

**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

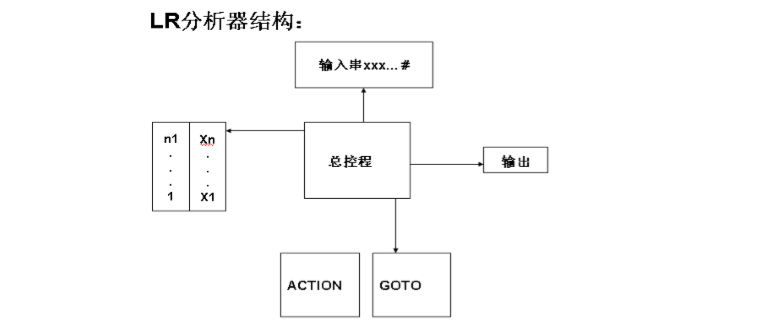
完成日期 ：2019 年 10 月 22 日

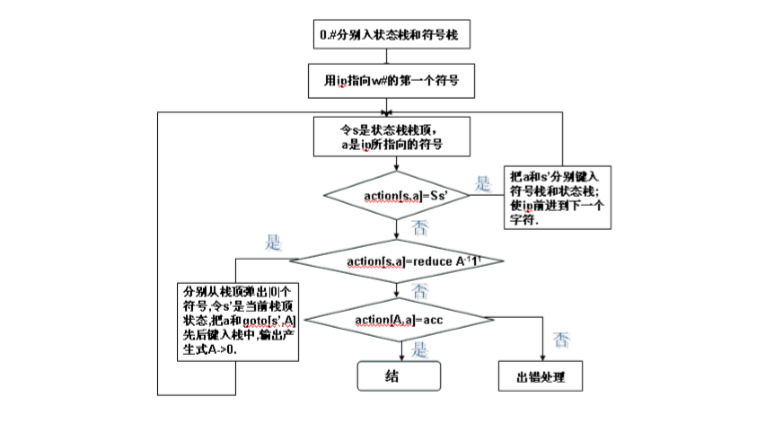
**实验三 LR(1)分析法**

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. **class** project{
3. **private** **int** indexOfGrammar;
4. **private** **int** indexOfNode;
5. **private** String extSymbol;
6. project(**int** indexOfGrammar,**int** index,String extSymbol){
7. **this**.indexOfGrammar = indexOfGrammar;
8. **this**.indexOfNode = index;
9. **this**.extSymbol = extSymbol;
10. }
11. **public** String[] getGrammar() {//获取产生式
12. **return** grammar.get(indexOfGrammar);
13. }
14. **public** **int** getIndex() {//获取所用产生式的编号
15. **return** indexOfNode;
16. }
17. **public** String getHead(){//获取产生式左边
18. **return** **this**.getGrammar()[0];
19. }
20. **public** String getExtSymbol() {
21. **return** extSymbol;
22. }
23. **public** String getRight(){//获取产生式右边点后面的部分
24. **return** **this**.getGrammar()[1].substring(indexOfNode);
25. }
26. **public** String getFirstSymbolAfterNode(){//获取右侧字符串的首字符
27. **if**(**this**.getRight().length()<1){
28. **return** "";
29. }
30. **else** {
31. **return** **this**.getRight().substring(0,1);
32. }
33. }
34. **public** String getRestStringAfterFirst(){//获取右侧首个字符的之后的字符串
35. **if**(**this**.getRight().length()<2){
36. **return** "";
37. }
38. **else** {
39. **return** **this**.getRight().substring(1);
40. }
41. }
42. @Override
43. **public** String toString() {
44. StringBuffer str = **new** StringBuffer(grammar.get(indexOfGrammar)[1]);
45. str.insert(indexOfNode,".");
46. str.insert(0,"["+grammar.get(indexOfGrammar)[0]+"->");
47. str.append(","+extSymbol+"]");
48. **return** str.toString();
49. }
50. @Override
51. **public** **boolean** equals(Object obj) {
52. **if**(!obj.getClass().equals(**this**.getClass())){
53. **return** **false**;
54. }
55. project cmp = ((project)obj);
56. **if**(  (cmp.indexOfGrammar == **this**.indexOfGrammar)  &&  (cmp.indexOfNode==**this**.indexOfNode)  && (cmp.extSymbol.equals(**this**.extSymbol)) ){
57. **return** **true**;
58. }
59. **return** **false**;
60. }
61. @Override
62. **public** **int** hashCode() {
63. String hash = indexOfGrammar+","+indexOfNode+","+extSymbol;
64. **return** hash.hashCode();
65. }
66. }//存储项目,(文法下标,点的位置,展望符)例如:A→α·Bβ,a  grammar: A→α.Bβ  index: location of(.)  extSymbol: a
68. **class** projectSet {
69. Set<project> projects;//集合
70. Set<Integer> indexOfProjects;//
71. 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
72. **public** projectSet(){
73. projects = **new** HashSet<>();
74. indexOfProjects = **new** HashSet<>();
75. sons = **new** HashMap<>();
76. }
77. **public** **boolean** add(project project){
78. **if**(projectList.contains(project)){
79. indexOfProjects.add(projectList.indexOf(project));
80. **return** projects.add(project);
81. }
82. **else** {
83. **return** **false**;
84. }
85. }
86. **public** Set<project> getSet() {
87. **return** projects;
88. }
89. **private** String toCompare(){
90. StringBuffer sb = **new** StringBuffer();
91. projects.forEach(S->sb.append(S.toString()));
92. **return** sb.toString();//比较用  set相同即可认为是相同的集合
93. }
94. @Override
95. **public** String toString() {
96. **return** cProjectSets.indexOf(**this**)+"";
97. }
98. @Override
99. **public** **boolean** equals(Object obj) {
100. **if**(!obj.getClass().equals(**this**.getClass())){
101. **return** **false**;
102. }
103. **return** **this**.toCompare().equals(((projectSet)obj).toCompare()) ;
104. }
105. @Override
106. **public** **int** hashCode() {
107. **return** **this**.toCompare().hashCode();
108. }
109. }//项目集 包含项目的set,以及对应符号的跳转项目集的下标 重写了equals和hashcode,使用set存储,判断是不是生成了重复的对象

112. 初始化文法
113. 读取文法,计算First集(使用实验2的算法即可)
115. 初始化项目List(){
116. 加入(S`->.S),#和(S`->S.,#)//开始文法 特殊情况 手动添加
117. **for**(文法 A->BC:所有文法){
118. **for**( index : 所有可以插入点的位置){
119. **for**(文法 当前文法:所有文法){
120. **if**(当前文法的右边含有A){
121. 把(A->BC,index,First(点后部的字符串))加入到项目集中
122. }
123. }
124. }
125. }
126. }
127. 初始化项目集C(){
128. 初始化栈
129. 创建一个项目集
130. 把开始文法的项目放入其中
131. 计算它的闭包
132. 把这个项目放入栈中
133. **while**(栈非空){
134. 当前项目集 = 栈.pop()
135. **for**(String Symbol : 每个符号){
136. 创建一个新的项目集
137. **for**(项目 : 当前项目集中的所有项目){
138. **if**(是形如 A->...•X... 的项目){
139. 点向后移动一位创建新的项目,加入到新的项目集中
140. }
141. }
142. 计算闭包(新的项目集)
143. **if**(如果新的项目集为空){
144. 标记当前项目集通过Symbol跳转到-1
145. }
146. **else** **if**(新的项目集已经存在){
147. 标记当前项目集通过Symbol跳转到 项目集List.indexOf(新的项目集)
148. }
149. **else**{
150. 把新的项目集加入到项目集List中
151. 标记当前项目集通过Symbol跳转到 项目集List.indexOf(新的项目集)
152. 栈.push(新的项目集)
153. }
154. }
155. }
156. }
157. 计算闭包(项目集){
158. 新建栈
159. 项目集.forEach(栈::push)
160. **while**(栈非空){
161. 当前项目 = 栈.pop()
162. 项目.add(当前项目);
163. 点后符号 = 当前项目的点后的第一个符号
164. **if**(点后符号是非终结符){
165. **for**(当前产生式:每个左边是B的产生式){
166. **for**(symbol:First(B之后的部分,当前项目的展望符)){
167. **new** 新项目(当前产生式,0,symbol);
168. **if**(项目原本不存在项目集中){
169. 栈.push(新项目)
170. }
171. }
172. }
173. }
174. }
175. }
176. Go(项目集,符号){
177. **if**(项目集通过(符号跳转) != -1){
178. **return** 项目集通过(符号跳转)的项目集
179. }
180. **else**
181. **return** 空集
182. }
183. 创建Action和Goto表(){
184. **for**(当前项目集:所有项目集){
185. **for**( 当前项目 : 所有项目 ){
186. 当前项目 刑如 [A->...•a...,b]
187. **if**(a是终结符){
188. ActionTable(当前项目.index,a) = S + Go(当前项目集,a).index
189. }
190. **if**(a是""){
191. ActionTable(当前项目.index,b) = R + 当前项目集.indexOf语法
192. }
193. }
194. }
195. ActionTable(初始项目.index,#) = acc
196. **for**(当前项目集:所有项目集){
197. **for**(当前符号:所有符号){
198. Goto(当前项目集,当前符号) = Go(当前项目集,当前符号)
199. }
200. }
201. 其他位置标记为err
202. }
203. 主控函数(){
204. 初始化输入串栈
205. 初始化符号栈
206. 初始化状态栈
207. **while**(结束标记不为结束){
208. String i =  状态栈顶
209. String a = 输入串栈顶
210. String action = ActionTable.(i,a)
211. **if**(action() == **null**或者er ){
212. 报错
213. 标记结束
214. }
215. **else** **if**(action == acc ){
216. 成功
217. 标记结束
218. }
219. **else** **if**(action == Si){
220. i入状态栈
221. a到文法符号栈
222. }
223. **else** **if**(action == Ri){
224. 用index产生式规约
225. 符号栈中取出文法的右侧的长度的字符
226. 再压栈文法左边的符号
227. 状态栈.push( GOTO ( 状态栈.top() , 符号栈.top() ) )
228. }
229. **else**{
230. 报错
231. 标记结束
232. }
233. }
234. }

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;

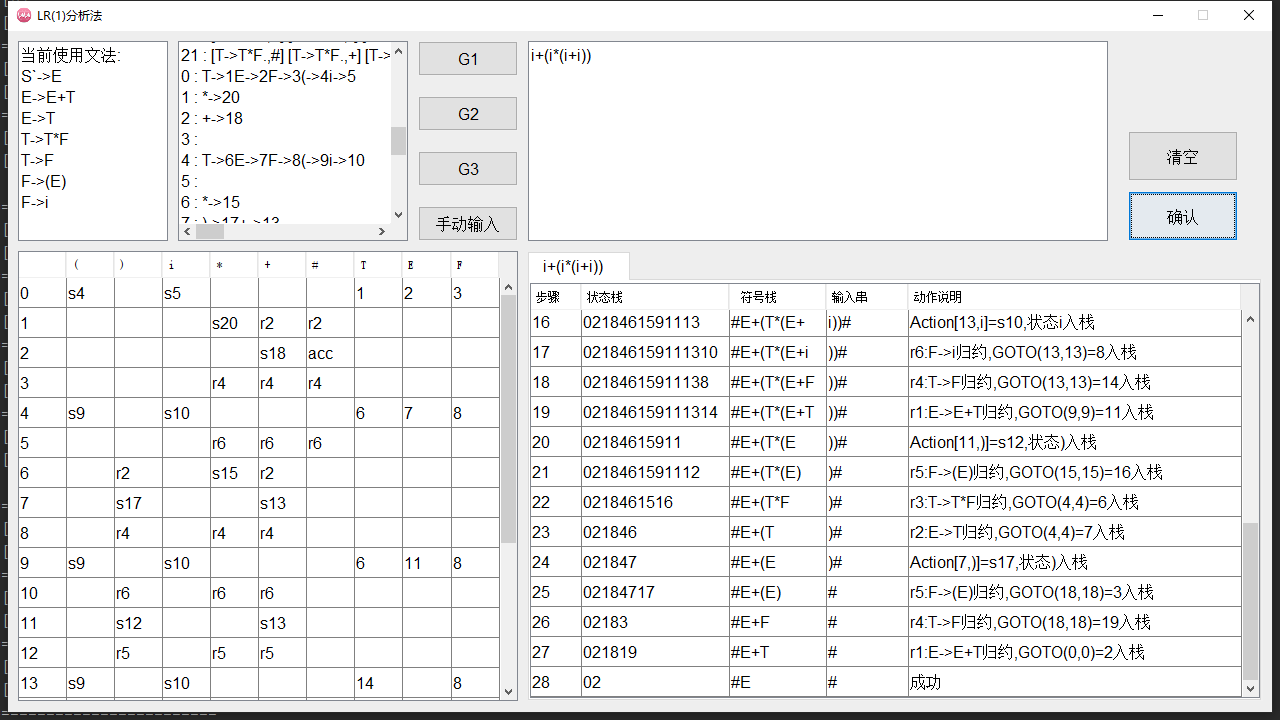
4. **import** java.util.\*;
5. **import** java.util.stream.Collectors;
7. **class** Solution {
8. **public** String grammarText;
9. **public** List<String[]> grammar = **new** ArrayList<>();//(T->SF)= {T,SF} forAll
10. **public** Set<String> nonTerminal = **new** HashSet<>();//非终结符
11. **public** Set<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++;

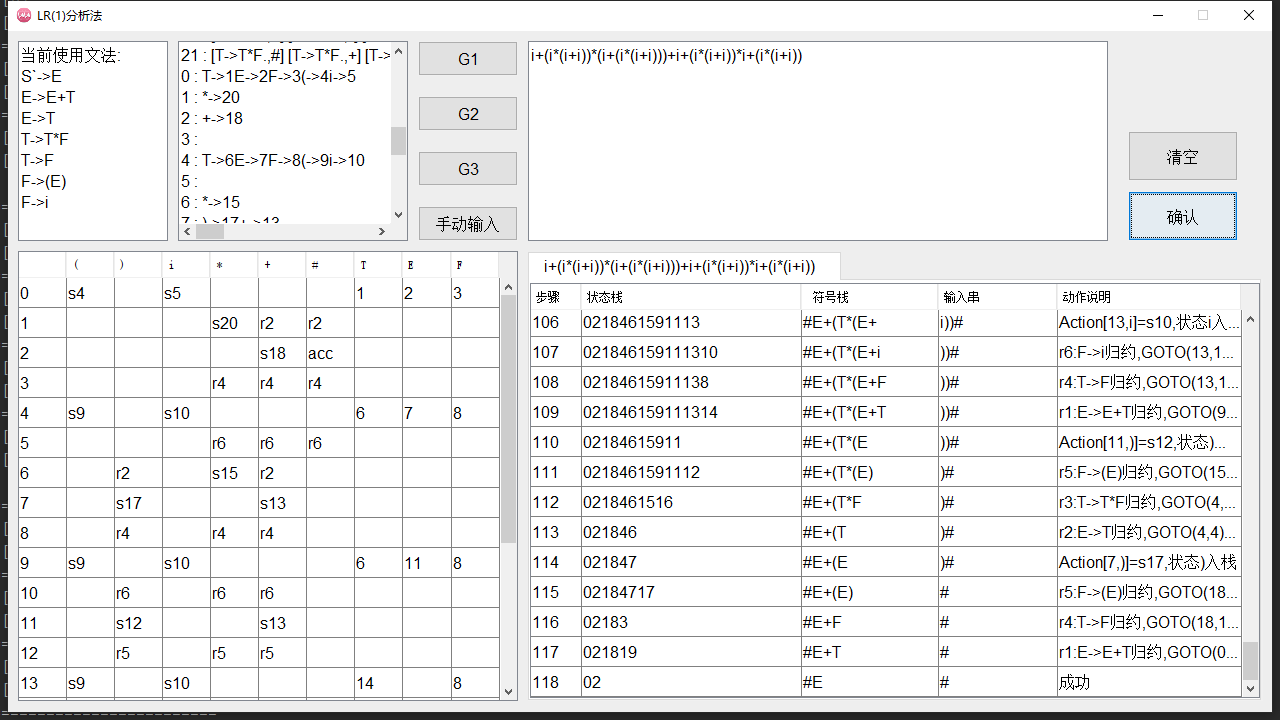
474. System.out.println("================================");
475. }
476. **return** processRecord;
477. }
478. Set<String> First(String beta,String a){
479. Set<String> res = **new** HashSet<>();
480. **if**(beta.length() == 0){
481. res.add(a);
482. **return** res;
483. }
484. **else** {
485. res.addAll(First(beta));
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. }

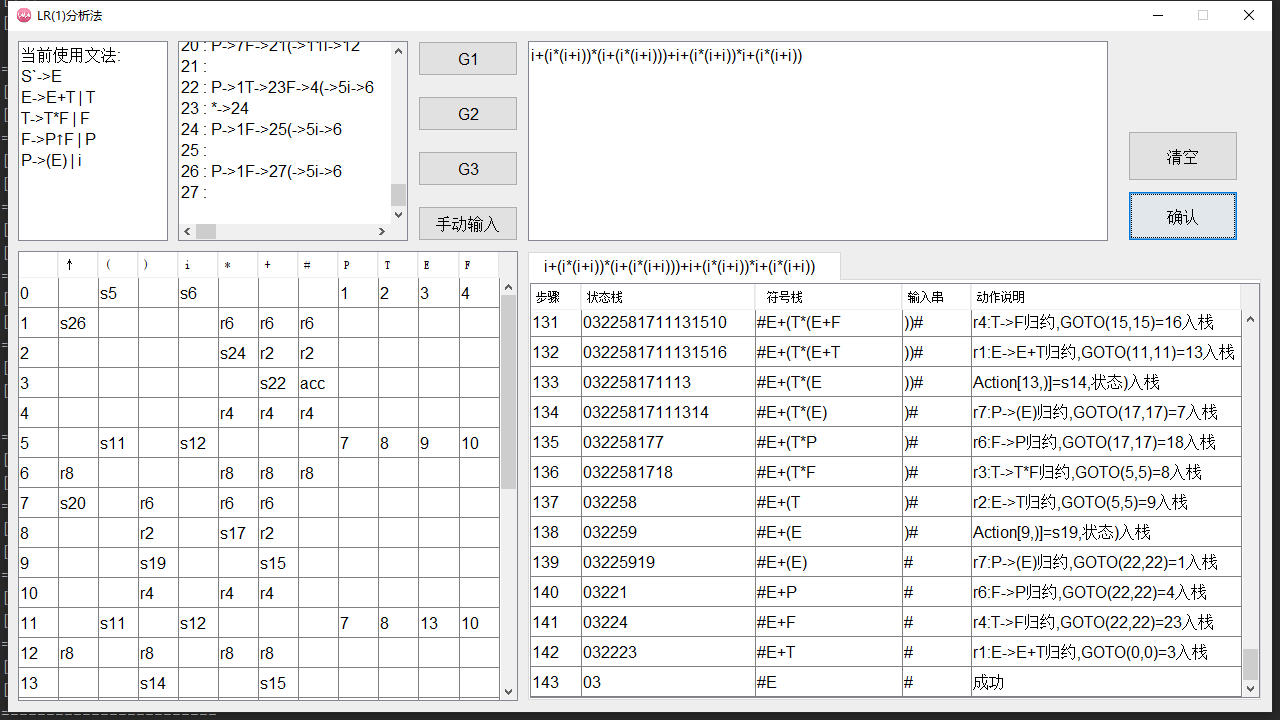
GUI.java

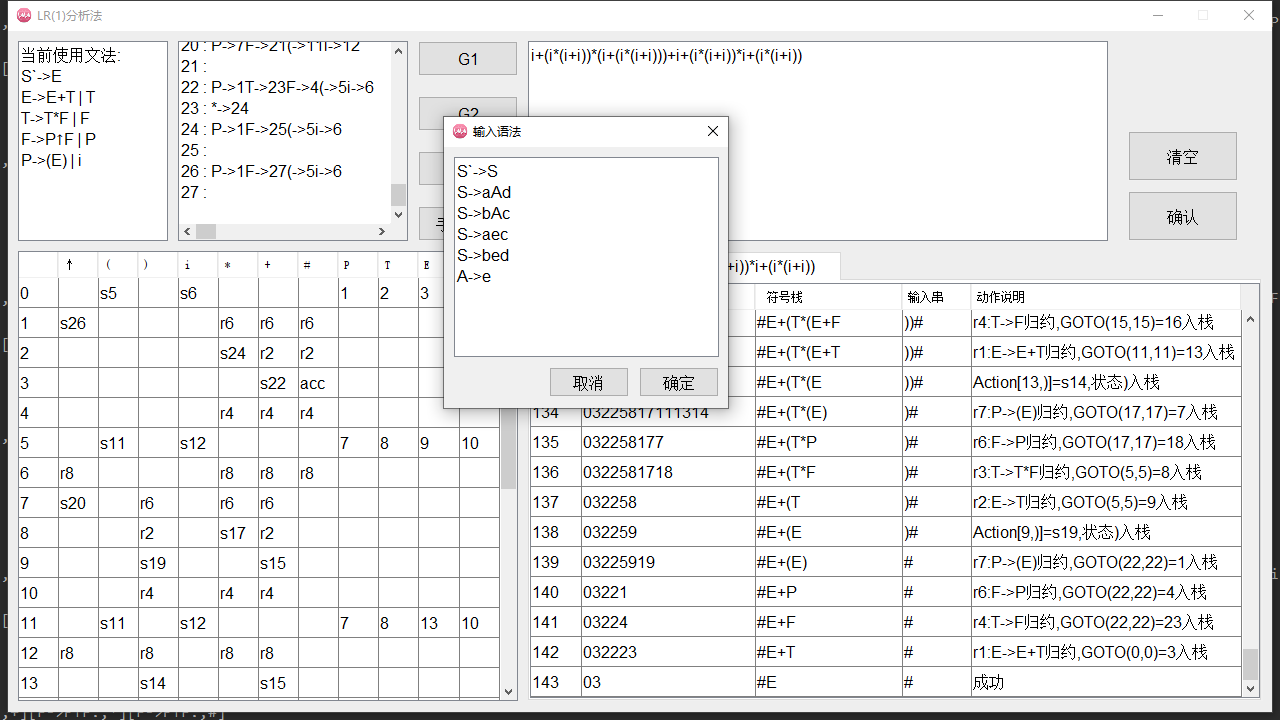
1. **package** 实验三\_LR1分析法;
3. **import** javax.swing.\*;
4. **import** java.awt.\*;
5. **import** java.awt.event.ActionEvent;
6. **import** java.awt.event.ActionListener;
7. **import** java.util.Vector;
8. **import** javax.swing.table.JTableHeader;
9. **import** javax.swing.table.TableColumn;
10. **import** 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。