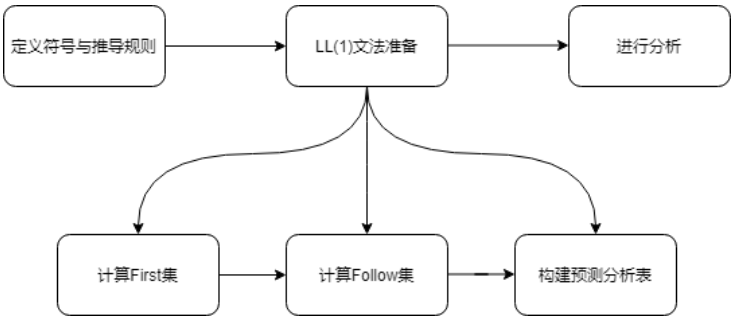
LL（1）语法分析

1.综述

LL(1) 语法使用python语言实现，本质上是使用python语言模拟LL(1)文法的构建与执行过程，依次实现了符号及推导规则的定义、First集与Follow集的计算、预测分析表的构建，以及最终对输入字符串进行分析的过程。具体流程如下所示：



接下来将依次介绍各部分的实现与代码。

2.确定终止符与非终止符

```python
# 定义终止符与非终止符
class ele:
    def __init__(self, char, type):
        self.char = char
        self.type = type  # 1代表非终止符，0代表终止符

#终止符与非终止符
E = ele('E', 1)
T = ele('T', 1)
F = ele('F', 1)
E_1 = ele('E\'', 1)
T_1 = ele('T\'', 1)
num = ele('num', 0)
add = ele('+', 0)
minus = ele('-', 0)
multi = ele('*', 0)
divis = ele('/', 0)
leftbrackets = ele('(', 0)
rightbrackets = ele(')', 0)
end = ele('$',0)
terminator=[num,add,minus,multi,divis,leftbrackets,rightbrackets]
nonterminator=[E,T,E_1,T_1,F]
```

该部分首先定义了 ele类 储存终止符与非终止符的相关信息，类中包含了符号的字符表示以及所属类型（终止符或非终止符）。

在将所有终止符与非终止符加入后，新建了两个数组，分别储存所有的终止符与非终止符，方便后续进行遍历。

3.确定推导规则

```
#语法推导规则的定义
rule_1 = [E,T,E_1]
rule_2 = [E_1,add,T,E_1]
rule_3 = [E_1,minus,T,E_1]
rule_4 = [E_1]#表示推导到空的情况
rule_5 = [T,F,T_1]
rule_6 = [T_1,multi,F,T_1]
rule_7 = [T_1,divis,F,T_1]
rule_8 = [T_1]
rule_9 = [F,leftbrackets,E,rightbrackets]
rule_10 = [F,num]
rule_all=[rule_1,rule_2,rule_3,rule_4,rule_5,
          rule_6,rule_7,rule_8,rule_9,rule_10]
```

我们首先将文法转化成非左递归的文法。

使用list类型储存所有的推导规则，在加入规则之后，新建了一个数组用来储存所有的规则，方便后续进行遍历。

4.构建First集和Follow集

对于文法G的非终止符 a_i，其First集合表示所有可由 a_i 推导出的所有开头终结符号的集合，即：

$$\mathrm{First(a_i)}=\{\alpha|\mathrm{a_i}\overset{*}{\rightarrow}\alpha\beta,\alpha\in\mathrm{V_T},\mathrm{a_i}、\beta\in(\mathrm{V_T}\cup\mathrm{V_N})^*\}$$

采用不断迭代，更新集合，直到集合中的内容不再变化为止的策略，实现代码如下：

```
#计算First集
def getFirst():
    First={'E':[],'T':[],'F':[],'E\'':[],'T\'':[]}
    #先将所有直接终止符前缀加入
    for rule in rule_all:
        if (len(rule)!=1 and rule[1].type==0):
            First[rule[0].char].append(rule[1])
    #接着一直迭代，直到集合不再变化
    count=countNum(First)#先计算First集合种元素总数
    unchange=0
    #迭代，每次迭代完成后比较集合种总数是否有变化
    while unchange==0:
        for rule in rule_all:
            if (len(rule) != 1 and rule[1].type == 1):
                First[rule[0].char].extend(First[rule[1].char])
                First[rule[0].char]=list(set(First[rule[0].char]))
        count_temp=count
        count = countNum(First)
        if (count_temp==count):
            unchange=1
    #集合不再变化后即计算完成
    return First

```

对于文法G的任何非终止符号 A，其Follow集是该文法的所有句型中紧跟在 A 中之后出现的非终止符或 $ 组成的集合，即：

$$\mathrm{Follow(A)}=\{\alpha|\mathrm{S}\overset{*}{\rightarrow}\cdots\mathrm{Aa}\cdots,\mathrm{a}\in\mathrm{V_T}\}$$

为了构建文法G的每个非终结符号 A 的Follow集合，我们采用如下策略，遍历所有的规则集合，不断将元素加入其Follow结合中，直到集合大小不再增大为止。

1. 对文法开始符号 S，置 $ 于 $\mathrm{Follow(S)}$ 中。
2. 若有产生式 $A \to \alpha B \beta$，则把$\mathrm{Follow}(\beta)$ 中所有非 ϵ 元素加入到 $\mathrm{Follow(B)}$ 中。
3. 若有产生式 $A \to \alpha B$,或者有产生式 $A \to \alpha B \beta$，但是 $\epsilon \in \mathrm{First}(\beta)$，则把 $\mathrm{First(A)}$ 中的所有元素加入到 $\mathrm{Follow(B)}$ 中

```
#计算Follow集
def getFollow():
    Follow = {'E': [], 'T': [], 'F': [], 'E\'': [], 'T\'': []}
    #先将直接出现在后面的非终止符的Fist集合与终止符加入
    for rule in rule_all:
        if(len(rule)!=1):
            for index in range(1, len(rule) - 1):
                if (rule[index].type == 1 and rule[index + 1].type ==
0):
                    Follow[rule[index].char].append(rule[index + 1])
                    Follow[rule[index].char] =
list(set(Follow[rule[index].char]))
                elif (rule[index].type == 1 and rule[index + 1].type ==
1):
                    Follow[rule[index].char].extend(FirstSet[rule[index
+ 1].char])
                    Follow[rule[index].char] =
list(set(Follow[rule[index].char]))
    #接着将推导式左侧非终止符的Follow集合加入
    for rule in rule_all:
        if(len(rule)!=1):
            for index in range(1, len(rule) - 1):
                if (rule[index].type == 1 and rule[index + 1].type == 1
and isNullInFirst(rule[index + 1])):

 Follow[rule[index].char].extend(Follow[rule[0].char])
                    Follow[rule[index].char] =
list(set(Follow[rule[index].char]))
            if (rule[len(rule) - 1].type == 1):
                Follow[rule[len(rule) -
1].char].extend(Follow[rule[0].char])
                Follow[rule[len(rule) - 1].char] =
list(set(Follow[rule[len(rule) - 1].char]))
    return Follow
```

5.构建预测分析表

预测分析表采用如下策略进行构造：

如果有产生式 $A \to \alpha$，当 A 呈现在分析栈栈顶时

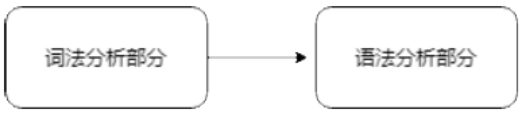
1. 如果当前输入符号 $a \in \mathrm{First}(\alpha)$ 时，α 应被选作 A 的唯一合法代表去执行分析任务，即表项M[A,a] 中应放入产生式 $A \to \alpha$
2. 如果 $\epsilon \in \mathrm{First}(\alpha)$ 并且当前输入符号 $b \in \mathrm{FOLLOW(A)}$，则应把产生式 $A \to \alpha$放入表项M[A,b] 中

实现代码如下所示

```
def createPredictionTabel():
    PredictionTabel = {'E': {}, 'T': {}, 'F': {}, 'E\'': {}, 'T\'': {}}
    for rule in rule_all:
        # 先处理First集的情况
        if(len(rule)!=1):
            #推导式右侧第一个字符是终止符
            if(rule[1].type==0):
                PredictionTabel[rule[0].char][rule[1].char]=rule
            #推导式右侧第一个字符是非终止符
            if(rule[1].type==1):
                for ele in FirstSet[rule[1].char]:
                    PredictionTabel[rule[0].char][ele.char] = rule
        #接着处理Follow集的情况
        else:
            for ele in FollowSet[rule[0].char]:
                PredictionTabel[rule[0].char][ele.char] = rule
    #最后再处理推导至空的情况
            PredictionTabel[rule[0].char]['$'] = rule
    return PredictionTabel
```

6.进行语法分析

具体分析部分分为词法分析与语法分析两个部分。词法分析部分依次读入输入的字符，将其转化成词素，此处特别注意的是数字恶识别。语法分析部分则模拟分析栈与分析表的运行，并在分析的过程中将中间过程输出。若分析过程出错，则也会有提示信息。



该部分的实现代码如下：

```
def analysis(str):
    #定义输入栈
    status=1
    inputStack=[]
    for index in range(len(str)):
        if(str[index].isdigit()):
            if(index == (len(str)-1) or (1 - str[index+1].isdigit())):
                inputStack.append('num')#对于数字的识别并进行特殊处理
        else:
            inputStack.append(str[index])
    inputStack.append('$')
    #定义分析栈
    analysisStack=[end,E]
    #开始进行分析
    while (len(analysisStack)!=1):
        if(len(inputStack)==1 and len(analysisStack)==1):#当输入栈为空时，结束分析
            break
        #进行分析，分别考虑分析栈栈尾为终止符与非终止符的情况
        if(analysisStack[len(analysisStack)-1].type==0):
            if(analysisStack.pop().char!=inputStack.pop(0)):
                error()
                status=-1
                break
```

```
            else:
                printList(analysisStack)
                printStr(inputStack)
                print()
        else:
            if(inputStack[0] in
PredictionTable[analysisStack[len(analysisStack)-1].char].keys()):
                temp=PredictionTable[analysisStack.pop().char]
[inputStack[0]]
                for index in range(1,len(temp)):
                    analysisStack.append(temp[len(temp)-index])
                printList(analysisStack)
                printStr(inputStack)
                printRule(temp)
                print()
            else:
                error()
                status=-1
                break
    if(len(inputStack)!=1):
        if(status==1):
            error()
    else:
        print("分析成功")
```