# 《编译原理》

## ——第五次实验报告

学　号：　**15281106**

姓　名：　徐开元

专　业：　计算机科学与技术

学　院：　计算机与信息技术学院

提交日期：　**2018** 年 **5** 月 **26** 日

指导老师：　徐金安

# 一、实验内容：

[实验项目] 完成以下描述赋值语句 SLR(1)文法语法制导生成中间代码四元式的过程。

G[A]:A→V=E

E→E+T | E-T | T

T→T*F | T/F | F

F→(E) | i V→i

[设计说明] 终结符号 i 为用户定义的简单变量，即标识符的定义。

# 二、程序描述

## 2.1 程序基本思路逻辑

1、此程序以实验一的输出，即每一行的二元式为输入，输出为输入串是否为该文法定义的算术表达式的判断结果。能够检查简单的错误，并在分析过程应能发现输入串出错。

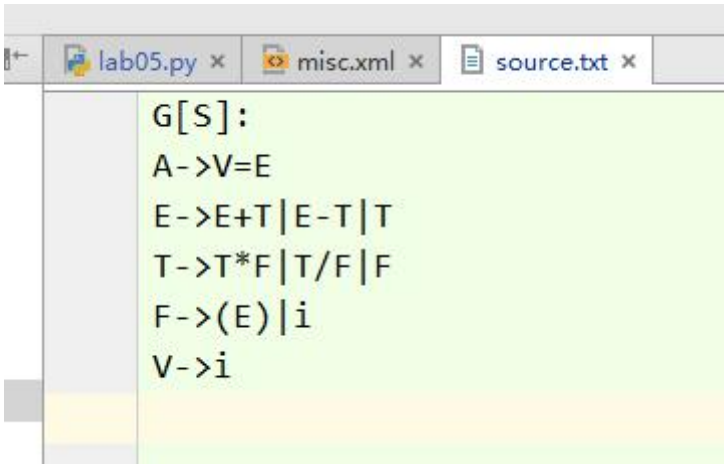
2、代码主要参照书中逻辑，并参照第四次实验代码，依次扫描存放文法的文件，生成文法字典、vt 集合、vn 集合，然后遍历终结符和非终结符通过 firstVT(P)={a|P->a…..|P->Qa….|firstVT(Q)}求出 firstVT 集合和通过 firstVT(P)={a|P->…a.|P->…aQ.|lastVT(Q)}求出 lastVT 集合。求集合时，因为要考虑到每一次考虑，因此将每一次变化的文法符号入栈，最后遍历栈，确保每个变化都使整个集合能得到更新。

3、由于这次实验时间和调 bug 时间能力不足，因此暂定在纸上写好造好表，LR（0）项目集、GOTO 表和 ACTION 表 SLR 分析表。程序主要实现分析过程。造表中对于当前所有的项目集族进行闭包操作举个例子，如果 A->α·aβ 属于 Ii，并且 GOTO[Ii，a]= Ij，那么 ACTION[i，a]= sj，如果 Ii 包含 A→α· ，那么[i，x]= rj，x 属于 FOLLOW(A)。如果 Ii 包含 S’→S·，action[i，$]= acc，如果 goto [Ii，A]= Ij，则 goto[i，A]= j

4、对于每个表格的打印，使用 python 的 prettytable 库进行美化输出。最后分析过程中，如果拒接接受，打印错误字符以及错误位置，方便定位。

## 2.2 程序输入源文件

输入源文件 source.txt 为实验要求的文法，以及两个示例输入分别为正确接收的 i+i*i# 和错误接收的 i=i+i*ii#：

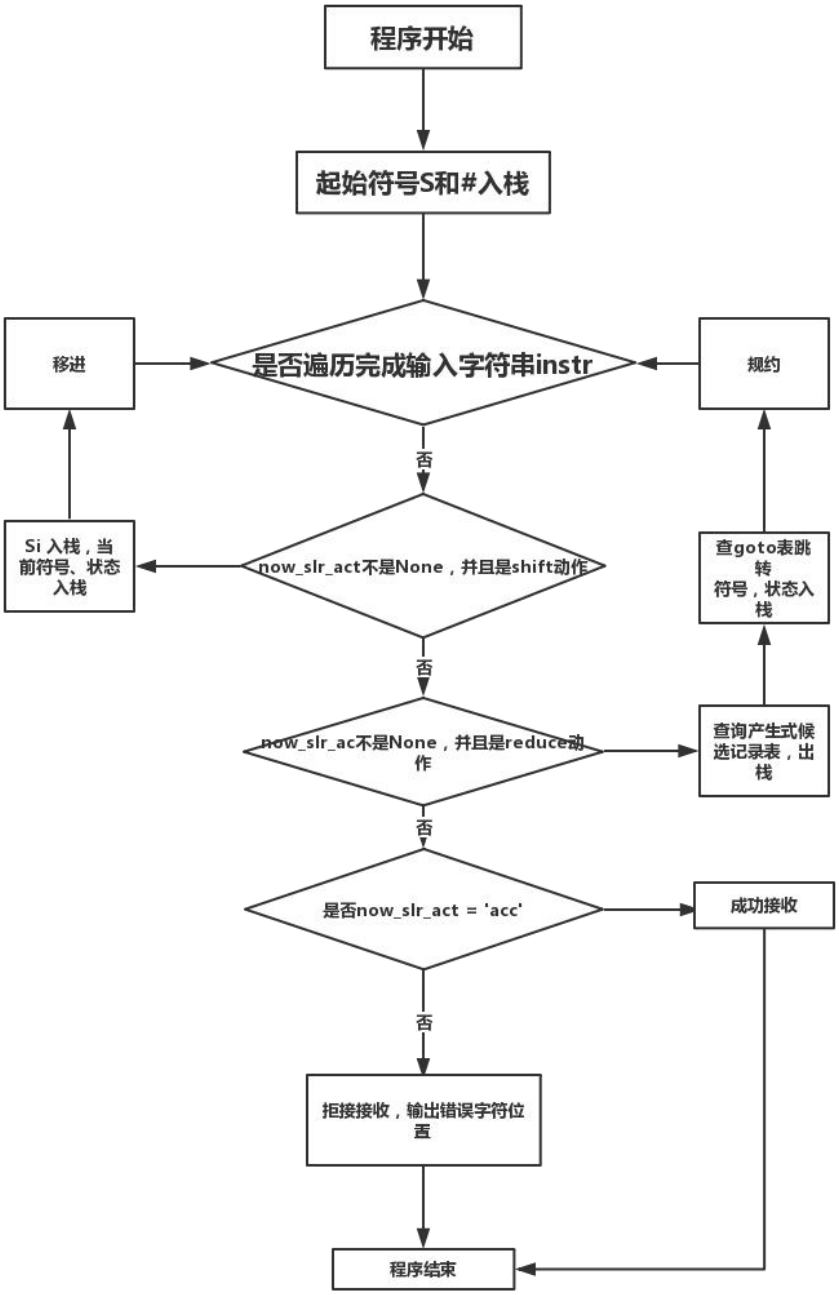



文法输入文件

```
signal_arr = []
signal_arr.append('#')
in_str = 'i=i+i*ii#'
# in_str = 'i=i*(i+i)#'
print('\t\t\t\tSLR(1)分析过程如下')
tb = PrettyTable(["状态栈", '符号栈', '输入符号串', 'ACTION', 'GOTO'])
while True:
```

输入字符串

## 2.3 程序运行流程图




主函数 SLR（1）分析流程图

## 2.4 程序示例

关键代码如下：代码均在 lab05.py 中。添加输入美化表格库，如果不存在则进行安装。

```
# 添加表格包
try:
    from prettytable import PrettyTable
except Exception as e:
    print(e)
    try:
        os.system('python -m pip install prettytable')
        print("安装prettytable包成功")
        from prettytable import PrettyTable
    except Exception as e:
        print(e)


# 申明全局变量用于修改。
def gen_lang():
    global LANGUAGE
    global VT
    global VN
    global START

    with open('source.txt', 'r', encoding='utf8')as f:
        ls = f.readlines()
        START = ls[0][2]
        for l in ls[1:]:
            l = l.strip('\n')
            l = l.replace('E`', 'Z')
            l = l.replace('T`', 'X')
            key, value = l.split("->")
            if key == 'T`':
                key = 'Z'
            if key == 'E`':
                key = 'X'
```

构建 GOTO 和 ACTION 表，并打印出来。

```
FIRST = {}
FOLLOW = {}
Table = {}
GOTO = {('0', ' i'): 's3', ('1', ' #'): 'acc', ('2', ' ='): ')', ('3', ' ='): ')', ('4', ' ('): 's8', ('4', ' i'): 's9',
        ('5', ' #'): 'r1', ('5', ' +'): 's10', ('5', ' -'): 's11', ('6', ' #'): 'r4', ('6', ' +'): 'r4',
        ('6', ' -'): 'r4', ('6', ' )'): 'r4', ('6', ' *'): 's12', ('6', ' /'): 's13', ('7', ' #'): 'r7',
        ('7', ' +'): 'r7', ('7', ' -'): 'r7', ('7', ' *'): 'r7', ('7', ' /'): 'r7', ('7', ' )'): 'r7',
        ('8', ' ('): 's8', ('8', ' i'): 's9', ('9', ' #'): 'r9', ('9', ' +'): 'r9', ('9', ' -'): 'r9',
        ('9', ' *'): 'r9', ('9', ' /'): 'r9', ('9', ' )'): 'r9', ('10', ' ('): 's8', ('10', ' i'): 's9',
        ('11', ' ('): 's8', ('11', ' i'): 's9', ('12', ' ('): 's8', ('12', ' i'): 's9', ('13', ' ('): 's8',
        ('13', ' i'): 's9', ('14', ' )'): 's19', ('14', ' +'): 's10', ('14', ' -'): 's11', ('15', ' #'): 'r2',
        ('15', ' +'): 'r2', ('15', ' -'): 'r2', ('15', ' )'): 'r2', ('15', ' *'): 's12', ('15', ' /'): 's13',
        ('16', ' #'): 'r3', ('16', ' +'): 'r3', ('16', ' -'): 'r3', ('16', ' )'): 'r3', ('16', ' *'): 's12',
        ('16', ' /'): 's13', ('17', ' #'): 'r5', ('17', ' +'): 'r5', ('17', ' -'): 'r5', ('17', ' *'): 'r5',
        ('17', ' /'): 'r5', ('17', ' )'): 'r5', ('18', ' #'): 'r6', ('18', ' +'): 'r6', ('18', ' -'): 'r6',
        ('18', ' *'): 'r6', ('18', ' /'): 'r6', ('18', ' )'): 'r6', ('19', ' #'): 'r8', ('19', ' +'): 'r8',
        ('19', ' -'): 'r8', ('19', ' *'): 'r8', ('19', ' /'): 'r8', ('19', ' )'): 'r8'}

ACTION = {('0', ' A'): '1', ('0', ' V'): '2', ('4', ' E'): '5', ('4', ' T'): '6', ('4', ' F'): '7', ('8', ' E'): '14',
          ('8', ' T'): '6', ('8', ' F'): '7', ('10', ' T'): '15', ('10', ' F'): '7', ('11', ' T'): '16',
          ('11', ' F'): '7', ('12', ' F'): '17', ('13', ' F'): '18'}
```

构建 SLR（1）分析表

```
SLR_TABLE = [[None, None, None, None, None, None, None, 's3', None, 1, 2, None, None, None],
             [None, None, None, None, None, None, None, None, 'acc', None, None, None, None, None],
             ['s4', None, None, None, None, None, None, None, None, None, None, None, None, None],
             ['r10', None, None, None, None, None, None, None, None, None, None, None, None, None],
             [None, None, None, None, None, 's8', None, 's9', None, None, None, 5, 6, 7],
             [None, 's10', 's11', None, None, None, None, None, 'r1', None, None, None, None, None],
             [None, 'r4', 'r4', 's12', 's13', None, 'r4', None, 'r4', None, None, None, None, None],
             [None, 'r7', 'r7', 'r7', 'r7', None, 'r7', None, 'r7', None, None, None, None, None],
             [None, None, None, None, None, 's8', None, 's9', None, None, None, 14, 6, 7],
             [None, 'r9', 'r9', 'r9', 'r9', None, 'r9', None, 'r9', None, None, None, None, None],
             [None, None, None, None, None, 's8', None, 's9', None, None, None, None, 15, 7],
             [None, None, None, None, None, 's8', None, 's9', None, None, None, None, 16, 7],
             [None, None, None, None, None, 's8', None, 's9', None, None, None, None, None, 17],
             [None, None, None, None, None, 's8', None, 's9', None, None, None, None, None, 18],
             [None, 's10', 's11', None, None, None, 's19', None, None, None, None, None, None, None],
             [None, 'r2', 'r2', 's12', 's13', None, 'r2', None, 'r2', None, None, None, None, None],
             [None, 'r3', 'r3', 's12', 's13', None, 'r3', None, 'r3', None, None, None, None, None],
             [None, 'r5', 'r5', 'r5', 'r5', None, 'r5', None, 'r5', None, None, None, None, None],
             [None, 'r6', 'r6', 'r6', 'r6', None, 'r6', None, 'r6', None, None, None, None, None],
             [None, 'r8', 'r8', 'r8', 'r8', None, 'r8', None, 'r8', None, None, None, None, None]]
```

SLR（1)文法函数，根据流程图实现代码。

```
now = 0
stage_arr = []
stage_arr.append(0)
# 创建符号栈
signal_arr = []
signal_arr.append('#')
in_str = 'i=i+i*ii#'
# in_str = 'i=i*(i+i)#'
print('\t\t\t\tSLR(1)分析过程如下')
tb = PrettyTable(["状态栈", '符号栈', '输入符号串', 'ACTION', 'GOTO'])
while True:
    row = []
    # 加入状态栈行和符号栈行
    row.append(str(stage_arr))
    row.append(str(signal_arr))
    # 读入下一个状态
    now_stage = stage_arr[-1]
    now_slr_act = SLR_TABLE[now_stage][find_vt(in_str[now])]
    if now_slr_act and 's' in now_slr_act:
        # 如果当前状态与字符指针所指字符在SLR表中是ACTION,状态转换，则把当前now_stage字符压栈，
        # 因为每次压栈时要同时压入字符和状态，此时压入的对应的状态就是ACTION里标明的状态，然后读取下一个字符
        next_status = int(now_slr_act[1:])
        stage_arr.append(next_status)
        signal_arr.append(in_str[now])
        row.append(in_str[now:])
        row.append(now_slr_act)
        now += 1
        if len(row) != 5:
            row.append('')
        tb.add_row(row)
    elif now_slr_act and 'r' in now_slr_act:
        # 如果当前状态与字符指针所指字符在SLR表中是REDUCE，首先查看Ri对应的产生式，
        # 比如说A->V=E，此时则需要弹出栈中的V=E，把A压栈
```

```
        row.append(now_slr_act)
        now += 1
        if len(row) != 5:
            row.append('')
        tb.add_row(row)
    elif now_slr_act and 'r' in now_slr_act:
        # 如果当前状态与字符指针所指字符在SLR表中是REDUCE，首先查看Ri对应的产生式，
        # 比如说A->V=E，此时则需要弹出栈中的V=E，把A压栈
        row.append(in_str[now:])
        row.append(now_slr_act)
        rulepos = int(now_slr_act[1:])
        count = 0
        while count != len(LANGUAGE[rulepos][1]):
            # 与A同时压栈的状态为当前的状态
            # 字符和状态一起弹出
            signal_arr.pop()
            stage_arr.pop()
            count += 1
        signal_arr.append(LANGUAGE[rulepos][0])
        now_siagnl = signal_arr[-1]
        while True:
            # 查找A在SLR表中对应的GOTO，如果当前状态为1，那么与A一起压栈的状态就是(1，A)对应的GOTO状态。
            now_stage = stage_arr[-1]
            now_slr_go = SLR_TABLE[now_stage][find_vn(now_siagnl) + len(VT) - 1]
            if now_slr_go:
                stage_arr.append(now_slr_go)
                row.append(now_slr_go)
                break
            else:
                stage_arr.pop()
        if len(row) != 5:
            row.append('')
        tb.add_row(row)
    # 如果状态是acc则结束分析，接受字符串
```

```
            tb.add_row(row)
        # 如果状态是acc则结束分析，接受字符串
        elif now_slr_act == 'acc':
            print(tb)
            print('接受字符串:%s ！' % in_str)
            break
        else:
            print(tb)
            print('%s为不合法字符串' % in_str)
            print('接受字符串失败，错误字符串:%s ,错误字符：%s！' % (in_str, in_str[0]))
            break
```

初始化读取文法结果,并计算 first 和 follow 集。

```
D:\python3\python.exe D:/pycharmproject/zuoye/编译原理/lab05.py
START: S VN: {'A', 'V', 'F', 'E', 'T'} VT: {'/', '+', '-', '=', ')', 'i', '(', '*'}
LANGUAGE: {'A': ['V=E'], 'E': ['E+T', 'E-T', 'T'], 'T': ['T*F', 'T/F', 'F'], 'F': ['(E)', 'i'], 'V': ['i']}
FIRST集为: {'A': ['i'], 'E': ['i', '('], 'T': ['i', '('], 'F': ['(', 'i'], 'V': ['i']}
FOLLOW集为: {'A': ['#'], 'E': [')', '-', '#', '+'], 'T': ['/', '+', '-', ')', '#', '*'], 'F': ['/', '+', '-', ')', '#', '*'], 'V': ['=']}
      GOTO表
0 + i -> s3   1 + # -> acc   2 + = -> ) 3 + = -> ) 4 + ( -> s8
```

GOTO 表和 ACTION 表

```
      GOTO表
0 + i -> s3   1 + # -> acc   2 + = -> ) 3 + = -> ) 4 + ( -> s8
4 + i -> s9   5 + # -> r1    5 + + -> s10   5 + - -> s11   6 + # -> r4
6 + + -> r4   6 + - -> r4    6 + ) -> r4    6 + * -> s12   6 + / -> s13
7 + # -> r7   7 + + -> r7    7 + - -> r7    7 + * -> r7    7 + / -> r7
7 + ) -> r7   8 + ( -> s8    8 + i -> s9    9 + # -> r9    9 + + -> r9
9 + - -> r9   9 + * -> r9    9 + / -> r9    9 + ) -> r9    10 + ( -> s8
10 + i -> s9  11 + ( -> s8   11 + i -> s9   12 + ( -> s8   12 + i -> s9
13 + ( -> s8  13 + i -> s9   14 + ) -> s19  14 + + -> s10  14 + - -> s11
15 + # -> r2  15 + + -> r2   15 + - -> r2   15 + ) -> r2   15 + * -> s12
15 + / -> s13 16 + # -> r3   16 + + -> r3   16 + - -> r3   16 + ) -> r3
16 + * -> s12 16 + / -> s13  17 + # -> r5   17 + + -> r5   17 + - -> r5
17 + * -> r5  17 + / -> r5   17 + ) -> r5   18 + # -> r6   18 + + -> r6
18 + - -> r6  18 + * -> r6   18 + / -> r6   18 + ) -> r6   19 + # -> r8
19 + + -> r8  19 + - -> r8   19 + * -> r8   19 + / -> r8   19 + ) -> r8
      ACTION表
0 + A -> 1 0 + V -> 2 4 + E -> 5 4 + T -> 6 4 + F -> 7
8 + E -> 14   8 + T -> 6 8 + F -> 7 10 + T -> 15   10 + F -> 7
11 + T -> 16  11 + F -> 7    12 + F -> 17   13 + F -> 18
```

SLR 分析表

SLR分析表

| | = | + | - | * | / | ( | ) | i | # | A | V | E | T | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | None | None | None | None | None | None | None | s3 | None | 1 | 2 | None | None | None |
| 1 | None | None | None | None | None | None | None | None | acc | None | None | None | None | None |
| 2 | s4 | None | None | None | None | None | None | None | None | None | None | None | None | None |
| 3 | r10 | None | None | None | None | None | None | None | None | None | None | None | None | None |
| 4 | None | None | None | None | None | s8 | None | s9 | None | None | None | 5 | 6 | 7 |
| 5 | None | s10 | s11 | None | None | None | None | None | r1 | None | None | None | None | None |
| 6 | None | r4 | r4 | s12 | s13 | None | r4 | None | r4 | None | None | None | None | None |
| 7 | None | r7 | r7 | r7 | r7 | None | r7 | None | r7 | None | None | None | None | None |
| 8 | None | None | None | None | None | s8 | None | s9 | None | None | None | 14 | 6 | 7 |
| 9 | None | r9 | r9 | r9 | r9 | None | r9 | None | r9 | None | None | None | None | None |
| 10 | None | None | None | None | None | s8 | None | s9 | None | None | None | None | 15 | 7 |
| 11 | None | None | None | None | None | s8 | None | s9 | None | None | None | None | 16 | 7 |
| 12 | None | None | None | None | None | s8 | None | s9 | None | None | None | None | None | 17 |
| 13 | None | None | None | None | None | s8 | None | s9 | None | None | None | None | None | 18 |
| 14 | None | s10 | s11 | None | None | None | s19 | None | None | None | None | None | None | None |
| 15 | None | r2 | r2 | s12 | s13 | None | r2 | None | r2 | None | None | None | None | None |
| 16 | None | r3 | r3 | s12 | s13 | None | r3 | None | r3 | None | None | None | None | None |
| 17 | None | r5 | r5 | r5 | r5 | None | r5 | None | r5 | None | None | None | None | None |
| 18 | None | r6 | r6 | r6 | r6 | None | r6 | None | r6 | None | None | None | None | None |
| 19 | None | r8 | r8 | r8 | r8 | None | r8 | None | r8 | None | None | None | None | None |

当输入字符串为 i=i*(i+i)#时，接受字符串，分析表情况如图。

SLR(1)分析过程如下

| 状态栈 | 符号栈 | 输入符号串 | ACTION | GOTO |
|---|---|---|---|---|
| [0] | ['#'] | i=i*(i+i)# | s3 | |
| [0, 3] | ['#', 'i'] | =i*(i+i)# | r10 | 2 |
| [0, 2] | ['#', 'V'] | =i*(i+i)# | s4 | |
| [0, 2, 4] | ['#', 'V', '='] | i*(i+i)# | s9 | |
| [0, 2, 4, 9] | ['#', 'V', '=', 'i'] | *(i+i)# | r9 | 7 |
| [0, 2, 4, 7] | ['#', 'V', '=', 'F'] | *(i+i)# | r7 | 6 |
| [0, 2, 4, 6] | ['#', 'V', '=', 'T'] | *(i+i)# | s12 | |
| [0, 2, 4, 6, 12] | ['#', 'V', '=', 'T', '*'] | (i+i)# | s8 | |
| [0, 2, 4, 6, 12, 8] | ['#', 'V', '=', 'T', '*', '('] | i+i)# | s9 | |
| [0, 2, 4, 6, 12, 8, 9] | ['#', 'V', '=', 'T', '*', '(', 'i'] | +i)# | r9 | 7 |
| [0, 2, 4, 6, 12, 8, 7] | ['#', 'V', '=', 'T', '*', '(', 'F'] | +i)# | r7 | 6 |
| [0, 2, 4, 6, 12, 8, 6] | ['#', 'V', '=', 'T', '*', '(', 'T'] | +i)# | r4 | 14 |
| [0, 2, 4, 6, 12, 8, 14] | ['#', 'V', '=', 'T', '*', '(', 'E'] | +i)# | s10 | |
| [0, 2, 4, 6, 12, 8, 14, 10] | ['#', 'V', '=', 'T', '*', '(', 'E', '+'] | i)# | s9 | |
| [0, 2, 4, 6, 12, 8, 14, 10, 9] | ['#', 'V', '=', 'T', '*', '(', 'E', '+', 'i'] | )# | r9 | 7 |
| [0, 2, 4, 6, 12, 8, 14, 10, 7] | ['#', 'V', '=', 'T', '*', '(', 'E', '+', 'F'] | )# | r7 | 15 |
| [0, 2, 4, 6, 12, 8, 14, 10, 15] | ['#', 'V', '=', 'T', '*', '(', 'E', '+', 'T'] | )# | r2 | 14 |
| [0, 2, 4, 6, 12, 8, 14] | ['#', 'V', '=', 'T', '*', '(', 'E'] | )# | s19 | |
| [0, 2, 4, 6, 12, 8, 14, 19] | ['#', 'V', '=', 'T', '*', '(', 'E', ')'] | # | r8 | 17 |
| [0, 2, 4, 6, 12, 17] | ['#', 'V', '=', 'T', '*', 'F'] | # | r5 | 6 |
| [0, 2, 4, 6] | ['#', 'V', '=', 'T'] | # | r4 | 5 |
| [0, 2, 4, 5] | ['#', 'V', '=', 'E'] | # | r1 | 1 |

接受字符串:i=i*(i+i)# !

花费时间: 0.009964227676391602

当输入字符串为 i=i+i*ii#时，拒绝接收字符串,错误字符为 i。

SLR(1)分析过程如下

| 状态栈 | 符号栈 | 输入符号串 | ACTION | GOTO |
|---|---|---|---|---|
| [0] | ['#'] | i=i+i*ii# | s3 | |
| [0, 3] | ['#', 'i'] | =i+i*ii# | r10 | 2 |
| [0, 2] | ['#', 'V'] | =i+i*ii# | s4 | |
| [0, 2, 4] | ['#', 'V', '='] | i+i*ii# | s9 | |
| [0, 2, 4, 9] | ['#', 'V', '=', 'i'] | +i*ii# | r9 | 7 |
| [0, 2, 4, 7] | ['#', 'V', '=', 'F'] | +i*ii# | r7 | 6 |
| [0, 2, 4, 6] | ['#', 'V', '=', 'T'] | +i*ii# | r4 | 5 |
| [0, 2, 4, 5] | ['#', 'V', '=', 'E'] | +i*ii# | s10 | |
| [0, 2, 4, 5, 10] | ['#', 'V', '=', 'E', '+'] | i*ii# | s9 | |
| [0, 2, 4, 5, 10, 9] | ['#', 'V', '=', 'E', '+', 'i'] | *ii# | r9 | 7 |
| [0, 2, 4, 5, 10, 7] | ['#', 'V', '=', 'E', '+', 'F'] | *ii# | r7 | 15 |
| [0, 2, 4, 5, 10, 15] | ['#', 'V', '=', 'E', '+', 'T'] | *ii# | s12 | |
| [0, 2, 4, 5, 10, 15, 12] | ['#', 'V', '=', 'E', '+', 'T', '*'] | ii# | s9 | |

i=i+i*ii#为不合法字符串
接受字符串失败，错误字符串:i=i+i*ii# ,错误字符: i!
花费时间: 0.00601959228515625

# 三、实验心得

因为最后要把所有做过的实验汇总成为一个系统，因此准备都用 python 来实现，界面也可以用 pyqt 或者 html+python 来实现。本次实验用了 python3 语言来进行实现，虽成该实验花费了许多时间，但是我从过程中我学到了很多东西，对程序设计理解，算法的设计，汇编语法分析流程都有了不小的提高。

在实验过程中遇到了许多挫折，及时老师努力将其具体化形象化，但是在某些地方我还是存在不解，课本很复杂，也很难看懂。但是通过这次实验的实现，我更加全面的理解并了编译原理程序构造的一般原理和基本实现方法。并能够把在课堂上学的知识通过实验呈现表示出来，加深了对理论知识的理解。其中 SLR 文法很容易产生移入规约冲突，在 LR(1)分析表中引入后续符号结合很好的解决了这个问题。编译原理的处理问题的思想，思考问题的角度，都让我大受启发。编译原理博大精深，比如说对于文法的处理分析、中间代码的生成等很多方面，都值得我认真揣摩。