实验二 LL(1)分析法

2.1 实验目的

通过完成预测分析法的语法分析程序,了解预测分析法和递归子程序法的区别和联系。使学生了解语法分析的功能,掌握语法分析程序设计的原理和构造方法,训练学生掌握开发应用程序的基本方法。有利于提高学生的专业素质,为培养适应社会多方面需要的能力。

2.2 实验内容

1) 根据某一文法编制调试LL(1)分析程序,以便对任意输入的符号串进行分析。

2) 构造预测分析表,并利用分析表和一个栈来实现对上述程序设计语言的分析程序。

3) 分析法的功能是利用 LL(1)控制程序根据显示栈栈顶内容、向前看符号以及 LL(1)分析表,对输入符号串自上而下的分析过程。

2.3 实验环境

硬件：

Dell G3 3579；

软件：

OS：Ubuntu 16.04.06；

IDE：IntelliJ IDEA Ultimate Edition（2019.1.3）；

编程语言：Scala、Java。

2.4 LL(1)文法分析实验设计思想及算法

2.4.1 实验基本思路

首先给出 LL(1)文法的定义[2]：

一个文法 G 被称为 LL(1)文法，如果它满足以下条件：

（1）文法不含左递归。

（2）对于文法中每一个非终结符 A 的各个产生式的候选首字符集两两不相交。即，若

$$A \rightarrow \alpha_1|\alpha_2|...|\alpha_n$$

则　$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \Phi$　　$(i \neq j)$

（3）对文法中的每个非终结符 A，若它存在某个候选首字符集包含ε，则

$$FIRST(\alpha_i) \cap FOLLOW(A) \quad (i = 1, 2, , ..., n)$$

这里，LL(1)的第一个 L 表示从左向右扫描输入串，第二个 L 表示最左推导，1 表示分析时每一步只需向前查看一个符号。

对于一个 LL(1)文法，可以对其输入穿进行有效的无回溯的自上而下分析。假设要用非终结符 A 进行匹配，面临的输入符号为 a，A 的所有产生式为：

$$A \rightarrow \alpha_1|\alpha_2|...|\alpha_n$$

（1） 若 $a \in FIRST(\alpha_i)$，则指派α_i 去执行匹配任务。

（2） 若 a 不属于任何一个候选首符集，则：

①若ε属于某个 $FIRST(\alpha_i)$且 $a \in FOLLOW(A)$，则让 A 与ε自动匹配；

②否则，a 的出现是一种错误。

综述，本实验对教材上的描述的几个算法进行了实现，成功达成了 LL(1)文法分析，并进行了简单的测试。

2.4.2 算法流程

LL(1)文法分析的架构如图 8 所示。

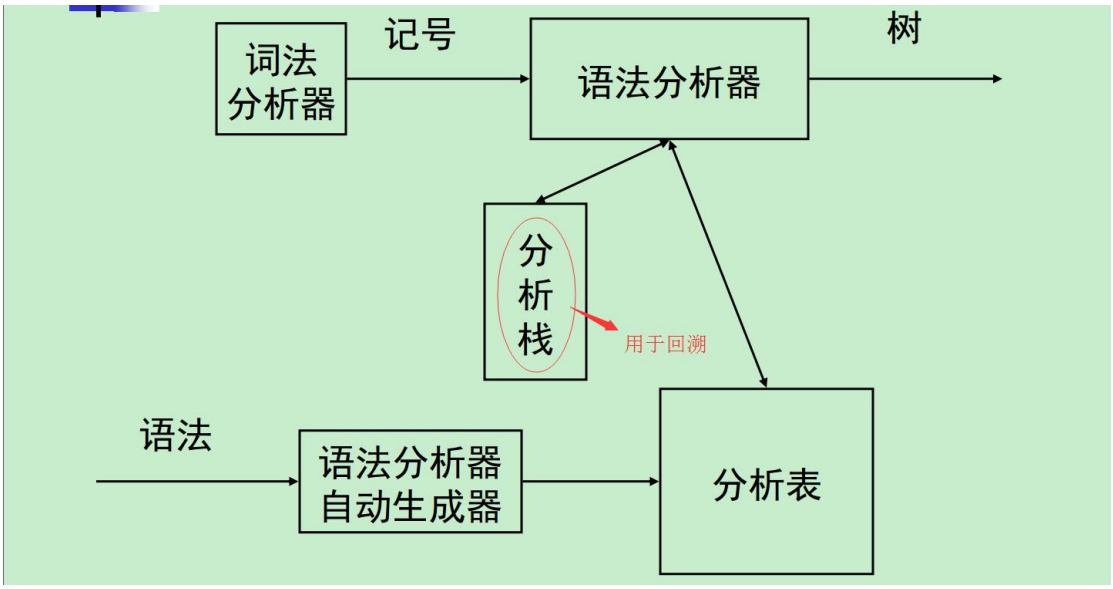


Figure 8 LL 文法分析架构

2.4.3 主要函数及其功能

函数名	参数类型	返回类型	功能	备注
eliminateLeftRecursion	无	ArrayBuffer[(String, String, String)]	消除形式上的左递归	Scala
FIRST	ArrayBuffer[(String, String)]	Map[String, String]	求解指定文法 FIRST 集	迭代求解，因此代码较长，Scala
dfsFOLLOW	String	String	DFS 寻找各个非终结符的 FOLLOW 集元素	Scala
FOLLOW	ArrayBuffer[(String, String)]	Map[String, String]	根据 dfsFOLLOW 函数，获取各个非终结符的 FOLLOW 集元素	Scala
initiateMatrix	无	Array[Array[String]]	初始化分析表	Scala
createMatrix	无	Array[Array[String]]	构造分析表	Scala
analyse	String	Boolean	对指定的字符串 LL(1)分析	Scala
GUI1	无	无	实现图形化界面展示	开始界面，Java
GUI2	无	无	实现图形化界面展示	分析界面，Java

Table 2 LL(1)文法分析实验代码主要函数概览

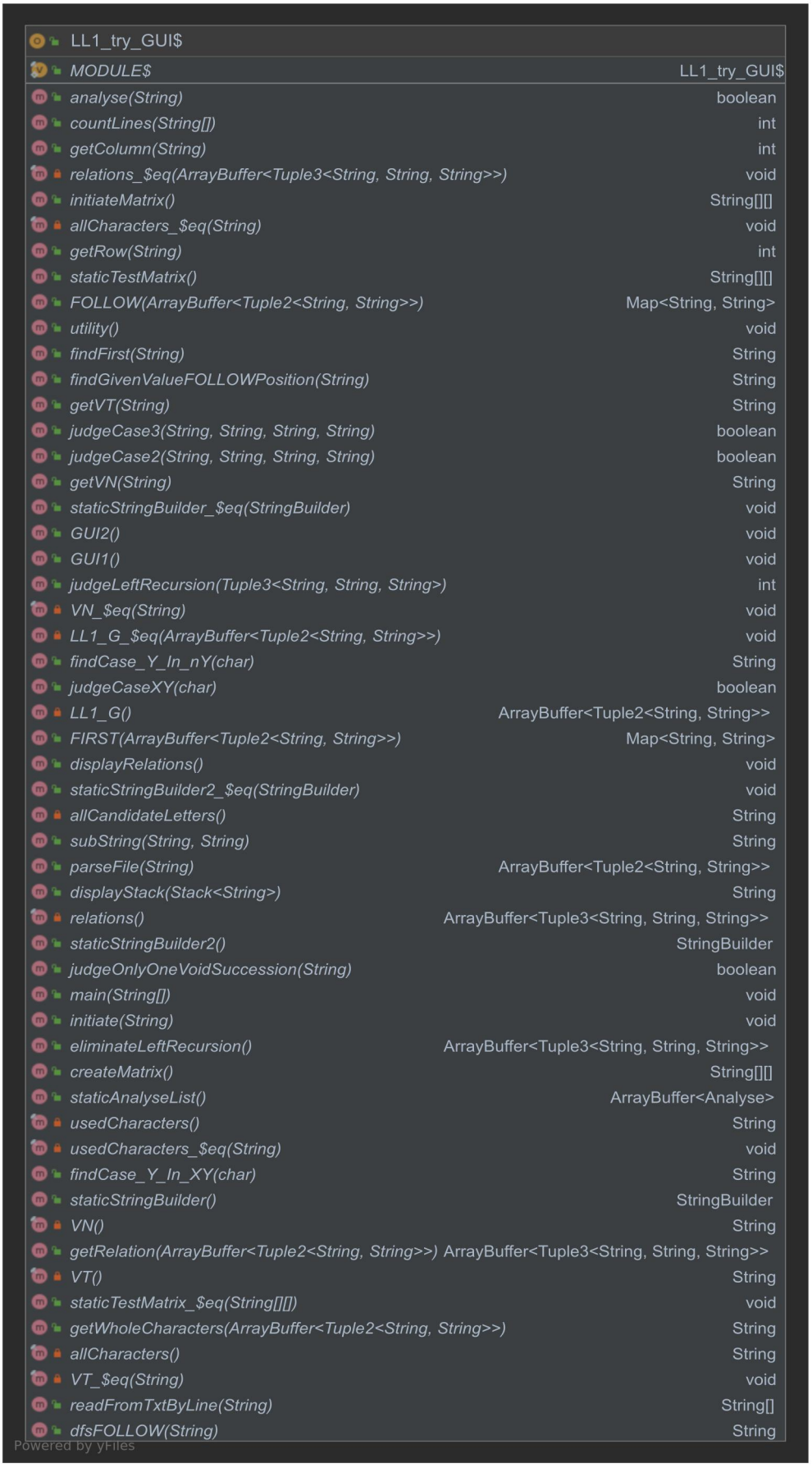


Figure 9 LL(1)文法分析实验代码完整函数

2.4.4 核心代码形式化描述及其实现

实现 LL(1)文法分析的一种有效方法是使用一张分析表和一个栈进行联合控制，而分析表的构造需要用到給定文法的 FIRST 集与 FOLLOW 集[2]。下面依次介绍预测分析程序的总控程序、FIRST 集、FOLLOW 集与分析表的构造流程。

预测分析程序的总控程序形式化描述[1]：

```
let a be the first symbol of w;
let X be the top stack symbol;
while ( X ≠ $ ) { /* stack is not empty */
        if ( X = a ) pop the stack and let a be the next symbol of w;
        else if ( X is a terminal ) error();
        else if ( M[X, a] is an error entry ) error();
        else if ( M[X, a] = X → Y1 Y2 ··· Yk ) {
                output the production X → Y1 Y2 ··· Yk;
                pop the stack;
                push Yk, Yk−1, ... , Y1 onto the stack, with Y1 on top;
        }
        let X be the top stack symbol;
}
```

Figure 10 LL(1)预测分析总控形式化描述

FIRST 集构造流程[1]：

To compute FIRST(X) for all grammar symbols X, apply the following rules until no more terminals or ϵ can be added to any FIRST set.

1. If X is a terminal, then FIRST(X) = $\{X\}$.
2. If X is a nonterminal and $X \to Y_1 Y_2 \cdots Y_k$ is a production for some $k \geq 1$, then place a in FIRST(X) if for some i, a is in FIRST(Y_i), and ϵ is in all of FIRST(Y_1), ... , FIRST(Y_{i-1}); that is, $Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$. If ϵ is in FIRST(Y_j) for all $j = 1, 2, \ldots, k$, then add ϵ to FIRST(X). For example, everything in FIRST(Y_1) is surely in FIRST(X). If Y_1 does not derive ϵ, then we add nothing more to FIRST(X), but if $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, then we add FIRST($Y_2$), and so on.
3. If $X \to \epsilon$ is a production, then add ϵ to FIRST(X).

Figure 11 FIRST 集构造流程

FOLLOW 集构造流程[1]：

To compute FOLLOW (A) for all nonterminals A , apply the following rules until nothing can be added to any FOLLOW set.

1. Place $ in FOLLOW (S), where S is the start symbol, and $ is the input right endmarker.
2. If there is a production A → αBβ , then everything in FIRST (β) except ε is in FOLLOW (B).
3. If there is a production A → αB, or a production A → αBβ, where FIRST (β) contains ε, then everything in FOLLOW (A) is in FOLLOW (B).

Scala 实现 FIRST 函数：

```
def FIRST( string: ArrayBuffer[ (String, String) ] ): Map[ String, String ] = {
        val FIRST_Group = Map[ String, String ]()
        val wholeCharacters = allCharacters
        val localVT = VT
        val localVN = VN
        for( character <- wholeCharacters ) {
            // case 1
            if( localVT.contains(character) ) {
                //if there exist the original key that equals the current one
                if( FIRST_Group.contains(character.toString) == true ) {
                    val tmp = character.toString + FIRST_Group(character.toString)
                    FIRST_Group(character.toString) = tmp.distinct
                }
                //otherwise
                else {
                    FIRST_Group(character.toString) = character.toString
                }
            }
            // case 2
            if( localVN.contains(character.toString) == true ) {
                // case 2.1
                val value = findFirst(character.toString)
                if ( value.length != 0 ) {
                    if ( FIRST_Group.contains(character.toString) == true ) {
                        for( ch <- value ) {
                            val tmp = ch + FIRST_Group(character.toString)
                            FIRST_Group(character.toString) = tmp.distinct
                        }
                    }
```

```
else {
                                FIRST_Group(character.toString) = value.toString
                        }
                    }
                    // case 2.2
                    if( judgeOnlyOneVoidSuccession(character.toString) == true ) {
                        if ( FIRST_Group.contains(character.toString) == true ) {
                            val tmp = "ε" + FIRST_Group(character.toString)
                            FIRST_Group(character.toString) = tmp.distinct
                        }
                        else {
                            FIRST_Group(character.toString) = "ε"
                        }
                    }
                }
                for( character <- wholeCharacters ) {
                    // case 3.1
                    if( judgeCaseXY(character) == true ) {
                        val tmpReply = findCase_Y_In_XY(character)
                        for( eachTmpReply <- tmpReply ) {
                            if( FIRST_Group.contains(eachTmpReply.toString) == true ) {
                                for (ex <- FIRST_Group(eachTmpReply.toString)) {
                                    if (ex != 'ε') {
                                        if (FIRST_Group.contains(character.toString) == true) {
                                            val tmp = ex.toString + FIRST_Group(character.toString)
                                            FIRST_Group(character.toString) = tmp.distinct
                                        }
                                        else {
                                            FIRST_Group(character.toString) = ex.toString
                                        }
                                    }
                                }
                            }
                        }
                    }
                    // case 3.2
                    if( findCase_Y_In_nY(character).length > 0 ) {
                        var flag = true
                        val tmpReply = findCase_Y_In_nY(character)

                        for( ex <- tmpReply ) {
                            if( localVN.contains(ex.toString) && FIRST_Group.contains(ex.toString) == true )
{
                                if( FIRST_Group(ex.toString).contains("ε") == false ) {
                                    flag = false
                                }
                            }
                            else flag = false
```

```
if( flag == true ) {
                                    if (FIRST_Group.contains(character.toString) == true) {
                                        val tmp = FIRST_Group(ex.toString).replace( "ε", "" ) +
FIRST_Group(character.toString)
                                        FIRST_Group(character.toString) = tmp.distinct
                                    }
                                    else {
                                        FIRST_Group(character.toString)                                    =
FIRST_Group(ex.toString).replace( "ε", "" )
                                    }

                                }
                            }
                        }
                        // case 3.3
                        if( findCase_Y_In_nY(character).length > 0 ) {
                            var flag = true
                            val tmpReply = findCase_Y_In_nY(character)
                            for( ex <- tmpReply ) {
                                if( localVN.contains(ex.toString) && FIRST_Group.contains(ex.toString) == true )
{
                                    if( FIRST_Group(ex.toString).contains("ε") == false ) {
                                        flag = false
                                    }
                                }
                                else {
                                    flag = false
                                }
                                if( flag == true ) {

                                    if (FIRST_Group.contains(character.toString) == true) {
                                        val tmp = "ε" + FIRST_Group(character.toString)
                                        FIRST_Group(character.toString) = tmp.distinct
                                    }
                                    else {
                                        FIRST_Group(character.toString) = "ε"
                                    }
                                }
                            }
                        }
                    }
                }
                FIRST_Group
            }
```

Scala 实现 FOLLOW、dfsFOLLOW 与 analyse 函数：

```
def FOLLOW( string: ArrayBuffer[ (String, String) ] ): Map[ String, String ] = {
        val localVN = VN
        val FOLLOW_Group = Map[ String, String ]()
        for( ch <- localVN ) {
            FOLLOW_Group(ch.toString) = dfsFOLLOW(ch.toString)
        }
        FOLLOW_Group
    }
```

```
def dfsFOLLOW( ch: String ): String = {
        val FOLLOWPositions = Map[ String, String ]()
        val FOLLOW_Group = Map[ String, String ]()
        val localLL1_G = LL1_G
        val FIRST_Group = FIRST(localLL1_G)
        val localVN = VN
        for( ch <- localVN ) {
            FOLLOWPositions(ch.toString) = findGivenValueFOLLOWPosition(ch.toString)
            FOLLOW_Group(ch.toString) = "#"
        }
        var result = ""

        if( FOLLOWPositions(ch).length == 4 ) {
            if( FOLLOWPositions(ch)(1).toString == "T" ) {
                result += FIRST_Group( FOLLOWPositions(ch)(0).toString )
                FOLLOW_Group(ch) += result.distinct
            }
            else if( FOLLOWPositions(ch)(3).toString == "T" ) {
                result += FIRST_Group( FOLLOWPositions(ch)(2).toString )
                FOLLOW_Group(ch) += result.distinct
            }
            if( FOLLOWPositions(ch)(1).toString == "W" ) {
                result += dfsFOLLOW( FOLLOWPositions(ch)(0).toString )
                FOLLOW_Group(ch) = result.distinct
            }
            else if( FOLLOWPositions(ch)(3).toString == "W" ) {
                result += dfsFOLLOW( FOLLOWPositions(ch)(2).toString )
                FOLLOW_Group(ch) = result.distinct
            }
        }
```

```
if( FOLLOWPositions(ch).length == 2 ) {
                if( FOLLOWPositions(ch)(1).toString == "T" ) {
                        result += FIRST_Group( FOLLOWPositions(ch)(0).toString )
                        FOLLOW_Group(ch) = result.distinct
                }
                else if( FOLLOWPositions(ch)(1).toString == "W" ) {
                        result += dfsFOLLOW( FOLLOWPositions(ch)(0).toString )
                        FOLLOW_Group(ch) = result.distinct
                }
            }
            FOLLOW_Group(ch).replace("ε", "")
      }
```

```
def analyse( expression: String ): Boolean = {
            val stack = new mutable.Stack[String]()
            var localExpression = expression
            val table = createMatrix()
            val localVT = VT
            val localVN = VN
            val localRelations = relations
            stack.push("#")
            stack.push( localRelations(0)._1 )
            var cnt = 0
            staticAnalyseList.append(new Analyse("步骤","分析栈","剩余字符串","所用产生式","动作"));
            staticAnalyseList.append(new                                                    Analyse(cnt.toString,
displayStack(stack).reverse.toString,localExpression.toString,"","initiate"));
            while( stack.isEmpty == false ) {
                  val stackTop = stack.top
                  stack.pop()
                  // 栈顶符号属于  非终结符
                  if( localVN.contains(stackTop) == true ) {
                        // 栈顶符号与表达式左端首字符  存在  关系
                        if( table( getRow(stackTop) )( getColumn( localExpression(0).toString ) ) != null ) {
                              val                                       lastHalf                                =
table( getRow(stackTop) )( getColumn( localExpression(0).toString ) ).split( "->", 2 ).last
                              val length = lastHalf.length
                              for( i <- 0 to (length - 1) ) {
                                    if( lastHalf != "ε" ) {
                                          stack.push(lastHalf(length - 1 - i).toString)
                                    }
                              }
                              cnt += 1
```

```
if( lastHalf != "ε" ) {
                    staticAnalyseList.append(new Analyse(cnt.toString, displayStack(stack).reverse.toString, localExpression.toString, table(getRow(stackTop))(getColumn(localExpression(0).toString)), "POP, PUSH(" + lastHalf.reverse + ")"));
                }
                else {
                    staticAnalyseList.append(new Analyse(cnt.toString, displayStack(stack).reverse.toString, localExpression.toString, table(getRow(stackTop))(getColumn(localExpression(0).toString)), "POP"));
                }
            }
            // 栈顶符号与表达式左端首字符  不存在  关系
            else {
                // 栈顶符号 等于 表达式左端首字符
                if( stackTop == "#" && localExpression(0).toString == "#" ) {
                    println("11111")
                    return true
                }
                // 栈顶符号 不等于 表达式左端首字符
                else {
                    println("1 - error")
                    staticAnalyseList.append(new Analyse(cnt.toString, displayStack(stack).reverse.toString,localExpression.toString,"",""));
                    return false
                }
            }
        }
        // 栈顶符号属于  终结符
        if( localVT.contains(stackTop) == true ) {
            // 栈顶符号 等于 表达式左端首字符
            if( stackTop == localExpression(0).toString ) {
                if( stackTop == localExpression(0).toString ) {
                    //stack.pop()
                    localExpression = localExpression.drop(1)
                    cnt += 1
                    staticAnalyseList.append(new Analyse(cnt.toString, displayStack(stack).reverse.toString,localExpression.toString,"","GETNEXT(" + stackTop + ")"));
                }
                // 栈顶符号 不等于 表达式左端首字符
                else {
                    println("2 - error")
                    return false
                }
            }
        }
    }
    true
}
```

2.5 程序运行截图

Figure 12 开始界面

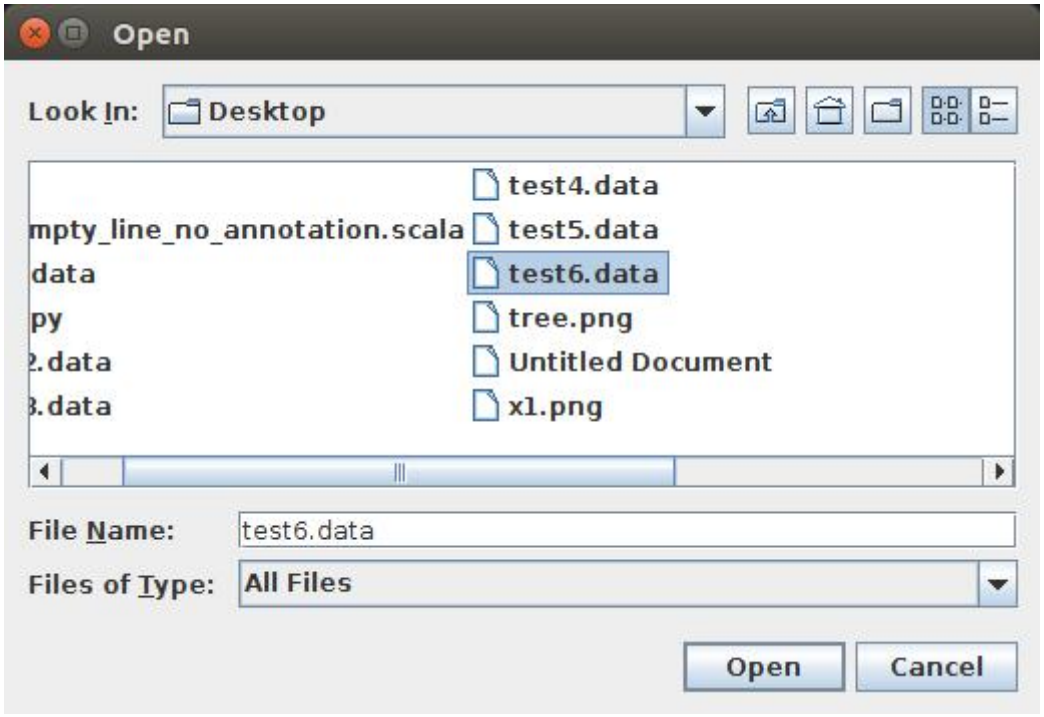


Figure 13 选择文件

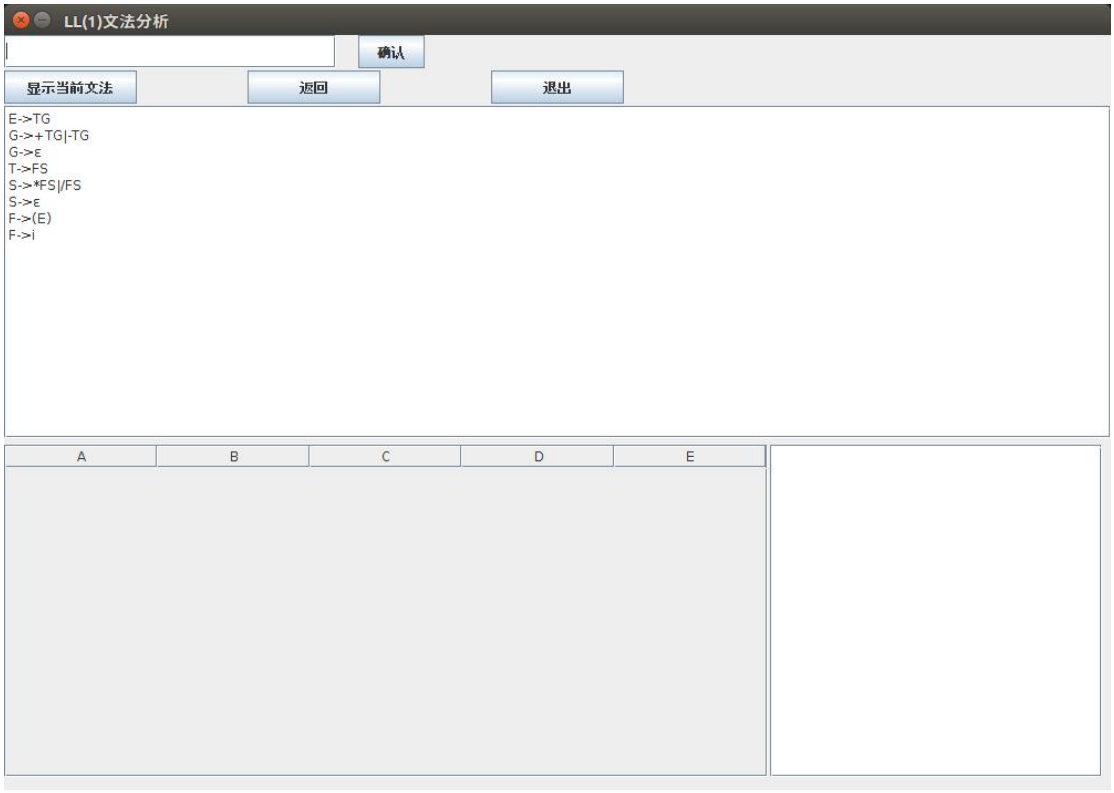


Figure 14 显示当前文法

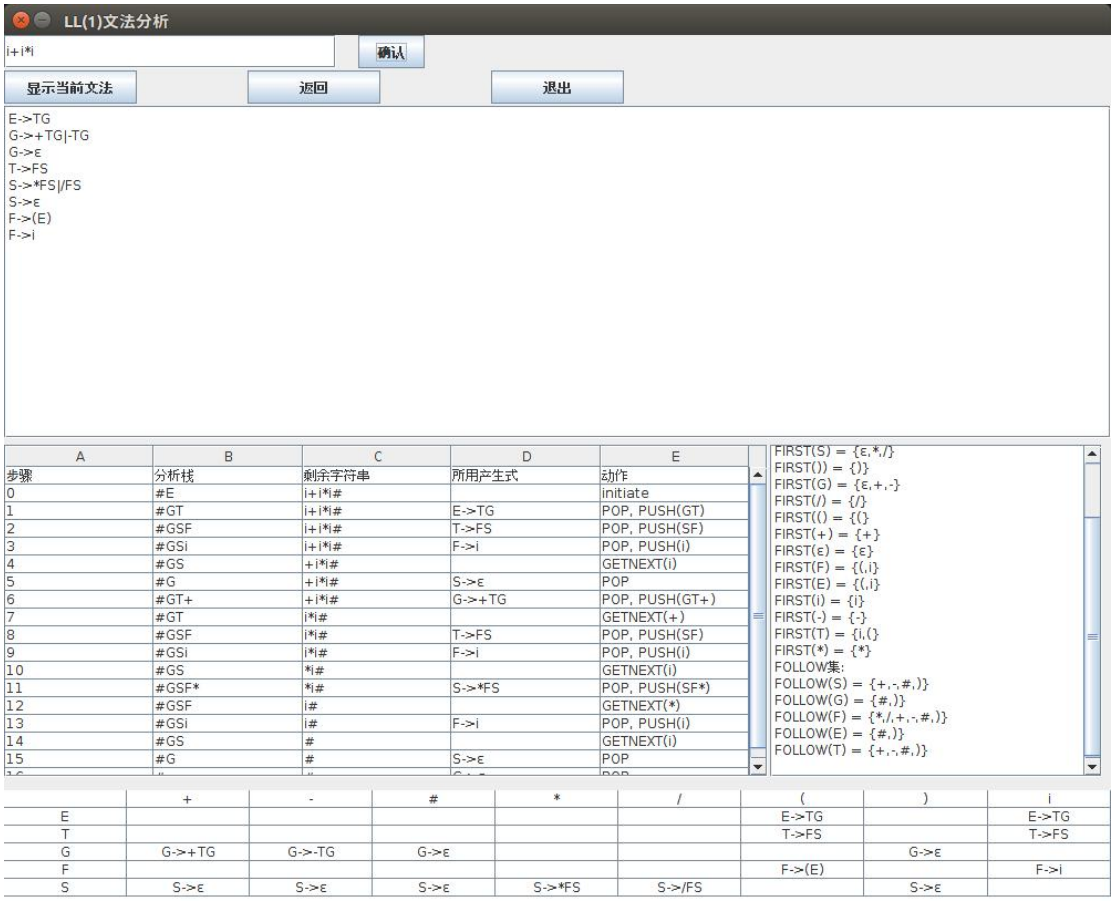
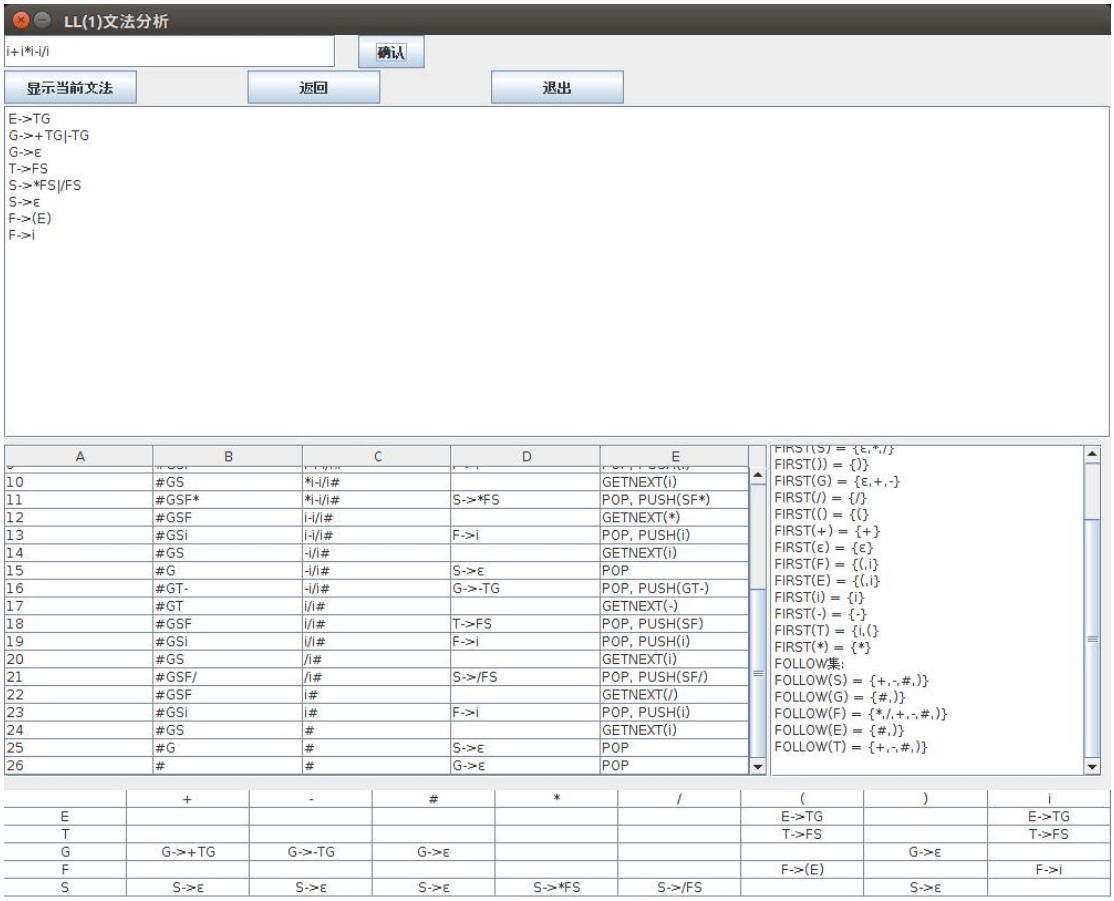


Figure 15 执行文法分析（输入表达式为“i+i*i”）



A	B	C	D	E
10	#GS	*i-i/i#		GETNEXT(i)
11	#GSF*	*i-i/i#	S->*FS	POP, PUSH(SF*)
12	#GSF	i-i/i#		GETNEXT(*)
13	#GSi	i-i/i#	F->i	POP, PUSH(i)
14	#GS	-i/i#		GETNEXT(i)
15	#G	-i/i#	S->ε	POP
16	#GT-	-i/i#	G->-TG	POP, PUSH(GT-)
17	#GT	i/i#		GETNEXT(-)
18	#GSF	i/i#	T->FS	POP, PUSH(SF)
19	#GSi	i/i#	F->i	POP, PUSH(i)
20	#GS	/i#		GETNEXT(i)
21	#GSF/	/i#	S->/FS	POP, PUSH(SF/)
22	#GSF	i#		GETNEXT(/)
23	#GSi	i#	F->i	POP, PUSH(i)
24	#GS	#		GETNEXT(i)
25	#G	#	S->ε	POP
26	#	#	G->ε	POP

	+	-	#	*	/	(	)	i
E						E->TG		E->TG
T						T->FS		T->FS
G	G->+TG	G->-TG	G->ε				G->ε	
F						F->(E)		F->i
S	S->ε	S->ε	S->ε	S->*FS	S->/FS		S->ε	

Figure 16 执行文法分析（输入表达式为“i+i*i-i/i”）