1. **简介**

该正则表达式暂时能识别 \*,|,(,)等特殊符号，如(a|b)\*abc。不过扩展到其他符号（如?）也相对比较容易，修改NFA中的构建规则即可。做这个东西只是想告诉自己很多事做起来并没有想像中难，开始了就离成功不远了。

1. **引擎的构建**

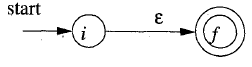
该正则表达式引擎的构建以《Compilers Principles,Techniques & Tools》3.7节为依据，暂时只能识别\*,|,(,)这几个特殊的字符，其工作过程为：构建NFA -> 根据NFA构建DFA -> 用DFA匹配。

1. 构建NFA

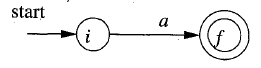
该NFA的构建以2条基本规则和3条组合规则为基础，采用归纳的思想构建而成。

1）2条基本的规则是:

a. 以一个空值ε构建一个NFA

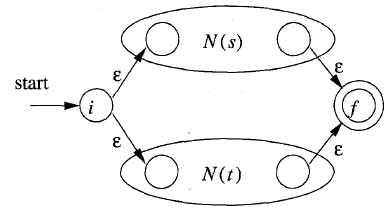


b. 以一个字符a构建一个NFA

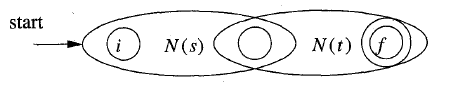


2) 3条组合规则是：

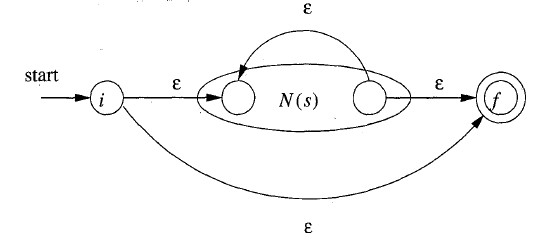
a. r = s | t （其中s和t都是NFA）



b. r = s t（其中s和t都是NFA）



c. r = s \*（其中s为NFA）



3) 如果需要识别如”?”等特殊符号，则可再加一些组合规则。

在具体的程序中，可以以下面的BNF为结构来实现。（具体见源程序regexp.cpp）

r -> r '|' s | r

s -> s t | s

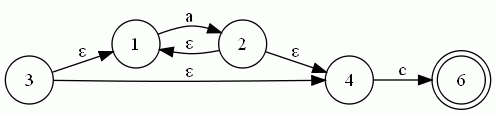
t -> a '\*' | a

a -> token | '(' r ')' |

1. 构建DFA

主要是求ε闭包的过程，从一个集合的ε闭包转移到一个集合的ε闭包。

以a\*c为例，其NFA图如下所示（用dot画的）



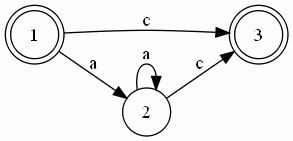
为例:

起始结点3的ε闭包集为 A = {3,1,4}

A遇上字母a的转移为MOV(A,a) = { 2 }，其ε闭包集为B = { 2,1,4 }

A遇上字母c的转移为MOV(A,c) = { 6 }，其ε闭包集为B = { 6 }

同理可求出其他转移集合，最后得到的DFA如下所示:



1. 匹配

每匹配成功一个字符则DFA移动到下个相应的结点。

1. **改进**
2. 如龙书中所说，有时候模拟NFA而不是直接构建DFA可能达到更好的效果。
3. 每次匹配不成功都需要回溯，这个地方也可以借鉴KMP算法（不过KMP对此好像有点不适用）
4. 其他改进方法可以看看《柔性字符串匹配》和龙书《Compilers Principles,Techniques & Tools》3.7节。