编译原理

算符优先分析实验报告

Operator Grammar 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

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→E+T｜E-T｜T

T→T*F｜T/F｜F

F→(E)｜i

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、构造该算符优先文法的优先关系矩阵或优先函数；

3、递归下降分析 程序应能发现输入串出错；

4、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

该实验运用算符优先分析法的基本原理，针对以上文法描述语言，利用 Python 语言实现了的算符优先分析程序。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，并进行算符优先分析，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、算符优先分析

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 算符优先分析

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，循环结束的条件时分析栈的长度等于 2 并且当前字符为#。

While 循环体中，首先判断当前栈顶元素是否是 Vt，如果是则代表当前栈顶指针 index 指向的是 Vt，否则将栈顶指针减一。接着进行查表操作，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 = F[stack[tempIndex]] > G[current]
except:
        error('输入符号错误')
```

如果查表成功则寻找素短语的末尾，即从当前位置向栈底寻找，一直找到某个字符，该字符的算符优先级大于下一个栈底字符，则该字符与栈顶算符之间夹的便是素短语。

该过程的主要代码如下：

```
    while(len(stack) != 2 or current !='#'):    # 结束条件
        if (top in Vt):
            tempIndex = index
        else:
            tempIndex = index -1
        try:
            result = F[stack[tempIndex]] > G[current]
        except:
            error('输入符号错误')
            return '输入符号错误'
```

```
if (result ):   # 寻找素短语起始位置
    tempTop = stack[tempIndex]
    tempIndex = tempIndex -1
    if (stack[tempIndex] in Vt):
        if(F[stack[tempIndex]] < G[tempTop]):   #可归约
            stack.pop()
            index = index -1
            entryStack('N')
    else: #找到起始操作符
        stack.pop()
        stack.pop()
        stack.pop()
        index = index -3
        entryStack('N')
    step = step + 1
else:           # <= 的都应该进栈

    entryStack(current)
    advance()
    step = step + 1
    component.append(tempComponent)
    continue    #进入下次循环
```

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，调用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、两个函数优先级表、一个表示终结符 Vt 的列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，初始化栈顶为#。

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#。

函数优先级表为 F 和 G，他们的结构如下：

F = {'(':1, ')':7, 'i':7, '*':5, '/':5, '+':3, '-':3, '#':1, }

G = {'(':6, ')':1, 'i':6, '*':4, '/':4, '+':2, '-':2, '#':1, }

终结符列表 Vt 表示文法中出现的所有终结符：

Vt = ['i','+','-','*','/','(',')','#']

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	查优先关系
analysis ()	no	no	OG 分析函数
main()	no	no	主函数

2.5 程序执行图

该程序根据下图实现：

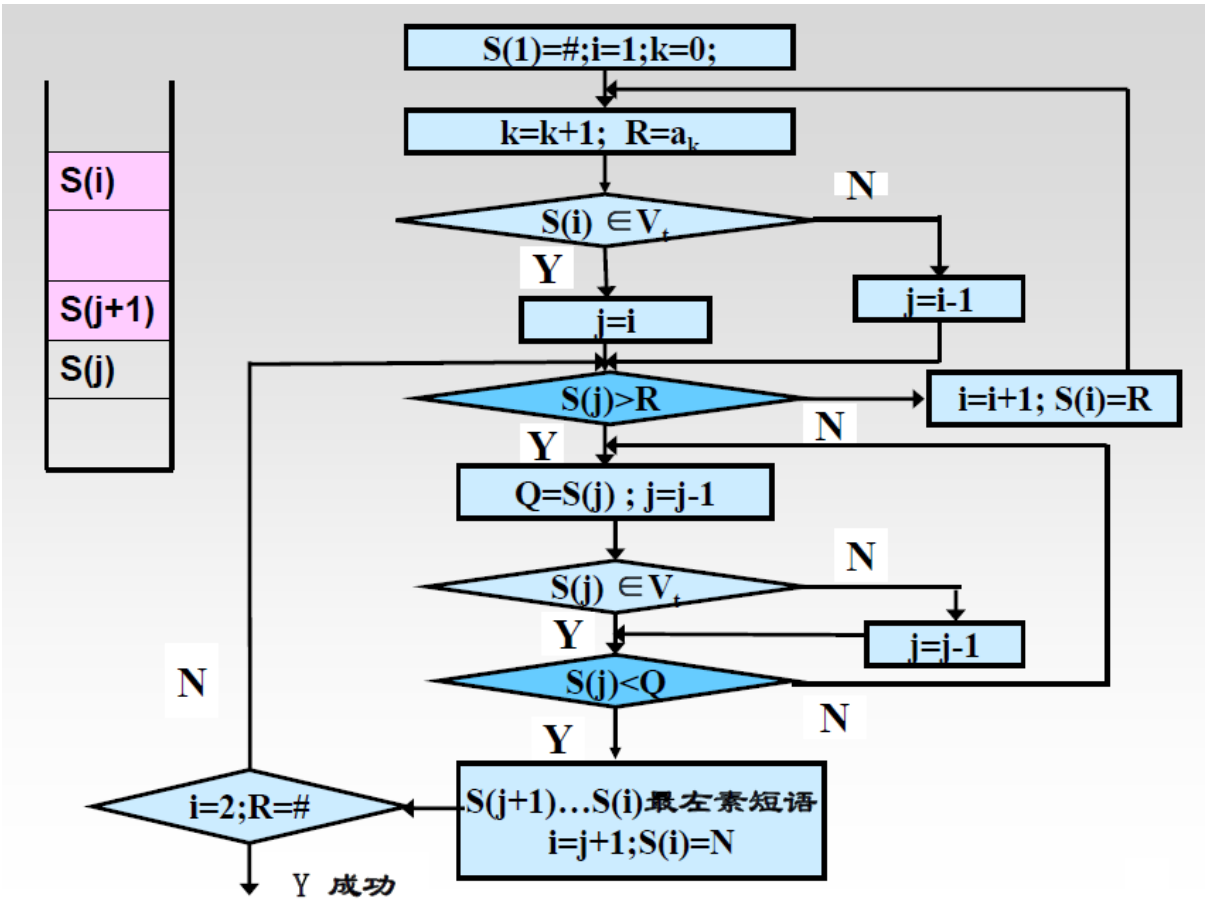

图 2-1 程序流程图

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：

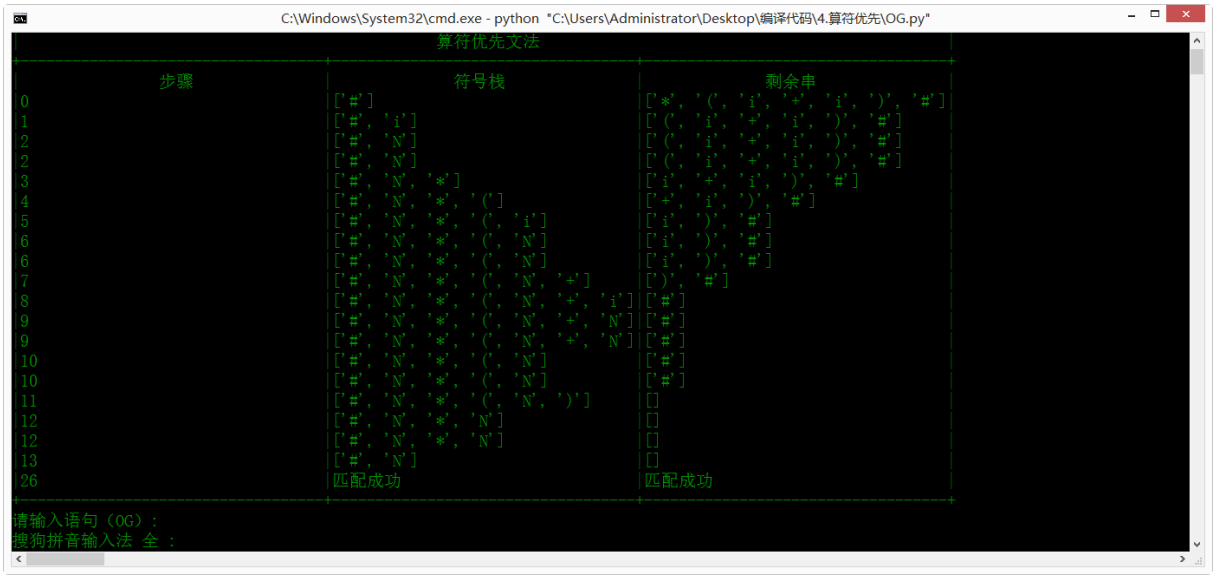

图 3-1 递归下降分析结果 1（部分图）

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：

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 -*-
# 算符优先
from drawTable import drawTable
global component # 表格内容
global inputString
global stack
global top
global current
global Vt
global F
global G
global index

def error(msg="分析错误，退出"):
    print(msg)
    exit(0)

def queryTable(A,a):
    result =[]
    try:
        result = table[A][a]
    except:
        error('查表出错')
    return result



def advance():  # 将输入串后移一位
    global current
    global inputString  #
    current = inputString.pop(0)

def entryStack(curr):   # 弹出栈顶，产生式右部逆序进栈
    global stack
    global top
    global index
```

```
    stack.append(curr)
    top = stack[-1]
    index = index + 1



def analysis():
    global stack
    global top
    global current
    global inputString  #
    global Vt
    global F
    global G
    global component # 表格内容
    global index
    F = {'(':1, ')':7, 'i':7, '*':5, '/':5, '+':3, '-':3, '#':1, }
    G = {'(':6, ')':1, 'i':6, '*':4, '/':4, '+':2, '-':2, '#':1, }
    Vt = ['i','+','-','*','/','(',')','#']
    stack =['#']    #初始化栈
    top = stack[-1]     #栈顶元素

    inputString = list(inputString)   # 源串
    inputString.append('#')
    current = inputString.pop(0)          # 当前字符
    index = 0
    tempIndex = 0           # 为找到最顶端的运算符而设
    tempTop = ' '           # 为了寻找素短语的头而设
    step = 0
    component = [] # 具体表格项
    while(len(stack) != 2 or current !='#'):  # 结束条件
        tempComponent = []
        tempComponent.append(step)
        tempComponent.append(str(stack))
        tempComponent.append(str(inputString))
        if (top in Vt):
            tempIndex = index
        else:
            tempIndex = index -1

        try:
            result = F[stack[tempIndex]] > G[current]
        except:
            error('输入符号错误')
```

```
        if (result ):  # 寻找素短语起始位置
            tempTop = stack[tempIndex]
            tempIndex = tempIndex -1
            if (stack[tempIndex] in Vt):

                if(F[stack[tempIndex]] < G[tempTop]):  #可归约

                    stack.pop()
                    index = index -1
                    entryStack('N')
                else: #找到起始操作符
                    stack.pop()
                    stack.pop()
                    stack.pop()
                    index = index -3
                    entryStack('N')
                step = step + 1
                component.append(tempComponent)

            else:           # <= 的都应该进栈

                entryStack(current)
                advance()
                step = step + 1
                component.append(tempComponent)
                continue   #进入下次循环

            tempComponent = []
            tempComponent.append(step)
            tempComponent.append(str(stack))
            tempComponent.append(str(inputString))
            component.append(tempComponent)
        step += step
        tempComponent = [step,'匹配成功','匹配成功']
        component.append(tempComponent)

def main():
    global inputString
    global component # 表格内容
    while(1):
        inputString = input("请输入语句（OG）:")
        inputString = inputString.replace(' ','')
        if(inputString == "exit"):
```

```
            break
        analysis()
        header = '算符优先文法'
        subHeader = [ '步骤','符号栈', '剩余串']#
        drawTable(header,subHeader,component,110,0)  # 最后一个参数为总长度


if __name__ == '__main__':
    main()
```

附录 B 测试用例

i*(i+i)

i-i+i*i

i/i-(i)