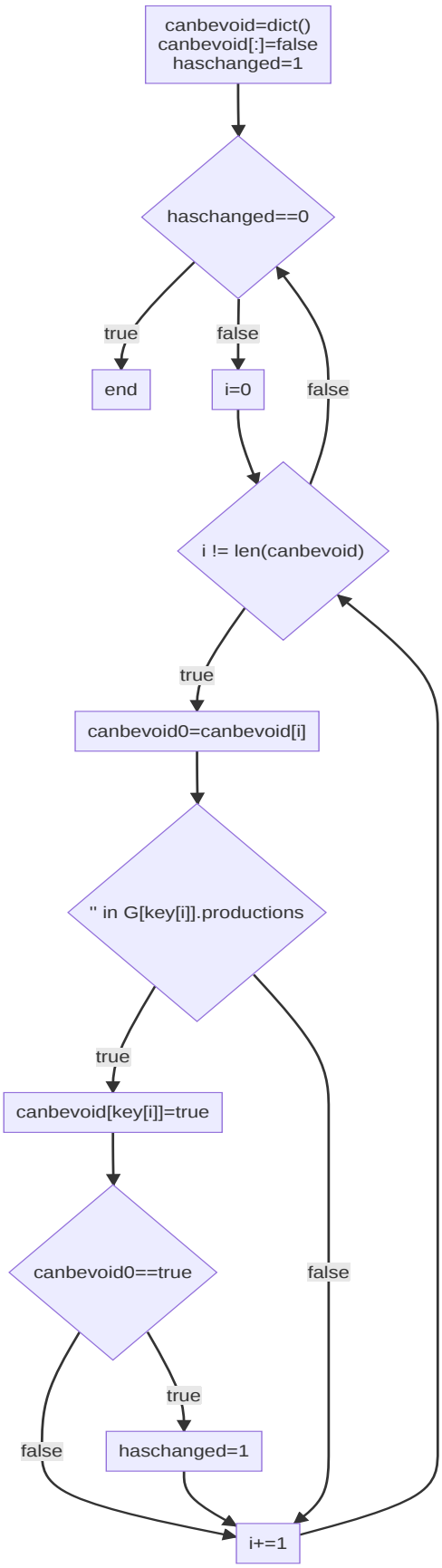
任务二 文法分析

1 设计思路

1.1 判断非终结符是否能推出空

算法流程图如下



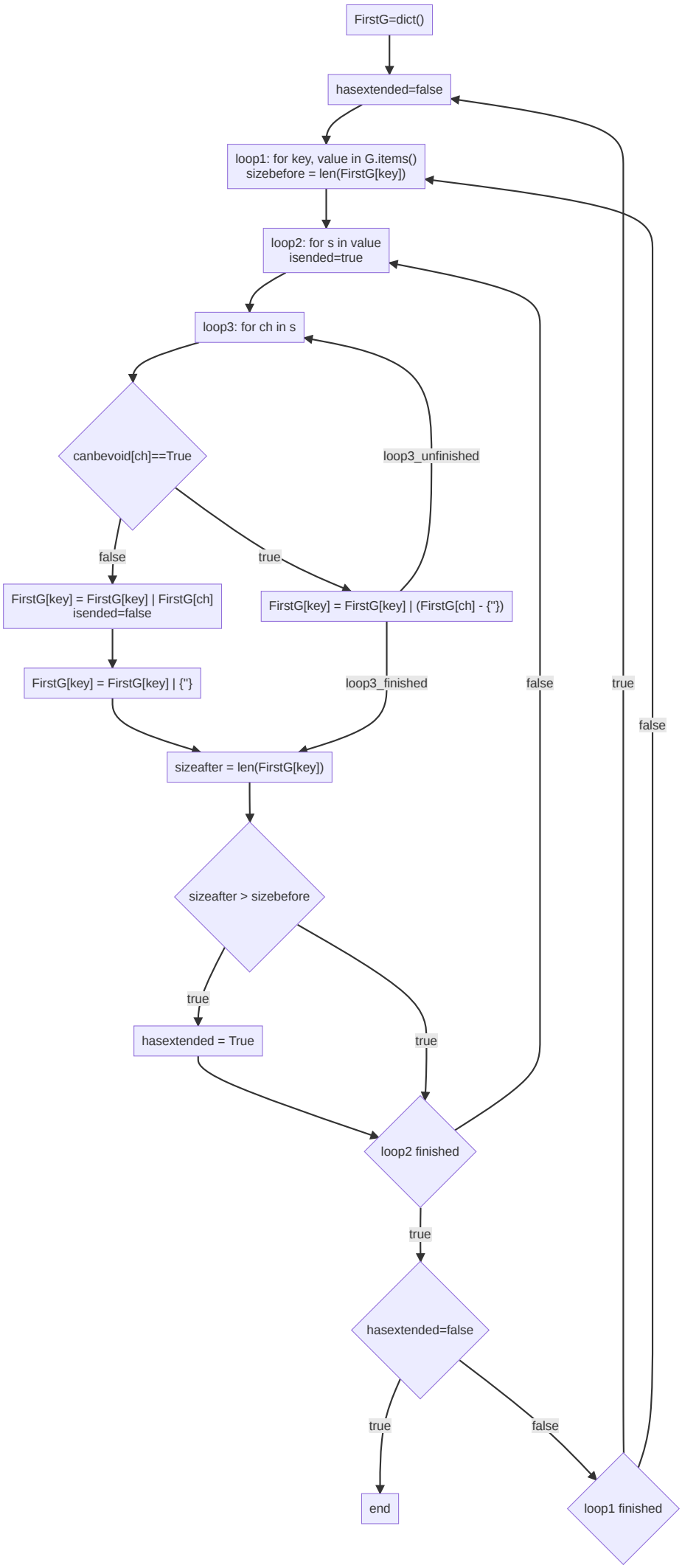
书中给出的判断是否能推出空产生式的算法有一些问题，可能会陷入死锁，故采用改进的判断空产生式算法。
与书中算法不一样的是，该算法将所有canbevoid全部置为false，即算法初始化将所有非终结符标为“不可产生空产生式”，之后循环遍历canbevoid数组，当非终结符产生式右侧能退出空时修改状态为true。当一轮循环后canbevoid数组没有变化则退出该循环，否则，则继续循环，直到canbevoid数组不再变化为止，这个状态量用haschanged保存。

1.2 计算first集

First集的计算使用python的数据结构set，set是一个不允许内容重复的组合，而且set里的内容位置是随意的数据结构

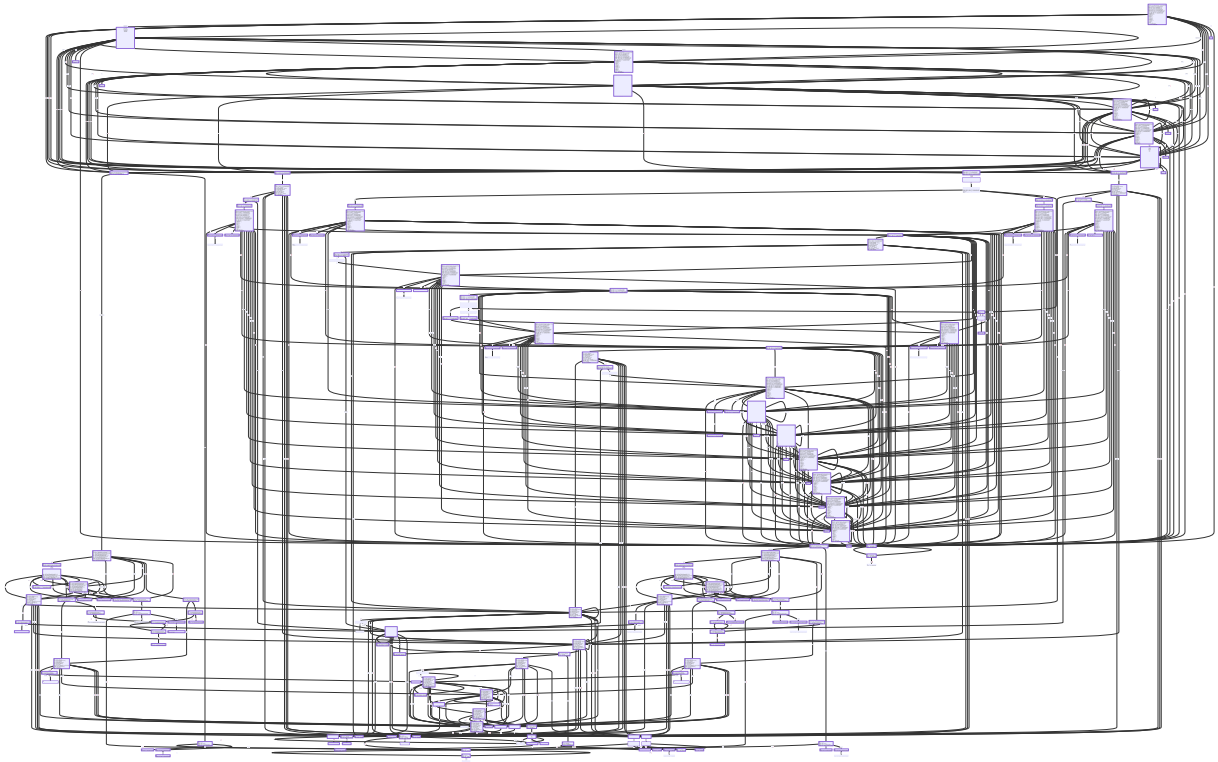
计算first集时要对右侧的产生式第一个符号进行分类.
1）当右侧的第一个符号不能推出空产生式，直接取原first集和右侧符号并集
2）当右侧第一个符号能推出空产生式，first集和右侧所有连续的VN且能能推出空产生式VN取并集
3）若右侧所有产生式均能推出空产生式，first集合加入”
4）否则first集并入右侧第一个不能推出空产生式的first集

算法流程图如下：

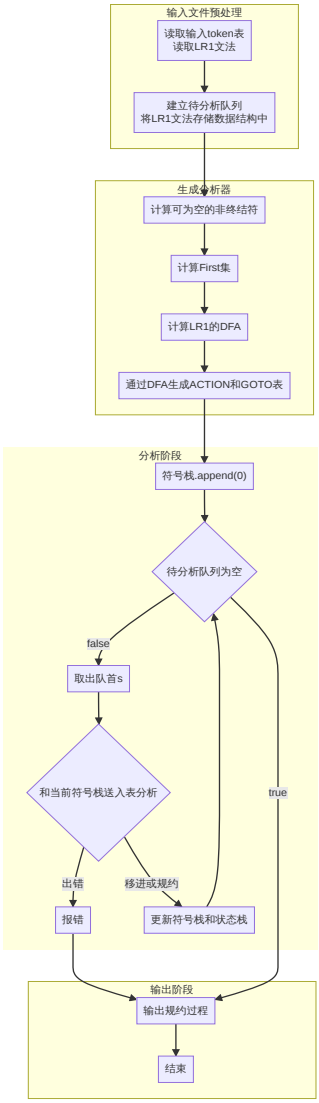

FirstG=dict()
hasextended=false
loop1: for key, value in G.items()
sizebefore = len(FirstG[key])
loop2: for s in value
isended=true
loop3: for ch in s
canbevoid[ch]==True
loop3_unfinished
false
true
FirstG[key] = FirstG[key] | FirstG[ch]
isended=false
FirstG[key] = FirstG[key] | (FirstG[ch] - {''})
FirstG[key] = FirstG[key] | {''}
loop3_finished
false
true
false
sizeafter = len(FirstG[key])
sizeafter > sizebefore
true
hasextended = True
true
loop2 finished
true
hasextended=false
true
false
end
loop1 finished

1.3 计算LR1的DFA

生成DFA的方式和词法分析中的方法类似。同样采用字典数据结构保存DFA，用DFS遍历转换边和状态，LR1_rule的转换表如图所示(可放大)



1.4 算法总流程图



2 代码函数解释

`def sortanddeduplicate(l)`：对于一个列表去重排序

`def checkcanbevoid(G)`：计算可以产生空产生式的VN

`def Firstset(G)`：计算First集

`def printerror(index)` 打印错误信息，index是分析队列中的位置

`def Formstat(rawstat)`：生成状态闭包

`def DFA(G)`：生成LR1转换表

`def LR1table(dfa: dict)`：通过DFA生成GOTO和ACTION表

`def LR1(s: list)`：进行LR1语法分析

3 程序说明

词法程序分析采用LR1文法分析。
程序的可执行文件名称为LR1_parser.exe，源代码文件为LR1_parser.py

3.1 操作说明

在命令行输入命令 ./LR1_parser -h获取帮助信息：

```
PS C:\Users\Anderson\Desktop\Gitworkspace\Compiler> ./LR1_parser -h
usage:
LR1 grammar parser by yuangyan
Courseworkwork for semester2, 2022
 [-h] [-s] [-i] [-r] [-o]

optional arguments:
  -h, --help         show this help message and exit
  -s, --showParsing  show Parsing Process
  -i , --input       input file name, 'lex_parsed.txt' by default
  -r , --rule        rule file name, 'grammar_rule.txt' by default
  -o , --output      output file name, '<input file name>_LR1.txt' by default
```

在词法分析程序中共有5个可选参数，分别为：

-h, --help: 显示帮助信息
-s, --showParsing: 在程序运行的过程中打印分析过程
-i, --input: 选择输入待分析文本路径，不选该参数则默认为同目录下'lex_parsed.txt'文件
-r, --rule: 选择词法分析规则路径，不选该参数则默认为同目录下'grammar_rule.txt'文件
-o, --output: 选择输出路径，不选该参数则默认为同目录下'<输入文件名>_LR1.txt'文件

输入命令格式为

./LR1_parser (-i [待分析文本路径]) (-r [LR1文法分析规则文件]) (-o [输出路径]) (–showParsing)
例如：./LR1_parser -i input.txt -o output.txt -r rule.txt –showParsing

3.2 输入输出样例

输入样例：lex.txt

```
while(a=b){
  a+=1
}
```

经过词法分析后：lex_parsed.txt

```
1 KEYWORD while
1 DELIMITER (
1 IDENTIFIER a
1 ASSIGNMENT_OPERATOR =
1 IDENTIFIER b
1 DELIMITER )
1 DELIMITER {
2 IDENTIFIER a
2 ASSIGNMENT_OPERATOR +=
2 INT 1
3 DELIMITER }
```

将lex_parsed送入分析后的输出

```
PS C:\Users\Anderson\Desktop\Gitworkspace\Compiler> ./LR1_parser
[ERROR] in line 3: while(a=b){
                           ^
invalid syntax: =
```

可见报错。
原来，在文法中`while`里面的语句只能是一个“表达式”，而不能是一个“赋值语句”，所以在这里报错，将a=b修改为a<b后再次输入，并且加入选项`–showParsing`，显示分析过程

输出结果：

```
PS C:\Users\Anderson\Desktop\Gitworkspace\Compiler> ./LR1_parser --showParsing
([0], ['#'], ['while', '(', 'id', 'BinaryOp', 'id', ')', '{', 'id',
'AssignmentOp', 'const', '}', '#'], 'S4')
([0, 4], ['#', 'while'], ['(', 'id', 'BinaryOp', 'id', ')', '{', 'id',
'AssignmentOp', 'const', '}', '#'], 'S21')
([0, 4, 21], ['#', 'while', '('], ['id', 'BinaryOp', 'id', ')', '{', 'id',
'AssignmentOp', 'const', '}', '#'], 'S27')
([0, 4, 21, 27], ['#', 'while', '(', 'id'], ['BinaryOp', 'id', ')', '{', 'id',
'AssignmentOp', 'const', '}', '#'], 'r: <ids> → <id>')
([0, 4, 21, 28], ['#', 'while', '(', 'ids'], ['BinaryOp', 'id', ')', '{', 'id',
'AssignmentOp', 'const', '}', '#'], 'r: <Expr> → <ids>')
([0, 4, 21, 23], ['#', 'while', '(', 'Expr'], ['BinaryOp', 'id', ')', '{', 'id',
'AssignmentOp', 'const', '}', '#'], 'S66')
([0, 4, 21, 23, 66], ['#', 'while', '(', 'Expr', 'BinaryOp'], ['id', ')', '{',
'id', 'AssignmentOp', 'const', '}', '#'], 'S68')
([0, 4, 21, 23, 66, 68], ['#', 'while', '(', 'Expr', 'BinaryOp', 'id'], [')',
'{', 'id', 'AssignmentOp', 'const', '}', '#'], 'r: <ids> → <id>')
([0, 4, 21, 23, 66, 28], ['#', 'while', '(', 'Expr', 'BinaryOp', 'ids'], [')',
'{', 'id', 'AssignmentOp', 'const', '}', '#'], 'r: <Expr> → <ids>')
```

```
([0, 4, 21, 23, 66, 67], ['#', 'while', '(', 'Expr', 'BinaryOp', 'Expr'], [')',
'{', 'id', 'AssignmentOp', 'const', '}', '#'], 'r: <Expr> → <Expr><BinaryOp>
<Expr>')
([0, 4, 21, 23], ['#', 'while', '(', 'Expr'], [')', '{', 'id', 'AssignmentOp',
'const', '}', '#'], 'S72')
([0, 4, 21, 23, 72], ['#', 'while', '(', 'Expr', ')'], ['{', 'id',
'AssignmentOp', 'const', '}', '#'], 'S73')
([0, 4, 21, 23, 72, 73], ['#', 'while', '(', 'Expr', ')', '{'], ['id',
'AssignmentOp', 'const', '}', '#'], 'S12')
([0, 4, 21, 23, 72, 73, 12], ['#', 'while', '(', 'Expr', ')', '{', 'id'],
['AssignmentOp', 'const', '}', '#'], 'r: <ids> → <id>')
([0, 4, 21, 23, 72, 73, 74], ['#', 'while', '(', 'Expr', ')', '{', 'ids'],
['AssignmentOp', 'const', '}', '#'], 'S123')
([0, 4, 21, 23, 72, 73, 74, 123], ['#', 'while', '(', 'Expr', ')', '{', 'ids',
'AssignmentOp'], ['const', '}', '#'], 'S132')
([0, 4, 21, 23, 72, 73, 74, 123, 132], ['#', 'while', '(', 'Expr', ')', '{',
'ids', 'AssignmentOp', 'const'], ['}', '#'], 'r: <Expr> → <const>')
([0, 4, 21, 23, 72, 73, 74, 123, 128], ['#', 'while', '(', 'Expr', ')', '{',
'ids', 'AssignmentOp', 'Expr'], ['}', '#'], 'r: <Assignment> → <ids>
<AssignmentOp><Expr>')
([0, 4, 21, 23, 72, 73, 80], ['#', 'while', '(', 'Expr', ')', '{', 'Assignment'],
['}', '#'], 'r: <S> → <Assignment>')
([0, 4, 21, 23, 72, 73, 78], ['#', 'while', '(', 'Expr', ')', '{', 'S'], ['}',
'#'], 'S92')
([0, 4, 21, 23, 72, 73, 78, 92], ['#', 'while', '(', 'Expr', ')', '{', 'S', '}'],
['#'], 'r: <Loop> → <while><(><Expr><)><{><S><}>')
([0, 10], ['#', 'Loop'], ['#'], 'r: <S> → <Loop>')
([0, 11], ['#', 'S'], ['#'], "r: S'→S")
accepted
```

语句被accepted，并且分析过程中显示了符号栈，已分析栈，待输入串和规约语法
在输出的lex_parsed_LR1.txt中记录了上述过程.