院系	年级专业	姓名	学号	实验日期
计算机学院	2019计科	吴家隆	1915404063	2021.9.11

编程语言：***python3.9***

实验内容

给定下面的语法

$$E->E+T$$
$$E->T$$
$$T->T*F$$
$$T->F$$
$$F->(E)$$
$$F->id$$

输出字符串 id + (id + id) * id的语法树

实验步骤

- 基于SLR(0)分析算法构建可行前缀的DFA
- 基于DFA构建预测分析表
- 基于预测分析表结合栈进行语法解析

代码详解

初始化变量

```
ACTION = [] # 存放ACTION函数
GOTO = [] # 存放GOTO函数
grams = [] # 存放产生式
FIRST = {} # 存放终结符号的First集合
FOLLOW = {} # 存放终结富豪的Follow集合
dot_grams = [] #存放加点之后的产生式
VN = [] # 存放非终结符
VN2Int = {}  # 非终结符映射
VT2Int = {}  # 终结符映射
VT = [] # 存放终结符
Vs = ["#"] # 存放所有的符号
items = [] # 存放一个状态
NodeSymbol = [] # 存放树节点
location = 0  # 输入位置
status_stack = []  # 状态栈
symbol_stack = []  # 符号栈
now_state = ''  # 栈顶状态
input_ch = ''  # 栈顶字符
input_str = ''  # 输入串
now_step = 0  # 当前步骤
```

对实验数据预处理

- 划分终结符和非终结符

```
def get_v():
    vn_num = 0
    vt_num = 0
    for s in grams:
        x, y = s.split("->")
        if x not in VN:
            VN.append(x)
            VN2Int.update({x: vn_num})
            vn_num = vn_num + 1
        for v in y:
            if v.isupper():
                if v not in VN:
                    VN.append(v)
                    VN2Int.update({v: vn_num})
                    vn_num = vn_num + 1
            else:
                if v not in VT:
                    VT.append(v)
                    VT2Int.update({v: vt_num})
                    vt_num = vt_num + 1
    for vn in VN:
        Vs.append(vn)
    for vt in VT:
        Vs.append(vt)
    VT.append("#")
    VT2Int.update({"#": vt_num})
    print("得到非终结符集合：" + str(VN))
    print("得到终结符集合：" + str(VT))
    print("所有的符号集合" + str(Vs))
```

在这一步之前是读取文档中的产生式，为了方便计算机处理字符，我们这里提前将id转化为b单字符，这样有利于判断终结符和非终结符。

```
with open("data.txt","r",encoding="utf-8") as x:
    grams = [i.strip().replace("id","b") for i in  x.readlines()]
```

- 为所有产生式加点

```
def dot_gram():
    # 拓广文法，加入S->E
    dot_grams.append("S->.E")
    dot_grams.append("S->E.")

    for gram in grams:
        ind = gram.find("->")
        for i in range(len(gram) - ind - 1):
            tmp = gram[:ind + 2 + i] + "." + gram[ind + 2 + i:]
            dot_grams.append(tmp)
```

构造可行前缀的DFA

通过 *get_VN_gram(v)* 函数返回个非终结符的.产生式，形如 *A->.aBb* 形式

```
def get_VN_gram(v):
    res = []
    for gram in dot_grams:
        ind = gram.find("->")
        if gram[0] == v and gram[ind + 2] == ".":
            res.append(gram)
    return res
```

通过*get_CLOSURE(tmp)*函数生成闭包并闭包，这里要注意要将新获得的res加入到遍历中

```
def get_CLOSURE(tmp):
    CLOSURE = []
    for it in tmp:
        if it not in CLOSURE:
            CLOSURE.append(it)
        x, y = it.split(".")
        if y == "":
            continue
        v = y[0]
        if v in VN:
            res = get_VN_gram(v)
            for re in res:
                if re not in CLOSURE:
                    CLOSURE.append(re)
                    tmp.append(re)
    return CLOSURE
```

通过is_inItems(new_item)判断item是否已经存在，存在则返回位置，不

存在返回-1

```
def is_inItems(new_item):
    if new_item is None:
        return -1
    new_set = set(new_item)
    num = 0
    for item in items:
        old_set = set(item)
        if old_set == new_set:
            return num
        num = num + 1
    return -1
```

通过go(item, v)函数生成并返回下一个item

```
def go(item, v):
    tmp = []
    for it in item:
        x, y = it.split(".")
        if y != "":
            if y[0] == v:
                new_it = x + y[0] + "." + y[1:]
                tmp.append(new_it)
    if len(tmp) != 0:
        new_item = get_CLOSURE(tmp)
        return new_item
```

get_items()函数构建item的集合，首先初始化加入拓广文法，然后遍历items添加新状态得到DFA集合

```
def get_items():
    item = []
    init_s = "S->.E"
    item.append(init_s)
    dot_gram()
    for it in item:
        v = it[it.find(".") + 1]
        if v in VN:
            res = get_VN_gram(v)
            for re in res:
                if re not in item:
                    item.append(re)
    items.append(item)
    num = 0
    for item in items:
        for v in Vs:
            new_item = go(item, v)
            if new_item is not None:
                if is_inItems(new_item) == -1:
                    items.append(new_item)
```

通过以上函数构造了可行前缀的DFA

基于DFA构建预测分析表

求FIRST集合和FOLLOW集合

```
def initail():
    for str in grams:
        part_begin = str.split("->")[0]
        part_end = str.split("->")[1]
        FIRST[part_begin] = ""
        FOLLOW[part_begin] = "#"
def getFirst():
    for str in grams:
        part_begin = str.split("->")[0]
        part_end = str.split("->")[1]
        if not part_end[0].isupper():
            FIRST[part_begin] = FIRST.get(part_begin) + part_end[0]

def getFirst_2():
    for str in grams:
        part_begin = ''
        part_end = ''
        part_begin += str.split('->')[0]
        part_end += str.split('->')[1]
        if part_end[0].isupper():
            FIRST[part_begin] = FIRST.get(part_begin) + FIRST.get(part_end[0])

def getFirst_3():
    while (1):
        test = FIRST
        getFirst_2()
        for i, j in FIRST.items():
            temp = ""
            for word in list(set(j)):
                temp += word
            FIRST[i] = temp
        if test == FIRST:
            break
def getFOLLOW_3():
    while (1):
        test = FOLLOW
        getFollow()
        ##去除重复项
        for i, j in FOLLOW.items():
            temp = ""
            for word in list(set(j)):
                temp += word
            FOLLOW[i] = temp
        if test == FOLLOW:
            break
```

init_lr_table()函数为初始化分析表的结构，为每个非终结符号初始化ACTION函数，为每个终结符号初始化了GOTO函数

```
def init_lr_table():
    for h in range(len(items)):
        ACTION.append([])
        GOTO.append([])
        for w1 in range(len(VT) + 1):
            ACTION[h].append("  ")
        for w2 in range(len(VN)):
            GOTO[h].append("  ")
```

lr_is_legal()函数判别lr是否合法，即既存在规约也存在移进或者多个规约项目并存则判断非法

```
def lr_is_legal():
    has_protocol = 0
    has_shift = 0
    for item in items:
        for it in item:
            x, y = it.split(".")
            if y == "":
                if has_protocol != 0 or has_shift != 0:
                    return False
                has_protocol = 1
            else:
                if y[0] in VT:
                    has_shift = 1
    return True
```

find_gram(it)函数查找it在grams中的位置

```
def find_gram(it):
    x, y = it.split(".")
    mgram = x + y
    try:
        ind = grams.index(mgram)
        return ind
    except ValueError:
        return -1
```

get_lr_table()函数构建slr分析表

- 从DFA构造SLR分析表
  - 状态 i 从 I_i 构造，它的*action*函数如下确定：
    - 如果$[A\to\alpha\cdot a\beta]$在I_i中，并且$goto(I_i, a)=I_j$，那么置$action[i, a]$为sj
    - 如果$[A\to\alpha\cdot]$在I_i中，那么对FOLLOW(A)中的所有a，置$action[i, a]$为rj，j是产生式$A\to\alpha$的编号
    - 如果$[S'\to S\cdot]$在I_i中，那么置$action[i, \$]$为接受$acc$
  - 使用下面规则构造状态i的*goto*函数：
    - 对所有的非终结符A，如果$goto(I_i, A)=I_j$，那么$goto[i, A]=j$
  - 不能由上面两步定义的条目都置为error
  - 分析器的初始状态是包含$[S'\to\cdot S]$的项目集对应的状态

```
def get_lr_table():
    initail()
    getFirst()
    getFirst_2()
    getFirst_3()
    getFirst_3()
    getFollow()
    getFOLLOW_3()
    init_lr_table()
    lr_is_legal()
    i = 0
    j = 0
    for item in items:
        for it in item:
            x, y = it.split(".")
            if y == "":
                if it == "S->E.":
                    ACTION[i][len(VT) - 1] = "acc"
                ind = find_gram(it)
                if ind != -1:
                    for k in range(len(ACTION[i])):
                        ACTION[i][k] = "r" + str(ind + 1)
            else:
                next_item = go(item, y[0])
                ind = is_inItems(next_item)
                if ind != -1:
                    if y[0] in VT:
                        j = VT2Int[y[0]]
                        ACTION[i][j] = "s" + str(ind)
                    if y[0] in VN:
                        j = VN2Int[y[0]]
                        GOTO[i][j] = ind
        i = i + 1
    print_lr_table()
```

基于预测分析表结合栈进行语法解析

is_end()函数判断规约是否结束，如果在符号栈中只剩下了E和#，则判断规约结束

```
def is_end():
    if input_str[location:len(input_str)] == '#':
        if symbol_stack[-1] == 'E' and symbol_stack[-2] == '#':
            output()
            return True
        else:
            return False
    else:
        return False
    return True
```

stipulations根据分析表进行规约，首先要判断is_end()一直为False，获取当前的DFA状态和input栈顶的元素，通过ACTION的函数获得当前应该进行的操作

- 如果为s，则进入到action，取出的元素入Symbol栈中
- 如果为r，则进入到goto，取出Symbol栈中元素进行规约

直到is_end()为True判断分析结束，则规约完成

在函数中，也有语法树的构建过程

在s中，构建子节点，在r中，将子节点作为新节点的孩子构建父节点

```
def stipulations():
    global location
    print('----分析过程----')
    print("index\t\t", end='')
    print('%-20s' % 'Status', end='')
    print('%-50s' % 'Symbol', end='')
    print('%-30s' % 'Input', end='')
    print('Action')
    for i in range(len(dot_grams)):
        print('---', end='')
    print()
    symbol_stack.append('#')
    status_stack.append(0)
    while not is_end():
        now_state = status_stack[-1]
        input_ch = input_str[location]
        if input_ch not in Vs:
            print(input_ch)
            print("错误字符")
            return -1
        output()
        find = ACTION[now_state][VT2Int[input_ch]]
        if find[0] == 's':
            num = "".join(find[1:])
            symbol_stack.append(input_ch)
            status_stack.append(int(num))
            NodeSymbol.append(Node(input_ch))
            location += 1
            print('action[%s][%s]=s%s' % (now_state, input_ch, num))
        elif find[0] == 'r':
            num = "".join(find[1:])
```

```
            num = int(num)
            g = grams[num - 1]
            right_num = count_right_num(g)
            for i in range(right_num):
                status_stack.pop()
                symbol_stack.pop()
            node = Node(g[0])
            for i in range(right_num):
                node.add_child(NodeSymbol.pop())
            node.l_child.reverse()
            NodeSymbol.append(node)
            symbol_stack.append(g[0])
            now_state = status_stack[-1]
            symbol_ch = symbol_stack[-1]
            find = GOTO[now_state][VN2Int.get(symbol_ch, -1)]
            print("find", find)
            if find == -1:
                print('分析失败')
                return -1
            status_stack.append(find)
            print('%s' % g)
        else:
            return -1
    print("\n is done")
    return 0
```

其他输出函数

```
def print_grams():
    print("----产生式集合----")
    num = 1
    for gram in grams:
        print("(%d)%s" % (num, str(gram)))
        num = num + 1
def print_items():
    print("----状态集合----")
    num = 0
    for it in items:
        print("(%d)%s" % (num, str(it)))
        num = num + 1
def print_lr_table():
    print('----LR分析表----')
    print('\t\t|\t', end='')
    print(('%4s' % '') * (len(VT) - 3), end='')
    print('Action', end='')
    print(('%4s' % '') * (len(VT) - 3), end='')
    print('\t|\t', end='')
    print(('%3s' % '') * (len(VN) - 2), end='')
    print('GOTO', end='')
    print(('%3s' % '') * (len(VN) - 2), end='')
    print('\t|')
    print('\t\t\t', end='')
    for i in VT:
        print('%3s\t' % i, end='')
    print('\t|\t', end='')
    k = 0
    for i in VN:
```

```
        print('%3s\t' % i, end='')
    print('\t|')
    for i in range(len(dot_grams)):
        print('---', end='')
    print()
    # 表体
    for i in range(len(items)):
        print('%5d\t|\t' % i, end='')
        for j in range(len(VT)):
            print('%4s' % ACTION[i][j], end='')
        print('\t|\t', end='')
        for j in range(len(VN)):
            if not GOTO[i][j] == -1:
                print('%4s' % GOTO[i][j], end='')
            else:
                print('\t', end='')
        print('\t|')
    for i in range(len(dot_grams)):
        print('---', end='')
    print()
```

main部分

将Node输出为语法树格式

```
def put2str(node):
    global res
    if node:
        res += node.val
    if node.l_child:
        for i in node.l_child:
            res += "["
            put2str(i)
            res += "]"
```

主程序的输入输出

```
if __name__ == '__main__':
    get_v()  # 分割终结符和非终结符
    print_grams()  # 输出文法产生式
    get_items()  # 生成状态集合
    print_items()  # 输出状态集合
    get_lr_table()  # 生成lr分析表
    input_str = "id+(id+id)*id#"  # 待分析字符串
    input_str = input_str.replace("id","b")
    stat = stipulations()  # 规约
    if stat == 0:
        print("\n %s 符合文法规则" % input_str)
    else:
        print("\n %s 不符合文法规则" % input_str)
    res = ""
    for i in NodeSymbol:
        put2str(i)
    res = '[' + res + ']'
    res = res.replace("b","id")
    print(res)
```

实验结果

该文法的符号集合

```
所有的符号集合['#', 'E', 'T', 'F', '+', '*', '(', ')', 'b']
```

产生式集合

```
----产生式集合----
(1)E->E+T
(2)E->T
(3)T->T*F
(4)T->F
(5)F->(E)
(6)F->b
```

状态集合

```
----状态集合----
(0)['S->.E', 'E->.E+T', 'E->.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.b']
(1)['S->E.', 'E->E.+T']
(2)['E->T.', 'T->T.*F']
(3)['T->F.']
(4)['F->(.E)', 'E->.E+T', 'E->.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.b']
(5)['F->b.']
(6)['E->E+.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.b']
(7)['T->T*.F', 'F->.(E)', 'F->.b']
(8)['F->(E.)', 'E->E.+T']
(9)['E->E+T.', 'T->T.*F']
(10)['T->T*F.']
(11)['F->(E).']
```

构建的SLR分析表

```
----LR分析表----
       |              Action              |      GOTO      |
             +    *    (    )    b    #     |     E    T    F     |
-------------------------------------------------------------------------------------------------------
    0  |                s4        s5        |       1    2    3     |
    1  |     s6                     acc     |                       |
    2  |     r2   s7        r2        r2     |                       |
    3  |     r4   r4        r4        r4     |                       |
    4  |                s4        s5        |       8    2    3     |
    5  |     r6   r6        r6        r6     |                       |
    6  |                s4        s5        |            9    3     |
    7  |                s4        s5        |                 10    |
    8  |     s6          s11                |                       |
    9  |     r1   s7        r1        r1     |                       |
   10  |     r3   r3        r3        r3     |                       |
   11  |     r5   r5        r5        r5     |                       |
-------------------------------------------------------------------------------------------------------
```

输入待分析字符串为**id+(id+id)*id#**的分析过程

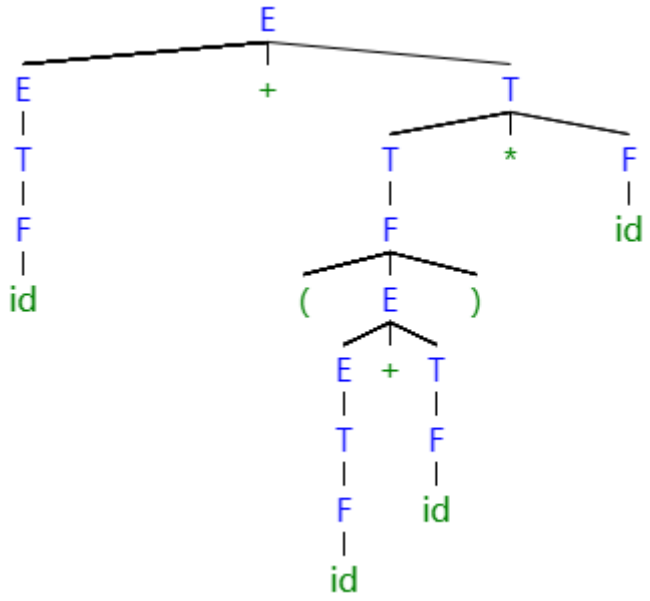
```
----分析过程----
index       Status              Symbol                                  Input                       Action
-------------------------------------------------------------------------------------------------------------------
0       [0]                 ['#']                                   b+(b+b)*b#                  action[0][b]=s5
1       [0, 5]              ['#', 'b']                              +(b+b)*b#                   F->b
2       [0, 3]              ['#', 'F']                              +(b+b)*b#                   T->F
3       [0, 2]              ['#', 'T']                              +(b+b)*b#                   E->T
4       [0, 1]              ['#', 'E']                              +(b+b)*b#                   action[1][+]=s6
5       [0, 1, 6]           ['#', 'E', '+']                         (b+b)*b#                    action[6][(]=s4
6       [0, 1, 6, 4]        ['#', 'E', '+', '(']                    b+b)*b#                     action[4][b]=s5
7       [0, 1, 6, 4, 5]     ['#', 'E', '+', '(', 'b']               +b)*b#                      F->b
8       [0, 1, 6, 4, 3]     ['#', 'E', '+', '(', 'F']               +b)*b#                      T->F
9       [0, 1, 6, 4, 2]     ['#', 'E', '+', '(', 'T']               +b)*b#                      E->T
10      [0, 1, 6, 4, 8]     ['#', 'E', '+', '(', 'E']               +b)*b#                      action[8][+]=s6
11      [0, 1, 6, 4, 8, 6]  ['#', 'E', '+', '(', 'E', '+']         b)*b#                       action[6][b]=s5
12      [0, 1, 6, 4, 8, 6, 5]['#', 'E', '+', '(', 'E', '+', 'b']    )*b#                        F->b
13      [0, 1, 6, 4, 8, 6, 3]['#', 'E', '+', '(', 'E', '+', 'F']    )*b#                        T->F
14      [0, 1, 6, 4, 8, 6, 9]['#', 'E', '+', '(', 'E', '+', 'T']    )*b#                        E->E+T
15      [0, 1, 6, 4, 8]     ['#', 'E', '+', '(', 'E']               )*b#                        action[8][)]=s11
16      [0, 1, 6, 4, 8, 11] ['#', 'E', '+', '(', 'E', ')']         *b#                         F->(E)
17      [0, 1, 6, 3]        ['#', 'E', '+', 'F']                    *b#                         T->F
18      [0, 1, 6, 9]        ['#', 'E', '+', 'T']                    *b#                         action[9][*]=s7
19      [0, 1, 6, 9, 7]     ['#', 'E', '+', 'T', '*']               b#                          action[7][b]=s5
20      [0, 1, 6, 9, 7, 5]  ['#', 'E', '+', 'T', '*', 'b']         #                           F->b
21      [0, 1, 6, 9, 7, 10] ['#', 'E', '+', 'T', '*', 'F']         #                           T->T*F
22      [0, 1, 6, 9]        ['#', 'E', '+', 'T']                    #                           E->E+T
23      [0, 1]              ['#', 'E']                              #
 is done
```

生成的语法树格式

```
 b+(b+b)*b# 符合文法规则
[E[E[T[F[id]]]][+][T[T[F[(][E[E[T[F[id]]]][+][T[F[id]]]][)]]]][*][F[id]]]]
```

$$[E[E[T[F[id]]]][+][T[T[F[(][E[E[T[F[id]]]][+][T[F[id]]]][)]]]][*][F[id]]]$$

将其生成树状图验证



验证结果正确

> 上述实验中基于文法中不会出现中括号
>
> 在实验预处理中已经将id转化为b处理，最后输出时重新转化为id