



第三章 正则表达式 2024

王作龙



正则表达式RE

- 3.1 Regular Expression(RE): 类似于算术表达式的代数式子，作为与符号串进行匹配的模式来定义语言，多用做文本匹配模式。
 - 正则表达式是如何构成的？
 - 正则表达式如何定义语言？
- 3.2 RE与DFA、NFA等价性
- 3.3 RE应用典型
- 3.4 为RE所满足的代数定律有哪些？



构造正则语言

- 观察字母表 Σ 上的语言 $L \subseteq \Sigma^*$:
 - 原子语言: $\{a\}, a \in \Sigma$
 - 原子语言: $\{\epsilon\}$
 - 原子语言: φ
- 原子语言都是正则语言吗?
 - 因为可用自动机定义, 所以显然是。
- 但语言 L 可能是正则语言也可能不是 (乔姆斯基体系)。
- 那么产生一种构造语言的思路:
 - 通过原子语言与语言上的运算构造出语言。
 - 同时断言如此构造的语言是正则语言。



语言的运算

- 令 L, L_1, L_2 是 Σ 上的语言。
- 语言连接运算 '.' 借助于符号串的连接运算计算结果
 - $L_1 \cdot L_2 = \{s \cdot t \mid s \in L_1, t \in L_2\}$
 - '.' 可省略
 - 满足结合律: $L(L_1 L_2) = (L L_1) L_2$
 - n 个 L 依次连接 (n -次幂) $L^n = \{s_1 s_2 \dots s_n \mid s_1, s_2, \dots, s_n \in L\}$
 - $L^0 = \{\epsilon\}$
- 语言上的闭包运算 '*'
- $L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{n \geq 0} L^n$
- 语言上的并运算 ' \cup ' 就是集合并运算



- $L_1 = \{0, 01\}$
- $L_2 = \{\epsilon, 1, 11, 111, \dots\}$
- $L_1 L_2 = \{0, 01, 011, 0111, \dots\} \cup \{01, 011, 0111, \dots\}$
- $L_1^2 = \{00, 001, 010, 0101\}$
- $L_2^n = L_2 \ (n \geq 1)$
- $L_1 \cup L_2 = \{0, 01, \epsilon, 1, 11, 111, \dots\}$



语言的闭包运算例

$$L_1 = \{0, 01\}, L_1^* = ?$$

L_1^* : 001000001属于 L_1^*

00110001不属于 L_1^*

10010001不属于 L_1^*

$$L_1^0 = \{\epsilon\}$$

$$L_1^1 = \{0, 01\}$$

$$L_1^2 = \{00, 001, 010, 0101\}$$

$$L_1^3 = \{000, 0001, 0010, 00101, \\ 0100, 01001, 01010, 010101\}$$

L_1^* 所有以0开头不含连续1
的串 (加上空串)



➤ 由任意个1组成的串的全集 L_2

$$L_2 = \{\epsilon, 1, 11, 111, \dots\}, \quad L_2^* = ?$$

$$L_2^0 = \{\epsilon\}$$

$$L_2^1 = L_2$$

$$L_2^2 = L_2$$

$$L_2^n = L_2 \quad (n \geq 1)$$

$$L_2^* = L_2^0 \cup L_2^1 \cup L_2^2 \cup \dots$$

$$= \{\epsilon\} \cup L_2^1 \cup L_2^2 \cup \dots$$

$$= L_2$$

$$L_2^* = L_2$$

$$\epsilon \in L^* ?$$

$$\{\}^* = ?$$

$$\{\epsilon\}^* = ?$$

$$\{1\}^* = L_2 ?$$

$$\epsilon \in L^*$$

$$\{\}^* = \{\epsilon\}$$

$$\{\epsilon\}^* = \{\epsilon\}$$

$$\{1\}^* = L_2$$



构造语言的方法

- 原子语言： Σ 上最基本的语言 φ 、 $\{\varepsilon\}$ 、 $\{a\}$, $a \in \Sigma$ 。
- 语言上运算：连接、闭包、并
- 构造方法：从原子语言开始，利用语言上的运算逐步组合而成，其中可使用括号。
- 满足性质：
 - 原子语言是 Σ 上语言。
 - 如果 L, L_1 是 Σ 上语言，那么 $L \cdot L_1$ 、 $L \cup L_1$ 、 L^* 都是。
 - 任何 Σ 上语言的运算结果仍然是 Σ 上语言。
- 语言构造过程就是语言运算表达式的计算过程。例子：
 - $L_1 = \{0\}(\{0\} \cup \{1\})^*$
 - $L_2 = (\{0\}\{1\}^*) \cup (\{1\}\{0\}^*)$
- 语言运算表达式模式化为正则表达式。



什么是正则表达式

原子语言的模式

- 语言 $\{a\}$ 模式化为RE a , 有 $L(a)=\{a\}$
- 语言 $\{\varepsilon\}$ 模式化为RE ε , 有 $L(\varepsilon)=\{\varepsilon\}$
- \emptyset 指代 $\{\}$, 即 $L(\emptyset)=\varphi$

运算符 \cup 改名为 $+$

RE运算符	语言运算符
连接'.'	连接'.'
并'+'	并' \cup '
闭包'*'	闭包'*'

例：语言 L_1 是0或由0开头的0-1串组成

- 语言运算表达式： $\{0\}(\{0\}\cup\{1\})^*$
- 对应正则表达式： $0(0+1)^*$
- $L(0(0+1)^*) = \{0\}(\{0\}\cup\{1\})^* = L_1$

例： $L_2 = (\{0\}\{1\}^*) \cup (\{1\}\{0\}^*)$

- 对应正则表达式： 01^*+10^*



构造正则表达式

- 字母表 Σ 上的正则表达式是使用如下规则形成的表达式。
- 基础：
 - 符号 \emptyset 和 ε 都是正则表达式；
 - Σ 中每个元素 a 都是正则表达式；
 - ~~引用某个语言的变量或者某个正则表达式的名字。~~
- 归纳：
 - 如果 R 和 S 是正则表达式，那么 $R+S$ 、 RS 和 R^* 都是；
 - 如果 R 是正则表达式，那么 (R) 也是。
- 正则表达式运算符的优先级
 - 优先级： '*' 最高， '.' 次之， '+' 最低
 - 计算顺序： 先括号里，后括号外，同级从左往右

a) $0(0 + 1)^*$

b) $01^* + 10^*$

c) $(0 + 1)^*01(0 + 1)^*$



归纳：字母表 Σ 上RE:r与 $L(r)$ 间的一一对应

➤ R, S, T 为正则表达式, $L(R)$ 为 R 定义的语言

正则表达式 r	正则语言 $L(r)$
\emptyset	$L(\emptyset) = \varphi$
ε	$L(\varepsilon) = \{\varepsilon\}$
a	$L(a) = \{a\}$
R	$L(R)$
$R \cdot S$	$L(R \cdot S) = L(R) \cdot L(S)$
$R + S$	$L(R + S) = L(R) \cup L(S)$
R^*	$L(R^*) = L(R)^*$
$R + S + T = (R + S) + T$	$L(R + S + T) = L(R + S) \cup L(T)$
$R \cdot S \cdot T = (R \cdot S) \cdot T$	$L(R \cdot S \cdot T) = L(R \cdot S) \cdot L(T)$
$R + S \cdot T = R + (S \cdot T)$	$L(R + S \cdot T) = L(R) \cup L(S \cdot T)$
$R \cdot S^* = R \cdot (S^*)$	$L(R \cdot S^*) = L(R) \cdot L(S^*)$
$R + S^* = R + (S^*)$	$L(R) \cup L(S^*)$



例：分析正则表达式

➤ 以下例子都是关于RE的判定性质

$0+1$

长度为1的0-1串 $=\{0, 1\}$

$(0+1)^*$

任意0-1串 $\{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$

$L((0+1)^*) = \{0, 1\}^*$

$(0+1)^*010$

以010结尾的0-1串

$(0+1)^*11(0+1)^*$

包含子串11的0-1串



例：分析正则表达式

$$((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$$

长度是偶数或是3的倍数的0-1串

$$((0+1)(0+1))^*$$

偶数长度

$$(0+1)(0+1)$$

长度为2

$$((0+1)(0+1)(0+1))^*$$

长度3的倍数

$$(0+1)(0+1)(0+1)$$

长度为3



例：分析正则表达式

$$((0+1)(0+1)+(0+1)(0+1)(0+1))^*$$

串可以分成段,
其中每段长度为2或3

$$(0+1)(0+1)+(0+1)(0+1)(0+1)$$

串长度为2或3

$$(0+1)(0+1)$$

长度2

$$(0+1)(0+1)(0+1)$$

长度3



例：分析正则表达式

$$(1+01+001)^*(\epsilon+0+00)$$

以最多两个0结束

连续的1之间被最多两个0分开

不会有连续三个0的串

$$L((1+01+001)^*(\epsilon+0+00)) = \{x: x \text{ 不含子串 } 000\}$$

ϵ

00

01100101110

00100110



例：写出正则表达式

➤ 该语言元素包含子串00

- $(0+1)^*00(0+1)^*$

➤ 该语言元素不含子串00

- $1^*(011^*)^*(\epsilon + 0)$

111|011|01|011|01|01|0

开头有些 1 每个 0 后跟 1 到多个 1 有可能以一个 0 为结束

- 可有全1前缀，也可没有
- 中间部分，每个 0 后跟1到多个1
- 可有0为后缀，也可没有



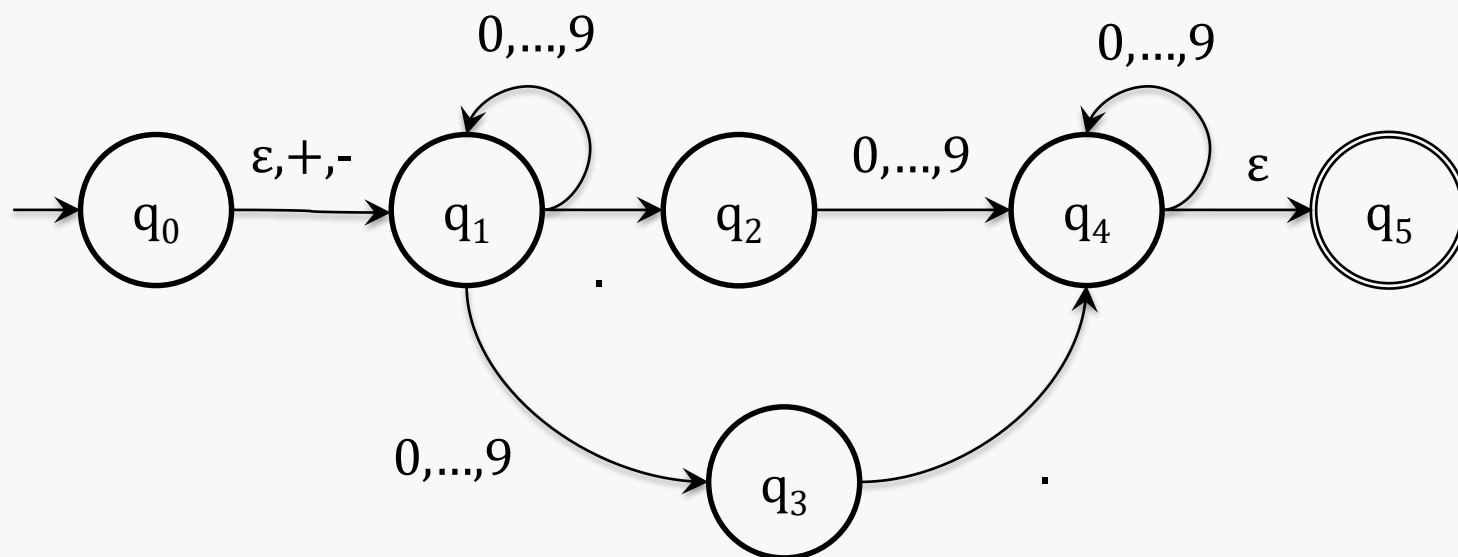
例：写出正则表达式

- 该语言元素都有偶数个0
 - $(1^*01^*01^*)^*$
- 该语言元素有偶数个0，或者有奇数个1
 - $(1^*01^*01^*)^* + (0^*10^*)(0^*10^*10^*)^*$
- 该语言元素有偶数个0，并且有奇数个1
 - ?



例：写出正则表达式

- 有符号十进制定点数。

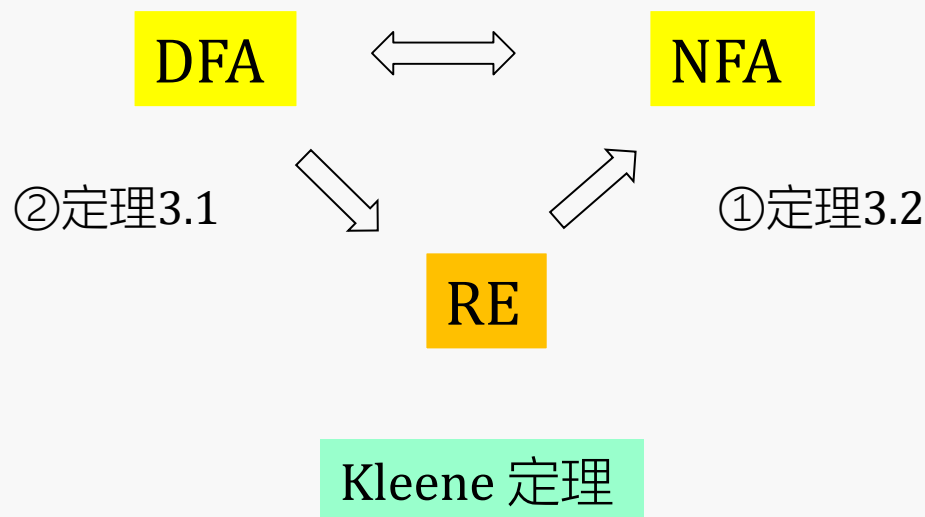


- $(\epsilon + \backslash + + -)((0 + \dots + 9)^* \cdot (0 + \dots + 9)(0 + \dots + 9)^* + (0 + \dots + 9)(0 + \dots + 9)^* \cdot (0 + \dots + 9)^*)$



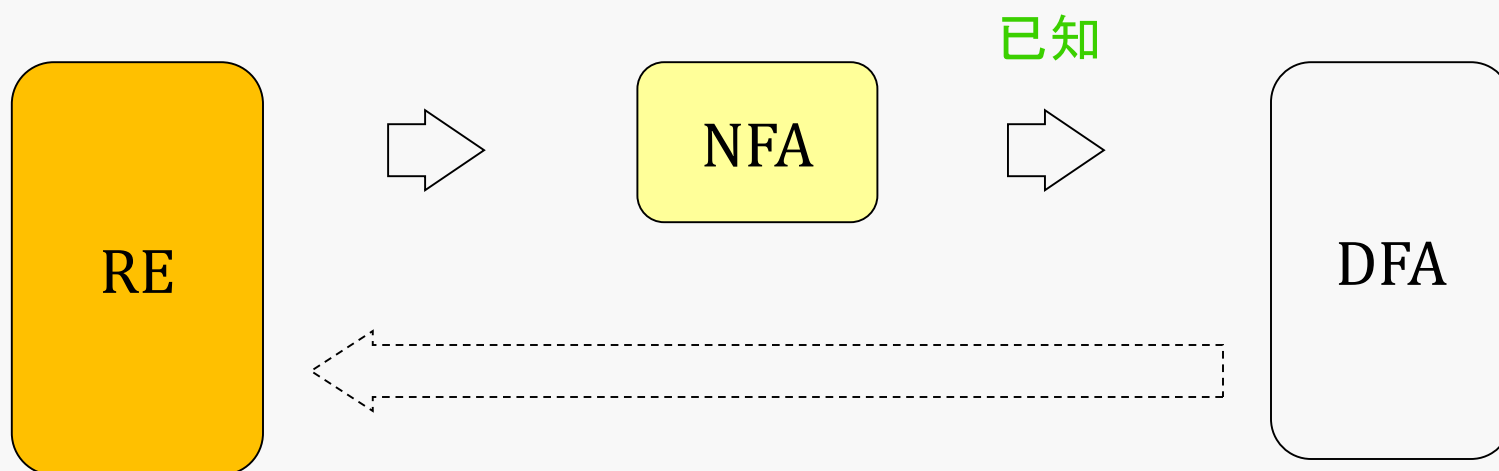
3.2 RE与DFA、NFA等价性

- 一个语言用正则表达式定义、也用有穷自动机定义，如何？
- 定理3.1: 对任一DFA A 存在正则表达式 R 使得 $L(R)=L(A)$
- 定理3.2: 对任意正则表达式 R 存在 ε -NFA E 使得 $L(E)=L(R)$





定理3.2证明路线图

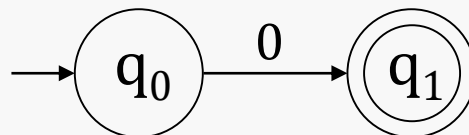


每一个用正则表达式来定义的语言也可用有穷自动机来定义



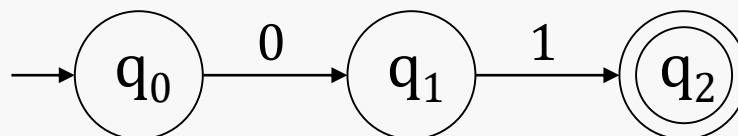
例：正则表达式 \Rightarrow NFA

$R_1 = 0$



原子

$R_2 = 01$

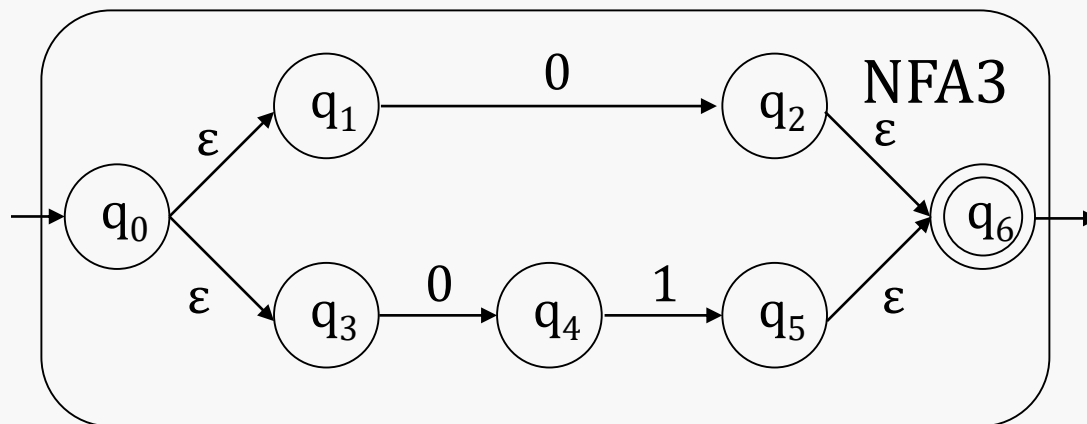


连接



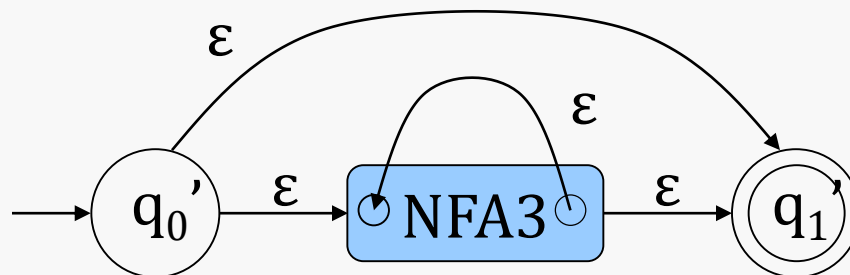
例：正则表达式 \Rightarrow NFA

$$R_3 = 0 + 01$$



并

$$R_4 = (0 + 01)^*$$



闭包

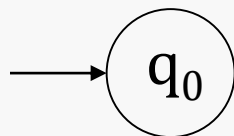


正则表达式转换为NFA通用方法

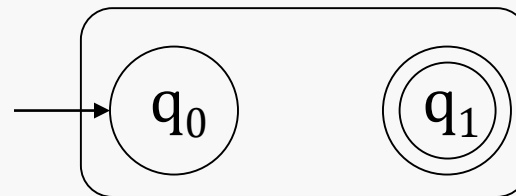
正则表达式

NFA

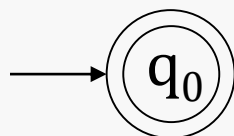
\emptyset



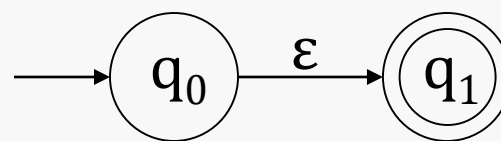
或者



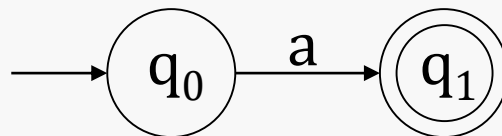
ϵ



或者



$a \in \Sigma$



基础:

- 正则表达式 \emptyset , ϵ 和 $a, a \in \Sigma$ 对应的NFA如图所示。

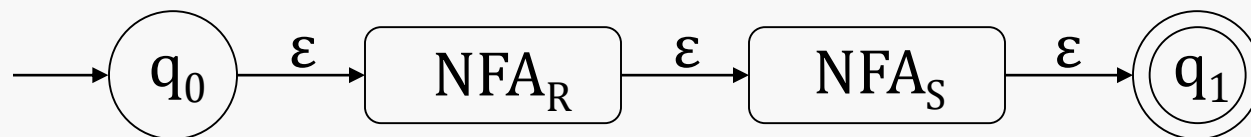
归纳

- 对于正则表达式 R 和 S , $R+S$, RS , R^* 和 (R) , ?

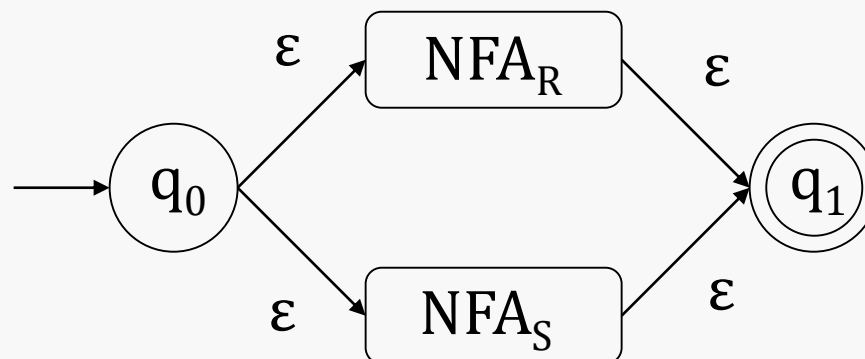


归纳证明示意

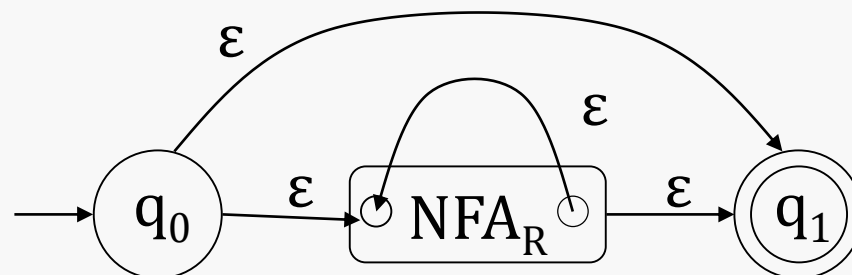
RS



R+S



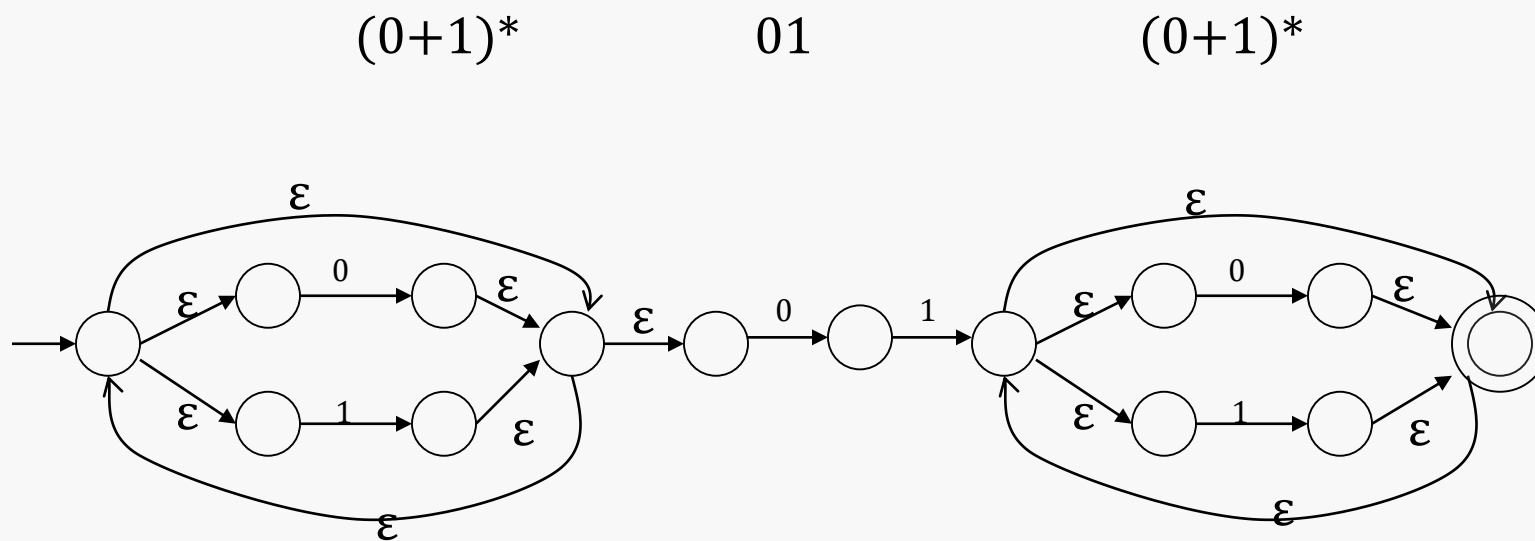
R^*





例：从RE构造NFA

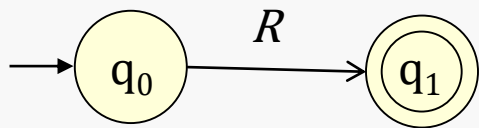
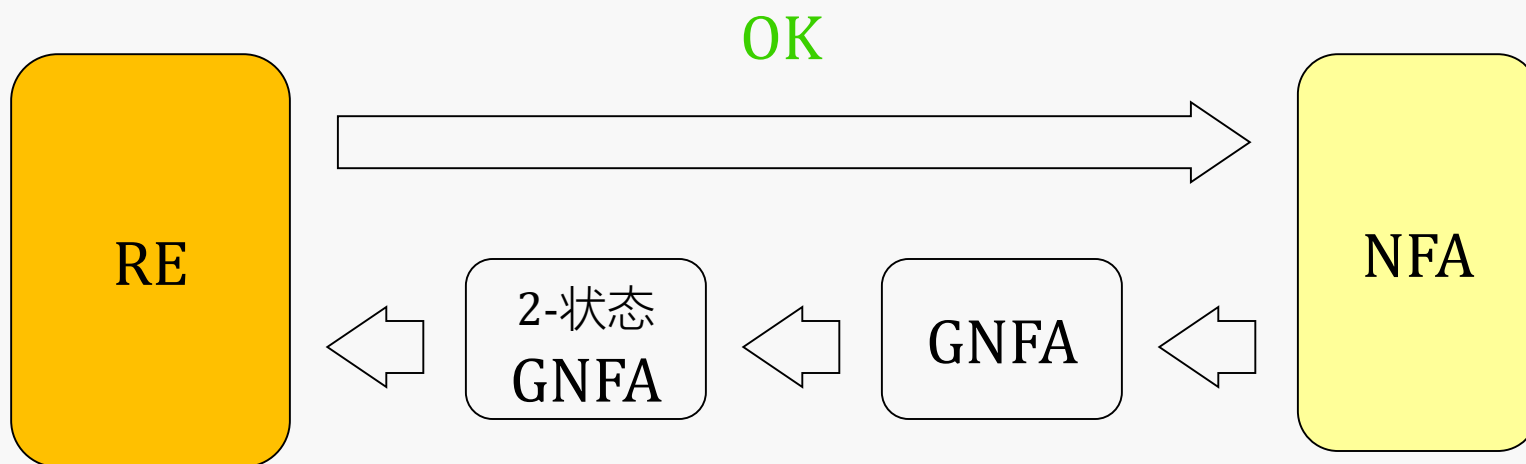
➤ $(0+1)^*01(0+1)^*$





有穷自动机转换为RE

- 对任一DFA A 存在正则表达式 R 使得 $L(R)=L(A)$
- DFA是NFA特例，所以关键是NFA到RE转换方法

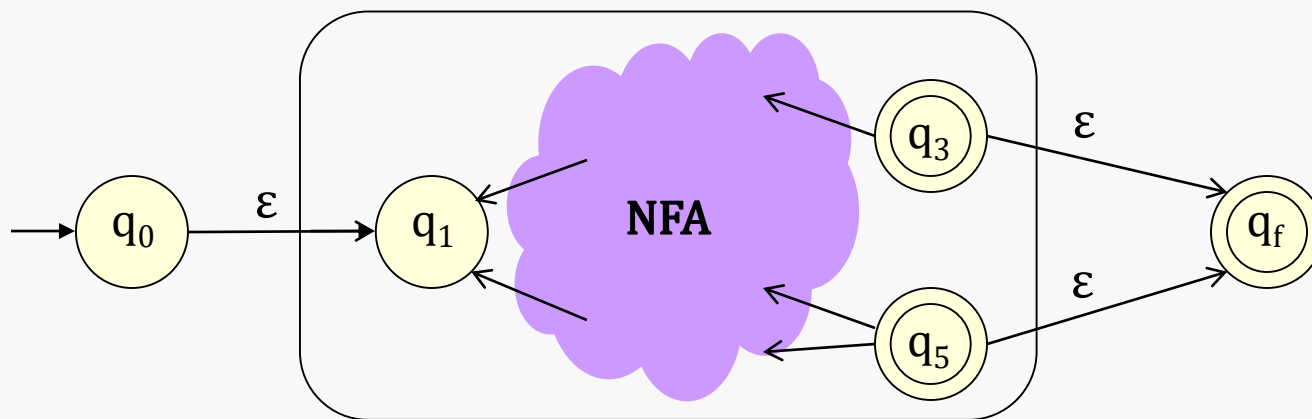


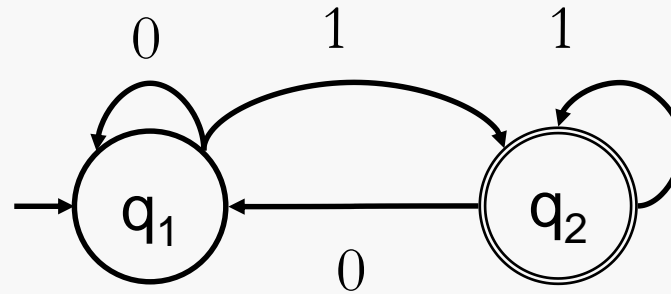
任一个 2-状态 GNFA 总与某个正则表达式 R 等价。



通用NFA (GNFA)

- 只有一个接受状态
- 没有射入到初始状态的弧
- 没有从接受状态射出的弧
- 弧上标记采用正则表达式作标记

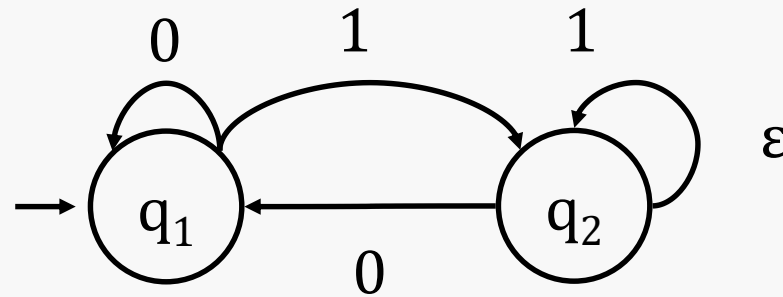




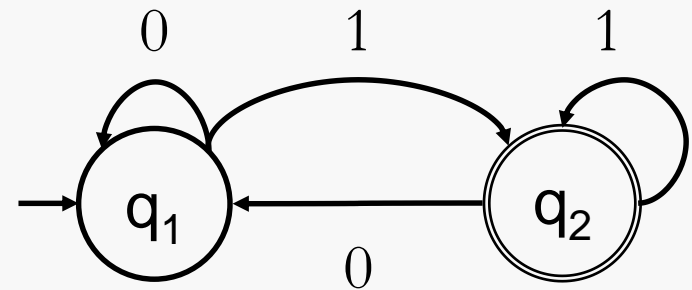
- 只有一个接受状态? YES
- 没有射入到初始状态的弧? NO
- 没有从接受状态射出的弧? NO
- 弧上标记是正则表达式? NO/YES

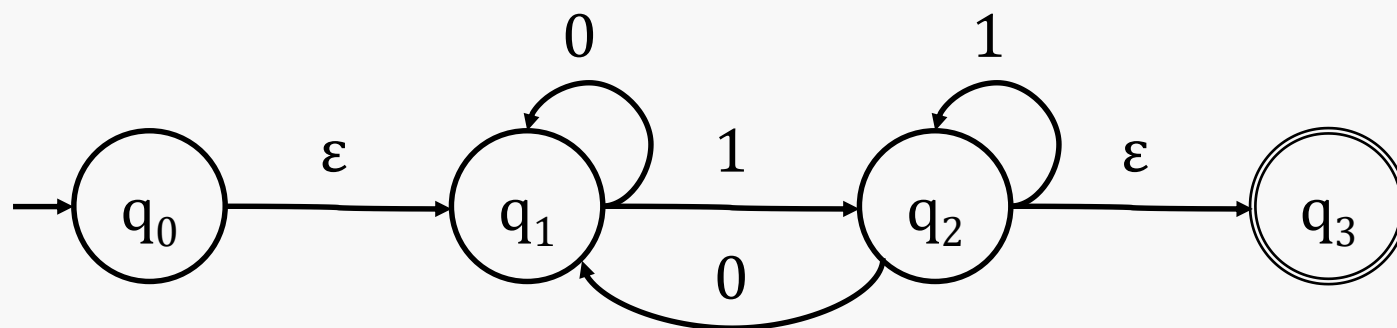


GNFA

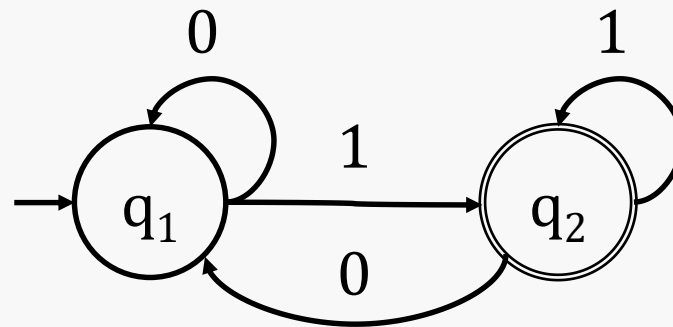


- ✓ • 只有一个接受状态
- ✓ • 没有射入到初始状态的弧
- ✓ • 没有从接受状态射出的弧





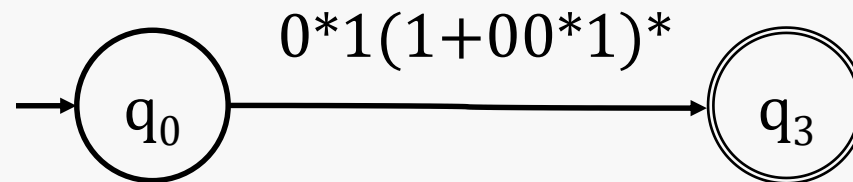
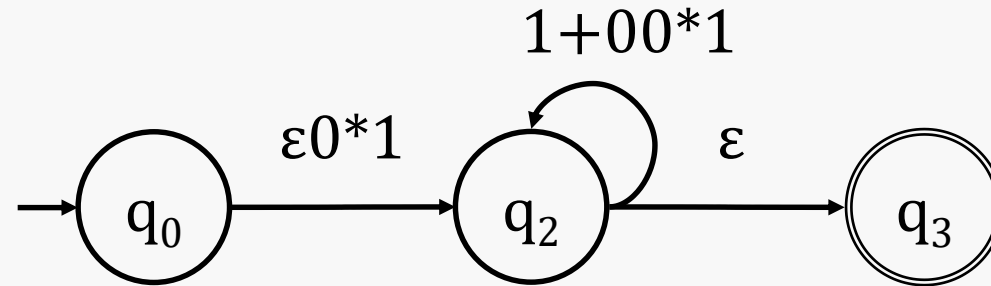
- ✓ • 只有一个接受状态
- ✓ • 没有射入到初始状态的弧
- ✓ • 没有从接受状态射出的弧



- ✓ • 只有一个接受状态
- ✓ • 没有射入到初始状态的弧
- ✓ • 没有从接受状态射出的弧

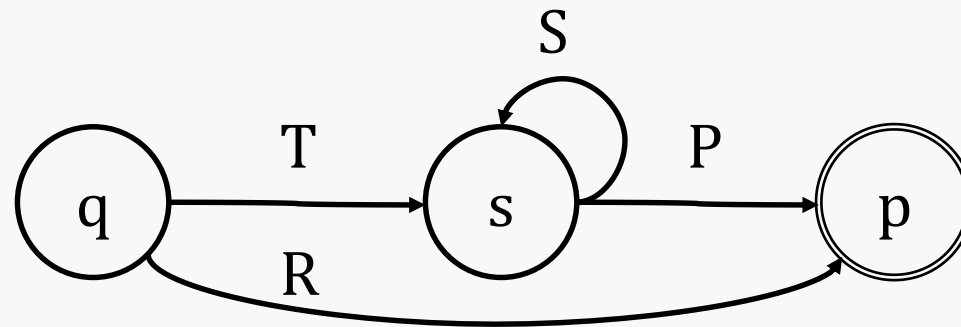


GNFA



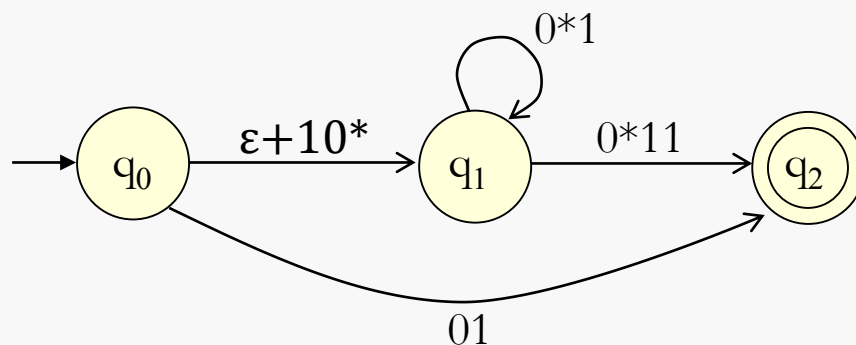


GNFA



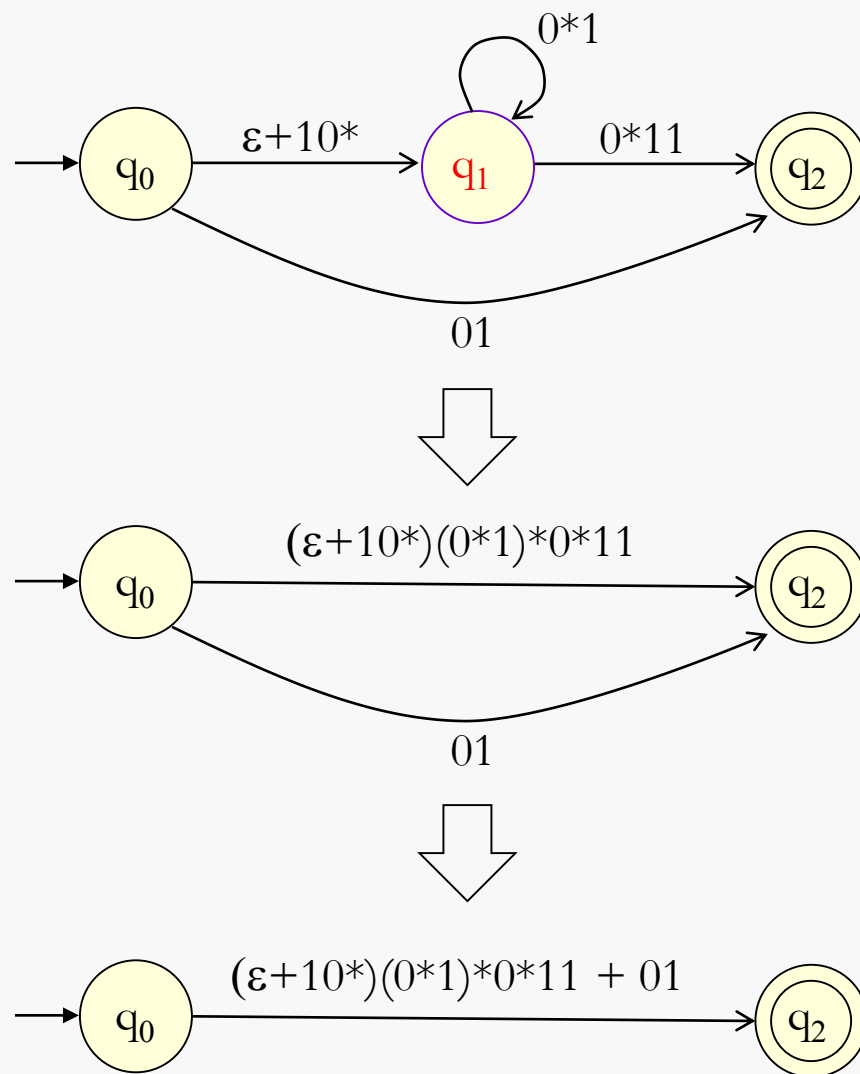


- GNFA弧上标记采用正则表达式作标记
- NFA的a弧看作正则表达式a做标记
- NFA的 ϵ 弧看作正则表达式 ϵ 做标记
- NFA的缺失弧可看作是正则表达式 \emptyset 做标记的弧
- 如此，NFA弧上标记都被当作正则表达式，再检查没有射入初始状态的弧和没有从接受状态射出的弧，即成为GNFA





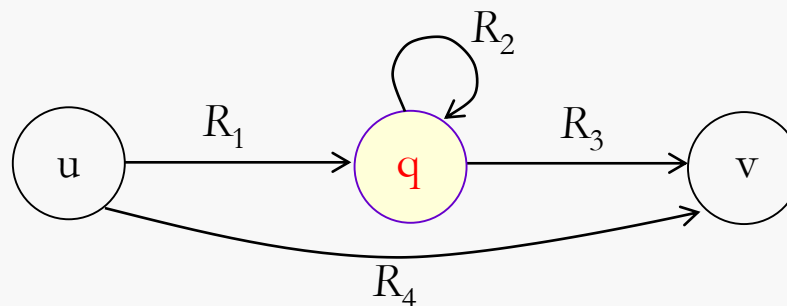
GNFA的状态约简



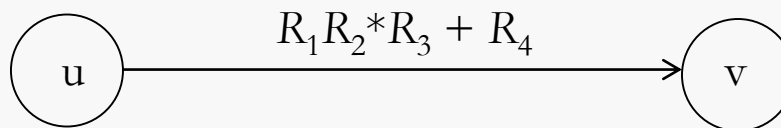


- 要消除状态 q , 对于每个状态对 (u, v)

替代



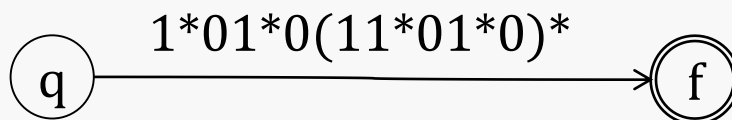
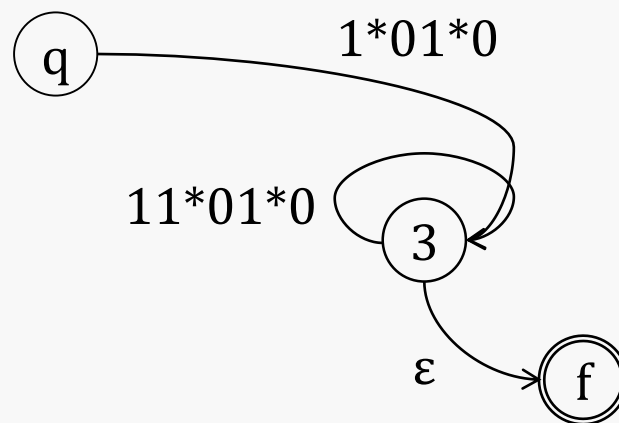
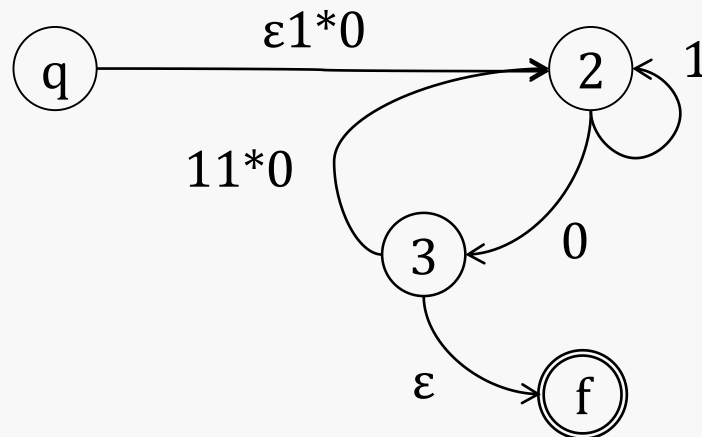
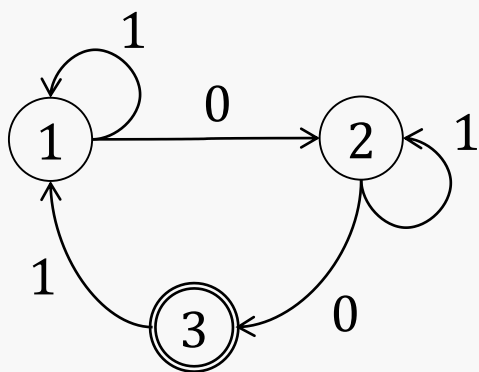
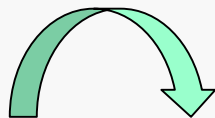
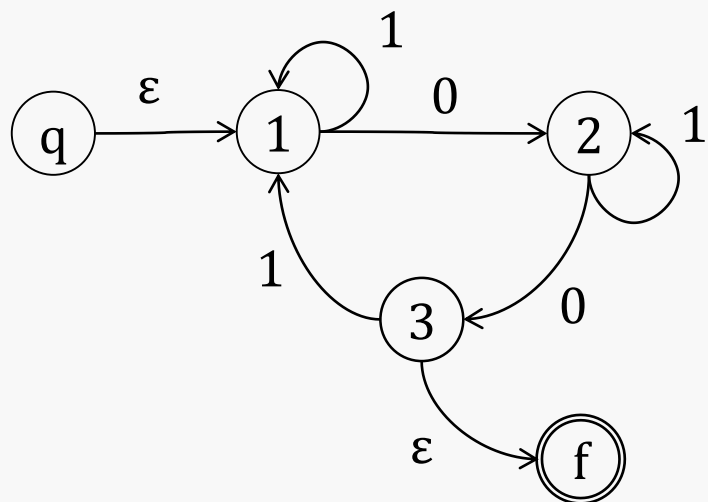
为



- 注意 $u=v$ 时也如此



例：GNFA状态约简





定理3.1证明

- 定理3.1 对于任意一个DFA A , 存在一个正则表达式 R , 使得 $L(A) = L(R)$ 。
- DFA $A = (Q, \Sigma, \delta, q_0, F)$
- $L(A) = \{w \mid w \in \Sigma^*, \delta(q_0, w) \in F\}$
- 证明的思路:
 - 考察 q_0 至 q , $q \in F$, 之间所有路径, 用正则表达式表示之, 即得。
 - 这些路径是否不无遗漏地都表示为正则表达式了呢?



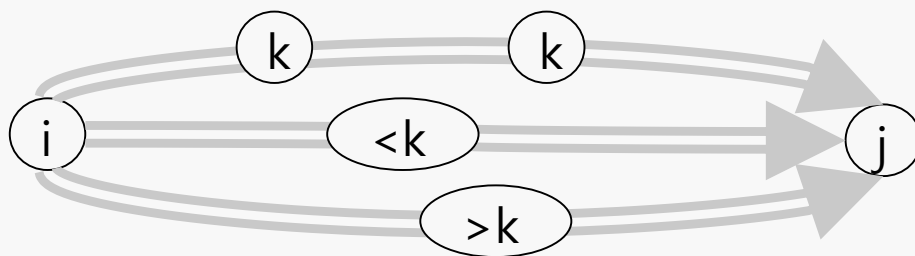
关键思路：用归纳法穷尽所有路径

- 对结点编号，对两端点间所有路径进行排序以便于归纳。
- 对DFA状态图中的顶点从1到 n ， $n = |Q|$ ，编号。
- 对始端 i 到末端 j 的所有路径， $1 \leq i, j \leq n$ ，进行如下排列：
 - 路径上除端点外的所有中间结点其编号不大于 k ，按照 $k = 0, 1, \dots, n$ 依次对路径进行排列
 - 注意 $k = 0$ 时满足条件的路径为只有端点，没有中间结点
- 那么针对路径 $PATH(i, j)$ 的归纳方式：
 - 基础：不经过任何中间结点的路径的标记为 $R_{ij}^{(0)}$
 - 归纳：经过不大于 k 的中间结点的路径的标记为
$$R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^*R_{kj}^{(k-1)} = R_{ij}^{(k)}$$



证明定理3.1

- 基础：不经过任何中间结点的路径为 $R_{ij}^{(0)}$ ，分两种情形：
 - 情形一： $i \neq j$;
 - $R_{ij}^{(0)} = \emptyset$;
 - $\text{for}(a \in \Sigma) \text{if}(j \in v(i, a)) R_{ij}^{(0)} = R_{ij}^{(0)} + a$;
 - 情形二： $i = j$;
 - $R_{ij}^{(0)} = \varepsilon$;
 - $\text{for}(a \in \Sigma) \text{if}(j \in v(i, a)) R_{ij}^{(0)} = R_{ij}^{(0)} + a$;
- 归纳： $\text{PATH}(i, j)$ 可分为三部分，



$\text{PATH}(i, j, R_{ij}^{(0)}, 0);$
 $\text{PATH}(i, j, R_{ij}^{(k-1)}, k-1);$
 $\text{PATH}(i, j, R_{ij}^{(k)}, k);$
 $\text{PATH}(i, j, ?, ?)$



证明定理3.1

- 归纳：假定顶点*i* 到顶点*j* 所有经过顶点不大于*k* -1的路径，它们的标记并在一起定义为 $R_{ij}^{(k-1)}$ ，那么*i* 到*j* 所有经过顶点不大于*k* 的路径，分为两种情形：
- 情形一：没有经过顶点*k*；
 - 根据归纳假设，这些路径的标记都属于 $L(R_{ij}^{(k-1)})$
- 情形二：经过顶点*k*；
 - 路径分段为：PATH(*i*,*k*)，PATH(*k*,*k*)，PATH(*k*,*j*)，且各段都属于情形一
 - 根据归纳假设，这三段组合成的所有路径，它们的标记构成的集合定义为 $R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^*R_{kj}^{(k-1)}$
- 最后，将两种情形合并在一起得到
 - $R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^*R_{kj}^{(k-1)}$



证明定理3.4

- 令 $n = |Q|$, Q 中元素编号为从1到 n , 其中 q_0 编号为1,
- 令 $r = +j \in \{q \text{ 的编号} \mid q \in F\} \cdot R_{1j}^{(n)}$
 - 即 $r = R_{1j_1}^{(n)} + \dots + R_{1j_m}^{(n)}$, 其中 j_1, \dots, j_m 是 F 中各元素的编号
- 则 $L(A) = L(r)$
- 证毕。



例：构造DFA的正则表达式

$$\triangleright R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^*R_{kj}^{(k-1)}$$

$$\triangleright r = R_{13}^{(3)} = R_{13}^{(2)} + R_{13}^{(2)}(R_{33}^{(2)})^*R_{33}^{(2)}$$

$$R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)}(R_{22}^{(1)})^*R_{23}^{(1)}$$

$$R_{13}^{(1)} = \emptyset$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)}(R_{11}^{(0)})^*R_{12}^{(0)}$$

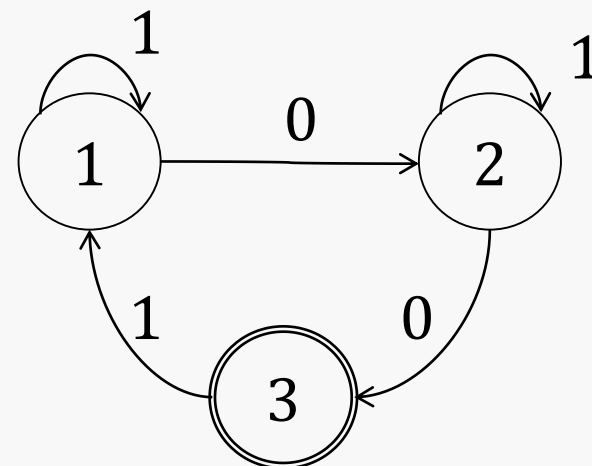
$$R_{22}^{(1)} = R_{22}^{(0)}$$

$$R_{23}^{(1)} = R_{23}^{(0)}$$

$$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)}(R_{22}^{(1)})^*R_{23}^{(1)}$$

$$R_{33}^{(1)} = R_{33}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)})^*R_{13}^{(0)}$$

$$R_{32}^{(1)} = R_{32}^{(0)} + R_{31}^{(0)}(R_{11}^{(0)})^*R_{12}^{(0)}$$



$$R_{11}^{(0)} = \epsilon + 1$$

$$R_{12}^{(0)} = 0$$

$$R_{22}^{(0)} = \epsilon + 1$$

$$R_{23}^{(0)} = 0$$

$$R_{33}^{(0)} = \epsilon$$

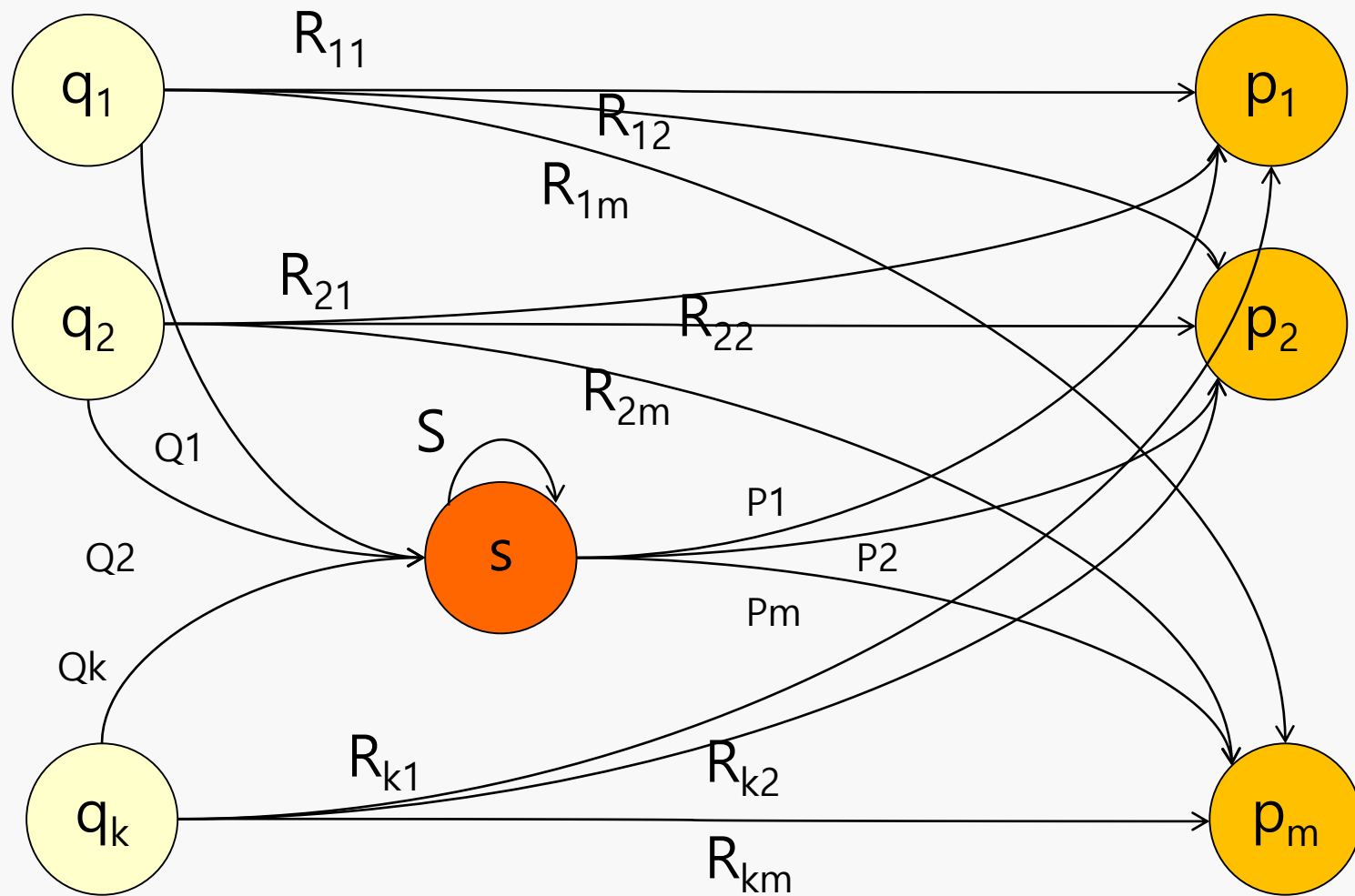
$$R_{13}^{(0)} = \emptyset$$

$$R_{31}^{(0)} = 1$$

$$R_{32}^{(0)} = \emptyset$$

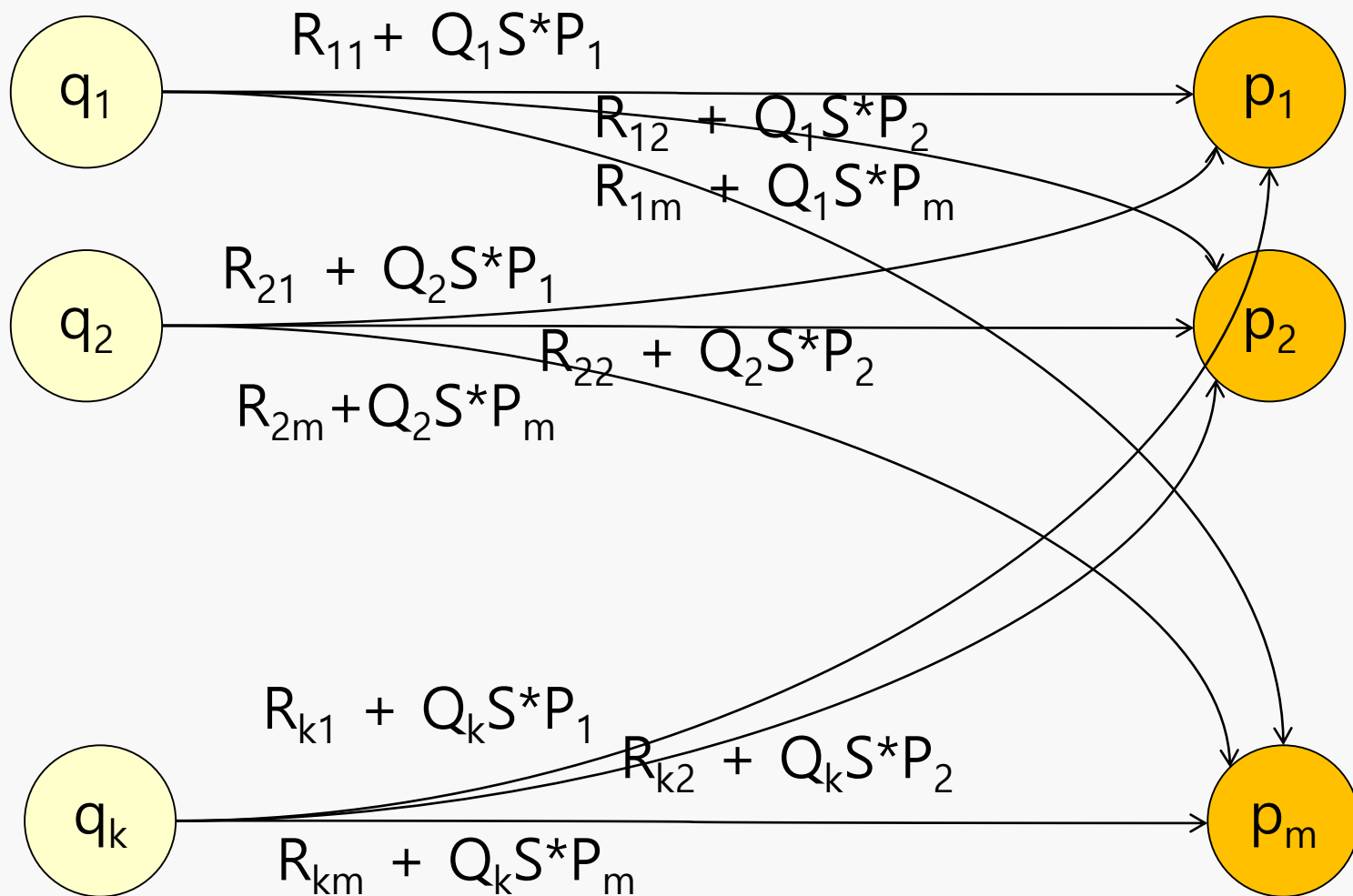


状态约简规则（图变换前）





状态约简规则 (消除s状态后)



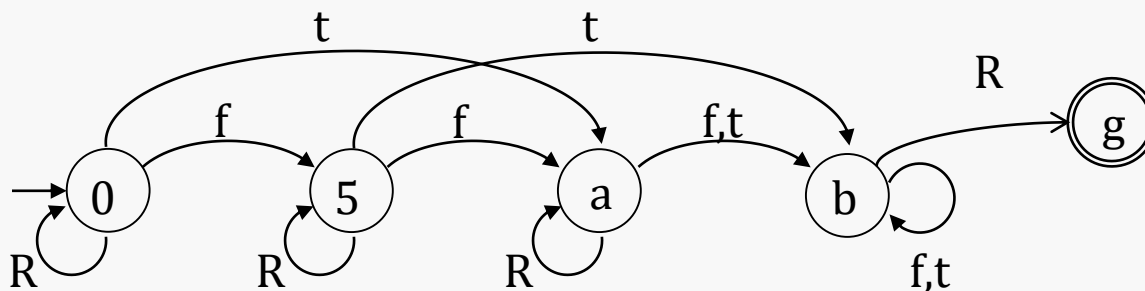


口香糖球机的语言

➤ 始端0末端g的路径有三条：

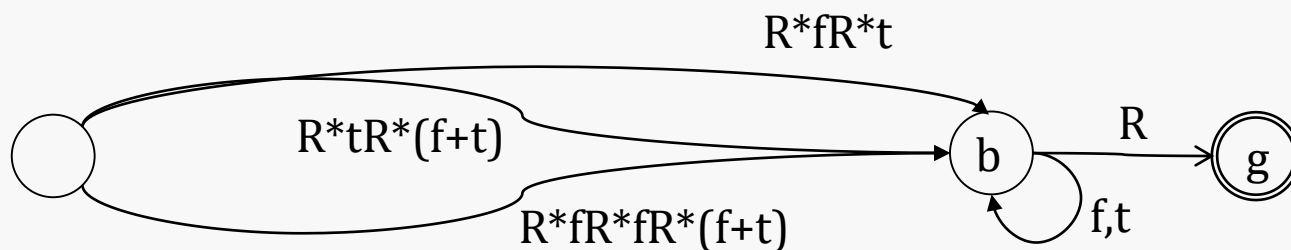
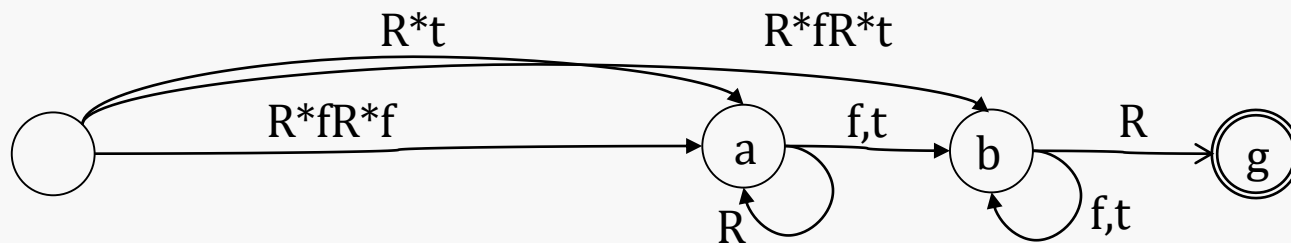
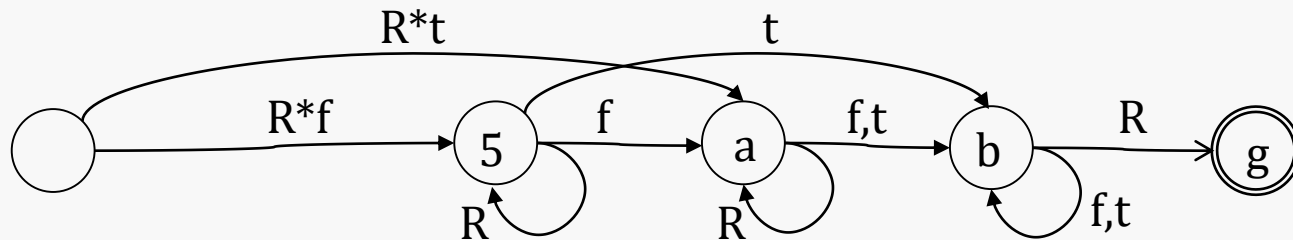
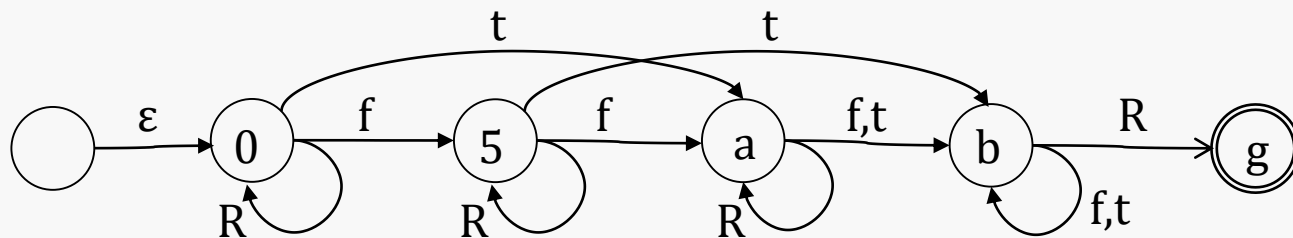
- 0,5,a,b,g $R^*fR^*fR^*(f+t)(f+t)^*R$
- 0,a,b,g $R^*tR^*(f+t)(f+t)^*R$
- 0,5,b,g $R^*fR^*t(f+t)^*R$

$$R^*fR^*fR^*(f+t)(f+t)^*R + R^*tR^*(f+t)(f+t)^*R + R^*fR^*t(f+t)^*R$$



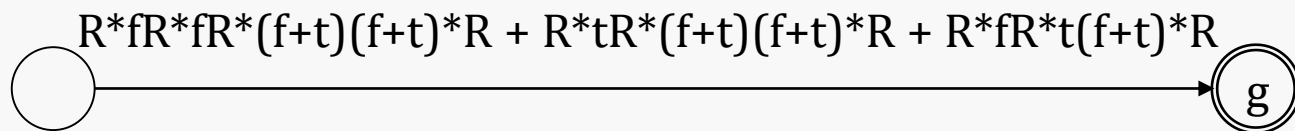
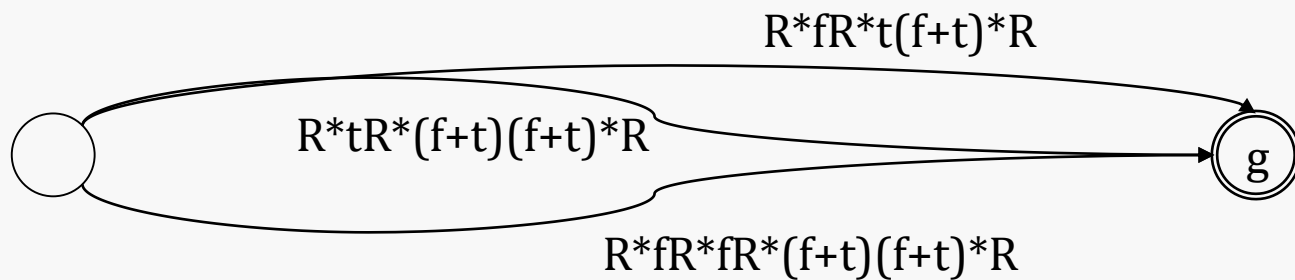
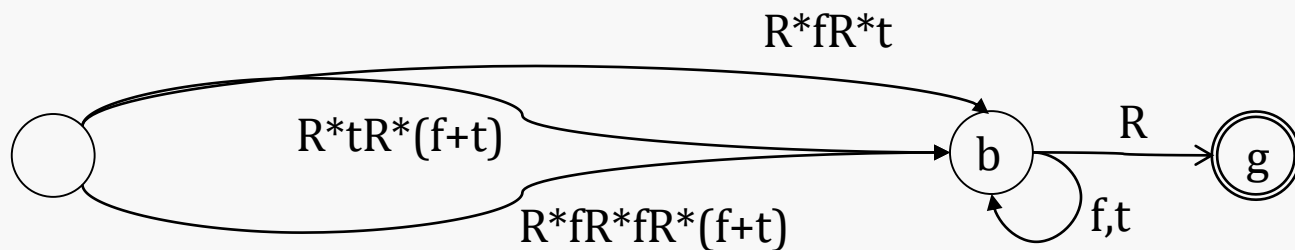


口香糖球机RE





口香糖球机RE





3.3 正则表达式应用举例

- 正则表达式是基于模式匹配定义了实际应用中感兴趣的符号串集合，可用于生成、检索、识别有用的文本内容。通常地，这种作为模式的正则表达式被自动构建成DFA以方便程序实现来完成上述功能。
- UNIX中的正则表达式扩展了RE的操作符
- 允许以RE为输入自动生成词法分析器
- 识别有用的文本内容



- 字母表较大时如ASCII字符集需要更为简洁的表达方式
- 元符号的概念，如元符号'.'表示任意字符，那么'\.'才是点
- 序列 $[a_1a_2...a_k]$ 表示 $a_1+a_2+...+a_k$
- 默认序列的子序列： $[0-9]$ 、 $[a-z]$ 、 $[A-Za-z0-9]$
- 增加的运算符： $R?$ 指 R 出现或不出现； $R+$ 指 R 重复1到多次； $R\{n\}$ 指恰有连续 n 个 R ；另外 $|$ 取代 $+$
- 整数： $[+-]?[1-9][0-9]^*|0$
- 定点数： $\langle\text{整数}\rangle\backslash.?(\langle\text{整数}\rangle|[+-]?)\backslash.[0-9]^*[1-9]$
- 浮点数： $\langle\text{定点数}\rangle[e|E]\langle\text{整数}\rangle$



以RE为输入自动生成词法分析器

- 工具lex、flex
- 以正则表达式作为所识别的单词的模式提供给工具
- 工具将生成一个词法分析器，它按照这些模式一个一个识别输入流中的单词并返回结果
- 具体将在后面课程中介绍



识别有用的文本内容

- 如识别街道地址
- `Street|St\.|Avenue|Ave\.|Road|Rd\.`
- 有些街道包含多个单词如华盛顿特区的Rhode Island Avenue（罗德岛大道）、Xianning West Rd.
- `([A-Z][a-z]*\s)+(Street|St\.|Avenue|Ave\.|Road|Rd\.)`
- `\s`匹配任何空白字符
- 加上门牌号码，如123A Main St
- `[0-9]+[A-Z]?\s([A-Z][a-z]*\s)+(Street|St\.|Avenue|Ave\.|Road|Rd\.)`
- 其它地址情形往上加...
- 28 XIANNING West Rd.
- 咸宁西路28号



识别有用的文本内容

- 正则表达式通常被用来检索、替换那些符合某个模式（规则）的文本。

常用正则表达式

- 用户名
- 电子邮箱
- IP地址

用户名	<code>/^[a-z0-9_-]{3,16}\$/</code>
密码	<code>/^[a-z0-9_-]{6,18}\$/</code>
十六进制值	<code>/^#?([a-f0-9]{6} [a-f0-9]{3})\$/</code>
电子邮箱	<code>/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.-]{2,6})\$/</code> <code>/^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.[a-z\d]+(\.[a-z\d]+)*)\$/</code>
URL	<code>/^(https?:\W)?([\da-z\.-]+)\.([a-z\.-]{2,6})([\Ww\.-]{2,5})?\$</code>
IP 地址	<code>/((2[0-4]\d 25[0-5] \d{1} 01)?\d\d?)\.){3}(2[0-4]\d 25[0-5] \d{1} 01)?\d\d?\$/</code> <code>/^(?:(?:25[0-5] 2[0-4][0-9] [01]?[0-9][0-9]?)\.){3}(?:25[0-5] 2[0-4][0-9] [01]?[0-9][0-9]?)\$/</code>
HTML 标签	<code>/^<([a-z]+)([^<]+)*(?:> (.*)<\1> \s+V>)\$/</code>



3.4 正则表达式的代数定律

- R和S为正则表达式
- $L(RS) = L(R)L(S)$
- $L(R+S) = L(R) \cup L(S)$
- $L(R^*) = (L(R))^*$
- 若 $w \in L(RS)$, 则 $w = xy$, 且 $x \in L(R)$, $y \in L(S)$
- 若 $w \in L(R+S)$, 则 $w \in L(R)$, 或者 $w \in L(S)$
- 若 $w \in L(R^*)$, 则 $w = x^*$, 且 $x \in L(R)$



- 交换律: $R+S = S+R$
- 结合律: $(R+S)+E = R+(S+E)$ 、 $(RS)E = R(SE)$
- 恒等律: $R+\varphi = R$ 、 $\varepsilon R = R\varepsilon = R$
- 零元律: $\emptyset R = R\emptyset = \emptyset$
- 分配律: $R(S+E) = RS + RE$ 、 $(S+E)R = SR+ER$
- 幂等律: $R + R = R$
- 涉及Kleene 闭包的:
 - ① $(R^*)^* = R^*$
 - ② $\emptyset^* = \varepsilon$
 - ③ $\varepsilon^* = \varepsilon$



P63 习题6 (2)

- A是任意正则表达式，证明 $(A^*)^* = A^*$
- 只须证明 $L((A^*)^*) = L(A^*)$

对任意 $\alpha \in L(A^*)$ ， $\alpha = A^k$ ，其中k为非负整数；
那么 α 可写成 $(A^k)^1$ ，即与正规式 $(A^*)^*$ 匹配；
因此， $\alpha \in L((A^*)^*)$ 。

对任意 $\alpha \in L((A^*)^*)$ ， $\alpha = (A^k)^j$ ，其中k和j为非负整数；
那么 α 可写成 A^{k*j} ，其中 $k*j$ 为非负整数；
所以 α 与正规式 A^* 匹配，从而， $\alpha \in L(A^*)$ 。



小结

- 知识点：原子语言、语言运算表达式、正则表达式，正则表达式的语言、UNIX中扩展的运算符
- 知识点：正则表达式与NFA、DFA等价转换
- 知识点：正则表达式代数定律

- 作业：P72 习题3.1(a)；习题3.2(b)(c)；习题3.4(b)；习题3.6(b)