南開大學
Nankai University

南　　开　　大　　学

编译系统实验报告

实现词法分析器构造算法

王思宇

年级：2021 级

专业：信息安全

2023 年 10 月 31 日

摘要

本次实验实现词法分析器构造算法：正则表达式->NFA 的 Thompson 构造法、NFA->DFA 的子集构造法、DFA 的最小化算法。

关键字：NFA、DFA、正则表达式、语法分析

目录

一、 正则表达式

(一) 操作数

本程序的支持的操作数为小写字母‘a’-‘z’。

(二) 运算符

正则表达式包含三个运算符
1）或运算符“|”
2）连接运算符“.”，一般省略不写，本程序中用“&”代替
3）闭包运算符“*”，即任意有限次的自重复连接
规定算符的优先顺序为：先“*”，再“.”，最后“|”。

二、 算法流程

首先需要对正则表达式进行预处理。然后将中缀表达式转化为后缀表达式，利用后缀表达式创建 NFA，然后将 NFA 转化为 DFA，最后将 DFA 最小化，

(一) 正则表达式的预处理

预处理是把表达式中省略的连接运算”.” 符加上，方便运算。
需要添加“&”的有六种情况，分别为“a a”、“a (”、“* a”、“* (”、“) a”、“) (”

(二) 中缀表达式转后缀表达式

定义一个数据结构：“栈”来实现该过程，具体操作如下：
1. 如果遇到操作数，直接将其输出。
2. 如果遇到运算符，
(1) 遇到‘(’直接压入栈中
(2) 遇到‘)’将栈中所有运算符出栈，直到遇到‘(’，将‘(’出栈但不输出
(3) 遇到‘*’‘&’‘|’运算符：
a. 如果栈为空，直接将运算符压入栈中；
b. 如果栈不为空，弹出栈中优先级大于等于当前运算符的运算符并输出，再将当前运算符入栈。
3. 当输入串全部读取之后，如果栈不为空，则将栈中的元素全部依次出栈并输出。
本程序为每一个运算符返回一个权值，通过权值的大小来比较优先级。
将‘(’权值设为 0，‘|’权值为 1，‘&’权值为 2，‘*’权值为 3。
具体代码：

优先级

```
int priority(char ch)
{

        if(ch == '*')
        {
                return 3;
```

```
        }

        if(ch == '&')
        {
                return 2;
        }

        if(ch == '|')
        {
                return 1;
        }

        if(ch == '(')
        {
                return 0;
        }
}
```

（三） 后缀表达式创建 NFA

1. 定义一个结构体 struct NfaState 来存储 NFA 的状态
如下图:

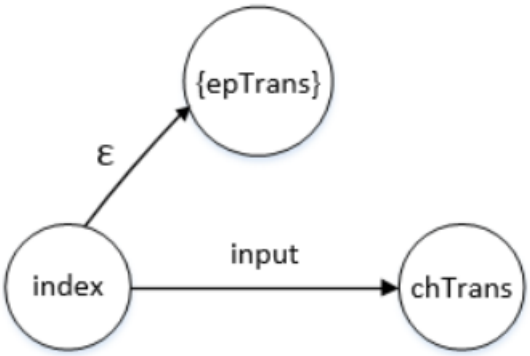

图 1: struct NfaState

index 表示 NFA 状态的状态号
input 表示 NFA 状态弧上的值，默认为‘#’
chTrans 表示 NFA 状态弧转移到的状态号，默认为-1
epTrans 表示当前状态通过 ϵ 转移到的状态号集合
2. 定义了一个结构体 struct NFA 存储 NFA 的结构，主要包括:
NfaState *head：NFA 的头指针
NfaState *tail：NFA 的尾指针
3. 定义了一个 NfaState 类型的数组 NfaStates 和一个 int 类型的全局变量 nfaStateNum，初值为 0，每次需要创建一个 NFA 时就通过 NfaStates[nfaStateNum] 和 NfaStates[nfaStateNum + 1] 从数组中取出两个状态，nfaStateNum 加 2，再更新 NFA 的头尾指针即可。
4. 转化过程需要一个 NFA 的栈，具体操作如下:

按顺序读取后缀表达式，每次读取一个字符

（1）如果遇到操作数 a，则新建一个 NFA，转化弧上的值为 a，将这个 NFA 压入栈中

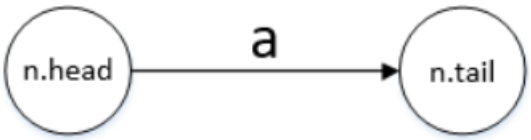

图 2: 操作数 a

（2）如果遇到‘*’，则新建一个 NFA n，从 NFA 栈中弹出一个元素 n1，将 NFA n 压入栈中，连接关系如下:

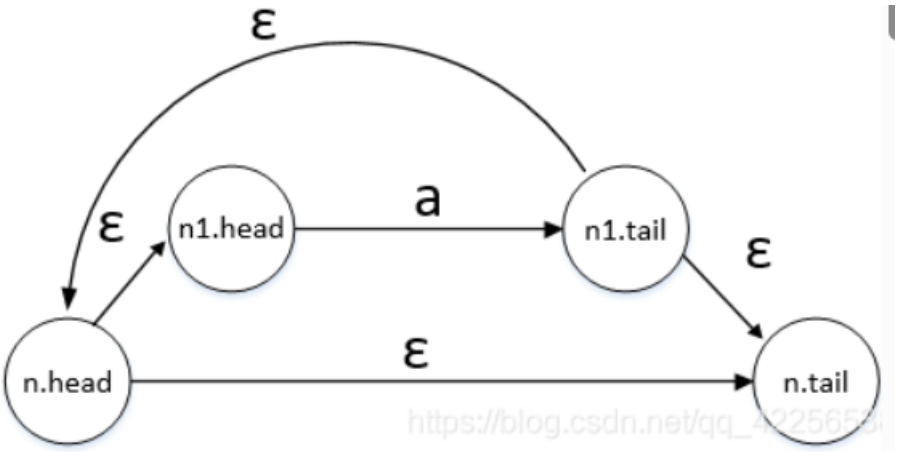

图 3: 运算符 *

（3）如果遇到运算符‘|’，则新建一个 FA n，并在 NFA 栈中弹出两个元素 n1，n2，将 NFA n 压入栈中，连接关系如下:

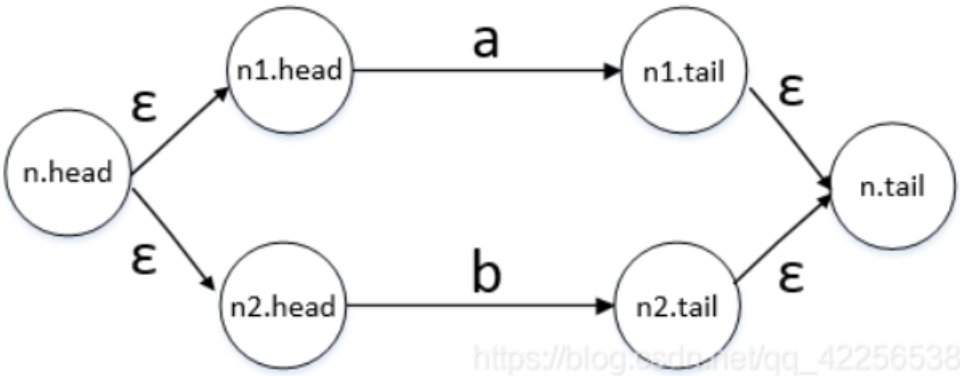

图 4: 运算符 |

（4）如果遇到运算符‘&’，不用新建 FA，只需要在栈中弹出两个元素 n1，n2，改变 n1，n2 的头尾指针，最后将 n 压入栈中，连接关系如下:

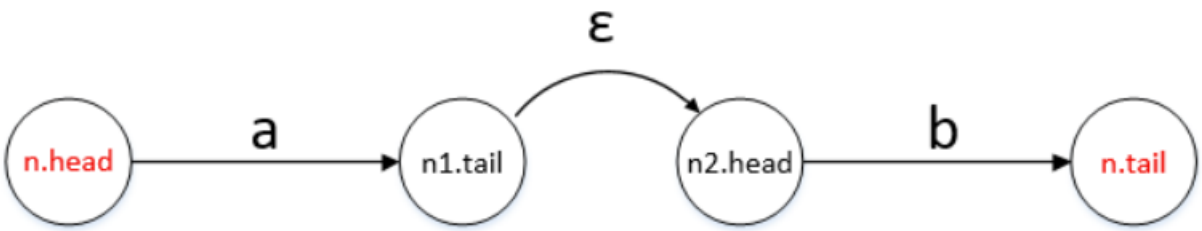

图 5: 运算符 &

（四） NFA 转化为 DFA

1. 定义一个结构体 struct Edge 为 DFA 的转换弧:
input：弧上的值
Trans：弧所指向的状态
2. 定义一个 struct DfaState 存储 DFA 的状态:

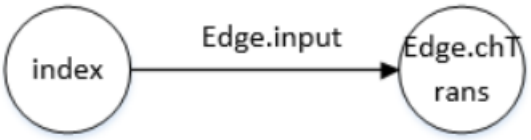

图 6: DfaState

index：DFA 状态的状态号
Edge 类型的数组表示 DFA 状态上的射出弧
3. 定义一个 struct DFA 表示 DFA 的结构:
包括 DFA 的初态，终态集，终结符集，DFA 的转移矩阵。

DFA 结构

```
struct DFA                          /*定义DFA结构*/
{

        int startState;                         /*DFA的初态*/

        set<int> endStates;                     /*DFA的终态集*/
        set<char> terminator;           /*DFA的终结符集*/

        int trans[MAX][26];             /*DFA的转移矩阵*/
};
```

算法过程如下:

NFA 转换为 DFA

```
set s;                  //用于判断集合是否出现过。如果没有出现，则需新建一个
    DFA状态
queue q;

```

```
dfa状态总数 = 0;                          //dfa状态总数

T0 = -closure(0);                  //计算 -closure(0), 令T0 = -closure(0)
s.insert(T0);                      //将子集T0加入子集族s中
q.push(T0);                               //将子集T0入队列

为T0新建一个dfa状态;
dfa状态总数++;                     //dfa状态总数加一

while(!q.empty())
{

        T = q.front();             //出队列
        q.pop();

        for ch in 终结符集         //对每个终结符进行 -closure(move(T, ch))运算
        {

                temp_set = -closure(move(T, ch));

                if(!temp_set.empty())          //如果子集不为空
                {
                        if(!s.count(temp_set))         //如果子集不在s中
                        {

                                为temp_set新建一个dfa状态;

                                s.insert(temp_set);    //将子集temp_set加入
                                    子集族s中
                                q.push(temp_set);      //将子集temp_set入队
                                    列

                                dfa状态总数++;
                        }
                        else    //该状态在子集族s中, 假设标号为T_temp
                        {

                                为当前状态T新建一条弧;

                                //弧上的值为当前终结符
                                弧上的值 = ch;
                                //弧转移到的状态为标T_temp
                                弧转移到的状态 = temp;
                        }
                }
        }
}
```

（五） DFA 最小化

定义一个缓冲区结构体 struct stateSe，用来存储划分集合中的元素和该集合转移到的集合号。[1]

index：该状态集转换到的状态集标号

Intset 集合：该状态集中的 dfa 状态号

如果某个 DFA 状态没有与某个终结符对应的弧，规定此类 DFA 状态转移到的集合号为-1。

判断当前划分集合是否需要进行划分的依据为缓冲区中的元素个数：

如果个数为 1，表示当前划分集合中的所有元素都转移到同一个集合中，则不需要划分。反之，如果个数大于 1，表示当前划分集合中的元素转移到不同集合中，则需要划分。

具体代码:

DFA 最小化

```
set PartSet[128];                    //用于存储所有的划分集合

//将终态和非终态划分开
for state in DfaStates               //遍历DFA状态数组
{
        if(state.isEnd == true)              //如果该DFA状态是终态
        {
                PartSet[0].insert(state);                    //加入到划分集合[0]中
        }
        else                                 //如果不是终态
        {
                PartSet[1].insert(state);                    //加入到划分集合[1]中
        }
}

//实际实现过程中为了遍历划分集合还应判断DFA中是否包含非终态。
//如果有则第一次划分后划分集合个数为2，如果没有则为1。

bool flag = true;                    //上次产生新的划分则为true，反之为false
while(flag)                  //一直循环，直到上次没有产生新的划分
{
        for set in PartSet           //对每个划分集合set
        {

                int cutCount = 0;            //划分次数

                for ch in 终结符集           //遍历每个终结符
                {
                        for state in set                    //遍历集合set中的每个DFA状态state
                        {

                                //判断该DFA状态是否存在与该终结符对应的弧
```

```
33                              bool haveEdge = false;
34
35                              for edge in sate.Edge    //遍历DFA状态sate的每
                                    条边edge
36                              {
37                                      //如果存在某条边的输入为当前终结符
38                                      if(set.state.edge.input == ch)
39                                      {
40                                              //找到该弧转换到的状态所属的
                                                划分集合号
41                                              setNum = findSet(set.state.
                                                    edge.Trans);
42                                              将该state加入到缓冲区中能转换
                                                    到setNum的状态集合中;
43
44                                              haveEdge = true;
45                                              break;
46                                      }
47                              }
48
49                              if(!haveEdge)
50                              {
51                                      将该state加入到缓冲区中能转换到-1的状
                                            态集合中;
52                              }
53                      }
54              }
55
56              if(缓冲区中元素个数 > 1)                    //缓冲区中元素个数大
                    于1则需要划分
57              {
58
59                      cutCount++;             //划分次数+1
60
61                      //这里是从1开始, 因为要将temp[0]中的元素保留在原划分
                            集合中
62                      for(i = 1; i < temp的元素个数; i++)
63                      {
64                              在原划分集合set中删除temp[i]中的元素;
65                              为temp[i]创建新的划分集合;
66                      }
67              }
68      }
69
70      if(cutCount > 0)        //划分次数大于0说明本次产生了新的划分
71      {
72              flag = true;
73      }
```

```
}

for set in PartSet
{
        为set创建一个DfaState;
}
```

至此，DFA 最小化完成。

三、 结果与总结

运行编写完成的代码:
测试样例: "(a*|b)c*"
程序输出结果:

```
中缀表达式为: (a*|b)&c*
后缀表达式为: a*b|c*&
------------------     NFA    ----------------
NFA总共有12个状态,
初态为6, 终态为11。
转移函数为:
0-->'a'-->1
1-->' '-->2     1-->' '-->3
2-->' '-->0     2-->' '-->3
3-->' '-->7
4-->'b'-->5
5-->' '-->7
6-->' '-->2     6-->' '-->4
7-->' '-->10
8-->'c'-->9
9-->' '-->10    9-->' '-->11
10-->' '-->8    10-->' '-->11

---------------     DFA     ----------------
DFA总共有4个状态, 初态为0
有穷字母表为 { a b c }
终态集为 { 0 1 2 3 }
转移函数为:
0-->'a'--><1>   0-->'b'--><2>   0-->'c'--><3>
1-->'a'--><1>   1-->'c'--><3>
2-->'c'--><3>
3-->'c'--><3>

转移矩阵为:
     a   b   c
<0>  1   2   3
<1>  1       3
<2>          3
<3>          3
```

图 7: NFA 和 DFA

图 8: 最小化 DFA

本次实现操作起来对我来说还是非常困难的，在网上查阅了很多资料，最后才将程序完整编写出来。主要是在设计 NFA 和 DFA 的数据结构上，它关系到三个算法。最终主要选取了结构体 struct 存储这些结构。通过对正则表达式转化为 NFA，NFA 转化为 DFA，DFA 最小化这三个过程进行算法编写，对这三个过程更加熟悉了解。但是程序还是存在一些不足，例如：没有处理空字符串：ϵ。所以编写的测试样例中也没有 ϵ。

参考文献

[1] 李阿祥丶. *C++* **实现正则表达式转最小化** *DFA* **过程详解**. 2023.