

# 语法分析器实验报告

FlyThief

2016/11/11

## 目录

<b>1 实验目的</b>	<b>2</b>
<b>2 实验要求</b>	<b>2</b>
<b>3 实验环境</b>	<b>2</b>
<b>4 实验原理</b>	<b>2</b>
4.1 实验目标实现文法 . . . . .	2
4.2 First 集的构造 . . . . .	2
4.3 Follow 集的构造 . . . . .	3
4.4 构造分析表 . . . . .	4
4.5 构造分析程序 . . . . .	5
<b>5 实验样例</b>	<b>6</b>
5.1 几点说明 . . . . .	6
5.2 在线运行简单介绍 . . . . .	7
5.3 自带的样例输出 . . . . .	9
<b>6 实验总结</b>	<b>9</b>

## 1 实验目的

1. 熟悉 LL(1) 文法的实现原理
2. 理解分析表的具体工作流程

## 2 实验要求

1. 构造出 LL(1) 文法的分析表
2. 生成 LL(1) 文法分析子串的分析程序，输出最左推导序列

## 3 实验环境

实验在 arch linux(类 unix 操作系统) 下进行，用 python 代码实现，实际程序运行时需要 python3.5 环境以及 python 的 prettytable 包支持

## 4 实验原理

### 4.1 实验目标实现文法

针对该文法实现：

$$E \rightarrow E + T | E - T | T$$
$$T \rightarrow T * F | T / F | F$$
$$F \rightarrow (E) | num$$

### 4.2 First 集的构造

第一次遍历所有的产生式，如果该非终结符指向的产生式中开头有终结符，则直接添加到它的 first 集中去；如果产生式开头是非终结符，则在它的 first 集中添加字符 'f' 加上该终结符，以待下次添加其具体内容。第二次遍历各非终结符的 first 集，把先前的 f 开头的集合替换为真实具体的终结符。至此，first 集产生结束。

```
def getFirstCollect(self):  
    '''compute first collection'''  
    #为每个非终结符构造一个集合 set 来存储其 first 集  
    for noter in self.noterSymbol:  
        self.firstCollect[noter] = set()  
        self.followCollect[noter] = set()  
  
    for noter in self.noterSymbol:  
        for m in self.allGenerator[noter]:  
            # judge empty generator
```

```

    if m == "empty":
        #如果产生式为空，则直接添加空到其 first 集中
        self.firstCollect[noter].add("empty")

        # if first symbol is terminal, add it
        #如果首字母是终结符，则直接添加首字母到其 first 集中
        if m[0:1] in self.terSymbol:
            self.firstCollect[noter].add(m[0:1])
        else:
            # if it's none terminal, add a symbol with (f + noter)
            #实际操作过程中非终结符可能有一个或两个字符，如T,T'，故需要分别判断
            if m[0:2] in self.noterSymbol:
                self.firstCollect[noter].add("f" + m[0:2])
            elif m[0:1] in self.noterSymbol:
                self.firstCollect[noter].add("f" + m[0:1])
            #替换f为具体 first 集
            # replace f with its real first collect
            for no in self.noterSymbol:
                for noter in self.noterSymbol:
                    if "f" + no in self.firstCollect[noter]:
                        self.firstCollect[noter].remove("f" + no)
                        self.firstCollect[noter] = self.firstCollect[noter].\\
union(self.firstCollect[no])

```

### 4.3 Follow 集的构造

先遍历所有的产生式，识别到当前字符是非终结符时，判断下一个字符是否是终结符，如果是，直接添加到该非终结符的 follow 集合中；如果不是，则添加后面的非终结符的 first 集，同时判断该产生式是否要添加左端字符的 follow 集（函数 isAddFollow 实现），如果要添加，则添加'o' + 该非终结符。

```

def getFollowCollect(self):
    '''get follow collection'''
    self.followCollect[self.startSymbol].add('$')
    for geneLine in self.allGenerator:
        for eachGene in self.allGenerator[geneLine]:
            if eachGene=="empty":
                pass
            else:
                m=0
                up=len(eachGene)
                while True:

```

```

.....
    # 判断当前的字符是非终结符时，添加相应的 follow 集或终结
    if self.isAddFollow(eachGene[l+1:]):
        self.followCollect[eachGene[start:end]] \\\
        .add("o"+geneLine)
    if eachGene[l+1:l+2] in self.terSymbol:
        self.followCollect[eachGene[start:end]] \\\
        .add(eachGene[l+1:l+2])
    if eachGene[l+1:l+3] in self.noterSymbol:
        self.followCollect[eachGene[start:end]] = \\\
        self.followCollect[eachGene[start:end]].union(
            self.firstCollect[eachGene[l + 1:l + 3]])
    elif eachGene[l+1:l+2] in self.noterSymbol:
        self.followCollect[eachGene[start:end]] = \\\
        self.followCollect[eachGene[start:end]].union(\\\
            self.firstCollect[eachGene[l + 1:l + 2]])
.....

#replace o+ noterminal to real follow(noterminal)
#把字符o替换成具体的终结符
for no in self.noterSymbol:
    for noter in self.noterSymbol:
        if "o" + no in self.followCollect[noter]:
            self.followCollect[noter].remove("o" + no)
            self.followCollect[noter] = \\\
self.followCollect[noter].union(self.followCollect[no])
#此处还需要再替换一次，由于可能一次替换不能完全替换
    for ..... # 此处为避免重复，不再粘贴代码
        .....

#去除集合中的 empty
for noter in self.noterSymbol:
    if "empty" in self.followCollect[noter]:
        self.followCollect[noter].remove("empty")

```

#### 4.4 构造分析表

由于构造分析表需要确定对应的非终结符遇到终结符时的产生式，所以实际上在构造 first 集和 follow 集之后可以构造出分析表。比如，对于 T 的 first 集，出现如果有 a 且识别到了  $T \rightarrow aE$ ，则需要把  $\text{analyzeTable}[T][a]$  赋值为  $T \rightarrow aE$  (即 T 遇到 a 时用  $T \rightarrow aE$ )。如果，找不到开头为 a 的，则找头部的 first 集中有 a 的产生式。同时，还要注意遇到 first 集里有 empty 的情况，要添加对应的 follow 的产生式。

```

def analyzeTableConstructor(self):
    for noter in self.noterSymbol:
        for first in self.firstCollect[noter]:
            #如果 first 集里有 empty, 处理 follow 的情况
            if first=="empty":
                realGenerator=""
            for gerator in self.allGenerator[noter]:
                if gerator=="empty" or self.isAddFollow(gerator):
                    realGenerator=gerator
                    break
            for item in self.followCollect[noter]:
                self.addAnalyzeItem(noter,item,realGenerator)
            #添加对应的 first 集产生式
        else:
            for gerator in self.allGenerator[noter]:
                if (gerator[0:1] == first) or (
                    gerator[0:2] in self.noterSymbol and \
                        first in self.firstCollect[gerator[0:2]]) \
                    or (gerator[0:1] in self.noterSymbol and \
                        first in self.firstCollect[gerator[0:1]]):
                    self.addAnalyzeItem(noter, first, gerator)
                    break

```

#### 4.5 构造分析程序

分析构造程序的构造过程中, 只需提取出当前的栈顶和输入的首字符, 然后查找分析表. 将非终结符弹出栈同时把产生式右部反向压入栈中. 同时记录左句型即可. 程序最后可输出最左推导的序列.

```

def analyze(self, inputStr):
    while target!=stack or target != '$':
        firstTarget=target[0] # get first char in target str
        #如果栈顶是终结符, 此时需要消除栈和输入端的一个字符, 并把该字符添
        #加到 isCompleted 中
        # when top of stack is terminal
        if stack[len(stack)-1:] in self.tempTerminal:
            isCompleted+=firstTarget
            stack=stack[0:len(stack)-1]
            inStack = self.reverse_gerator(stack)
            leftSentance = isCompleted + inStack[0:len(inStack) - 1]
            target=target[1:]
        #如果栈顶是非终结符, 则获取生成式, 并把生成式反向添加到栈中, 同时更
        #新已完成字符 isCompleted

```

```

# when top 2 in stack is not terminal
elif stack[len(stack) - 2:] in self.noterSymbol:
    # for handling real number rather 'n'
    if firstTarget in self.number:
        firstTarget='n'

# get generator from analyzeTable
temp=self.analyzeTable[stack[len(stack) - 2:]][firstTarget]
if temp[5:]=="empty":
    stack=stack[0:len(stack)-2]
else:
    # if it's n, means I should move real
    # number from target[0] to stack
    # else just other chars
    if temp[5:] == "n":
        stack = stack[0:len(stack) - 1] + target[0]
    else:
        stack=stack[0:len(stack)-1]+
        self.reverse__generator(temp[5:])

# leftSenctance= isCompleted + reverse(stack)
# 之后把左句型添加到序列表中，以备分析结束时输出

```

## 5 实验样例

关于运行测试实验程序的几点说明：

### 5.1 几点说明

1. 由于之前助教反应程序无法编译运行，本次实验找了一个在线运行 python 程序的环境，但是由于实验代码含有第三方库 (prettytable<sup>1</sup>)，在线环境无法使用第三方库显示该表格，图片附在下方。
2. 如果在本地安装了 prettytable 库的话可以直接运行，输出结果如下

---

<sup>1</sup>为了打印出相对优美的表格

```
Follow[F] : - * $ / + )
Follow[T^] : - $ + )
Follow[T] : - $ + )

AnalyzeTable is here:
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Noter | (      | *      | n      | -      | /      | +      | )      | $      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| E      | E-->TE^ | T^-->*FT^ | E-->TE^ | T^-->empty | T^-->/FT^ | T^-->empty | T^-->empty | T^-->empty |
| T^     | F-->(E) | F-->n     | F-->n     | T^-->empty | T^-->/FT^ | T^-->empty | T^-->empty | T^-->empty |
| F      | T-->FT^ | T-->FT^   | T-->FT^   | E^-->-TE^ | E^-->+TE^ | E^-->empty | E^-->empty | E^-->empty |
| T      |         |           |           |           |           |           |           |           |
| E^     |         |           |           |           |           |           |           |           |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

result for (4*5/3)*4 :
```

## 5.2 在线运行简单介绍

1. 登录网址<http://melpon.org/wandbox/>, 看到下图点左上角区域, 修改为 python3.5

Wandbox

Python

python 3.5.0

Runtime options:

```

300
301 #replace o+ noterminal to real follow(noterminal)
302 for no in self.noterSymbol:
303     for noter in self.noterSymbol:
304         if "o" + no in self.followCollect[noter]:
305             self.followCollect[noter].remove("o" + no)
306             self.followCollect[noter] = self.followCollect[noter].union(
307
308     for no in self.noterSymbol:
309         for noter in self.noterSymbol:
310             if "o" + no in self.followCollect[noter]:
311                 self.followCollect[noter].remove("o" + no)
312                 self.followCollect[noter] = self.followCollect[noter].union(
313
314     for noter in self.noterSymbol:
315         if "empty" in self.followCollect[noter]:
316             self.followCollect[noter].remove("empty")
317
318 def printFollowcollect(self):
$ python prog.py

```

Stdin

Run (or Ctrl+Enter)

2. 添加代码, 运行

```

1 #!/usr/bin/python3.5
2
3 #allGenerator['N']=['1','2','3','4','5','6','7','8','9','0']
4
5 class LL_Grammer():
6     def __init__(self):
7         '''define a LL grammer'''
8         self.noterSymbol = set()
9         self.noterSymbol.add('E')
10        self.noterSymbol.add('T')
11        self.noterSymbol.add('F')
12        self.terSymbol = set()
13        self.allGenerator = {}
14
15        self.startSymbol='E'
16        self.allGenerator['E'] = ['E+T', 'E-T', 'T']
17        self.allGenerator['T'] = ['T*F', 'T/F', 'F']
18        self.allGenerator['F'] = ['(E)', 'n'] # n means num
19
20 $ python prog.py

```

a. 把在线可运行代码copy到这里

b. run

Run (or Ctrl+Enter)

## 3. 可修改测试用例并查看输出结果

```

326 if __name__ == '__main__':
327     test = LL_Grammer()
328     test.killLeftCur() # 消去直接左递归
329     test.printGenerator() # 打印所有产生式
330     test.initAnalyzeTable() # 初始化分析表
331     test.getFirstCollect() # 获取First集
332     test.printFirstcollect() # 打印First集
333     test.getFollowCollect() # 获取Follow集
334     test.printFollowcollect() # 打印Follow集
335     test.analyzeTableConstructor() # 构造分析表
336     test.printAnalyzeTable() # 打印分析表
337     test.analyze("(4*5/3)*4") # 分析并输出分析结果
338     test.analyze("2*4/(3/1)")
339     test.analyze("(3*7+4-3*2+6/1)")
340     test.analyze("5*3/(2*4-6+2/8)") # you can add some other expressions and test them
341
342 $ python prog.py

```

代码最底部是测试用例

```

Stdin

Run (or Ctrl+Enter)

#2 x Share This Code

Code

result for 2*4/(3/1) :
E==>TE^==>FT^E^==>2T^E^==>2*FT^TE^
==>2*4T^TE^==>2*4/FT^TTE^==>2*4/(E)T^TTE^==>2*4/(TE^)T^TTE^==>2*4/(FT^E^)T^TTE^
==>2*4/(3T^E^)T^TTE^==>2*4/(3/FT^TE^)T^TTE^==>2*4/(3/1T^TE^)T^TTE^==>2*4/(3/1TE^)T^TTE^==>
==>2*4/(3/1)T^TTE^==>2*4/(3/1)TTE^==>2*4/(3/1)TE^==>2*4/(3/1)E^==>2*4/(3/1)
result for (3*7+4-3*2+6/1) :
E==>TE^==>FT^E^==>(E)T^E^==>(TE^)T^E^
==>(FT^E^)T^E^==>(3T^E^)T^E^==>(3*FT^TE^)T^E^==>(3*7T^TE^)T^E^==>(3*7TE^)T^E^
==>(3*7E^)T^E^==>(3*7+TE^E)T^E^==>(3*7+FT^E^E)T^E^==>(3*7+4T^E^E)T^E^==>(3*7+4E^E)T^E^
==>(3*7+4-TE^EE)T^E^==>(3*7+4-FT^E^EE)T^E^==>(3*7+4-3T^E^EE)T^E^==>(3*7+4-3*FT^TE^EE)T^E^==

```

输出结果



### 5.3 自带的样例输出

见附件.

## 6 实验总结

本次实验过程中, 我对 LL(1) 文法的分析过程有了更加深入的认识和理解, 对分析表的构成和分析程序的工作过程有了更深的体会. 本次实验实现的词法分析程序只能消除直接左递归, 并不能应对太复杂的语法状况. 希望以后还可以加以改进.