

北京交通大学

编译原理

LL(1) 分析实验报告

LL(1) Analysis Experimental Report

学 院： 计算机与信息技术

专 业： 计算机科学

学生姓名： 刘宜进

学 号： 14282008

指导教师： 徐金安

北京交通大学

2017 年 5 月

目 录

目 录	II
1 实验目的	3
2 实验内容	3
2.1 程序功能描述	3
2.2 程序结构	3
2.2.1 读取用户输入	4
2.2.2 LL(1)分析	4
2.2.3 过程展示	4
2.3 数据结构	5
2.4 主要函数	6
2.5 程序执行图	6
3 程序测试	7
3.1 测试用例	7
3.2 测试结果	7
3.3 结果分析	8
附 录	9

1 实验目的

完成以下描述算术表达式的 LL(1)文法的递归下降分析程序:

$$E \rightarrow TE$$
$$E' \rightarrow ATE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow MFT' \mid \varepsilon$$
$$F \rightarrow (E) \mid i$$
$$A \rightarrow + \mid -$$
$$M \rightarrow * \mid /$$

- 1、输入串应是词法分析的输出二元式序列，即某算术表达式“专题 1”的输出结果，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；
- 2、递归下降分析 程序应能发现输入串出错；
- 3、设计两个测试用例（尽可能完备，正确和出错），并给出测试结果。

2 实验内容

该实验运用 LL(1)分析法的基本原理，针对以上文法描述语言，利用 Python 语言实现了的 LL(1)分析程序。下面，通过程序功能描述、程序结构、数据结构、主要函数、程序执行图等五方面展开详细介绍。

2.1 程序功能描述

该程序能够持续读取用户输入，并进行 LL(1)分析，同时展示各个步骤的分析栈和剩余串情况。同时利用上个实验中写的绘制表格库函数 Drawtable 来进行结果展示。

2.2 程序结构

该程序主要有三大部分组成：

- 1、读取用户输入
- 2、LL(1)分析
- 3、中间过程展示、

2.2.1 读取用户输入

利用一个 `While(1)` 循环，持续读取用户输入，直至用户输入 “exit” 时退出程序。首先对用户输入源串进行基本的处理，如取出空格的影响。

```
inputString = input("请输入语句(递归下降): ")
```

再将用户输入转化为列表(List)形式以方便后期分析使用，同时在列表的最后添加上一个 “#” 表示源串的结束。

```
inputString = list(inputString)
```

```
inputString.append('#')
```

2.2.2 LL(1)分析

这是程序的核心部分，主要由 `Analysis()` 函数实现，函数首先对分析栈、剩余串进行初始化操作，然后进入一个 `While` 循环，循环结束标志 `Flag` 在栈顶元素和剩余串同时为 # 时置为成功。

`While` 循环体中，首先判断当前栈顶元素是否是 `Vt`，接着判断是否是当前分析字符，是则将字符串移进一位，否则报错 “非法字符”。接着判断栈顶元素是否是 #，如果当前字符也是 #，代表分析成功并退出，否则报错。

最后进行查表操作，注意可能产生非法表项。应该采用 `try`、`except` 语句进行异常处理，如下：

```
try:
```

```
    result = table[A][a]
```

```
except:
```

```
    error('查表出错')
```

```
    return False
```

如果查表成功则将字符栈退栈，并将产生式的右部逆序进栈，持续进行，直至循环分析完成，或者报错。

2.2.3 过程展示

为了方便看出在什么时候那个函数调用了那个函数，我特意写了一个方便展示结果的命令行端绘制表格工具 `DrawTable()`。

该函数接受五个参数：

Header 是字符串变量，表示表格的题目；

SubHeader 也是字符串变量，是表格的副标题；

Component 是二维链表，分别对应着表格的内容；

Length 是一个整数，它表示绘制表格的长度，缺省值为 80；

Center 是一个布尔值，**center = 1** 是表示居中显示，0 表示左对齐显示。

由于中文字符在命令行中的输出占据宽度是英文符号的两倍，为了表格的工整美观，我特意增加了一个判断表格各个表项中蕴含汉语的个数 **ContainChinese**。该函数接受一个字符串，返回字符串中包含中文的个数。

2.3 数据结构

该程序主要涉及一个分析栈、一个输入字符串、一个二维 **Dict** 表示的分析表以及一个表示表格内容的二维 **List**。

分析栈 **Stack** 属于 **List** 类，不过规定它的操作只能在栈顶进行，因此设置一个字符串变量 **top** 表示栈顶元素。

输入字符串为用户输入，为了方便分析将其转化为 **List** 形式，同时自动在其尾部添加一个 #。

最后是分析表，它的构造比较复杂，是由嵌套字典组成的一个字符表，它的具体定义如下：

```
table = {
    'E': {'i': ['E_', 'T'], '(': ['E_', 'T']},
    'E_': {'+': ['E_', 'T', 'A'], '-': ['E_', 'T', 'A'], ')': [], '#': []}, #用[]代表空串
    'T': {'i': ['T_', 'F'], '(': ['T_', 'F']},
    'T_': {'+': [], '-': [], '*': ['T_', 'F', 'M'], '/': ['T_', 'F', 'M'], ')': [], '#': []},
    'F': {'i': ['i'], '(': [], ')': [], 'E': []},
    'A': {'+': ['+'], '-': ['-']},
    'M': {'*': ['*'], '/': ['/']}
}
```

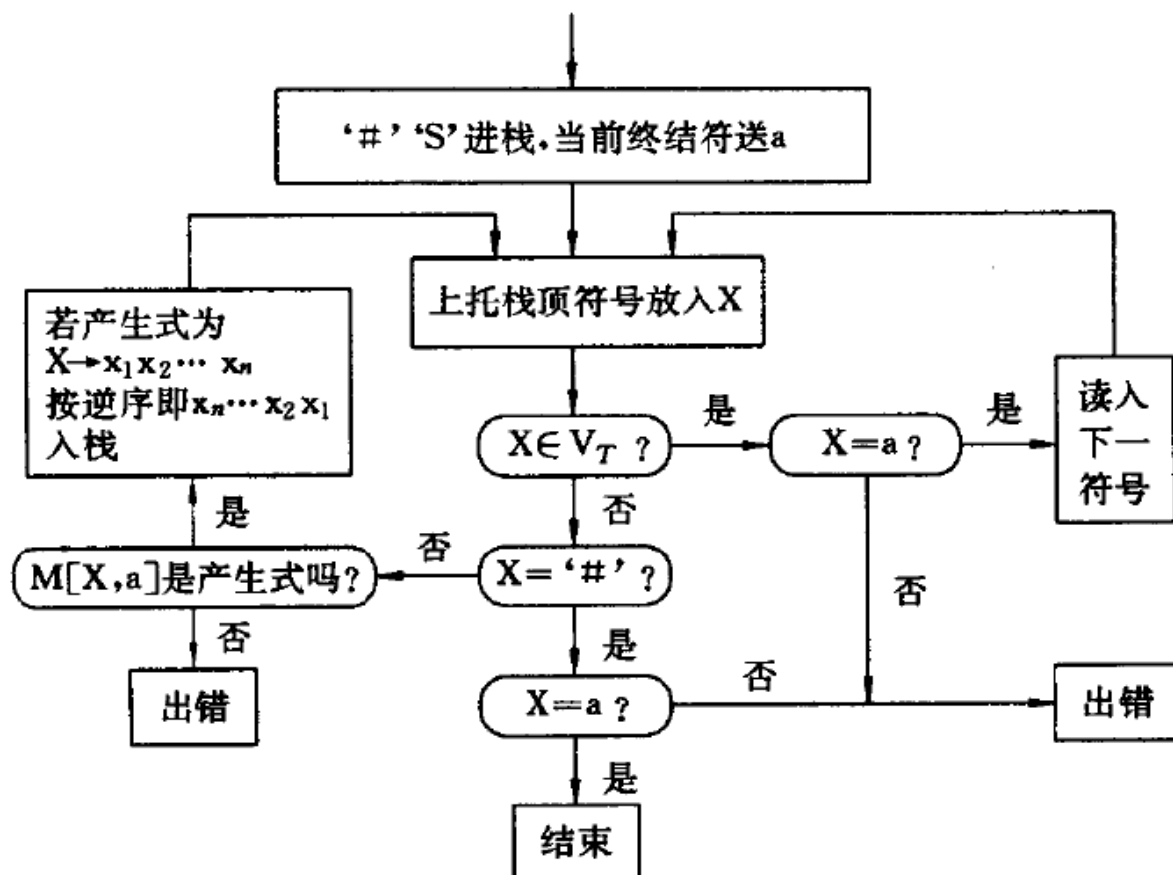
2.4 主要函数

表 2-1 主要函数及功能介绍

函数名	参数	返回值	用途
Advance()	no	no	推进一个字符
EntryStack()	Result: List(string)	no	将 result 进栈
error()	Msg:string	no	显示 msg 错误信息
queryTable()	(A:string, a:string)	Result:string	查分析表
analysis ()	no	no	LL(1)分析函数
main()	no	no	主函数

2.5 程序执行图

该函数结构相对清晰，这里直接采用已有框图。



3 程序测试

3.1 测试用例

测试语句采取相对简单的表达式，如下：

$i*(i+i)$

$i-i+i*i$

$i/i-(i)$

分别进行测试，并对测试结果进行测评。

3.2 测试结果

3 张分析结果图在“结果演示”文件夹中，这里只将第一张展示出，如下：

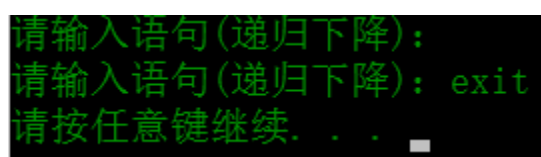
步骤	分析栈	剩余串
0	[#, 'E']	[*, '(', 'i', '+', 'i', ')', '#']
1	[#, 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
2	[#, 'E', 'T', 'F']	[*, '(', 'i', '+', 'i', ')', '#']
3	[#, 'E', 'T', 'F', 'i']	[*, '(', 'i', '+', 'i', ')', '#']
4	[#, 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
5	[#, 'E', 'T', 'F', 'M']	[*, '(', 'i', '+', 'i', ')', '#']
6	[#, 'E', 'T', 'F', '*']	[*, '(', 'i', '+', 'i', ')', '#']
7	[#, 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
8	[#, 'E', 'T', ')', 'E', '(']	[*, '(', 'i', '+', 'i', ')', '#']
9	[#, 'E', 'T', ')', 'E']	[*, '(', 'i', '+', 'i', ')', '#']
10	[#, 'E', 'T', ')', 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
11	[#, 'E', 'T', ')', 'E', 'T', 'F']	[*, '(', 'i', '+', 'i', ')', '#']
12	[#, 'E', 'T', ')', 'E', 'T', 'F', 'i']	[*, '(', 'i', '+', 'i', ')', '#']
13	[#, 'E', 'T', ')', 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
14	[#, 'E', 'T', ')', 'E']	[*, '(', 'i', '+', 'i', ')', '#']
15	[#, 'E', 'T', ')', 'E', 'T', 'A']	[*, '(', 'i', '+', 'i', ')', '#']
16	[#, 'E', 'T', ')', 'E', 'T', '+']	[*, '(', 'i', '+', 'i', ')', '#']
17	[#, 'E', 'T', ')', 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
18	[#, 'E', 'T', ')', 'E', 'T', 'F']	[*, '(', 'i', '+', 'i', ')', '#']
19	[#, 'E', 'T', ')', 'E', 'T', 'F', 'i']	[*, '(', 'i', '+', 'i', ')', '#']
20	[#, 'E', 'T', ')', 'E', 'T']	[*, '(', 'i', '+', 'i', ')', '#']
21	[#, 'E', 'T', ')', 'E']	[*, '(', 'i', '+', 'i', ')', '#']
22	[#, 'E', 'T', ')']	[*, '(', 'i', '+', 'i', ')', '#']
23	[#, 'E']	[*, '(', 'i', '+', 'i', ')', '#']
24	[#, 'E']	[*, '(', 'i', '+', 'i', ')', '#']
25	[#]	[*, '(', 'i', '+', 'i', ')', '#']
26	匹配成功	匹配成功

图 3-1 递归下降分析结果 1（部分图）

3.3 结果分析

采用以上两个测试用例分别进行测试，测试结果均显示正确，列表中第一列表示分析步骤，第二列表示中间过程中分析栈的情况，第三列表示剩余符号串。最后分析结束在表格最后一项添加内容“匹配成功”。

后经过多次测试验证了程序的正确性与健壮性，对于边沿性测试数据也有良好的表现，比如用户输入空串，则提醒用户继续输入。如果用户输入“exit”则退出程序。如下图所示：

A terminal window with a black background and green text. It shows three lines of text: '请输入语句(递归下降):', '请输入语句(递归下降): exit', and '请按任意键继续. . .'. A small white cursor is visible at the end of the third line.

```
请输入语句(递归下降):  
请输入语句(递归下降): exit  
请按任意键继续. . .
```

图 3-2 边沿数据处理图

附录

附录 A 程序代码

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from drawTable import drawTable # 绘表工具
global component                # 表格内容
global inputString              # 输入串
global stack                    # 分析栈
global top                      # 栈顶元素
global step                     # 步骤数
global current                  # 当前字符
global table                    # 分析表
table = {
    'E':{'i':['E_', 'T'], '(':['E_', 'T']},
    'E_':{'+':['E_', 'T', 'A'], '-':['E_', 'T', 'A'], ')':[], '#':[]}, #用[]代表空串
    'T':{'i':['T_', 'F'], '(':['T_', 'F']},
    'T_':{'+':[], '-':[], '*':['T_', 'F', 'M'], '/':['T_', 'F', 'M'],
    ')':[], '#':[],},
    'F':{'i':['i'], '(':['('), 'E', '(']},
    'A':{'+':['+'], '-':['-']},
    'M':{'*':['*'], '/':['/']}
}

# 报错
def error(msg="分析错误"):
    global component
    global step
    step = step + 1
    component.append([step, msg, ''])

    # exit(0)

# 查表
def queryTable(A, a):
    result = []
    try:
        result = table[A][a]
    except:
```

```

        error('查表出错')
        return False
    return result

# 推进
def advance():
    global stack
    global top
    global current
    global inputString
    stack.pop()
    top = stack[-1]
    current = inputString.pop(0)

# 进栈, 产生式右部逆序进栈
def entryStack(result):
    global stack
    global top
    stack.pop()
    stack.extend(result)
    top = stack[-1]

# 分析
def analysis():
    global stack
    global top
    global current
    global component
    global inputString
    global step
    Vt = ['i', '+', '-', '*', '/', '(', ')']
    Vn = ['E', 'E_', 'T', 'T_', 'F', 'A', 'M']
    stack = ['#', 'E']    # 初始化栈
    top = stack[-1]      # 栈顶元素
    inputString = list(inputString)    # 源串
    inputString.append('#')    # 末尾添加#
    current = inputString.pop(0)    # 当前字符
    flag = True    # 循环标志
    step = 0
    component = []
    while(flag):
        tempComponent = []    # 表格每一行的内容
        tempComponent.append(step)
        tempComponent.append(str(stack))

```

```

tempComponent.append(str(inputString))
component.append(tempComponent)
if (top in Vt):                                # 判断首字符是否是 vt
    if (top == current):
        advance()
    else:
        error(' 非法字符 ')
        break
elif(top == '#'):
    if(current == '#'):
        flag = False                            # 匹配成功, 可以退出
    else:
        error("非法结束")
        break
else:
    result = queryTable(top,current)
    if(False != result):
        entryStack(result)                      # 进栈
    else:                                        # 查表出错
        break
step = step + 1
if(flag == False):
    tempComponent = [step,'匹配成功','匹配成功']
    component.append(tempComponent)

def main():
    global inputString
    global component
    while(1):
        inputString = input("请输入语句 (LL1):")
        inputString = inputString.replace(' ','')
        if(inputString == "exit"):
            break
        analysis()
        header = 'LL1 分析'
        subHeader = ['步骤','分析栈','剩余串']#
        drawTable(header,subHeader,component,150,0) #表格长度 110 不居中

if __name__ == '__main__':
    main()

```

附录 B 测试用例

 $i*(i+i)$ $i-i+i*i$ $i/i-(i)$