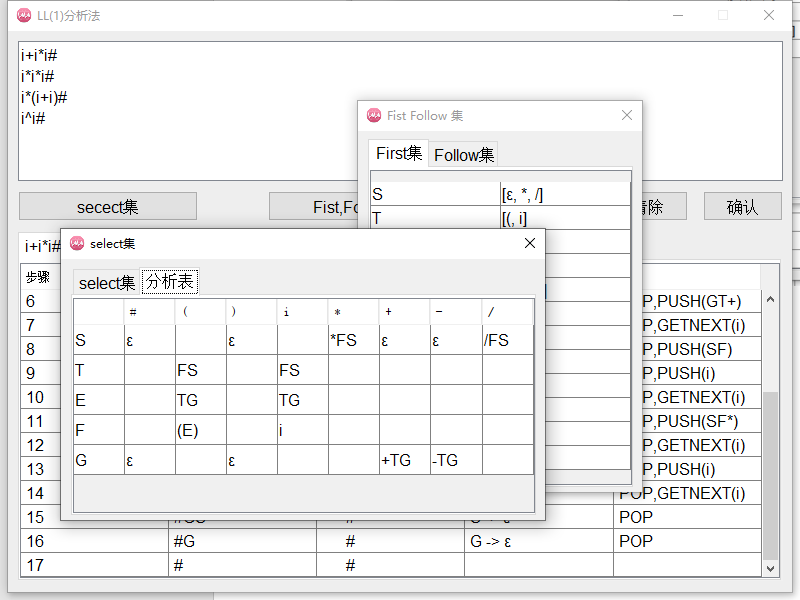












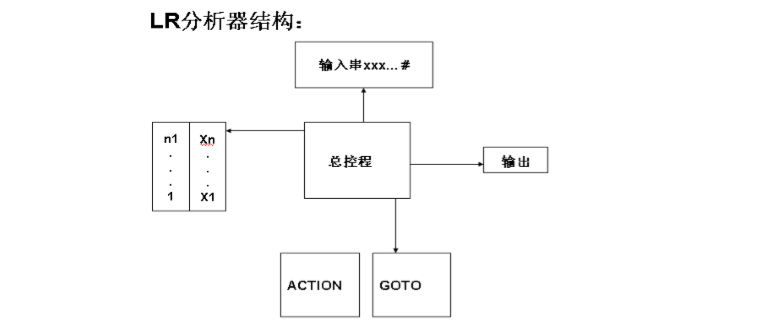
4.实验收获

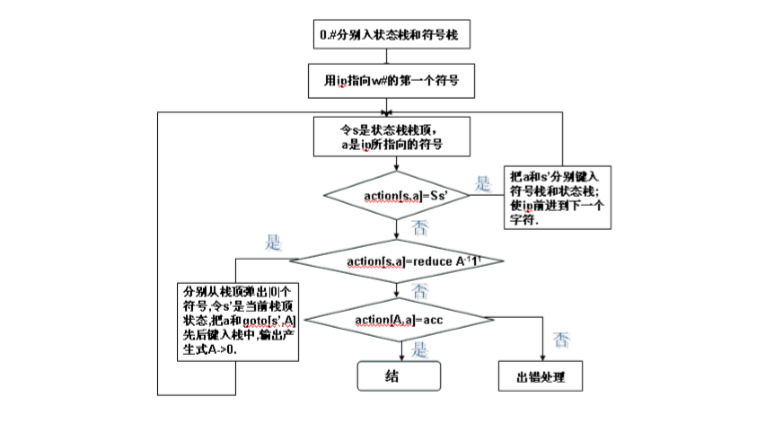
本次实验设计的程序层次分明，程序界面与处理程序低耦合高内聚，分析器使用文法作为参数，对传入的字符串进行分析，并返回分析结果，此外通过调用类中的方法可以返回对应的First集Follow集等，便于显示。LL1对分析法有了更好的理解和掌握。

**实验三 LR(1)分析法**

1. 数据结构及算法描述
   1. **public** String grammarText;
   2. **public** List<String[]> grammar = **new** ArrayList<>();//语法格式:(T->SF)= {T,SF}
   3. **public** Set<String> nonTerminal = **new** HashSet<>();//非终结符
   4. **public** Set<String>  terminal = **new** HashSet<>();//终结符
   5. **public** Set<String> allSymbol = **new** HashSet<>();//全部符号
   6. **public** Map<String,Set<String>> First = **new** HashMap<>();//First集
   7. **public** List<project> projectList = **new** ArrayList<>();//所有项目
   8. **public** List<projectSet> cProjectSets = **new** ArrayList<>();//项目集C
   9. **public** Map<point,String> ActionTable = **new** HashMap<>();//action表
   10. **public** Map<point,String> GOTO = **new** HashMap<>();//goto表
   11. **public** String[][] ActionAndGoTo;//Action和GOTO表
   12. **public** String startSymbol = "S`";//文法开始符号
2. 初始化文法
3. 读取文法,计算First集(使用实验2的算法即可)
4. 初始化项目List(){
5. 加入(S`->.S),#和(S`->S.,#)//开始文法 特殊情况 手动添加
6. **for**(文法 A->BC:所有文法){
7. **for**( index : 所有可以插入点的位置){
8. **for**(文法 当前文法:所有文法){
9. **if**(当前文法的右边含有A){
10. 把(A->BC,index,First(点后部的字符串))加入到项目集中
11. }
12. }
13. }
14. }
15. }
16. 初始化项目集C(){
17. 初始化栈
18. 创建一个项目集
19. 把开始文法的项目放入其中
20. 计算它的闭包
21. 把这个项目放入栈中
22. **while**(栈非空){
23. 当前项目集 = 栈.pop()
24. **for**(String Symbol : 每个符号){
25. 创建一个新的项目集
26. **for**(项目 : 当前项目集中的所有项目){
27. **if**(是形如 A->...•X... 的项目){
28. 点向后移动一位创建新的项目,加入到新的项目集中
29. }
30. }
31. 计算闭包(新的项目集)
32. **if**(如果新的项目集为空){
33. 标记当前项目集通过Symbol跳转到-1
34. }
35. **else** **if**(新的项目集已经存在){
36. 标记当前项目集通过Symbol跳转到 项目集List.indexOf(新的项目集)
37. }
38. **else**{
39. 把新的项目集加入到项目集List中
40. 标记当前项目集通过Symbol跳转到 项目集List.indexOf(新的项目集)
41. 栈.push(新的项目集)
42. }
43. }
44. }
45. }
46. 计算闭包(项目集){
47. 新建栈
48. 项目集.forEach(栈::push)
49. **while**(栈非空){
50. 当前项目 = 栈.pop()
51. 项目.add(当前项目);
52. 点后符号 = 当前项目的点后的第一个符号
53. **if**(点后符号是非终结符){
54. **for**(当前产生式:每个左边是B的产生式){
55. **for**(symbol:First(B之后的部分,当前项目的展望符)){
56. **new** 新项目(当前产生式,0,symbol);
57. **if**(项目原本不存在项目集中){
58. 栈.push(新项目)
59. }
60. }
61. }
62. }
63. }
64. }
65. Go(项目集,符号){
66. **if**(项目集通过(符号跳转) != -1){
67. **return** 项目集通过(符号跳转)的项目集
68. }
69. **else**
70. **return** 空集
71. }
72. 创建Action和Goto表(){
73. **for**(当前项目集:所有项目集){
74. **for**( 当前项目 : 所有项目 ){
75. 当前项目 刑如 [A->...•a...,b]
76. **if**(a是终结符){
77. ActionTable(当前项目.index,a) = S + Go(当前项目集,a).index
78. }
79. **if**(a是""){
80. ActionTable(当前项目.index,b) = R + 当前项目集.indexOf语法
81. }
82. }
83. }
84. ActionTable(初始项目.index,#) = acc
85. **for**(当前项目集:所有项目集){
86. **for**(当前符号:所有符号){
87. Goto(当前项目集,当前符号) = Go(当前项目集,当前符号)
88. }
89. }
90. 其他位置标记为err
91. }
92. 主控函数(){
93. 初始化输入串栈
94. 初始化符号栈
95. 初始化状态栈
96. **while**(结束标记不为结束){
97. String i =  状态栈顶
98. String a = 输入串栈顶
99. String action = ActionTable.(i,a)
100. **if**(action() == **null**或者er ){
101. 报错
102. 标记结束
103. }
104. **else** **if**(action == acc ){
105. 成功
106. 标记结束
107. }
108. **else** **if**(action == Si){
109. i入状态栈
110. a到文法符号栈
111. }
112. **else** **if**(action == Ri){
113. 用index产生式规约
114. 符号栈中取出文法的右侧的长度的字符
115. 再压栈文法左边的符号
116. 状态栈.push( GOTO ( 状态栈.top() , 符号栈.top() ) )
117. }
118. **else**{
119. 报错
120. 标记结束
121. }
122. }
123. }

2.算法流程图





3.源码及测试结果

Main.java：

1. **package** 实验三\_LR1分析法;
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;

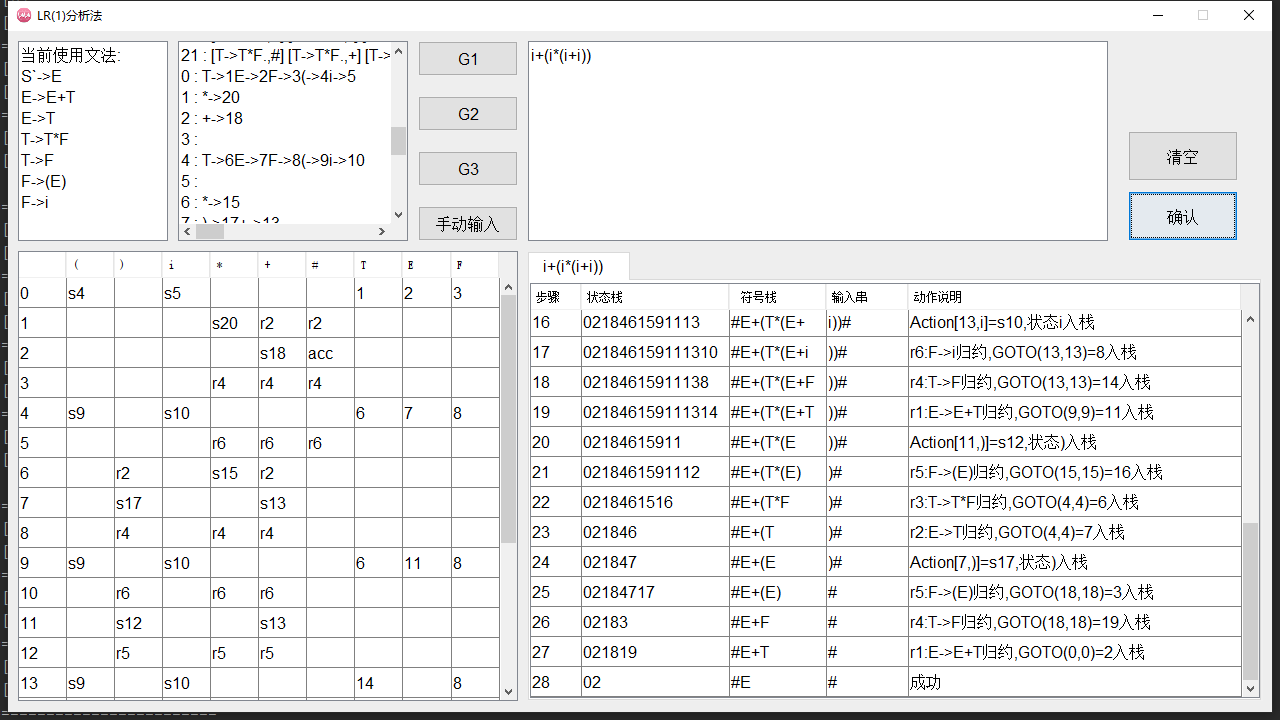
4. **import** java.util.\*;
5. **import** java.util.stream.Collectors;
7. **class** Solution {
8. **public** String grammarText;
9. **public** List<String[]> grammar = **new** ArrayList<>();//(T->SF)= {T,SF} forAll
10. **public** Set<String> nonTerminal = **new** HashSet<>();//非终结符
11. **public** Set<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→α·Bβ,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**.indexOfNode)  && (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.sons.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));
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(Collectors.toSet()));
200. **if**(First.get(first).contains("ε")){//是否可以推出空
201. setSingleFirst(lam.substring(1),G);//扫描下一个
202. }
203. }
204. **else** {
205. System.out.println("ERROR");
206. }
207. }
208. **private** **void** setProjectList(){
209. projectList.add(**new** project(0,0,"#"));
210. projectList.add(**new** project(0,1,"#"));//开始符号 特殊  手动添加
211. **for** (**int** indexOfGrammar = 0; indexOfGrammar < grammar.size(); indexOfGrammar++) {
212. **for** (**int** indexOfNode = 0; indexOfNode <= grammar.get(indexOfGrammar)[1].length() ; indexOfNode++) {
213. String A = grammar.get(indexOfGrammar)[0];//A->BC  A
214. Set<String> a = **new** HashSet<>();
215. **for** (String[] strings : grammar) {
216. **if**(strings[1].contains(A)){
217. **int** index = strings[1].indexOf(A)+1;
218. String sub = strings[1].substring(index);
219. a.addAll(First(sub));
220. }
221. }
222. **for** (String s : a) {
223. projectList.add( **new** project(indexOfGrammar,indexOfNode,s) );
224. }
225. }
226. }
227. System.out.println("项目s");
228. **for** (**int** i = 0; i < projectList.size(); i++) {
229. System.out.println(i+" : " +projectList.get(i));
230. }
231. }//读取项目
232. **private** projectSet extendSingleClosure(projectSet closure) {
233. Stack<project> stack = **new** Stack<>();
234. closure.getSet().forEach(stack::push);
235. **while**(!stack.empty()){
236. project top = stack.pop();
237. String B = top.getFirstSymbolAfterNode();
238. closure.projects.add(top);
239. **if**(nonTerminal.contains(B)){//如果是 A->...•B...
240. List<String[]> GsHeadIsB =  grammar.stream().filter(G->G[0].equals(B)).collect(Collectors.toList());//每个左边是B的产生式
241. String beta = top.getRestStringAfterFirst();//获得B之后的部分
242. String a = top.getExtSymbol();//产生式之后的符号
243. **for** (String[] strings : GsHeadIsB) {
244. **for** (String b : First(beta, a)) {
245. project newProject = **new** project(grammar.indexOf(strings),0,b);
246. **if**(!closure.getSet().contains(newProject)){
247. stack.push(newProject);
248. }
249. }
250. }
251. }
252. }
254. **return** closure;
255. }//传入一个非空的项目集 将其扩充到不改变大小为止  返回此项目集
256. **private** **void** setCanonicalCollection(){
257. System.out.println("创建集合C:");
258. projectSet StartI = **new** projectSet();
259. **for** (project project : projectList) {
260. **if**(project.getGrammar()[0].equals(startSymbol) && project.getIndex()==0){
261. StartI.add(project);
262. }
263. }
264. extendSingleClosure(StartI);
265. Stack<projectSet> stack = **new** Stack();
266. stack.push(StartI);
267. cProjectSets.add(StartI);
268. **while** (!stack.empty()){//DFS顺序去创建
269. projectSet current = stack.pop();
270. //System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
271. //System.out.print(current+" : ");
272. //System.out.println(current.getSet().toString());
273. **for** (String s : allSymbol) {
274. List<project> temp = **new** ArrayList<>();
275. **for** (project project : current.getSet()) {
276. **if**(project.getFirstSymbolAfterNode().equals(s)){//对于每个形如 A->...•X... 的项目
277. //if(s.equals("L"))
278. //System.out.println("使用 : "+project  +  "  s = "+ s);
279. project newProject = **new** project(project.indexOfGrammar,project.indexOfNode+1,project.getExtSymbol());
280. //if(s.equals("L"))
281. //System.out.println("得到了 : "+newProject);
282. temp.add(newProject);
283. }
284. }
285. //System.out.println("最终list : "+temp);
287. projectSet nextI = **new** projectSet();
288. nextI.projects.addAll(temp);
289. extendSingleClosure(nextI);
290. //System.out.println("最终set "+nextI.getSet());
292. **if**(nextI.getSet().size()==0){
293. current.sons.put(s,-1);
294. **continue**;
295. }
296. **if**(cProjectSets.contains(nextI)){
297. current.sons.put(s,cProjectSets.indexOf(nextI));
298. **continue**;
299. }
300. stack.push(nextI);
301. Integer index = cProjectSets.size();
302. cProjectSets.add(nextI);
303. current.sons.put(s,index);
304. }
305. //System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");
306. }
307. }//设定项目集C
308. **private** projectSet GO(projectSet I,String X ){//如果sons可以得出 则直接返回结果 结果应当是扩充过的  如果不能得出 则去计算 返回的也是计算过闭包的
310. **if**(I.sons.containsKey(X) && I.sons.get(X)!=-1){
311. **return** cProjectSets.get(I.sons.get(X));
312. }
313. **return** **new** projectSet();
314. }
315. **private** **void** setActionAndGOTOTable(){
316. **for** (projectSet projectSet : cProjectSets) {
317. **for** (project project : projectSet.getSet()) {
318. String a = project.getFirstSymbolAfterNode();
319. **if**(terminal.contains(a)){//项目 [A->...•a...,b] a是终结符
320. ActionTable.put(**new** point(projectSet.toString(),a),"s"+GO(projectSet,a).toString());
321. }
322. }
323. }//<1>
324. **for** (projectSet projectSet : cProjectSets) {
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().equals("")){
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.get(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("EqualsErr");
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
461. String nextSta = GOTO.get(**new** point(statusStack.getTop(),symbolStack.getTop()));
462. currentStep[4] = action+":"+grammar.get(index)[0]+"->"+grammar.get(index)[1]+"归约,GOTO("+statusStack.getTop()+","+statusStack.getTop()+")="+nextSta+"入栈";
463. statusStack.push(nextSta);
464. }
465. **else** {
466. System.out.println("err");
467. currentStep[4] = "ERROR";
468. iFlag = **false**;
469. }
470. currentStep[0] = count+"\t";
471. count++;

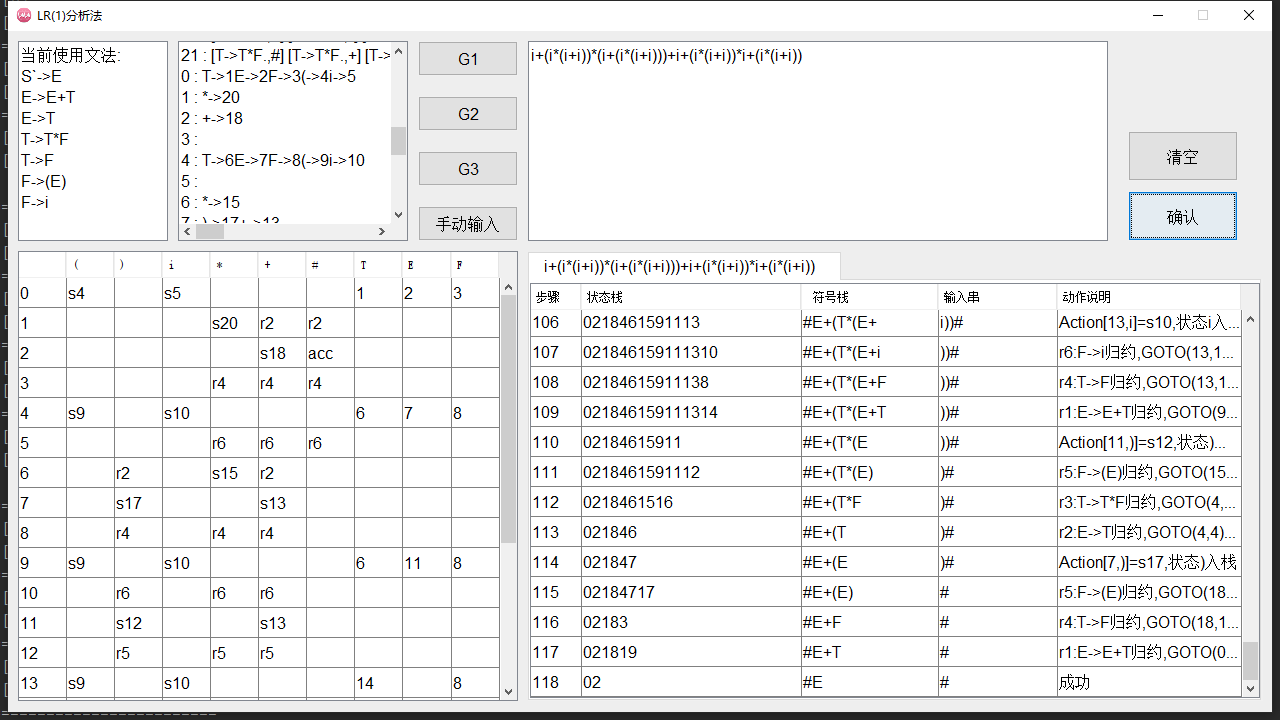
474. System.out.println("================================");
475. }
476. **return** processRecord;
477. }
478. Set<String> First(String beta,String a){
479. Set<String> res = **new** HashSet<>();
480. **if**(beta.length() == 0){
481. res.add(a);
482. **return** res;
483. }
484. **else** {
485. res.addAll(First(beta));
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. }
521. **class** point {
522. String head, tail;
524. point(String head, String tail) {
525. **this**.head = head;
526. **this**.tail = tail;
527. }
529. **public** String getHead() {
530. **return** head;
531. }
533. **public** String getTail() {
534. **return** tail;
535. }
537. @Override
538. **public** **int** hashCode() {
539. **return** (head + "->" + tail).hashCode();
540. }
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. }
551. }
553. **class** MyStack {
554. List<String> s;
556. MyStack() {
557. s = **new** LinkedList<>();
558. }
560. **void** push(String value) {
561. s.add(value);
562. }
564. **void** push(String... values) {
565. **for** (String value : values) {
566. push(value);
567. }
568. }
570. String pop() {
571. **return** s.remove(s.size() - 1);
572. }
574. String getTop() {
575. **return** s.get(s.size() - 1);
576. }
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. }
587. **public** Boolean isEmpty() {
588. **return** s.size() == 0;
589. }
590. }

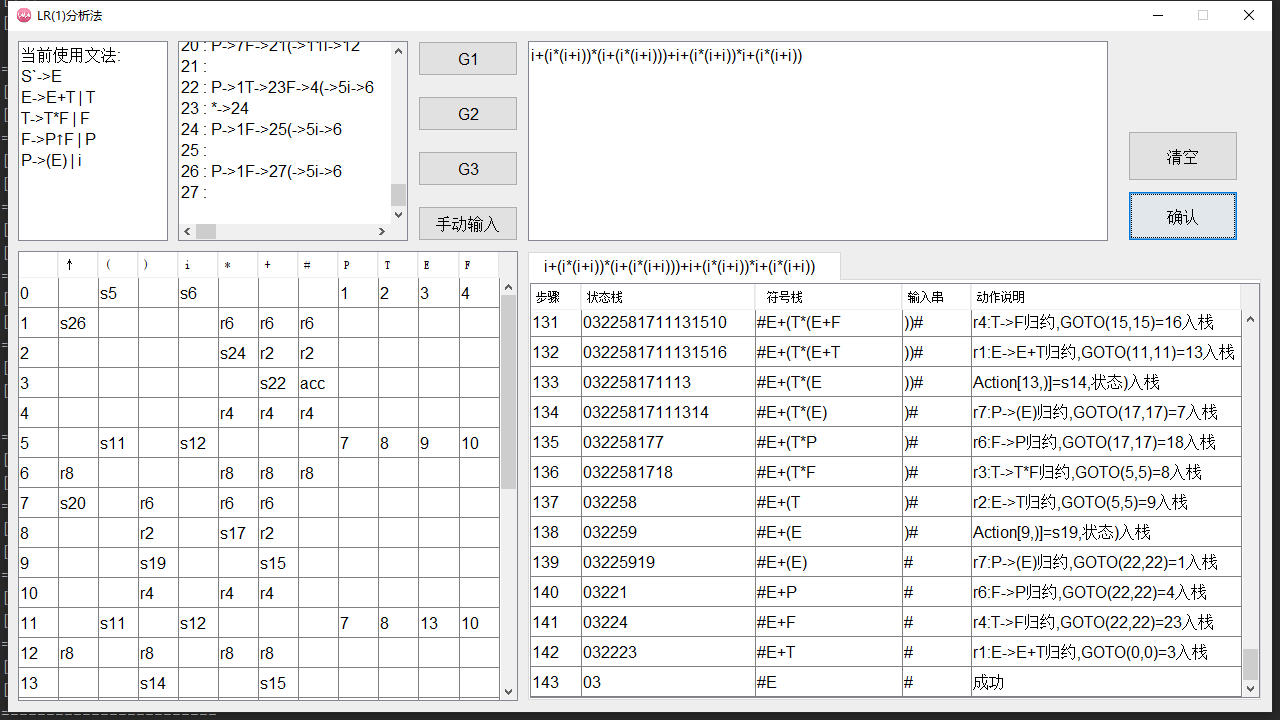
GUI.java

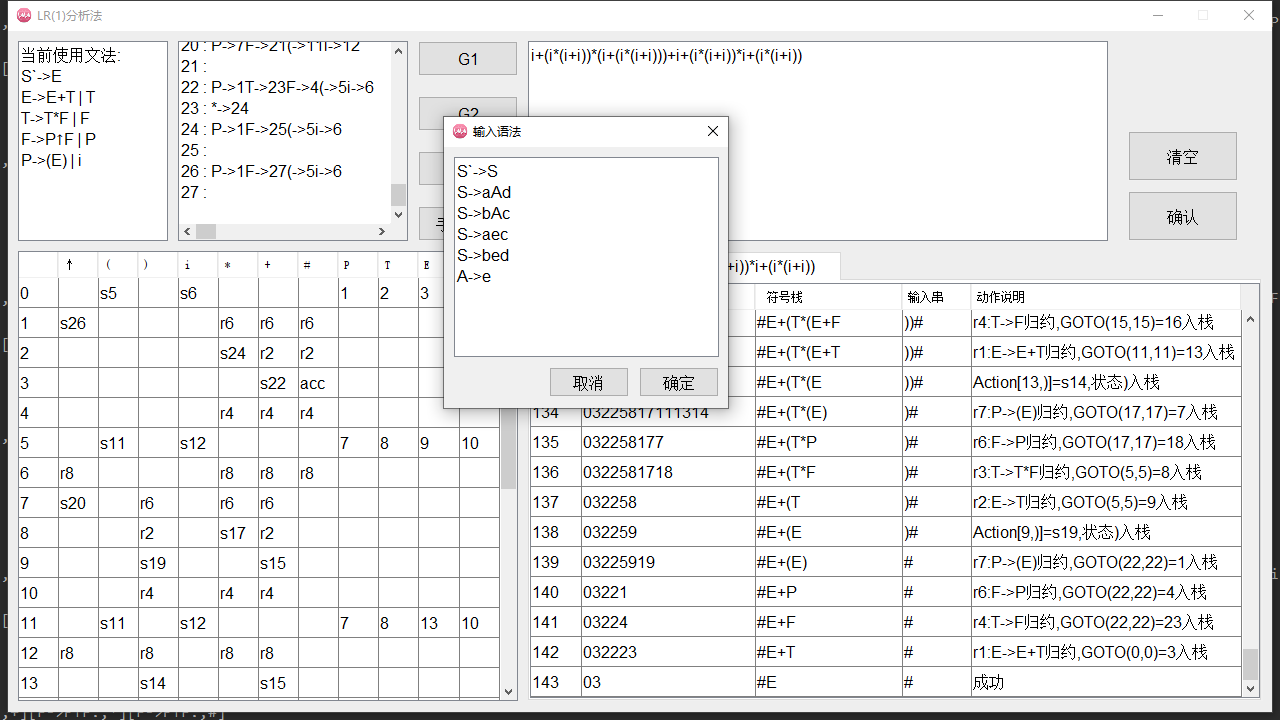
1. **package** 实验三\_LR1分析法;
3. **import** javax.swing.\*;
4. **import** java.awt.\*;
5. **import** java.awt.event.ActionEvent;
6. **import** java.awt.event.ActionListener;
7. **import** java.util.Vector;
8. **import** javax.swing.table.JTableHeader;
9. **import** javax.swing.table.TableColumn;
10. **import** java.util.\*;
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;
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", "ScrollPane", "Table","TabbedPane"};
26. **for** (String item : names) {
27. UIManager.put(item+ ".font",f);
28. }
29. UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
30. }**catch**(Exception e){}
31. sol = **new** Solution("S`->E\n"+
32. "E->E+T\n" +
33. "E->T\n" +
34. "T->T\*F\n" +
35. "T->F\n" +
36. "F->(E)\n" +
37. "F->i");
38. init();
39. setSize(1280,720);//初始大小
40. setLocation(100,80);//初始位置
41. setVisible(**true**);//是否可视
42. setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);//X退出
43. }
44. **void** init(){
45. setTitle("LR(1)分析法");
46. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
47. setVisible(**true**);
48. setResizable(**false**);
49. setLayout(**null**);
50. initGrammarText();
51. initActTable();
52. initButton();
53. initInputArea();
54. initResultTable();
55. initProjectPane();
56. }
57. **void** initGrammarText(){
58. grammarTextArea = **new** JTextArea("当前使用文法:\n"+sol.grammarText);
59. grammarTextArea.setEditable(**false**);
60. JScrollPane textAreaRollPane = **new** JScrollPane(grammarTextArea);
61. textAreaRollPane.setBounds(10,10,150,200);
62. 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↑F | 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();
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("err")){
214. data[i][j] = "";
215. }
216. }
217. }
218. JTable actGoTable;
219. actGoTable = **new** JTable(data,head);
220. actGoTable.setRowHeight(30);
221. tablePane.setViewportView(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** preferedWidth = (**int**) myTable.getCellRenderer(row, col).getTableCellRendererComponent(myTable,
242. myTable.getValueAt(row, col), **false**, **false**, row, col).getPreferredSize().getWidth();
243. width = Math.max(width, preferedWidth);
244. }
245. header.setResizingColumn(column);
246. column.setWidth(width + myTable.getIntercellSpacing().width+10);
247. }
248. }
249. }

运行结果：









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。