

**编译原理实验报告**

学生姓名 ：林天岳

学 号 ：2017217893

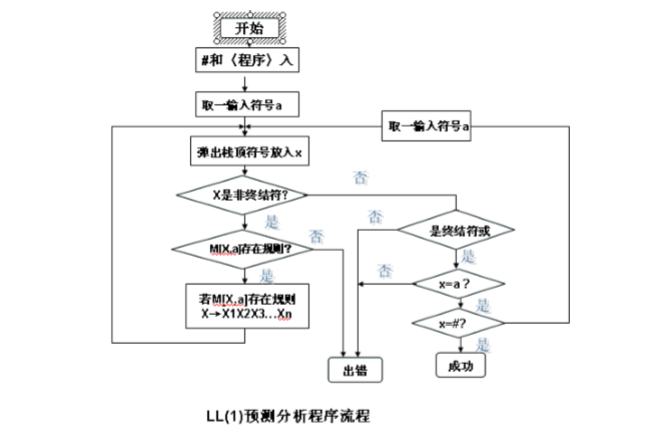
班 级 ：计算机科学与技术2017-5

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. }

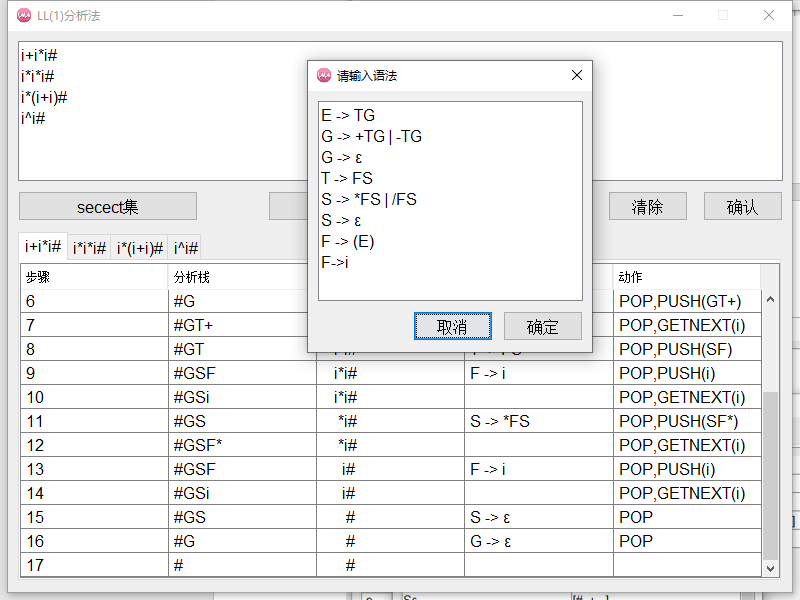


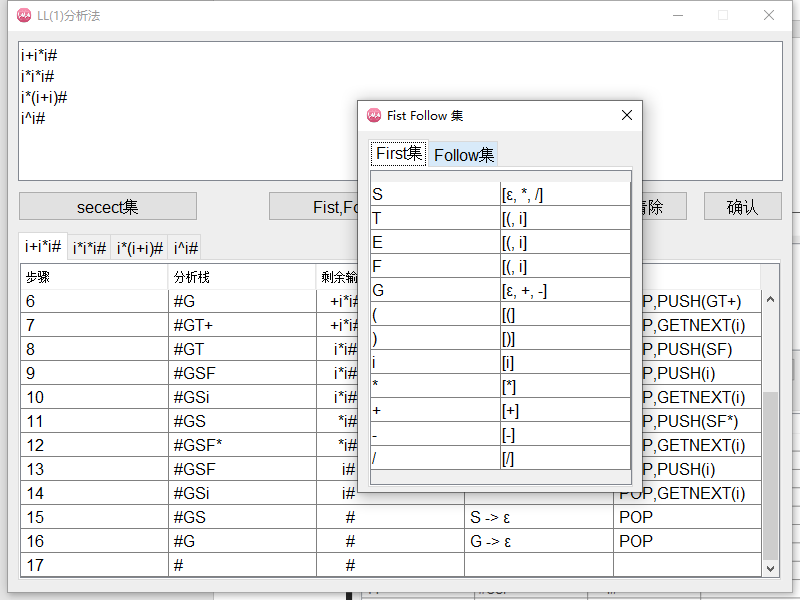
318. }
319. }

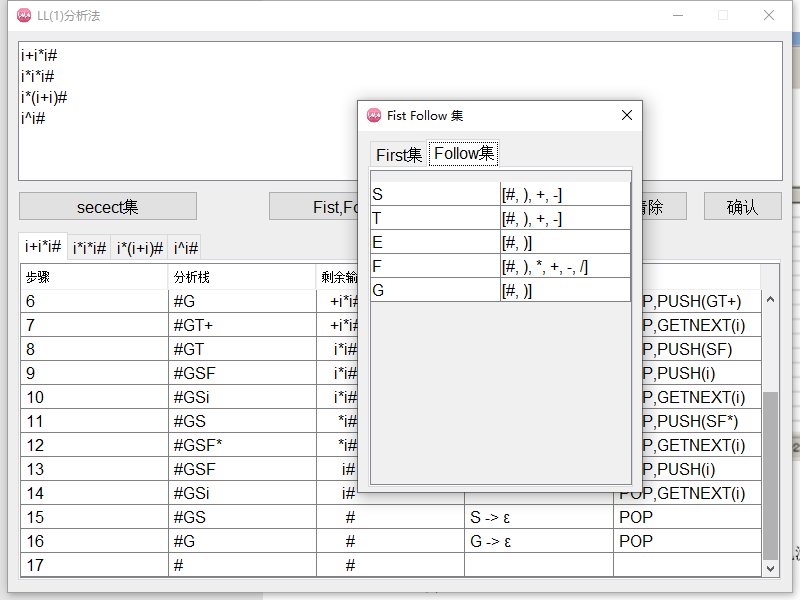
322. JButton select;
323. select = **new** JButton("secect集");
324. select.setBounds(10,160,180,30);
325. select.addActionListener(**new** selectListen());
326. add(select);
328. add(clear);
329. add(confirm);
330. add(setG);
331. add(FF);
332. }
333. **void** initText(){
334. textArea = **new** JTextArea("i+i\*i#\ni\*i\*i#\ni\*(i+i)#\ni^i#");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

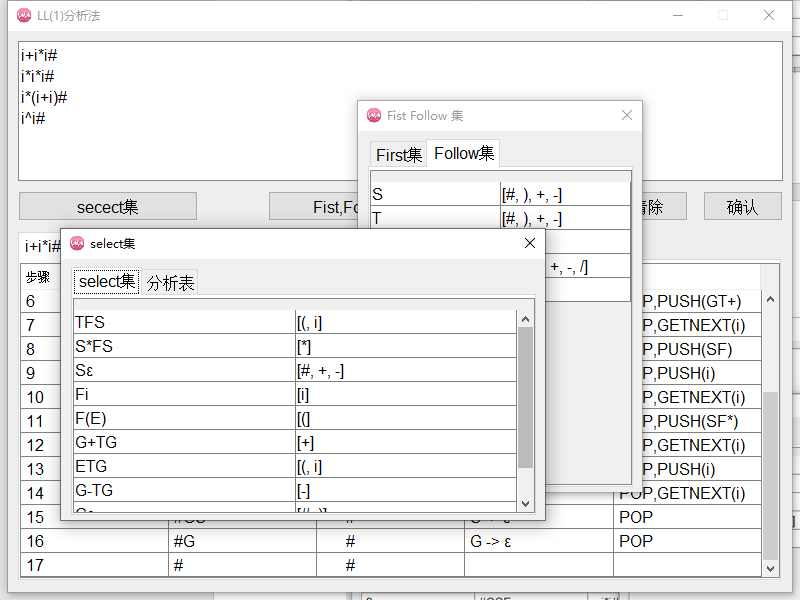
运行结果：

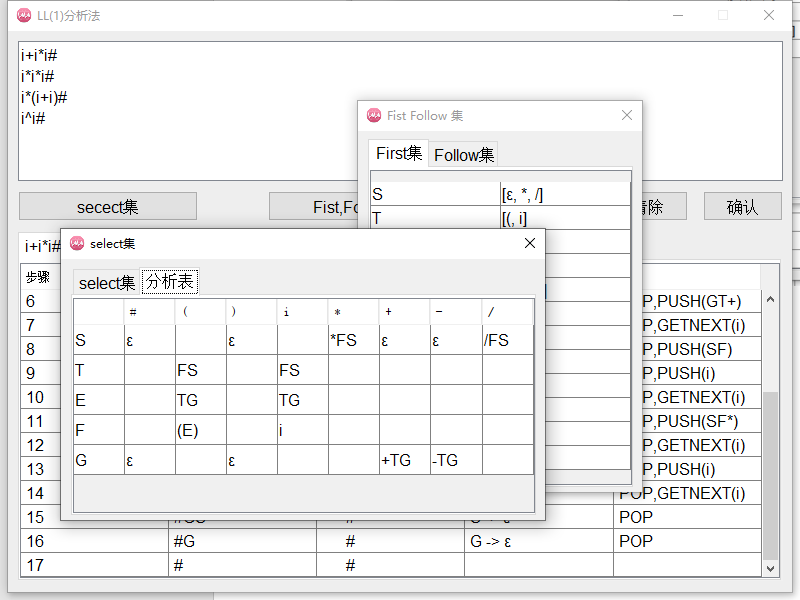












4.实验收获

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。