实验三 LR(1)分析法

3.1 实验目的

构造 LR(1)分析程序，利用它进行语法分析,判断给出的符号串是否为该文法识别的句子，了解 LR(K)分析方法是严格的从左向右扫描，和自底向上的语法分析方法。

3.2 实验内容

对下列文法，用 LR(1)分析法对任意输入的符号串进行分析:

(1)E-> E+T

(2)E->T

(3)T-> T*F

(4)T->F

(5)F-> (E)

(6)F-> i

3.3 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。

3.4 LR(1)文法分析实验设计思想及算法

3.4.1 实验基本思路[2]

LR 文法的每个项目的一般形式是[A→α·β, $a_1a_2...a_k$]，此处，A→α·β 是一个 LR(0)项目，每一个 a 都是终结符。这样的一个项目称为一个 LR(k)项目。项目中的 $a_1a_2...a_k$ 称为它的向前搜索符串（或展望串）。向前搜索符串仅对规约项目[A→α·β, $a_1a_2...a_k$]有意义。对于任何移进或待约项目[A→α·β, $a_1a_2...a_k$]，β≠ε，搜索符串 $a_1a_2...a_k$ 没有作用。规约项目[A→α·, $a_1a_2...a_k$]意味着：当它所属的状态呈现在栈顶且后续的 k 个输入符号为 $a_1a_2...a_k$ 时，才可以把栈顶上的α规约为 A。我们只对 k≤1 的情形感兴趣，因为，对多数程序语言的语法来说，向前搜索（展望）一个符号就多半可以确定“移进”或“规约”。

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 LR(1)文法分析，并进行了简单的测试。

3.4.2 算法流程

LR 文法分析的架构如图 17 所示。

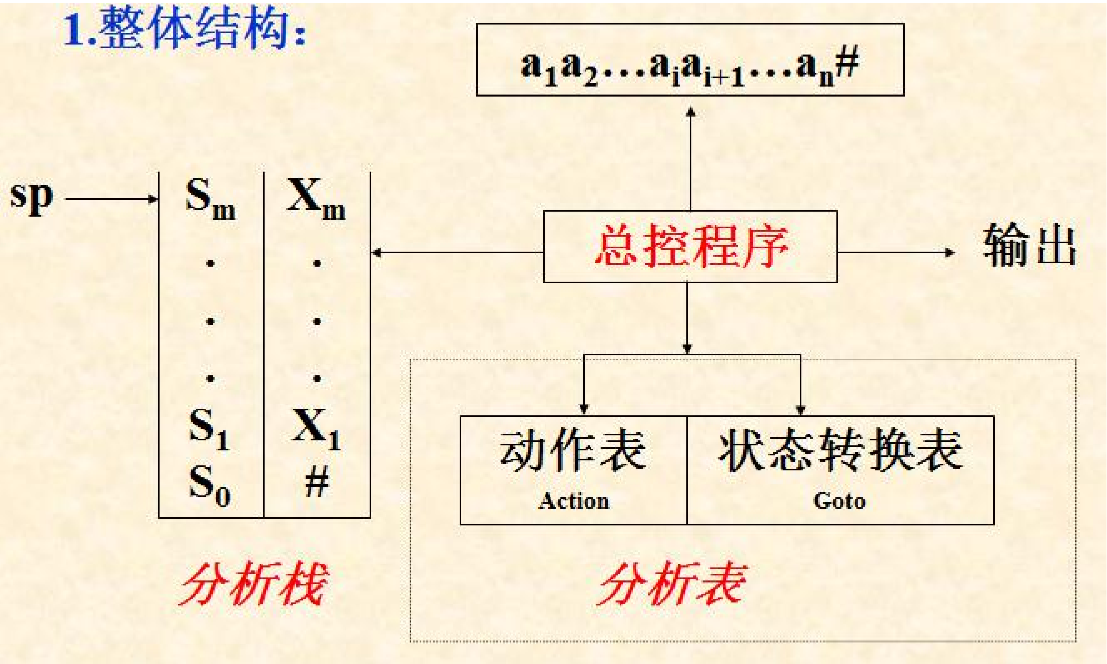


Figure 17 LR 文法分析架构

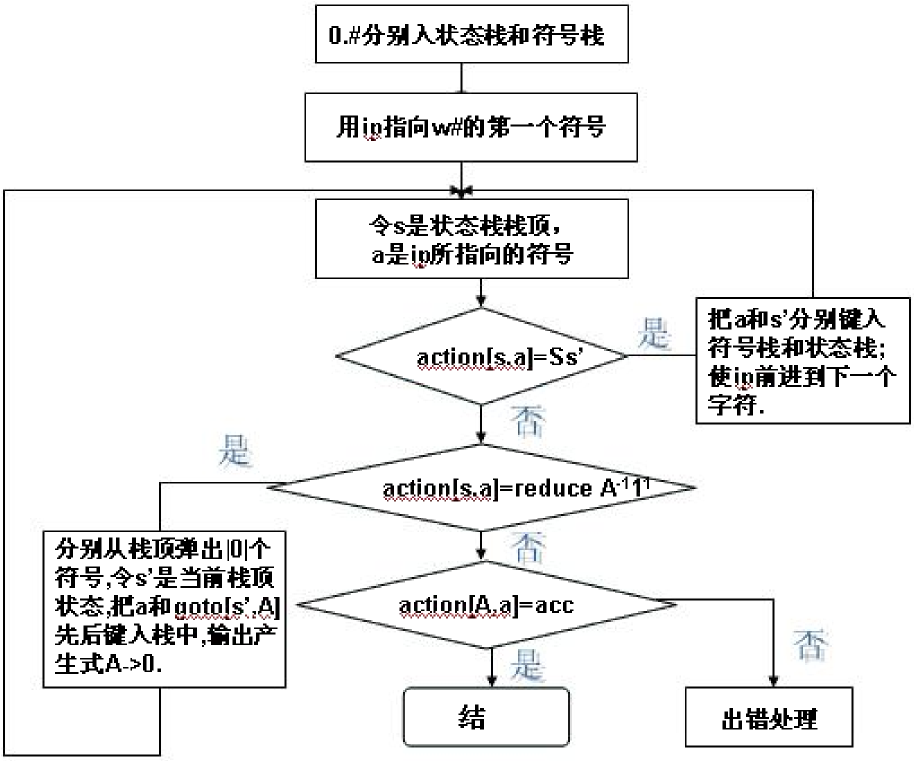


Figure 18 LR 文法分析流程图

3.4.3 主要函数及其功能

函数名	参数类型	返回类型	功能	备注
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
getClosure	ArrayBuffer[(String, String, String)]	ArrayBuffer[(String, String, String)]	求給定项目集的闭包	Scala
go	ArrayBuffer[(String, String, String), String	ArrayBuffer[(String, String, String)]	求給定项目对于特定字符的下一状态	Scala
createMatrix	无	Array[Array[String]]	构造 ACTION 与 GOTO 分析表	Scala
getItemGroup	无	无	建立初始化的项目集	Scala
analyse	String	Boolean	对指定的字符串进行 LR(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 3 LR(1)文法分析实验代码主要函数概览

3.4.4 核心代码形式化描述及其实现

求解闭包伪代码[1]：

```
SetOfItems CLOSURE (I) {
    repeat
        for ( each item [A → α·Bβ, a] in I )
            for ( each production B → γ in G’ )
                for ( each terminal b in FIRST(βa) )
                    add [ B → ·γ, b ] to set I ;
    until no more items are added to I ;
    return I ;
}
```

GOTO 函数伪代码[1]：

```
SetOfItems GOTO ( I, X ) {
    initialize J to be the empty set;
    for ( each item [A → α·Xβ, a] in I )
        add item [A → α·Xβ, a] to set J;
    return CLOSURE(J);
}
```

求解项目集族伪代码[1]：

```
void items ( G’ ) {
    initialize C to { CLOSURE( { [S’ → ·S, $] } ) };
    repeat
        for ( each set of items I in C )
            for ( each grammar symbol X )
                if ( GOTO ( I, X ) is not empty and not in C )
                    add GOTO ( I, X ) to C;
    until no new sets of items are added to C;
}
```

```
def getClosure( items: ArrayBuffer[ (String, String, String) ] ): ArrayBuffer[ (String, String, String) ] = {
    val result = new ArrayBuffer[ (String, String, String) ]()
    result.appendAll(items)
    val localFIRST = FIRST()
    var addFlag = true
    var cnt = 1
    while (addFlag == true ) {
        val originalResult = new ArrayBuffer[(String, String, String)]()
        originalResult.appendAll(result)
        for (ex <- result) {

            val pointPosition = ex._2.indexOf("·")
            //• 不在最右边
            if (pointPosition < ex._2.length - 1) {
                //B 在 • 的右边
                val B = ex._2(pointPosition + 1)
                val a = ex._3

                // case 1: β != Φ and a != # or
                // case 2: β != Φ and a = #
                if (pointPosition < ex._2.length - 2) {
                    val β = ex._2(pointPosition + 2)
                    // ξ
                    val rightExpressionsOfB = getRightExpressions(B.toString)
                    val FIRST_Of_βa = localFIRST(β.toString)
                    for (b <- FIRST_Of_βa) {
                        for (ksi <- rightExpressionsOfB) {
                            val tmp = ((B.toString, "·" + ksi, b.toString))

                            if (result.contains(tmp) == false) {
                                result += tmp
                            }
                        }
                    }
                }
                // case 3: β = Φ and a equals any character
                if (pointPosition == ex._2.length - 2) {
                    val rightExpressionsOfB = getRightExpressions(B.toString)
                    val FIRST_Of_βa = localFIRST(a.toString)
                    for (b <- FIRST_Of_βa) {
                        for (ksi <- rightExpressionsOfB) {
                            val tmp = ((B.toString, "·" + ksi, b.toString))
                            if (result.contains(tmp) == false) {
                                result += tmp
                            }
                        }
                    }
                }
}
                    }
                    if (result != originalResult) {
                          originalResult.remove(0, originalResult.length)
                          originalResult.appendAll(result)
                          cnt += 1
                    }
                    else {
                          addFlag = false
                          cnt += 1
                    }
              }
              result
        }
```

Scala 实现 go 函数：

```
def go( I: ArrayBuffer[ (String, String, String) ], X: String ): ArrayBuffer[ (String, String, String) ] = {
            //GO(I, X) = CLOSURE(J)
            //J = {任何形如[A->αX・ β, a]的项目|[A->α ・Xβ, a]∈I}
            val ans = new ArrayBuffer[ (String, String, String) ]()
            val items = new ArrayBuffer[ (String, String, String) ]()

            for( ex <- I ) {
                  val pointPosition = ex._2.indexOf("·")
                  //・  不在最右边
                  if (pointPosition < ex._2.length - 1) {
                        val A = ex._1
                        val possibleX = ex._2( pointPosition + 1)
                        //   αXβ
                        val noPointExpressionPart2 = ex._2.replace("·", "")
                        if( X == possibleX.toString ) {
                              //   αX·β
                              val newPart2 = noPointExpressionPart2.substring(0, pointPosition + 1) + "·" +
                                          noPointExpressionPart2.substring(pointPosition          +          1,
noPointExpressionPart2.length)
                              val a = ex._3
                              items += ( (A, newPart2, a) )
                        }
                  }
            }
            ans.appendAll( getClosure(items) )
            ans
      }
```

```
def getItemGroup(): Unit = {
        val Idx = ( relations(0)._1, "·" + relations(0)._2, "#" )
        val I0 = getClosure( ArrayBuffer(Idx) )
        val wholeCharacters = allCharacters
        var tot = 0
        itemGroup(I0) = tot
        var appendFlag = true
        while (appendFlag == true) {
            var originalAns = Map[ ArrayBuffer[ (String, String, String) ], Int ]()
            originalAns = itemGroup.clone()
            //为什么用 I 作为遍历变量不行？！
            for(item <- itemGroup.keys) {
                for (ch <- wholeCharacters) {
                    val newItem = go(item, ch.toString).sorted
                    if (newItem.isEmpty == false && itemGroup.contains(newItem) == false) {
                        tot += 1
                        itemGroup(newItem) = tot
                    }
                }
            }
            if( originalAns.equals(itemGroup) == true ) {
                appendFlag = false
            }
            else {
                originalAns.clear()
                originalAns = itemGroup.clone()
            }
        }
    }
```

Scala 实现 createMatrix 函数：

```
def createMatrix(): Array[ Array[String] ] = {
        val result = initiateMatrix()
        val localVT = VT
        val localVN = VN
        case class getColumn( ch: String ) {
            val matrix = initiateMatrix()
            var ans = -1
            for( j <- 0 to (columnLength - 1) ) {
                if( matrix(0)(j) == ch ) {
                    ans = j
                }
            }
        }
```

```
    for( ex <- itemGroup ) {
        for( tx <- ex._1 ) {
            val pointPosition = tx._2.indexOf("·")
            // •  不在最右边
            //若项目[A->α  •aβ] ∈ Ik，且 GO(Ik, a) = Ij，a 为终结符，则置 ACTION[k, a]
为“sj”
            if (pointPosition < tx._2.length - 1) {
                val a = tx._2( pointPosition + 1 )
                if( localVT.contains(a) == true && findItemOrder(ex._1, a.toString) != -1 ) {
                    val j = findItemOrder(ex._1, a.toString)
                    var tmpRow = -1
                    tmpRow = ex._2 + 1
                    result(tmpRow)( getColumn(a.toString).ans ) = "S" + j.toString
                }
            }
            if (pointPosition == tx._2.length - 1 ) {
                val a = tx._3
                var tmpRow = -1
                tmpRow = ex._2 + 1
                result(tmpRow)(getColumn(a).ans) = "r" + ( findRelationOrder( (tx._1,
                        tx._2.replace("·", "") ) ) )
            }
            if( tx._1 == relations(0)._1 && tx._2 == relations(0)._2 + "·" && tx._3 == "#" ) {
                var tmpRow = -1
                tmpRow = ex._2 + 1
                result(tmpRow)( getColumn("#").ans ) = "acc"
            }
        }
        for( ch <- localVN ) {
            if( findItemOrder(ex._1, ch.toString) != -1 ) {
                val gotoNumber = findItemOrder(ex._1, ch.toString)
                var tmpRow = -1
                tmpRow = ex._2 + 1
                //A = ch
                result(tmpRow)( getColumn(ch.toString).ans ) = gotoNumber.toString
            }
        }
    }
    result
}
```

3.5 程序运行截图

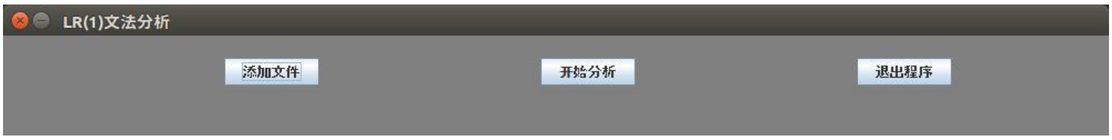
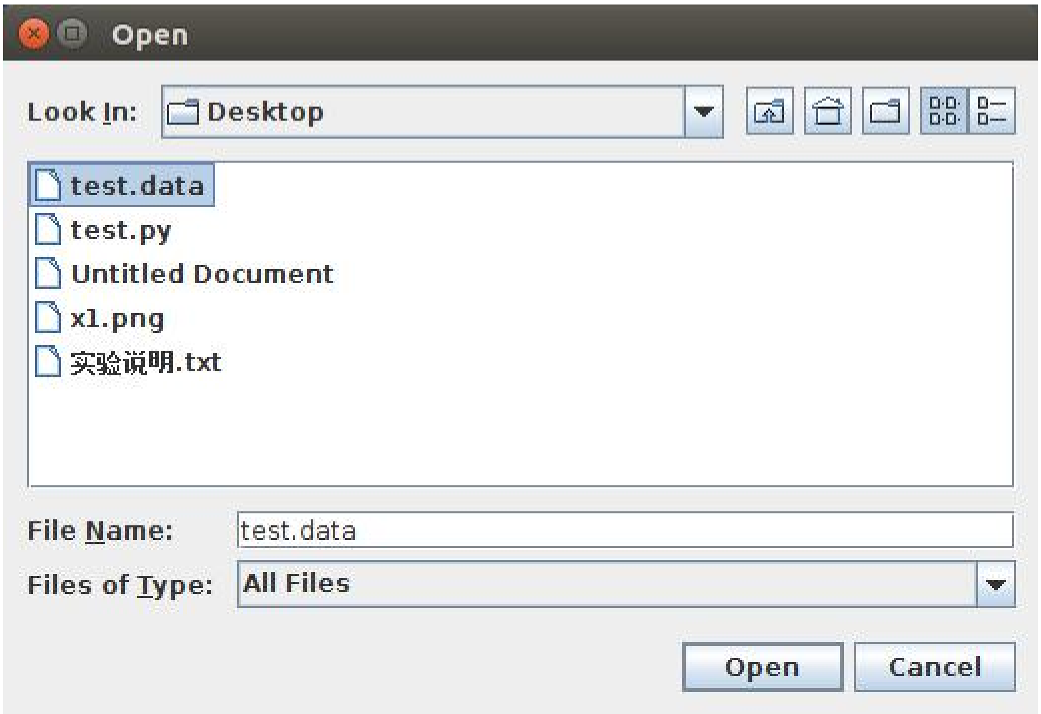


Figure 19 开始界面

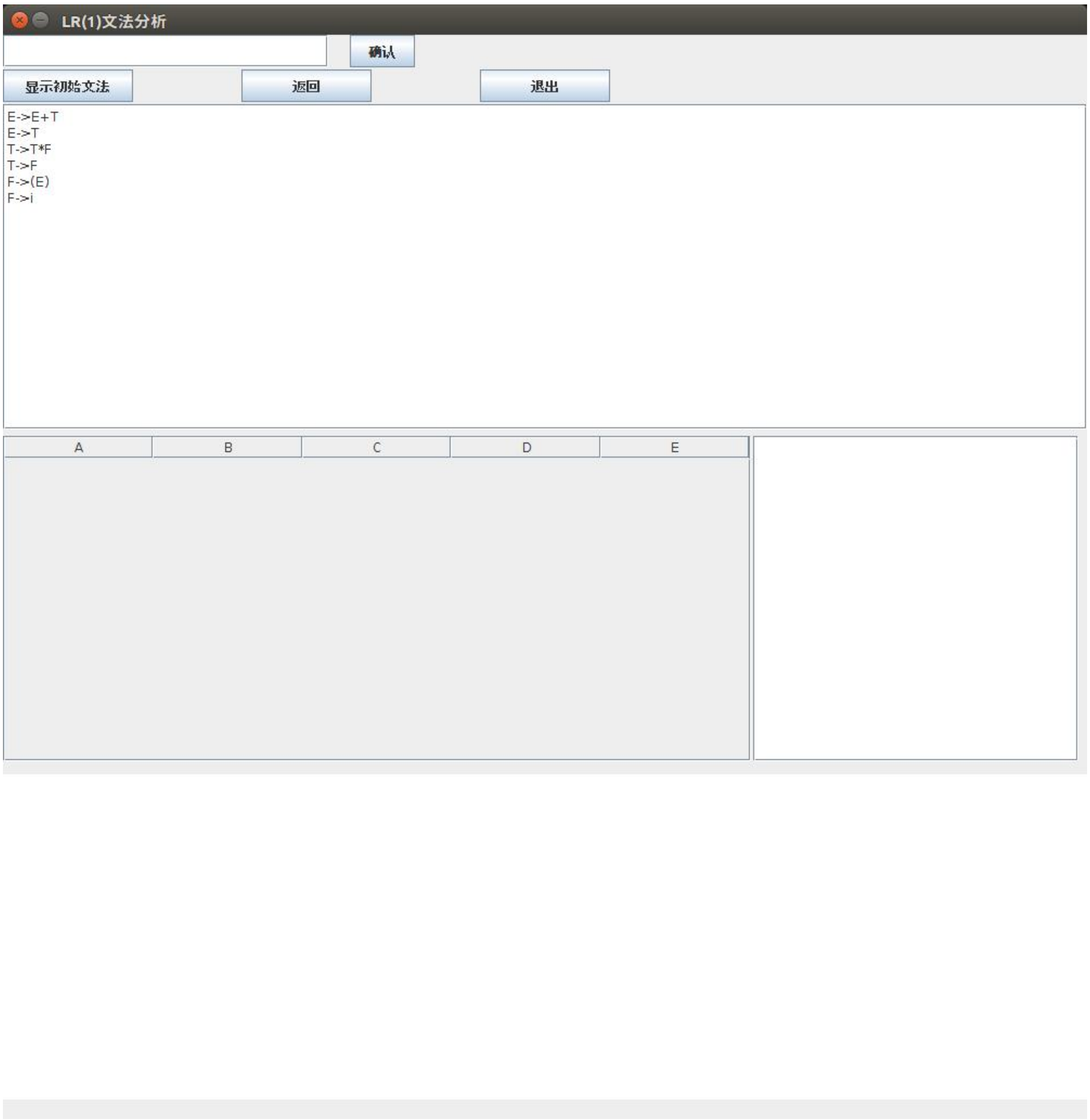

Look In: Desktop

test.data
test.py
Untitled Document
x1.png
实验说明.txt

File Name: test.data

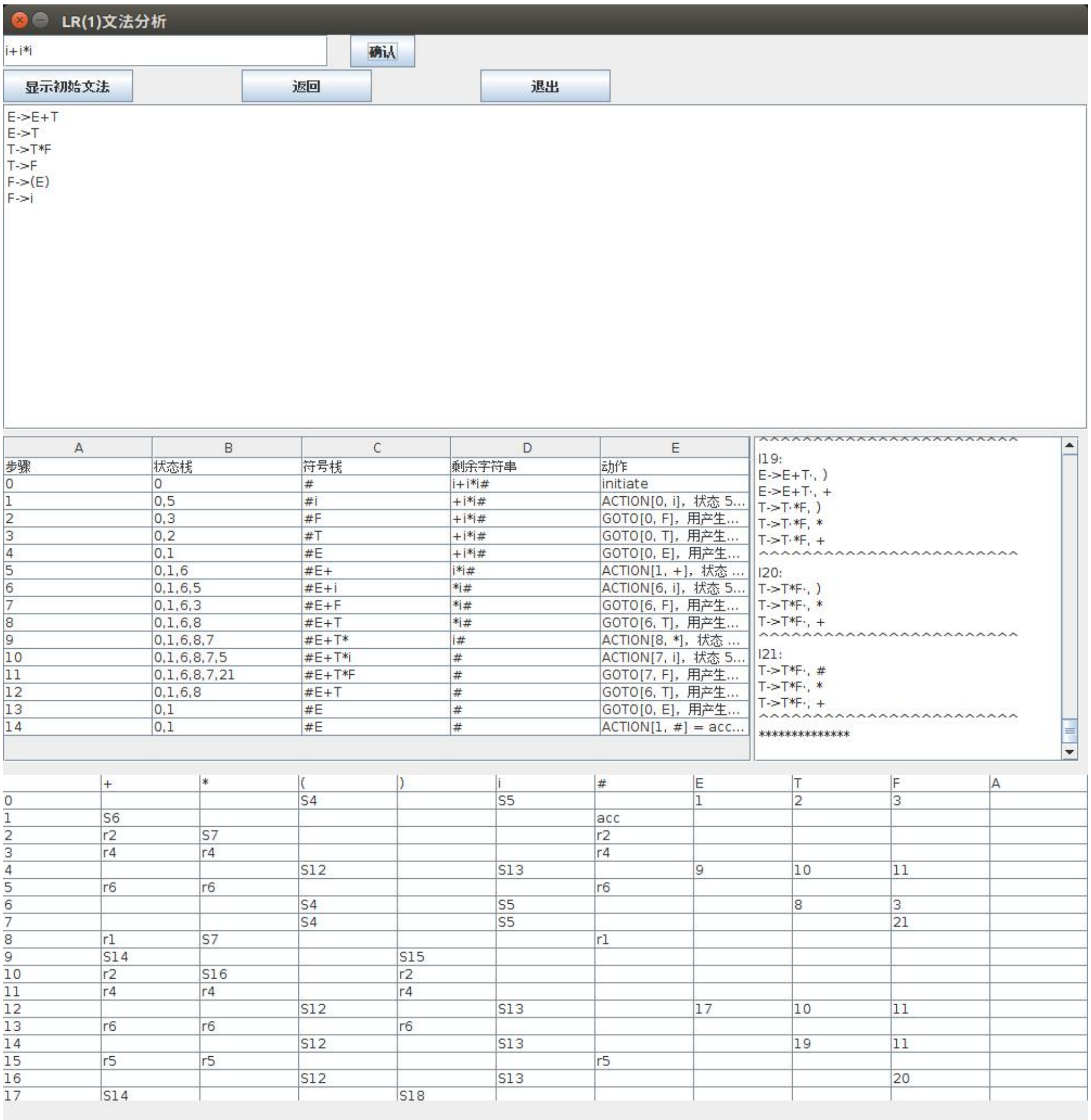
Files of Type: All Files

Figure 20 选择文件



```
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
```

Figure 21 显示初始文法



i+i*i

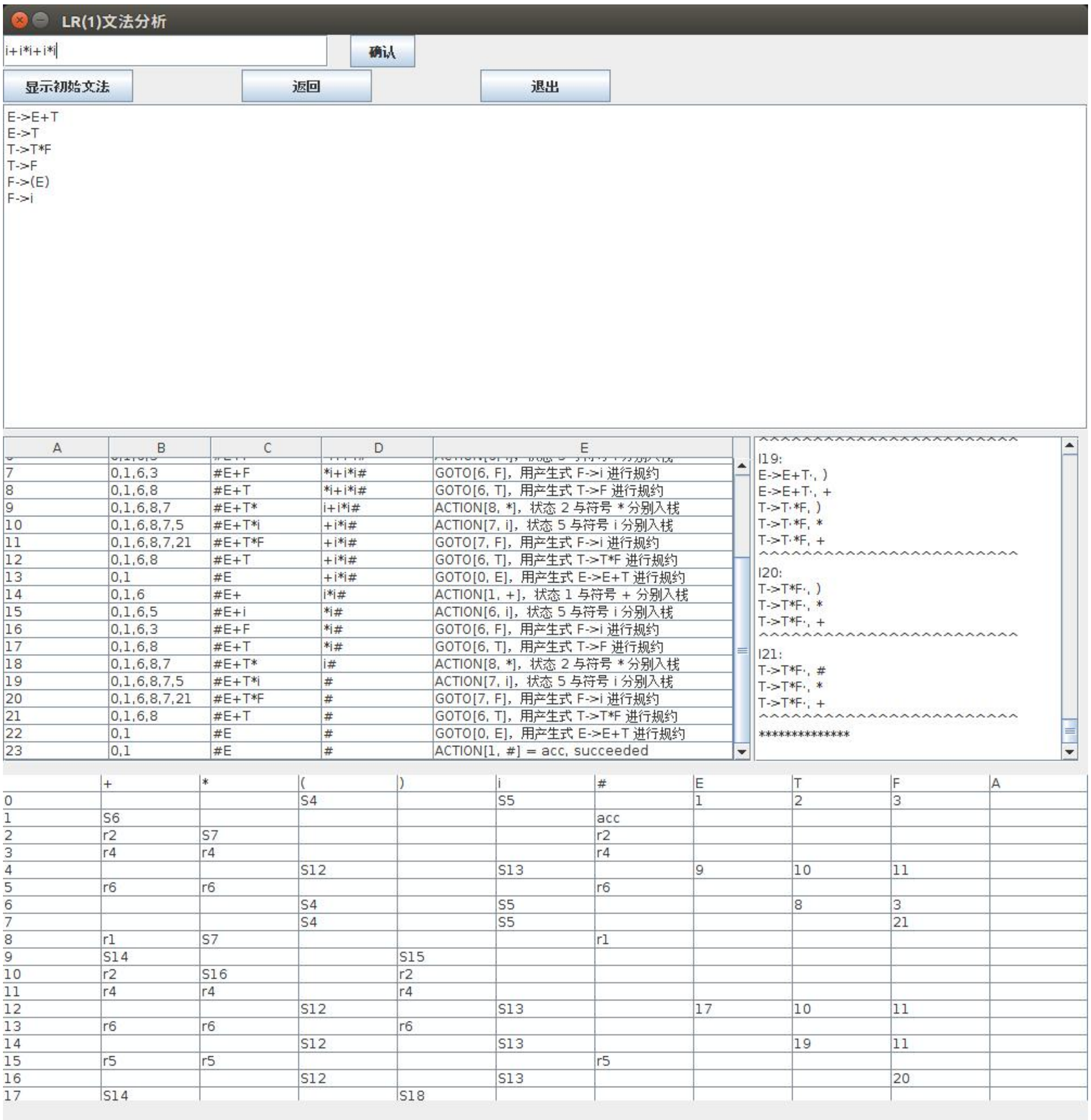
```
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
```

A	B	C	D	E
步骤	状态栈	符号栈	剩余字符串	动作
0	0	#	i+i*i#	initiate
1	0,5	#i	+i*i#	ACTION[0, i]，状态 5...
2	0,3	#F	+i*i#	GOTO[0, F]，用产生...
3	0,2	#T	+i*i#	GOTO[0, T]，用产生...
4	0,1	#E	+i*i#	GOTO[0, E]，用产生...
5	0,1,6	#E+	i*i#	ACTION[1, +]，状态 ...
6	0,1,6,5	#E+i	*i#	ACTION[6, i]，状态 5...
7	0,1,6,3	#E+F	*i#	GOTO[6, F]，用产生...
8	0,1,6,8	#E+T	*i#	GOTO[6, T]，用产生...
9	0,1,6,8,7	#E+T*	i#	ACTION[8, *]，状态 ...
10	0,1,6,8,7,5	#E+T*i	#	ACTION[7, i]，状态 5...
11	0,1,6,8,7,21	#E+T*F	#	GOTO[7, F]，用产生...
12	0,1,6,8	#E+T	#	GOTO[6, T]，用产生...
13	0,1	#E	#	GOTO[0, E]，用产生...
14	0,1	#E	#	ACTION[1, #] = acc...

```
^^^^^^^^^^^^^^^^^^^^^^^^^^
I19:
E->E+T·, )
E->E+T·, +
T->T·*F, )
T->T·*F, *
T->T·*F, +
^^^^^^^^^^^^^^^^^^^^^^^^^^
I20:
T->T*F·, )
T->T*F·, *
T->T*F·, +
^^^^^^^^^^^^^^^^^^^^^^^^^^
I21:
T->T*F·, #
T->T*F·, *
T->T*F·, +
^^^^^^^^^^^^^^^^^^^^^^^^^^
**************
```

	+	*	(	)	i	#	E	T	F	A
0			S4		S5		1	2	3	
1	S6					acc				
2	r2	S7				r2				
3	r4	r4				r4				
4			S12		S13		9	10	11	
5	r6	r6				r6				
6			S4		S5			8	3	
7			S4		S5				21	
8	r1	S7				r1				
9	S14			S15						
10	r2	S16		r2						
11	r4	r4		r4						
12			S12		S13		17	10	11	
13	r6	r6		r6						
14			S12		S13			19	11	
15	r5	r5				r5				
16			S12		S13				20	
17	S14			S18						

Figure 22 分析完成（输入表达死为“i+i*i”）



```
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
```

A	B	C	D	E
7	0,1,6,3	#E+F	*i+i*i#	GOTO[6, F]，用产生式 F->i 进行规约
8	0,1,6,8	#E+T	*i+i*i#	GOTO[6, T]，用产生式 T->F 进行规约
9	0,1,6,8,7	#E+T*	i+i*i#	ACTION[8, *]，状态 2 与符号 * 分别入栈
10	0,1,6,8,7,5	#E+T*i	+i*i#	ACTION[7, i]，状态 5 与符号 i 分别入栈
11	0,1,6,8,7,21	#E+T*F	+i*i#	GOTO[7, F]，用产生式 F->i 进行规约
12	0,1,6,8	#E+T	+i*i#	GOTO[6, T]，用产生式 T->T*F 进行规约
13	0,1	#E	+i*i#	GOTO[0, E]，用产生式 E->E+T 进行规约
14	0,1,6	#E+	i*i#	ACTION[1, +]，状态 1 与符号 + 分别入栈
15	0,1,6,5	#E+i	*i#	ACTION[6, i]，状态 5 与符号 i 分别入栈
16	0,1,6,3	#E+F	*i#	GOTO[6, F]，用产生式 F->i 进行规约
17	0,1,6,8	#E+T	*i#	GOTO[6, T]，用产生式 T->F 进行规约
18	0,1,6,8,7	#E+T*	i#	ACTION[8, *]，状态 2 与符号 * 分别入栈
19	0,1,6,8,7,5	#E+T*i	#	ACTION[7, i]，状态 5 与符号 i 分别入栈
20	0,1,6,8,7,21	#E+T*F	#	GOTO[7, F]，用产生式 F->i 进行规约
21	0,1,6,8	#E+T	#	GOTO[6, T]，用产生式 T->T*F 进行规约
22	0,1	#E	#	GOTO[0, E]，用产生式 E->E+T 进行规约
23	0,1	#E	#	ACTION[1, #] = acc, succeeded

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
I19:
E->E+T·, )
E->E+T·, +
T->T·*F, )
T->T·*F, *
T->T·*F, +
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
I20:
T->T*F·, )
T->T*F·, *
T->T*F·, +
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
I21:
T->T*F·, #
T->T*F·, *
T->T*F·, +
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
**************
```

	+	*	(	)	i	#	E	T	F	A
0			S4		S5		1	2	3	
1	S6					acc				
2	r2	S7				r2				
3	r4	r4				r4				
4			S12		S13		9	10	11	
5	r6	r6				r6				
6			S4		S5			8	3	
7			S4		S5				21	
8	r1	S7				r1				
9	S14			S15						
10	r2	S16		r2						
11	r4	r4		r4						
12			S12		S13		17	10	11	
13	r6	r6		r6						
14			S12		S13			19	11	
15	r5	r5				r5				
16			S12		S13				20	
17	S14			S18						

Figure 23 分析完成（输入表达死为“i+i*i+i*i”，单元格拉长）

参考文献

[1]Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers Principles, Techniques and Tools[M]. New York: Pearson Addison Wesley, 2006.

[2]陈火旺，钱家骅，孙永强. 程序设计语言编译原理（第 3 版）[M]. 北京：国防工业出版社, 1999.

附（实验代码链接）：

实验 1 博客：

https://blog.csdn.net/u25th_engineer/article/details/102458531

实验 2 github 地址：

https://github.com/25thengineer/Compile_Experiment_LL_1

实验 3 github 地址：

https://github.com/25thengineer/Compile_Experiment_LR_1