词法分析器报告

词法分析器

1.相关信息

a) 成员及分工

代码开发: 石子跃 3020244294
报告: 石子跃
流程图: 徐心蕙 3020244310
输出说明: 徐心蕙
ppt : 徐心蕙
讲解: 胡豆豆 3020244319
刘玲君 3020244151

b) 开发环境

```
1. PS C:\Users\Ziyueshi\Desktop\大二下\编译原理\NFADFA> g++ -v
2. Using built-in specs.
3.  COLLECT_GCC=/usr/bin/g++
4.  COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-cygwin/11/lto-wrapper.exe
5. Target: x86_64-pc-cygwin
6. Configured with: /mnt/share/cygpkgs/gcc/gcc.x86_64/src/gcc-11.2.0/configure -
    -srcdir=/mnt/share/cygpkgs/gcc/gcc.x86_64/src/gcc-11.2.0   --prefix=/usr   --
    exec-prefix=/usr       --localstatedir=/var       --sysconfdir=/etc       --
    docdir=/usr/share/doc/gcc     --htmldir=/usr/share/doc/gcc/html     -C     --
    build=x86_64-pc-cygwin  --host=x86_64-pc-cygwin  --target=x86_64-pc-cygwin  --
    without-libiconv-prefix  --without-libintl-prefix  --libexecdir=/usr/lib  --
    with-gcc-major-version-only  --enable-shared  --enable-shared-libgcc  --enable-
    static  --enable-version-specific-runtime-libs  --enable-bootstrap  --enable-
    __cxa_atexit --with-dwarf2 --with-tune=generic --disable-bootstrap --enable-
```

```
languages=c,c++,fortran,lto,objc,obj-c++,jit --enable-graphite --enable-
threads=posix --enable-libatomic --enable-libgomp --enable-libquadmath --
enable-libquadmath-support --disable-libssp --enable-libada --disable-symvers
--with-gnu-ld --with-gnu-as --with-cloog-include=/usr/include/cloog-isl --
without-libiconv-prefix --without-libintl-prefix --with-system-zlib --enable-
linker-build-id --with-default-libstdcxx-abi=gcc4-compatible --enable-
libstdcxx-filesystem-ts
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.2.0 (GCC)
```

c) 文件说明

i. N2D.h：规定了 NFA 确定化算法中所用到的数据结构。
ii. N2D_Method.cpp：定义了 NFA 确定化算法中所用到的方法。
iii. N2D_Main.cpp：将读取 regex_rule.txt 中所定义的正则表达式，并进行确定化。
iv. SQLCIFA.h：定义了符号表，Token 三元组和词法分析的方法。
v. SQLCFIFA.cpp：实现了上述各种函数，为本次实验的主体代码部分。
vi. CIFA.cpp：读取 sql.txt 中的 sql 语句进行词法分析。

2.NFA 确定化

a) 正则表达式

——Read_Regex_Line(&now,"regex_rule.txt")函数

regex_rule.txt 中定义正则表达式如下
转换 IDN，STR，FLOAT，INT 类型

```
IDN:(`character`|`_`)(`digit`|`_`|`character`)*
STR:`"`,(`character`|`digit`)*,`"`
INT:(-)?`digit`,`digit`*
FLOAT:(-)?`digit`,`digit`*,`.`,`digit`,`digit`*
```

其中digit 代表 0-9 的数字
Character 代表A-Z, a-z 的英文字母;
？ 代表可选
*代表闭包
|代表或

```
PS C:\Users\Ziyueshi\Desktop\大二下\编译原理\NFADFA> ./a
IDN : (`character`|`_`)(`digit`|`_`|`character`)*
STR : `"`,(`character`|`digit`)*,`"`
INT : (-)?`digit`,`digit`*
FLOAT : (-)?`digit`,`digit`*,`.`,`digit`,`digit`*
```

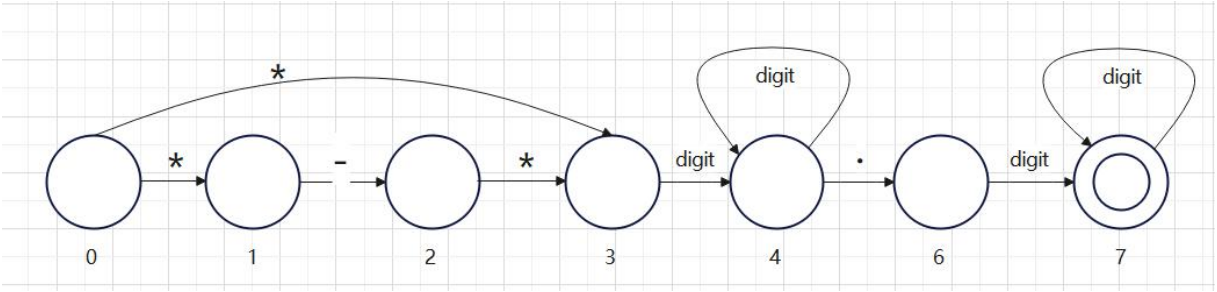
b) 正则表达式转换为NFA

——Build_NFA(&now);函数

以 FLOAT 的 NFA 输出为例：

```
Token: FLOAT 得到的NFA如下
0 -- * --> 1
2 -- * --> 3
0 -- * --> 3
1 -- - --> 2
4 -- . --> 6
4 -- digit --> 4
7 -- digit --> 7
6 -- digit --> 7
3 -- digit --> 4
----------------------------------------
     NFA状态如下
状态: 0 是N
状态: 1 是N
状态: 2 是N
状态: 3 是N
状态: 4 是N
状态: 5 是N
状态: 6 是N
状态: 7 是T
```

根据上述可画出 NFA 图：（0 为开始节点）



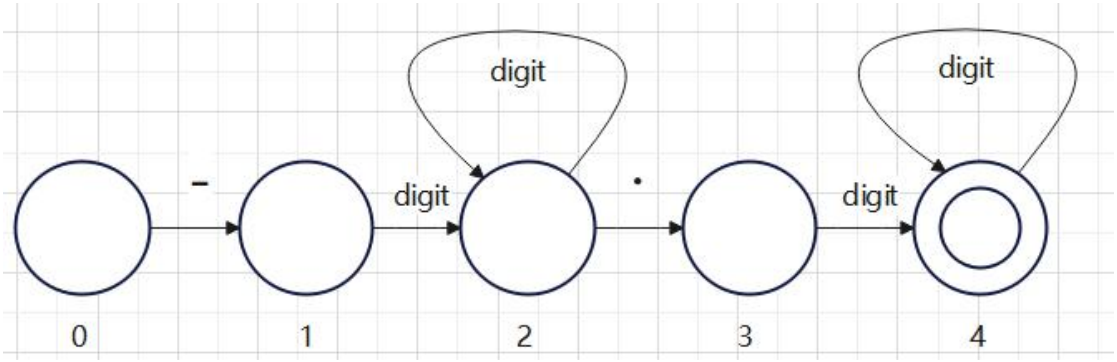
c) **NFA 确定化 DFA**

——NFA2DFA(&now);函数

以 FLOAT 的 DFA 为例

```
Token: FLOAT 得到的DFA如下
0 -- - --> 1
2 -- . --> 3
0 -- digit --> 2
1 -- digit --> 2
2 -- digit --> 2
3 -- digit --> 4
4 -- digit --> 4
----------------------------------------
     DFA状态如下
状态: 0 是N
状态: 1 是N
状态: 2 是N
状态: 3 是N
状态: 4 是T
```

可画出 DFA 图



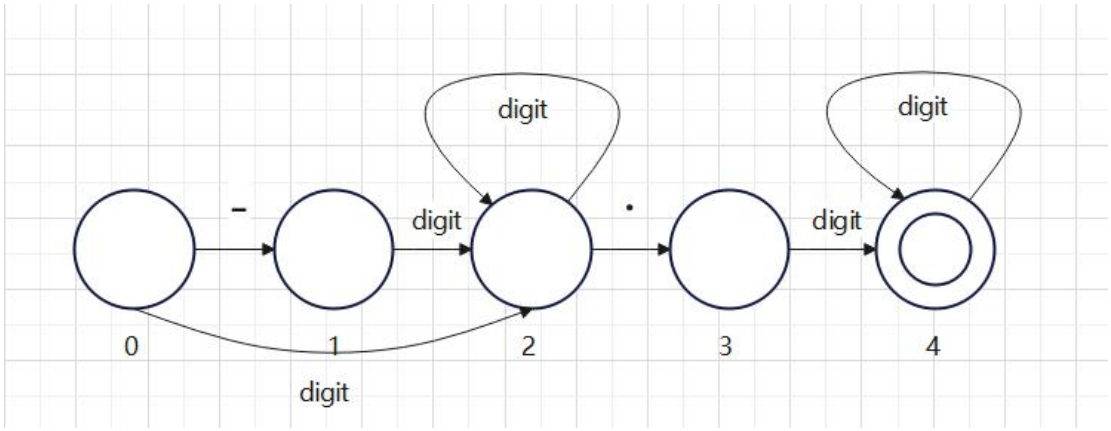
d) **DFA 的最小化**

——Min_DFA(&now);

——Build_MDFA(&now);函数

```
Token: FLOAT 得到的MDFA如下
0 -- - --> 1
2 -- . --> 3
0 -- digit --> 2
1 -- digit --> 2
2 -- digit --> 2
3 -- digit --> 4
4 -- digit --> 4
-----------------------------------------
     MDFA状态如下
状态: 0 是N
状态: 1 是N
状态: 2 是N
状态: 3 是N
状态: 4 是T
```

可画出 FLOAT 的 MDFA：



e) **输出说明**

上文所述环境下，在NFADFA文件夹目录下，打开终端，通过 g++ N2D.h 生成可执行文件 a.exe

随后执行./a.exe 即可

终端中依次输出需要转换的正则表达式，相应的 NFA，相应的 DFA，相应的最小数 DFA。

3.词法分析

a) **算法描述**

```
Sqlprase::ParseMain(void)
{
```

```
vector<Token> returntoken;
while (还未读到末尾)
{
    while (如果当前指向空格向后移动)
        now++;
    if (开头如果是字母，开始寻找标识符（IDN）)
    {注意 KW 可能会识别成 IDN 需要特别识别
     为字母、数字和下划线（_）组成的不以数字开头的串
        在关键字 KW 中可以找到，则识别为 KW
            这里需要注意在 OP 中会出现 AND，OR，XOR 也是 reg 的格
        式，所以要在这里判断returntoken.push_back(ParseIDN());
    }
    else if (开头是数字，则开始寻找整（INT）浮点数（FLOAT）)
    {
        如果中间有.，则更新为 FLOAT
        否 则 为 INT;
         returntoken.push_back(ParseINT());
    }
    else if (如果是 OP 运算符则开始寻找 OP)
    {
        如果><!=.
            继续判断>=,<=,!=
        其他：判断&&，||
         returntoken.push_back(ParseOP());
    }
    else if(如果类型为 SE)
    {
         returntoken.push_back(ParseSE());
    }
    else  if( 如 果 开 头 是 " 则 识 别 为
        STR){ 继续向后寻找，直到遇见下
        一个“ 中间则为 STR;
         returntoken.push_back(ParseSTR());
    }

}
return returntoken;
};
```

注：使用的 DFA 和上文中求得的有一定更改。

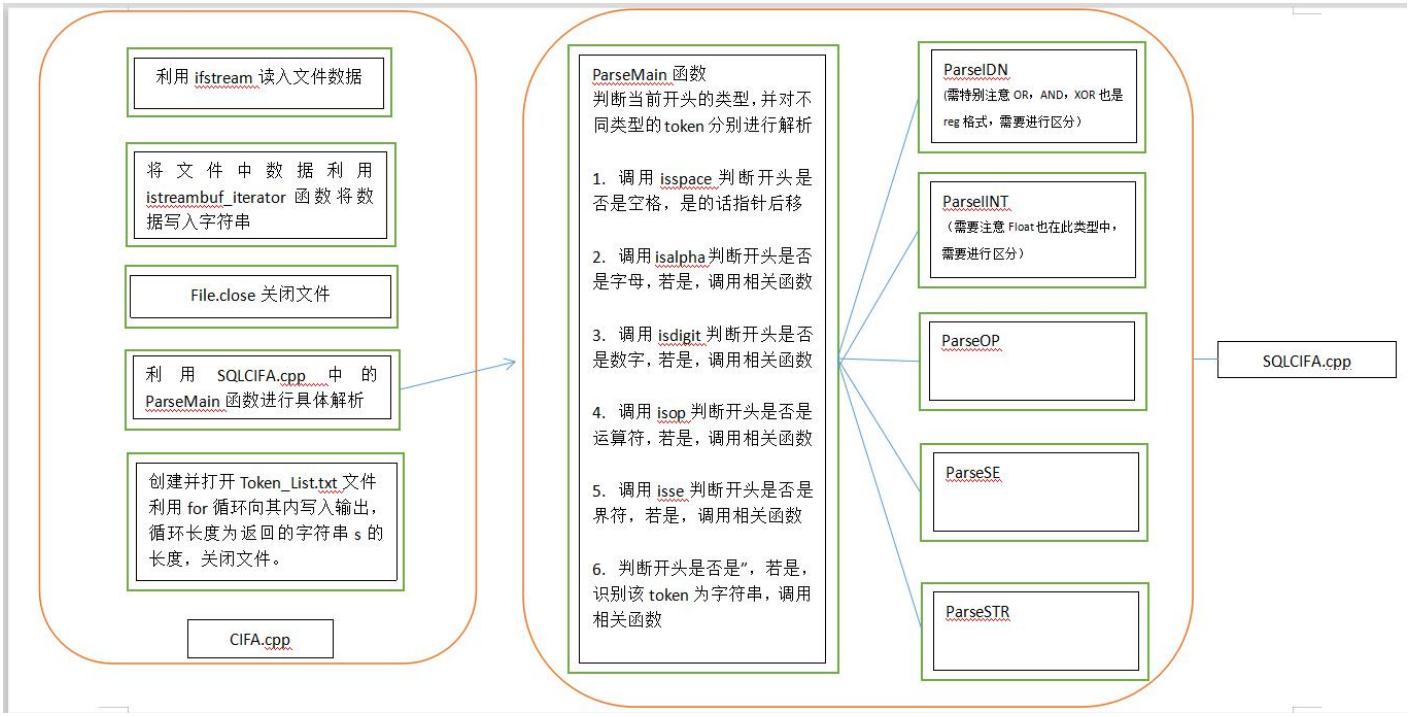


图 1词法分析器流程图

b) 输出格式说明

在 Lexical_Analysis 文件夹下，打开命令端后使用 g++ CIFA.cpp 生成可执行文件 a.exe，在上述环境中，运行./a.exe，即可将 sql.txt 中的 sql 语句进行词法分析输出相应分析结果。

输出要求格式形如[待测代码中的单词符号] [TAB] <[单词符号种别],[单词符号内容]>的输出内容，单词符号种别为 KW（关键字）、OP（运算符）、SE（界符）、IDN（标识符）INT，（整形数）、FLOAT（浮点数）、STR（字符串）；单词符号内容 KW、OP、SE 为其编号其余类型为其值。示例如图2所示，实际输出如图3所示。

```
SELECT  <KW,1>
t   <IDN,t>
.   <OP,13>
c   <IDN,c>
FROM    <KW,2>
t   <IDN,t>
WHERE   <KW,3>
t   <IDN,t>
.   <OP,13>
a   <IDN,a>
>   <OP,2>
0   <INT,0>
```

代码 2 Token 序列示例

图 2输出示例

利用所给的示例（如下），可以输出如图3所示的输出

```
SELECT from_._1_,SUM(from_._2_) FROM from_ JOIN _1A ON
from_._1_=_1A.cr7 WHERE from_._2_>1 AND from_._3_<3.1415926 OR
1.25 IS NOT NULL GROUP BY from_._2_ HAVING from_._3_="ORDER BY
#><=="
```

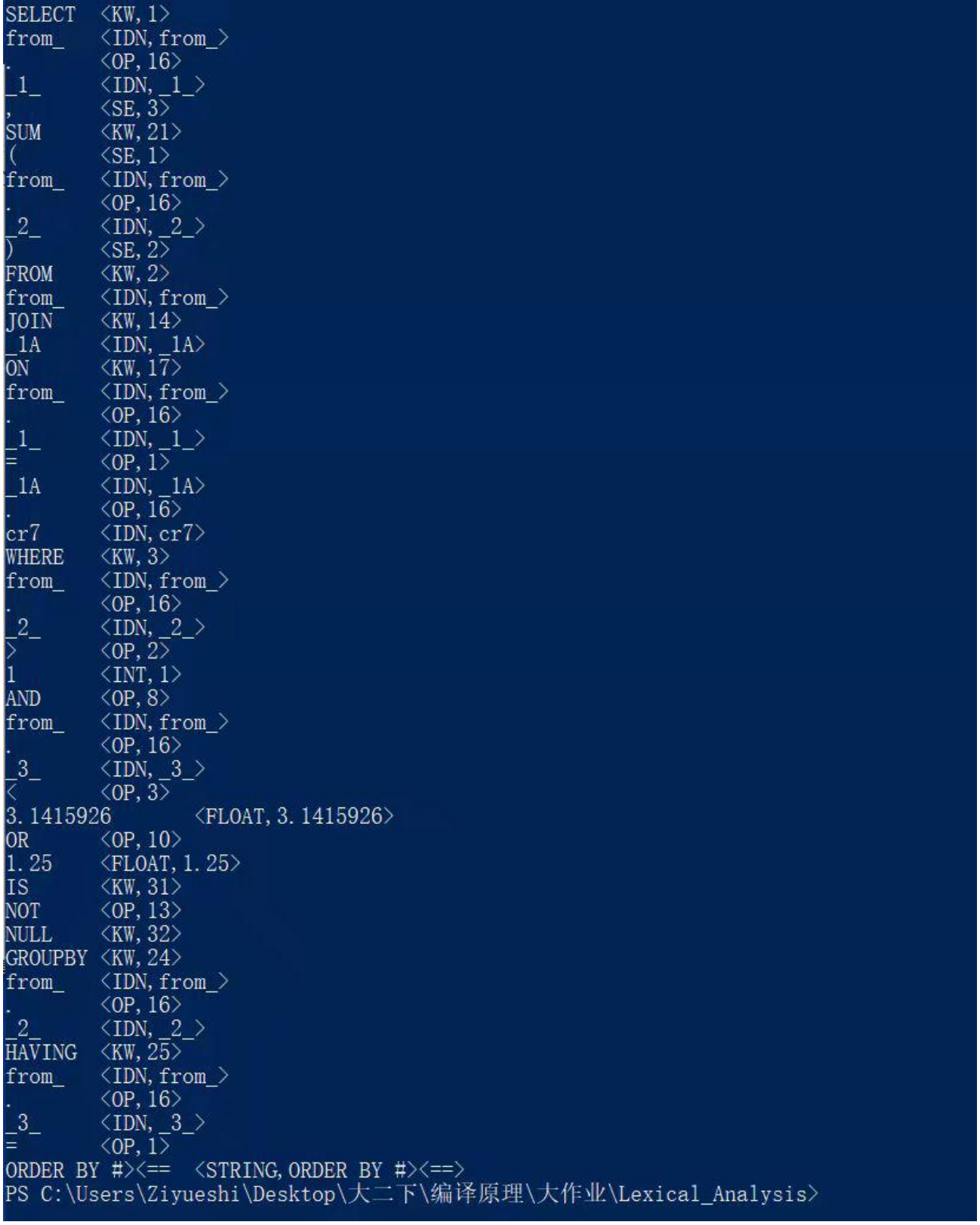


图 3

经分析可得结果正确