《编译原理》专题6设计

目标任务

实验项目

将词法分析程序设计原理与实现（专题1）和基于SLR(1)分析法的语法制导翻译及中间代码生成程序设计原理与实现（专题5）形成一个程序，即程序的输入为符号串源程序，输出为中间代码四元式序列。

设计要求

1. 词法分析的输入为符号串，词法分析的输出为二元式形式的中间文件；
2. 语法制导翻译读取二元式形式的中间文件并生成中间代码四元式形式的中间文件；
3. 设计两个测试用例（尽可能完备），并给出程序执行结果。

程序功能描述

1. 通过PyQt的UI展示分析过程
2. 自定义单词的匹配规则和名称, 具体包括标识符, 整数, +-*/运算符, 两种注释类型
3. 根据正则表达式识别输入串的单词
4. 将识别的单词和名称以二元式的方式输出到中间文件
5. 将标识符替换成保留字
6. 构造给定的文法的拓展文法
7. 识别拓展文法的活前缀的DFA
8. 构造活前缀DFA的SLR（1）分析表
9. SLR（1）语法分析
10. 将赋值语句翻译成四元式

数据结构

二元式类

```
class Binary(object):
    def __init__(self, code, token):
        """
        初始化
        :param code: 字符编码
        :param token: 字符的值
        """
        self.code = code
        self.token =
```

词法

```
#保留字表
reserved = ["void", "int", "float", "double", "else", "if", "do", "while",
"break", "for"]
# 其余符号
reserve = 5            #保留字
ID = 6                 #标识符
INT = 7                #整数
LT = 8                 # <
LE = 9                 # <=
EQ = 10                # =
EE = 11                # ==
NE = 12                # !=
GT = 13                # >
GE = 14                # >=
ADD = 15               # +
SUB = 16               # -
MUL = 17               # *
SC = 18                # ;
DIV = 19               # /
MCOMMENT = 20          # /*
COMMENT = 21           # //
LBRACKET = 22          # (
RBRACKET = 23          # )
LBRACE = 24            # {
RBRACE = 25            # }
PLUS = 26              # ++
MINUS = 27             # --
SHIFT_RIGHT = 28       # >>
SHIFT_LEFT = 29        # <<
AND = 30               # &&
OR = 31                # ||
NOT = 32               # !
```

PyQt类

```python
class Slr_Analyse(Ui_Dialog):
    def __init__(self):
        super(Slr_Analyse, self).__init__()
        # 将onclick函数关联到分析按钮上
        self.analyse.clicked.connect(self.analyse_onclick)
```

编译器界面

```python
class Ui_Dialog(QtWidgets.QDialog):
    def __init__(self):
        super(Ui_Dialog, self).__init__()
        self.init_ui()

    def init_ui(self):
        #窗体属性
        self.setObjectName("compiler")
        self.resize(379, 497)

        #定义控件
        self.input_line = QtWidgets.QLabel(self)
        self.input_line.setGeometry(QtCore.QRect(30, 70, 111, 16))
        self.input_line.setObjectName("input_line")
        self.title = QtWidgets.QLabel(self)
        self.title.setGeometry(QtCore.QRect(50, 10, 281, 51))
        self.title.setAlignment(Qt.AlignCenter)
        font = QtGui.QFont()
        font.setFamily("STHeiti")
        font.setPointSize(21)
        font.setBold(True)
        font.setWeight(75)
        self.title.setFont(font)
        self.title.setObjectName("title")
        self.input_text = QtWidgets.QLineEdit(self)
        self.input_text.setGeometry(QtCore.QRect(30, 100, 251, 71))
        self.input_text.setObjectName("input")
        self.output_line = QtWidgets.QLabel(self)
        self.output_line.setGeometry(QtCore.QRect(30, 240, 131, 41))
        self.output_line.setObjectName("output_line")
        self.analyse = QtWidgets.QPushButton(self)
        self.analyse.setGeometry(QtCore.QRect(100, 190, 113, 32))
        self.analyse.setObjectName("analyse")
        self.output = QtWidgets.QTextEdit(self)
        self.output.setGeometry(QtCore.QRect(30, 290, 251, 71))
        self.output.setObjectName("output")
        self.error_line = QtWidgets.QLabel(self)
        self.error_line.setGeometry(QtCore.QRect(30, 370, 60, 16))
        self.error_line.setObjectName("error_text")
        self.error_text = QtWidgets.QLineEdit(self)
        self.error_text.setGeometry(QtCore.QRect(30, 400, 251, 71))
        self.error_text.setObjectName("error_line")

        self.setWindowTitle("编译器系统")
        self.title.setText("简易编译器中间代码生成系统")
        self.input_line.setText("请输入赋值语句")
        self.analyse.setText("分析")
        self.output_line.setText("生成的四元式序列")
        self.error_line.setText("错误信息")
```

四元式类

```python
class Quater (object):
    def __init__(self, op, arg1, arg2, result):
        """
        初始化
        :param op: 运算符
        :param arg1: 第一个操作数
        :param arg2: 第二个操作数
        :param result: 运算结果
        """
        self.op = op
        self.arg1 = arg1
        self.arg2 = arg2
        self.result = result
```

存储规则左部和右部字符的类

```python
class Rule(object):
    def __init__(self):
        self.left = ""
        self.right = []
```

项目集族类

```python
class Proj(Rule):
    def __init__(self):
        Rule.__init__(self)
        # '.'在产生式右部的位置
        self.part = 0
```

存储状态转换情况的类

```
class StatusTrans(object):
    def __init__(self, status_init, status_trans, x):
        """
        :param status_init: 初始状态
        :param status_trans: 接收x后转换到的状态
        :param x: 初始状态接收非终结符x
        """
        self.status_init = status_init
        self.status_trans = status_trans
        self.x = x
```

栈

```
class Stack(object):
    def __init__(self):
        """
        栈初始化
        """
        self.items = []

    def push(self, item):
        """
        入栈
        :param item: 入栈的字符
        :return: None
        """
        self.items.append(item)

    def pop(self):
        """
        出栈
        :return: 出栈的字符
        """
        return self.items.pop()

    def get_stackelem(self):
        """
        获得栈中的全部元素
        :return: 含有栈中全部元素的列表
        """
        return self.items

    def get_top(self):
        """
        获得栈顶元素
        :return: 栈顶元素
        """
        return self.items[-1]
```

文法

```
# 终结符
VT = []
# 非终结符
VN = []
# 规则集合
Rules = []
# first集
First = []
# follow集
Follow = []
# 存储规则左部和右部的集合
rule_list = []
# SLR(1)分析表
SLR1 = []
# 项目集族
proj = []
# 所有状态转换集合
status_trans = []
```

程序结构描述

词法分析

```
def scanner(line):
    """
    词法分析程序
    :param line: 输入的字符串
    :return: 二元式数组scanner_output;错误类型error
    """
    global error
    error = -1
    scanner_output = []
    line = line.replace('\n', '')
    line = line.replace(';', '')
    tokens = line.split(' ')
    j = 0
    while j != len(tokens):
        # 判断是否为标识符
        if tokens[j][0].isalpha() and tokens[j].isalnum():
            #判断是否为保留字
            if tokens[j] in reserved:
                reserved_binary = Binary(reserve, tokens[j])
                scanner_output.append(reserved_binary)
            else:
                identifier = Binary(ID, tokens[j])
                scanner_output.append(identifier)
            j += 1
        # 判断是否为整数
        elif tokens[j].isdigit():
            integer = Binary(INT, tokens[j])
            scanner_output.append(integer)
            j += 1
        elif tokens[j] == '>':
            gt = Binary(GT, tokens[j])
            scanner_output.append(gt)
            j += 1
        elif tokens[j] == '>=':
            ge = Binary(GE, tokens[j])
            scanner_output.append(ge)
            j += 1
        elif tokens[j] == '<':
            lt = Binary(LT, tokens[j])
            scanner_output.append(lt)
            j += 1
        elif tokens[j] == '<=':
            le = Binary(LE, tokens[j])
            scanner_output.append(le)
            j += 1
        elif tokens[j] == '=':
            eq = Binary(EQ, tokens[j])
            scanner_output.append(eq)
            j += 1
        elif tokens[j] == '==':
            ee = Binary(EE, tokens[j])
            scanner_output.append(ee)
            j += 1
        elif tokens[j] == '+':
            add = Binary(ADD, tokens[j])
            scanner_output.append(add)
            j += 1
        elif tokens[j] == '++':
            plus = Binary(PLUS, tokens[j])
            scanner_output.append(plus)
            j += 1
        elif tokens[j] == '-':
            sub = Binary(SUB, tokens[j])
            scanner_output.append(sub)
            j += 1
        elif tokens[j] == '--':
            minus = Binary(MINUS, tokens[j])
            scanner_output.append(minus)
            j += 1
        elif tokens[j] == '*':
            mul = Binary(MUL, tokens[j])
            scanner_output.append(mul)
            j += 1
        elif tokens[j] == '/':
            if tokens[j+1] == '0':
                print('除数不能为0\n')
                error = 1
                break
            div = Binary(DIV, tokens[j])
            scanner_output.append(div)
            j += 1
        elif tokens[j].startswith('/'):
            if tokens[j][1] == '/':
                comment = Binary(COMMENT, tokens[j])
                scanner_output.append(comment)
                j += 1
            elif tokens[j][1] == '*':
                flag = 0
                for k in range(i+1, len(lines)):
                    if '*/' in lines[k]:
                        flag = 1
                        i = k
                        break
                if flag:
                    mcomment = Binary(MCOMMENT, tokens[j])
                    scanner_output.append(mcomment)
                    j += 1
                else:
                    print("不完整的注释\n")
                    error = 2
                    break
            else:
                print('注释缺少/\n')
                error = 3
                break
        elif tokens[j] == '{':
            lbrace = Binary(LBRACE, tokens[j])
            scanner_output.append(lbrace)
            j += 1
        elif tokens[j] == '}':
            rbrace = Binary(RBRACE, tokens[j])
            scanner_output.append(rbrace)
            j += 1
        elif tokens[j].startswith('('):
            bracket_complete = 0
            lbracket = Binary(LBRACKET, tokens[j])
            scanner_output.append(lbracket)
            j += 1
            for n in range(j+1, len(tokens)):
                if ')' in tokens[n]:
                    bracket_complete = 1
                    break
            if not bracket_complete:
                print('找不到匹配的右括号\n')
                error = 4
                break
        elif tokens[j] == ')':
            rbracket = Binary(RBRACKET, tokens[j])
            scanner_output.append(rbracket)
            j += 1
        elif tokens[j] == '&&':
            and_binary = Binary(AND, tokens[j])
            scanner_output.append(and_binary)
            j += 1
        elif tokens[j] == '||':
            or_binary = Binary(OR, tokens[j])
            scanner_output.append(or_binary)
            j += 1
        elif tokens[j] == '!':
            not_binary = Binary(NOT, tokens[j])
            scanner_output.append(not_binary)
            j += 1
        elif tokens[j] == '!=':
            ne = Binary(NE, tokens[j])
            scanner_output.append(ne)
            j += 1
        else:
            print('%s无法被识别\n' % tokens[j])
            j += 1
    return scanner_output, error
```

PyQt前端渲染

```
class Slr_Analyse(Ui_Dialog):
    def __init__(self):
        super(Slr_Analyse, self).__init__()
        # 将onclick函数关联到分析按钮上
        self.analyse.clicked.connect(self.analyse_onclick)

    def analyse_onclick(self):
        self.output.setText("")
        self.error_text.setText("")
        #得到用户输入的赋值语句
        expression = self.input_text.text()
        output_binary, iserror = scanner_module.scanner(expression)
        if iserror == -1:
            output_list = slr1(output_binary)
            for i in range(len(output_list)):
                self.output.append(
                    '(' + output_list[i].op + ',' + output_list[i].arg1 +
',' + output_list[i].arg2 + ',' +
                    output_list[i].result + ')')
        else:
            if iserror == 1:
                self.error_text.setText("除数为0!")
            elif iserror == 2:
                self.error_text.setText("不完整的多行注释!")
            elif iserror == 3:
                self.error_text.setText("注释缺少'/'!")
            elif iserror == 4:
                self.error_text.setText("找不到匹配的右括号!")
```

PyQt界面

```
class Ui_Dialog(QtWidgets.QDialog):
    def __init__(self):
        super(Ui_Dialog, self).__init__()
        self.init_ui()

    def init_ui(self):
        #窗体属性
        self.setObjectName("compiler")
        self.resize(379, 497)

        #定义控件
        self.input_line = QtWidgets.QLabel(self)
        self.input_line.setGeometry(QtCore.QRect(30, 70, 111, 16))
        self.input_line.setObjectName("input_line")
        self.title = QtWidgets.QLabel(self)
        self.title.setGeometry(QtCore.QRect(50, 10, 281, 51))
        self.title.setAlignment(Qt.AlignCenter)
        font = QtGui.QFont()
        font.setFamily("STHeiti")
        font.setPointSize(21)
        font.setBold(True)
        font.setWeight(75)
        self.title.setFont(font)
        self.title.setObjectName("title")
        self.input_text = QtWidgets.QLineEdit(self)
        self.input_text.setGeometry(QtCore.QRect(30, 100, 251, 71))
        self.input_text.setObjectName("input")
        self.output_line = QtWidgets.QLabel(self)
        self.output_line.setGeometry(QtCore.QRect(30, 240, 131, 41))
        self.output_line.setObjectName("output_line")
        self.analyse = QtWidgets.QPushButton(self)
        self.analyse.setGeometry(QtCore.QRect(100, 190, 113, 32))
        self.analyse.setObjectName("analyse")
        self.output = QtWidgets.QTextEdit(self)
        self.output.setGeometry(QtCore.QRect(30, 290, 251, 71))
        self.output.setObjectName("output")
        self.error_line = QtWidgets.QLabel(self)
        self.error_line.setGeometry(QtCore.QRect(30, 370, 60, 16))
        self.error_line.setObjectName("error_text")
        self.error_text = QtWidgets.QLineEdit(self)
        self.error_text.setGeometry(QtCore.QRect(30, 400, 251, 71))
        self.error_text.setObjectName("error_line")

        self.setWindowTitle("编译器系统")
        self.title.setText("简易编译器中间代码生成系统")
        self.input_line.setText("请输入赋值语句")
        self.analyse.setText("分析")
        self.output_line.setText("生成的四元式序列")
        self.error_line.setText("错误信息")


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ui = Ui_Dialog()
    ui.show()
    sys.exit(app.exec_())
```

SLR(1)分析过程

```
def slr1():
    """
    SLR(1)分析过程
    :return: 输入的字符串是否为SLR(1)文法
    """
    current = 0
    # 创建状态栈
    status_stack = Stack()
    status_stack.push(0)
    # 创建符号栈
    sym_stack = Stack()
    sym_stack.push('#')
    with open('src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readline()
        src = src.replace('\n', '')
    with open('output.txt', 'w', encoding='utf-8') as output_file:
        output_file.write('对%s进行SLR(1)分析法的分析过程如下\n' % src)
        output_file.write('状态栈\t\t\t符号栈\t\t输入串
\t\tACTION\t\tGOTO\t\n')
        while True:
            s = status_stack.get_stackelem()
            str_sta = [str(i) for i in s]
            str_sta = " ".join(str_sta)
            output_file.write("%-18s" % str_sta)
            sy = sym_stack.get_stackelem()
            output_file.write('\t')
            str_sym = [str(j) for j in sy]
            str_sym = "".join(str_sym)
            output_file.write("%-10s" % str_sym)
            output_file.write('\t')
            sta_top = status_stack.get_top()
            if SLR1[sta_top][VT.index(src[current])] != -1 and 'S' in
SLR1[sta_top][VT.index(src[current])]:
                next_status = int(SLR1[sta_top][VT.index(src[current])][1:])
                status_stack.push(next_status)
                sym_stack.push(src[current])
                output_file.write('%10s\t\t' % src[current:])
                output_file.write('%s\t\t' % SLR1[sta_top]
[VT.index(src[current])])
                output_file.write('\n')
                current += 1
            elif SLR1[sta_top][VT.index(src[current])] != -1 and 'R' in
SLR1[sta_top][VT.index(src[current])]:
                output_file.write('%10s\t\t' % src[current:])
                output_file.write('%s\t\t' % SLR1[sta_top]
[VT.index(src[current])])
                rulepos = int(SLR1[sta_top][VT.index(src[current])][1:])
                count = 0
                while count != len(rule_list[rulepos].right):
                    sym_stack.pop()
                    status_stack.pop()
                    count += 1
                sym_stack.push(rule_list[rulepos].left)
                sym_top = sym_stack.get_top()
                while True:
                    sta_top = status_stack.get_top()
                    if SLR1[sta_top][VN.index(sym_top) + len(VT) - 1] != -1:
                        status_stack.push(SLR1[sta_top][VN.index(sym_top) +
len(VT) - 1])
                        output_file.write('%d\n' % SLR1[sta_top]
[VN.index(sym_top) + len(VT) - 1])
                        break
                    else:
```

```
                    status_stack.pop()
            elif SLR1[sta_top][VT.index(src[current])] == 'acc':
                output_file.write('%10s\t\t' % src[current:])
                output_file.write('%s\n' % SLR1[sta_top]
[VT.index(src[current])])
                output_file.write('\n')
                output_file.write('%s为合法字符串\n' % src)
                break
            else:
                output_file.write('\n')
                output_file.write('%s为不合法字符串\n' % src)
                break
```

输出SLR(1)分析表

```
def print_slr1():
    """
    打印slr1分析表，将其写入slr1.txt中
    :return:
    """
    with open('slr1.txt', 'w', encoding='utf-8') as slr1_file:
        slr1_file.write('SLR(1)分析表如下\n')
        slr1_file.write('状态\t\tACTION\t\t\t\t\tGOTO\t\t\n')
        slr1_file.write('\t')
        for i in range(len(VT)):
            slr1_file.write('%s\t' % VT[i])
        for j in range(1, len(VN)):
            slr1_file.write('%s\t' % VN[j])
        slr1_file.write('\n')
        for m in range(len(SLR1)):
            slr1_file.write('%d\t' % m)
            for n in range(len(SLR1[m])):
                slr1_file.write('%s\t' % SLR1[m][n])
            slr1_file.write('\n')
```

建立SLR(1)分析表

```
def create_slr1():
    """
    建立SLR1分析表
    :return: SLR1分析表
    """
    for i in range(len(status_trans)):
        if status_trans[i].x in VT:
            SLR1[status_trans[i].status_init][VT.index(status_trans[i].x)] =
'S' + str(status_trans[i].status_trans)
        else:
            SLR1[status_trans[i].status_init][VN.index(status_trans[i].x) +
len(VT) - 1] = status_trans[i].status_trans
    for j in range(len(proj)):
        for k in range(len(proj[j])):
            p = proj[j][k].part
            if p == len(proj[j][k].right) and proj[j][k].left == VN[0]:
                SLR1[j][VT.index('#')] = 'acc'
            elif p == len(proj[j][k].right):
                rule_pos = get_rule_pos(proj[j][k])
                for q in range(len(Follow[VN.index(proj[j][k].left)])):
                    if Follow[VN.index(proj[j][k].left)][q] in VT:
                        SLR1[j][VT.index(Follow[VN.index(proj[j][k].left)]
[q])] = 'R' + str(rule_pos)
```

输出项目集族和与活前缀DFA

```
def print_proj():
    """
    打印项目集族与识别活前缀的DFA
    :return: None
    """
    with open('proj.txt', 'w', encoding='utf-8') as proj_file:
        proj_file.write('项目集族如下\n')
        for i in range(len(proj)):
            proj_file.write('I%d\n' % i)
            for j in range(len(proj[i])):
                p = proj[i][j].part
                s = proj[i][j].right[:]
                s.insert(p, '.')
                proj_file.write('%s -> %s\n' % (proj[i][j].left,
"".join(s)))
        proj_file.write('\n')
        proj_file.write('识别活前缀的DFA如下\n')
        proj_file.write('初始状态\t接收终结符\t到达的状态\t\n')
        for k in range(len(status_trans)):
            proj_file.write('I%d\t\t%s\t\tI%d\t\t\n' %
(status_trans[k].status_init, status_trans[k].x,
status_trans[k].status_trans))
            proj_file.write('\n')
```

获取rule_list中产生式的位置

```python
def get_rule_pos(temp):
    """
    得到某条产生式在规则集rule_list中的位置
    :param temp: 待查询的规则
    :return: 该规则在rule_list中的位置
    """
    for i in range(len(rule_list)):
        if rule_list[i].left == temp.left and rule_list[i].right ==
temp.right:
            return i
```

项目集族的读操作

```python
def go(n, x):
    """
    项目集族的读操作
    :param n: 第n个状态
    :param x: 第n个状态接收非终结符x
    :return: 状态n接收x后转换到的新状态
    """
    c = []
    for i in range(len(proj[n])):
        # 存储一个项目集的全部项目
        p = proj[n][i].part
        if p != len(proj[n][i].right) and proj[n][i].right[p] == x:
            new_proj = Proj()
            new_proj.left = proj[n][i].left
            new_proj.right = proj[n][i].right
            new_proj.part = proj[n][i].part + 1
            c.append(new_proj)
    enclosure(c)
    if is_contained(c) == -1:
        proj.append(c)
        new_status = StatusTrans(n, len(proj) - 1, x)
        status_trans.append(new_status)
    else:
        status_pos = is_contained(c)
        new_status = StatusTrans(n, status_pos, x)
        status_trans.append(new_status)
```

生成项目集族

```python
def gen_proj():
    # 生成c0
    c_0 = Proj()
    c_0.left = rule_list[0].left
    c_0.right = rule_list[0].right
    c_0.part = 0
    proj.append([c_0, ])
    enclosure(proj[0])
    l = len(proj)
    i = 0
    # 读操作
    while i != l:
        for j in range(len(proj[i])):
            p = proj[i][j].part
            if p != len(proj[i][j].right):
                if j > 0 and proj[i][j-1].part != len(proj[i][j-1].right):
                    if proj[i][j].right[p] == proj[i][j - 1].right[proj[i][j
- 1].part]:
                        continue
                go(i, proj[i][j].right[p])
                l = len(proj)
        i = i + 1
```

项目集族的闭包操作

```
def enclosure(c):
    i = 0
    l = len(c)
    while i != l:
        pos = c[i].part
        if pos != len(c[i].right) and c[i].right[pos] in VN:
            if i > 0:
                if c[i].right[pos] == c[i - 1].right[c[i-1].part]:
                    i += 1
                    continue
            for j in range(len(rule_list)):
                if rule_list[j].left == c[i].right[pos]:
                    temp = Proj()
                    temp.left = rule_list[j].left
                    temp.right = rule_list[j].right
                    temp.part = 0
                    c.append(temp)
        l = len(c)
        i += 1
```

创建First集

```
def create_first_set(ch):
    for i in range(0, len(rule_list)):
        if ch == rule_list[i].left:
            # 如果规则右部的第一个字符为终结符或者空串, 则将他们加入ch的first集
            if rule_list[i].right[0] in VT or rule_list[i].right[0] == 'ε':
                if rule_list[i].right[0] not in First[VN.index(ch)]:
                    First[VN.index(ch)].append(rule_list[i].right[0])
            else:
                a = VN.index(rule_list[i].right[0])
                # 如果右部第一个字符为非终结符，且该字符的First集还不存在，则递
归的调用该函数求右部第一个字符的first集
                if not First[a] and rule_list[i].right[0] !=
rule_list[i].left:
                    create_first_set(rule_list[i].right[0])
                # 将右部第一个字符的first集去掉空串加入到ch的first集中
                if 'ε' in First[VN.index(rule_list[i].right[0])]:
                    temp = First[VN.index(rule_list[i].right[0])][:]
                    First[VN.index(ch)] = temp.remove('ε')
                else:
                    for c in First[VN.index(rule_list[i].right[0])]:
                        if c not in First[VN.index(ch)]:
                            First[VN.index(ch)].extend(c)
```

创建Follow集

```
def create_first_set(ch):
    for i in range(0, len(rule_list)):
        if ch == rule_list[i].left:
            # 如果规则右部的第一个字符为终结符或者空串，则将他们加入ch的first集
            if rule_list[i].right[0] in VT or rule_list[i].right[0] == 'ε':
                if rule_list[i].right[0] not in First[VN.index(ch)]:
                    First[VN.index(ch)].append(rule_list[i].right[0])
            else:
                a = VN.index(rule_list[i].right[0])
                # 如果右部第一个字符为非终结符，且该字符的First集还不存在，则递
归的调用该函数求右部第一个字符的first集
                if not First[a] and rule_list[i].right[0] !=
rule_list[i].left:
                    create_first_set(rule_list[i].right[0])
                # 将右部第一个字符的first集去掉空串加入到ch的first集中
                if 'ε' in First[VN.index(rule_list[i].right[0])]:
                    temp = First[VN.index(rule_list[i].right[0])][:]
                    First[VN.index(ch)] = temp.remove('ε')
                else:
                    for c in First[VN.index(rule_list[i].right[0])]:
                        if c not in First[VN.index(ch)]:
                            First[VN.index(ch)].extend(c)
```

测试

测试输入

```
i=i*(i+i)#
```

SLR分析表输出

```
SLR(1)分析表如下
状态      ACTION                        GOTO
    =   +   -   *   /   (   )   i   #   A   V   E   T   F
0   -1  -1  -1  -1  -1  -1  -1  S3  -1  1   2   -1  -1  -1
1   -1  -1  -1  -1  -1  -1  -1  -1  acc -1  -1  -1  -1  -1
2   S4  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
3   R10 -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
4   -1  -1  -1  -1  -1  S8  -1  S9  -1  -1  -1  5   6   7
5   -1  S10 S11 -1  -1  -1  -1  -1  R1  -1  -1  -1  -1  -1
6   -1  R4  R4  S12 S13 -1  R4  -1  R4  -1  -1  -1  -1  -1
7   -1  R7  R7  R7  R7  -1  R7  -1  R7  -1  -1  -1  -1  -1
8   -1  -1  -1  -1  -1  S8  -1  S9  -1  -1  -1  14  6   7
9   -1  R9  R9  R9  R9  -1  R9  -1  R9  -1  -1  -1  -1  -1
10  -1  -1  -1  -1  -1  S8  -1  S9  -1  -1  -1  -1  15  7
11  -1  -1  -1  -1  -1  S8  -1  S9  -1  -1  -1  -1  16  7
12  -1  -1  -1  -1  -1  S8  -1  S9  -1  -1  -1  -1  -1  17
13  -1  -1  -1  -1  -1  S8  -1  S9  -1  -1  -1  -1  -1  18
14  -1  S10 S11 -1  -1  -1  S19 -1  -1  -1  -1  -1  -1  -1
15  -1  R2  R2  S12 S13 -1  R2  -1  R2  -1  -1  -1  -1  -1
16  -1  R3  R3  S12 S13 -1  R3  -1  R3  -1  -1  -1  -1  -1
17  -1  R5  R5  R5  R5  -1  R5  -1  R5  -1  -1  -1  -1  -1
18  -1  R6  R6  R6  R6  -1  R6  -1  R6  -1  -1  -1  -1  -1
19  -1  R8  R8  R8  R8  -1  R8  -1  R8  -1  -1  -1  -1  -1
```

分析过程输出

```
对i=i*(i+i)#进行SLR(1)分析法的分析过程如下
状态栈         符号栈     输入串     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
0 1                 #A                  #      acc

i=i*(i+i)#为合法字符串
```

源代码

```python
# -*- coding:utf-8 -*-
# SLR(1)分析法


# 存储规则左部和右部字符的类
class Rule(object):
    def __init__(self):
        self.left = ""
        self.right = []


#项目集族类
class Proj(Rule):
    def __init__(self):
        Rule.__init__(self)
        # '.'在产生式右部的位置
        self.part = 0


# 存储状态转换情况的类
class StatusTrans(object):
    def __init__(self, status_init, status_trans, x):
        """
        :param status_init: 初始状态
        :param status_trans: 接收x后转换到的状态
        :param x: 初始状态接收非终结符x
        """
        self.status_init = status_init
        self.status_trans = status_trans
        self.x = x


# 栈
```

```python
class Stack(object):
    def __init__(self):
        """
        栈初始化
        """
        self.items = []

    def push(self, item):
        """
        入栈
        :param item: 入栈的字符
        :return: None
        """
        self.items.append(item)

    def pop(self):
        """
        出栈
        :return: 出栈的字符
        """
        return self.items.pop()

    def get_stackelem(self):
        """
        获得栈中的全部元素
        :return: 含有栈中全部元素的列表
        """
        return self.items

    def get_top(self):
        """
        获得栈顶元素
        :return: 栈顶元素
        """
        return self.items[-1]

# 终结符
VT = []
# 非终结符
VN = []
# 规则集合
Rules = []
# first集
First = []
# follow集
Follow = []
# 存储规则左部和右部的集合
rule_list = []
# SLR(1)分析表
```

```
SLR1 = []
# 项目集族
proj = []
# 所有状态转换集合
status_trans = []


# 从输入的规则中找出终结符和非终结符
def identify_vt_and_vn():
    for i in range(0, len(rule_list)):
        #把规则左部加入到非终结符集合中
        if rule_list[i].left not in VN:
            VN.append(rule_list[i].left)
        #将规则右部的终结符和非终结符加入到相应的集合
        for j in range(len(rule_list[i].right)):
            if rule_list[i].right[j].isupper():
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
            elif rule_list[i].right[j] != 'ε' and "'" not in
rule_list[i].right[j]:
                if rule_list[i].right[j] not in VT:
                    VT.append(rule_list[i].right[j])
            elif "'" in rule_list[i].right[j]:
                if rule_list[i].right[j] not in VN:
                    VN.append(rule_list[i].right[j])
    VT.append('#')


# 得到每个规则的左部和右部
def create_rule_list():
    for i in range(0, len(Rules)):
        # 去掉空格
        Rules[i] = Rules[i].replace(' ', '')
        rule = Rule()
        rule_list.append(rule)
    for j in range(0, len(Rules)):
        arrow_pos = Rules[j].find('-')
        rule_list[j].left = Rules[j][0:arrow_pos]
        #将规则右部转换成列表
        rule_list[j].right = list(Rules[j][arrow_pos + 2:])
        while "'" in rule_list[j].right:
            pos = rule_list[j].right.index("'")
            new_sym = "".join(rule_list[j].right[pos - 1: pos + 1])
            del rule_list[j].right[pos]
            del rule_list[j].right[pos - 1]
            if new_sym not in rule_list[j].right:
                rule_list[j].right.append(new_sym)
```

```
# 创建first集
def create_first_set(ch):
    for i in range(0, len(rule_list)):
        if ch == rule_list[i].left:
            # 如果规则右部的第一个字符为终结符或者空串，则将他们加入ch的first集
            if rule_list[i].right[0] in VT or rule_list[i].right[0] == 'ε':
                if rule_list[i].right[0] not in First[VN.index(ch)]:
                    First[VN.index(ch)].append(rule_list[i].right[0])
            else:
                a = VN.index(rule_list[i].right[0])
                # 如果右部第一个字符为非终结符，且该字符的First集还不存在，则递
归的调用该函数求右部第一个字符的first集
                if not First[a] and rule_list[i].right[0] !=
rule_list[i].left:
                    create_first_set(rule_list[i].right[0])
                # 将右部第一个字符的first集去掉空串加入到ch的first集中
                if 'ε' in First[VN.index(rule_list[i].right[0])]:
                    temp = First[VN.index(rule_list[i].right[0])][:]
                    First[VN.index(ch)] = temp.remove('ε')
                else:
                    for c in First[VN.index(rule_list[i].right[0])]:
                        if c not in First[VN.index(ch)]:
                            First[VN.index(ch)].extend(c)


# 创建follow集
def create_follow_set(ch):
    if '#' not in Follow[0]:
        Follow[0].append('#')
    for i in range(len(rule_list)):
        if ch in rule_list[i].right:
            ch_pos = rule_list[i].right.index(ch)
            # 如果ch为最后一个字符，则将产生式左部字符的Follow集加入ch的Follow
集
            if ch_pos == len(rule_list[i].right) - 1:
                for c in Follow[VN.index(rule_list[i].left)]:
                    if c not in Follow[VN.index(ch)]:
                        Follow[VN.index(ch)].extend(c)
            # 如果ch后的一个字符为终结符，则将该终结符加入ch的Follow集
            elif rule_list[i].right[ch_pos + 1] in VT:
                if rule_list[i].right[ch_pos + 1] not in
Follow[VN.index(ch)]:
                    Follow[VN.index(ch)].extend(rule_list[i].right[ch_pos +
1])
            # 如果ch后的一个字符的first集有空串，且该字符为最后一个元素，则将左
部的Follow集加入ch的follow集
            elif ch_pos + 1 == len(rule_list[i].right) - 1 and 'ε' in
First[VN.index(rule_list[i].right[ch_pos + 1])]:
                for t in Follow[VN.index(rule_list[i].left)]:
                    if t not in Follow[VN.index(ch)]:
                        Follow[VN.index(ch)].extend(t)
            # 如果ch后的一个字符为非终结符，则将该非终结符的first集去掉空串加入
ch的Follow集
            if ch_pos != len(rule_list[i].right) - 1 and
rule_list[i].right[ch_pos + 1] in VN:
                if 'ε' in First[VN.index(rule_list[i].right[ch_pos + 1])]:
                    temp = First[VN.index(rule_list[i].right[ch_pos + 1])]
[:]
                    temp.remove('ε')
                    for char in temp:
                        if char not in Follow[VN.index(ch)]:
                            Follow[VN.index(ch)].append(char)
                else:
                    for e in First[VN.index(rule_list[i].right[ch_pos +
1])]:
                        if e not in Follow[VN.index(ch)]:
                            Follow[VN.index(ch)].extend(e)


def is_contained(c):
    """
    判断某个项目集是否已经存在与总的项目集中
    :param c: 待判断的项目集列表
    :return: 如果项目集已经存在，返回项目集的位置，如果不存在，返回-1
    """

    for i in range(len(proj)):
        count = 0
        for j in range(len(proj[i])):
            for k in range(len(c)):
                if c[k].left == proj[i][j].left and c[k].right == proj[i]
[j].right and c[k].part == proj[i][j].part:
                    count += 1
                    break
        if count == len(proj[i]):
            return i
    return -1


# 项目集族的闭包操作
def enclosure(c):
    i = 0
    l = len(c)
    while i != l:
        pos = c[i].part
        if pos != len(c[i].right) and c[i].right[pos] in VN:
            if i > 0:
                if c[i].right[pos] == c[i - 1].right[c[i-1].part]:
```

```python
                    i += 1
                    continue
            for j in range(len(rule_list)):
                if rule_list[j].left == c[i].right[pos]:
                    temp = Proj()
                    temp.left = rule_list[j].left
                    temp.right = rule_list[j].right
                    temp.part = 0
                    c.append(temp)
        l = len(c)
        i += 1


def go(n, x):
    """
    项目集族的读操作
    :param n: 第n个状态
    :param x: 第n个状态接收非终结符x
    :return: 状态n接收x后转换到的新状态
    """
    c = []
    for i in range(len(proj[n])):
        # 存储一个项目集的全部项目
        p = proj[n][i].part
        if p != len(proj[n][i].right) and proj[n][i].right[p] == x:
            new_proj = Proj()
            new_proj.left = proj[n][i].left
            new_proj.right = proj[n][i].right
            new_proj.part = proj[n][i].part + 1
            c.append(new_proj)
    enclosure(c)
    if is_contained(c) == -1:
        proj.append(c)
        new_status = StatusTrans(n, len(proj) - 1, x)
        status_trans.append(new_status)
    else:
        status_pos = is_contained(c)
        new_status = StatusTrans(n, status_pos, x)
        status_trans.append(new_status)


# 生成项目集族
def gen_proj():
    # 生成c0
    c_0 = Proj()
    c_0.left = rule_list[0].left
    c_0.right = rule_list[0].right
    c_0.part = 0
    proj.append([c_0, ])
    enclosure(proj[0])
    l = len(proj)
    i = 0
    # 读操作
    while i != l:
        for j in range(len(proj[i])):
            p = proj[i][j].part
            if p != len(proj[i][j].right):
                if j > 0 and proj[i][j-1].part != len(proj[i][j-1].right):
                    if proj[i][j].right[p] == proj[i][j - 1].right[proj[i][j
- 1].part]:
                        continue
                go(i, proj[i][j].right[p])
                l = len(proj)
        i = i + 1


def print_proj():
    """
    打印项目集族与识别活前缀的DFA
    :return: None
    """
    with open('proj.txt', 'w', encoding='utf-8') as proj_file:
        proj_file.write('项目集族如下\n')
        for i in range(len(proj)):
            proj_file.write('I%d\n' % i)
            for j in range(len(proj[i])):
                p = proj[i][j].part
                s = proj[i][j].right[:]
                s.insert(p, '.')
                proj_file.write('%s -> %s\n' % (proj[i][j].left,
"".join(s)))
        proj_file.write('\n')
        proj_file.write('识别活前缀的DFA如下\n')
        proj_file.write('初始状态\t接收终结符\t到达的状态\t\n')
        for k in range(len(status_trans)):
            proj_file.write('I%d\t\t%s\t\tI%d\t\t\n' %
(status_trans[k].status_init, status_trans[k].x,
status_trans[k].status_trans))
            proj_file.write('\n')


def get_rule_pos(temp):
    """
    得到某条产生式在规则集rule_list中的位置
    :param temp: 待查询的规则
    :return: 该规则在rule_list中的位置
    """
    for i in range(len(rule_list)):
        if rule_list[i].left == temp.left and rule_list[i].right ==
temp.right:
            return i


def create_slr1():
    """
    建立SLR1分析表
    :return: SLR1分析表
    """
    for i in range(len(status_trans)):
        if status_trans[i].x in VT:
            SLR1[status_trans[i].status_init][VT.index(status_trans[i].x)] =
'S' + str(status_trans[i].status_trans)
        else:
            SLR1[status_trans[i].status_init][VN.index(status_trans[i].x) +
len(VT) - 1] = status_trans[i].status_trans
    for j in range(len(proj)):
        for k in range(len(proj[j])):
            p = proj[j][k].part
            if p == len(proj[j][k].right) and proj[j][k].left == VN[0]:
                SLR1[j][VT.index('#')] = 'acc'
            elif p == len(proj[j][k].right):
                rule_pos = get_rule_pos(proj[j][k])
                for q in range(len(Follow[VN.index(proj[j][k].left)])):
                    if Follow[VN.index(proj[j][k].left)][q] in VT:
                        SLR1[j][VT.index(Follow[VN.index(proj[j][k].left)]
[q])] = 'R' + str(rule_pos)


def print_slr1():
    """
    打印slr1分析表，将其写入slr1.txt中
    :return:
    """
    with open('slr1.txt', 'w', encoding='utf-8') as slr1_file:
        slr1_file.write('SLR(1)分析表如下\n')
        slr1_file.write('状态\t\tACTION\t\t\t\t\tGOTO\t\t\n')
        slr1_file.write('\t')
        for i in range(len(VT)):
            slr1_file.write('%s\t' % VT[i])
        for j in range(1, len(VN)):
            slr1_file.write('%s\t' % VN[j])
        slr1_file.write('\n')
        for m in range(len(SLR1)):
            slr1_file.write('%d\t' % m)
            for n in range(len(SLR1[m])):
                slr1_file.write('%s\t' % SLR1[m][n])
            slr1_file.write('\n')
```

```
def slr1():
    """
    SLR(1)分析过程
    :return: 输入的字符串是否为SLR(1)文法
    """
    current = 0
    # 创建状态栈
    status_stack = Stack()
    status_stack.push(0)
    # 创建符号栈
    sym_stack = Stack()
    sym_stack.push('#')
    with open('src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readline()
        src = src.replace('\n', '')
    with open('output.txt', 'w', encoding='utf-8') as output_file:
        output_file.write('对%s进行SLR(1)分析法的分析过程如下\n' % src)
        output_file.write('状态栈\t\t\t符号栈\t\t输入串
\t\tACTION\t\tGOTO\t\n')
        while True:
            s = status_stack.get_stackelem()
            str_sta = [str(i) for i in s]
            str_sta = " ".join(str_sta)
            output_file.write("%-18s" % str_sta)
            sy = sym_stack.get_stackelem()
            output_file.write('\t')
            str_sym = [str(j) for j in sy]
            str_sym = "".join(str_sym)
            output_file.write("%-10s" % str_sym)
            output_file.write('\t')
            sta_top = status_stack.get_top()
            if SLR1[sta_top][VT.index(src[current])] != -1 and 'S' in
SLR1[sta_top][VT.index(src[current])]:
                next_status = int(SLR1[sta_top][VT.index(src[current])][1:])
                status_stack.push(next_status)
                sym_stack.push(src[current])
                output_file.write('%10s\t\t' % src[current:])
                output_file.write('%s\t\t' % SLR1[sta_top]
[VT.index(src[current])])
                output_file.write('\n')
                current += 1
            elif SLR1[sta_top][VT.index(src[current])] != -1 and 'R' in
SLR1[sta_top][VT.index(src[current])]:
                output_file.write('%10s\t\t' % src[current:])
                output_file.write('%s\t\t' % SLR1[sta_top]
[VT.index(src[current])])
                rulepos = int(SLR1[sta_top][VT.index(src[current])][1:])
                count = 0
                while count != len(rule_list[rulepos].right):
                    sym_stack.pop()
                    status_stack.pop()
                    count += 1
                sym_stack.push(rule_list[rulepos].left)
                sym_top = sym_stack.get_top()
                while True:
                    sta_top = status_stack.get_top()
                    if SLR1[sta_top][VN.index(sym_top) + len(VT) - 1] != -1:
                        status_stack.push(SLR1[sta_top][VN.index(sym_top) +
len(VT) - 1])
                        output_file.write('%d\n' % SLR1[sta_top]
[VN.index(sym_top) + len(VT) - 1])
                        break
                    else:
                        status_stack.pop()
            elif SLR1[sta_top][VT.index(src[current])] == 'acc':
                output_file.write('%10s\t\t' % src[current:])
                output_file.write('%s\n' % SLR1[sta_top]
[VT.index(src[current])])
                output_file.write('\n')
                output_file.write('%s为合法字符串\n' % src)
                break
            else:
                output_file.write('\n')
                output_file.write('%s为不合法字符串\n' % src)
                break


if __name__ == '__main__':
    with open('rule.txt', 'r', encoding='utf-8') as rule_file:
        Rules = rule_file.readlines()
        for i in range(-1, len(Rules)):
            Rules[i] = Rules[i].replace('\n', '')
    create_rule_list()
    identify_vt_and_vn()
    for j in range(len(VN)):
        First.append([])
        Follow.append([])
    for k in range(0, len(VN)):
        create_first_set(VN[k])
    for p in range(0, len(VN)):
        create_follow_set(VN[p])
    gen_proj()
    print_proj()
    SLR1 = [[-1 for col in range(len(VT) + len(VN) - 1)]for row in
range(len(proj))]
    create_slr1()
    print_slr1()
    slr1()
```