

常规语言和自动机：第1部分

1正则表达式

参见画布上的视频“正则表达式”（16分钟）。

让我们来看看这两个问题吧。

1. 当一个字符串包含至少8个字符和至少2个数字时，它为“有效密码”。给定一个字符串，假设它是否为一个有效的密码。
2. 给定一个字符串(例如。一个文件)，列出其中所有出现的电子邮件地址。每个出现都应该表示为一对数字(i, m)，其中i是起始位置。0表示在文件开头出现的内容)，m为长度。

第一个是一个匹配问题的一个例子。第二个是一个发现的问题。这样的问题——以及变化——经常出现在计算中，所以人们已经制作了工具来有效地解决它们。要使用这样的工具，你必须指定一个单词何时是有效的密码或电子邮件地址或其他什么。通常，我们通过正则表达式来实现这一点。

. 11定义

将调用字母表(字符集)。Σ为了简单起见，假设是ac。在实际情况下，它可能是ASCII字母表，它有128个字符。也可能是Unicode字母表，它有137,439个字符。Σ在任何情况下，我们都将假设这是一个有限的集，并且至少包含两个字符。

我们写Σ*对于所有单词的集合。一种语言是一组单词，i.e.的子集Σ*。例如：有效密码集是一种语言，电子邮件地址集也是如此。一个正则表达式(regex)，如c(bb|ca)*，表示一种语言，就像一个算术表达式，如2+(5-3)，表示一个数字一样。x

现在让我解释一下规则程序是如何工作的。

regexpa只匹配单词a。

regexpb只匹配单词b。

regexpc只匹配单词c。

regexpe只匹配空字e。

如果E和F是regexps，那么regexpEF匹配任何由E匹配的单词和由F匹配的单词的连接单词。

如果E和F是|，则||匹配E或F匹配的任何单词。

如果E是一个回归pp，那么回归pE*匹配任何由几个单词连接起来的单词。由E匹配的单词。

(很少使用的) regexp0不匹配任何单词。

1.2优先权

对于算术表达式，其优先级高于+，知道这一点使我们能够将表达式3+4*2解析为3+(4*2)。xxx对于regexps，优先级规律如下：并置(即“把东西放在一起”)的优先级高于|，优先级低于*。知道这一点就可以我们来分析c(bb|ca)*作为c((bb)|(ca))*例如

更多你可以使用的操作符：

• E^+ 是 EE 的缩写*. 它匹配任何由 E 匹配的一个或多个单词的连接单词。

• $E?$ 是“ $|E$ ”的缩写。

这些基因的优先级与*。

一些工具提供了额外的操作符，这使表达更高级的语言成为可能。使用这些附加操作符的表达式在工具文档中称为“正则表达式”，但技术上它们不是正则的。

示例1 1. $\text{regexpc}(bb/ca)^*$ 匹配ccacabb吗？是/不是。

2. $\text{regexpc}(bb/ca)^*$ 匹配cbbcacac吗？是/不是。

3. $\text{regexpc}(c(bb/ca)^*)^*$ 匹配ccackacbbca吗？是/不是。

4. 做 $(a|b)c^*$ 和 $ac^*|bc^*$ 代表相同的语言？是/不是。

5. 做 $(a/b)c^*$ 和 ac^* 代表相同的语言？是/不是。

2. 关于正则表达式的一些问题

在我们更详细地研究规则程序之前，让我们考虑一些问题。对于其中一些来说，答案远不明显，但将在接下来的讲座中出现。

2.1 有规则的语言和不规则的语言

回想一下，一个 regexpc 代表了一种语言。任何语言 $L \subseteq \Sigma^*$ 可以用这种方式表示的，也就是规则的。问题：

1. 有什么语言是不规则的语言吗？回答：是的，我们会看到一些例子。

2. 一种常规语言的补充总是规则的吗？（例如，对于 $c(bb|ca)$ 不匹配的单词是否有一个规则*？）答：是的。

3. 两种正则语言的交集总是正则的吗？（例如，对于两个 $cc(bb|ca)^*$ 和 $c(bbbb|cca)^*$ ？）答：是的。

2.2 可判定性问题

*决策问题是一个对于任何给定的论证，答案都是/否的问题。*例如，上面的查找问题不是一个决策问题，因为答案（对于一个给定的文件）是一组数字对。一个决策问题被认为是可决定的，当有一些程序，给定一个争论，说明答案是是的还是No. 的

关于计划，我们可以问以下问题：

1. 是回归程序 $c(bb|ca)$ 的匹配问题吗*可决定的换句话说，有没有一些程序，当给定一个单词 w 超过我们的字母表 $=\{a, b, c\}$ ，如果 w 匹配 $c(bb|ca)$ ，返回 True ，如果没有，那也是假的吗？（如果 w 不是我们字母表上的一个单词，那么发生什么并不重要。）回答：是的。

2. 是 regexpc 的匹配问题 $(c(bb|ca))^*$ 可决定的答：是的。

3. 对于每个 regexpc ， E 的匹配问题都是可决定的吗？答：是的。

4. 规则表达式的匹配问题是可决定的吗？换句话说，是否有一些程序，当给定一个 regexpc 和单词 w 时，如果 w 与 E 匹配，则返回 True ，如果它不匹配，则返回 False ？答：是的。

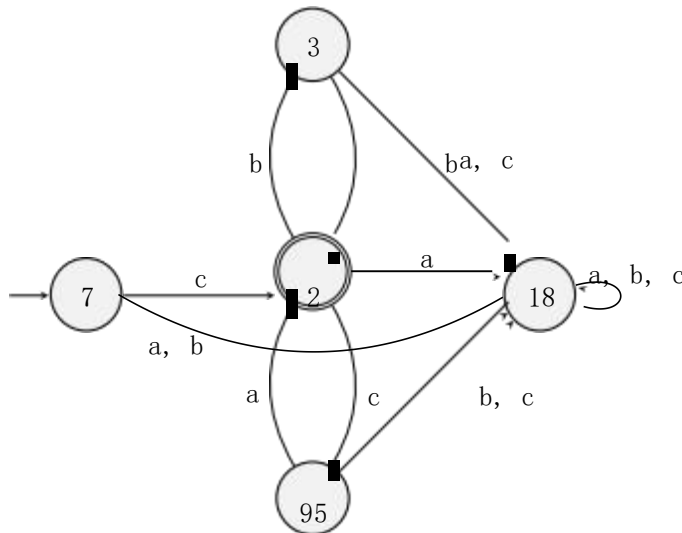
5. 重新规范的语言平等是可以决定的吗？换句话说，当给定 regexps 程序 E 和 F 时，如果它们表示相同的语言，是否会返回 True ，否则就会返回 False ？答：是的。

2.3效率问题

在上一节中，我们询问了某些问题是否完全可以得到解决。另一个问题是：它们能被有效地解决吗？毕竟，你的客户不愿意等待很长时间才能从你的程序中得到答案。这个问题不是很精确，但它很重要。我们将看到，对于其中一些这些问题，我们可以给出一个合理有效的解决方案。

3介绍自动机

参见画布上的视频“自动机介绍”（10分钟）。



回想一下，我们想要一个程序来解决 $c(bb|ca)^*$ 的匹配问题*. 这可以通过如图所示的自动机来实现。有五个状态，用圆圈表示。自动机通过从初始状态开始（由！），并在它输入每个字母时执行一个转换。当输入整个单词时，如果当前状态为接受，自动机返回“是”，用双环表示；否则返回No.

这是一个确定性有限自动机 (DFA)。“确定性”，因为指定的初始状态和每个转换的结果。“有限的”，因为状态集是有限的。

定义1: 一个确定性的有限自动机由以下数据组成。

一个状态的有限集合 X 。

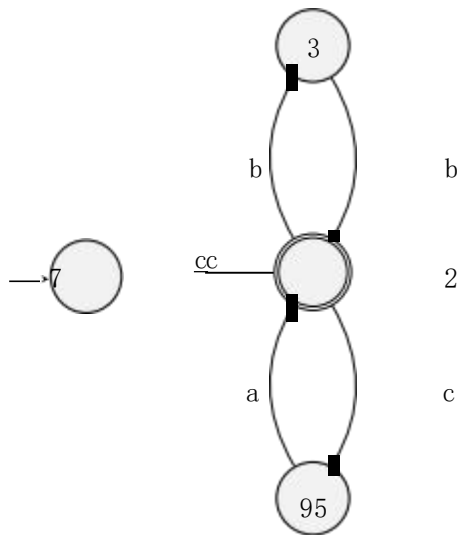
一个初始状态 $p \in X$ 。

一个转换函数 $\delta: X \times \Sigma \rightarrow X$ 。

一组接受状态 $Acc \subseteq X$ 。

在上面的例子中，

$$\begin{aligned} X &= \{7, 2, 3, 95, 18\} \\ p &= 7 \\ \delta &= \{(7, a) \rightarrow 18, (7, b) \rightarrow 18, (7, c) \rightarrow 2, \\ &\quad (2, a) \rightarrow 18, (2, b) \rightarrow 3, (2, c) \rightarrow 95, \\ &\quad (3, a) \rightarrow 18, (3, b) \rightarrow 2, (3, c) \rightarrow 18, \\ &\quad (95, a) \rightarrow 2, (95, b) \rightarrow 18, (95, c) \rightarrow 18, \\ &\quad (18, a) \rightarrow 18, (18, b) \rightarrow 18, (18, c) \rightarrow 18\} \\ Acc &= \{2\} \end{aligned}$$

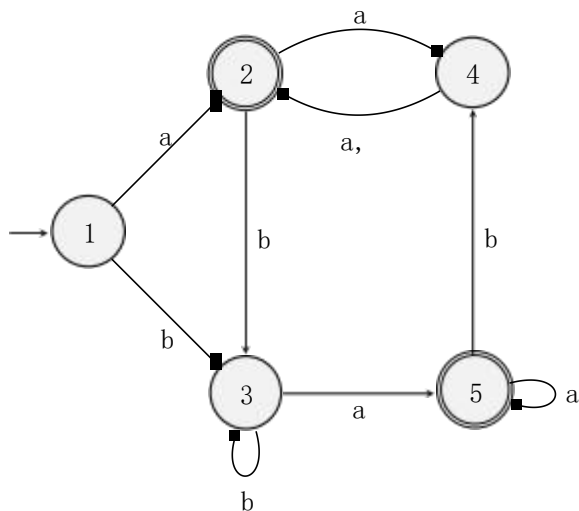


这个自动机比上面的更有效。它是一个部分的DFA，这意味着6仅仅是一个部分的函数，i。e. 它有时可能是不被定义的。一旦不能输入一个字符，这个词就会被拒绝。例如，单词只要用三个字符就会被拒绝。（部分DFA也可以没有初始状态，但是每个单词都会立即被立即拒绝，所以这不是很有用。）

我们可以很容易地将一个部分DFA转换为一个总DFA；只需添加一个额外的不接受状态，称为错误状态（示例中为18）。在部分DFA中未定义的转换将进入错误状态。从错误状态到错误状态都到错误状态。（如果部分DFA没有初始状态，则错误状态将为初始状态。）

在上面的例子中，状态是数字，但实际上它们是什么并不重要。所以，当绘制一个自动机时，保留圆圈是可以的，除非我们想参考一个特定的状态。

示例2这里是字母表上的一个DFA。这些话被接受了吗？



abab	Y/N
ababba	Y/N
ababbbaa	Y/N
bab baa	Y/N
e	Y/N

例子3那么这个DFA呢？

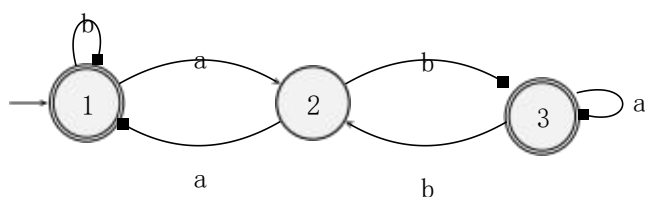
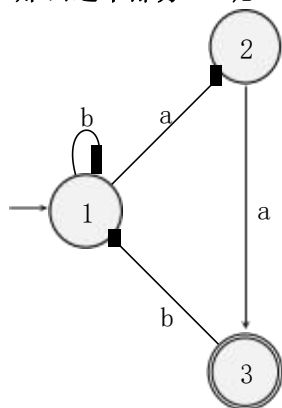


abb Y/N
 abbaba Y/N
 bba Y/N
 e Y/N

示例4那么这个部分DFA呢？

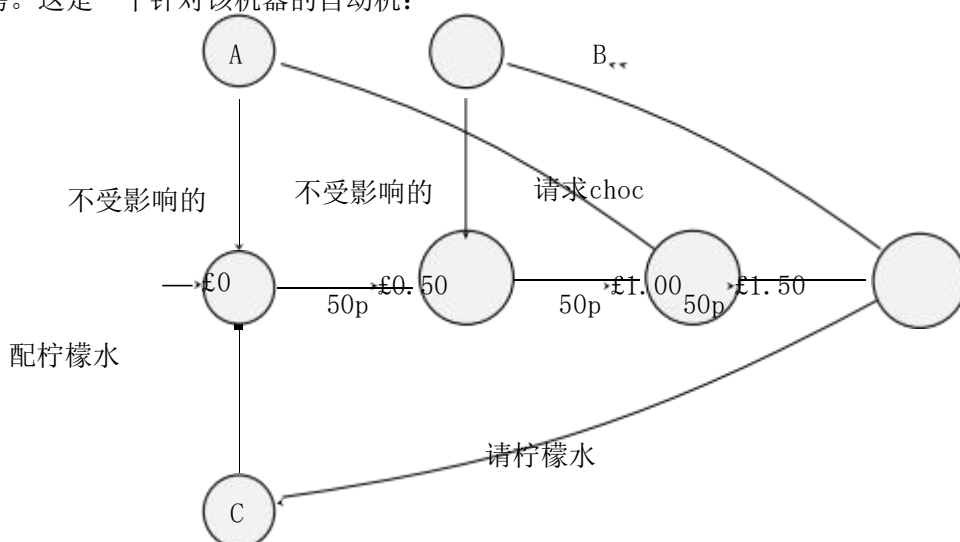


aaab Y/N
 aabaa Y/N
 abaab Y/N
 aa Y/N

3.1 台自动售货机

DFA的概念是为了一个特定的目的而发明的：解决一种语言的匹配问题。DFA所能做的就是输入字母，并说出被读到的单词是否被接受。但对于其他目的，还有其他类型的自动机。

在大厅里有一个自动售货机，可以收到50便士的硬币，最高可达1.5英镑。巧克力要11英镑，柠檬水要1.5英镑。这是一个针对该机器的自动机：

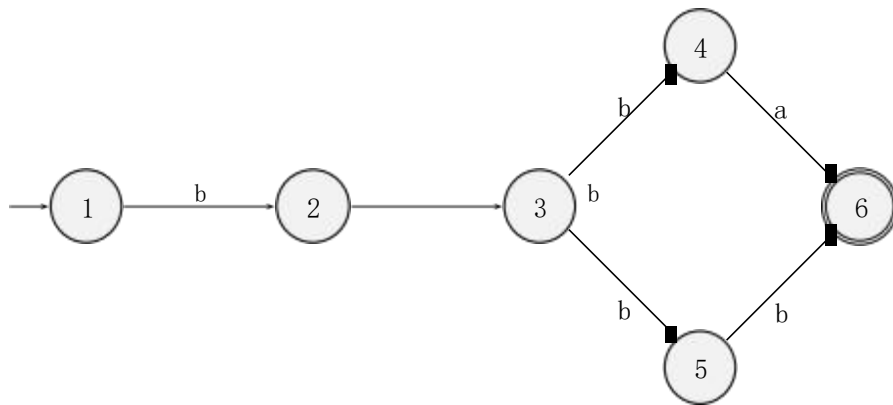


请注意，这个自动机可以输入金钱和请求，也可以输出巧克力和柠檬水。没有接受状态，因为识别单词不是这台机器的目的。

4 非确定性自动机

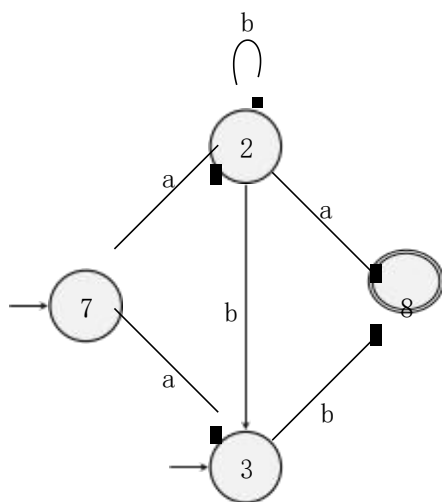
参见画布上的视频“不确定性有限自动机”（6分钟）。

有时很难获得一个regex的DFA，但我们可以更容易地获得一个非确定性有限自动机 (NFA)。下面是一个关于语言bb(ba|bb)的例子。



非确定性有限自动机 (NFA) 与 DFA 有两个不同之处。首先，一个 NFA 可以有多个初始状态。其次，从一个给定的状态中，当输入一个（或任何其他字符）时，可能会有几种可能的下一个状态。因此， δ 是一个关系，而不是一个函数。自动机选择其初始状态，并在输入字符时选择要移动到的状态。当单词 w 是从初始状态到接受状态时，单词 w 是可以接受的。

例子5这是字母表的 NFA。这些话可以接受吗？



abbb Y/N

abb Y/N

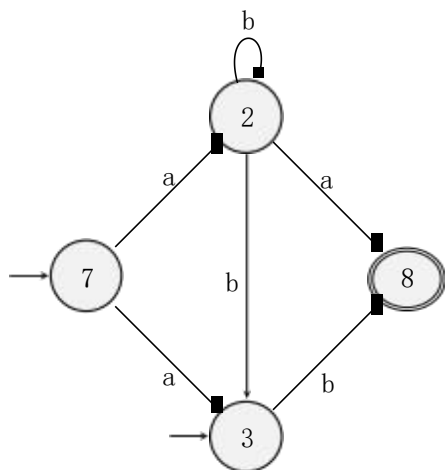
ϵ Y/N

阿巴 Y/N

4.1 确定 NFA：将 NFA 转换为 DFA

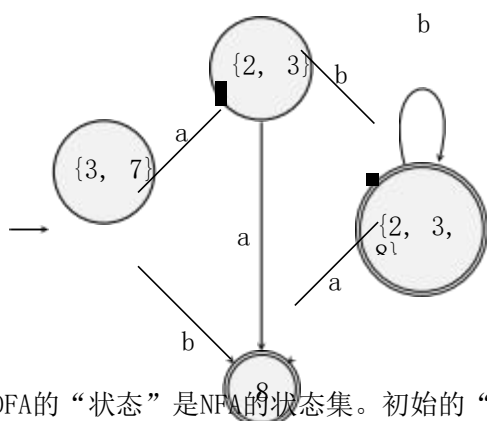
参见画布上的视频“确定一个 NFA”（10 分钟）。

NFA 在实践中是无用的：我们想要一个总是对好词说是，对坏词说是的程序，而 NFA 不会这样做。但我们可以确定它，i. e. 将它转换为识别相同语言的 DFA。要了解这是如何工作的，请查看下面的示例。



让我们想想如何知道abb这个词是否可以接受。我们可以通过尝试来实现，但有一种算法方法：跟踪当前可能的状态集。最初，可能的状态的集合是{3, 7}。输入a后，可能的状态集合为{2, 3}。输入b后，可能的状态集合为{2, 3, 8}。再次输入b后，可能的状态集合为{2, 3, 8}。我们已经到了单词的末尾，我们注意到目前可能的状态之一，即。8、接受。因此，abb这个词是可以接受的。

这个算法实际上给出了以下的DFA：



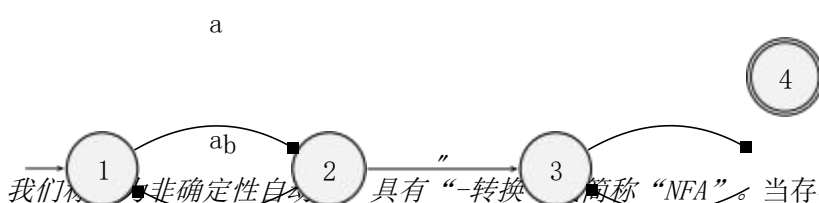
DFA的“状态”是NFA的状态集。初始的“状态”是NFA的所有初始状态的集合。当一个“状态”包含一个NFA的接受状态时，它就表示接受。从一个给定的“状态”中，当我们输入一个字符时，我们会收集所有可能的下一个状态。这个过程被称为确定化。

我们看到一个词w被DFA接受，如果它被NFA接受。因此，它们代表着同一种语言。

5e转换

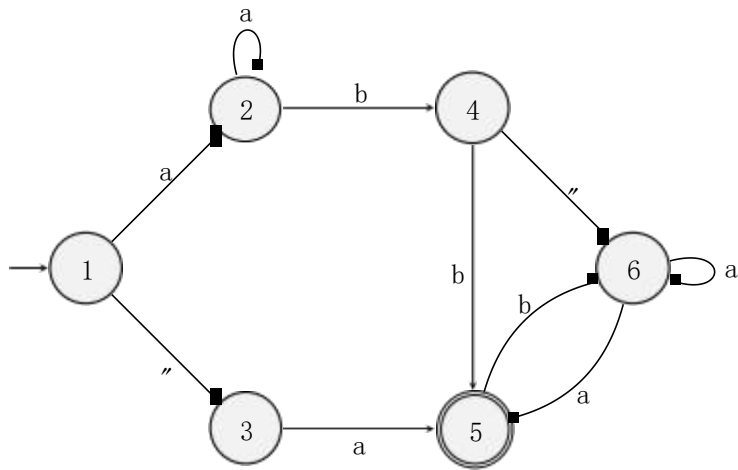
参见画布上的视频“Epsilon过渡”（6分钟）。

有时甚至是一个NFA也很难获得，但我们可以获得一个花一些时间思考的自动机。正如它所认为的那样，它从一种状态移动到另一种状态，而不输入任何字符。这里有一个例子，对于 $\text{regex } (aa)^*b(bb)^*$ 。



我们称这种非确定性自动机为“NFA”。当存在从初始状态到接受状态的路径时，单词w可以通过w的字符，并填充了“转换”。

例子6，这里是一个“NFA”。这些话可以接受吗？



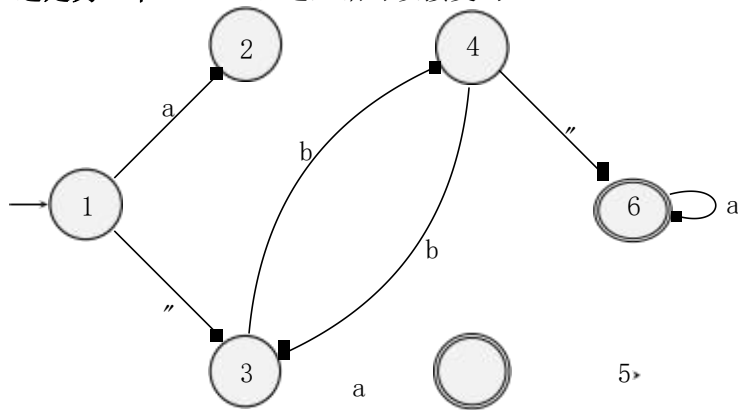
aba Y/N

ab Y/N

aaabb Y/N

a Y/N

示例7这是另一个“NFA”。这些话可以接受吗？



a Y/N

aba Y/N

bbb Y/N

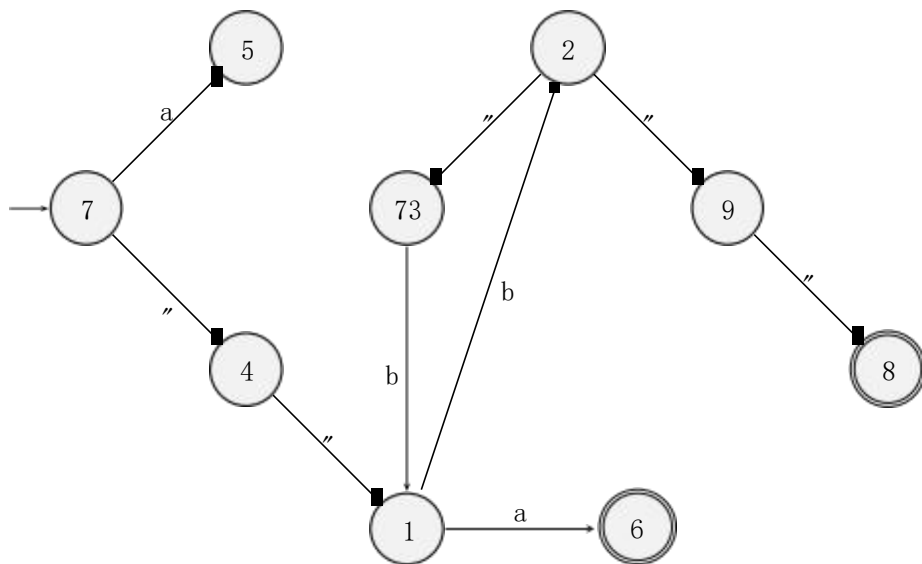
bbba Y/N

ab Y/N

5.1 删除空子集

参见画布上的视频“消除空子集”（9分钟）。

令人高兴的是，它可以从一个NFA中删除“ε-转换”，即。将其转换为能够识别相同语言的NFA。为了了解这个过程的想法，让我们看看下面的“NFA”。



bbb可以接受，因为路径如下：

$7 \xrightarrow{a} 5 \xrightarrow{b} 4 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{b} 9 \xrightarrow{b} 8$

此路径由以下几个部分组成：

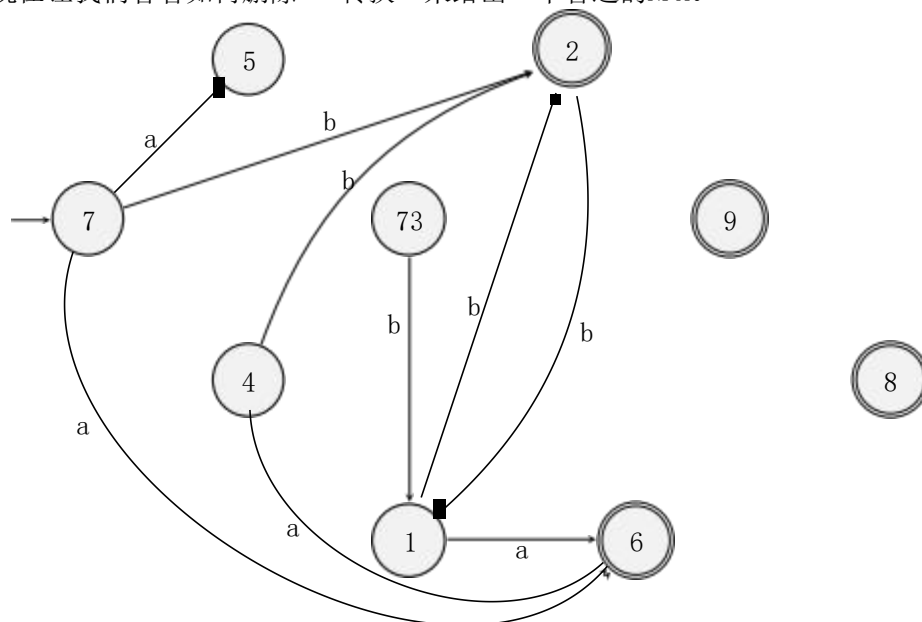
$7 \xrightarrow{a} 5 \xrightarrow{b} 4 \xrightarrow{b} 1 \xrightarrow{b} 2$ 是一个从7到2的缓慢的b过渡；

$2 \xrightarrow{b} 3 \xrightarrow{b} 1$ 是一个从2到1的缓慢的b过渡；

$1 \xrightarrow{b} 2$ 是一个从1到2的缓慢的b过渡；

$2 \xrightarrow{b} 9 \xrightarrow{b} 8$ 正在慢慢接受。

您可以看到，这条路径由三个缓慢的b-转换和缓慢的接受转换组成。一个缓慢的b过渡由几个（零或更多）“ \rightarrow ”的过渡组成，最终形成一个b过渡。缓慢的接受包括几个“ \rightarrow ”过渡，最终达到一个接受状态。现在让我们看看如何删除“ \rightarrow ”转换来给出一个普通的NFA。



该自动机看起来与之前相似：状态相同，初始状态相同。不同之处在于这里看到的转换是缓慢的转换，这里看到的接受状态是缓慢的接受状态。（状态8、9和4可以被删除，因为它们无法访问。）

6 克莱恩定理

参见画布上的视频“Kleene定理”（28分钟）。

到目前为止，我们已经看到了一些例子

用一个规则程序来描述一种语言

在DFA上解决一个匹配问题。

事实证明，这两件事是紧密相连的。

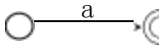
关于一种语言 L 的定理1 (Kleene定理) Σ^* ，以下是等价的。


1. L 是规则的，即。它可以用 $regexp$ 来描述。

2. L 的匹配问题可以用DFA来解决。

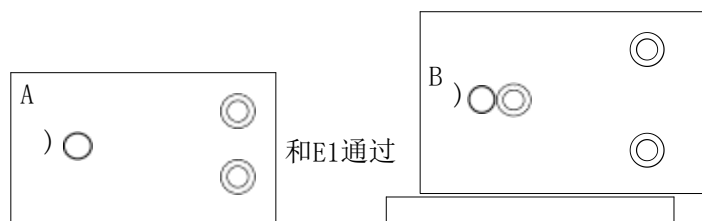
我们将只研究 (1)) (2) , i 的证明。e. 我们将看到如何将 $regexp$ 转换为识别相同语言的DFA。正如我们所看到的，构造一个“NFA就足够了，因为这样我们就删除“-转换来获得一个NFA，最后我们确定获得一个DFA。

所以我们想将一个 $regexp$ 转换为一个“识别相同语言的NFA”。

$regexp$ a 被识别为  同样地，如果 E 是 b 或 c 。

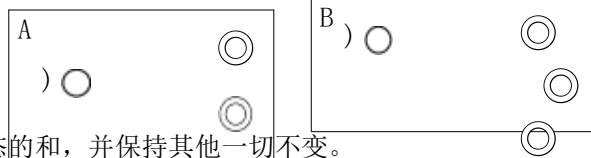
“ $regexp$ ”被认可) 

如果 $e0$ 被识别为



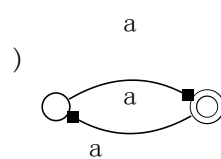
和 $E1$ 通过

那时 $E0|E1$ 被识别

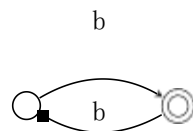


也就是说，我们取这两组状态的和，并保持其他一切不变。

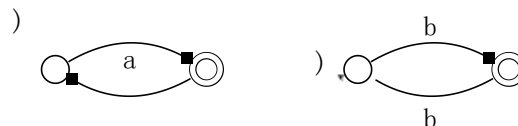
例如，知道 $a(aa)^*$ 是公认的



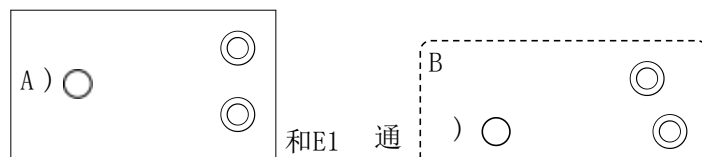
和 $b(bb)^*$ 通过)



我们推导出 $a(aa)^*|b(bb)^*$ 是公认的

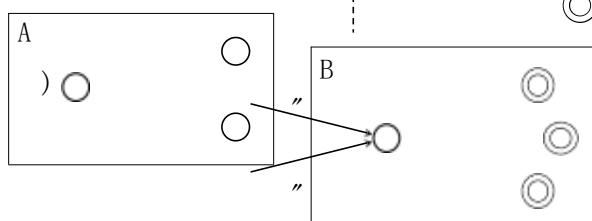


如果 $e0$ 被识别为

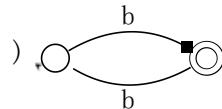
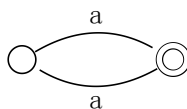


和 $E1$ 通

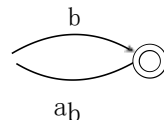
那时 $E0E1$ 是公认的



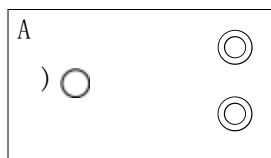
例如，知道 $a(aa)^*$ 是公认的 $)^*$ 和 $b(bb)^*$ 通过



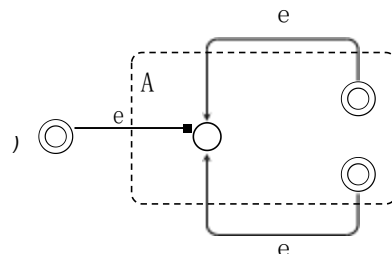
我们推导出 $a(aa)^*b(bb)^*$ 是公认的 $)^*$



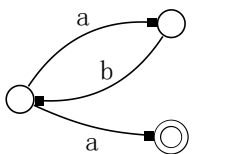
如果E被识别为



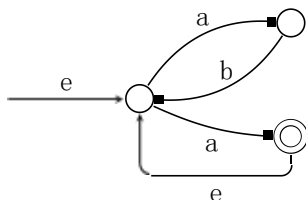
然后 E^* 是公认的



例如， $(ab)^*a$ 被认可



所以 $(ab)^*a)^*$ 已被认可

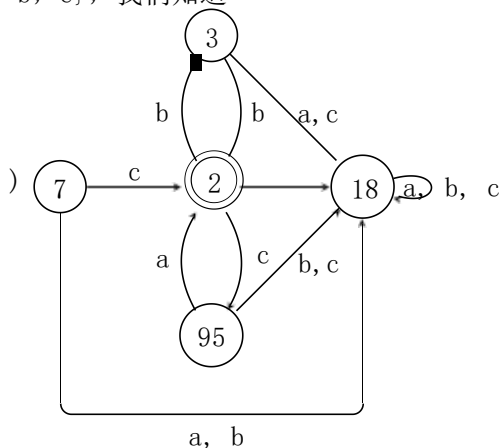


最后， \emptyset 被空自动机(无状态的部分DFA)识别。

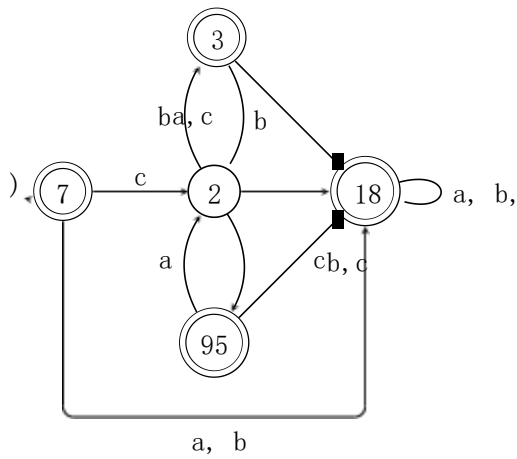
6.1使用克莱恩定理

虽然重构自动机和有限自动机是等价的，但在自动机上操作通常更容易。例如，我们问：一个常规语言的补充是规则的吗？这对于regexps来说并不明显，但使用（总）dfa很明显：只是用不接受的状态代替接受的状态，反之亦然。例如，对于字母表 $\{a, b, c\}$ ，我们知道

$c(bb|ca)^*$ 是公认的



所以它的补体被识别出来



这意味着正则语言 L 和 M 的交集是正则的，因为 $M \setminus L = L \cap M$ 。但我们也可以直接看到这一点。假设 \overline{L} 被自动机 A 识别， M 被 B 识别。如果你给你一个单词 w ，你想知道它是否在 $L \setminus M$ 中，只通过 w 一次，你可以在你阅读 w 的时候同时跟踪你在 A 中的位置和你在 B 中的位置。所以你的状态在任何时候都是一对 (x, y) ，其中 x 是 A 中的状态， y 是 B 中的状态。这将立即给你一个DFA。

例如，假设您希望对最后一个讲义中的密码问题使用DFA。您可以创建一个DFA，它确定一个单词是否至少有3个字符，另一个决定它是否有一个字母，另一个决定它是否有一个数字。然后我们需要一个DFA来表示这些语言的DFA。克莱恩定理告诉我们，一定有一些相应的回归法。