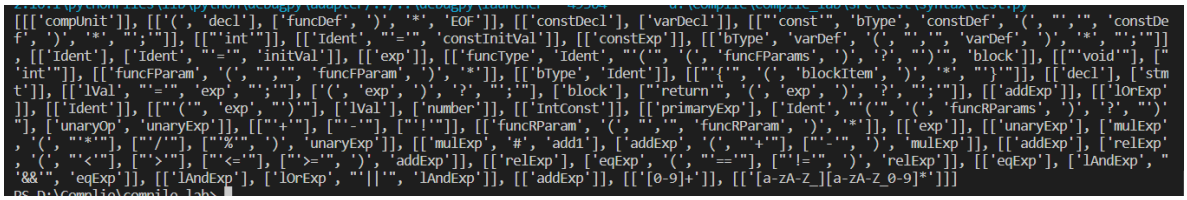
调试getnoOrList方法：

```
listkey= getnoOrList(GrammartextTodict().values())

print(listkey)
```

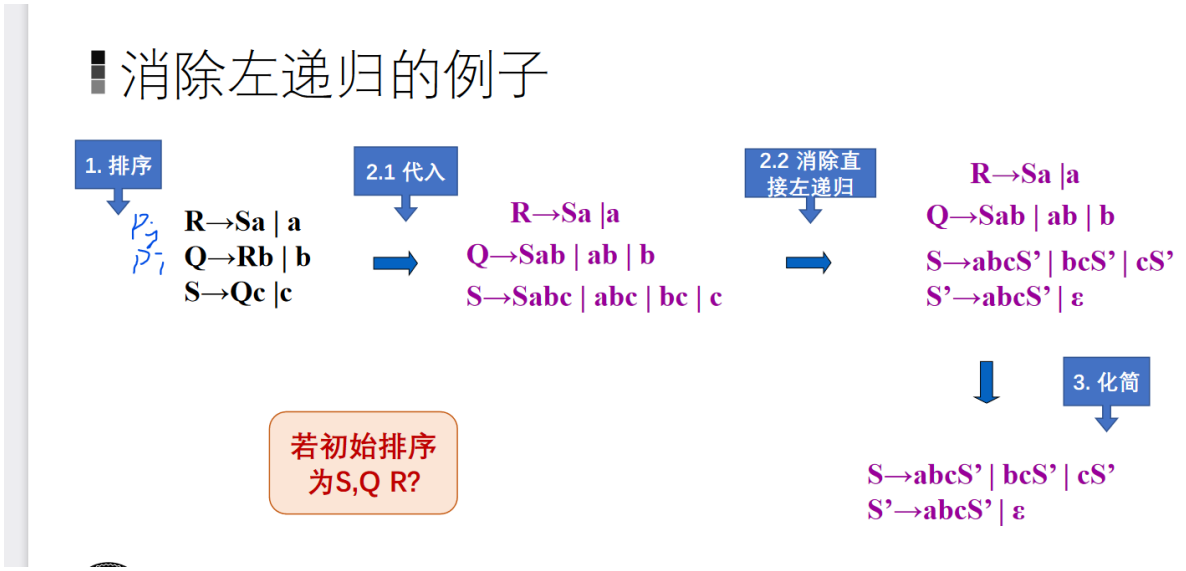
结果：



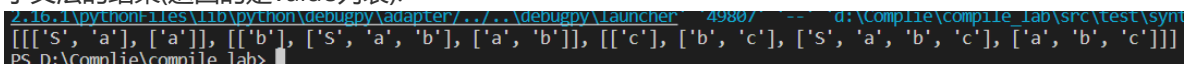
以'|'为界把文法右部分成多个子列表

调试DisDirectRecursionToDirect方法：

```
# listvalueNoor= getnoOrList(GrammartextTodict().values())
# listkey = list(GrammartextTodict().keys())

# Direct_list_no_key = DisDirectRecursionToDirect(listkey,listvalueNoor)

# print(Direct_list_no_key)

d = {'R':['S','a','|','a'],'Q':['R','b','|','b'],'S':['Q','c','|','c']}

listvalueNoor=getnoOrList(d.values())
listkey = list(d.keys())

Direct_list_no_key = DisDirectRecursionToDirect(listkey,listvalueNoor)
```

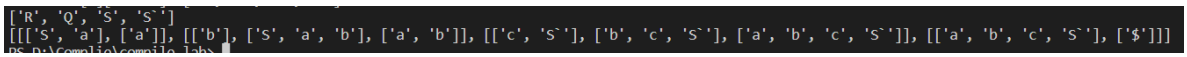


小文法的结果(返回的是value列表):



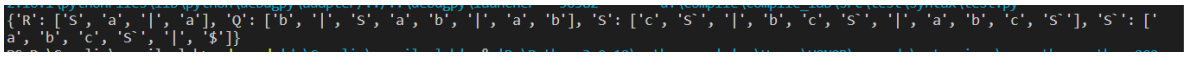
调试remove_direct_left_recursion方法：

```
d = {'R':['S','a','|','a'],'Q':['R','b','|','b'],'S':['Q','c','|','c']}

listvalueNoor=getnoOrList(d.values())
listkey = list(d.keys())

Direct_list_no_key = DisDirectRecursionToDirect(listkey,listvalueNoor)

tuplee =remove_direct_left_recursion(listkey,Direct_list_no_key)
print(tuplee[0])
print(tuplee[1])
```



调试twoListTodict方法：

```
d = {'R':['S','a','|','a'],'Q':['R','b','|','b'],'S':['Q','c','|','c']}

listvalueNoor=getnoOrList(d.values())
listkey = list(d.keys())

Direct_list_no_key = DisDirectRecursionToDirect(listkey,listvalueNoor)

tuplee =remove_direct_left_recursion(listkey,Direct_list_no_key)

dictre = twoListTodict(tuplee[0],tuplee[1])
```



调试simplified方法：

这个方法测试小文法时需要修改两个地方：

原simplified方法：

（使用isalpha()方法需要下载一个包

(192条消息) 超详细的解决ModuleNotFoundError: No module named '*curses*'*错误的方法!!!*troublemaker、的博客-CSDN博客)

```
def simplified(dic:dict):#化简 TODO
    newdict = {'Program':['compUnit']}
    listkey = list(newdict.keys())
    for key in listkey:
        for value in newdict[key]:
            if (value.isalpha() or value == key+'`') and not value == 'EOF'  :
                newdict[value] = dic[value]
                listkey.append(value)
                print('value=',value)#调试用的，可注释掉
    return newdict

    pass
```

第一

```
def simplified(dic:dict):#化简 TODO
    newdict = {'S':['c', 'S`', '|', 'b', 'c', 'S`', '|', 'a', 'b', 'c', 'S`']}
```

初始化字典时，设为小文法的开始符号S的键值关系。如果是c--文法，应该是设置Program的消除递归后的键值关系（可以从之前的步骤中得到）。

第二， 判断条件

```
    if value in string.ascii_uppercase or value == key+'`':
```

小文法是以大写字母为非终结符，c--文法是以全有字母组成的为非终结符

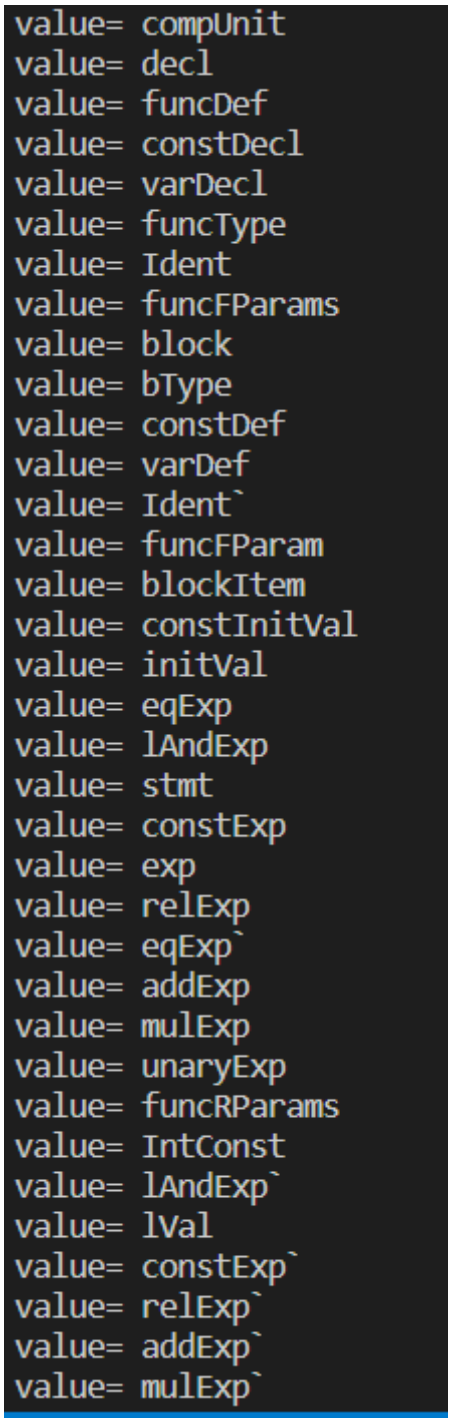
```
d = {'R':['S','a','|','a'],'Q':['R','b','|','b'],'S':['Q','c','|','c']}

listvalueNoor=getnoOrList(d.values())
listkey = list(d.keys())

Direct_list_no_key = DisDirectRecursionToDirect(listkey,listvalueNoor)

tuplee =remove_direct_left_recursion(listkey,Direct_list_no_key)

dictre = twoListTodict(tuplee[0],tuplee[1])

dictresult = simplified(dictre)

print(dictresult)
```

结果：

```
{'S': ['c', 'S`', '|', 'b', 'c', 'S`', '|', 'a', 'b', 'c', 'S`'], 'S`': ['a', 'b', 'c', 'S`', '|', '$']}
PS D:\Compile\compile_lab>
```

用c--文法调试：

```
listvalueNoor= getnoOrList(GrammartextTodict().values())
listkey = list(GrammartextTodict().keys())

Direct_list_no_key = DisDirectRecursionToDirect(listkey,listvalueNoor)

tuplee =remove_direct_left_recursion(listkey,Direct_list_no_key)

dictre = twoListTodict(tuplee[0],tuplee[1])

dictresult = simplified(dictre)

print(dictresult)
```



value= IntConst


```
{'Program': ['compUnit'], 'compUnit': ['(', 'decl', '|', 'funcDef', ')', '*', 'EOF'], 'decl': ['constDecl', '|', 'varDecl'], 'funcDef': ['funcType', '
Ident', "'('", '(', 'funcFParams', ')', '?', "')'", 'block'], 'constDecl': ["'const'", 'bType', 'constDef', '(', "','", 'constDef', ')', '*', "';'"],
'varDecl': ["'int'", 'varDef', '(', "','", 'varDef', ')', '*', "';'"], 'funcType': ["'void'", '|', "'int'"], 'Ident': ['[a-zA-Z_][a-zA-Z_0-9]*', 'Iden
t`'], 'funcFParams': ['funcFParam', '(', "','", 'funcFParam', ')', '*'], 'block': ["'{'", '(', 'blockItem', ')', '*', "'}'"], 'bType': ["'int'"], 'con
stDef': ['Ident', "'='", 'constInitVal'], 'varDef': ['Ident', '|', 'Ident', "'='", 'initVal'], 'Ident`': ['(', "'*'", 'Ident`', '|', '(', "'+'", 'Iden
t`', '|', '(', "'<'", 'Ident`', '|', '(', "'=='", 'Ident`', '|', "'&&'", 'eqExp', 'Ident`', '|', "'||'", 'lAndExp', 'Ident`', '|', '$'], 'funcFParam':
 ["'int'", 'Ident'], 'blockItem': ['stmt', '|', "'const'", 'bType', 'constDef', '(', "','", 'constDef', ')', '*', "';'", '|', "'int'", 'varDef', '(',
"','", 'varDef', ')', '*', "';'"], 'constInitVal': ['constExp'], 'initVal': ['exp'], 'eqExp': ["'!='", ')', 'relExp', 'eqExp`', '|', "'>'", 'eqExp`',
'|', "'<='", 'eqExp`', '|', "'>='", ')', 'addExp', 'eqExp`', '|', "'-'", ')', 'mulExp', 'eqExp`', '|', "'/'", '#', 'add1', 'eqExp`', '|', "'%'", ')',
'unaryExp', '#', 'add1', 'eqExp`', '|', 'Ident', "'('", '(', 'funcRParams', ')', '?', "')'", '#', 'add1', 'eqExp`', '|', "'('", 'exp', "')'", '#', 'ad
d1', 'eqExp`', '|', 'Ident', '#', 'add1', 'eqExp`', '|', 'IntConst', '#', 'add1', 'eqExp`', '|', "'+'", 'unaryExp', '#', 'add1', 'eqExp`', '|', "'-'",
 'unaryExp', '#', 'add1', 'eqExp`', '|', "'!'", 'unaryExp', '#', 'add1', 'eqExp`', '|', 'mulExp', '(', "'*'", '(', "'*'", '#', 'add1', 'eqExp`', '|',
"'/'", '(', "'*'", '#', 'add1', 'eqExp`', '|', "'%'", ')', 'unaryExp', '(', "'*'", '#', 'add1', 'eqExp`', '|', 'Ident', "'('", '(', 'funcRParams', ')'
, '?', "')'", '(', "'*'", '#', 'add1', 'eqExp`', '|', "'('", 'exp', "')'", '(', "'*'", '#', 'add1', 'eqExp`', '|', 'Ident', '(', "'*'", '#', 'add1', '
eqExp`', '|', 'IntConst', '(', "'*'", '#', 'add1', 'eqExp`', '|', "'+'", 'unaryExp', '(', "'*'", '#', 'add1', 'eqExp`', '|', "'-'", 'unaryExp', '(', "
'*'", '#', 'add1', 'eqExp`', '|', "'!'", 'unaryExp', '(', "'*'", '#', 'add1', 'eqExp`', '|', 'addExp', '(', "'+'", '(', "'+'", 'eqExp`', '|', "'-'", '
)', 'mulExp', '(', "'+'", 'eqExp`', '|', "'/'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'%'", ')', 'unaryExp', '#', 'add1', '(', "'+'", 'eqExp`', '|
', 'Ident', "'('", '(', 'funcRParams', ')', '?', "')'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'('", 'exp', "')'", '#', 'add1', '(', "'+'", 'eqExp`',
'|', 'Ident', '#', 'add1', '(', "'+'", 'eqExp`', '|', 'IntConst', '#', 'add1', '(', "'+'", 'eqExp`', '|', "'+'", 'unaryExp', '#', 'add1', '(', "'+'",
 'eqExp`', '|', "'-'", 'unaryExp', '#', 'add1', '(', "'+'", 'eqExp`', '|', "'!'", 'unaryExp', '#', 'add1', '(', "'+'", 'eqExp`', '|', 'mulExp', '(', "
'*'", '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'/'", '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'%'", ')', 'unaryExp', '(', "'*'
", '#', 'add1', '(', "'+'", 'eqExp`', '|', 'Ident', "'('", '(', 'funcRParams', ')', '?', "')'", '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'
('", 'exp', "')'", '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', 'Ident', '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', 'IntConst', '(', "'
*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'+'", 'unaryExp', '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', "'-'", 'unaryExp', '(', "'*'", '#
', 'add1', '(', "'+'", 'eqExp`', '|', "'!'", 'unaryExp', '(', "'*'", '#', 'add1', '(', "'+'", 'eqExp`', '|', relExp', '(', "'<'", '(', "'<'", 'eqExp
', '|', "'>'", '(', "'<'", 'eqExp`', '|', "'<='", '(', "'<'", 'eqExp`', '|', "'>='", ')', 'addExp', '(', "'<'", 'eqExp`', '|', "'-'", ')', 'mulExp', '
(', "'<'", 'eqExp`', '|', "'/'", '#', 'add1', '(', "'<'", 'eqExp`', '|', "'%'", ')', 'unaryExp', '#', 'add1', '(', "'<'", 'eqExp`', '|', 'Ident', "'('
", '(', 'funcRParams', ')', '?', "')'", '#', 'add1', '(', "'<'", 'eqExp`', '|', "'('", 'exp', "')'", '#', 'add1', '(', "'<'", 'eqExp`', '|', 'Ident',
'#', 'add1', '(', "'<'", 'eqExp`', '|', 'IntConst', '#', 'add1', '(', "'<'", 'eqExp`', '|', "'+'", 'unaryExp', '#', 'add1', '(', "'<'", 'eqExp`', '|',
 "'-'", 'unaryExp', '#', 'add1', '(', "'<'", 'eqExp`', '|', "'!'", 'unaryExp', '#', 'add1', '(', "'<'", 'eqExp`', '|', 'mulExp', '(', "'*'", '(', "'*'
", '#', 'add1', '(', "'<'", 'eqExp`', '|', "'/'", '(', "'*'", '#', 'add1', '(', "'<'", 'eqExp`', '|', "'%'", ')', 'unaryExp', '(', "'*'", '#', 'add1',
```