

## 6. 深入词法分析

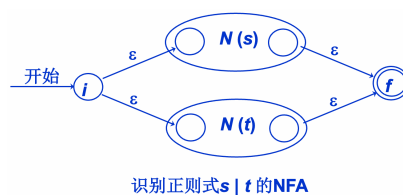
我们在前面的章节中实现了一个简单的面向 TEST 语言的编译器，对构建一个编译器有了初步的认识，后续将逐步深入构建编译器的各个环节。首先深入研究词法分析器，用有限状态自动机实现正则表达式，以能够识别出符合正则表达式的词法单元。实现步骤分三步，第一步：由正则表达式生成非确定自动机 NFA；第二步：非确定自动机 NFA 转化为确定自动机 DFA；第三步：DFA 化简为最简 DFA。

### 6.1 正则表达式 $\rightarrow$ NFA

首先从简单的表达式开始，逐步构建更加复杂的表达式。识别基本表达式的 NFA 为：



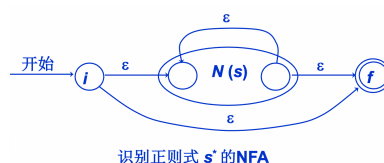
继而构造“或”关系的 NFA：



构造“与”关系的 NFA：



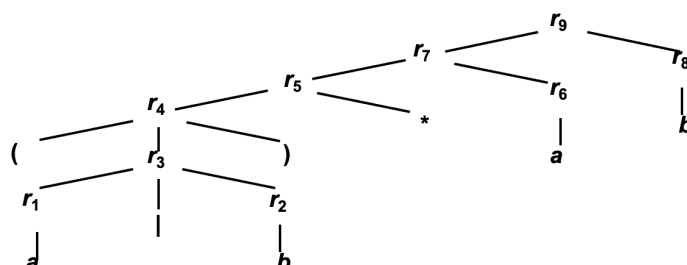
构造闭包表达式的 NFA：



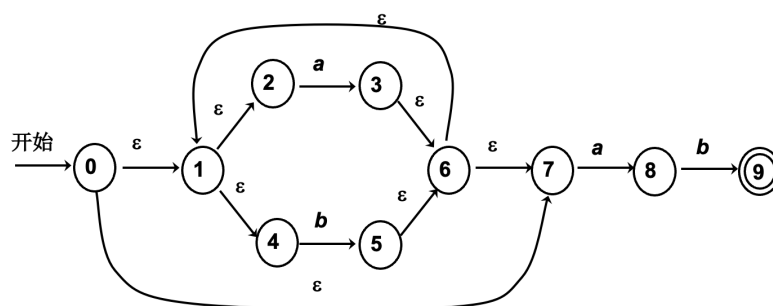
对于加括号的正则式( $s$ )，使用  $N(s)$  本身作为它的 NFA。

当面对一个复杂正则表达式时，首先分析该表达式的结构，例如： $(a|b)^*ab$ ，

其结构如下图：



通过深度优先遍历该树，每个  $r$  节点代表一个自动机，逐步由小的自动机构建出完整的自动机，如下图：



至于如何构造上面这棵树，有兴趣的同学可以用自下向上的语法分析方法，

根据以下语法分析 $(a|b)^*ab$ ：

$r \rightarrow a$   
 $r \rightarrow b$   
 $r \rightarrow r| r$   
 $r \rightarrow ( r )$   
 $r \rightarrow r^*$   
 $r \rightarrow rr$

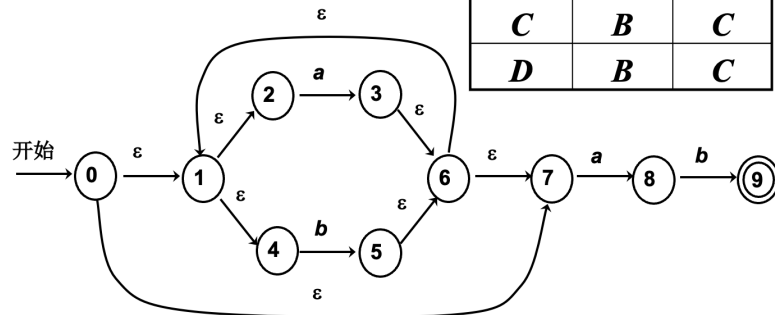
目前只要求能手工画出正则表达式的结构树，然后构造 NFA 即可。

## 6.2 NFA $\rightarrow$ DFA

运用最小子集构造法将 NFA 转化为 DFA，如以上例子的 NFA 将被转化为以下具有 **A**，**B**，**C**，**D** 四个状态的 DFA。

$A = \{0, 1, 2, 4, 7\}$   
 $B = \{1, 2, 3, 4, 6, 7, 8\}$   
 $C = \{1, 2, 4, 5, 6, 7\}$   
 $D = \{1, 2, 4, 5, 6, 7, 9\}$

状态	输入符号	
	<i>a</i>	<i>b</i>
<i>A</i>	<i>B</i>	<i>C</i>
<i>B</i>	<i>B</i>	<i>D</i>
<i>C</i>	<i>B</i>	<i>C</i>
<i>D</i>	<i>B</i>	<i>C</i>



最小子集构造法：首先构造某个状态集合  $T$  的闭包函数

```

把  $T$  的所有状态压入栈；
 $\epsilon$ -closure( $T$ )的初值置为  $T$ ；
while 栈非空
    把栈顶元素  $t$  弹出栈；
    for 每个状态  $u$  (条件是从  $t$  到  $u$  的边上的标记为 $\epsilon$ )
        if  $u$  不在 $\epsilon$ -closure( $T$ )中
            把  $u$  加入 $\epsilon$ -closure( $T$ )；
            把  $u$  压入栈；
        end
    end
end
end

```

然后构造 DFA：

```

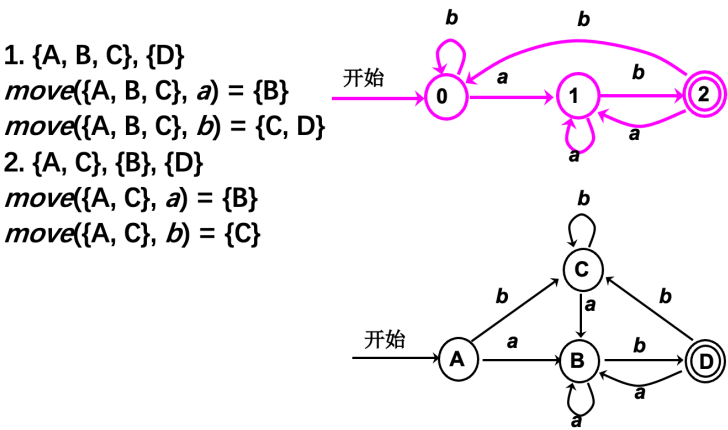
初始， $\epsilon$ -closure( $s_0$ )是  $Dstates$  仅有的状态，并且尚未标记；
while  $Dstates$  有尚未标记的状态  $T$ 
    标记  $T$ ；
    for 每个输入符号  $a$ 
         $U := \epsilon$ -closure(move( $T, a$ )) ；
        if  $U$  不在  $Dstates$  中
            把  $U$  作为尚未标记的状态加入  $Dstates$ ；
         $Dtran[T, a] := U$ ；
    end
end
end

```

## 6.3 DFA 化简为最简 DFA

依据可区别状态进行化简，对于不可区分的多个状态，可将此多个状态化简

为一个状态。化简过程实例如下图，最下方的 DFA 有 4 个状态，经过多个步骤之后，发现状态 A 和 C 不可区分，则将 A、C 化简为一个状态，最终自动机为图中最上方的自动机。



其算法如下：

构造状态集合的初始划分  $\pi$ ：两个子集——接受状态子集 F 和非接受状态子集 S - F

应用下面的过程构造  $\pi_{new}$

最初，令  $\pi_{new} = \pi$

For  $\pi$  中的每个子集 G

把 G 划分为若干子集，使得两个状态 s 和 t 在同一子集中，当且仅当对任意输入符号 a，s 和 t 的 a 转换都到  $\pi$  的同一子集中

在  $\pi_{new}$  中，用 G 的划分代替 G

End

如果  $\pi_{new} = \pi$ ，则  $\pi_{final} = \pi$ ；否则令  $\pi = \pi_{new}$ ，转上步

在  $\pi_{final}$  的每个状态子集中选一个状态代表它，即为最简 DFA 的状态

## 6.4 作业

实现以上算法，并分别测试以正则表达式  $(a|b)^*abb$  以及  $10|(0|11)0^*1$  为输入，

输出各自的最简 DFA，输出形式为 DFA 的状态转移表。