天津大學

《编译原理与技术》实验报告

实验题目　词法分析器与语法分析器的实现

专　　业　计算机科学与技术

组　　长　李润泽

学　　号　3019244266

组　　员　李佳兴

学　　号　3019244276

组　　员　赖坤丰

学　　号　3019244261

组　　员　何子越

学　　号　3019244279

2022 年 5 月 5 日

一、实验目标

本次大作业为编写一个编译器前端（包括词法分析器和语法分析器）

（1）使用自动机理论编写词法分析器

（2）自上而下或者自下而上的语法分析方法编写语法分析器

具体地，我们需要完成以下内容：

（1）编写 SQL--语言的词法分析器，理解词法分析器的工作原理，熟练掌握词法分析器的工作流程并编写源代码，识别出单词的二元属性，填写符号表

（2）编写 SQL--语言的语法分析器，理解自上而下/自下而上的语法分析算法的工作原理；理解词法分析与语法分析之间的关系。语法分析器的输入为 SQL--语言源代码，输出为按扫描顺序进行推导或归约的正确/错误的判别结果，以及按照最左推导顺序/规范规约顺序生成语法树所用的产生式序列。

二、编程语言及环境

编程语言：C++

运行环境：Windows

三、实验原理及开发过程

（一）词法分析器

3.1 词法分析器算法描述

3.1.1 算法思想

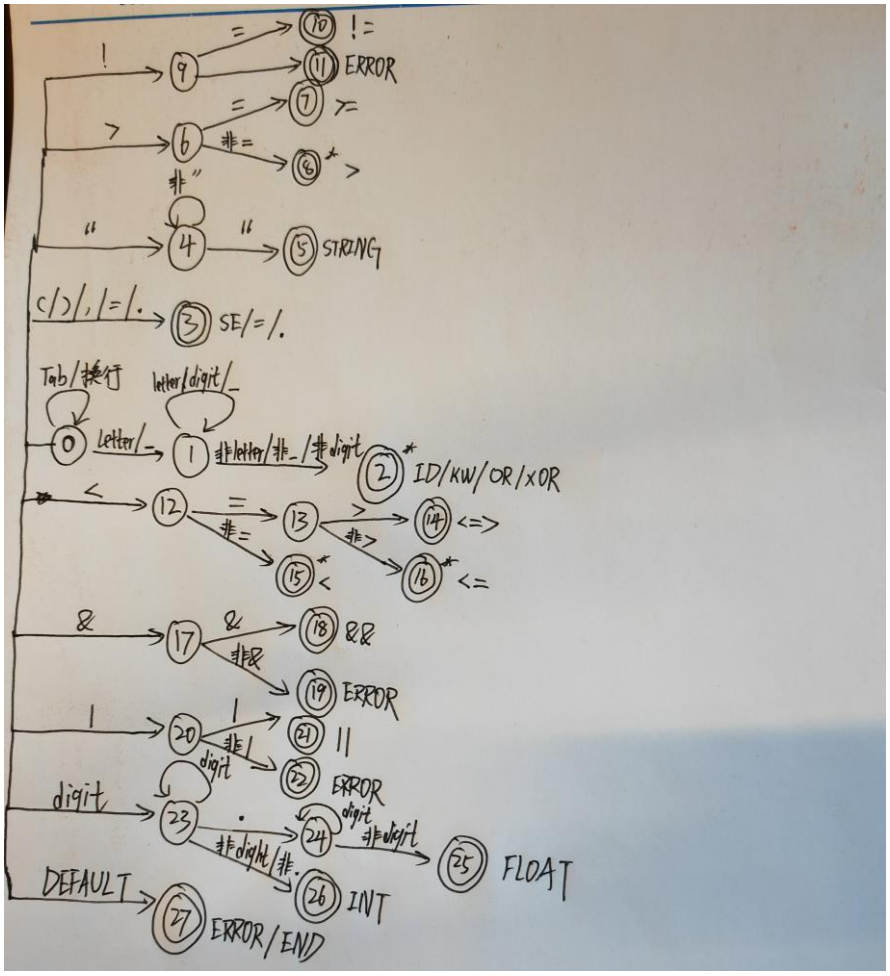
在这个分析器中,用解析的原理将程序语句分成一部分部分的,用多个 case 分支语句来解析关键字、运算符、界符等，但数字、标志符就得用另外的方法；从第 0 个元素开始，往前读取字符，一直遇到界符和运算符（包括空格或者回车)结束;

先判断第一元素的性质，如果是数字，那么，这个单词要么是数字，要么是非法字符，如果不是数字，该单词解析结束;如果是数字，继续解析，直到结束;

同理，如果第一元素是字母，那么该单词要么是关键字，要么是标志符，要么是非法字符;如果第一元素是界符，或者运算符,或者空格或者回车等，则继续读取。

3.1.2 算法实现

（1）状态转换图如下图所示：



1) 预处理

对于一个文件，我们首要进行将其读入，并以 String 形式存储紧接着，再对 String 进行处理，将换行符等取消，并切分为一句句的代码。

2) NFA 确定化和 DFA 最小化思想

NFA 确定化算法：

NFA 转换为 DFA 使用子集构造算法，其中包含两个重要函数：

- Get_eclosure(cur)：当前状态集合 cur 的闭包
- GetMoveTo(cur,alphaList[i])：状态集合 cur 在输入符号为 alphaList[i]的情况下转入目标状态的集合

先初始化空闭包不断利用传播性原则 GetMoveTo(cur,alphaList[i])直至闭包 closure(cur)的规模不再变化，再将多个等效状态合并成一个状态，最后删去不可到达的状态（死状态）。

给出伪代码如下：

```
void deterNFA(int start){
    //初始化
    cur.insert(startSet);
    //遍历状态集合，直到集合尾，期间不断更新状态集合
    for(int i=0;i<stateSet.size();i++){
        for(int j=0;j<alphaSize;j++){
            //求 MoveTo 集合
            cur=getMoveTo(cur,alphaList[j]);
            //求 e_closure
            cur=get_eclosure(cur);
            if(cur.size()>0){
                //判断当前集合是否在状态集合中
                //不存在则加入集合
                    stateSet.push_back(cur);
            }
            else //否则表示当前的状态无法扩展出节点，将无法扩展出的状态标
记为-1
                stateList[i][j+1]=-1;
        }
    }

    //遍历所有的集合并且重新记录终止状态
    for(int i=0;i<stateSet.size();i++){
        cur=stateSet[i];
        //如果当前集合包含终止状态，则将当前集合记录为终止集合
    }
}
```

DFA 最小化为 minDFA 使用等价划分算法：

- 将 DFA 的所有状态划分为两个集合：不包含接受状态的集合 T1，包含接受状态的集合 T2；以这两个为基础，进行集合划分
- 对于现有状态集合中任意一个状态集合 T，对于一个输入符号 a。新建状态集合数组 U，数组对应长度为当前 DFA 状态个数。设立 flag 标识，为 0 表示不需要再划分
- 记录 T 中第一个状态经过 a 到达的状态 x。依次检查 T 中第二个、第三个状态...；如果第 i 个状态经过 a 到的状态不为 x 而为 y，那么表示当前划分集合仍需要划分，将 flag 设置为 1，同时将第 i 个状态从状态集合 T 中删除。同时将状态 i 加入一个 y 对应的 U 数组中的集合。如果第 j 个状态进过 a 到达的状态也为 y，就将状态 j 也加入到 y 对应的 U 数组中的集合
- 将新产生的划分 U 加入到原来的划分 T 中

- 回到第二步执行，直到 flag 为 0，说明不需要再进行划分，minDFA 状态集合构造完毕

给出伪代码如下：

```
void minDFA(){
    初始化分为接受状态组和非接受状态组
    对于当前每一组进行划分
    while(flag 为 1,仍有组需要划分){
        flag = 0;
        数组 goalSet（与当前 minDFA 分组数目相同）记录组别对应产生的新组
            for(对于当前分组中的所有状态){
                for(对于每一个输入符号){
                    if(i 为组内第一个状态){
                        记录 i 经过符号 j 到达状态组 T
                    }
                    else if(如果 i 经过符号 j 到达的状态组为 Q){
                        flag=1;
                        将 i 从状态组 T 中删除
                        将 i 加入到 Q 对应的 goalSet 中
                        }
                    }
                }
                将 goalSet 中不为空的组并入旧组
                当前 minDFA 状态数目等于原本状态数目+新状态数目
            }
        划分结束，构造状态转换表
    }
}
```

3) 细节实现说明

- 使用二元组的形式分析词法类别和种别码：

```
typedef struct //用二元组的形式分析
{
    int typenum; //单词种别: KW,OP,SE,INT,STRING,FLOAT
    char * word;
}WORD;
```

- 通过 p_input 指针定位读入，每次读取的字符为 ch，扫描到一个词后，根据具体读入状况判别种别码 typenum，并且得到 tokens 返回，每次读取字符通过 m_getch()实现：

```
char m_getch()
{
    ch = input[p_input];
    p_input++;
    return ch;
}
```

- 忽略空白符号通过 getbc()实现：

```
void getbc()
{
    while (ch == ' ' || ch == 10)
    {
        ch = input[p_input];
        p_input++;
    }
}
```

- 识别到连续字符时，需要拼接通过 concat()实现：

```
void concat()
{
    token[p_token] = ch;
    p_token++;
    token[p_token] = '\0';
}
```

- 回退一个字符通过 retract()实现：

```
void retract()
{
    p_input--;
}
```

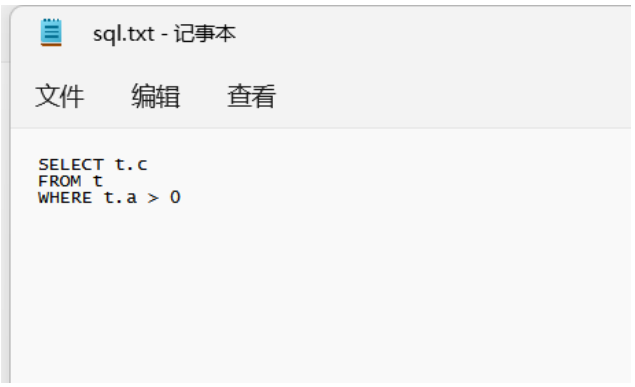
- 设置 KW 的种别码为 1-32；OP 的种别码为 35-50，SE 的种别码为 51-53；特殊的 FLOAT、INT 和 STRING 分别为 55、56、57

3.2 输出格式说明

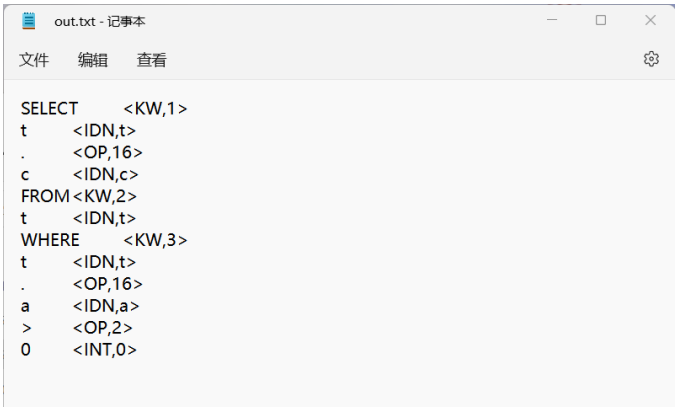
通过编写的词法分析器读入含有.sql 测试文件，并将其词法分析结果输出为语法分析器可接受的序列，写为 out.tsv 文件，输出文件地址在源文件根目录目录下，输出序列组成为：

[输入内容中的单词符号] [空格] <[词语或符号的类别],[词语或符号的内容]>

例如，原输入测试 sql 文件为：



经词法编译器编译后得到的输出样式为：



（二）语法分析器

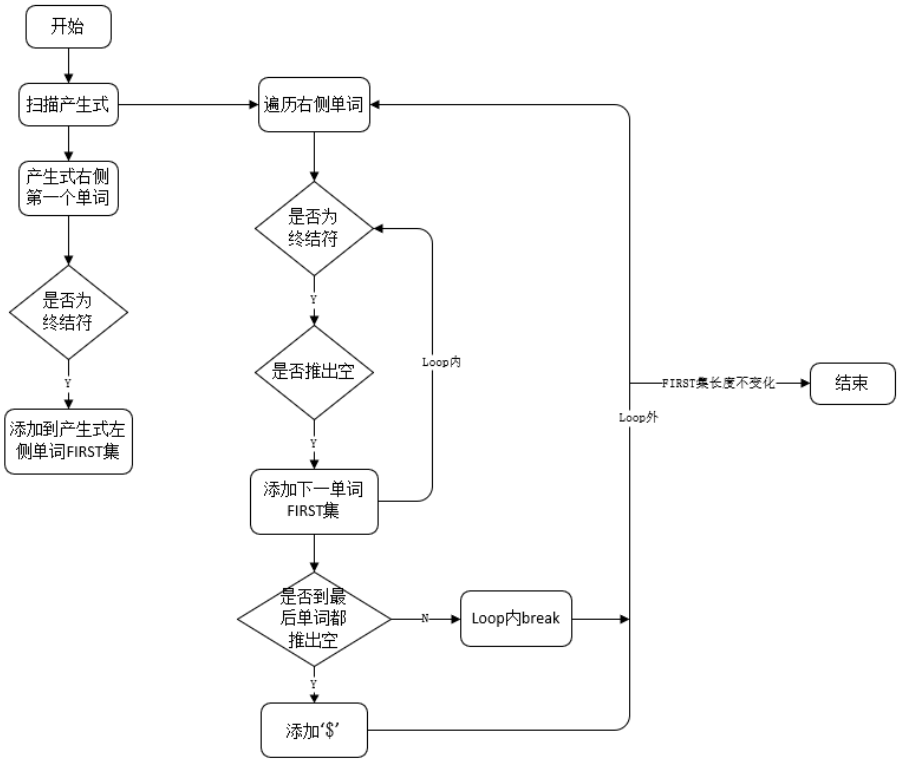
3.3 输入文法

文法要求：

- 从文件读入，每条产生式占用一行
- 文法为 LL(1)文法

从文件中读入文法，从键盘上输入待分析的符号串，采用 LL(1)分析算法判断该符号串是否为该文法的句子。

3.4 求 FIRST 集

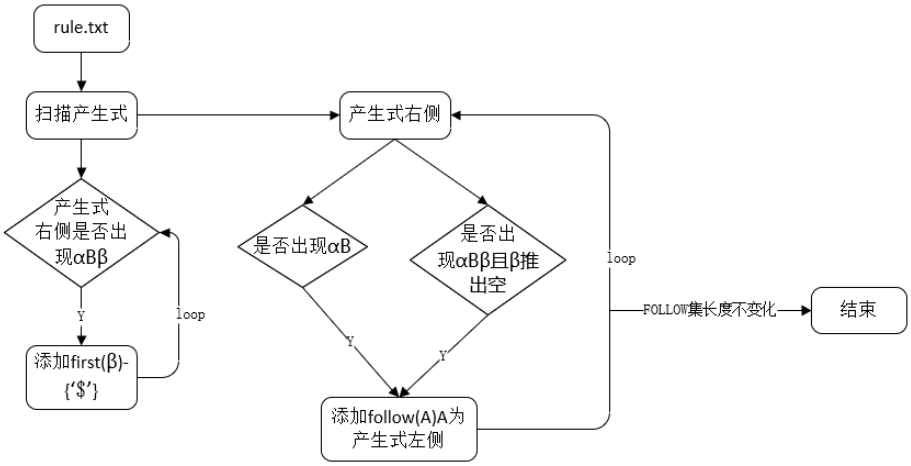


如果产生式右部第一个字符为终结符，则将其计入左部 FIRST 集。对所有产生式扫描 n+1 次，第一次构建只包含终结符的 FIRST 集，随后 n 次利用第一次扫描构造出的结果按照规则扩充 FIRST 集，直到所有非终结符的 FIRST 集都不再增大。

计算方法如下：

1) 若 X-> ε ,添加 ε 到 FIRST（X）中
2) 若 X->a…，添加 a 到 FIRST（X）中
3) 若 X->Y，则将 FIRST（Y）/ ε 添加到 FIRST（X）中
4) 若 X->Y1Y2Y2…Yk，且 Y1Y2Y…Yi-1 的 FIRST 集合均包含 ε，则将 FIRST（Yi）的所有非 ε 添加到 FIRST（X）中

3.5 求 FOLLOW 集

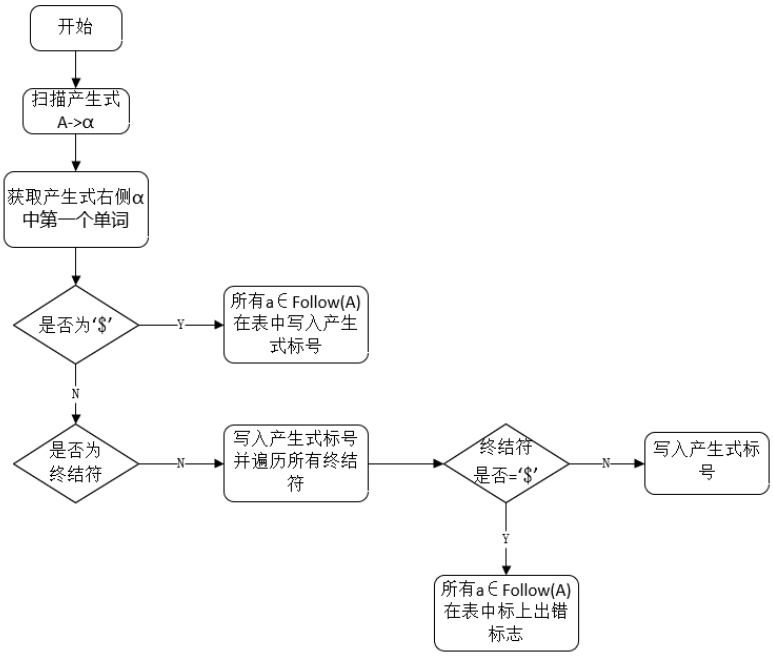


对所有的产生式扫描 n+1 次，首先通过构造好的 FIRST 集构造基础集合，然后 n 次利用第一次扫描构造出的结果按照规则扩充 FOLLOW 集，直到所有非终结符的 FOLLOW 集都不再增大。

计算 FOLLOW 集方法如下：

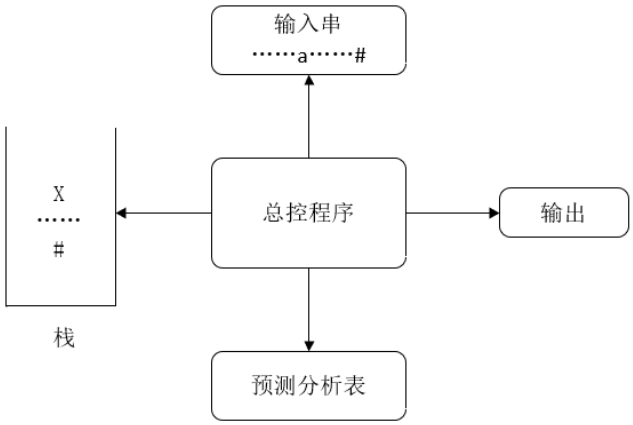
1) 若 X 为文法开始符，则将#（结束符）添加到 FOLLOW（X）中
2) 若有 A->αBβ，则将 FIRST（β）/ε 加入到 FOLLOW（B）中
3) 若 A->αB 或者 A->αBβ 且 ε 属于 FIRST（β），则将 FOLLOW（A）添加到 FOLLOW（B）中

3.6 构建预测分析表



遍历一次所有的产生式 A->α，根据构造预测分析表的规则将产生式的编号放入表中。对文法 G 的每个产生式 A->a 执行接下来两步：对每个终结符 a∈FIRST(a)，把 A->a 加至 M[A,a]中；若 ε ∈FIRST(a)，则把任何 b∈FOLLOW(A)把 A->a 加至 M[A,b]中。把所有无定义的 M[A,a]标上出错标志。

3.7 预测算法



1) 首先输入 token.txt
2) 预测算法的总控程序在任何时候都是按 STACK 栈顶符号 X 和当前的输入符号行事的，对于任何(X,a)，总控程序每次都执行下述三种可能的动作之一：

- 若 X=a=”#”，则宣布分析成功，停止分析过程。
- 若 X=a≠”#”，则把 X 从 STACK 栈顶逐出，让 a 指向下一个输入符号。
- 若 X 是一个非终结符，则查看分析表 M，若 M[A,a]中存放着关于 X 的一个产生式，则首先把 X 逐出 STACK 栈顶，然后把产生式的右部符号串按反序一一推进 STACK 栈(若右部符号为 ε，则意味着不推什么东西进栈)。在把产生式的右部符号推进栈的同时应做这个产生式相应得语义动作，若 M[A,a]中存放着”出错标志”，则调用出错诊察程序 ERROR。

四、测试报告

4.1 词法分析器编译运行步骤

在 lexical_analyzer 文件夹下，

在终端输入 g++ Lexical_Analyzer.cpp -o Lexical_Analyzer.exe

编译之后输入 ./Lexical_Analyzer.exe

4.2 语法分析器编译运行步骤

在 syntactic_analyzer 文件夹下进入终端，输入

python Syntactic_Analyzer.py

4.3 测试结果

小组成员使用提供的测试文件 test0A.sql 与 test0B.sql 进行测试，测试结果如下：

测试用例	词法编译器编译结果	语法分析器编译结果
test0A.sql	0lexA.tsv - 记事本 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H) INSERT <KW,6> INTO <KW,7> _tb1 <IDN,_tb1> VALUES <KW,8> (<SE,1> 1 <INT,1> , <SE,3> 1.78 <FLOAT,1.78> , <SE,3> "SELECT" <STRING,SELECT>) <SE,2>	0graA.tsv - 记事本 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H) 26 125 expressionOrDefaultListRec#, reduction 27 / ,#, move 28 127 expressionOrDefault#FLOAT reduction 29 58 expression#FLOAT reduction 30 72 predicate#FLOAT reduction 31 75 expressionAtom#FLOAT reduction 32 80 constant#FLOAT reduction 33 83 decimalLiteral#FLOAT reduction 34 / FLOAT#FLOAT move 35 74 predicateRight#, reduction 36 61 expressionRight#, reduction 37 125 expressionOrDefaultListRec#, reduction 38 / ,#, move 39 127 expressionOrDefault#STRING reduction 40 58 expression#STRING reduction 41 72 predicate#STRING reduction 42 75 expressionAtom#STRING reduction 43 79 constant#STRING reduction 44 97 stringLiteral#STRING reduction 45 / STRING#STRING move 46 74 predicateRight#) reduction 47 61 expressionRight#) reduction 48 126 expressionOrDefaultListRec#) reduction 49 /)#) move 50 122 expressionsWithDefaultsListRec## reduction 51 / # accept
test0B.sql	0lexB.tsv - 记事本 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H) SELECT <KW,1> from_ <IDN,from_> . <OP,16> _1_ <IDN,_1_> , <SE,3> SUM <KW,21> (<SE,1> from_ <IDN,from_> . <OP,16> _2_ <IDN,_2_>) <SE,2> FROM <KW,2> from_ <IDN,from_> JOIN <KW,14> _1A <IDN,_1A> ON <KW,17> from_ <IDN,from_> . <OP,16> _1_ <IDN,_1_> = <OP,1> _1A <IDN,_1A> . <OP,16>	0graB.tsv - 记事本 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H) 166 22 havingClause#HAVING reduction 167 / HAVING#HAVING move 168 58 expression#IDN reduction 169 72 predicate#IDN reduction 170 76 expressionAtom#IDN reduction 171 49 fullColumnName#IDN reduction 172 48 uid#IDN reduction 173 / IDN#IDN move 174 50 dottedId#. reduction 175 / .#. move 176 52 dottedIdOrStar#IDN reduction 177 48 uid#IDN reduction 178 / IDN#IDN move 179 73 predicateRight#= reduction 180 85 comparisonOperator#= reduction 181 / =#= move 182 72 predicate#STRING reduction 183 75 expressionAtom#STRING reduction 184 79 constant#STRING reduction 185 97 stringLiteral#STRING reduction 186 / STRING#STRING move 187 74 predicateRight## reduction 188 61 expressionRight## reduction 189 25 orderByClause## reduction 190 8 unionStatements## reduction 191 / ### accept

4.4 其他输出文件

此外，在语法分析器中，我们还输出了以下内容：

4.4.1 FIRST 集

FIRST.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
root=[SELECT,(,DELETE,INSERT,UPDATE]
dmlStatement=[DELETE,INSERT,SELECT,UPDATE,(]
selectStatement=[SELECT,(]
unionStatements=[UNION,$]
unionStatement=[UNION]
unionStatementKey=[UNION]
unionStatementQuery=[SELECT,(]
unionType=[DISTINCT,ALL,$]
querySpecification=[SELECT,(]
selectClause=[$,GROUP,ORDER,HAVING,FROM]
fromClause=[FROM,$]
groupByClause=[$,GROUP]
havingClause=[HAVING,$]
orderByClause=[ORDER,$]
selectElements=[SUM,MIN,MAX,IDN,*,AVG]
selectElementHead=[SUM,MIN,MAX,IDN,*,AVG]
selectElementListRec=[,,$]
selectElement=[SUM,MIN,MAX,IDN,AVG]
elementNameAlias=[IDN,$,AS]
tableSources=[IDN,(]
tableSourceListRec=[,,$]
tableSource=[IDN,(]
```

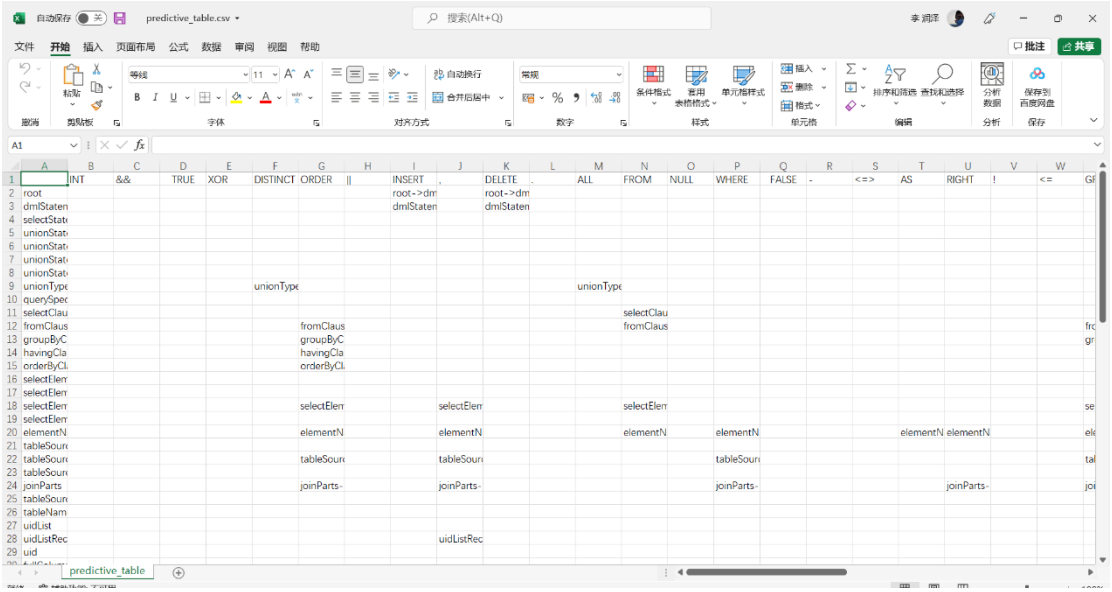
4.4.2 FOLLOW 集

FOLLOW.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

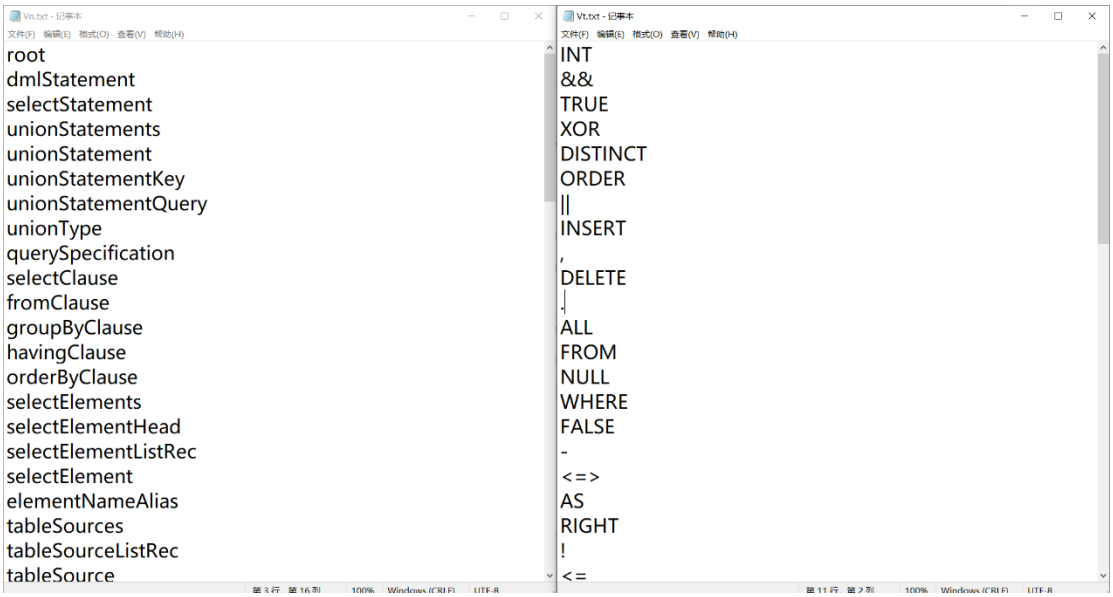
```
root=[#]
dmlStatement=[#]
selectStatement=[#]
unionStatements=[#]
unionStatement=[UNION,#]
unionStatementKey=[SELECT,(]
unionStatementQuery=[UNION,#]
unionType=[SELECT,SUM,MIN,(,MAX,IDN,*,AVG]
querySpecification=[),UNION,#]
selectClause=[),UNION,#]
fromClause=[ORDER,),HAVING,GROUP,UNION,#]
groupByClause=[),#,UNION,ORDER,HAVING]
havingClause=[),UNION,ORDER,#]
orderByClause=[),UNION,#]
selectElements=[ORDER,FROM,),GROUP,#,UNION,HAVING]
selectElementHead=[ORDER,FROM,),,,,HAVING,GROUP,UNION,#]
selectElementListRec=[),HAVING,GROUP,#,UNION,ORDER,FROM]
selectElement=[ORDER,),,,,HAVING,GROUP,#,UNION,FROM]
elementNameAlias=[JOIN,ORDER,,,LEFT,UNION,FROM,#,WHERE,ON,RIGHT,),GROUP,SET,HAVING]
tableSources=[ORDER,),HAVING,GROUP,UNION,#,WHERE]
tableSourceListRec=[),GROUP,#,UNION,ORDER,HAVING,WHERE]
tableSource=[ORDER,),,,,GROUP,#,UNION,HAVING,WHERE]
```

4.4.3 预测分析表



	INT	&&	TRUE	XOR	DISTINCT	ORDER	||	INSERT	,	DELETE	.	ALL	FROM	NULL	WHERE	FALSE	-	<=>	AS	RIGHT	!	<=
1																						
2	root							root->dm		root->dm												
3	dmlStaten							dmlStaten		dmlStaten												
4	selectStat																					
5	unionStat																					
6	unionStat																					
7	unionStat																					
8	unionStat																					
9	unionTyp				unionTyp							unionTyp										
10	querySpec																					
11	selectClau												selectClau									
12	fromClaus					fromClaus							fromClaus									
13	groupByC					groupByC																
14	havingCla					havingCla																
15	orderByCl					orderBycl																
16	selectElem																					
17	selectElem																					
18	selectElem					selectElem			selectElem				selectElem									
19	selectElem																					
20	elementN					elementN			elementN				elementN		elementN				elementN	elementN		
21	tableSour																					
22	tableSour					tableSour			tableSour						tableSour							
23	tableSour																					
24	joinParts					joinParts-			joinParts-						joinParts-					joinParts-		
25	tableSour																					
26	tableNam																					
27	uidList																					
28	uidListRec								uidListRec													
29	uid																					

4.4.4 终结符列表 Vt 与非终结符列表 Vn



Vn.txt - 记事本

```
root
dmlStatement
selectStatement
unionStatements
unionStatement
unionStatementKey
unionStatementQuery
unionType
querySpecification
selectClause
fromClause
groupByClause
havingClause
orderByClause
selectElements
selectElementHead
selectElementListRec
selectElement
elementNameAlias
tableSources
tableSourceListRec
tableSource
```

Vt.txt - 记事本

```
INT
&&
TRUE
XOR
DISTINCT
ORDER
||
INSERT
,
DELETE
.
ALL
FROM
NULL
WHERE
FALSE
-
<=>
AS
RIGHT
!
<=
```