合肥工業大學
HEFEI UNIVERSITY OF TECHNOLOGY

编译原理实验报告

学生姓名 ： 林天岳

学 号 ： 2017217893

班 级 ： 计算机科学与技术 2017-5

完成日期 ： 2019 年 10 月 22 日

实验三　LR(1)分析法

1.数据结构及算法描述

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

```
1.  class project{
2.      private int indexOfGrammar;
3.      private int indexOfNode;
4.      private String extSymbol;
5.      project(int indexOfGrammar,int index,String extSymbol){
6.          this.indexOfGrammar = indexOfGrammar;
7.          this.indexOfNode = index;
8.          this.extSymbol = extSymbol;
9.      }
10.     public String[] getGrammar() {//获取产生式
11.         return grammar.get(indexOfGrammar);
12.     }
13.     public int getIndex() {//获取所用产生式的编号
14.         return indexOfNode;
15.     }
16.     public String getHead(){//获取产生式左边
17.         return this.getGrammar()[0];
18.     }
19.     public String getExtSymbol() {
20.         return extSymbol;
21.     }
```

```
22.     public String getRight(){//获取产生式右边点后面的部分
23.         return this.getGrammar()[1].substring(indexOfNode);
24.     }
25.     public String getFirstSymbolAfterNode(){//获取右侧字符串的首字符
26.         if(this.getRight().length()<1){
27.             return "";
28.         }
29.         else {
30.             return this.getRight().substring(0,1);
31.         }
32.     }
33.     public String getRestStringAfterFirst(){//获取右侧首个字符的之后的字符串
34.         if(this.getRight().length()<2){
35.             return "";
36.         }
37.         else {
38.             return this.getRight().substring(1);
39.         }
40.     }
41.     @Override
42.     public String toString() {
43.         StringBuffer str = new StringBuffer(grammar.get(indexOfGrammar)[1]);

44.         str.insert(indexOfNode,".");
45.         str.insert(0,"["+grammar.get(indexOfGrammar)[0]+"->");
46.         str.append(","+extSymbol+"]");
47.         return str.toString();
48.     }
49.     @Override
50.     public boolean equals(Object obj) {
51.         if(!obj.getClass().equals(this.getClass())){
52.             return false;
53.         }
54.         project cmp = ((project)obj);
55.         if(  (cmp.indexOfGrammar == this.indexOfGrammar)  &&  (cmp.indexOfNo
    de==this.indexOfNode)  && (cmp.extSymbol.equals(this.extSymbol)) ){
56.             return true;
57.         }
58.         return false;
59.     }
60.     @Override
61.     public int hashCode() {
62.         String hash = indexOfGrammar+","+indexOfNode+","+extSymbol;
63.         return hash.hashCode();
```

```
64.     }
65. }//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
66. 
67. class projectSet {
68.     Set<project> projects;//集合
69.     Set<Integer> indexOfProjects;//
70.     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
71.     public projectSet(){
72.         projects = new HashSet<>();
73.         indexOfProjects = new HashSet<>();
74.         sons = new HashMap<>();
75.     }
76.     public boolean add(project project){
77.         if(projectList.contains(project)){
78.             indexOfProjects.add(projectList.indexOf(project));
79.             return projects.add(project);
80.         }
81.         else {
82.             return false;
83.         }
84.     }
85.     public Set<project> getSet() {
86.         return projects;
87.     }
88.     private String toCompare(){
89.         StringBuffer sb = new StringBuffer();
90.         projects.forEach(S->sb.append(S.toString()));
91.         return sb.toString();//比较用  set 相同即可认为是相同的集合
92.     }
93.     @Override
94.     public String toString() {
95.         return cProjectSets.indexOf(this)+"";
96.     }
97.     @Override
98.     public boolean equals(Object obj) {
99.         if(!obj.getClass().equals(this.getClass())){
100.             return false;
101.         }
102.         return this.toCompare().equals(((projectSet)obj).toCompare()) ;
103.     }
104.     @Override
105.     public int hashCode() {
```

```
106.         return this.toCompare().hashCode();
107.     }
108. }//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
109.
110.
111. 初始化文法
112. 读取文法,计算 First 集(使用实验 2 的算法即可)
113.
114. 初始化项目 List(){
115.     加入(S`->.S),#和(S`->S.,#)//开始文法 特殊情况 手动添加
116.     for(文法 A->BC:所有文法){
117.         for( index : 所有可以插入点的位置){
118.             for(文法 当前文法:所有文法){
119.             if(当前文法的右边含有 A){
120.                 把(A->BC,index,First(点后部的字符串))加入到项目集中
121.             }
122.         }
123.         }
124.     }
125. }
126. 初始化项目集 C(){
127.     初始化栈
128.     创建一个项目集
129.     把开始文法的项目放入其中
130.     计算它的闭包
131.     把这个项目放入栈中
132.     while(栈非空){
133.         当前项目集 = 栈.pop()
134.         for(String Symbol : 每个符号){
135.             创建一个新的项目集
136.             for(项目 : 当前项目集中的所有项目){
137.                 if(是形如 A->...•X... 的项目){
138.                     点向后移动一位创建新的项目,加入到新的项目集中
139.                 }
140.             }
141.             计算闭包(新的项目集)
142.             if(如果新的项目集为空){
143.                 标记当前项目集通过 Symbol 跳转到-1
144.             }
145.             else if(新的项目集已经存在){
146.                 标记当前项目集通过 Symbol 跳转到 项目集 List.indexOf(新的项目集)
147.             }
```

```
148.                 else{
149.                     把新的项目集加入到项目集 List 中
150.                     标记当前项目集通过 Symbol 跳转到 项目集 List.indexOf(新的项目
    集)
151.                     栈.push(新的项目集)
152.                 }
153.             }
154.         }
155. }
156. 计算闭包(项目集){
157.     新建栈
158.     项目集.forEach(栈::push)
159.     while(栈非空){
160.         当前项目 = 栈.pop()
161.         项目.add(当前项目);
162.         点后符号 = 当前项目的点后的第一个符号
163.         if(点后符号是非终结符){
164.             for(当前产生式:每个左边是 B 的产生式){
165.                 for(symbol:First(B 之后的部分,当前项目的展望符)){
166.                     new 新项目(当前产生式,0,symbol);
167.                     if(项目原本不存在项目集中){
168.                         栈.push(新项目)
169.                     }
170.                 }
171.             }
172.         }
173.     }
174. }
175. Go(项目集,符号){
176.     if(项目集通过(符号跳转) != -1){
177.         return 项目集通过(符号跳转)的项目集
178.     }
179.     else
180.         return 空集
181. }
182. 创建 Action 和 Goto 表(){
183.     for(当前项目集:所有项目集){
184.         for( 当前项目 : 所有项目 ){
185.             当前项目 刑如 [A->...•a...,b]
186.             if(a 是终结符){
187.                 ActionTable(当前项目.index,a) = S + Go(当前项目集,a).index
188.             }
189.             if(a 是""){
```

```
                ActionTable(当前项目.index,b) = R + 当前项目集.indexOf 语法
            }
        }
    }
    ActionTable(初始项目.index,#) = acc
    for(当前项目集:所有项目集){
        for(当前符号:所有符号){
            Goto(当前项目集,当前符号) = Go(当前项目集,当前符号)
        }
    }
    其他位置标记为 err
}
主控函数(){
    初始化输入串栈
    初始化符号栈
    初始化状态栈
    while(结束标记不为结束){
        String i =  状态栈顶
        String a = 输入串栈顶
        String action = ActionTable.(i,a)
        if(action() == null 或者 er ){
            报错
            标记结束
        }
        else if(action == acc ){
            成功
            标记结束
        }
        else if(action == Si){
            i 入状态栈
            a 到文法符号栈
        }
        else if(action == Ri){
            用 index 产生式规约
            符号栈中取出文法的右侧的长度的字符
            再压栈文法左边的符号
            状态栈.push( GOTO ( 状态栈.top() , 符号栈.top() ) )
        }
        else{
            报错
            标记结束
        }
    }
```

```
233. }
```

2.算法流程图


LR分析器结构：
输入串xxx...#
n1
.
.
.
1
Xn
.
.
.
X1
总控程
输出
ACTION
GOTO
0.#分别入状态栈和符号栈
用ip指向w#的第一个符号
令s是状态栈栈顶，
a是ip所指向的符号
action[s,a]=Ss'
是
把a和s'分别键入符号栈和状态栈；使ip前进到下一个字符.
否
是
action[s,a]=reduce A->1
分别从栈顶弹出|0|个符号,令s'是当前栈顶状态,把a和goto[s',A]先后键入栈中,输出产生式A->0.
否
action[A,a]=acc
否
是
结
出错处理

3.源码及测试结果

Main.java：

```
1.  package 实验三_LR1 分析法;
2.
3.  public class Main {
4.      public static void main(String[] args) {
5.          new Thread(() -> new Windows()).start();
6.      }
7.  }
```

Solution.java

```
1.  package 实验三_LR1 分析法;
2.
3.
4.  import java.util.*;
5.  import java.util.stream.Collectors;
6.
7.  class Solution {
8.      public String grammarText;
9.      public List<String[]> grammar = new ArrayList<>();//(T->SF)= {T,SF} forA
    ll
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.index
    OfNode==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));
159.
160.         allSymbol.addAll(terminal);
161.         allSymbol.addAll(nonTerminal);
162.         System.out.println("非终结符"+nonTerminal);
163.         System.out.println("终结符"+terminal);
164.         System.out.println();
165.     }
166.     private void setFirst(){
167.         nonTerminal.forEach(S->First.put(S,new HashSet<>()));
168.         terminal.forEach(S->First.put(S,new HashSet<>()));
169.         terminal.forEach(S->First.get(S).add(S));//终结符的 First 集是本身
170.         int FirstSize = 0;
171.         do{
172.             FirstSize = 0;
173.             for (String s1 : First.keySet()) {
174.                 FirstSize+=First.get(s1).size();
175.             }//记录原本大小
176.             for (String[] strings : grammar) {
177.                 String lam = strings[1];
178.                 String G = strings[0];
179.                 setSingleFirst(lam,G);//计算 First 集过程
180.             }
181.             for (String s1 : First.keySet()) {
182.                 FirstSize-=First.get(s1).size();
183.             }//计算修改后大小
184.         }while (FirstSize != 0);//如果大小不在变化 则停下
185.         System.out.println("First");
186.         for (String s : First.keySet()) {
187.             System.out.println(s+":"+First.get(s));
188.         }
189.     }
190.     private void setSingleFirst(String lam,String G){
191.         String first = lam.substring(0,1);
192.         if(terminal.contains(first)){//终结符
193.             First.get(G).add(first);
194.         }
195.         else if(first.equals("ε")){//符号空
```

```
196.             First.get(G).add("ε");
197.         }
198.         else if(nonTerminal.contains(first)){//非终结符
199.             First.get(G).addAll(First.get(first).stream().filter(S->!S.equa
    ls("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(); index
    OfGrammar++) {
212.             for (int indexOfNode = 0; indexOfNode <= grammar.get(indexOfGra
    mmar)[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(st
    rings),0,b);
246.                            if(!closure.getSet().contains(newProject)){
247.                                stack.push(newProject);
248.                            }
249.                        }
250.                    }
251.                }
252.            }
253.
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.getIn
    dex()==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.indexOfGra
    mmar,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);
286.
287.                 projectSet nextI = new projectSet();
288.                 nextI.projects.addAll(temp);
289.                 extendSingleClosure(nextI);
290.                 //System.out.println("最终 set "+nextI.getSet());
291.
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
309.
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.getFirstSym
    bolAfterNode().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),"#"),"a
    cc");
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()).revers
    e()+"\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
460.
461.                 String nextSta = GOTO.get(new point(statusStack.getTop(),sy
    mbolStack.getTop()));
462.                 currentStep[4] = action+":"+grammar.get(index)[0]+"->"+gram
    mar.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++;
472.
473.
474.             System.out.println("================================");
475.         }
476.         return processRecord;
477.     }
478.     Set<String> First(String beta,String a){
479.         Set<String> res = new HashSet<>();
480.         if(beta.length() == 0){
481.             res.add(a);
482.             return res;
483.         }
484.         else {
485.             res.addAll(First(beta));
486.             if(First(beta).contains("ε")){
487.                 res.add(a);
488.                 res.remove("ε");
489.             }
490.         }
491.         return res;
492.     }
493.     Set<String> First(String s){//单个字串的 First 集
494.         Set<String> res = new HashSet<>();
495.         if(s.length() == 0){
496.             res.add("#");
497.             return res;
498.         }
499.         for (String symbol : s.split("")) {
500.             if(First.containsKey(symbol)){
501.                 res.addAll(First.get(symbol));
502.                 if(!First.get(symbol).contains("ε")){//如果 当前的字 可以推出
    空 看向字串的下一个字
503.                     break;
504.                 }
505.             }
506.         }
507.         Boolean elicitNull = true;
508.         for (String symbol : s.split("")){
```

```
509.             if(First.containsKey(symbol)  && !First.get(symbol).contains("ε
    ")){
510.                 elicitNull = false;
511.                 break;
512.             }
513.         }
514.         if(!elicitNull){
515.             res.remove("ε");
516.         }//只有 所有的字 都能推出空 这个字串才可以推出空
517.         return res;
518.     }
519. }
520.
521. class point {
522.     String head, tail;
523.
524.     point(String head, String tail) {
525.         this.head = head;
526.         this.tail = tail;
527.     }
528.
529.     public String getHead() {
530.         return head;
531.     }
532.
533.     public String getTail() {
534.         return tail;
535.     }
536.
537.     @Override
538.     public int hashCode() {
539.         return (head + "->" + tail).hashCode();
540.     }
541.
542.     @Override
543.     public boolean equals(Object o) {
544.         if (o.getClass() != this.getClass()) {
545.             return false;
546.         } else {
547.             return ((point) o).getHead().equals(this.head) && ((point) o).g
    etTail().equals(this.tail);
548.         }
549.     }
550.
```

```
}

class MyStack {
    List<String> s;

    MyStack() {
        s = new LinkedList<>();
    }

    void push(String value) {
        s.add(value);
    }

    void push(String... values) {
        for (String value : values) {
            push(value);
        }
    }

    String pop() {
        return s.remove(s.size() - 1);
    }

    String getTop() {
        return s.get(s.size() - 1);
    }

    @Override
    public String toString() {
        StringBuffer sb = new StringBuffer();
        for (String value : s) {
            sb.append(value);
        }
        return sb.toString();
    }

    public Boolean isEmpty() {
        return s.size() == 0;
    }
}
```

GUI.java

```java
package 实验三_LR1 分析法;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableColumn;
import java.util.*;

class Windows extends JFrame {
    Solution sol;
    JButton clear,confirm;
    JTextArea grammarTextArea,inputTextArea,projectArea;
    JScrollPane tablePane;
    JTabbedPane resultTable;
    JScrollPane projectList;

    Windows(){
        setVisible(false);
        try{
            setIconImage(new ImageIcon("bilibili.PNG").getImage());
            Font f = new Font("Yahei Consolas Hybrid",Font.PLAIN,16);
            String   names[]={ "MenuBar","Menu","MenuItem", "TextArea", "Button", "ScrollPane", "Table","TabbedPane"};
            for (String item : names) {
                UIManager.put(item+ ".font",f);
            }
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
        }catch(Exception e){}
        sol = new Solution("S`->E\n"+
                "E->E+T\n" +
                "E->T\n" +
                "T->T*F\n" +
                "T->F\n" +
                "F->(E)\n" +
                "F->i");
        init();
        setSize(1280,720);//初始大小
        setLocation(100,80);//初始位置
        setVisible(true);//是否可视
        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 "};
```

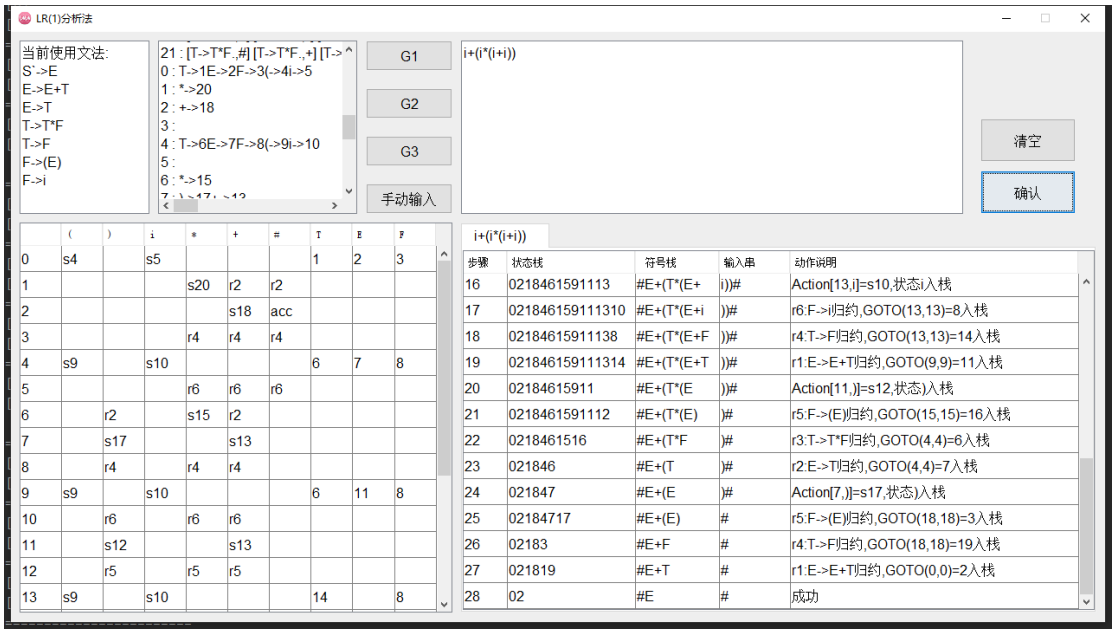
```
                String[][] data = new String[result.size()][5];
                int i = 0;
                for (String[] strings : result) {
                    data[i] = strings;
                    i++;
                }
                JTable singleResult = new JTable(data,head);
                FitTableColumns(singleResult);
                singleResult.setRowHeight(30);
                JScrollPane resultTablePane = new JScrollPane(singleResult);

                resultTable.addTab("  "+inputText+"     ",resultTablePane);

            }
        });
        JButton G1 = new JButton("G1");
         G1.setBounds(410,10,100,35);
         add(G1);
         G1.addActionListener(actionEvent -> {
             updateGrammar("S`->E\nE->E+T\nE->T\nT->T*F\nT->F\nF->(E)\nF->i"
);
         });
         JButton G2 = new JButton("G2");
         G2.setBounds(410,65,100,35);
         add(G2);
         G2.addActionListener(actionEvent -> {
             updateGrammar("S`->E\nE->E+T | T\nT->T*F | F\nF->P↑F | P\nP->(E
) | i\n");
         });
         JButton G3 = new JButton("G3");
         G3.setBounds(410,120,100,35);
         add(G3);
         G3.addActionListener(actionEvent -> {
             updateGrammar("S`->S\nS->aAd\nS->bAc\nS->aec\nS->bed\nA->e");
         });
         JButton more = new JButton("手动输入");
         more.setBounds(410,175,100,35);
         add(more);
         class MyDialog extends JDialog implements ActionListener{
             JTextArea input;
             JButton confirm,cancel;
             String title;
             MyDialog(){
                 setLayout(null);
```

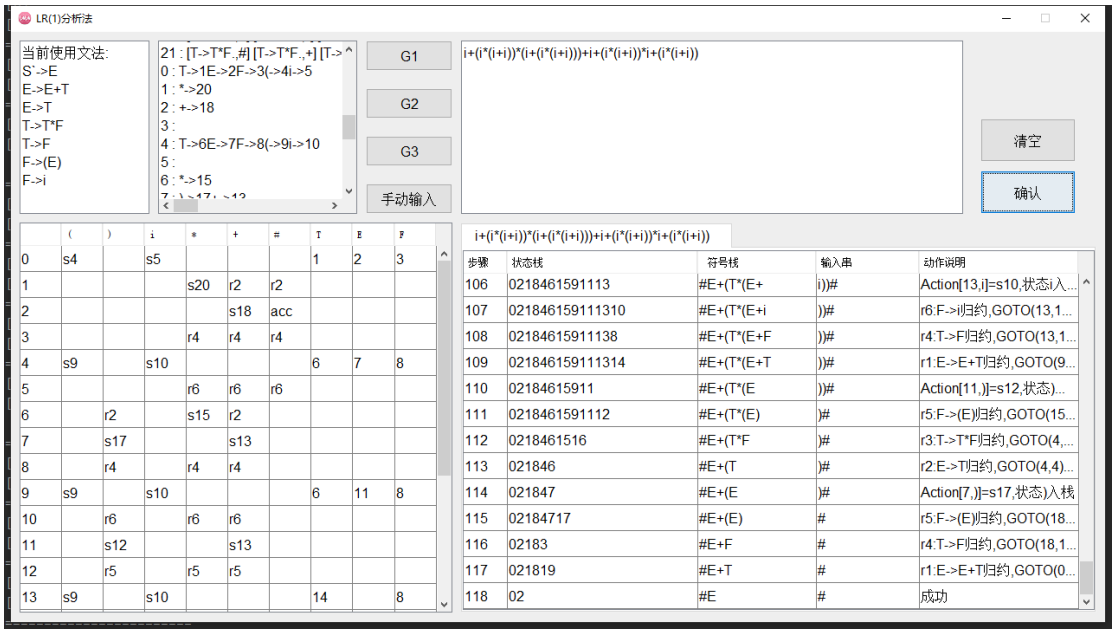
```
                setResizable(false);
                setIconImage(new ImageIcon("bilibili.PNG").getImage());
                setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                setTitle("输入语法");
                input=new JTextArea();
                JScrollPane jScrollPane = new JScrollPane(input);
                jScrollPane.setBounds(10,10,265,200);
                add(jScrollPane);
                class confirmListener implements ActionListener{
                    @Override
                    public void actionPerformed(ActionEvent e){
                        updateGrammar(input.getText());
                        setVisible(false);
                    }
                }
                confirm=new JButton("确定");
                confirm.addActionListener(new confirmListener());
                confirm.setBounds(195,220,80,30);
                add(confirm);
                class cancelListener implements ActionListener{
                    @Override
                    public void actionPerformed(ActionEvent e){
                        setVisible(false);
                    }
                }
                cancel=new JButton("取消");
                cancel.addActionListener(new cancelListener());
                cancel.setBounds(105,220,80,30);
                add(cancel);
                setBounds(600,260,300,300);
                setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            }
            public void actionPerformed(ActionEvent e){
                setVisible(true);
            }
        }
        more.addActionListener(new MyDialog());
    }
    private void initInputArea(){
        inputTextArea = new JTextArea();
        inputTextArea.setLineWrap(true);
        JScrollPane inputAreaPane = new JScrollPane(inputTextArea);
        inputAreaPane.setBounds(520,10,580,200);
        add(inputAreaPane);
```

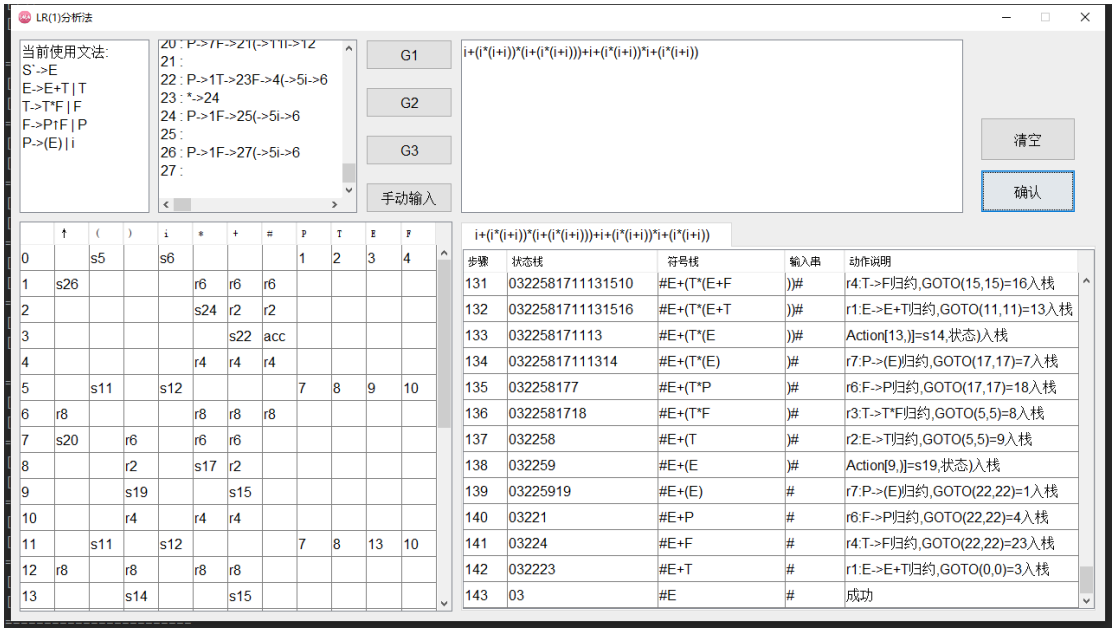
```
170.     }
171.     private void initResultTable(){
172.         resultTable = new JTabbedPane();
173.         resultTable.setBounds(520,220,735,450);
174.         add(resultTable);
175.     }
176.     private void initProjectPane(){
177.         projectArea = new JTextArea();
178.         projectList = new JScrollPane(projectArea);
179.         projectList.setBounds(170,10,230,200);
180.         add(projectList);
181.         updateProjectPane();
182.
183.     }
184.     private void updateProjectPane(){
185.         StringBuffer sb = new StringBuffer();
186.         int count = 0;
187.         for (Solution.projectSet projectSet : sol.cProjectSets) {
188.             sb.append( (count++ )+" : ");
189.             for (Solution.project project : projectSet.getSet()) {
190.                 sb.append(project+" ");
191.             }
192.             sb.append("\n");
193.         }
194.          count = 0;
195.         for (Solution.projectSet projectSet : sol.cProjectSets) {
196.             sb.append( (count++ )+" : ");
197.             for (String s : sol.allSymbol) {
198.                 int index = projectSet.sons.get(s);
199.                 if(index !=-1){
200.                     sb.append(s+"->"+index);
201.                 }
202.             }
203.             sb.append("\n");
204.         }
205.         projectArea.setText(sb.toString());
206.         projectList.updateUI();
207.     }
208.     private void updateActTable(){
209.         String[] head =sol.getHeader();
210.         String [][] data = sol.getActionAndGoTo();
211.         for (int i = 0; i < data.length; i++) {
212.             for (int j = 0; j < data[i].length; j++) {
213.                 if(data[i][j].equals("err")){
```

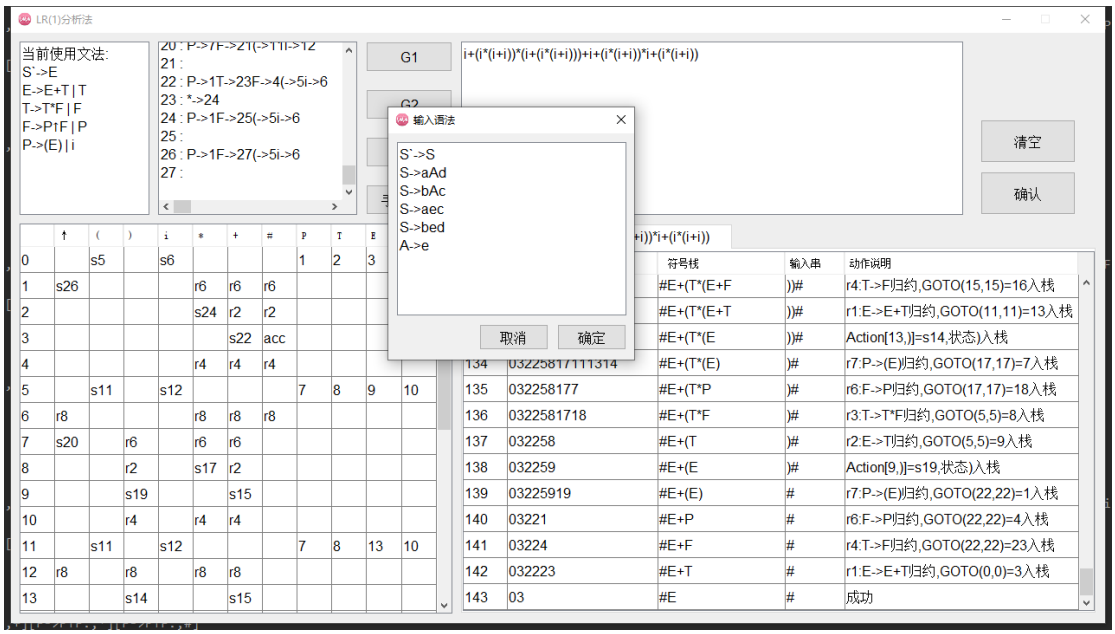
```
214.                     data[i][j] = "";
215.                 }
216.             }
217.         }
218.         JTable actGoTable;
219.         actGoTable = new JTable(data,head);
220.         actGoTable.setRowHeight(30);
221.         tablePane.setViewportView(actGoTable);
222.         tablePane.updateUI();
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.getIden
    tifier());
237.             int width = (int) myTable.getTableHeader().getDefaultRenderer()

238.                     .getTableCellRendererComponent(myTable, column.getIdent
    ifier()
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, co
    l).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：


LR(1)分析法
当前使用文法:
S`->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
21 : [T->T*F.,#] [T->T*F.,+] [T->
0 : T->1E->2F->3(->4i->5
1 : *->20
2 : +->18
3 :
4 : T->6E->7F->8(->9i->10
5 :
6 : *->15
G1
G2
G3
手动输入
i+(i*(i+i))
清空
确认
i+(i*(i+i))
步骤 状态栈 符号栈 输入串 动作说明
16 0218461591113 #E+(T*(E+ i))# Action[13,i]=s10,状态i入栈
17 021846159111310 #E+(T*(E+i))# r6:F->i归约,GOTO(13,13)=8入栈
18 02184615911138 #E+(T*(E+F))# r4:T->F归约,GOTO(13,13)=14入栈
19 021846159111314 #E+(T*(E+T))# r1:E->E+T归约,GOTO(9,9)=11入栈
20 02184615911 #E+(T*(E))# Action[11,)]=s12,状态)入栈
21 0218461591112 #E+(T*(E))# r5:F->(E)归约,GOTO(15,15)=16入栈
22 0218461516 #E+(T*F)# r3:T->T*F归约,GOTO(4,4)=6入栈
23 021846 #E+(T)# r2:E->T归约,GOTO(4,4)=7入栈
24 021847 #E+(E)# Action[7,)]=s17,状态)入栈
25 02184717 #E+(E) # r5:F->(E)归约,GOTO(18,18)=3入栈
26 02183 #E+F # r4:T->F归约,GOTO(18,18)=19入栈
27 021819 #E+T # r1:E->E+T归约,GOTO(0,0)=2入栈
28 02 #E # 成功


LR(1)分析法
当前使用文法:
S`->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
21 : [T->T*F.,#] [T->T*F.,+] [T->
0 : T->1E->2F->3(->4i->5
1 : *->20
2 : +->18
3 :
4 : T->6E->7F->8(->9i->10
5 :
6 : *->15
G1
G2
G3
手动输入
i+(i*(i+i))*(i+(i*(i+i)))+i+(i*(i+i))*i+(i*(i+i))
清空
确认
i+(i*(i+i))*(i+(i*(i+i)))+i+(i*(i+i))*i+(i*(i+i))
步骤 状态栈 符号栈 输入串 动作说明
106 0218461591113 #E+(T*(E+ i))# Action[13,i]=s10,状态i入...
107 021846159111310 #E+(T*(E+i))# r6:F->i归约,GOTO(13,1...
108 02184615911138 #E+(T*(E+F))# r4:T->F归约,GOTO(13,1...
109 021846159111314 #E+(T*(E+T))# r1:E->E+T归约,GOTO(9...
110 02184615911 #E+(T*(E))# Action[11,)]=s12,状态)...
111 0218461591112 #E+(T*(E))# r5:F->(E)归约,GOTO(15...
112 0218461516 #E+(T*F)# r3:T->T*F归约,GOTO(4,...
113 021846 #E+(T)# r2:E->T归约,GOTO(4,4)...
114 021847 #E+(E)# Action[7,)]=s17,状态)入栈
115 02184717 #E+(E) # r5:F->(E)归约,GOTO(18...
116 02183 #E+F # r4:T->F归约,GOTO(18,1...
117 021819 #E+T # r1:E->E+T归约,GOTO(0...
118 02 #E # 成功


LR(1)分析法
当前使用文法:
S`->E
E->E+T | T
T->T*F | F
F->P↑F | P
P->(E) | i
20 : P->7F->21(->11T->12
21 :
22 : P->1T->23F->4(->5i->6
23 : *->24
24 : P->1F->25(->5i->6
25 :
26 : P->1F->27(->5i->6
27 :
G1
G2
G3
手动输入
i+(i*(i+i))*(i+(i*(i+i)))+i+(i*(i+i))*i+(i*(i+i))
清空
确认
步骤 状态栈 符号栈 输入串 动作说明
131 0322581711131510 #E+(T*(E+F))# r4:T->F归约,GOTO(15,15)=16入栈
132 0322581711131516 #E+(T*(E+T))# r1:E->E+T归约,GOTO(11,11)=13入栈
133 032258171113 #E+(T*(E))# Action[13,)]=s14,状态)入栈
134 03225817111314 #E+(T*(E))# r7:P->(E)归约,GOTO(17,17)=7入栈
135 032258177 #E+(T*P)# r6:F->P归约,GOTO(17,17)=18入栈
136 0322581718 #E+(T*F)# r3:T->T*F归约,GOTO(5,5)=8入栈
137 032258 #E+(T)# r2:E->T归约,GOTO(5,5)=9入栈
138 032259 #E+(E)# Action[9,)]=s19,状态)入栈
139 03225919 #E+(E) # r7:P->(E)归约,GOTO(22,22)=1入栈
140 03221 #E+P # r6:F->P归约,GOTO(22,22)=4入栈
141 03224 #E+F # r4:T->F归约,GOTO(22,22)=23入栈
142 032223 #E+T # r1:E->E+T归约,GOTO(0,0)=3入栈
143 03 #E # 成功


输入语法
S`->S
S->aAd
S->bAc
S->aec
S->bed
A->e
取消
确定

4.实验收获

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