一． 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。

二． 引擎的构建

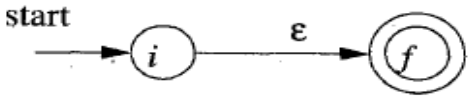
该正则表达式引擎的构建以《Compilers Principles,Techniques & Tools》3.7 节为依据，暂时只能识别*,|,(,)这几个特殊的字符，其工作过程为：构建 NFA -> 根据 NFA 构建 DFA -> 用 DFA 匹配。

1. 构建 NFA

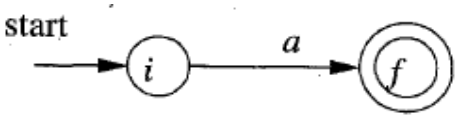
该 NFA 的构建以 2 条基本规则和 3 条组合规则为基础，采用归纳的思想构建而成。

1）2 条基本的规则是:

a. 以一个空值ε 构建一个 NFA

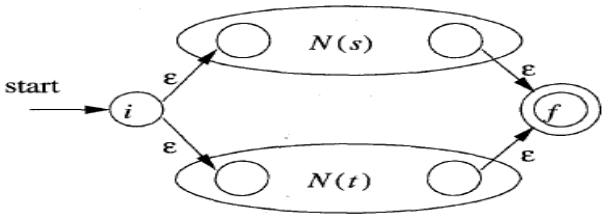


b. 以一个字符 a 构建一个 NFA

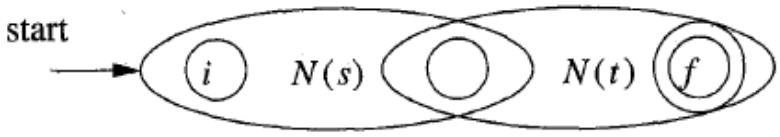


2) 3 条组合规则是：

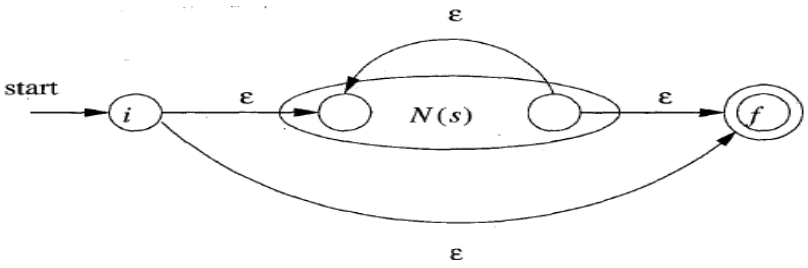
a. r = s | t （其中 s 和 t 都是 NFA）



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。

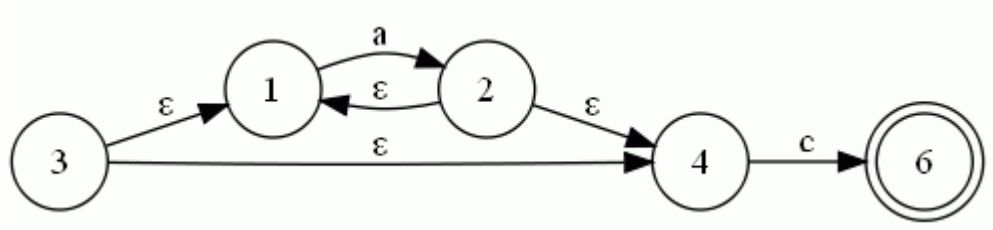
在具体的程序中，可以以下面的 BNF 为结构来实现。（具体见源程序 regexp.cpp）

```
r -> r '|' s | r
s -> s t | s
t -> a '*' | a
a -> token | '(' r ')' |
```

2. 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）



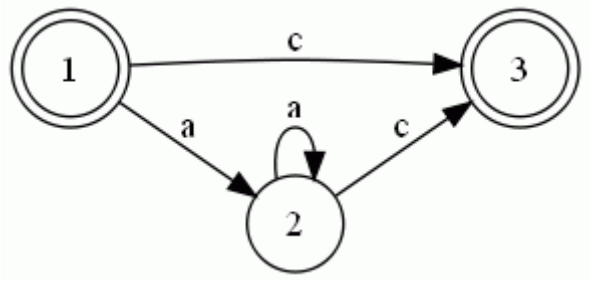
为例:

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:



3. 匹配

每匹配成功一个字符则 DFA 移动到下个相应的结点。

三. 改进

1. 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。
2. 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）
3. 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。