**实 验 报 告**

**课程名称** 编译原理

**实验项目** 自顶向下语法分析器的设计与实现

**实验仪器**  PC机

**系 别**  计算机

**专 业**  计算机科学与技术

**班级/学号**  计科1606班/2016011440

**学生姓名**  刘逸芸

**实验日期**  2019/6/3

**成 绩**

**指导教师**  范艳芳

**实验二 自顶向下语法分析器的设计与实现**

**一、实验目的**

语法分析是编译程序中的核心部分。本实验通过设计一个典型的自顶向下语法分析程序——LL(1) 语法分析程序，进一步理解并掌握语法分析的原理和实现技术。

**二、实验原理：**

语法分析是编译过程的核心部分。他的任务是在词法分析识别出单词符号串的基础上，分析并判定程序的语法结构是否符合语法规则。语法分析器的工作本质就是按文法的产生式，识别输入串是否为一个句子。

按照生成语法树的方向不同，常用的语法分析方法有两类：自顶向下分析和自底向上分析。自顶向下分析是从文法的开始符出发，试图推导出与输入单词串相匹配的句子。自底向上分析也称移进 — 归约分析方法，从输入单词串开始，试图归约到文法的开始符。本次实验主要使用自顶向下分析的方法。

自上而下分析的主旨是，对任何输入串，试图用一切可能的方法，从文法开始符号（根结）出发，自上而下地为输入串建立一棵语法树，或者为输入串寻找一个最左推导。自上而下分析方法不允许文法含有任何左递归。LL(1)文法就是一种不带回溯的自上而下分析的文法，也称预测分析法。LL(1)文法既不是二义性的，也不含左递归。

预测分析法（LL(1)方法）的基本思想是：从文法开始符S出发，从左到右扫描源程序，每次通过向前查看 1 个字符，选择合适的产生式，生成句子的最左推导。

**三、程序功能与框架**

程序的主要功能是对一个输入的文法，进行语法分析，求解first集和follow集，然后构造预测分析表，判断文法是否为LL(1)文法。再判断输入的语句是否为该文法的语句。

本程序是通过Java实现的，主要框架就是实现预测分析法的整个过程框架，包括求Vt集和Vn集，求First集和Follow集，求文法的LL1预测分析表，以及求句子的预测分析过程。

**四、设计说明**

1. 数据结构

本程序定义了一个MyLL1类，用来实现分析过程，myLL1的UML图如图1所示。这个类中用到的数据结构有string数组、HashMap和HashSet，HashMap的键值对可以很方便的表示每一个非终结符的所有产生式、first集和follow集，而HashSet可以很方便的遍历所有的非终结符和终结符。

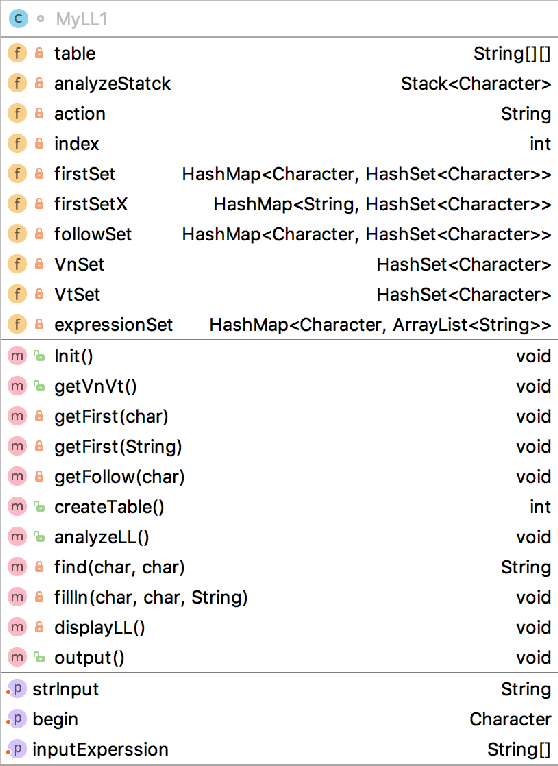


图2 myLL1的UML图

2. 主要变量

myLL1的数据域主要有：

String[][] table用来存储预测分析表，

Stack analyzeStack 表示符号栈，

String action 表示下一步动作，

用HashMap来存储first集、follow集和生成式，

用HashSet来存储Vt集和Vn集，

String strInput为输入的语句（以“#”结尾），

String[] inputExpression 为输入的文法（以“$结尾”），

Character begin为文法的开始符，默认输入的文法的第一句的第一个符号就是文法的开始符。

3. 算法思路

首先，用户通过控制台，根据提示输入文法（以“$结尾”）和待分析的语句（以“#”结尾）。根据输入的文法，获得了开始符，然后遍历所有的文法句子，求非终结符集和终结符集。“->”左边的都是非终结符，对于“->”右边的所有符号，不是非终结符的就都是终结符了。再遍历文法句子，就可以获取所有非终结符的生成式，“->”左边的为key，对于key相同的所有句子，“->”左边的存储在一个arraylist中，然后每一个key都会对应一个array list，list中就是这个key（非终结符）的所有产生式。获取了所有的产生式后，就可以开始构造非终结符的first集。First集的构造规则为：

（1）若X为终结符，则FIRST（X）={X}

（2）若X为非终结符，且有产生式X->a…，则把a加入到FIRST(X)中；若空串也是X的一个产生式，则把空也加入到FIRST(X)中

（3）若X->Y…是一个产生式且Y为非终结符，则将FIRST(Y)中的所有非空元素都加到FIRST(X)中；若X->Y1Y2…Yk是一个产生式，Y1，…，Yi-1都是非终结符，而且对于任何[1,i-1]中的j，FIRST(Yj)都包含了空，则把FIRST(Yi)中的所有非空元素添加到FIRST(X)中；特别是，若所有的FIRST(Yj)均含有空，则把空添加到FIRST(X)中。

由于在遍历过程中，FIRST集会不断扩大，因此我们采取递归回溯的方法，对于每一个非终结符，如果其某条产生式中遇到的非终结符的FIRST集还没有求解，则先求解改非终结符的FIRST集，并持续递归，直到求解出来了某个非终结符的FIRST集，开始回溯。最后解得所有非终结符的FIRST集。

求解了非终结符的FIRST集后，求非终结符的FOLLOW集。对于文法G的每个非终结符A构造FOLLOW(A)的规则是：

（1）对于文法的开始符号S，置#于FOLLOW(S)中

（2）若A->aBb是一个产生式，则把FIRST(b)中所有不为空的元素添加到FOLLOW(A)中。

（3）若A->aB是一个产生式，或A->aBb是一个产生式且b为空，则把FOLLOW(A)添加到 FOLLOW(B)中。

同理，FOLLOW集也在不断扩大，故也采用递归回溯的方法。

之后就可以开始求文法的预测分析表了。构造分析表M的算法是：

（1）对文法G的每个产生式A->x执行第2、3步

（2）对每个终结符a属于first(x),把A->x加入M[A,a]中

（3）若～属于first(x),则对任何b属于follow(A)，把A->~加入M[A,b]中

（4）把所有无定义的项标上出错标志

由于LL1文法G的预测分析表M不含多重定义入口，因此如果在构造预测分析表的过程中，发现有多重定义入口，则该文法不是LL1的，程序就会结束。

最后，自顶向下分析输入的语句。首先把#和文法开始符号推进符号栈中，然后依次读入句子的每一个输入符号a，取栈顶符号X，如果X为终结符，只有X与a相同时，X出栈，否则分析出错，该语句不符合文法，且当X=a=#时，分析成功，停止分析过程；如果X为非终结符，查看分析表M，如果M[A,a]存放着关于X的一个产生式，那么X先出栈，然后将产生式的右部符号串按反序意义入栈；如果M[A,a]中为错误标志，分析出错，该语句不符合文法；如果M[A,a]中为空，则X出栈后，无任何元素入栈。

由于完整的算法流程图太长，故下图只包含了求终结符、非终结符以及求FIRST集的算法流程图。

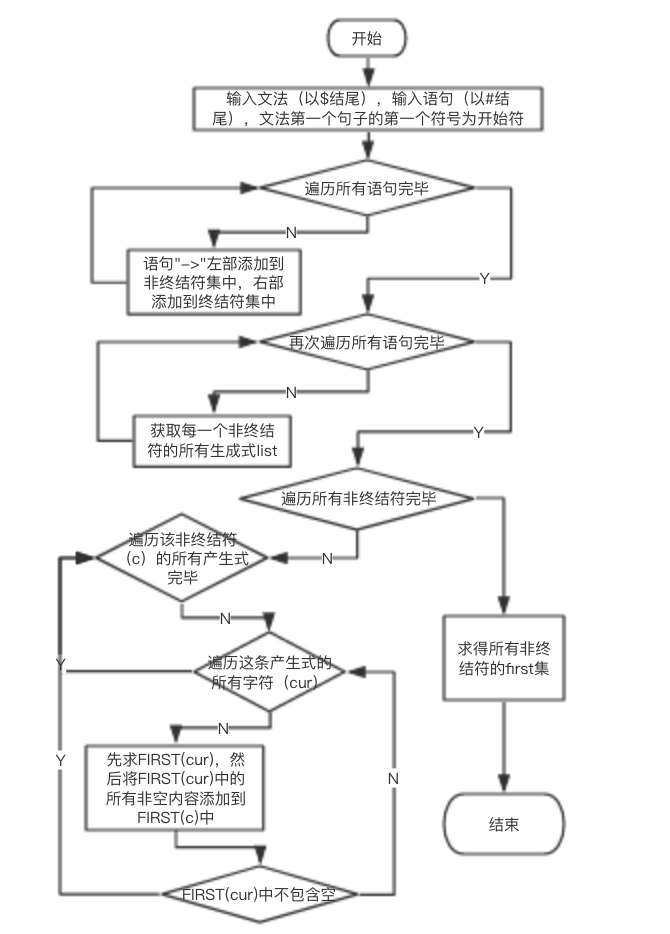


图2 部分算法流程图

**五、用户操作指南**

在IDE中运行LL1.java文件，根据控制台的提示，先输入文法，一行一句，以$结尾，后输入句子，以#结尾。然后能在控制台看到运行结果，first集、follow集、预测分析表和预测分析过程。如果文法不是LL1的，在输出预测分析表的时候，会输出“文法不是LL(1)文法”；如果语句不符合文法，会输出“不符合文法”，并能指出错误的位置。

**六、调试报告**

本次实验中，只实现了用户输入一个文法，语法分析器直接分析文法是否为LL1文法，并自顶向下分析用户输入的语句是否符合文法。我们知道，自顶向下的分析法存在很多困难和缺点。本实验只改进了能够在报告分析不成功的时候指出输入串中出错的确切位置，还存在缺陷有：不能分析包含左递归的文法；未消除回溯。在后续的改进中，可以实现这两个功能。

**七、测试数据及运行结果**

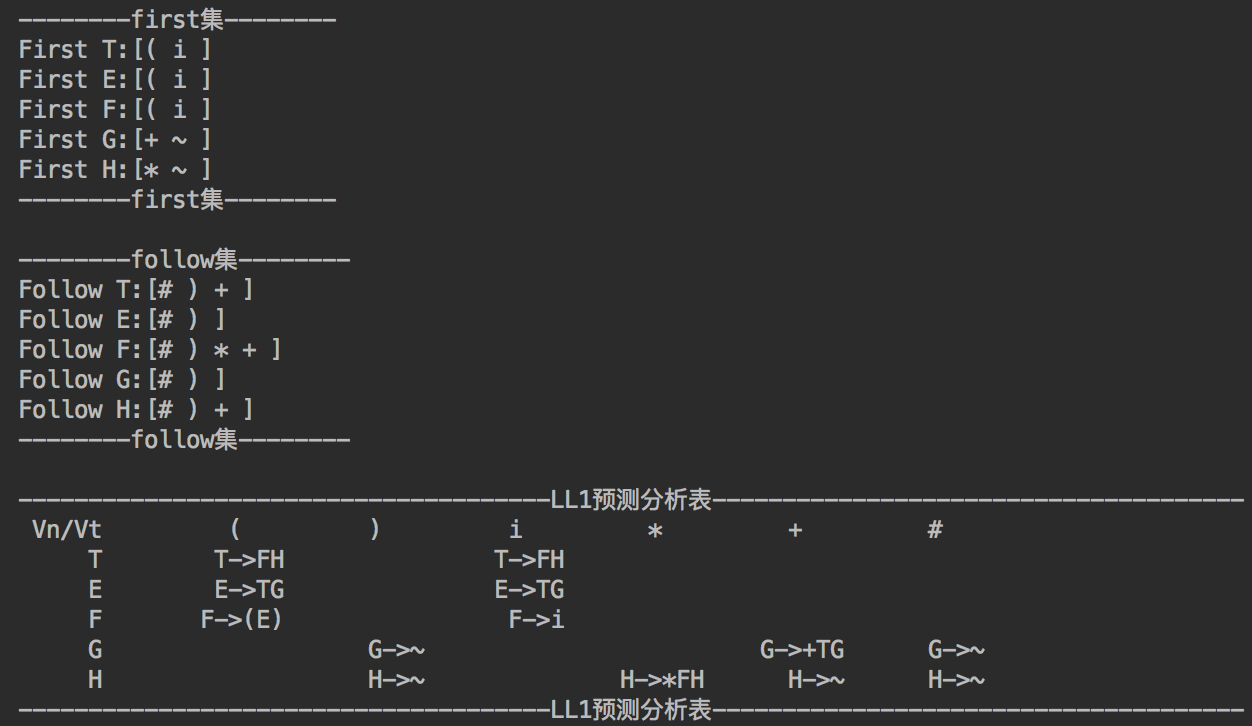
测试案例1:

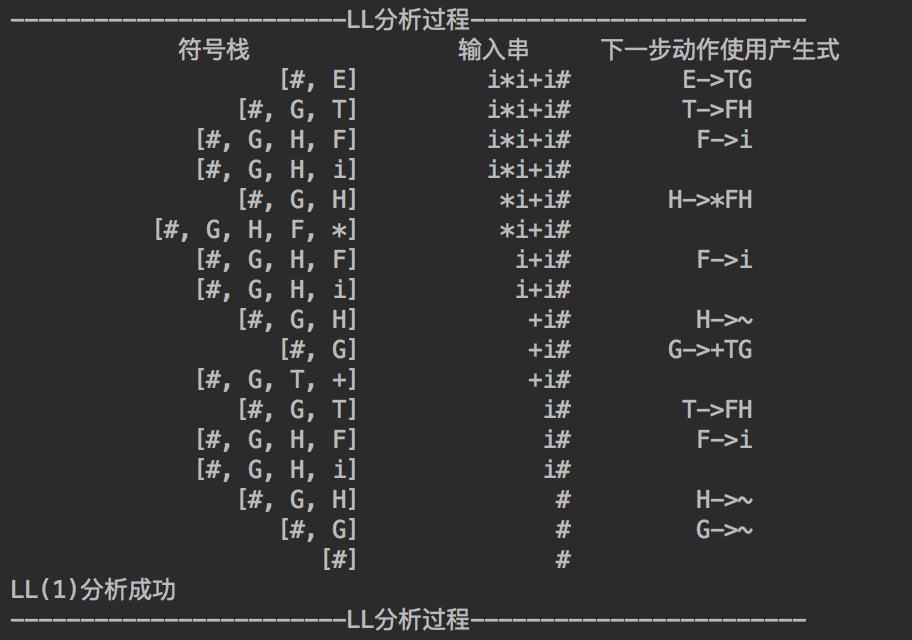
文法： E->TG G->+TG G->~ T->FH

H->\*FH H->~ F->(E) F->i $

语句：i\*i+i#

运行结果：





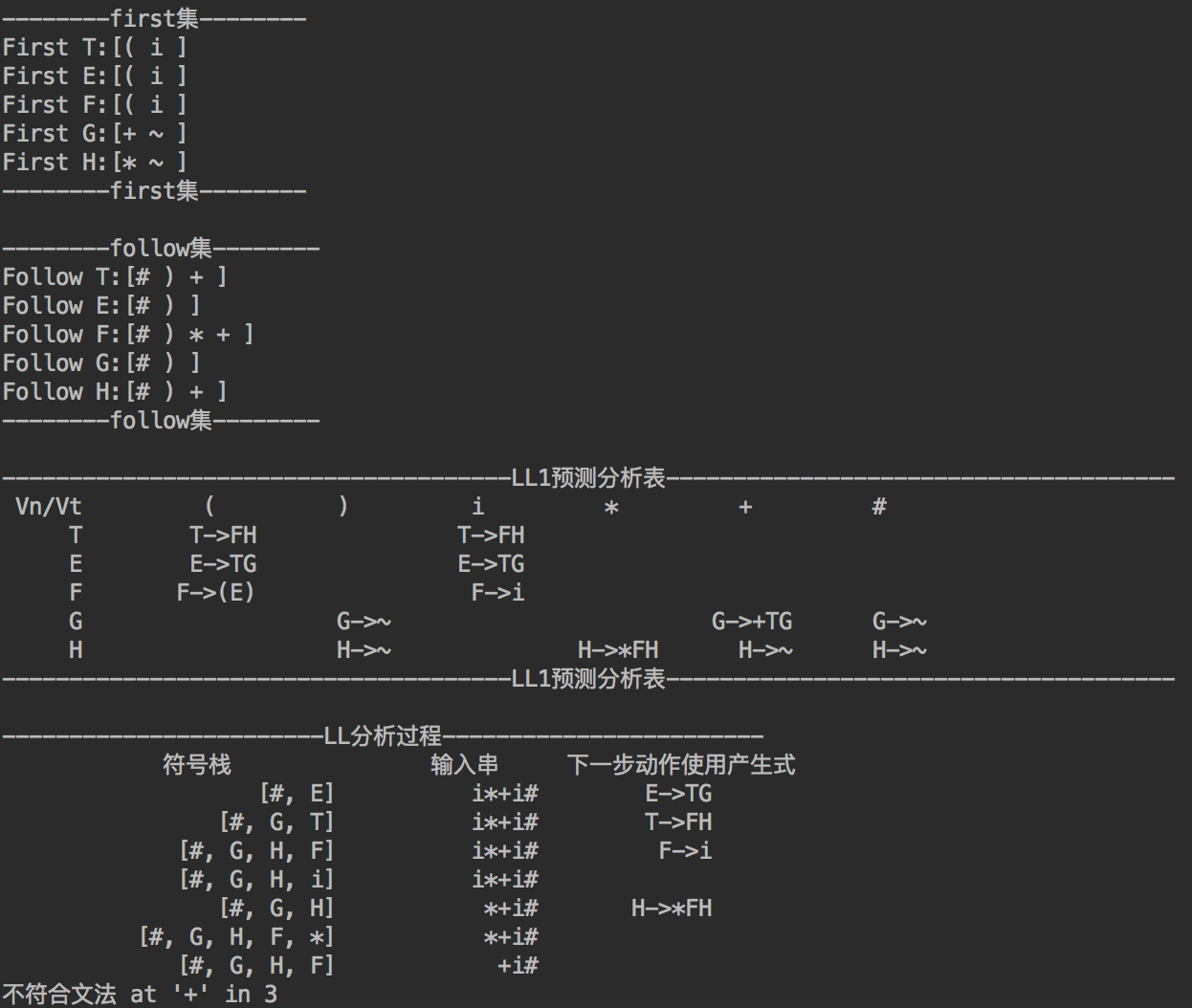
测试案例2:

文法： E->TG G->+TG G->~ T->FH

H->\*FH H->~ F->(E) F->i $

语句：i\*+i#

运行结果：

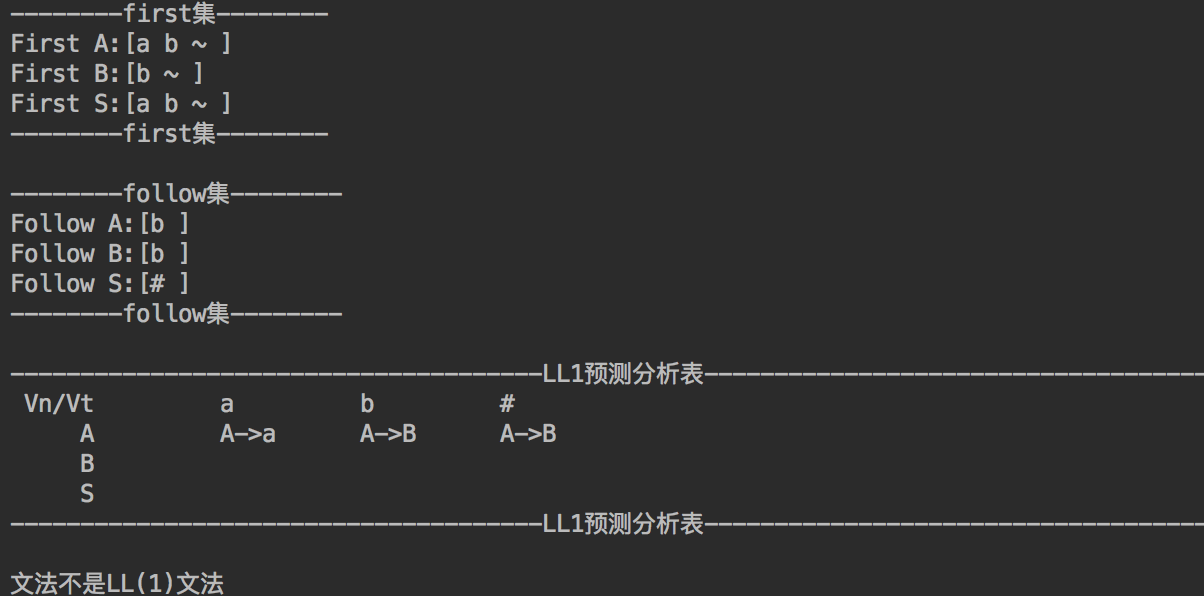


测试案例3:

文法： S->Ab A->a A->B A->~ B->b B->~ $

语句：abc#

运行结果：



在这个案例中，对于非终结符A，A->~也是一条产生式，而First(A)与Follow(A)的交集不为空，易看出这个文法不是LL1的。

附录1:（自顶向下的语法分析器源代码）

1. **package** LL1;
3. **import** java.util.ArrayList;
4. **import** java.util.HashMap;
5. **import** java.util.HashSet;
6. **import** java.util.Scanner;
7. **import** java.util.Stack;
9. //E->TG
10. //G->+TG
11. //G->~
12. //T->FH
13. //H->\*FH
14. //H->~
15. //F->(E)
16. //F->i
17. //$
18. //i\*i+i#
19. **public** **class** LL1 {
20. **public** **static** **void** main(String[] args) {
22. Scanner n = **new** Scanner(System.in);
23. ArrayList<String> grammer = **new** ArrayList<>();
24. MyLL1 myLL1 = **new** MyLL1();
25. //输入文法
26. System.out.println("请输入文法：（以$结尾，用～代替空串）");
27. **for**(**int** i=0;**true**;i++) {
29. String s1 = n.nextLine();
30. **if**(i == 0){
31. myLL1.setBegin(s1.charAt(0));
32. }
33. **if**(s1.equals("$")) {
34. **break**;
35. }
36. grammer.add(s1);
37. }
38. myLL1.setInputExperssion(grammer.toArray(**new** String[grammer.size()]));
39. //输入要判断的句子
40. System.out.println("请输入句子：(以#结尾)");
41. String sentence = n.nextLine();
42. myLL1.setStrInput(sentence);
44. myLL1.getVnVt();//求文法的终结符和非终结符
45. myLL1.Init();//
46. //如果预测分析表的填入过程中不产生冲突，那么该文法是LL1文法，否则该文法不是LL1文法
47. **if**(myLL1.createTable()==0) {
48. myLL1.output();
49. System.out.println("文法不是LL(1)文法");
50. **return**;
51. }
52. myLL1.output();
53. myLL1.analyzeLL();
54. }
55. }
57. **class** MyLL1 {
58. **private** String[][] table;
59. **private** String[] inputExperssion = {};
60. **private** Stack<Character> analyzeStatck = **new** Stack<>();
61. **private** String strInput = "";
62. **private** String action = "";
63. **private** Character begin;
64. **private** **int** index = 0;
65. //单个符号first集
66. **private** HashMap<Character, HashSet<Character>> firstSet = **new** HashMap<>();
67. //符号串first集
68. **private** HashMap<String, HashSet<Character>> firstSetX = **new** HashMap<>();
69. **private** HashMap<Character, HashSet<Character>> followSet = **new** HashMap<>();
70. //非终结符
71. **private** HashSet<Character> VnSet = **new** HashSet<>();
72. //终结符
73. **private** HashSet<Character> VtSet = **new** HashSet<>();
74. //非终结符-产生式集合
75. **private** HashMap<Character, ArrayList<String>> expressionSet = **new** HashMap<>();
77. **public** **void** setInputExperssion(String[] inputExperssion) {
78. **this**.inputExperssion = inputExperssion;
79. }
81. **public** **void** setStrInput(String strInput) {
82. **this**.strInput = strInput;
83. }
85. **public** **void** setBegin(Character begin) {
86. **this**.begin = begin;
87. }
89. //求非终结符和终结符
90. **public** **void** getVnVt() {
91. **for** (String e : inputExperssion)
92. //如果是产生式左侧的，就是非终结符
93. VnSet.add(e.split("->")[0].charAt(0));
94. **for** (String e : inputExperssion)
95. **for** (**char** c : e.split("->")[1].toCharArray())
96. //如果不是非终结符，那么就是终结符
97. **if** (!VnSet.contains(c))
98. VtSet.add(c);
99. }
100. //初始化，获取生成式并构造FIRST集和FOLLOW集
101. **public** **void** Init() {
102. //获取生成式
103. **for** (String e : inputExperssion) {
104. String[] str = e.split("->");
105. **char** c = str[0].charAt(0);
106. ArrayList<String> list = expressionSet.containsKey(c) ? expressionSet.get(c) : **new** ArrayList<>();
107. list.add(str[1]);
108. expressionSet.put(c, list);
109. }
110. //构造非终结符的first集
111. **for** (**char** c : VnSet)
112. getFirst(c);
113. //构造开始符的follow集
114. getFollow(begin);
115. //构造非终结符的follow集
116. **for** (**char** c : VnSet)
117. getFollow(c);
118. }
119. //求first集
120. **private** **void** getFirst(**char** c) {
121. //如果处理过了该符号，直接返回
122. **if** (firstSet.containsKey(c))
123. **return**;
124. //暂存集合
125. HashSet<Character> set = **new** HashSet<>();
126. // 若c属于终结符;
127. **if** (VtSet.contains(c)) {
128. set.add(c);
129. firstSet.put(c, set);
130. **return**;
131. }
132. // 处理其每条产生式
133. **for** (String s : expressionSet.get(c)) {
134. //如果c为~，First(c)=～
135. **if** ("~".equals(c)) {
136. set.add('~');
137. } **else** {
138. //如果c->a...，则把a加入到First(c)集中；如果c->Y....,则先求first(Y)，再把first(Y)的所有非空内容添加到First(c)中
139. **for** (**char** cur : s.toCharArray()) {
140. **if** (!firstSet.containsKey(cur))
141. getFirst(cur);
142. HashSet<Character> curFirst = firstSet.get(cur);
143. set.addAll(curFirst);
144. //找到产生式中第1个first集不含空的非终结符，
145. //将该非终结符以及之前所有非终结符的first中的所有元素（不包括空）添加到first集中
146. **if** (!curFirst.contains('~'))
147. **break**;
148. }
149. }
150. }
151. firstSet.put(c, set);
152. }
153. //求产生式A->a的first(a)
154. **private** **void** getFirst(String s) {
155. **if** (firstSetX.containsKey(s))
156. **return**;
157. HashSet<Character> set = **new** HashSet<>();
158. // 从左往右扫描该式
159. **int** i = 0;
160. **while** (i < s.length()) {
161. **char** cur = s.charAt(i);
162. **if** (!firstSet.containsKey(cur))
163. getFirst(cur);
164. HashSet<Character> rightSet = firstSet.get(cur);
165. // 将其非空first集加入左部
166. set.addAll(rightSet);
167. // 若包含空串 处理下一个符号
168. **if** (rightSet.contains('~'))
169. i++;
170. **else**
171. **break**;
172. // 若到了尾部 即所有符号的first集都包含空串 把空串加入fisrt集
173. **if** (i == s.length()) {
174. set.add('~');
175. }
176. }
177. firstSetX.put(s, set);
178. }
180. **private** **void** getFollow(**char** c) {
181. ArrayList<String> list = expressionSet.get(c);
182. HashSet<Character> leftFollowSet = followSet.containsKey(c) ? followSet.get(c) : **new** HashSet<>();
183. //如果是开始符 添加 #
184. **if** (c == begin)
185. leftFollowSet.add('#');
186. //查找输入的所有产生式，添加c的后跟终结符
187. **for** (**char** ch : VnSet)
188. **for** (String s : expressionSet.get(ch))
189. **for** (**int** i = 0; i < s.length(); i++)
190. **if** (c == s.charAt(i) && i + 1 < s.length() && VtSet.contains(s.charAt(i + 1)))
191. leftFollowSet.add(s.charAt(i + 1));
192. followSet.put(c, leftFollowSet);
193. //从右往左扫描处理c的每一条产生式
194. **for** (String s : list) {
195. **int** i = s.length() - 1;
196. **while** (i >= 0) {
197. **char** cur = s.charAt(i);
198. //只处理非终结符  I->i(E)SL
199. **if** (VnSet.contains(cur)) {
200. // 都按 A->αBβ 形式处理
201. //1.若β存在，把β的非空first集  加入followB
202. //2.若β不存在或者β存在且first(β)包含空串，将followA加入followB
203. String right = s.substring(i + 1);
204. HashSet<Character> rightFirstSet;
205. **if**(!followSet.containsKey(cur))
206. getFollow(cur);
207. HashSet<Character> curFollowSet = followSet.get(cur);
208. //如果β不存在，将followA加入followB
209. **if** (right.length() == 0) {
210. curFollowSet.addAll(leftFollowSet);
211. } **else** {//如果β存在，先找出first(β),将非空的加入followB
212. **if** (1 == right.length()) {
213. **if** (!firstSet.containsKey(right.charAt(0)))
214. getFirst(right.charAt(0));
215. rightFirstSet = firstSet.get(right.charAt(0));
216. } **else** {
217. **if** (!firstSetX.containsKey(right))
218. getFirst(right);
219. rightFirstSet = firstSetX.get(right);
220. }
221. **for** (**char** var : rightFirstSet)
222. **if** (var != '~')
223. curFollowSet.add(var);
224. // 若first(β)包含空串,将followA加入followB
225. **if** (rightFirstSet.contains('~'))
226. curFollowSet.addAll(leftFollowSet);
227. }
228. followSet.put(cur, curFollowSet);
229. }
230. i--;
231. }
232. }
233. }
235. //    对文法G的每个产生式A->x执行第2、3步
236. //    对每个终结符a属于first(x),把A->x加入M[A,a]中
237. //    若～属于first(x),则对任何b属于follow(A)，把A->~加入M[A,b]中
238. //    把所有无定义的项标上出错标志
239. **public** **int** createTable() {
240. Object[] VtArray = VtSet.toArray();
241. Object[] VnArray = VnSet.toArray();
242. // 预测分析表初始化
243. table = **new** String[VnArray.length + 1][VtArray.length + 1];
244. table[0][0] = "Vn/Vt";
245. //初始化首行首列
246. **for** (**int** i = 0; i < VtArray.length; i++)
247. table[0][i + 1] = (VtArray[i].toString().charAt(0) == '~') ? "#" : VtArray[i].toString();
248. **for** (**int** i = 0; i < VnArray.length; i++)
249. table[i + 1][0] = VnArray[i] + "";
250. //全部置error
251. **for** (**int** i = 0; i < VnArray.length; i++)
252. **for** (**int** j = 0; j < VtArray.length; j++)
253. table[i + 1][j + 1] = " ";
254. //对于每一个非终结符，判断其每一个产生式
255. **for** (**char** A : VnSet) {
256. **for** (String s : expressionSet.get(A)) {
257. **if** (!firstSetX.containsKey(s))
258. getFirst(s);
259. HashSet<Character> set = firstSetX.get(s);
260. **for** (**char** a : set)
261. //如果无冲突
262. **if**(find(A, a).equals(" ")||find(A, '#').equals(s))
263. //填表
264. fillIn(A, a, s);
265. **else** **return** 0;
266. //如果first集中含有空串
267. **if** (set.contains('~')) {
268. HashSet<Character> setFollow = followSet.get(A);
269. //如果follow集中含有"#"
270. **if** (setFollow.contains('#'))
271. //如果无冲突
272. **if**(find(A, '#').equals(" ")||find(A, '#').equals(s))
273. //填表
274. fillIn(A, '#', s);
275. **else** **return** 0;
276. //如果无冲突，将s填入任何属于follow集的终结符项中
277. **for** (**char** b : setFollow)
278. **if**(find(A, b).equals(" ")) {
279. fillIn(A, b, s);
280. }**else** **if**(find(A, '#').equals(s)){}
281. **else**{ **return** 0;}
282. }
283. }
284. }
285. **return** 1;
286. }
288. **public** **void** analyzeLL() {
289. System.out.println("------------------------LL分析过程------------------------");
290. System.out.println("            符号栈             输入串     下一步动作使用产生式");
291. //#入栈
292. analyzeStatck.push('#');
293. analyzeStatck.push(begin);
294. //取符号栈顶
295. **char** item = analyzeStatck.peek();
296. **while** (item != '#') {
297. **char** a = strInput.charAt(index);
298. **if** (item == a) {
299. action = "";
300. displayLL();
301. analyzeStatck.pop();
302. index++;
303. } **else** **if** (VtSet.contains(item)) {
304. action = "";
305. displayLL();
306. System.out.println("不符合文法 at '" + strInput.charAt(index) + "' in " + (index+1));
307. **return**;
308. } **else** **if** (find(item, a).equals(" ")) {
309. action = "";
310. displayLL();
311. System.out.println("不符合文法 at '" + strInput.charAt(index) + "' in " + (index+1));
312. **return**;
313. } **else** **if** (find(item, a).equals("~")) {
314. action = item + "->~";
315. displayLL();
316. analyzeStatck.pop();
317. } **else** {
318. String str = find(item, a);
319. **if** (str != "") {
320. action = item + "->" + str;
321. displayLL();
322. analyzeStatck.pop();
323. **int** len = str.length();
324. **for** (**int** i = len - 1; i >= 0; i--)
325. analyzeStatck.push(str.charAt(i));
326. } **else** {
327. action = "";
328. displayLL();
329. System.out.println("不符合文法 at " + strInput.charAt(index) + "' in " + (index+1));
330. **return**;
331. }
332. }
333. item = analyzeStatck.peek();
335. }
336. action="";
337. displayLL();
338. System.out.println("LL(1)分析成功");
339. System.out.println("------------------------LL分析过程------------------------\n");
340. }
342. **private** String find(**char** X, **char** a) {
343. //如果a为空串，
344. **if** (a == '~') a = '#';
345. **for** (**int** i = 1; i < VnSet.size() + 1; i++) {
346. **if** (table[i][0].charAt(0) == X)
347. **for** (**int** j = 1; j < VtSet.size() + 1; j++) {
348. **if** (table[0][j].charAt(0) == a)
349. **return** table[i][j];
350. }
351. }
352. **return** "";
353. }
355. **private** **void** fillIn(**char** X, **char** a, String s) {
356. **if** (a == '~') a = '#';
357. **for** (**int** i = 0; i < VnSet.size() + 1; i++) {
358. **if** (table[i][0].charAt(0) == X)
359. **for** (**int** j = 0; j < VtSet.size() + 1; j++) {
360. **if** (table[0][j].charAt(0) == a) {
361. table[i][j] = s;
362. **return**;
363. }
364. }
365. }
366. }
368. **private** **void** displayLL() {
369. // 输出 LL1
370. Stack<Character> s = analyzeStatck;
371. System.out.printf("%25s", s);
372. System.out.printf("%15s", strInput.substring(index));
373. System.out.printf("%13s", action);
374. System.out.println();
375. }
377. **public** **void** output() {
378. System.out.println("--------first集--------");
379. **for** (Character c : VnSet) {
380. HashSet<Character> set = firstSet.get(c);
381. System.out.print("First " + c + ":[");
382. **for** (Character var : set)
383. System.out.print(var+" ");
384. System.out.print("]");
385. System.out.println();
386. }
387. System.out.println("--------first集--------\n");
388. System.out.println("--------follow集--------");
390. **for** (Character c : VnSet) {
391. HashSet<Character> set = followSet.get(c);
392. System.out.print("Follow " + c + ":[");
393. **for** (Character var : set)
394. System.out.print(var+" ");
395. System.out.print("]");
396. System.out.println();
397. }
398. System.out.println("--------follow集--------\n");
400. System.out.println("--------------------------------------LL1预测分析表--------------------------------------");
402. **for** (**int** i = 0; i < VnSet.size() + 1; i++) {
403. **for** (**int** j = 0; j < VtSet.size() + 1; j++) {
404. **if**(i==0||j==0){
405. System.out.printf("%10s", table[i][j] + "    ");
406. }**else** {
407. **if**(table[i][j].equals(" ")){
408. System.out.printf("%10s", table[i][j] + " ");
409. }**else** {
410. System.out.printf("%10s", table[i][0]+"->"+table[i][j] + " ");
411. }
413. }
415. }
416. System.out.println();
417. }
418. System.out.println("--------------------------------------LL1预测分析表--------------------------------------\n");
419. }
420. }