《编译原理》专题3设计

目标任务

实验项目

实现LL(1)分析中控制程序（表驱动程序）；完成以下描述赋值语句的LL(1)文法的LL(1)分析过程。

```
G[S]:
S→V=E
E→TE’
E’→ATE’\|ε
T→FT’
T’→MFT’\|ε
F→ (E)\|i
A→+\|-
M→\*\|/
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。

设计要求

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简单的语法错误；
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；
4. 选做：如有可能，考虑如何用文法描述C语言的if语句，使整个文法仍然为LL1文法，并使得你的递归下降程序可以分析赋值语句和if语句。

程序功能描述

1. 解析 LL(1) 文法
2. 输入, 解析一个二元式数组文件
3. 根据LL(1)文法识别,分析二元式文件并输出结果

数据结构

二元式文件结构

```
type TokenEntity struct {
    text string
    tokenName string
}

var tokenEntityList  []tokenEntity
```

LL(1)文法

```
//First和Last集
var first := []string
var last := []string

//产生式
type Production struct{
    from string
    to string
}

type LL1Analyzer struct{
    analysisStack string            //分析栈
    leftString string               //输入串
    analysisTable [][]Production    //分析表
}
```

程序结构描述

构建分析表

```
func (analyzer *LL1Analyzer) buildTable() {
    analyzer.analysisTable['E']['i'] = Production{from: "E", to: "Te"}
    analyzer.analysisTable['E']['('] = Production{from: "E", to: "Te"}
    analyzer.analysisTable['e']['+'] = Production{from: "e", to: "ATe"}
    analyzer.analysisTable['e']['-'] = Production{from: "e", to: "ATe"}
    analyzer.analysisTable['e']['('] = Production{from: "e", to: ""}
    analyzer.analysisTable['e'][')'] = Production{from: "e", to: ""}
    analyzer.analysisTable['T']['i'] = Production{from: "T", to: "Ft"}
    analyzer.analysisTable['T']['('] = Production{from: "T", to: "Ft"}
    analyzer.analysisTable['t']['+'] = Production{from: "t", to: ""}
    analyzer.analysisTable['t']['-'] = Production{from: "t", to: ""}
    analyzer.analysisTable['t']['*'] = Production{from: "t", to: "MFt"}
    analyzer.analysisTable['t']['/'] = Production{from: "t", to: "MFt"}
    analyzer.analysisTable['t'][')'] = Production{from: "t", to: ""}
    analyzer.analysisTable['t']['#'] = Production{from: "t", to: ""}
    //......
}
```

LL1分析法的实现

```
def ll1():
    flag = True
    with open('src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readlines()
    out_file = open('output.txt', 'w', encoding='utf-8')
    for i in range(len(src)):
        src[i] = src[i].replace('\n', '')
        current = 0
        ll1_stack = Stack()
        # '#'和起始符号进栈
        ll1_stack.push('#')
        ll1_stack.push(VN[0])
        a = src[i][current]
        while flag:
            x = ll1_stack.pop()  # x为栈顶元素
            if x in VT and x != '#':
                if x == a:
                    current += 1
                    a = src[i][current]     # a为当前终结符
                else:
                    flag = False
            elif x == '#':
                if a == '#':
                    break
                else:
                    flag = False
            else:
                if LL1[VN.index(x)][VT.index(a)] != 0:
                    if 'ε' not in LL1[VN.index(x)][VT.index(a)].right:
                        rlist = LL1[VN.index(x)][VT.index(a)].right[:]
                        rlist.reverse()
                        #产生式右部反序进栈
                        for c in range(len(rlist)):
                            ll1_stack.push(rlist[c])
                else:
                    flag = False
        if flag:
            out_file.write('%s为合法字符串\n' % src[i])
        else:
            out_file.write('%s为不合法字符串\n' % src[i])
    out_file.close()
```

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)
```

打印LL1分析表

```
def print_ll1_chart():
    with open('ll1chart.txt', 'w', encoding='utf-8') as chart_write:
        chart_write.write('生成的LL1分析表如下\n')
        for c in VT:
            chart_write.write("%s       \t" % c)
        chart_write.write("\n")
        for i in range(len(LL1)):
            for j in range(len(LL1[i])):
                if LL1[i][j] != 0:
                    chart_write.write("%s -> %s     \t" % (LL1[i][j].left,
"".join(LL1[i][j].right)))
                else:
                    chart_write.write("%s       \t" % LL1[i][j])
            chart_write.write("\n")
```

测试

测试用例输入

```
i+i*i#
i*i+i#
i*(i+i#
```

测试用例输出

```
i+i*i#为合法字符串
i*i+i#为合法字符串
i*(i+i#为不合法字符串
```

源代码

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# LL(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 = []
```

```
# 分析栈
class Stack(object):
    # 初始化为空的list
    def __init__(self):
        self.items = []

    # 入栈
    def push(self, item):
        self.items.append(item)

    # 出栈
    def pop(self):
        return self.items.pop()


#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 = []
# LL(1)分析表
LL1 = []


# 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]:
                    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)
```

```python
# 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)


# 创建LL1分析表
def create_ll1_chart(ch):
    for i in range(len(rule_list)):
        if ch == rule_list[i].left:
```

```
      # 若该条规则右部的第一个元素为非终结符，则应将该规则填入右部第一个元素的first集中的元素对应的区域内
      if rule_list[i].right[0] in VN:
        for c in VT:
          if c in First[VN.index(rule_list[i].right[0])]:
            LL1[VN.index(ch)][VT.index(c)] = rule_list[i]
      # 若该条规则右部第一个元素为终结符，则将该规则填入该终结符所对应的区域
      elif rule_list[i].right[0] in VT:
        LL1[VN.index(ch)][VT.index(rule_list[i].right[0])] = rule_list[i]
      # 若该条规则右部第一个元素为空串，则将该规则填入左部字符的follow集中的元素所对应的区域内
      else:
        for s in Follow[VN.index(ch)]:
          LL1[VN.index(ch)][VT.index(s)] = rule_list[i]


# 打印first集和follow集
def print_sets():
  with open('set.txt', 'w', encoding='utf-8') as set_write:
    set_write.write("生成的first集如下\n")
    for k in range(len(VN)):
      set_write.write("%3s:\t" % VN[k])
      for p in First[k]:
        set_write.write("%s\t" % p)
      set_write.write("\n")
      set_write.write("\n")
    set_write.write("生成的follow集如下\n")
    for m in range(len(VN)):
      set_write.write("%3s:\t" % VN[m])
      for n in Follow[m]:
        set_write.write("%s\t" % n)
      set_write.write("\n")
      set_write.write("\n")


# 打印LL1分析表
def print_ll1_chart():
  with open('ll1chart.txt', 'w', encoding='utf-8') as chart_write:
    chart_write.write('生成的LL1分析表如下\n')
    for c in VT:
      chart_write.write("%s   \t" % c)
    chart_write.write("\n")
    for i in range(len(LL1)):
      for j in range(len(LL1[i])):
        if LL1[i][j] != 0:
          chart_write.write("%s -> %s   \t" % (LL1[i][j].left, "".join(LL1[i][j].right)))
```

```python
                else:
                    chart_write.write("%s       \t" % LL1[i][j])
            chart_write.write("\n")


# LL1分析法的实现
def ll1():
    flag = True
    with open('src.txt', 'r', encoding='utf-8') as src_file:
        src = src_file.readlines()
    out_file = open('output.txt', 'w', encoding='utf-8')
    for i in range(len(src)):
        src[i] = src[i].replace('\n', '')
        current = 0
        ll1_stack = Stack()
        # '#'和起始符号进栈
        ll1_stack.push('#')
        ll1_stack.push(VN[0])
        a = src[i][current]
        while flag:
            x = ll1_stack.pop()  # x为栈顶元素
            if x in VT and x != '#':
                if x == a:
                    current += 1
                    a = src[i][current]     # a为当前终结符
                else:
                    flag = False
            elif x == '#':
                if a == '#':
                    break
                else:
                    flag = False
            else:
                if LL1[VN.index(x)][VT.index(a)] != 0:
                    if 'ε' not in LL1[VN.index(x)][VT.index(a)].right:
                        rlist = LL1[VN.index(x)][VT.index(a)].right[:]
                        rlist.reverse()
                        #产生式右部反序进栈
                        for c in range(len(rlist)):
                            ll1_stack.push(rlist[c])
                else:
                    flag = False
        if flag:
            out_file.write('%s为合法字符串\n' % src[i])
        else:
            out_file.write('%s为不合法字符串\n' % src[i])
    out_file.close()
```

```python
if __name__ == '__main__':
    with open('rules.txt', 'r', encoding='utf-8') as rule_file:
        Rules = rule_file.readlines()
        for i in range(0, 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([])
    LL1 = [[0 for col in range(len(VT))]for row in range(len(VN))]
    for k in range(0, len(VN)):
        create_first_set(VN[k])
    for p in range(0,len(VN)):
        create_follow_set(VN[p])
    print_sets()
    for ch in VN:
        create_ll1_chart(ch)
    print_ll1_chart()
    ll1()
```