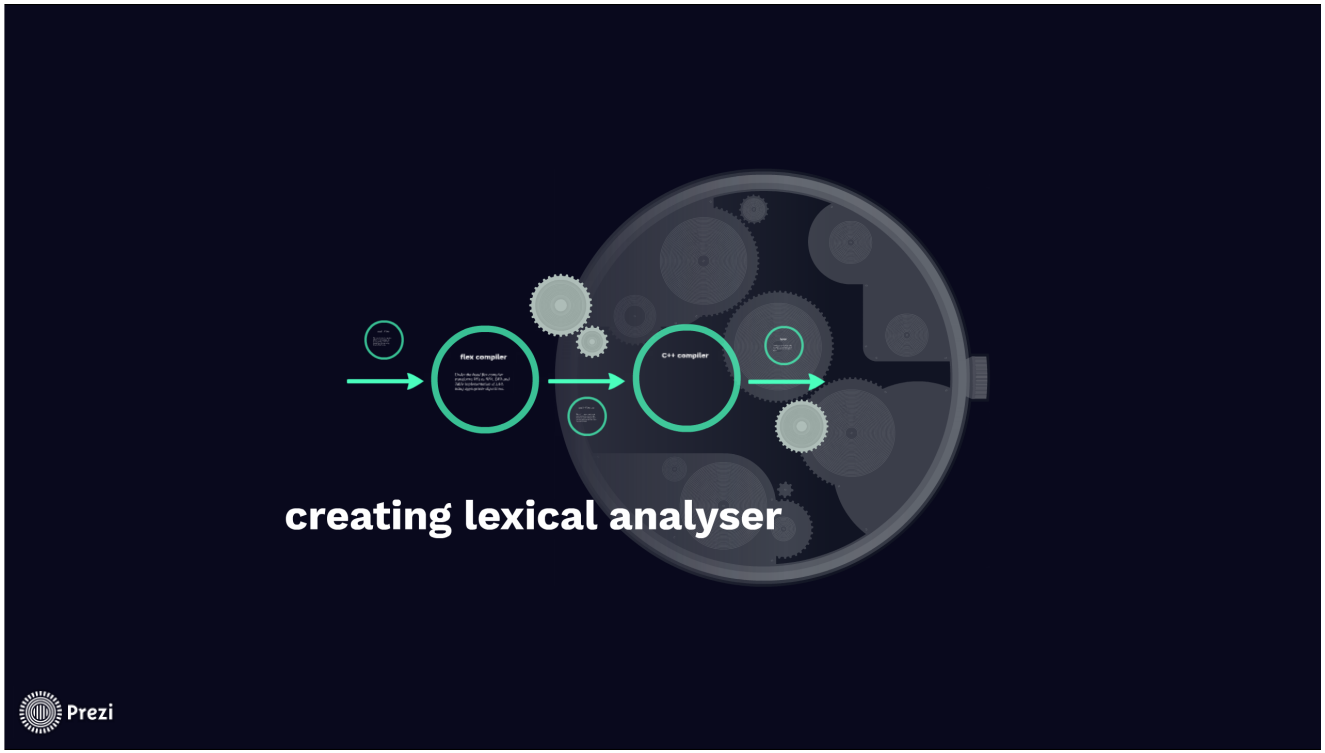
一、词法分析
(3. 正则表达式与自动机理论)

魏恒峰

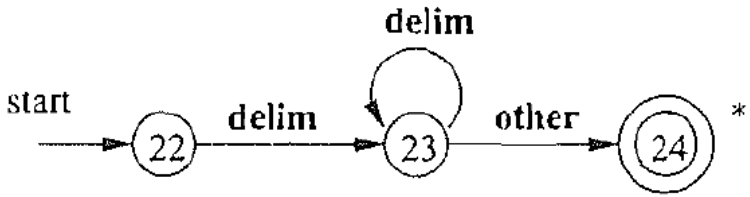
hfwei@nju.edu.cn

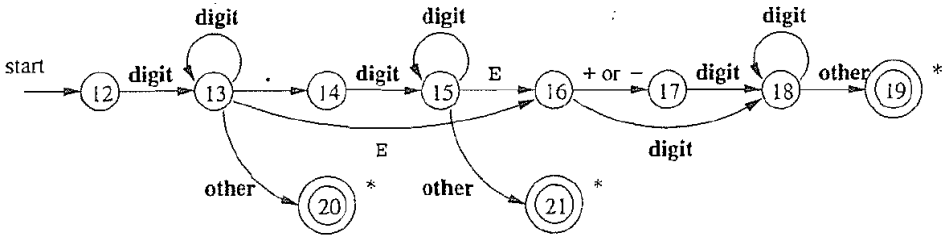
2024 年 03 月 15 日 (周五)

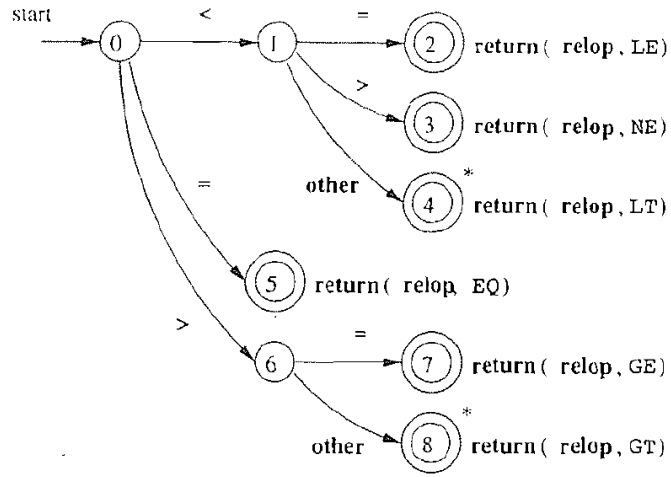


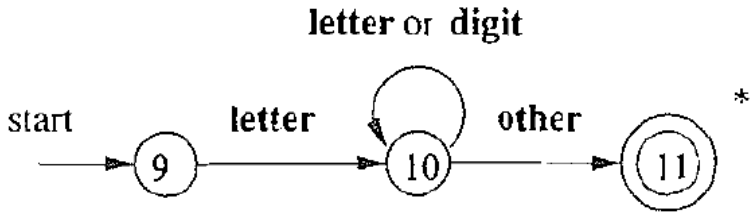
自动化词法分析器生成器


```
LPAREN : '(' ;
RPAREN : ')' ;

ID : (LETTER | '_') WORD* ;
INT : '0' | ([1-9] DIGIT*) ;
FLOAT : INT '.' DIGIT*
      | '.' DIGIT+
      ;

WS : [ \t\r\n]+ -> skip ;

//SL_COMMENT : '//' .*? '\n' -> skip ;
SL_COMMENT2 : '//' ~[\n]* '\n' -> skip;
DOC_COMMENT : '/**' .*? '*/' -> skip ;
ML_COMMENT : '/*' .*? '*/' -> skip ;
```







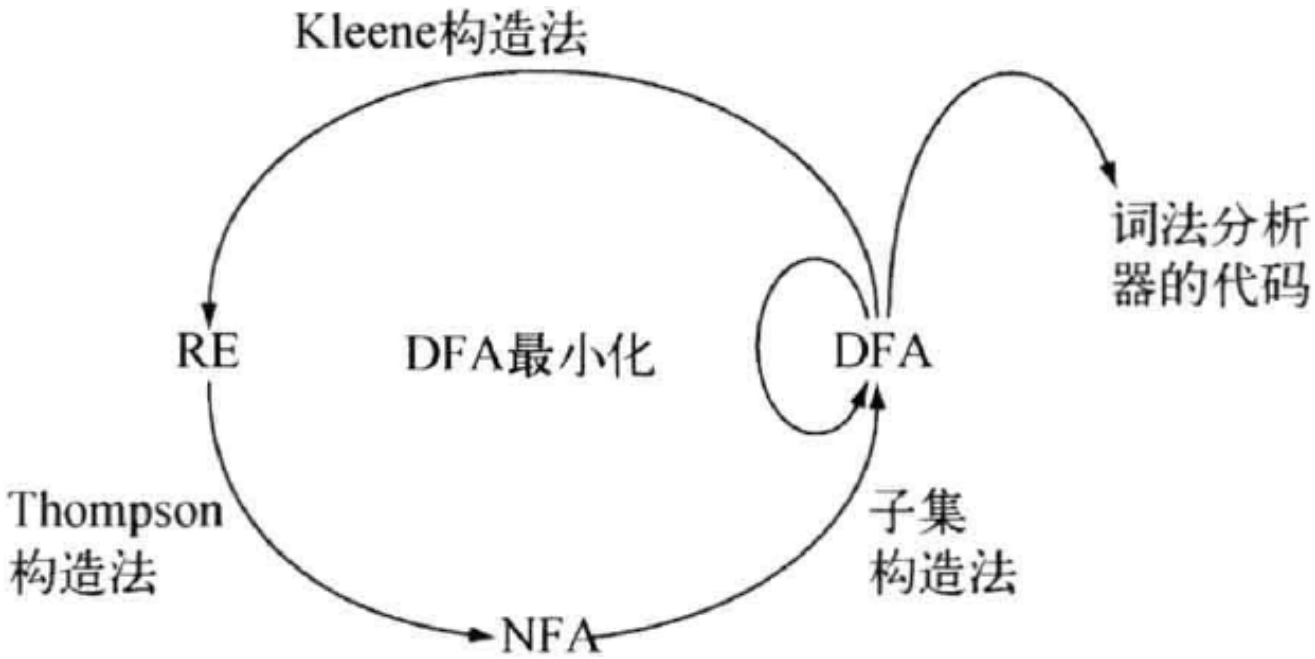


关键点: 合并 22, 12, 9, 0, 根据**下一个字符**即可判定词法单元的类型

否则, 调用错误处理模块 (对应 `other`), 报告**该字符有误**, 忽略该字符。

注意, 在 **real** 与 **sci** 中, 有时需要**回退**, 寻找最长匹配。

目标：正则表达式 RE $\Longrightarrow$ 词法分析器



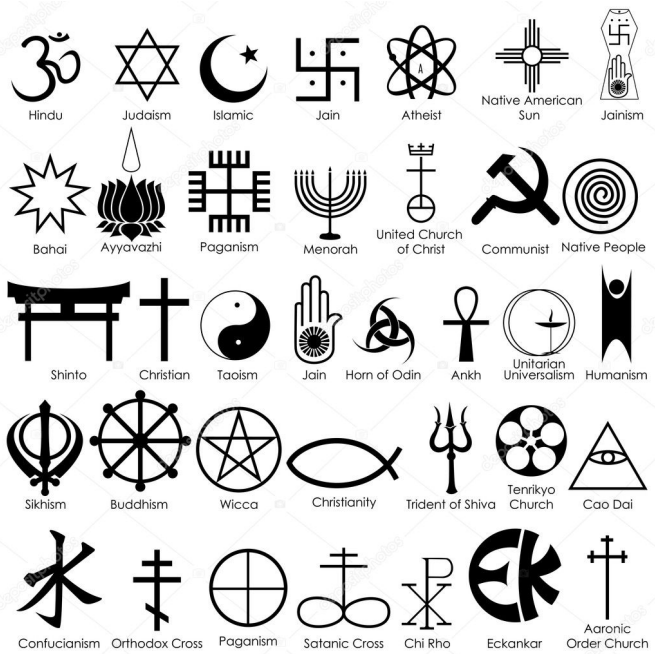
终点固然令人向往，这一路上的风景更是美不胜收


Rethinking what
language is
Stillman
Translations

语言是字符串构成的集合

Definition (字母表)

字母表 Σ 是一个有限的符号集合。



Definition (串)

字母表 Σ 上的**串** (s) 是由 Σ 中符号构成的一个**有穷**序列。

空串：$|\epsilon| = 0$

Definition (串上的 “连接” 运算)

$$x = dog, y = house \qquad xy = doghouse$$

$$s\epsilon = \epsilon s = s$$

Definition (串上的 “连接” 运算)

$$x = dog, y = house \qquad xy = doghouse$$

$$s\epsilon = \epsilon s = s$$

Definition (串上的 “指数” 运算)

$$s^0 \triangleq \epsilon$$

$$s^i \triangleq ss^{i-1}, i > 0$$

Definition (语言)

语言是给定字母表 Σ 上一个任意的**可数的**串集合。

$$\emptyset$$

$$\{\epsilon\}$$

Definition (语言)

语言是给定字母表 Σ 上一个任意的**可数的**串集合。

$$\emptyset$$

$$\{\epsilon\}$$

$$\mathbf{id} : \{a, b, c, a1, a2, \dots\}$$

$$\mathbf{ws} : \{\mathbf{blank}, \mathbf{tab}, \mathbf{newline}\}$$

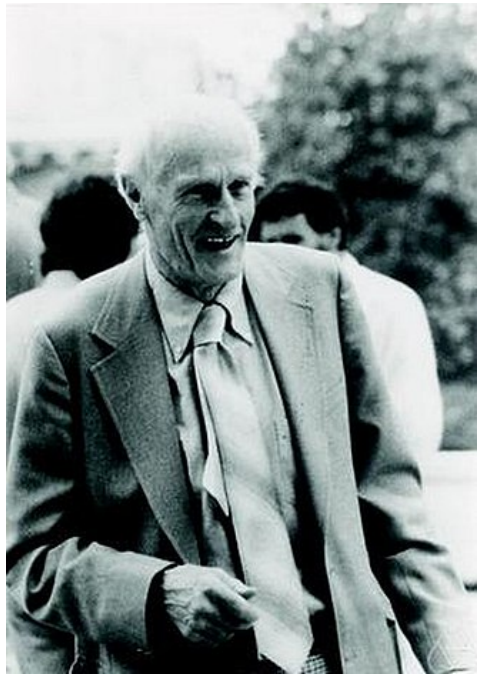
$$\mathbf{if} : \{\texttt{if}\}$$

语言是串的集合

因此, 我们可以通过集合操作**构造**新的语言。

运算	定义和表示
L和M 的并	$L \cup M \doteq \{s \mid s$ 属于L 或者s 属于 $M\}$
L和M 的连接	$LM = \{st \mid s$ 属于 L且　t 属于 $M\}$
L的 Kleene 闭包	$L^* = \cup_{i=0}^{\infty} L^i$
L的正闭包	$L^+ = \cup_{i=1}^{\infty} L^i$

L^* (L^+) 允许我们构造**无穷**集合



Stephen Kleene
(1909 ∼ 1994)

$$L = \{A, B, \ldots, Z, a, b, \ldots, z\}$$

$$D = \{0, 1, \ldots, 9\}$$

运算	定义和表示
L和M 的并	$L \cup M \doteq \{s \mid s$ 属于L 或者s 属于 $M\}$
L和M 的连接	$LM = \{st \mid s$ 属于 L且　t 属于 $M\}$
L的 Kleene 闭包	$L^* = \cup_{i=0}^{\infty} L^i$
L的正闭包	$L^+ = \cup_{i=1}^{\infty} L^i$

$$L = \{A, B, \ldots, Z, a, b, \ldots, z\}$$

$$D = \{0, 1, \ldots, 9\}$$

运算	定义和表示
L和M 的并	$L \cup M \doteq \{s \mid s$ 属于L 或者s 属于 $M\}$
L和M 的连接	$LM = \{st \mid s$ 属于 L且　t 属于 $M\}$
L的 Kleene 闭包	$L^* = \cup_{i=0}^{\infty} L^i$
L的正闭包	$L^+ = \cup_{i=1}^{\infty} L^i$

$L \cup D$　　LD　　L^4　　L^*　　D^+　　$L(L \cup D)^*$

下面向大家隆重介绍简洁、优雅、强大的**正则表达式**

每个正则表达式 r 对应一个正则语言 $L(r)$

正则表达式是**语法**, 正则语言是**语义**

`ID: [a-zA-Z][a-zA-Z0-9]*` $\{a1, a2, ab, \ldots\}$

Definition (正则表达式)

给定字母表 Σ, Σ 上的正则表达式**由且仅由**以下规则定义:

(1) ϵ 是正则表达式;

(2) $\forall a \in \Sigma$, a 是正则表达式;

(3) 如果 r 是正则表达式, 则 (r) 是正则表达式;

(4) 如果 r 与 s 是正则表达式, 则 $r|s$, rs, r^* 也是正则表达式。

运算优先级: $() \succ * \succ$ 连接 $\succ |$

$$(a)|((b)^*(c)) \equiv a|b^*c$$

每个正则表达式 r 对应一个正则语言 $L(r)$

Definition (正则表达式对应的正则语言)

$$L(\epsilon) = \{\epsilon\} \tag{1}$$

$$L(a) = \{a\}, \forall a \in \Sigma \tag{2}$$

$$L((r)) = L(r) \tag{3}$$

$$L(r|s) = L(r) \cup L(s) \quad L(rs) = L(r)L(s) \quad L(r^*) = (L(r))^* \tag{4}$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$L((a|b)^*)$$

$$\Sigma = \{a, b\}$$

$$L(a|b) = \{a, b\}$$

$$L((a|b)(a|b))$$

$$L(a^*)$$

$$L((a|b)^*)$$

$$L(a|a^*b)$$


So
EASY

表达式	匹配	例子
c	单个非运算符字符 c	a
\c	字符 c 的字面值	*
"s"	串 s 的字面值	"**"
.	除换行符以外的任何字符	a.*b
^	一行的开始	^abc
$	行的结尾	abc$
[s]	字符串 s 中的任何一个字符	[abc]
[^s]	不在串 s 中的任何一个字符	[^abc]
r*	和 r 匹配的零个或多个串连接成的串	a*
r+	和 r 匹配的一个或多个串连接成的串	a+
r?	零个或一个 r	a?
$r\{m,n\}$	最少 m 个，最多 n 个 r 的重复出现	a{1,5}
r_1r_2	r_1 后加上 r_2	ab
$r_1 \mid r_2$	r_1 或 r_2	a|b
(r)	与 r 相同	(a|b)
r_1/r_2	后面跟有 r_2 时的 r_1	abc/123

表达式	匹配	例子
c	单个非运算符字符 c	a
\c	字符 c 的字面值	*
"s"	串 s 的字面值	"**"
.	除换行符以外的任何字符	a.*b
^	一行的开始	^abc
$	行的结尾	abc$
[s]	字符串 s 中的任何一个字符	[abc]
[^s]	不在串 s 中的任何一个字符	[^abc]
r*	和 r 匹配的零个或多个串连接成的串	a*
r+	和 r 匹配的一个或多个串连接成的串	a+
r?	零个或一个 r	a?
r{m,n}	最少 m 个，最多 n 个 r 的重复出现	a{1,5}
r_1r_2	r_1 后加上 r_2	ab
$r_1 \mid r_2$	r_1 或 r_2	a|b
(r)	与 r 相同	(a|b)
r_1/r_2	后面跟有 r_2 时的 r_1	abc/123

$[0-9]$ $[a-zA-Z]$ ^ $

正则表达式简记法

Vim	Java	ASCII	Description
	\p{ASCII}	[\x00-\x7F]	ASCII characters
	\p{Alnum}	[A-Za-z0-9]	Alphanumeric characters
\w	\w	[A-Za-z0-9_]	Alphanumeric characters plus "_"
\W	\W	[^A-Za-z0-9_]	Non-word characters
\a	\p{Alpha}	[A-Za-z]	Alphabetic characters
\s	\p{Blank}	[\t]	Space and tab
\< \>	\b	(?<=\W)(?=\w)|(?<=\w)(?=\W)	Word boundaries
	\B	(?<=\W)(?=\W)|(?<=\w)(?=\w)	Non-word boundaries
	\p{Cntrl}	[\x00-\x1F\x7F]	Control characters
\d	\p{Digit} or \d	[0-9]	Digits
\D	\D	[^0-9]	Non-digits
	\p{Graph}	[\x21-\x7E]	Visible characters
\l	\p{Lower}	[a-z]	Lowercase letters
\p	\p{Print}	[\x20-\x7E]	Visible characters and the space character
	\p{Punct}	[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
_s	\p{Space} or \s	[\t\r\n\v\f]	Whitespace characters
\S	\S	[^ \t\r\n\v\f]	Non-whitespace characters
\u	\p{Upper}	[A-Z]	Uppercase letters
\x	\p{XDigit}	[A-Fa-f0-9]	Hexadecimal digits


THAT'S IT?
TOO EASY.
memegenerator.net

$$\Big(0|\big(1(01^*0)^*1\big)\Big)^*$$

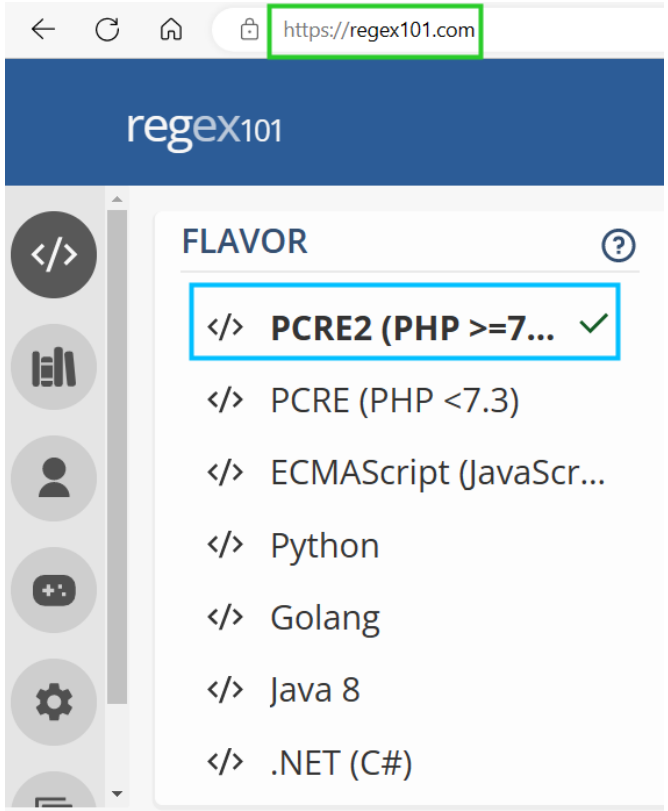
REGULAR EXPRESSION v1

/ ^(0|(1(01*0)*1))*$

TEST STRING

```
0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
10000
10001
10010
10011
10100
```

https://regex101.com/r/C0m3kB/1



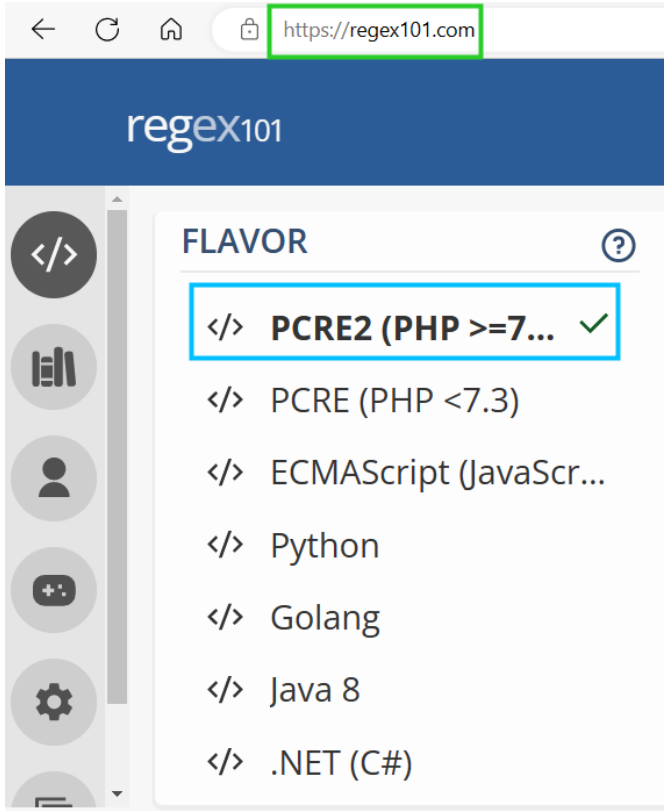
REGULAR EXPRESSION v1

/ ^(0|(1(01*0)*1))*$

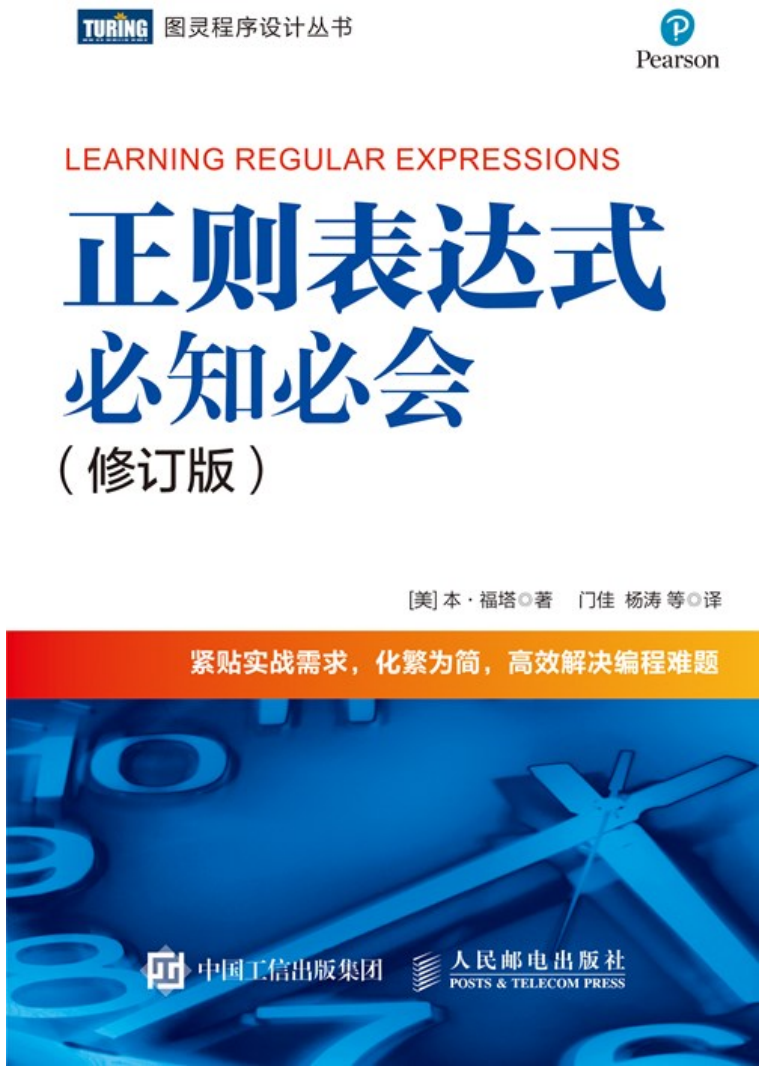
TEST STRING

0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
10000
10001
10010
10011
10100

https://regex101.com/r/C0m3kB/1



3 的倍数 (二进制表示)


TURING 图灵程序设计丛书
Pearson
LEARNING REGULAR EXPRESSIONS
正则表达式
必知必会
（修订版）
[美] 本·福塔◎著　门佳 杨涛 等◎译
紧贴实战需求，化繁为简，高效解决编程难题
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Definition (NFA (Nondeteministic Finite Automaton))

非确定性有穷自动机 $\mathcal{A}$ 是一个五元组 $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$:

(1) 字母表 Σ ($\epsilon \notin \Sigma$)

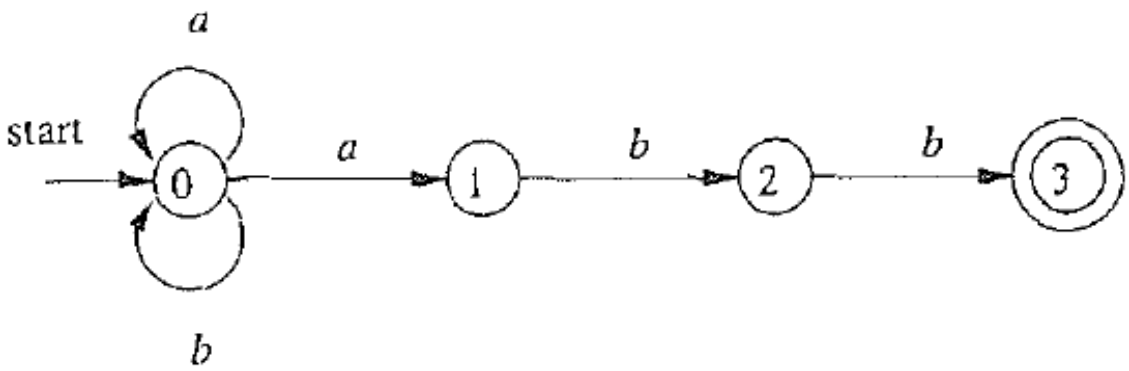
(2) 有穷的**状态集合** S

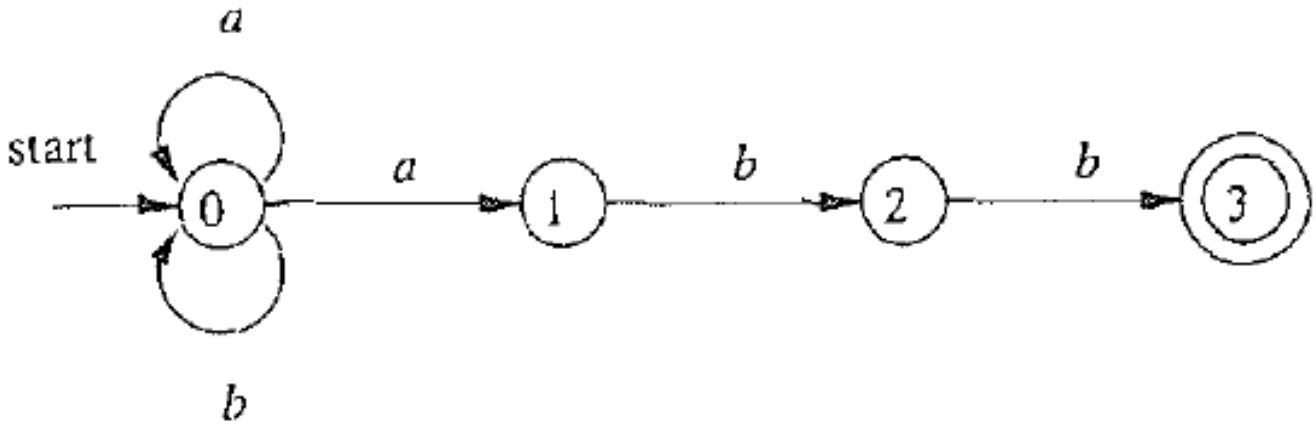
(3) 唯一的初始状态 $s_0 \in S$

(4) **状态转移函数** δ

$$\delta : S \times (\Sigma \cup \{\epsilon\}) \to 2^S$$

(5) 接受状态集合 $F \subseteq S$





状态	a	b	ϵ
0	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

约定: 所有没有对应出边的字符默认指向**“空状态”** $\emptyset$

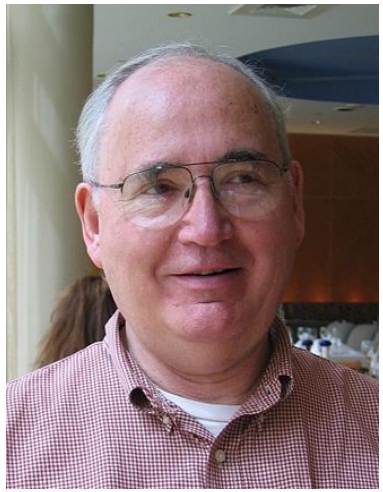
Michael O. Rabin
(1931 ∼)

Finite Automata and Their Decision Problems‡

Abstract: Finite automata are considered in this paper as instruments for classifying finite tapes. Each one-tape automaton defines a set of tapes, a two-tape automaton defines a set of pairs of tapes, et cetera. The structure of the defined sets is studied. Various generalizations of the notion of an automaton are introduced and their relation to the classical automata is determined. Some decision problems concerning automata are shown to be solvable by effective algorithms; others turn out to be unsolvable by algorithms.

发表于 1959 年;

1976 年, 共享图灵奖



Dana Scott (1932 ∼)

"which introduced the idea of ***nondeterministic machines****,*
which has proved to be an enormously valuable concept."

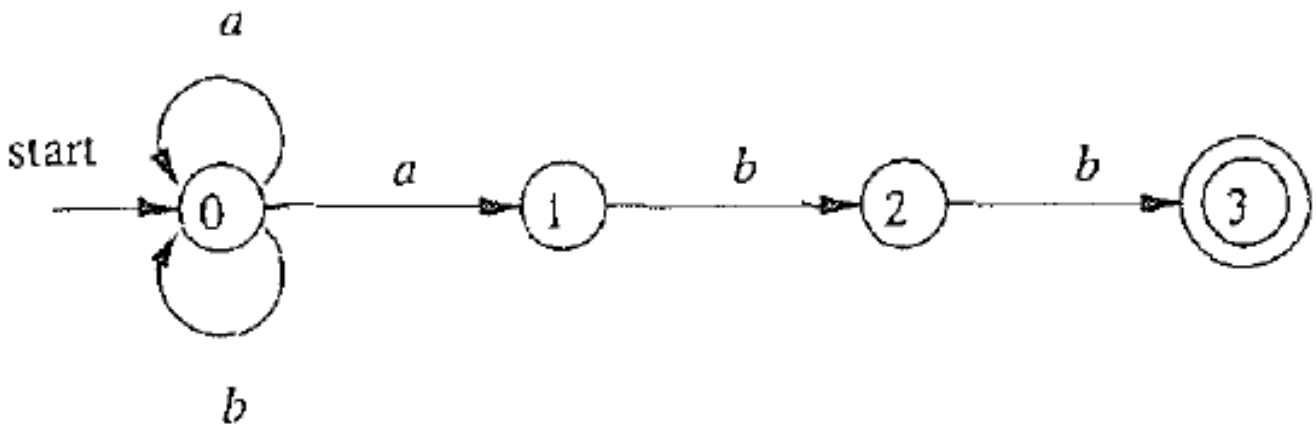
(非确定性) 有穷自动机是一类极其简单的**计算**装置

它可以**识别** (接受/拒绝) Σ 上的字符串

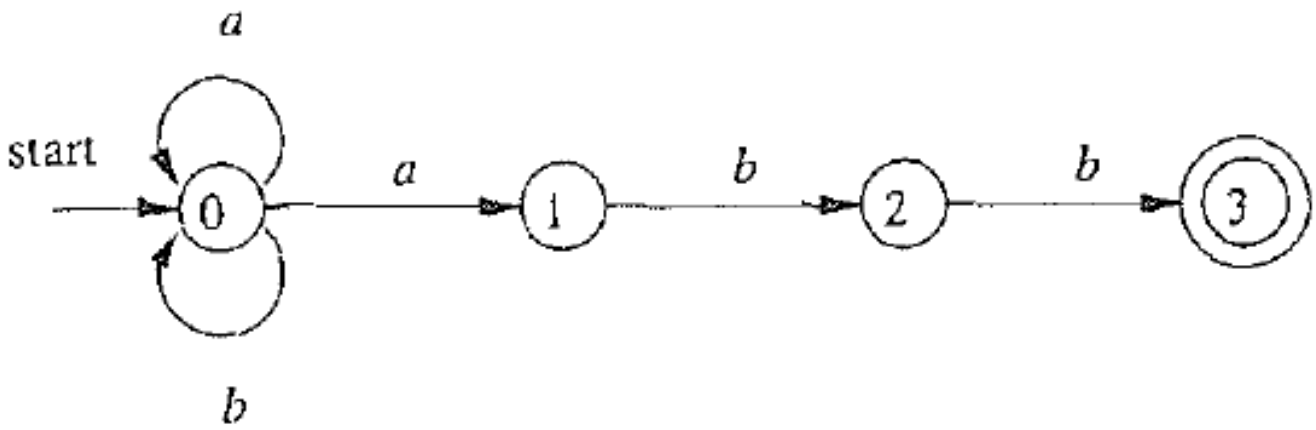
Definition (接受 (Accept))

(非确定性) 有穷自动机 $\mathcal{A}$ 接受字符串 x, 当且仅当**存在**一条从开始状态 s_0 到**某个**接受状态 $f \in F$、标号为 x 的路径。

因此, $\mathcal{A}$ 定义了一种**语言** $L(\mathcal{A})$: 它能接受的所有字符串构成的集合

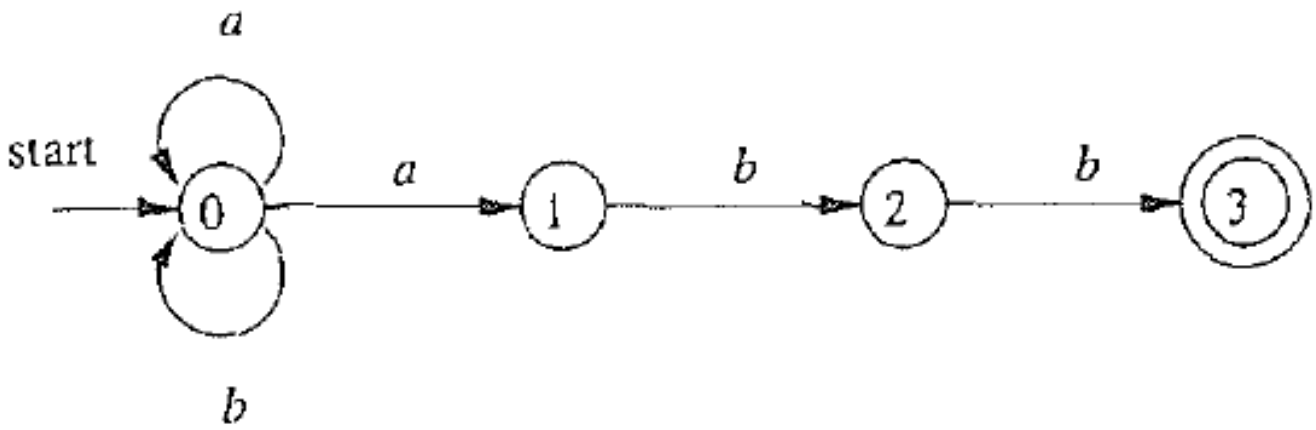


$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$



$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$

$$L(\mathcal{A}) =$$



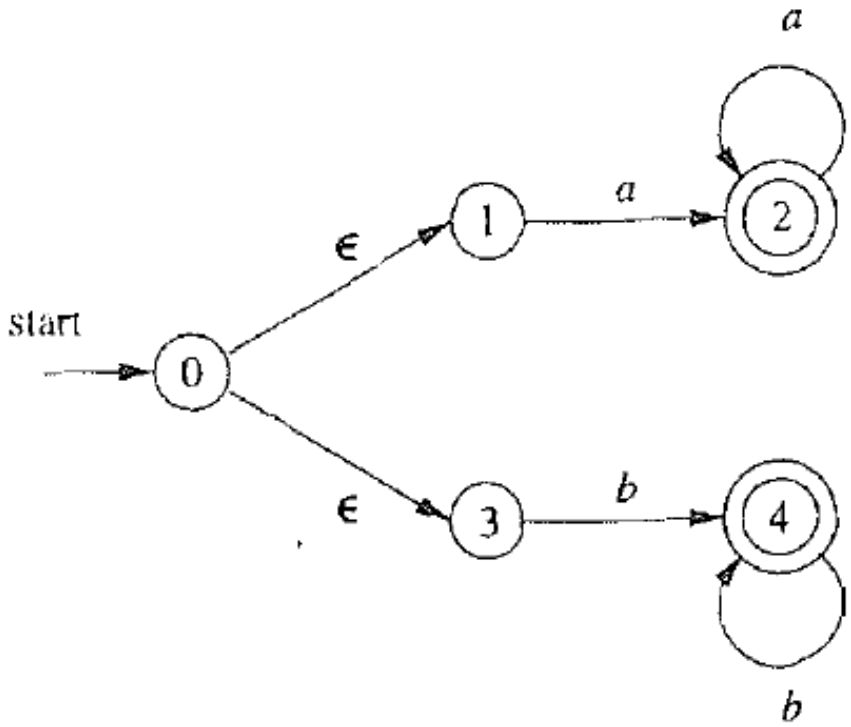
$$aabb \in L(\mathcal{A}) \qquad abababab \notin L(\mathcal{A})$$

$$L(\mathcal{A}) = L((a|b)^*abb)$$

关于自动机 $\mathcal{A}$ 的**两个基本问题**:

- **Membership 问题:** 给定字符串 x, $x \in L(\mathcal{A})$?
- $L(\mathcal{A})$ 究竟是什么?

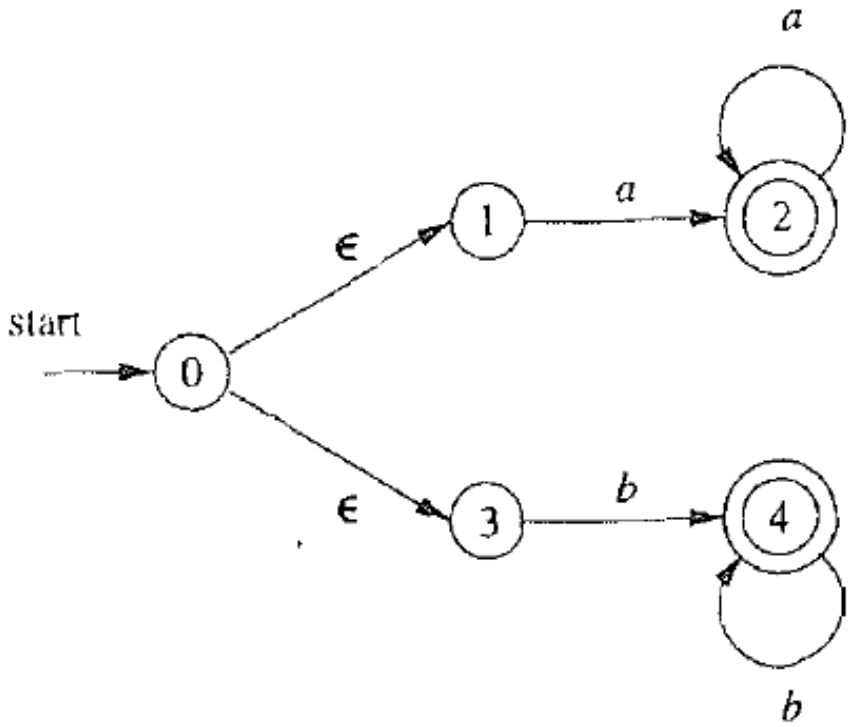
JFLAP



$aaa \in \mathcal{A}?$ $aab \in \mathcal{A}?$

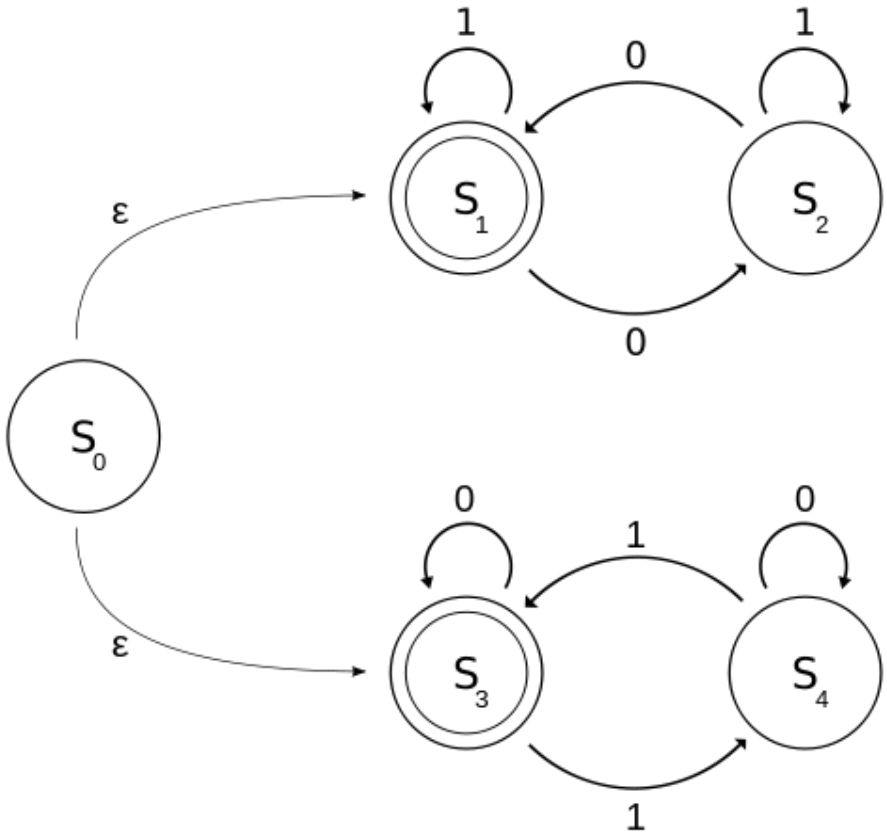
$L(\mathcal{A}) =$

JFLAP

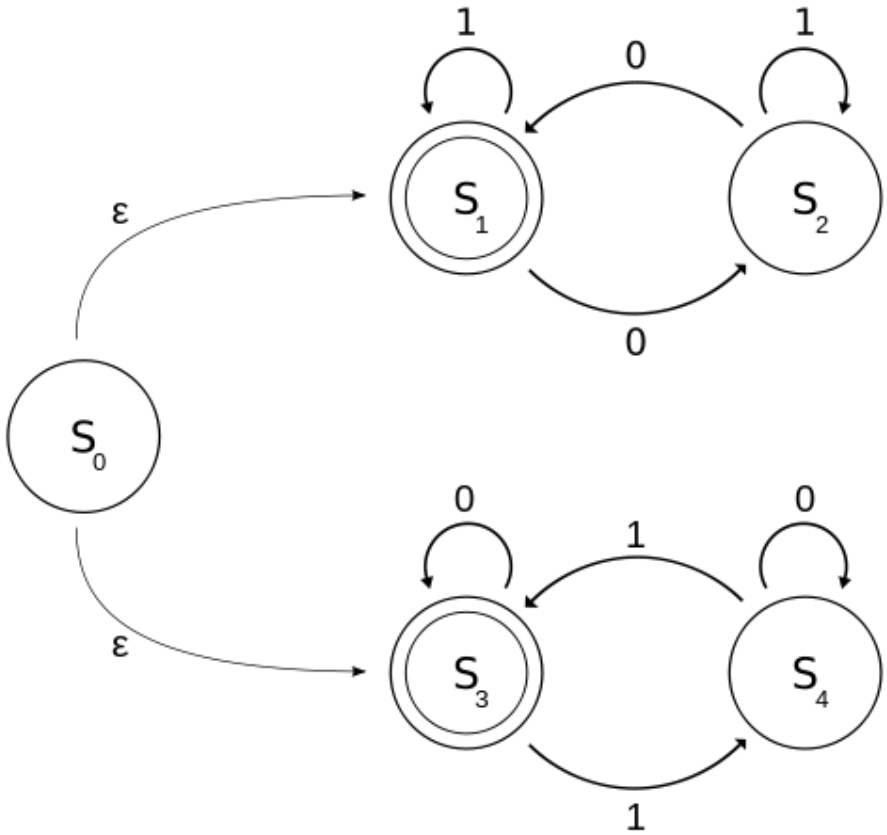


$aaa \in \mathcal{A}?$ $aab \in \mathcal{A}?$

$$L(\mathcal{A}) = L((aa^*|bb^*))$$



$1011 \in L(\mathcal{A})?$ $0011 \in L(\mathcal{A})?$



$1011 \in L(\mathcal{A})?$ $0011 \in L(\mathcal{A})?$

$L(\mathcal{A}) = \{$包含偶数个 1 或偶数个 0 的 01 串$\}$

Definition (DFA (Deterministic Finite Automaton))

确定性有穷自动机 $\mathcal{A}$ 是一个五元组 $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$:

(1) 字母表 Σ ($\epsilon \notin \Sigma$)

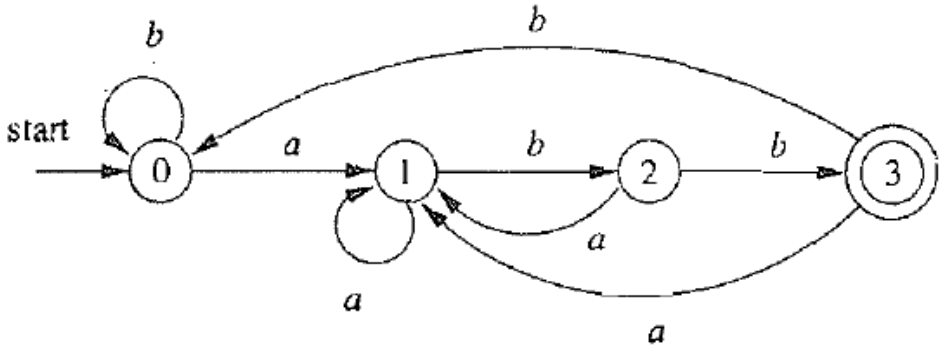
(2) 有穷的**状态集合** S

(3) **唯一**的初始状态 $s_0 \in S$

(4) **状态转移函数** δ

$$\delta : S \times \Sigma \to S$$

(5) 接受状态集合 $F \subseteq S$



Definition (DFA (Deterministic Finite Automaton))

确定性有穷自动机 $\mathcal{A}$ 是一个五元组 $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$:

(1) 字母表 Σ ($\epsilon \notin \Sigma$)

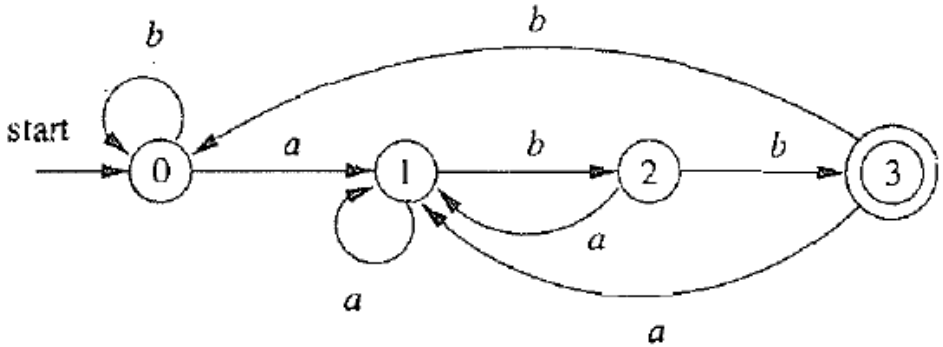
(2) 有穷的**状态集合** S

(3) **唯一**的初始状态 $s_0 \in S$

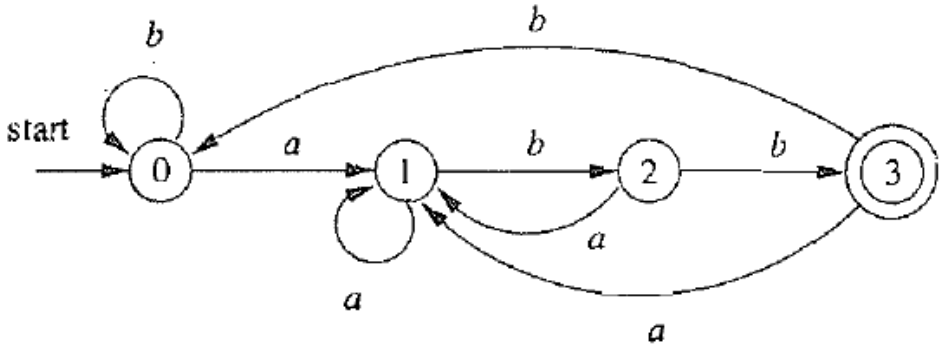
(4) **状态转移函数** δ

$$\delta : S \times \Sigma \to S$$

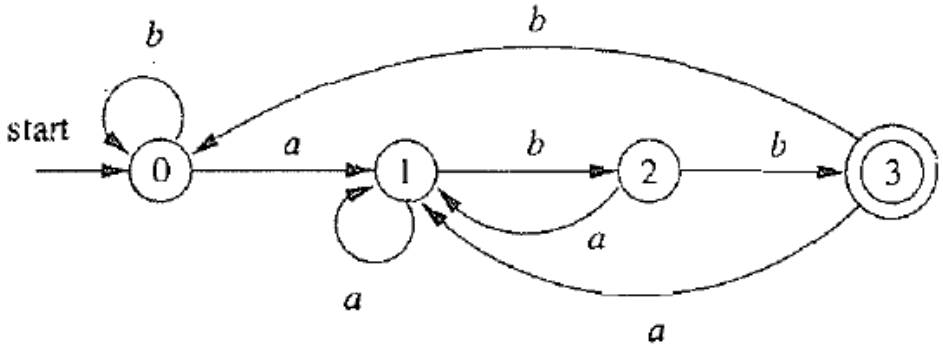
(5) 接受状态集合 $F \subseteq S$



约定: 所有没有对应出边的字符默认指向一个 **"死状态"**

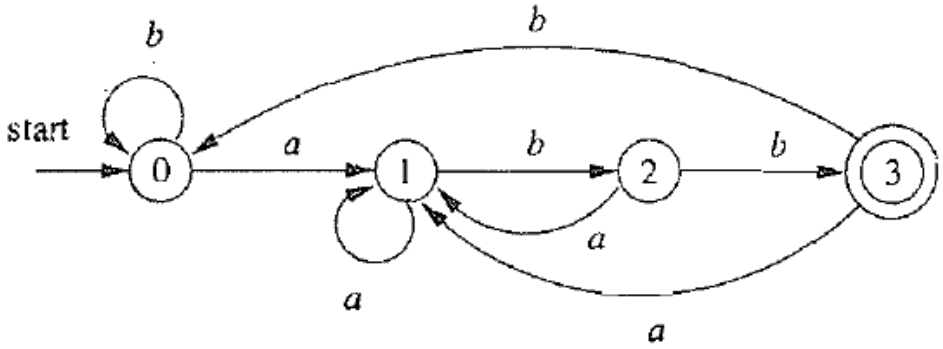


$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$



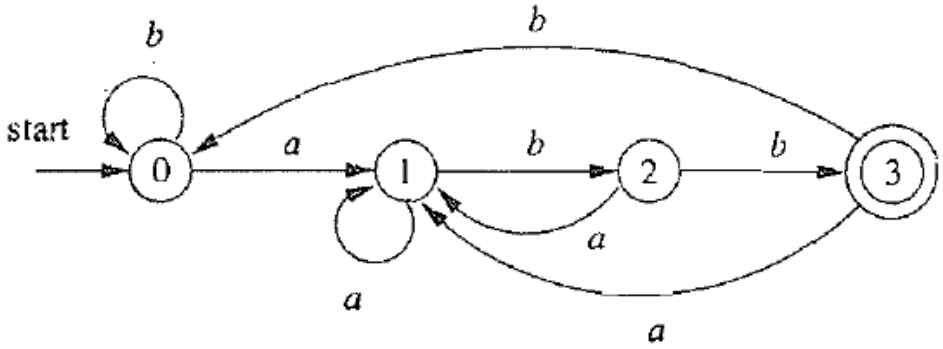
$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$

$$L(\mathcal{A}) =$$



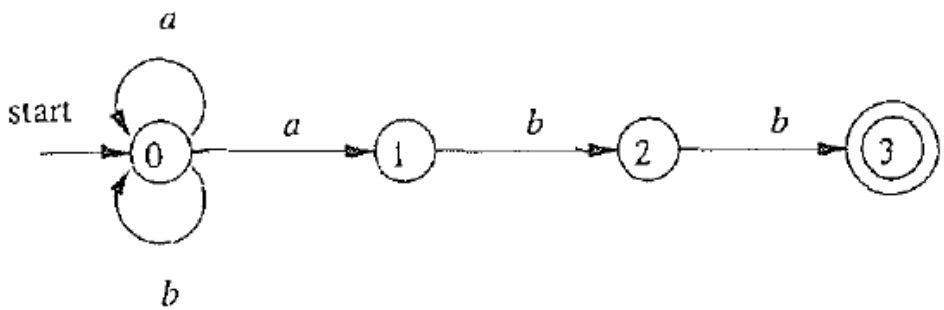
$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$

$$L(\mathcal{A}) = L((a|b)^*abb)$$



$$aabb \in L(\mathcal{A}) \qquad ababab \notin L(\mathcal{A})$$

$$L(\mathcal{A}) = L((a|b)^*abb)$$



NFA 简洁易于理解, 便于描述语言 $L(\mathcal{A})$

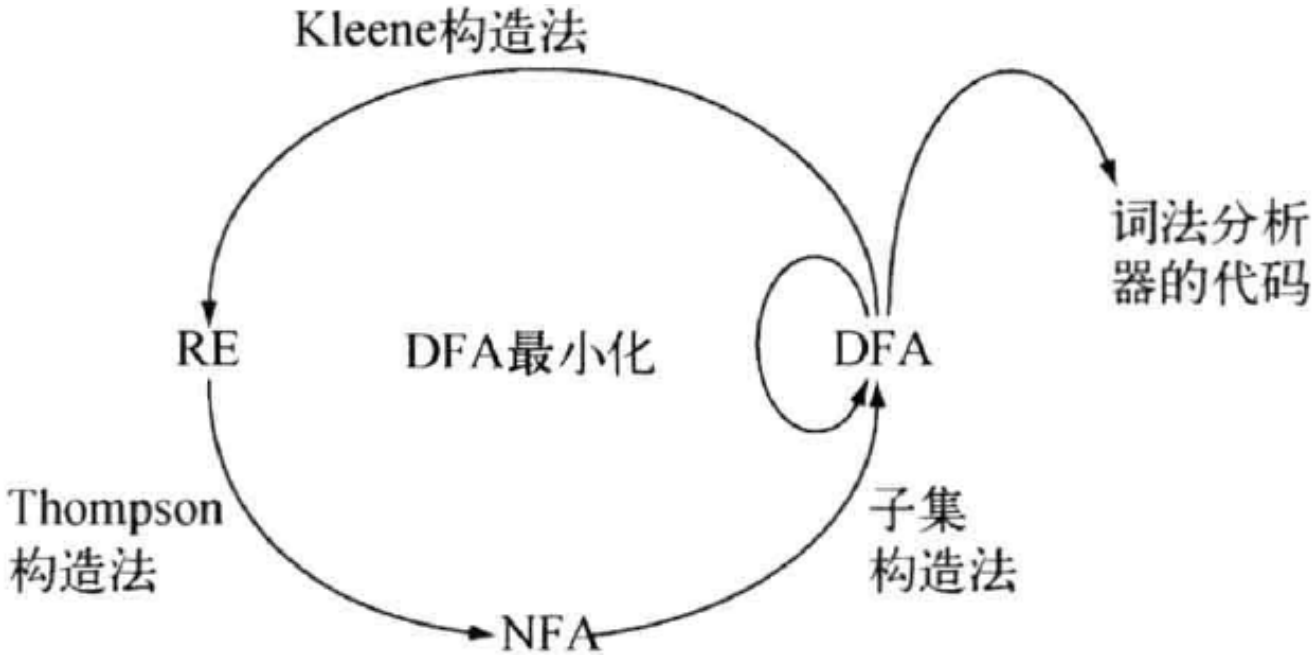
DFA 易于判断 $x \in L(\mathcal{A})$, 适合产生词法分析器

NFA 简洁易于理解, 便于描述语言 $L(\mathcal{A})$

DFA 易于判断 $x \in L(\mathcal{A})$, 适合产生词法分析器

用 NFA 描述语言, 用 DFA 实现词法分析器

$$\text{RE} \Longrightarrow \text{NFA} \Longrightarrow \text{DFA} \Longrightarrow \text{词法分析器}$$



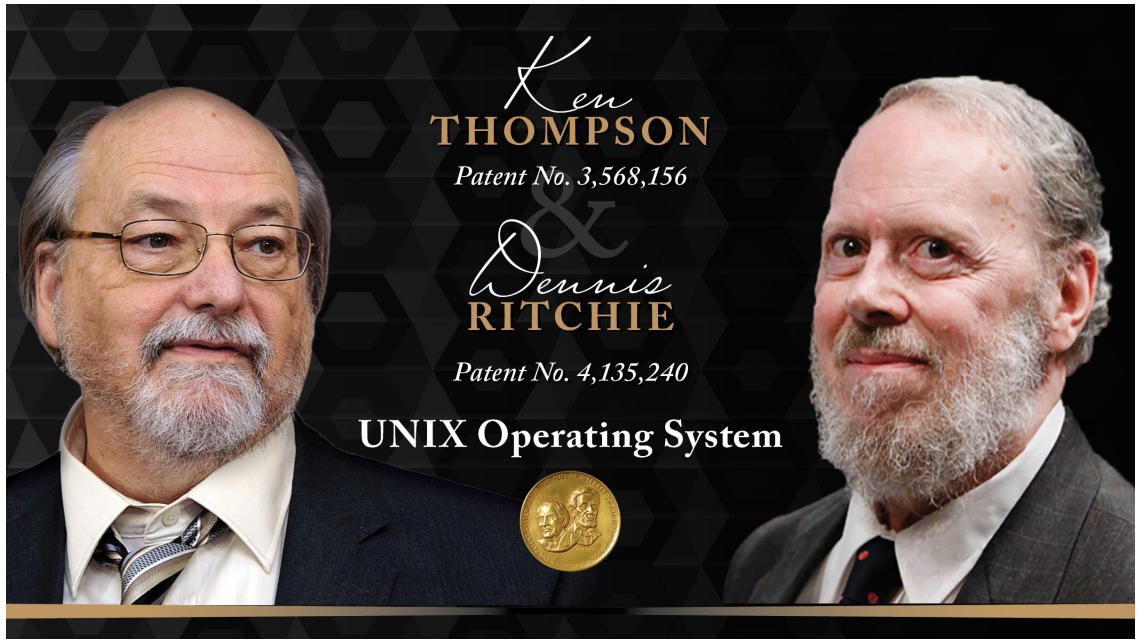
$$\text{RE} \implies \text{NFA}$$

$$\boxed{r \implies N(r)}$$

要求：$L(N(r)) = L(r)$

从 RE 到 NFA: **Thompson 构造法**

从 RE 到 NFA: Thompson 构造法



Turing Award, 1983

Thompson 构造法的基本思想: **按结构归纳**

Definition (正则表达式)

给定字母表 Σ, Σ 上的正则表达式**由且仅由**以下规则定义:

(1) ϵ 是正则表达式;

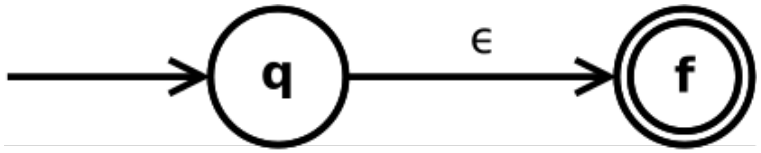
(2) $\forall a \in \Sigma$, a 是正则表达式;

(3) 如果 s 是正则表达式, 则 (s) 是正则表达式;

(4) 如果 s 与 t 是正则表达式, 则 $s|t$, st, s^* 也是正则表达式。

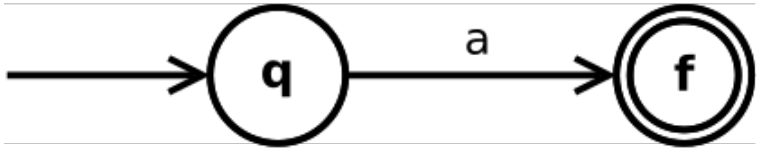
ϵ 是正则表达式。

ϵ 是正则表达式。



$a \in \Sigma$ 是正则表达式。

$a \in \Sigma$ 是正则表达式。



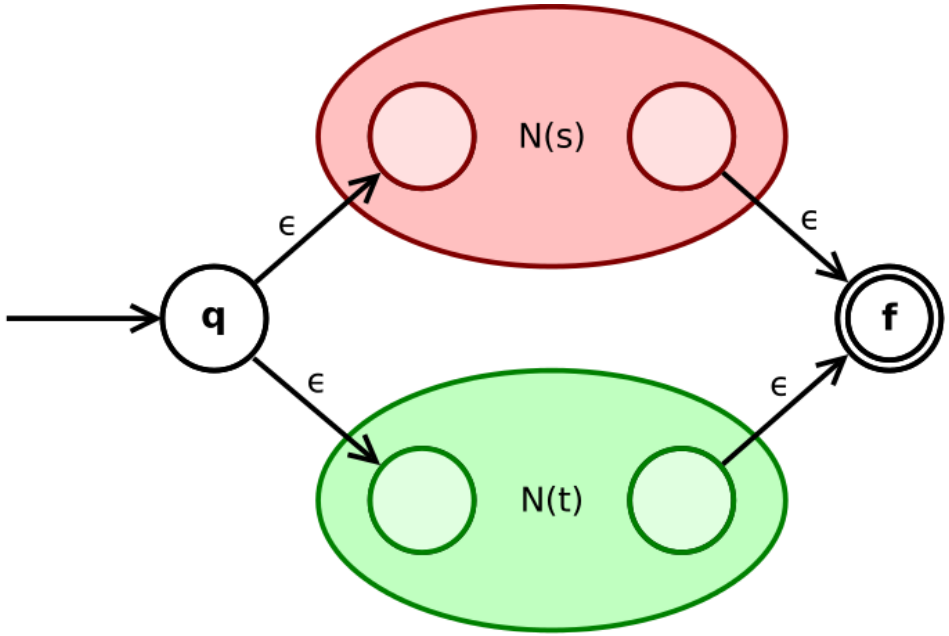
如果 s 是正则表达式, 则 (s) 是正则表达式。

如果 s 是正则表达式, 则 (s) 是正则表达式。

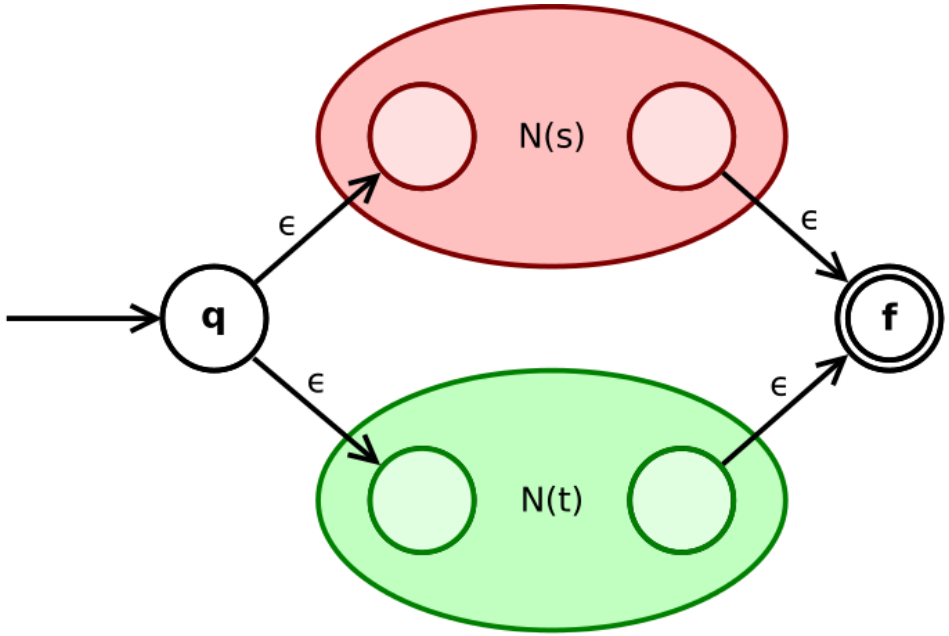
$$N((s)) = N(s).$$

如果 s, t 是正则表达式, 则 $s|t$ 是正则表达式。

如果 s, t 是正则表达式, 则 $s|t$ 是正则表达式。

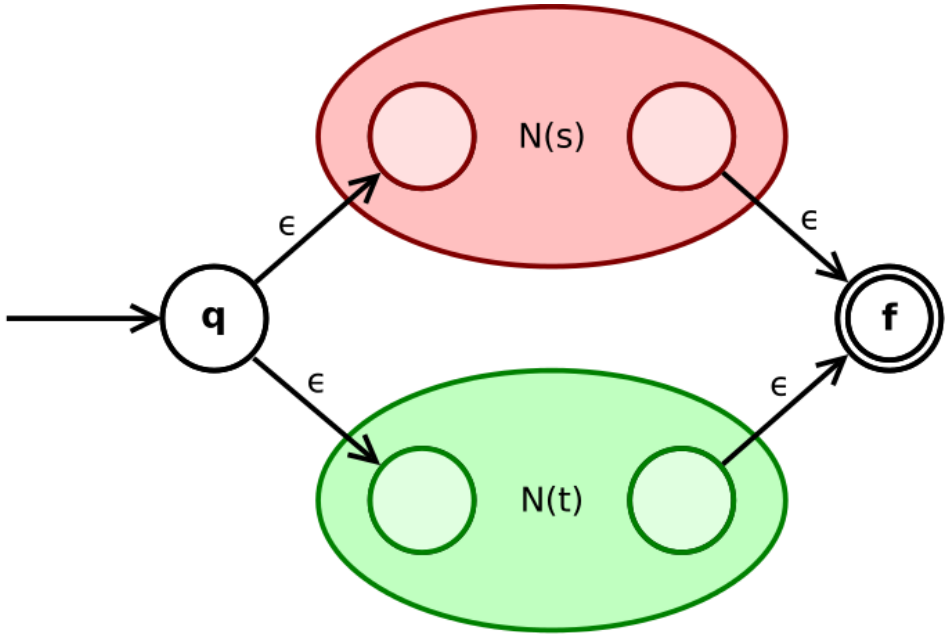


如果 s, t 是正则表达式, 则 $s|t$ 是正则表达式。



Q : 如果 $N(s)$ 或 $N(t)$ 的开始状态或接受状态不唯一, 怎么办?

如果 s, t 是正则表达式, 则 $s|t$ 是正则表达式。

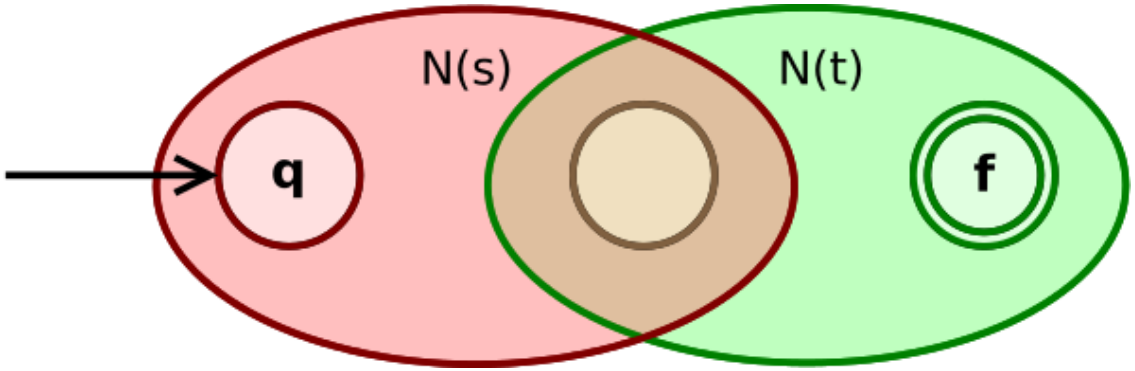


Q : 如果 $N(s)$ 或 $N(t)$ 的开始状态或接受状态不唯一, 怎么办?

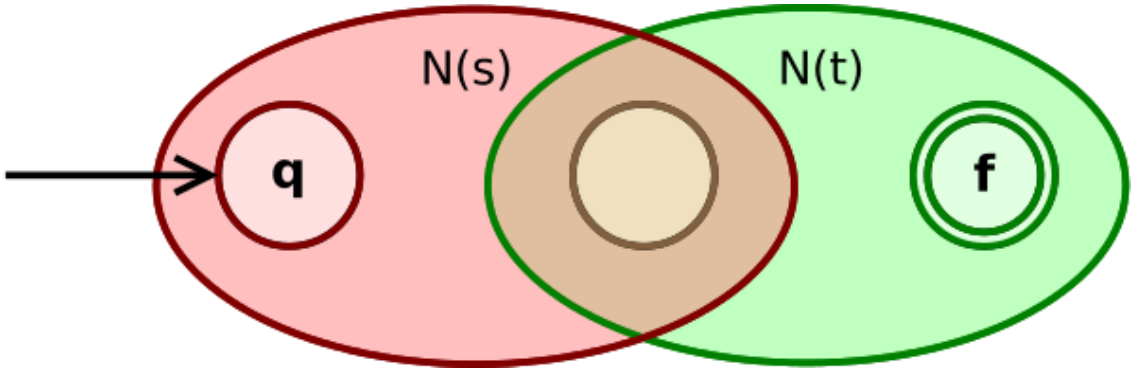
根据**归纳假设**, $N(s)$ 与 $N(t)$ 的开始状态与接受状态均**唯一**。

如果 s, t 是正则表达式, 则 st 是正则表达式。

如果 s, t 是正则表达式, 则 st 是正则表达式。



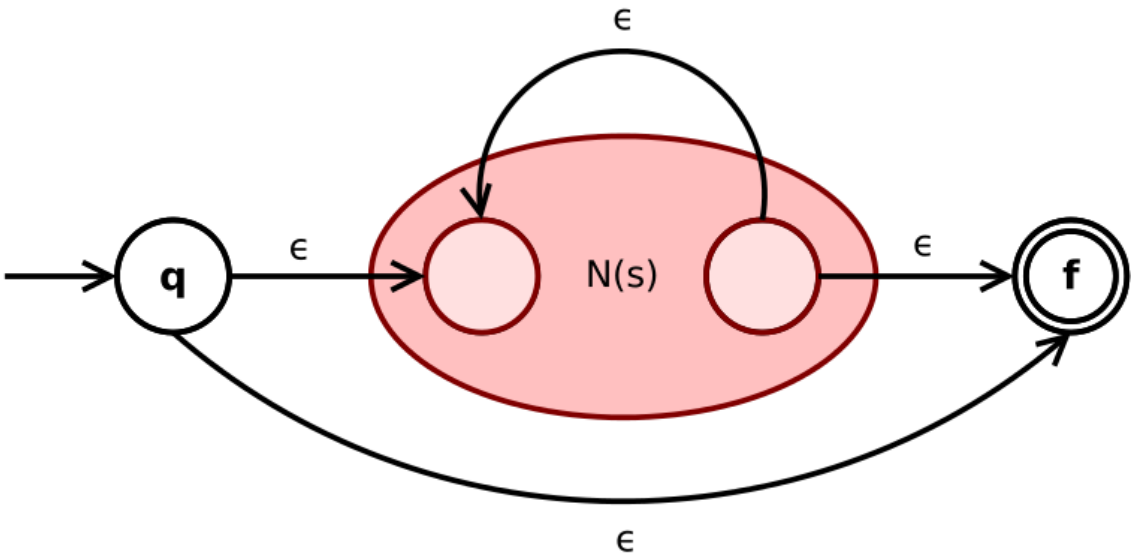
如果 s, t 是正则表达式, 则 st 是正则表达式。



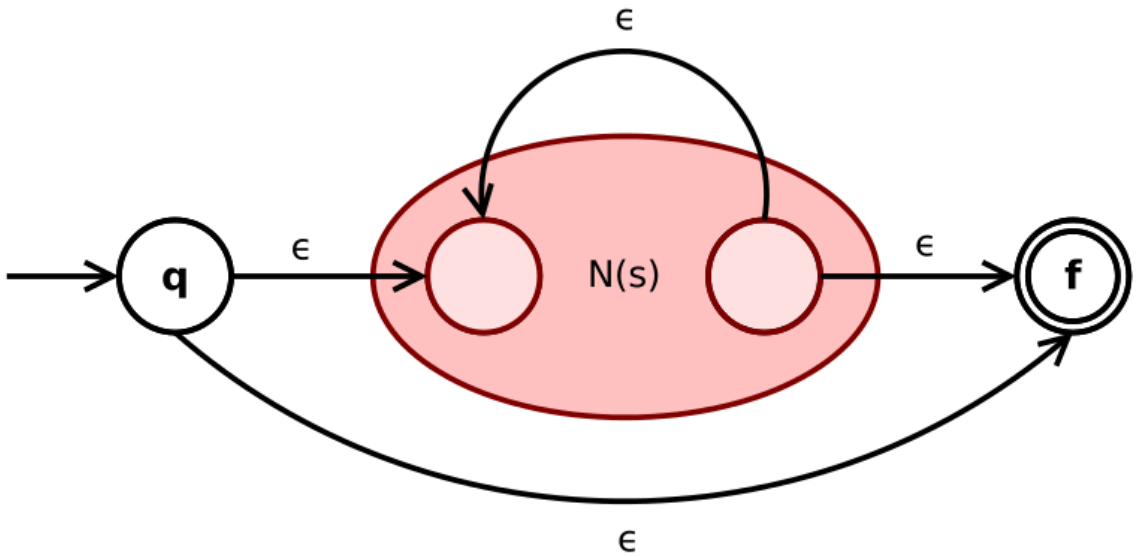
根据**归纳假设**, $N(s)$ 与 $N(t)$ 的开始状态与接受状态均**唯一**。

如果 s 是正则表达式, 则 s^* 是正则表达式。

如果 s 是正则表达式, 则 s^* 是正则表达式。



如果 s 是正则表达式, 则 s^* 是正则表达式。



根据**归纳假设**, $N(s)$ 的开始状态与接受状态**唯一**。

$N(r)$ 的**性质**以及 Thompson 构造法**复杂度分析**

1. $N(r)$ 的开始状态与接受状态均**唯一**。
2. 开始状态没有入边, 接受状态没有出边。

$N(r)$ 的**性质**以及 Thompson 构造法**复杂度分析**

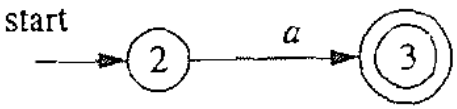
1. $N(r)$ 的开始状态与接受状态均**唯一**。
2. 开始状态没有入边, 接受状态没有出边。
3. $N(r)$ 的**状态数** $|S| \leq 2 \times |r|$。
   ($|r|$: r 中运算符与运算分量的总和)

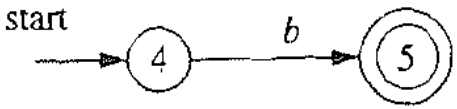
$N(r)$ 的**性质**以及 Thompson 构造法**复杂度分析**

1. $N(r)$ 的开始状态与接受状态均**唯一**。
2. 开始状态没有入边, 接受状态没有出边。
3. $N(r)$ 的**状态数** $|S| \leq 2 \times |r|$。
   ($|r|$: r 中运算符与运算分量的总和)
4. 每个状态最多有两个 ϵ-入边与两个 ϵ-出边。
5. $\forall a \in \Sigma$, 每个状态最多有一个 a-入边与一个 a-出边。

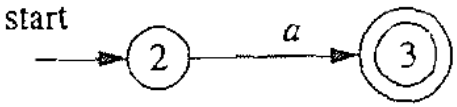
$$r = (a|b)^*abb$$

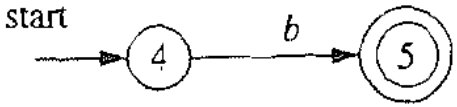
$$r = (a|b)^*abb$$

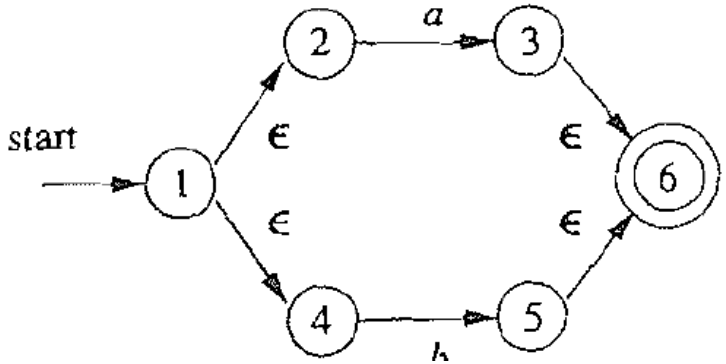




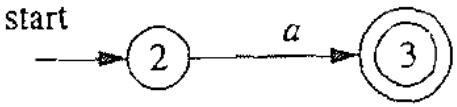
$$r = (a|b)^*abb$$

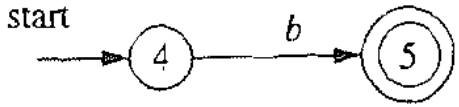


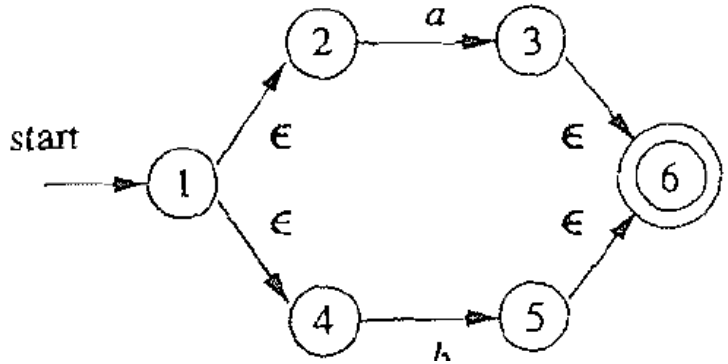


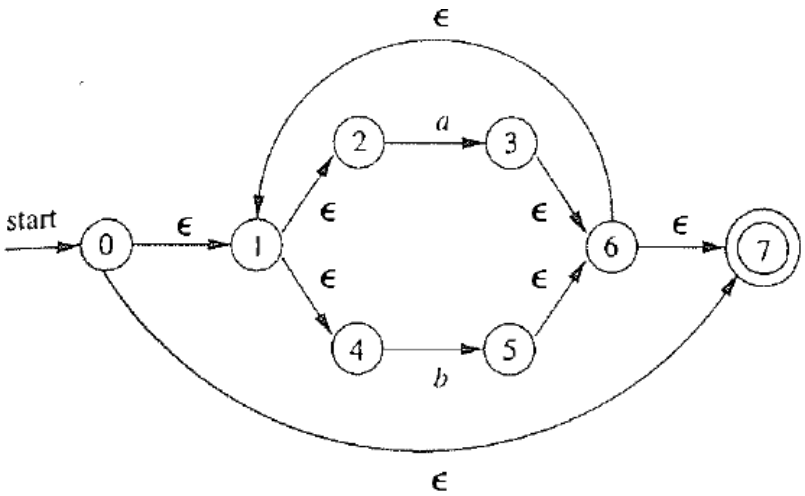


$$r = (a|b)^*abb$$

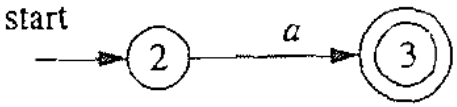

start
2
a
3

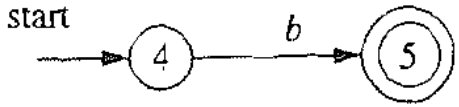

start
4
b
5

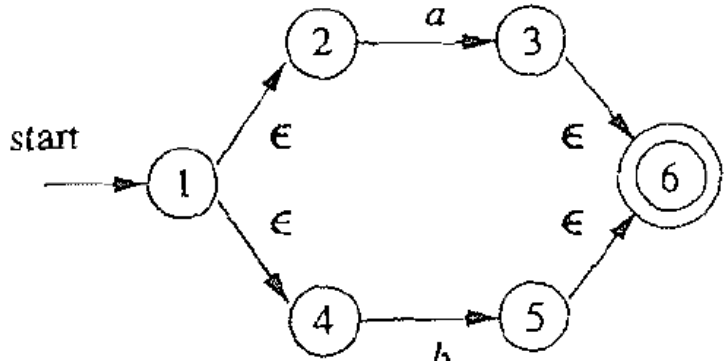

start
1
2
a
3
ϵ
ϵ
ϵ
ϵ
4
5
6
b

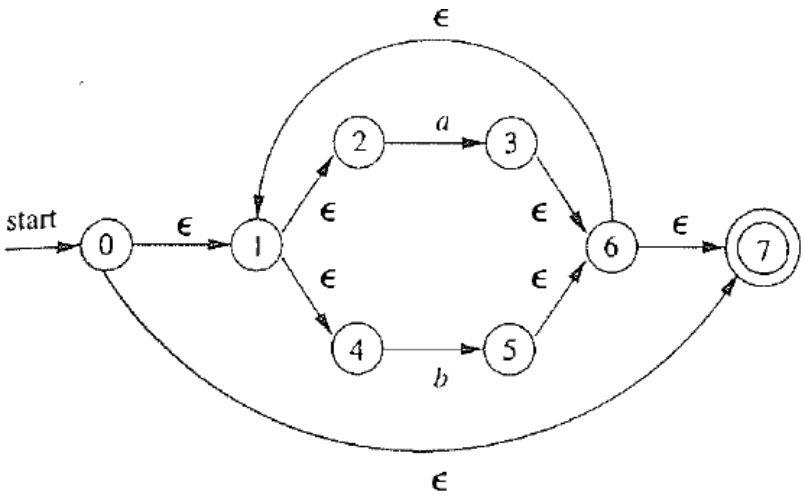

start
0
ϵ
1
2
a
3
ϵ
ϵ
ϵ
ϵ
4
5
6
ϵ
7
b
ϵ

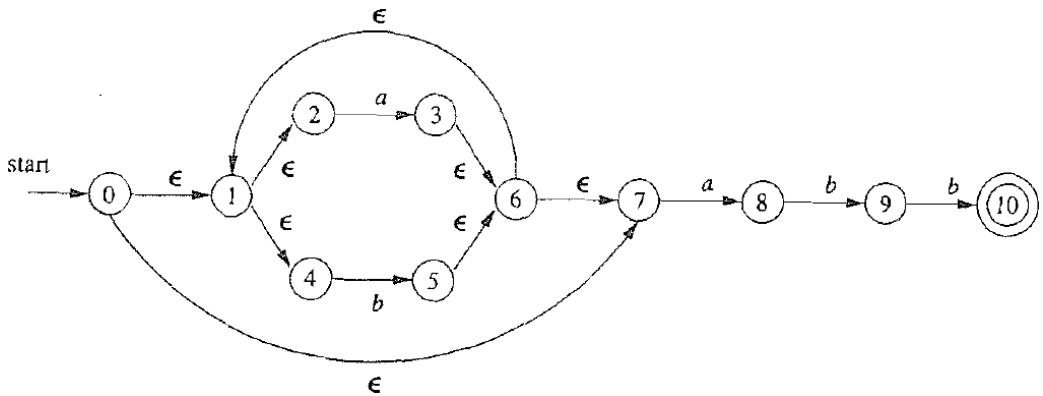
$$r = (a|b)^*abb$$

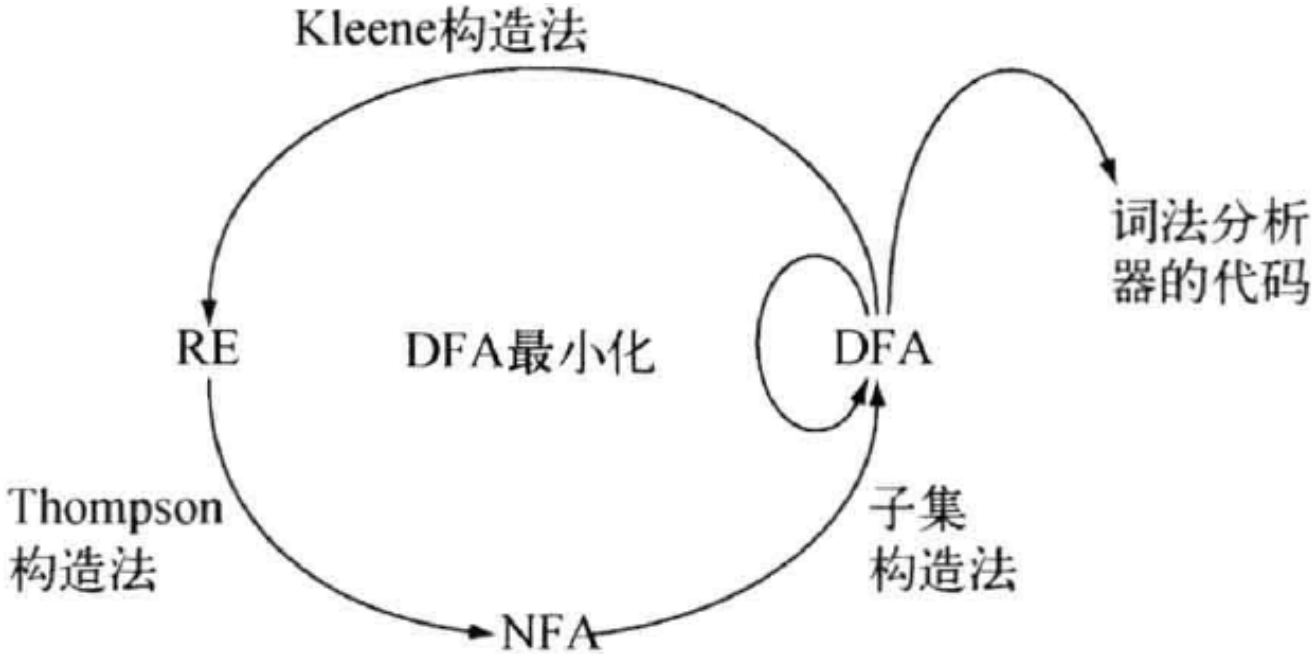

start
2
a
3


start
4
b
5


start
1
2
a
3
ϵ
ϵ
ϵ
ϵ
6
4
5
b


start
0
ϵ
1
2
a
3
ϵ
ϵ
ϵ
ϵ
6
ϵ
7
4
5
b
ϵ
ϵ


start
0
ϵ
1
2
a
3
ϵ
ϵ
ϵ
ϵ
6
ϵ
7
a
8
b
9
b
10
4
5
b
ϵ
ϵ



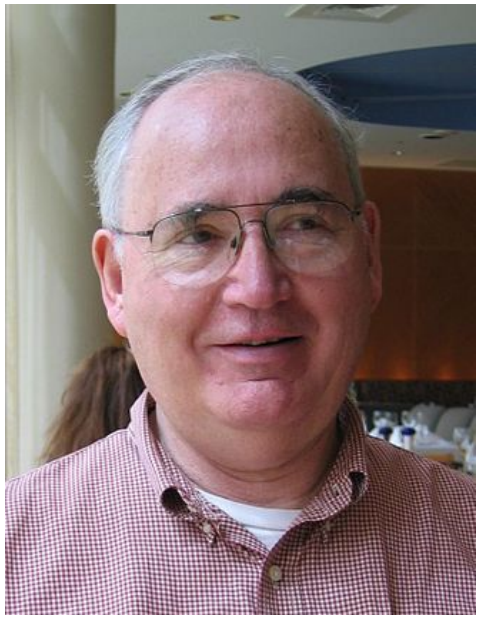
$$\text{NFA} \Longrightarrow \text{DFA}$$

$$\boxed{N \Longrightarrow D}$$

要求：$L(D) = L(N)$

从 NFA 到 DFA 的转换: **子集构造法** (Subset/Powerset Construction)

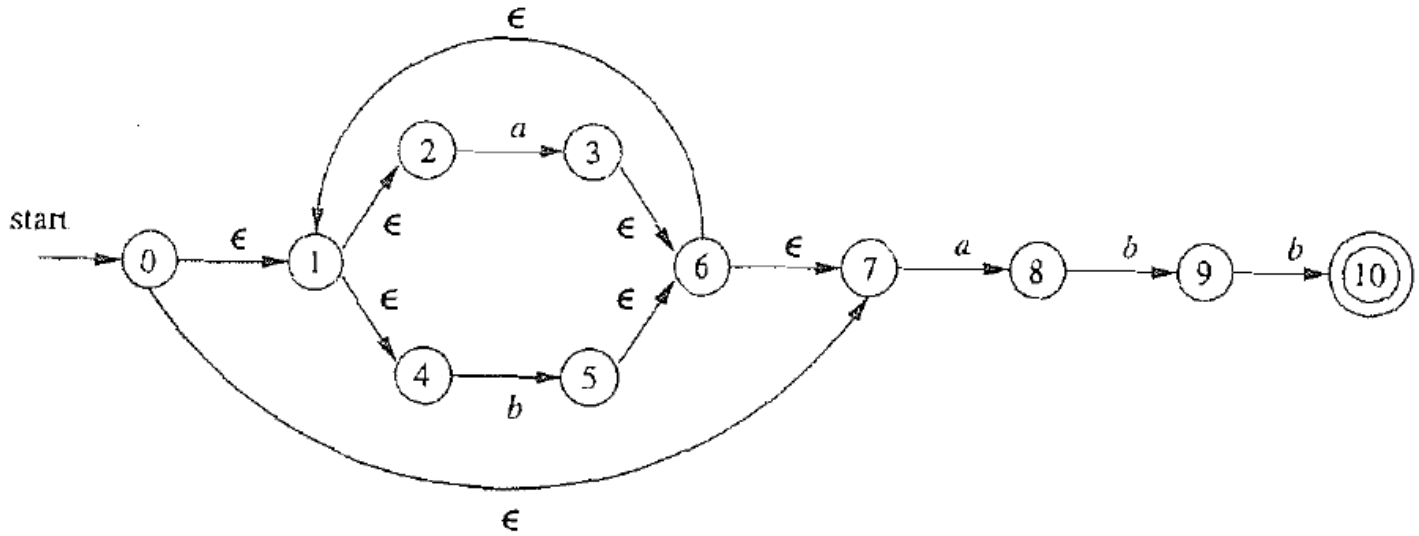
Michael O. Rabin (1931 ∼)



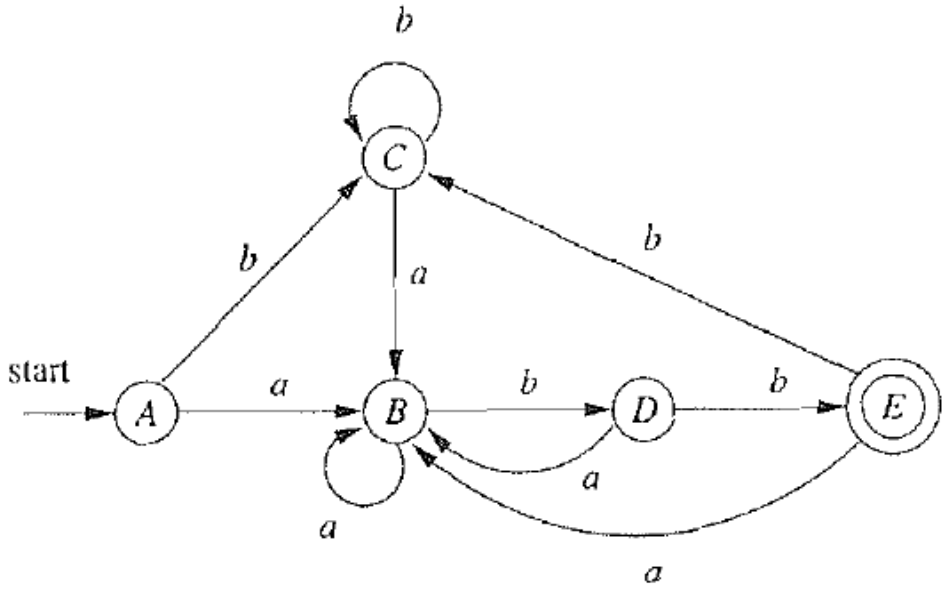
Dana Scott (1932 ∼)

思想: 用 DFA 模拟 NFA

用 DFA 模拟 NFA

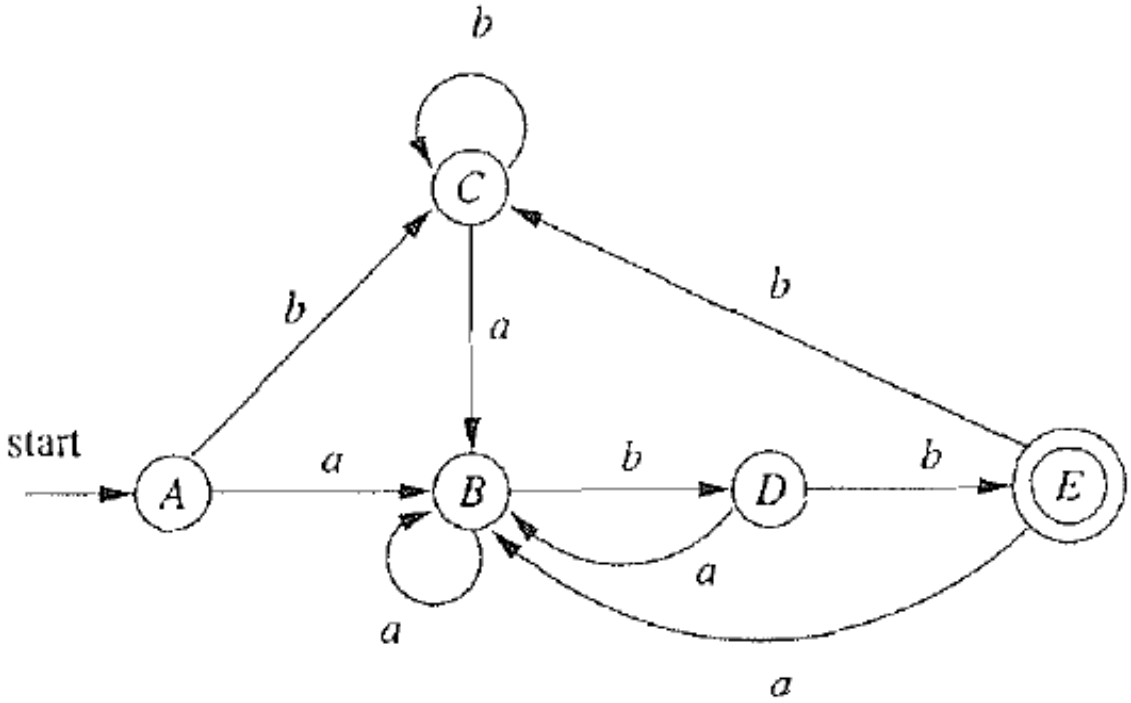


$$L(\mathcal{A}) = L((a|b)^*abb)$$



板书演示算法过程

NFA 状态	DFA 状态	a	b
$\{0,1,2,4,7\}$	A	B	C
$\{1,2,3,4,6,7,8\}$	B	B	D
$\{1,2,4,5,6,7\}$	C	B	C
$\{1,2,4,5,6,7,9\}$	D	B	E
$\{1,2,4,5,6,7,10\}$	E	B	C



从状态 s 开始, 只通过 ϵ-转移可达的状态集合

$$\epsilon\text{-closure}(s) = \{t \in S_N | s \xrightarrow{\epsilon^*} t\}$$

从状态 s 开始, 只通过 ϵ-转移可达的状态集合

$$\epsilon\text{-closure}(s) = \{t \in S_N | s \xrightarrow{\epsilon^*} t\}$$

$$\epsilon\text{-closure}(T) = \bigcup_{s \in T} \epsilon\text{-closure}(s)$$

从状态 s 开始, 只通过 ϵ-转移可达的状态集合

$$\epsilon\text{-closure}(s) = \{t \in S_N | s \xrightarrow{\epsilon^*} t\}$$

$$\epsilon\text{-closure}(T) = \bigcup_{s \in T} \epsilon\text{-closure}(s)$$

$$\text{move}(T, a) = \bigcup_{s \in T} \delta(s, a)$$

子集构造法 ($N \implies D$) 的**原理**:

$$N : (\Sigma_N, S_N, n_0, \delta_N, F_N)$$

$$D : (\Sigma_D, S_D, d_0, \delta_D, F_D)$$

子集构造法 $(N \implies D)$ 的**原理**:

$$N : (\Sigma_N, S_N, n_0, \delta_N, F_N)$$

$$D : (\Sigma_D, S_D, d_0, \delta_D, F_D)$$

$$\Sigma_D = \Sigma_N$$

子集构造法 ($N \implies D$) 的**原理**:

$$N : (\Sigma_N, S_N, n_0, \delta_N, F_N)$$

$$D : (\Sigma_D, S_D, d_0, \delta_D, F_D)$$

$$\Sigma_D = \Sigma_N$$

$$S_D \subseteq 2^{S_N} \qquad (\forall s_D \in S_D : s_D \subseteq S_N)$$

子集构造法 ($N \implies D$) 的**原理**:

$$N : (\Sigma_N, S_N, n_0, \delta_N, F_N)$$

$$D : (\Sigma_D, S_D, d_0, \delta_D, F_D)$$

$$\Sigma_D = \Sigma_N$$

$$S_D \subseteq 2^{S_N} \qquad (\forall s_D \in S_D : s_D \subseteq S_N)$$

初始状态 $d_0 = \epsilon\text{-closure}(n_0)$

子集构造法 $(N \implies D)$ 的**原理**:

$$N : (\Sigma_N, S_N, n_0, \delta_N, F_N)$$

$$D : (\Sigma_D, S_D, d_0, \delta_D, F_D)$$

$$\Sigma_D = \Sigma_N$$

$$S_D \subseteq 2^{S_N} \qquad (\forall s_D \in S_D : s_D \subseteq S_N)$$

初始状态 $d_0 = \epsilon\text{-closure}(n_0)$

转移函数 $\forall a \in \Sigma_D : \delta_D(s_D, a) = \epsilon\text{-closure}(\text{move}(s_D, a))$

子集构造法 $(N \implies D)$ 的**原理**:

$$N : (\Sigma_N, S_N, n_0, \delta_N, F_N)$$

$$D : (\Sigma_D, S_D, d_0, \delta_D, F_D)$$

$$\Sigma_D = \Sigma_N$$

$$S_D \subseteq 2^{S_N} \qquad (\forall s_D \in S_D : s_D \subseteq S_N)$$

初始状态 $d_0 = \epsilon\text{-closure}(n_0)$

转移函数 $\forall a \in \Sigma_D : \delta_D(s_D, a) = \epsilon\text{-closure}(\text{move}(s_D, a))$

接受状态集 $F_{\mathcal{D}} = \{s_D \in S_{\mathcal{D}} \mid \exists f \in F_N.\ f \in s_D\}$

子集构造法的**复杂度分析**:
$(|S_N| = n)$

$$|S_D| = \Theta(2^n) = O(2^n) \cap \Omega(2^n)$$

对于任何算法, 最坏情况下, $|S_D| = \Omega(2^n)$

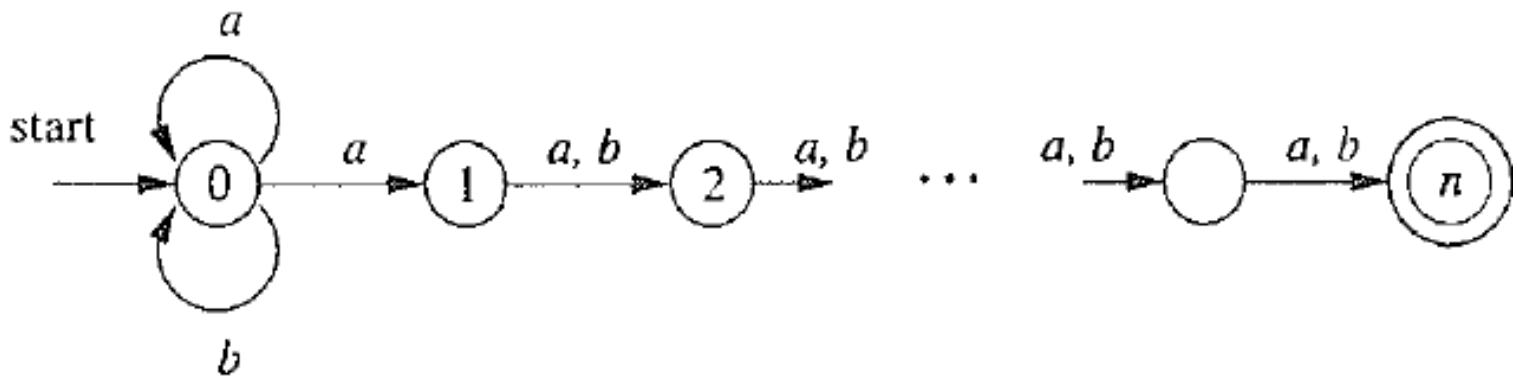
“长度为 $m \geq n$ 个字符的 a, b 串, 且倒数第 n 个字符是 a”

“长度为 $m \geq n$ 个字符的 a, b 串, 且倒数第 n 个字符是 a”

$$L_n = (a|b)^* a (a|b)^{n-1}$$

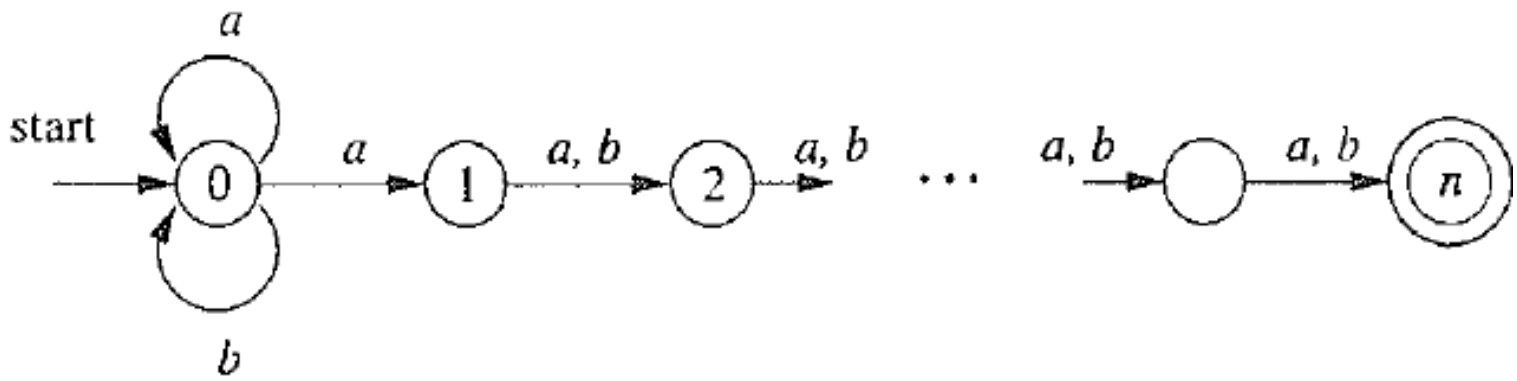
“长度为 $m \geq n$ 个字符的 a, b 串, 且倒数第 n 个字符是 a”

$$L_n = (a|b)^* a (a|b)^{n-1}$$

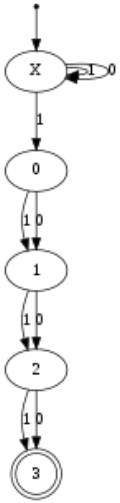


“长度为 $m \geq n$ 个字符的 a, b 串, 且倒数第 n 个字符是 a”

$$L_n = (a|b)^* a (a|b)^{n-1}$$



$$n = 4$$


X
0
1
2
3


{1, X}
{2, X}
{0, 3, X}
{0, X}
{X}
{3, X}
{1, 3, X}
{0, 2, X}
{2, 3, X}
{0, 1, 2, X}
{0, 1, 2, 3, X}
{1, 2, 3, X}
{0, 2, 3, X}
{0, 1, 3, X}
{1, 2, X}
{0, 1, X}

闭包 (Closure): f-closure(T)

闭包 (Closure): f-closure(T)

ϵ-closure(T)

闭包 (Closure): f-closure(T)

$$\epsilon\text{-closure}(T)$$

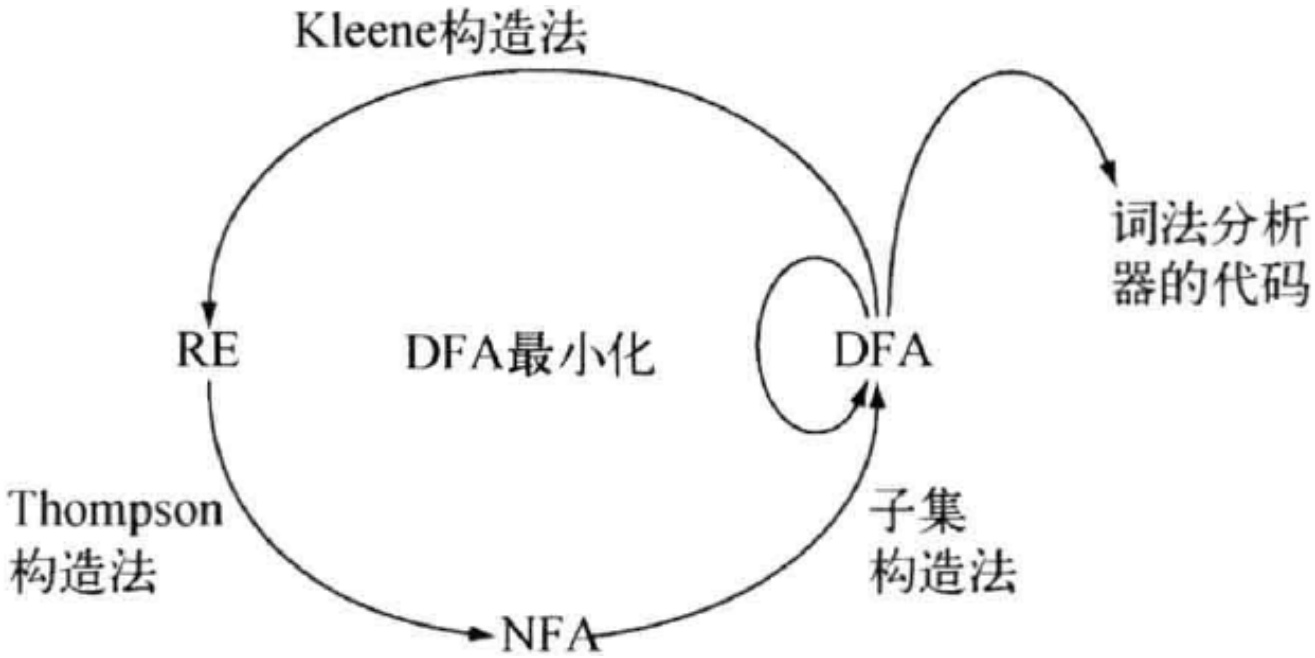
$$T \implies f(T) \implies f(f(T)) \implies f(f(f(T))) \implies \dots$$

闭包 (Closure): f-closure(T)

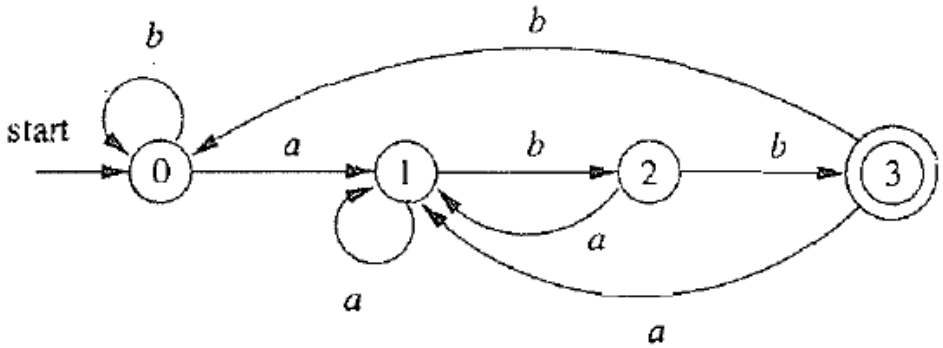
$$\epsilon\text{-closure}(T)$$

$$T \Longrightarrow f(T) \Longrightarrow f(f(T)) \Longrightarrow f(f(f(T))) \Longrightarrow \ldots$$

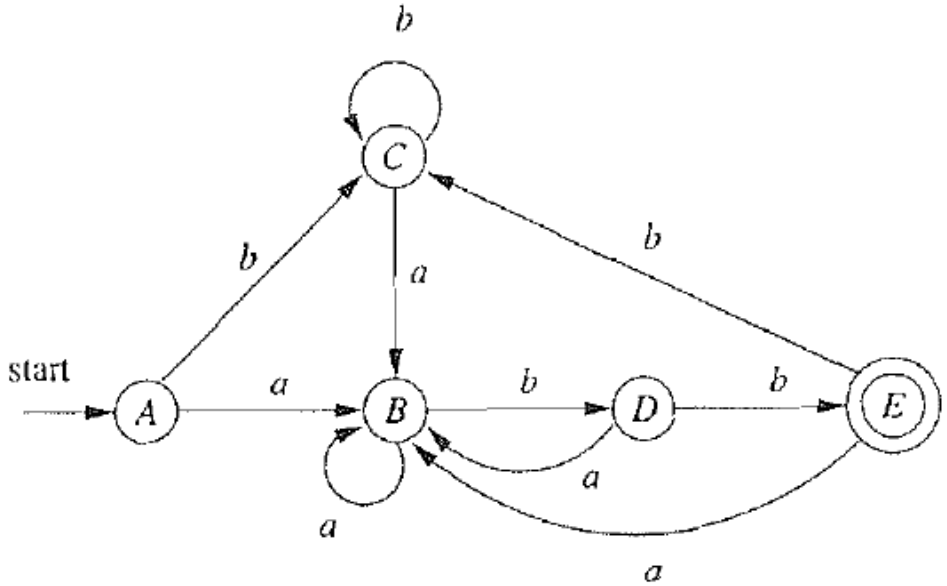
直到找到 x 使得 $f(x) = x$ (x 称为 f 的**不动点**)



DFA 最小化



$$L(\mathcal{A}) = L((a|b)^*abb)$$



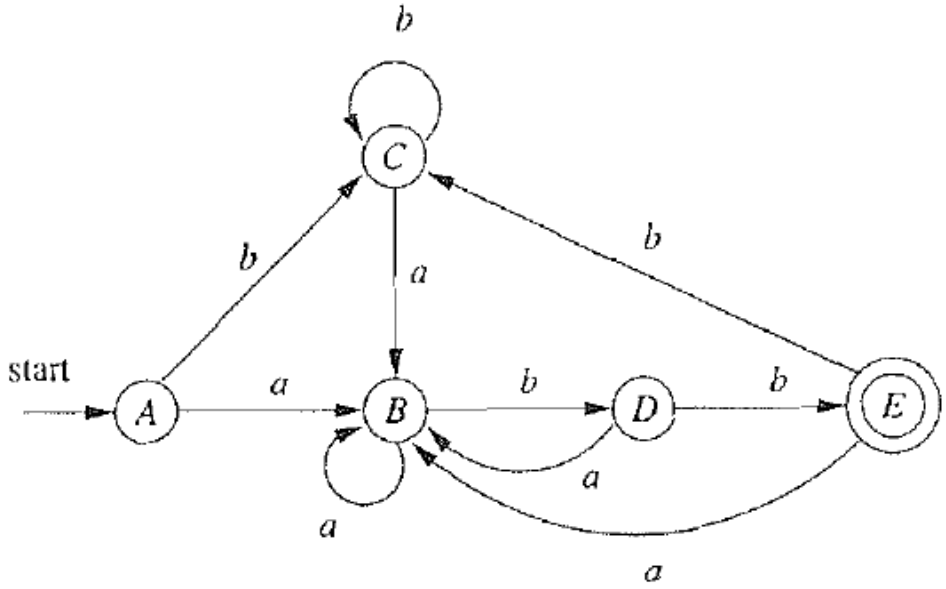
子集构造法得到的 DFA

DFA 最小化算法基本思想: 等价的状态可以合并

John Hopcroft (1939 ∼)

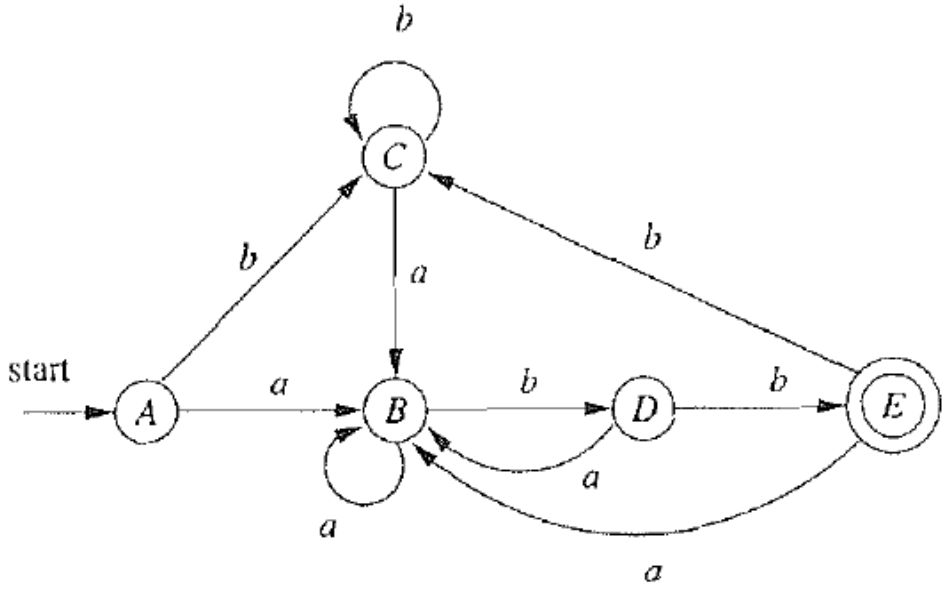
*"With Robert E. Tarjan, for fundamental achievements in the design and analysis of **algorithms and data structures**"*
— Turing Award, 1986

如何定义**等价**状态?



$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' = t').$$

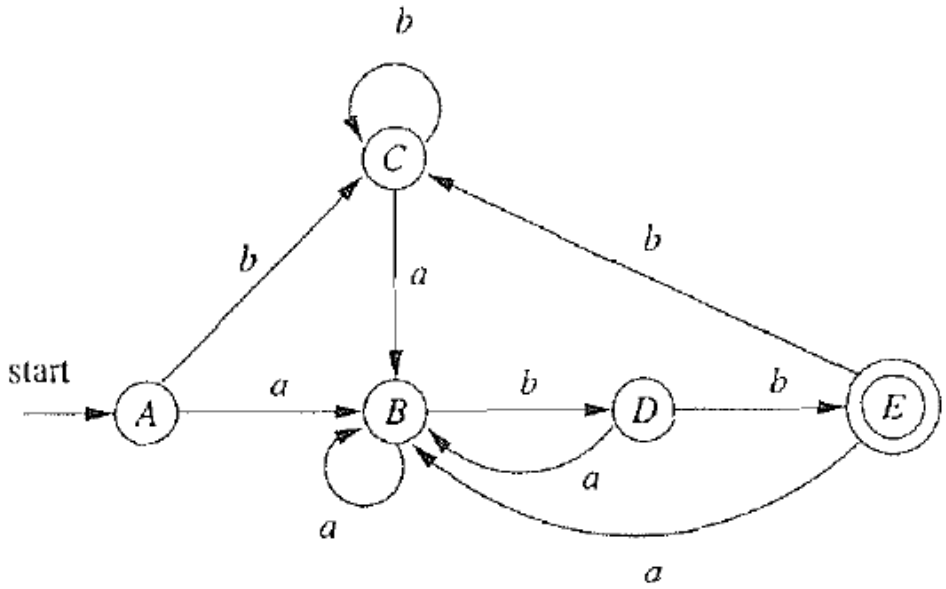
如何定义**等价**状态?



$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' = t').$$

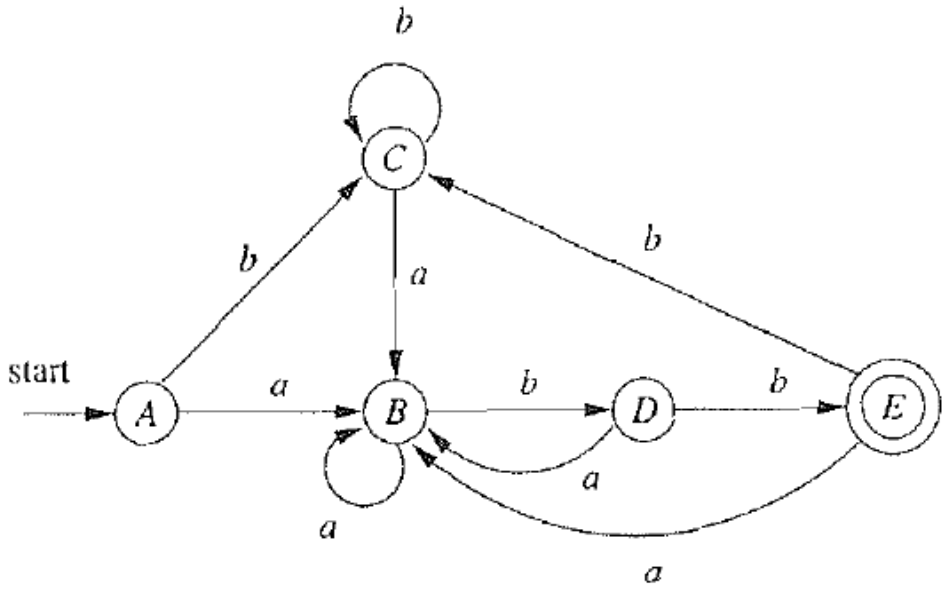
但是, 这个定义是错误的

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' = t').$$



$$A \sim C \sim E$$

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' = t').$$

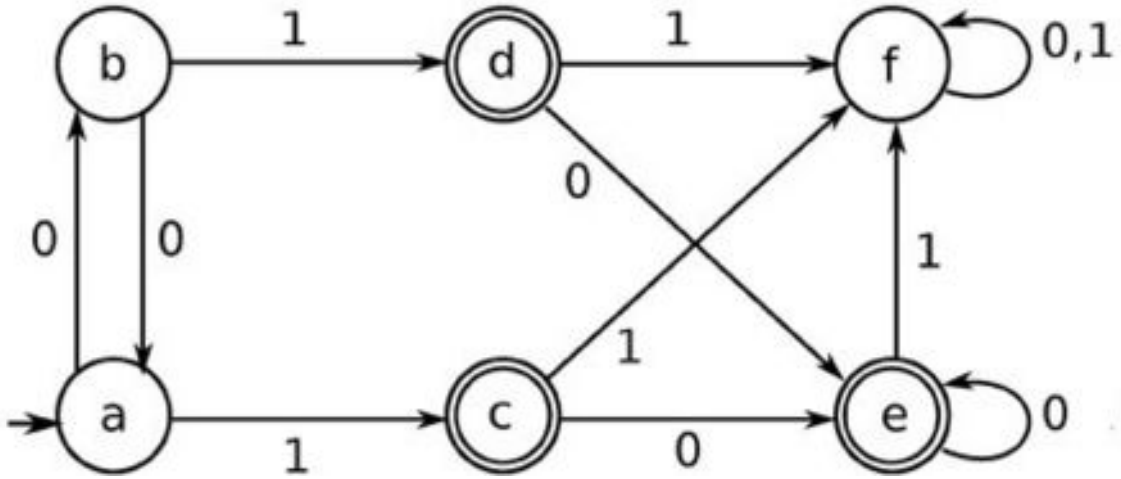


$$A \sim C \sim E$$

但是, 接受状态与非接受状态必定不等价

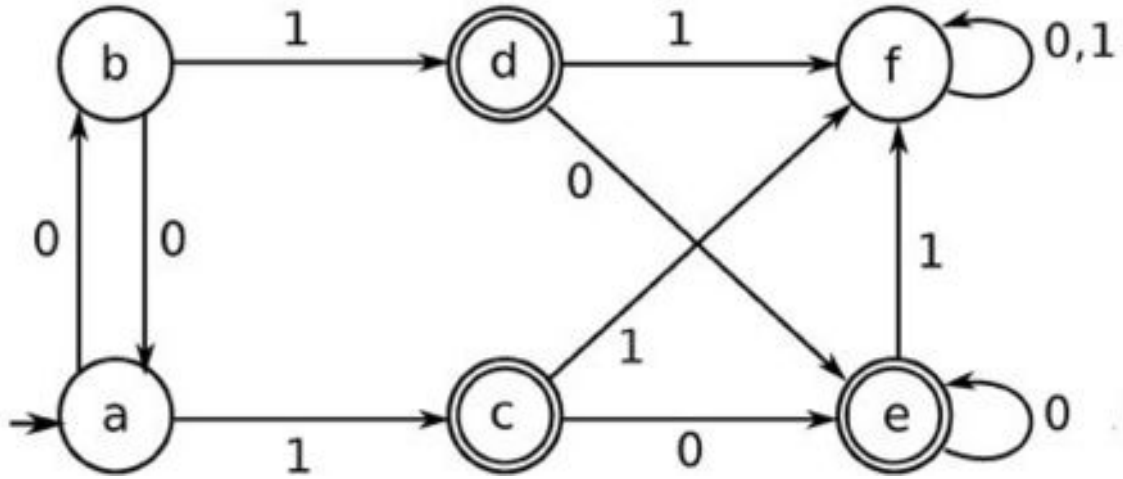
空串 ϵ **区分**了这两类状态

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' = t').$$

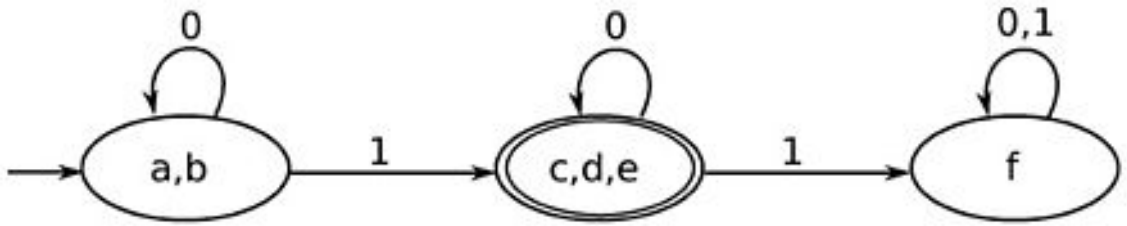


$$c \sim d \sim e \qquad a \nsim b$$

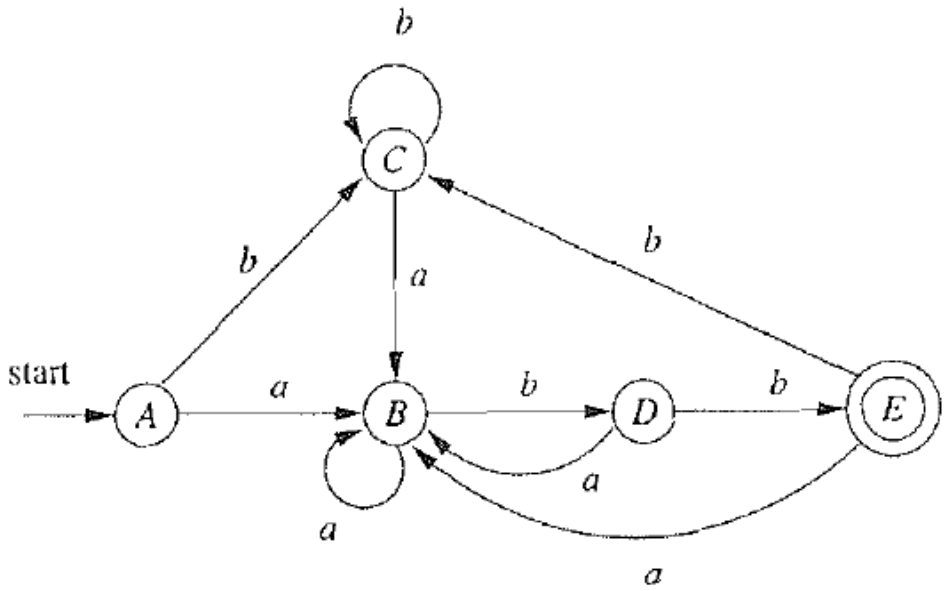
$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' = t').$$



$$c \sim d \sim e \qquad a \nsim b$$

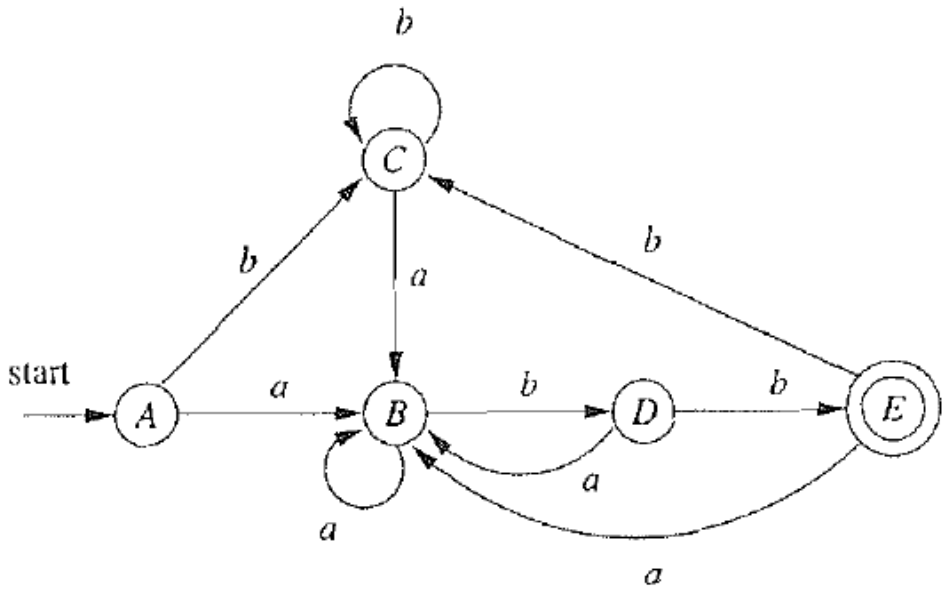


如何定义等价状态?



$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

如何定义等价状态?



$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

基于该定义, 不断**合并**等价的状态, 直到无法合并为止

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

基于该定义, 不断**合并**等价的状态, 直到无法合并为止

但是, 这是一个递归定义, 从哪里开始呢?

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

基于该定义, 不断**合并**等价的状态, 直到无法合并为止

但是, 这是一个递归定义, 从哪里开始呢?

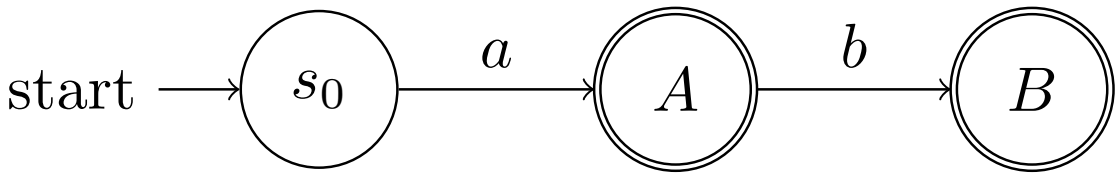
Q : 所有接受状态都是等价的吗?

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

基于该定义, 不断**合并**等价的状态, 直到无法合并为止

但是, 这是一个递归定义, 从哪里开始呢?

Q : 所有接受状态都是等价的吗?



$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

缺少基础情况, 不知从何下手

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

缺少基础情况, 不知从何下手

$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

$$s \sim t \iff \forall a \in \Sigma.\ \Big((s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t')\Big) \implies (s' \sim t').$$

缺少基础情况, 不知从何下手

$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

划分, 而非合并

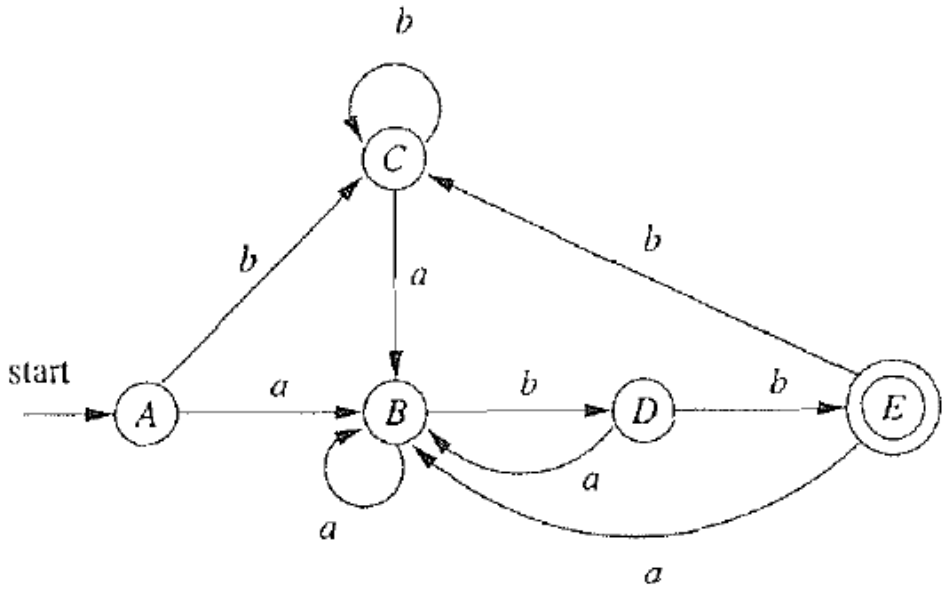
$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

从哪里开始呢?

$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

从哪里开始呢?

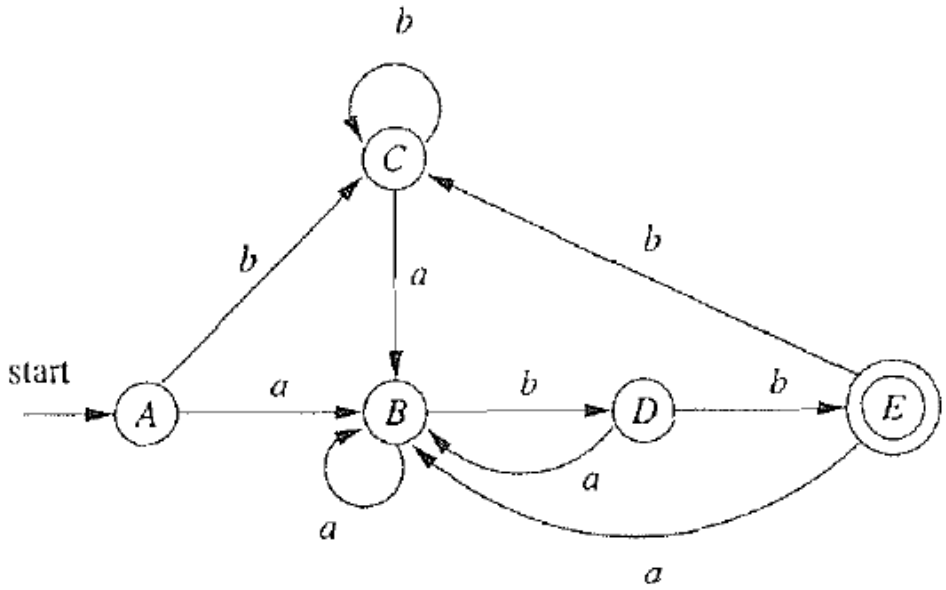


$$\Pi = \{F, S \setminus F\}$$

接受状态与非接受状态必定不等价

$$s \nsim t \iff \exists a \in \Sigma.\ (s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$$

从哪里开始呢?



$$\Pi = \{F, S \setminus F\}$$

接受状态与非接受状态必定不等价

空串 ϵ **区分**了这两类状态

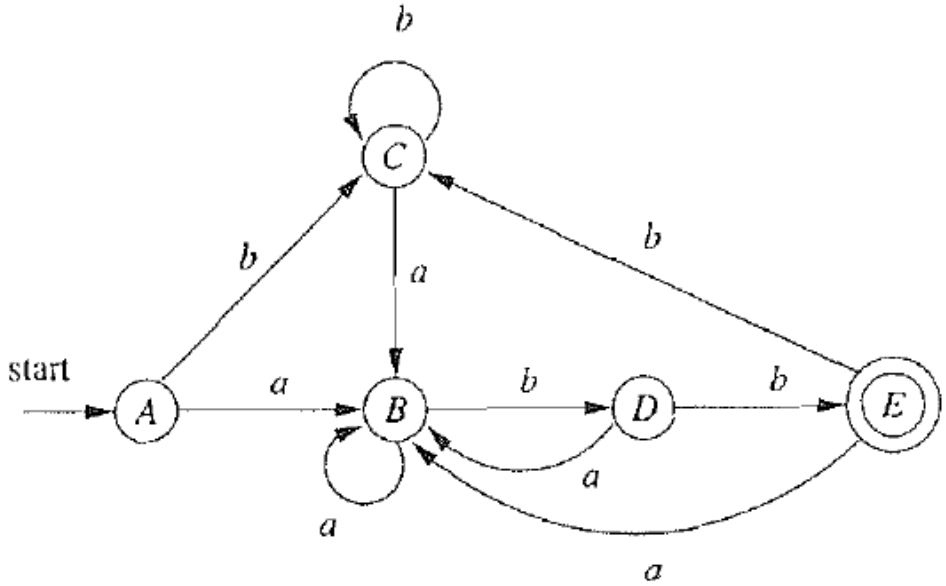
板书演示算法过程

$$\Pi_0 = \{\{A, B, C, D\}, \{E\}\}$$

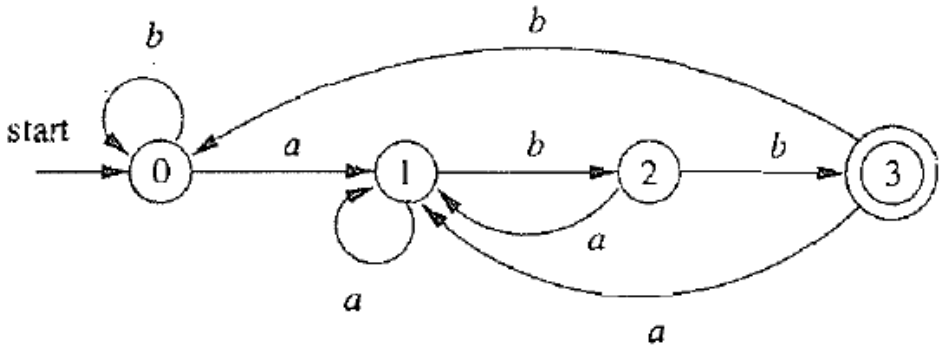
$$\Pi_1 = \{\{A, B, C\}, \{D\}, \{E\}\}$$

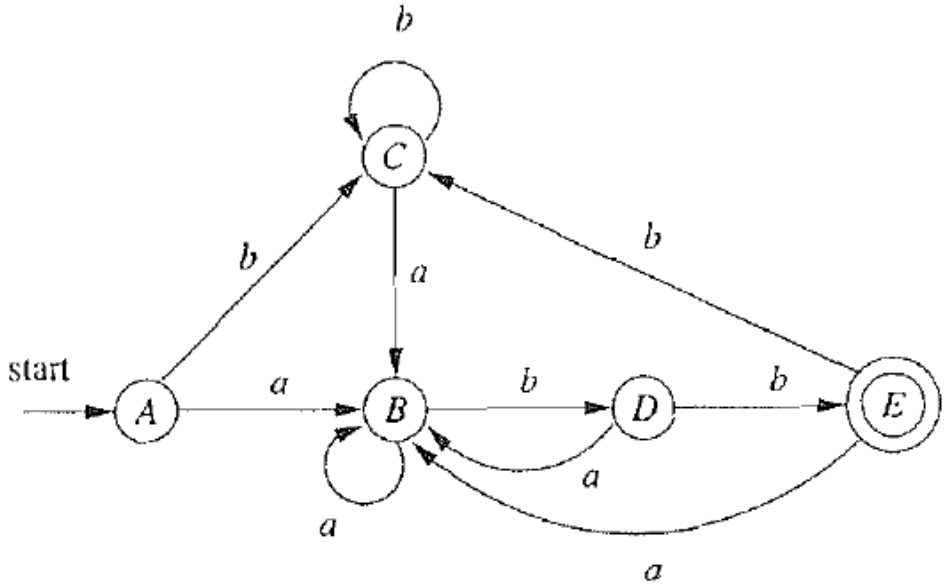
$$\Pi_2 = \{\{A, C\}, \{B\}, \{D\}, \{E\}\}$$

$$\Pi_3 = \Pi_2 = \Pi_{\text{final}}$$

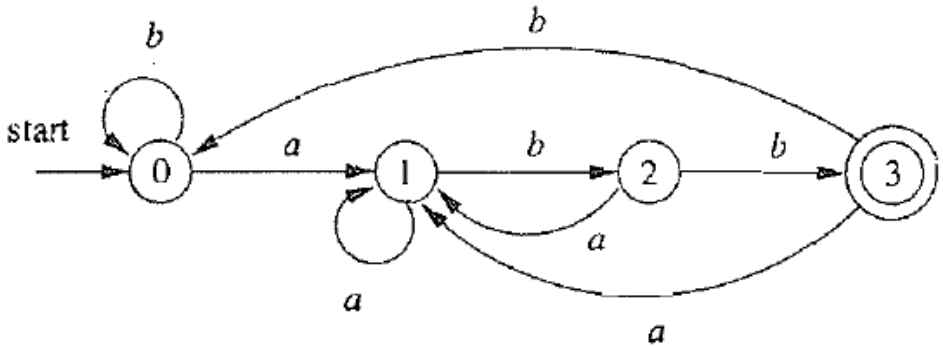


合并等价状态 $A \sim C$





合并等价状态 $A \sim C$



Q：合并后是否一定还是 DFA? 初始状态、接受状态是哪些?

DFA 最小化**等价状态划分**方法

$$\Pi = \{F, S \setminus F\}$$

最初，令 $\Pi_{\text{new}} = \Pi$;
for (Π中的每个组G) {
　　将G分划为更小的组，使得两个状态s和t在同一小组中当且仅当对于所有
　　　　的输入符号a，状态s和t在a上的转换都到达Π中的同一组；
　　/* 在最坏情况下，每个状态各自组成一个组*/
　　在Π_{new}中将G替换为对G进行分划得到的那些小组；
}

直到再也无法**划分**为止　(不动点!)

然后，将同一等价类里的状态**合并**

这是 DFA 最小化算法

这是 **DFA** 最小化算法

所以，要注意处理“死状态”

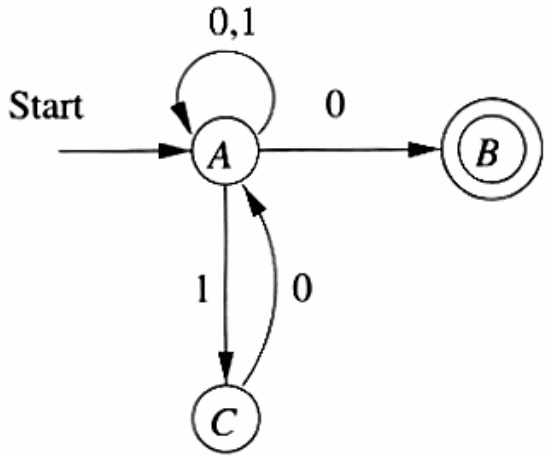


Figure 4.13: An NFA that cannot be minimized by state equivalence

不适用于 NFA 最小化; NFA 最小化问题是 PSPACE-complete 的

如何分析 DFA 最小化算法的**复杂度**?

如何分析 DFA 最小化算法的**复杂度**?

为什么 DFA 最小化算法是**正确**的?

如何分析 DFA 最小化算法的**复杂度**?

为什么 DFA 最小化算法是**正确**的?

最小化 DFA 是**唯一**的吗?

如何分析 DFA 最小化算法的**复杂度**?

为什么 DFA 最小化算法是**正确**的?

最小化 DFA 是**唯一**的吗?

DFA Minimization @ wiki

Definition (可区分的 (Distinguishable); 等价的 (Equivalent))

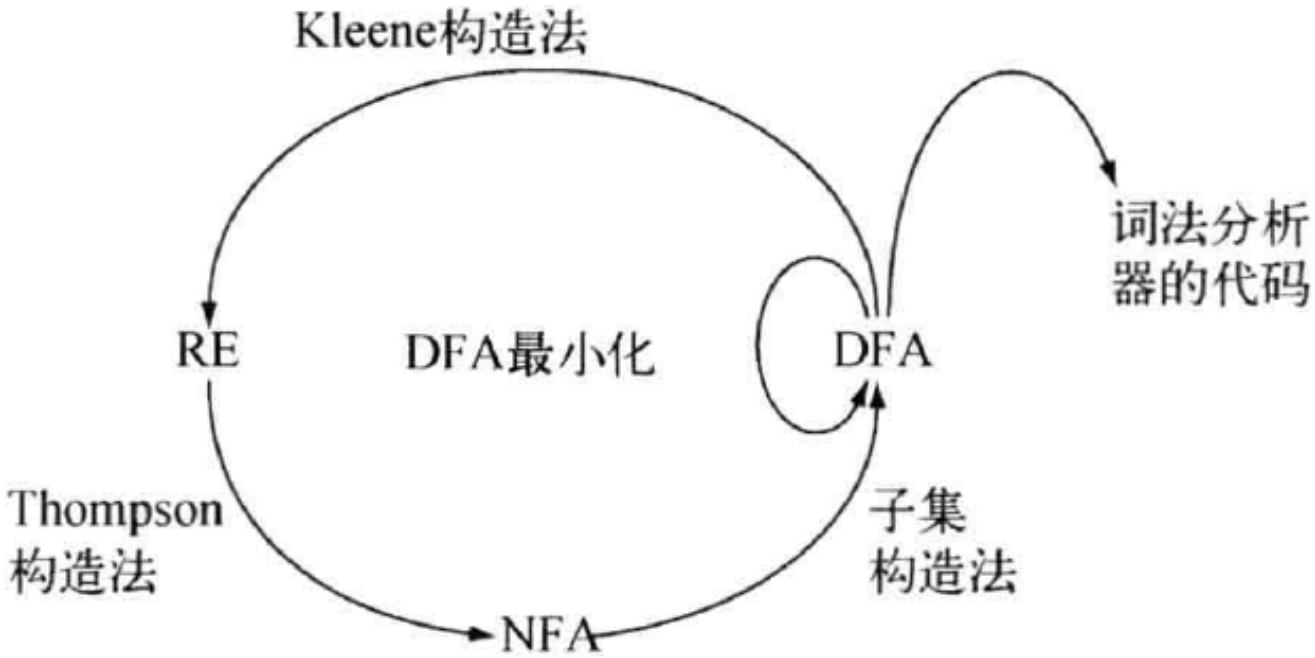
如果存在某个能区分状态 s 与 t 的字符串, 则称 s 与 t 是**可区分的**; 否则, 称 s 与 t 是**等价的**。

Definition (可区分的 (Distinguishable); 等价的 (Equivalent))

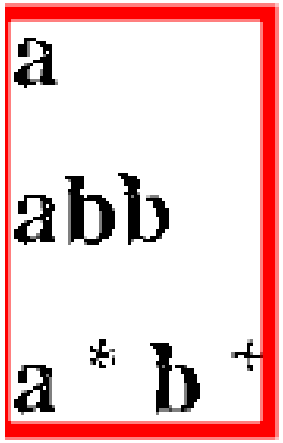
如果存在某个能区分状态 s 与 t 的字符串, 则称 s 与 t 是**可区分的**; 否则, 称 s 与 t 是**等价的**。

Definition (字符串 x 区分状态 s 与 t)

如果分别从 s 与 t 出发, 沿着标号为 x 的路径到达的两个状态中只有一个是接受状态, 则称 x **区分**了状态 s 与 t。



DFA $\Longrightarrow$ 词法分析器



最前优先匹配: abb (比如 **关键字**)

最长优先匹配: $aabbb$

根据正则表达式构造相应的 NFA

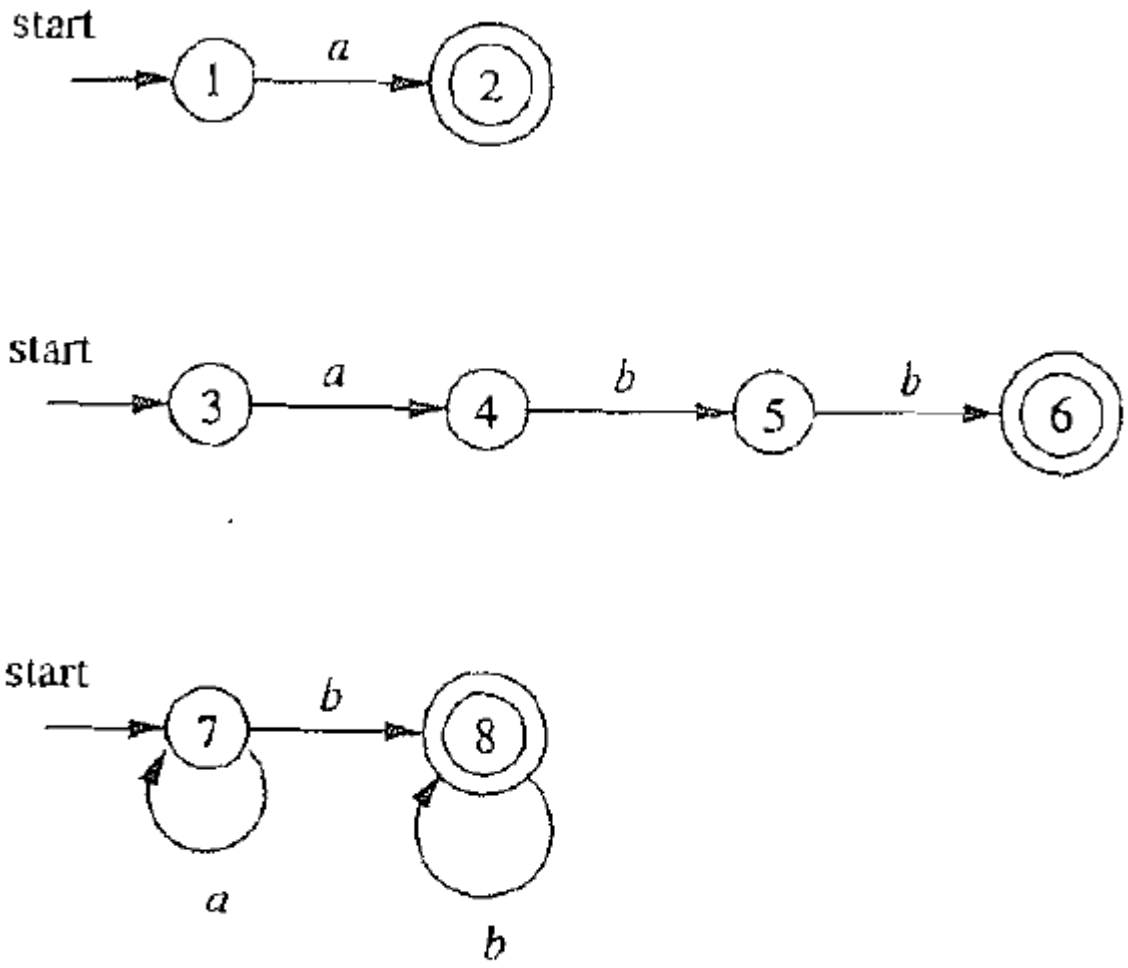
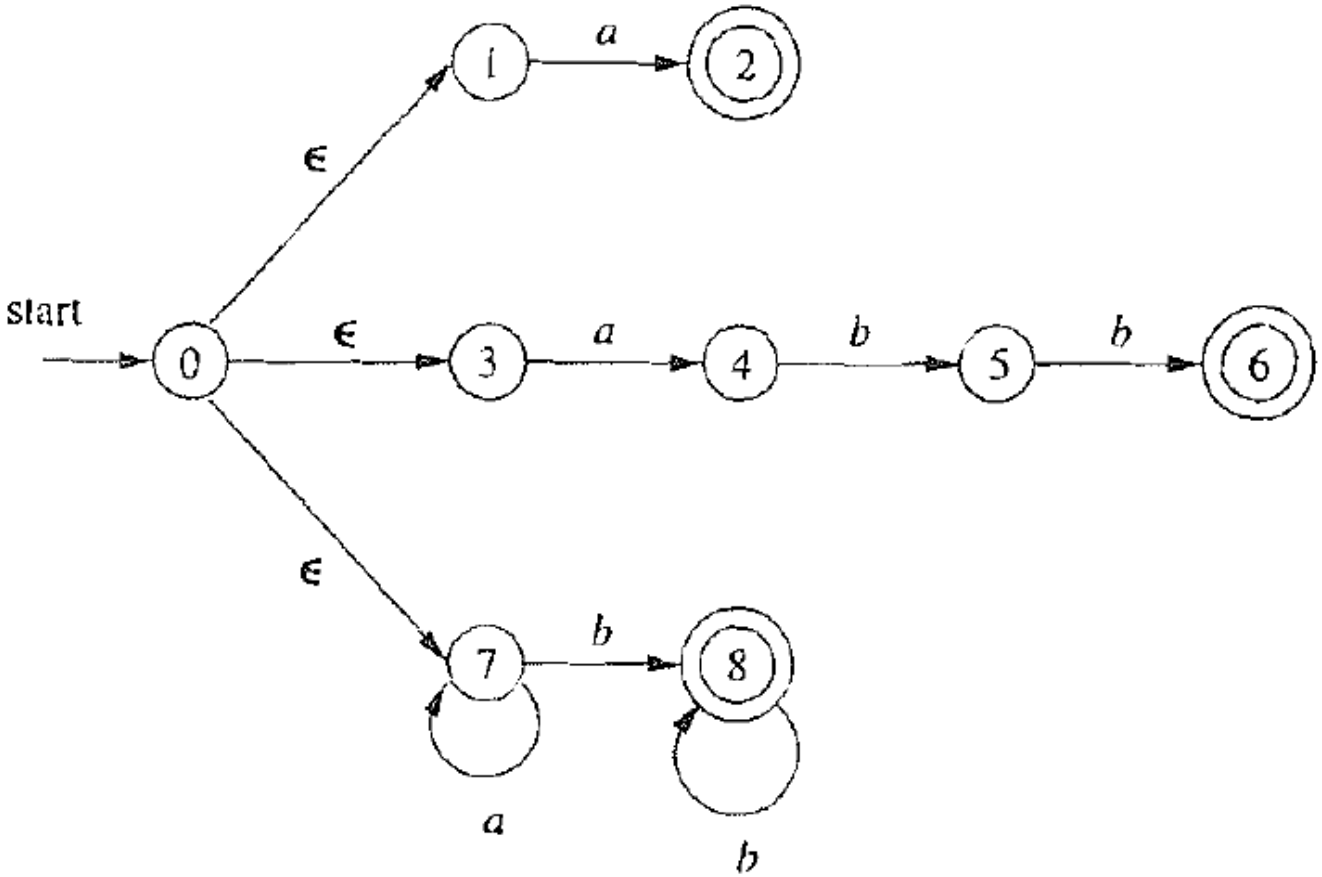
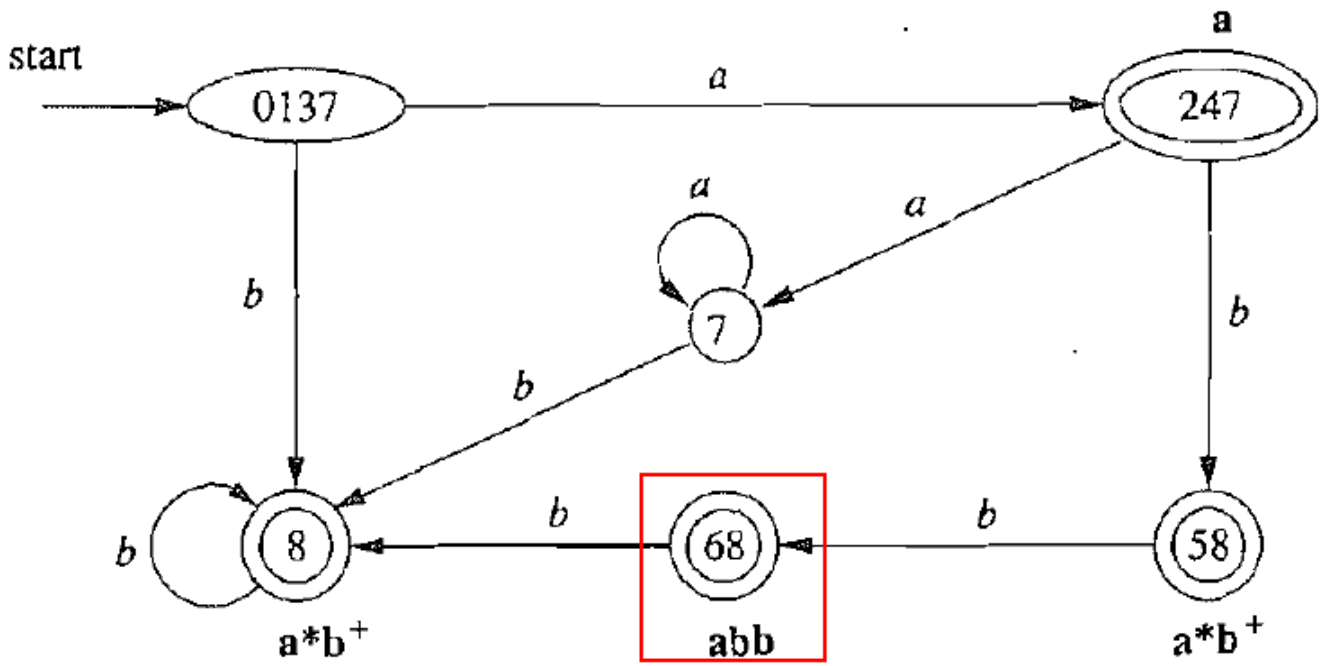


图 3-51　a、abb 和 a^*b^+ 的 NFA

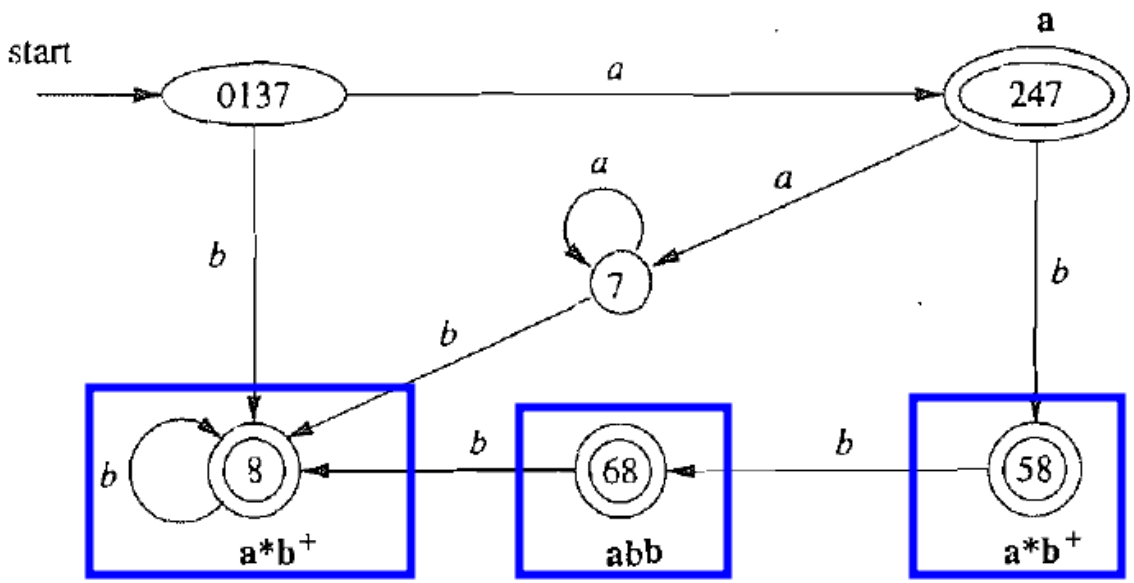
合并这三个 NFA, 构造 $a|abb|a^*b^+$ 对应的 NFA



使用**子集构造法**将 NFA 转化为等价的 DFA (并消除 “死状态”)



注意: 要保留各个 NFA 的**接受状态**信息, 并采用**最前优先匹配**原则

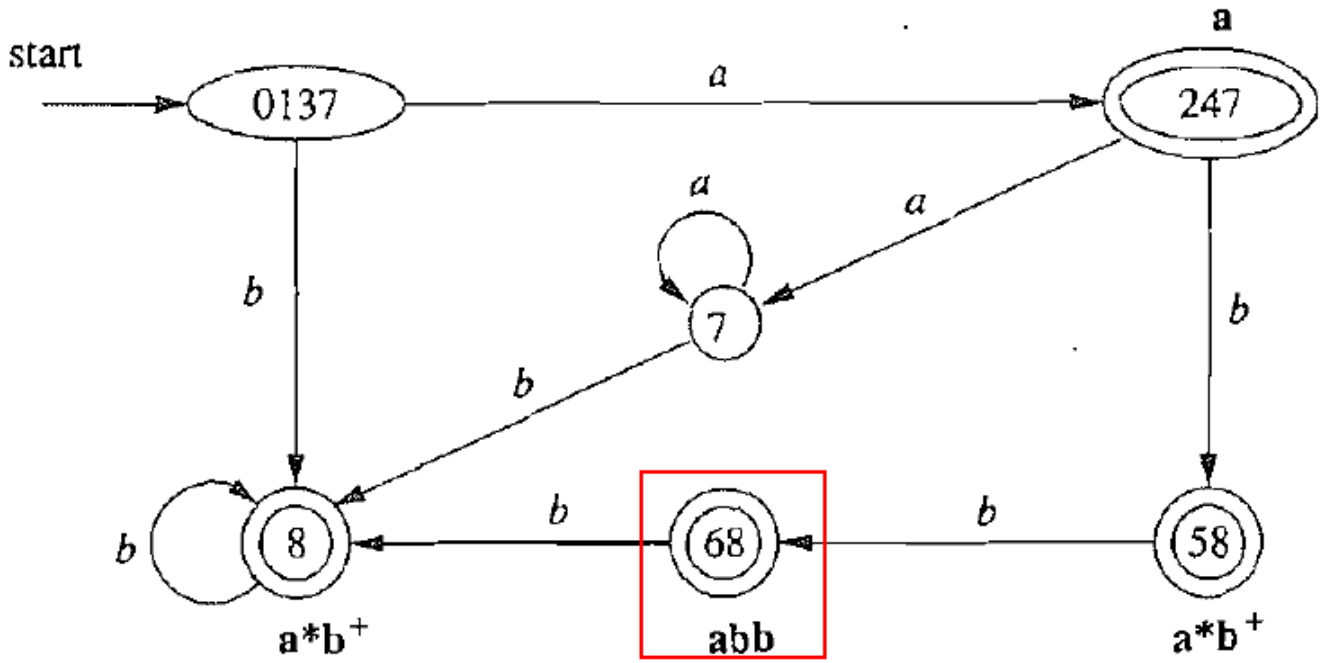


$$\forall a \in \Sigma.\ \delta(\emptyset, a) = \emptyset$$

需要**消除**“死状态”，避免词法分析器徒劳消耗输入流

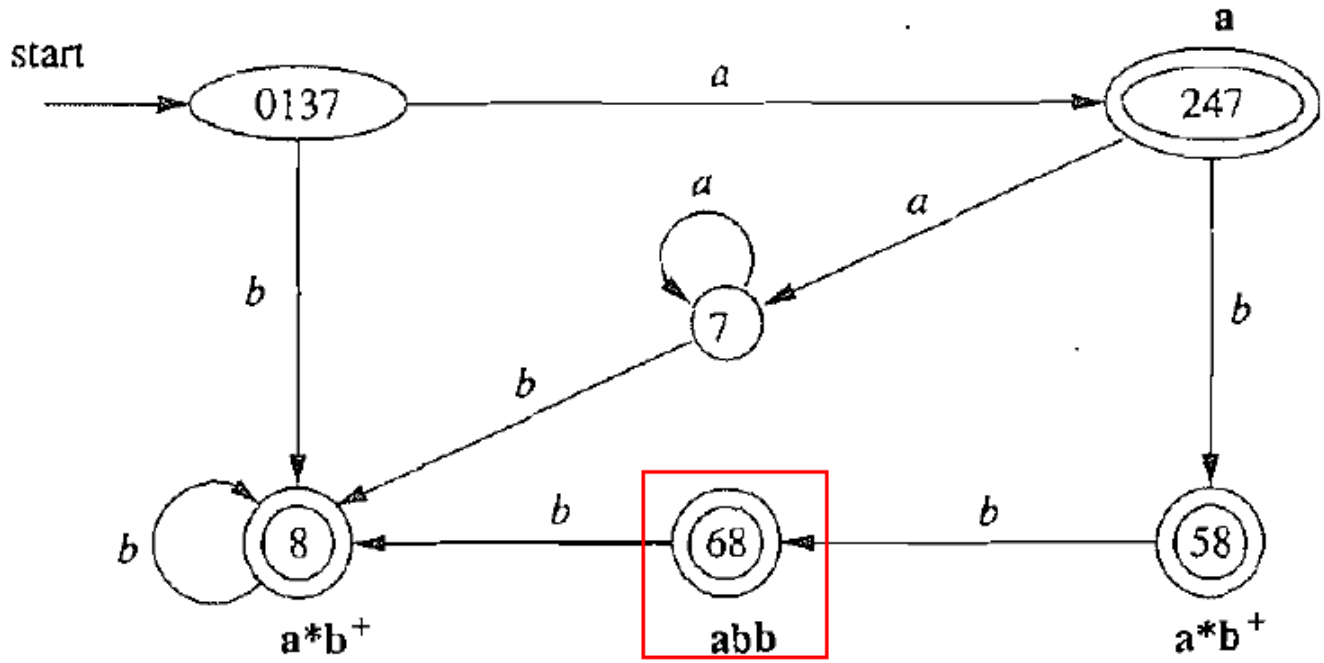
模拟运行该 DFA, 直到无法继续为止 (输入结束或状态无转移);

假设此时状态为 s



模拟运行该 DFA, 直到无法继续为止 (输入结束或状态无转移);

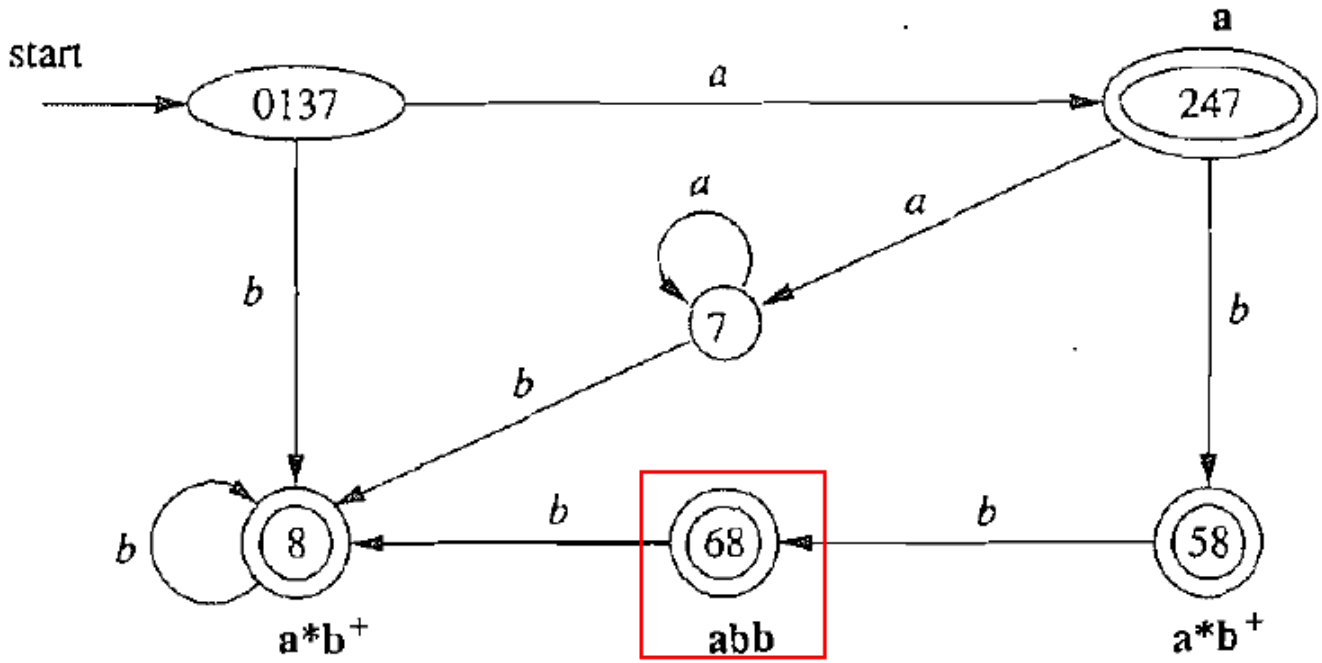
假设此时状态为 s



若 s 为**接受状态**, 则识别成功;

模拟运行该 DFA, 直到无法继续为止 (输入结束或状态无转移);

假设此时状态为 s

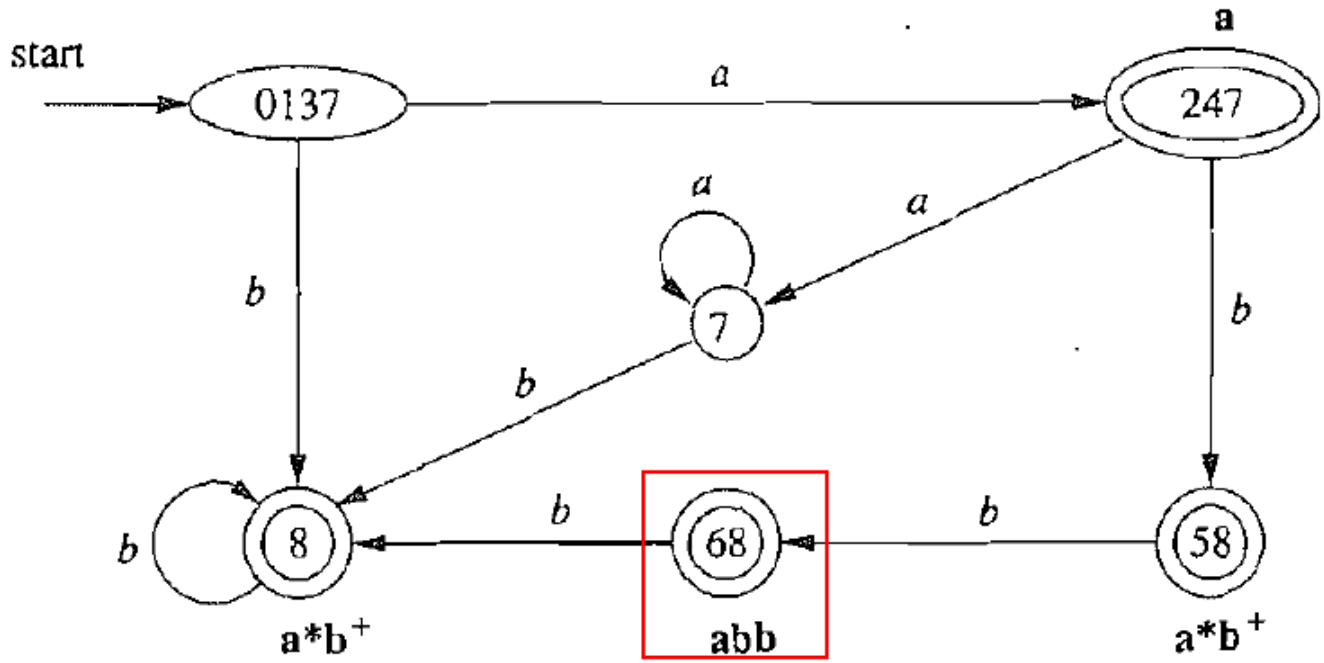


若 s 为**接受状态**, 则识别成功;

否则, **回溯** (包括状态与输入流) 至最近一次经过的接受状态, 识别成功;

模拟运行该 DFA, 直到无法继续为止 (输入结束或状态无转移);

假设此时状态为 s



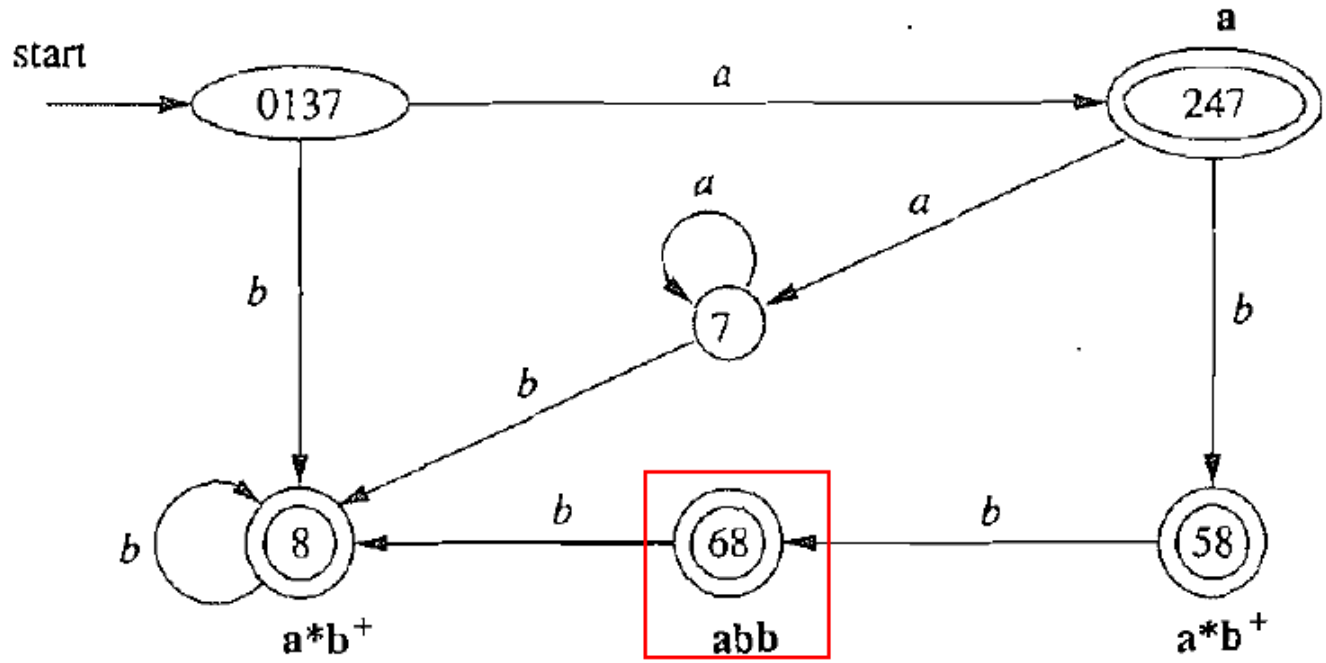
若 s 为**接受状态**, 则识别成功;

否则, **回溯** (包括状态与输入流) 至最近一次经过的接受状态, 识别成功;

若没有经过任何接受状态, 则**报错** (忽略第一个字符)

模拟运行该 DFA, 直到无法继续为止 (输入结束或状态无转移);

假设此时状态为 s

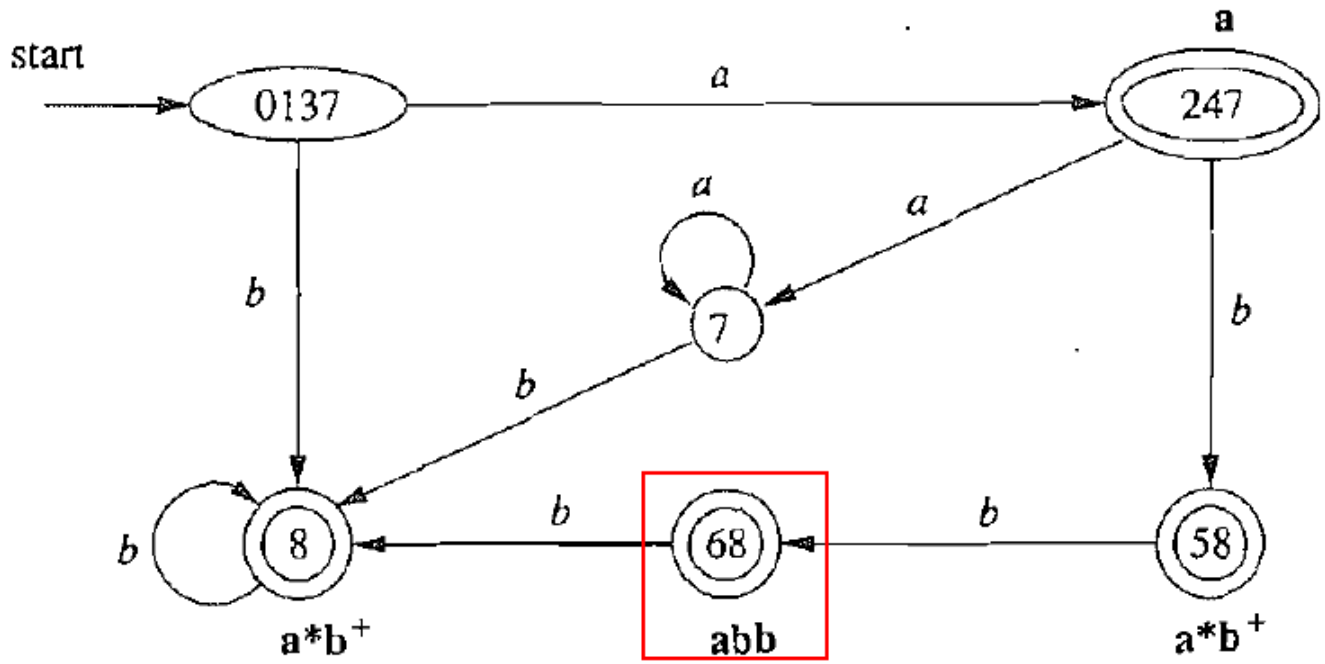


若 s 为**接受状态**, 则识别成功;

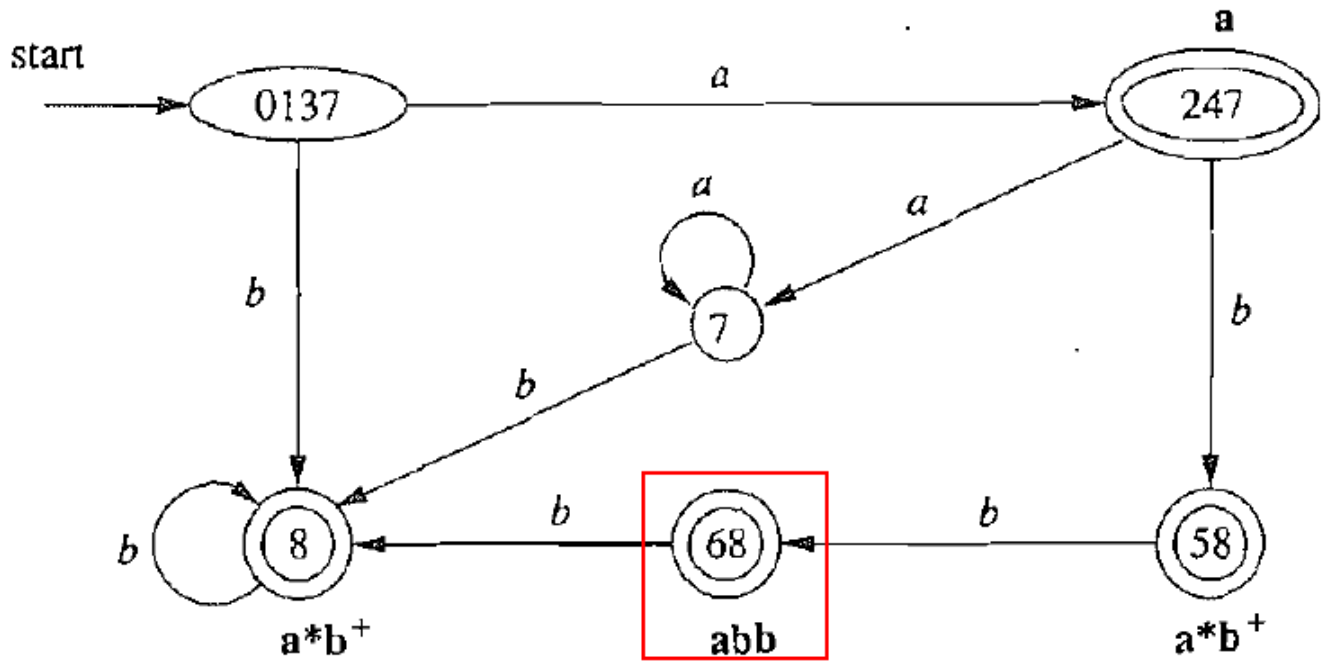
否则, **回溯** (包括状态与输入流) 至最近一次经过的接受状态, 识别成功;

若没有经过任何接受状态, 则**报错** (忽略第一个字符)

无论成功还是失败, 都从**初始状态**开始继续识别下一个词法单元

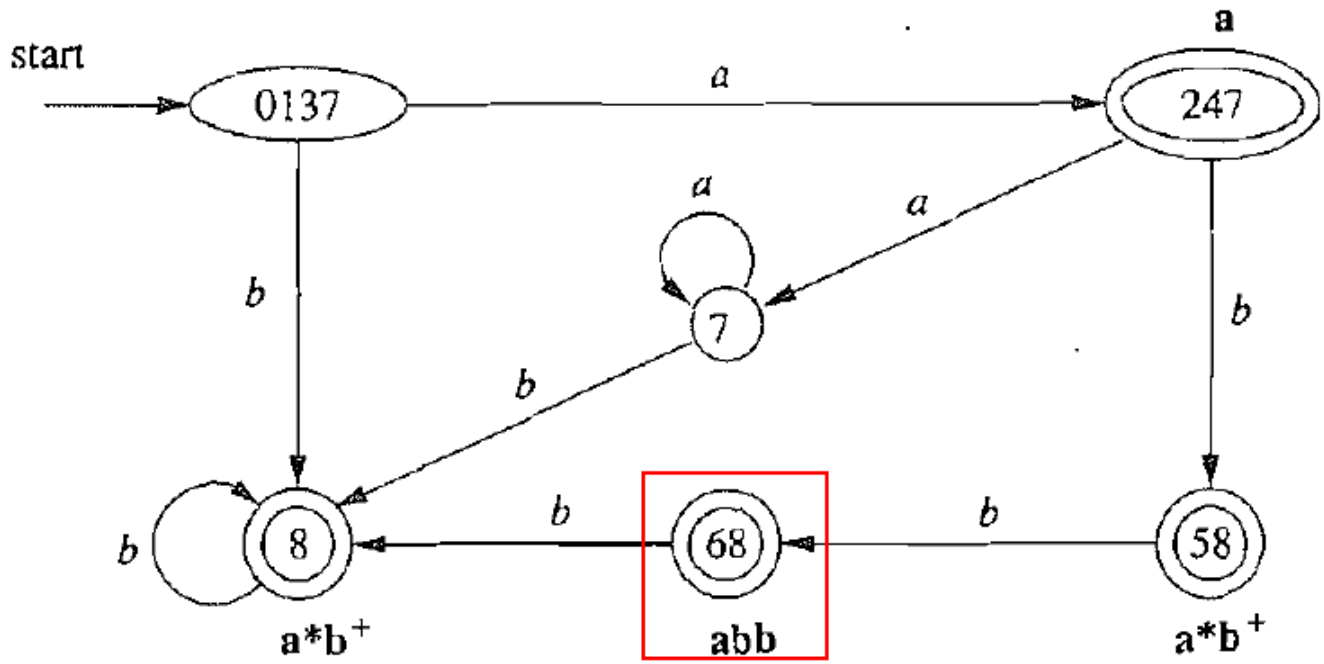


$x = a$　　输入结束; 接受; 识别出 a



$x = a$　　输入结束; 接受; 识别出 a

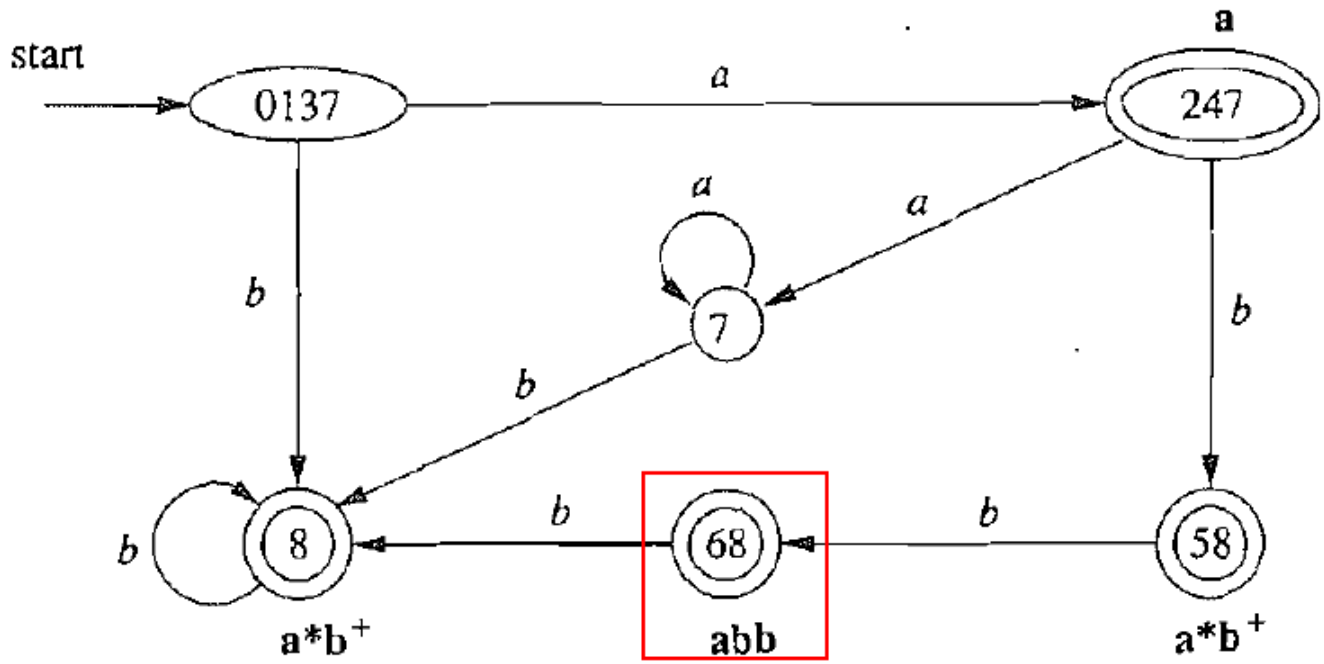
$x = abba$　　状态无转移; 回溯成功; 识别出 abb



$x = a$　　输入结束; 接受; 识别出 a

$x = abba$　　状态无转移; 回溯成功; 识别出 abb

$x = aaaa$　　输入结束; 回溯成功; 识别出 a



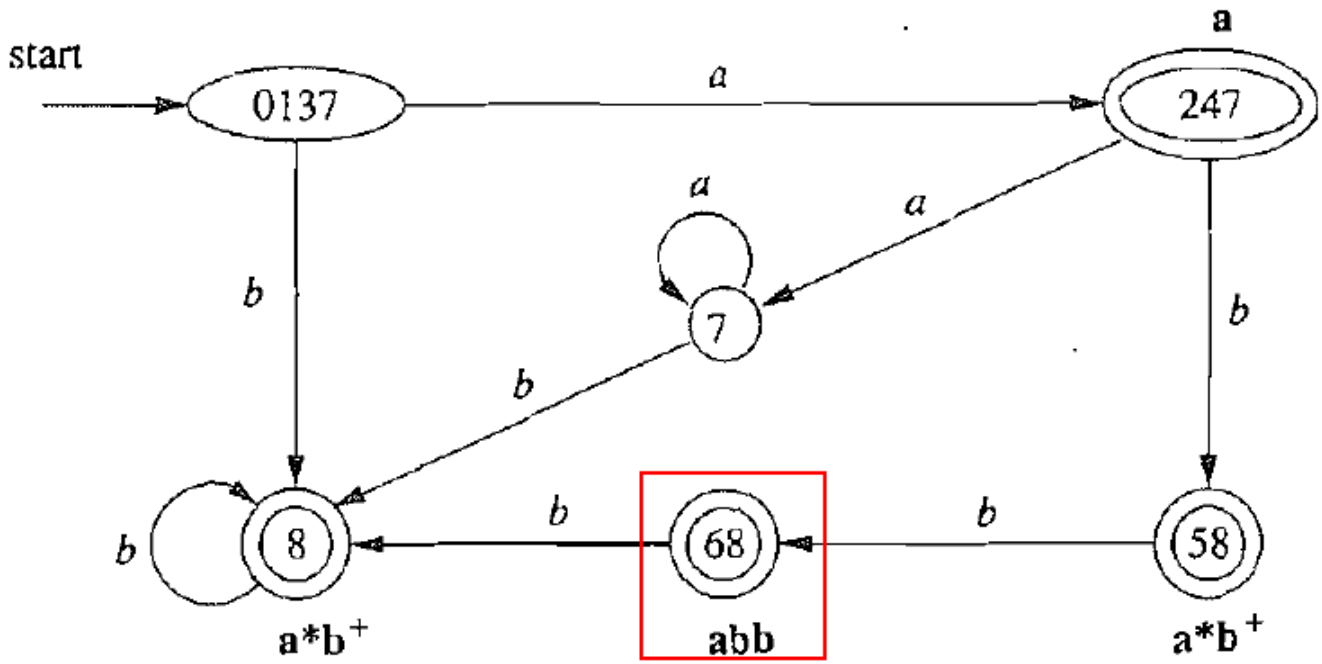
$x = a$ 输入结束; 接受; 识别出 a

$x = abba$ 状态无转移; 回溯成功; 识别出 abb

$x = aaaa$ 输入结束; 回溯成功; 识别出 a

$x = cabb$ 状态无转移; 回溯失败; 报错 c

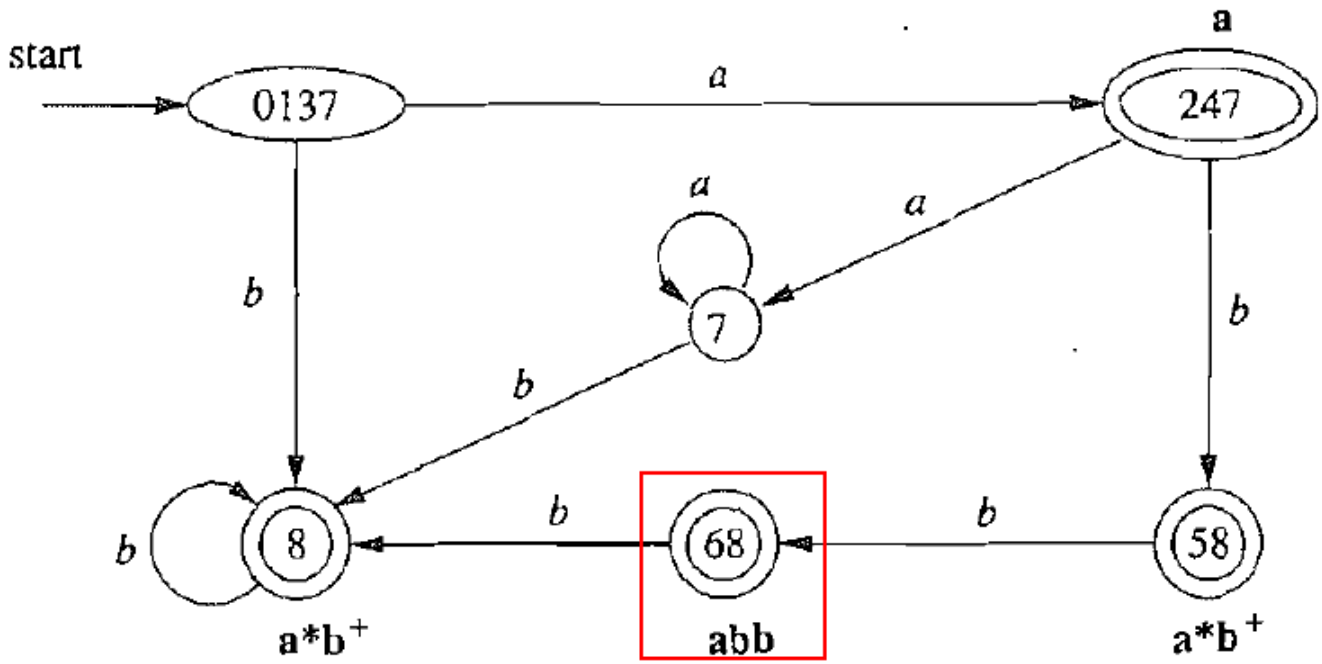
特定于词法分析器的 DFA 最小化方法



初始划分需要考虑不同的**词法单元**

$$\Pi_0 = \{\{0137, 7\}, \{247\}, \{8, 58\}, \{68\}, \{\emptyset\}\}$$

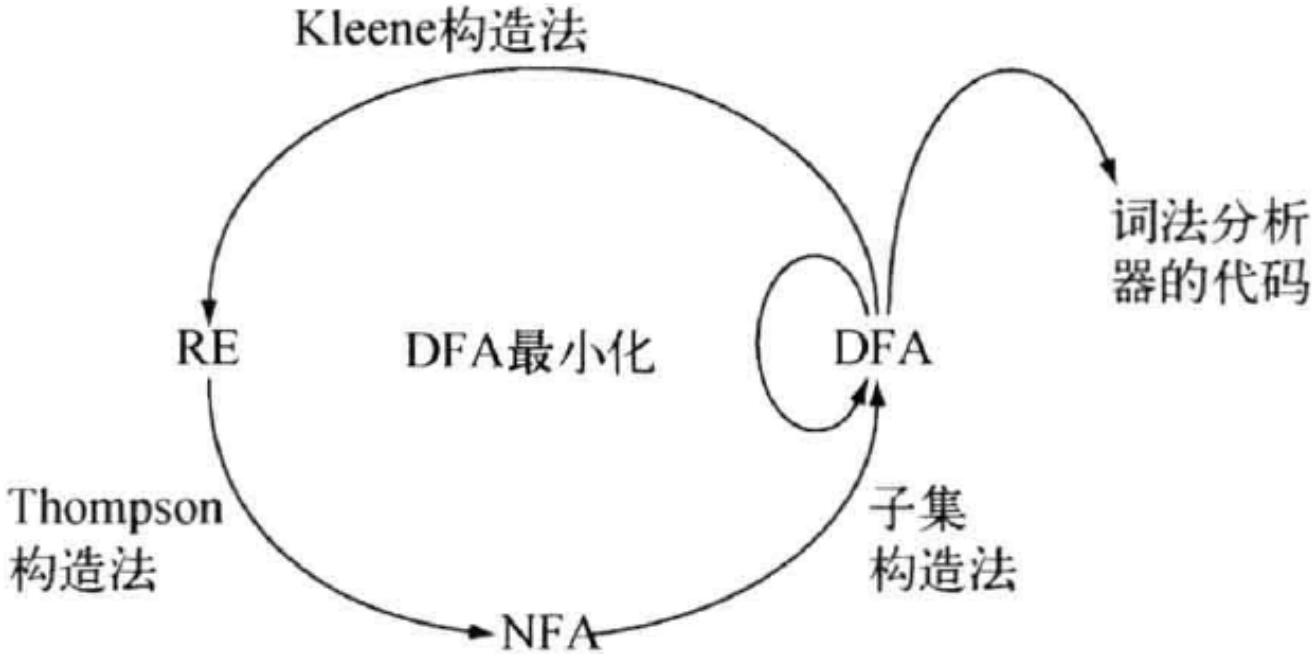
特定于词法分析器的 DFA 最小化方法



初始划分需要考虑不同的**词法单元**

$$\Pi_0 = \{\{0137, 7\}, \{247\}, \{8, 58\}, \{68\}, \{\emptyset\}\}$$

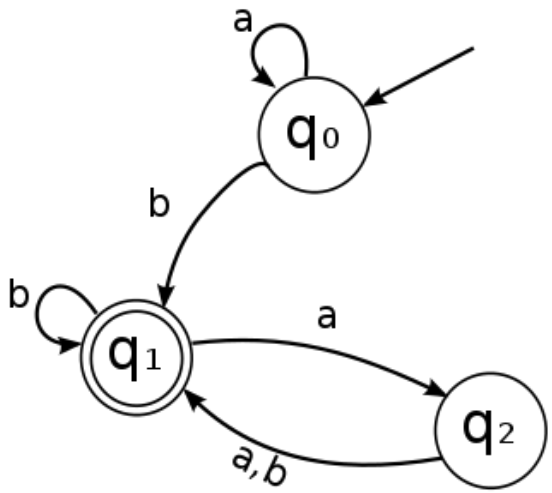
$$\Pi_1 = \{\{0137\}, \{7\}, \{247\}, \{8\}, \{58\}, \{68\}\}$$

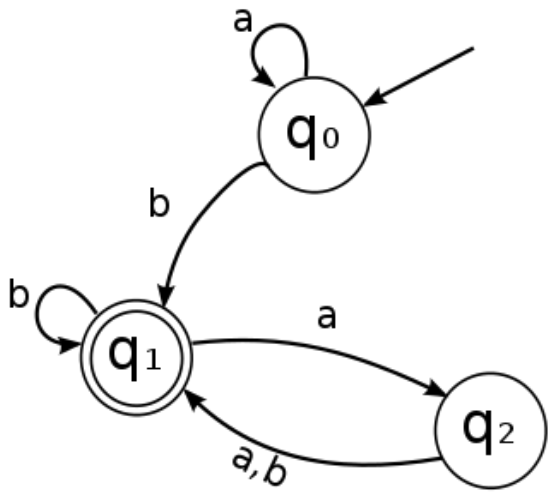


$$\text{DFA} \Longrightarrow \text{RE}$$

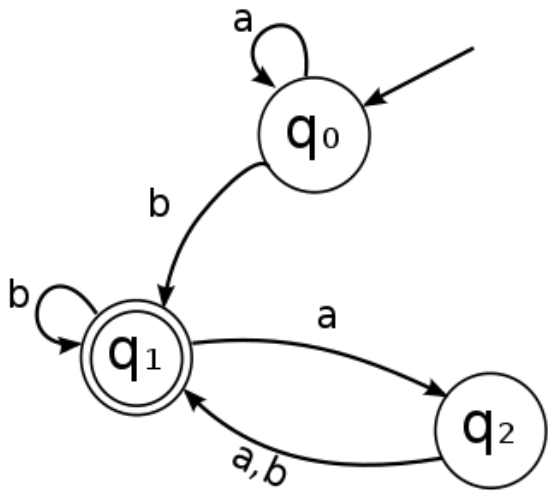
$$\boxed{D \Longrightarrow r}$$

要求：$L(r) = L(D)$


a
q0
b
b
q1
a
q2
a,b

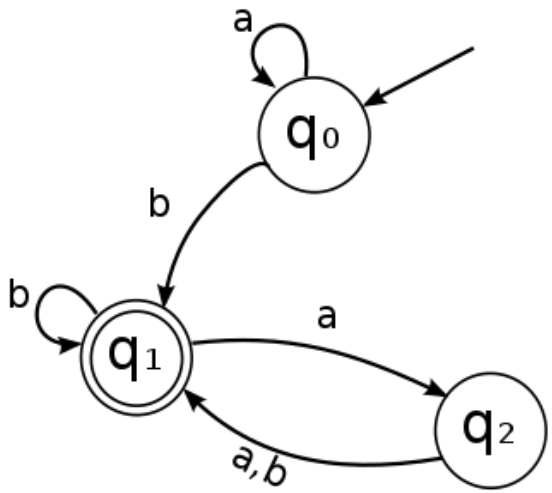


$$L(D) = \{x \mid \exists f \in F_D.\ s_0 \xrightarrow{x} f\}$$



$$L(D) = \{x \mid \exists f \in F_D.\ s_0 \xrightarrow{x} f\}$$

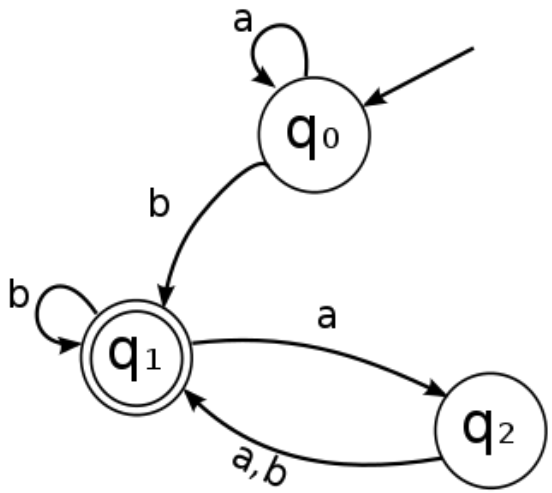
$$r = \left|_{x \in L(D)}\right. \ x$$



$$L(D) = \{x \mid \exists f \in F_D.\ s_0 \xrightarrow{x} f\}$$

$$r = \left|_{x \in L(D)}\right. \ x$$

字符串 x 对应于有向图中的路径

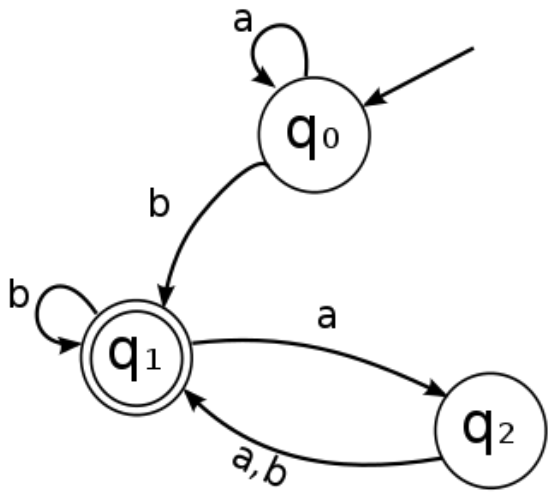


$$L(D) = \{x \mid \exists f \in F_D.\ s_0 \xrightarrow{x} f\}$$

$$r = \left|_{x \in L(D)}\right.\ x$$

字符串 x 对应于有向图中的路径

求有向图中所有 (从初始状态到接受状态的) 路径



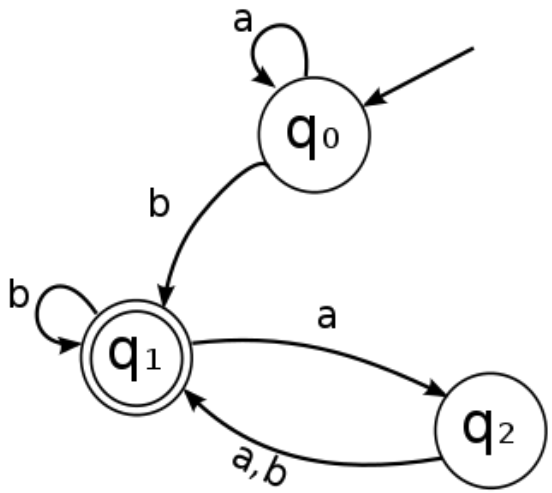
$$L(D) = \{x \mid \exists f \in F_D.\ s_0 \xrightarrow{x} f\}$$

$$r = \left|_{x \in L(D)}\right.\ x$$

字符串 x 对应于有向图中的路径

求有向图中所有 (从初始状态到接受状态的) 路径

但是, 如果有向图中含有**环**, 则存在无穷多条路径



$$L(D) = \{x \mid \exists f \in F_D.\ s_0 \xrightarrow{x} f\}$$

$$r = \left|_{x \in L(D)}\right.\ x$$

字符串 x 对应于有向图中的路径

求有向图中所有 (从初始状态到接受状态的) 路径

但是, 如果有向图中含有**环**, 则存在无穷多条路径

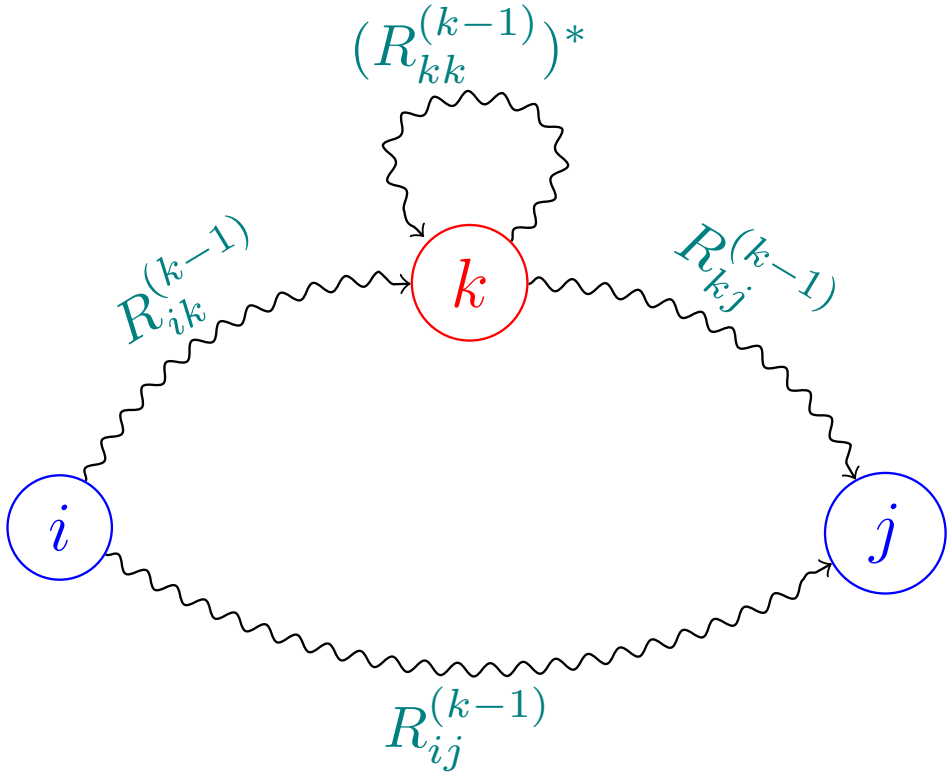
不要怕, 我们有 Kleene 闭包

假设有向图中节点编号为 0 (初始状态) 到 $n-1$

R_{ij}^k：从节点 i 到节点 j、且**中间节点**编号 $\leq k$ 的所有路径

假设有向图中节点编号为 0 (初始状态) 到 $n-1$

R_{ij}^k：从节点 i 到节点 j、且**中间节点**编号 $\leq k$ 的所有路径



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

Q：如何初始化?

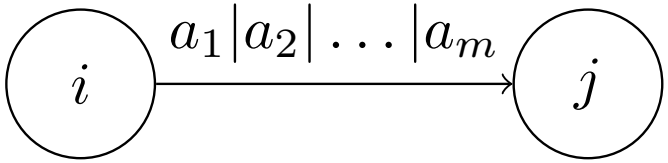
R_{ij}^{-1}：

Q：如何初始化?

R_{ij}^{-1}：从节点 i 到节点 j、且**不经过中间节点**的所有路径

Q：如何初始化?

R_{ij}^{-1}：从节点 i 到节点 j、且**不经过中间节点**的所有路径

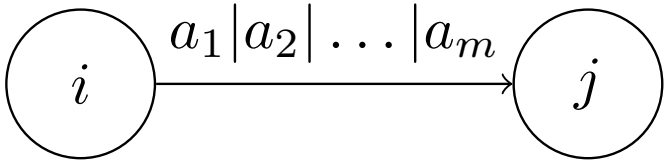


$$R_{ij}^{-1} = a_1|a_2|\ldots|a_m$$

$R_{ij}^{-1} = \emptyset$ (“无路可走”)

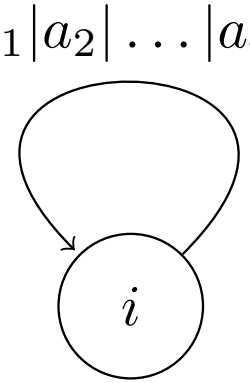
Q：如何初始化?

R_{ij}^{-1}：从节点 i 到节点 j、且**不经过中间节点**的所有路径

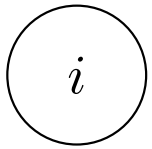


$$R_{ij}^{-1} = a_1|a_2|\ldots|a_m$$

$$R_{ij}^{-1} = \emptyset \text{ (“无路可走”)}$$



$$R_{ii}^{-1} = a_1|a_2|\ldots|a_m|\epsilon$$



$$R_{ii}^{-1} = \epsilon \text{ (“无需走路”)}$$

关于 $\emptyset$ (注意: 它不是正则表达式) 的规定

$$\emptyset r = r\emptyset = \emptyset$$

$$\emptyset | r = r$$

Q : r 的最终结果是什么?

求有向图中所有 (从初始状态到接受状态的) 路径

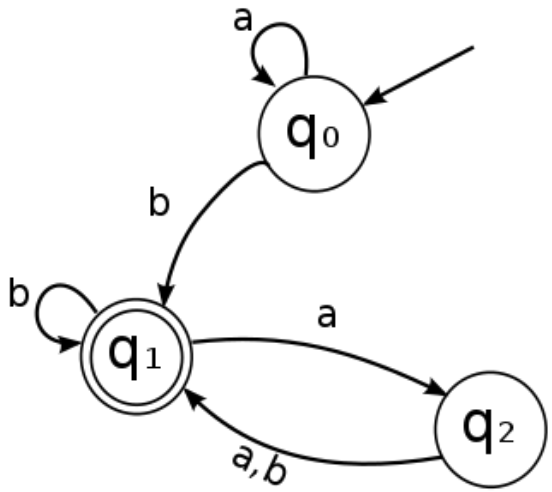
Q：r 的最终结果是什么?

求有向图中所有 (从初始状态到接受状态的) 路径

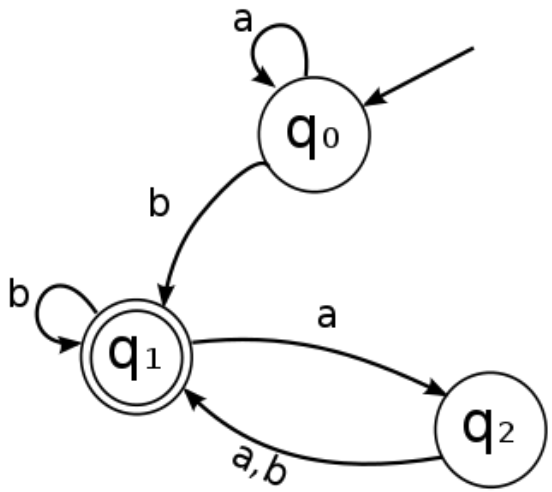
$$r = |_{s_j \in F_D} R_{0j}^{|S_D|-1}$$

```
for i = 0 to |D| − 1
    for j = 0 to |D| − 1
        R_ij^{-1} = { a | δ(d_i, a) = d_j }
        if (i = j) then
            R_ij^{-1} = R_ij^{-1} | { ε }
for k = 0 to |D| − 1
    for i = 0 to |D| − 1
        for j = 0 to |D| − 1
            R_ij^k = R_ik^{k-1} (R_kk^{k-1})* R_kj^{k-1} | R_ij^{k-1}
L = |_{s_j ∈ D_A}  R_0j^{|D|-1}
```

$|D|$: 状态数 ($|S_D|$); D_A: 接受状态集合 (F_D)

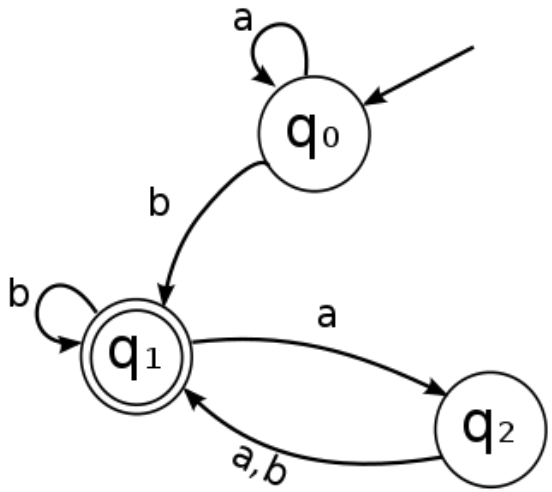


$$
\begin{aligned}
R_{00}^{-1} &= a \mid \varepsilon \\
R_{01}^{-1} &= b \\
R_{02}^{-1} &= \varnothing \\
R_{10}^{-1} &= \varnothing \\
R_{11}^{-1} &= b \mid \varepsilon \\
R_{12}^{-1} &= a \\
R_{20}^{-1} &= \varnothing \\
R_{21}^{-1} &= a \mid b \\
R_{22}^{-1} &= \varepsilon
\end{aligned}
$$



Step 0

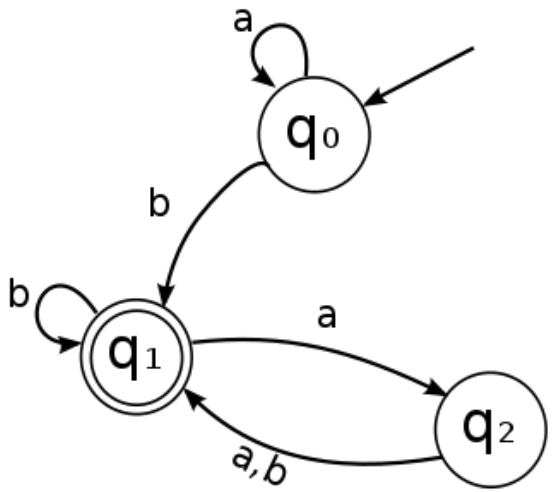
$$
\begin{array}{llll}
R_{00}^{0} & = R_{00}^{-1} (R_{00}^{-1})^{*} R_{00}^{-1} \mid R_{00}^{-1} & = (a \mid \varepsilon)\,(a \mid \varepsilon)^{*}\,(a \mid \varepsilon) \mid a \mid \varepsilon & = a^{*} \\
R_{01}^{0} & = R_{00}^{-1} (R_{00}^{-1})^{*} R_{01}^{-1} \mid R_{01}^{-1} & = (a \mid \varepsilon)\,(a \mid \varepsilon)^{*}\, b \mid b & = a^{*}\, b \\
R_{02}^{0} & = R_{00}^{-1} (R_{00}^{-1})^{*} R_{02}^{-1} \mid R_{02}^{-1} & = (a \mid \varepsilon)\,(a \mid \varepsilon)^{*}\, \varnothing \mid \varnothing & = \varnothing \\
R_{10}^{0} & = R_{10}^{-1} (R_{00}^{-1})^{*} R_{00}^{-1} \mid R_{10}^{-1} & = \varnothing\ (a \mid \varepsilon)^{*}\,(a \mid \varepsilon) \mid \varnothing & = \varnothing \\
R_{11}^{0} & = R_{10}^{-1} (R_{00}^{-1})^{*} R_{01}^{-1} \mid R_{11}^{-1} & = \varnothing\ (a \mid \varepsilon)^{*}\, b \mid b \mid \varepsilon & = b \mid \varepsilon \\
R_{12}^{0} & = R_{10}^{-1} (R_{00}^{-1})^{*} R_{02}^{-1} \mid R_{12}^{-1} & = \varnothing\ (a \mid \varepsilon)^{*}\, \varnothing \mid a & = a \\
R_{20}^{0} & = R_{20}^{-1} (R_{00}^{-1})^{*} R_{00}^{-1} \mid R_{20}^{-1} & = \varnothing\ (a \mid \varepsilon)^{*}\,(a \mid \varepsilon) \mid \varnothing & = \varnothing \\
R_{21}^{0} & = R_{20}^{-1} (R_{00}^{-1})^{*} R_{01}^{-1} \mid R_{21}^{-1} & = \varnothing\ (a \mid \varepsilon)^{*}\, b \mid a \mid b & = a \mid b \\
R_{22}^{0} & = R_{20}^{-1} (R_{00}^{-1})^{*} R_{02}^{-1} \mid R_{22}^{-1} & = \varnothing\ (a \mid \varepsilon)^{*}\, \varnothing \mid \varepsilon & = \varepsilon
\end{array}
$$



Step 1

$$
\begin{array}{llllll}
R^1_{00} & = R^0_{01} (R^0_{11})^* R^0_{10} \mid R^0_{00} & = a^*b & (b \mid \varepsilon)^* \varnothing & \mid a^* & = a^* \\
R^1_{01} & = R^0_{01} (R^0_{11})^* R^0_{11} \mid R^0_{01} & = a^*b & (b \mid \varepsilon)^* (b \mid \varepsilon) & \mid a^* b & = a^* b^* b \\
R^1_{02} & = R^0_{01} (R^0_{11})^* R^0_{12} \mid R^0_{02} & = a^*b & (b \mid \varepsilon)^* a & \mid \varnothing & = a^* b^* ba \\
R^1_{10} & = R^0_{11} (R^0_{11})^* R^0_{10} \mid R^0_{10} & = (b \mid \varepsilon) & (b \mid \varepsilon)^* \varnothing & \mid \varnothing & = \varnothing \\
R^1_{11} & = R^0_{11} (R^0_{11})^* R^0_{11} \mid R^0_{11} & = (b \mid \varepsilon) & (b \mid \varepsilon)^* (b \mid \varepsilon) & \mid b \mid \varepsilon & = b^* \\
R^1_{12} & = R^0_{11} (R^0_{11})^* R^0_{12} \mid R^0_{12} & = (b \mid \varepsilon) & (b \mid \varepsilon)^* a & \mid a & = b^* a \\
R^1_{20} & = R^0_{21} (R^0_{11})^* R^0_{10} \mid R^0_{20} & = (a \mid b) & (b \mid \varepsilon)^* \varnothing & \mid \varnothing & = \varnothing \\
R^1_{21} & = R^0_{21} (R^0_{11})^* R^0_{11} \mid R^0_{21} & = (a \mid b) & (b \mid \varepsilon)^* (b \mid \varepsilon) & \mid a \mid b & = (a \mid b) b^* \\
R^1_{22} & = R^0_{21} (R^0_{11})^* R^0_{12} \mid R^0_{22} & = (a \mid b) & (b \mid \varepsilon)^* a & \mid \varepsilon & = (a \mid b) b^* a \mid \varepsilon
\end{array}
$$

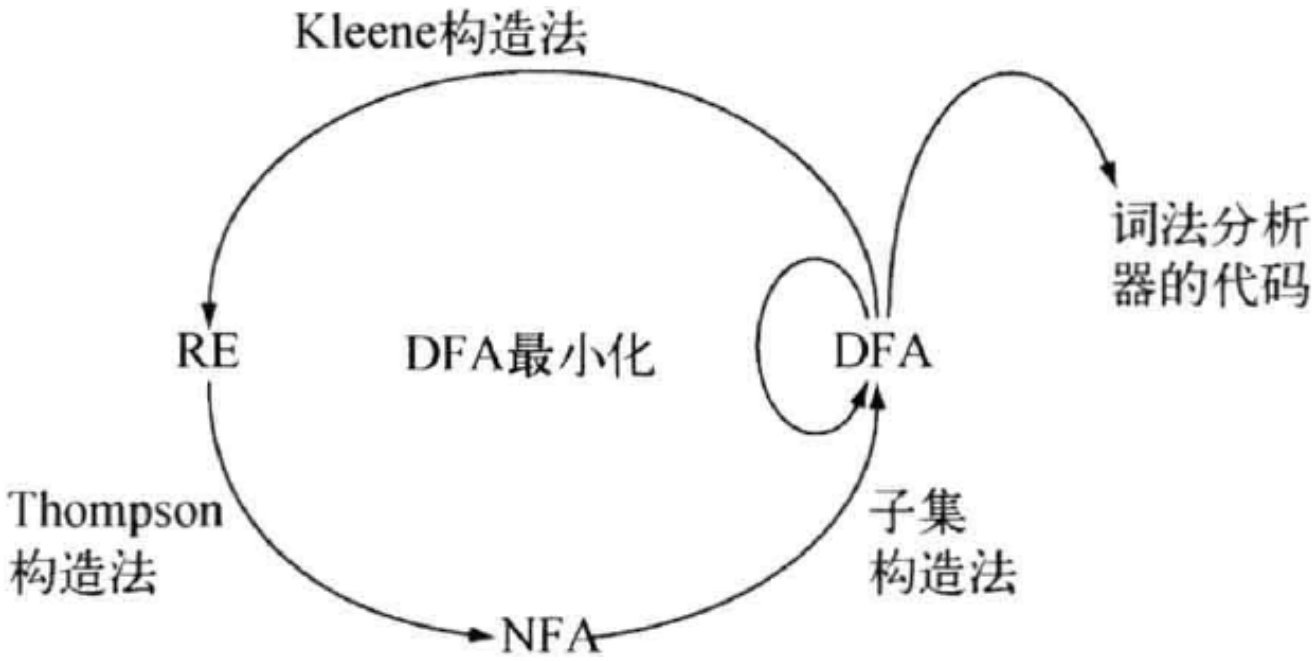
$$a^*b(a(a|b)|b)^*$$

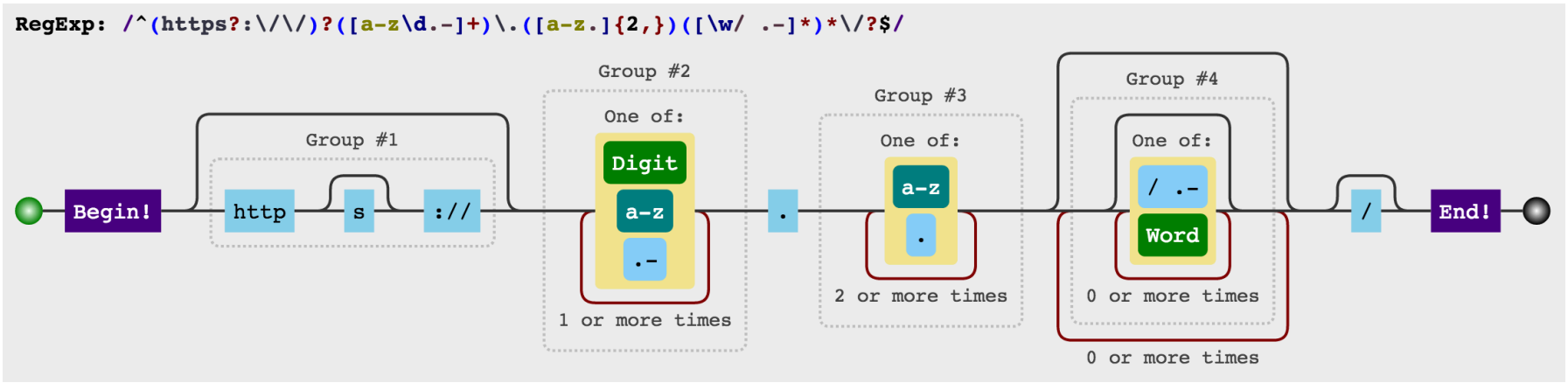


Step 2

R^2_{00}	$= R^1_{02}(R^1_{22})^* R^1_{20} \mid R^1_{00}$	$= a^*b^*ba$	$((a\|b)b^*a \mid \varepsilon)^* \varnothing$	$\mid a^*$	$= a^*$
R^2_{01}	$= R^1_{02}(R^1_{22})^* R^1_{21} \mid R^1_{01}$	$= a^*b^*ba$	$((a\|b)b^*a \mid \varepsilon)^* (a\|b)b^*$	$\mid a^* b^* b$	$= a^* b (a (a \mid b) \mid b)^*$
R^2_{02}	$= R^1_{02}(R^1_{22})^* R^1_{22} \mid R^1_{02}$	$= a^*b^*ba$	$((a\|b)b^*a \mid \varepsilon)^* ((a\|b)b^*a \mid \varepsilon)$	$\mid a^* b^* ba$	$= a^* b^* b (a (a \mid b) b^*)^* a$
R^2_{10}	$= R^1_{12}(R^1_{22})^* R^1_{20} \mid R^1_{10}$	$= b^* a$	$((a\|b)b^*a \mid \varepsilon)^* \varnothing$	$\mid \varnothing$	$= \varnothing$
R^2_{11}	$= R^1_{12}(R^1_{22})^* R^1_{21} \mid R^1_{11}$	$= b^* a$	$((a\|b)b^*a \mid \varepsilon)^* (a\|b)b^*$	$\mid b^*$	$= (a (a \mid b) \mid b)^*$
R^2_{12}	$= R^1_{12}(R^1_{22})^* R^1_{22} \mid R^1_{12}$	$= b^* a$	$((a\|b)b^*a \mid \varepsilon)^* ((a\|b)b^*a \mid \varepsilon)$	$\mid b^* a$	$= (a (a \mid b) \mid b)^* a$
R^2_{20}	$= R^1_{22}(R^1_{22})^* R^1_{20} \mid R^1_{20}$	$= ((a\|b)b^*a \mid \varepsilon)$	$((a\|b)b^*a \mid \varepsilon)^* \varnothing$	$\mid \varnothing$	$= \varnothing$
R^2_{21}	$= R^1_{22}(R^1_{22})^* R^1_{21} \mid R^1_{21}$	$= ((a\|b)b^*a \mid \varepsilon)$	$((a\|b)b^*a \mid \varepsilon)^* (a\|b)b^*$	$\mid (a \mid b) b^*$	$= (a \mid b) (a (a \mid b) \mid b)^*$
R^2_{22}	$= R^1_{22}(R^1_{22})^* R^1_{22} \mid R^1_{22}$	$= ((a\|b)b^*a \mid \varepsilon)$	$((a\|b)b^*a \mid \varepsilon)^* ((a\|b)b^*a \mid \varepsilon)$	$\mid (a \mid b) b^* a \mid \varepsilon$	$= ((a \mid b) b^* a)^*$

目标：正则表达式 RE $\Longrightarrow$ 词法分析器





Learn More about Regular Expressions

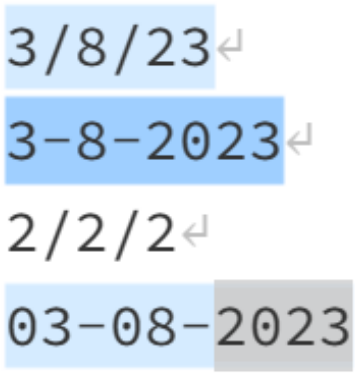
However, many tools, libraries, and engines that provide such constructions still use the term *regular expression* for their patterns. This has led to a nomenclature where the term regular expression has different meanings in formal language theory and pattern matching. For this reason, some people have taken to using the term *regex*, *regexp*, or simply *pattern* to describe the latter. Larry Wall, author of the Perl programming language, writes in an essay about the design of Raku:

> "Regular expressions" [...] are only marginally related to real regular expressions. Nevertheless, the term has grown with the capabilities of our pattern matching engines, so I'm not going to try to fight linguistic necessity here. I will, however, generally call them "regexes" (or "regexen", when I'm in an Anglo-Saxon mood).[18]

https://en.wikipedia.org/wiki/Regular_expression#Patterns_for_non-regular_languages

```
body {
  background-color: #fefbd8;
}

h1 {
  background-color: #0000ff;
}

div {
  background-color: #d0f4e6;
}

span {
  background-color: #f08970;
}
```

https://regex101.com/r/jucEtW/1 (regex/bg-color)



https://regex101.com/r/jchuZs/1 (regex/date)

```
1001: $496.80
1002: $1290.89
1003: $26.43
1004: 613.42
1005: 7.61
1006: $414.90
1007: $25.00
```

https://regex101.com/r/fWJkCF/1 (regex/dollar)

The cat scattered his food all over the room.

https://regex101.com/r/K5MCMZ/1 (regex/cat)

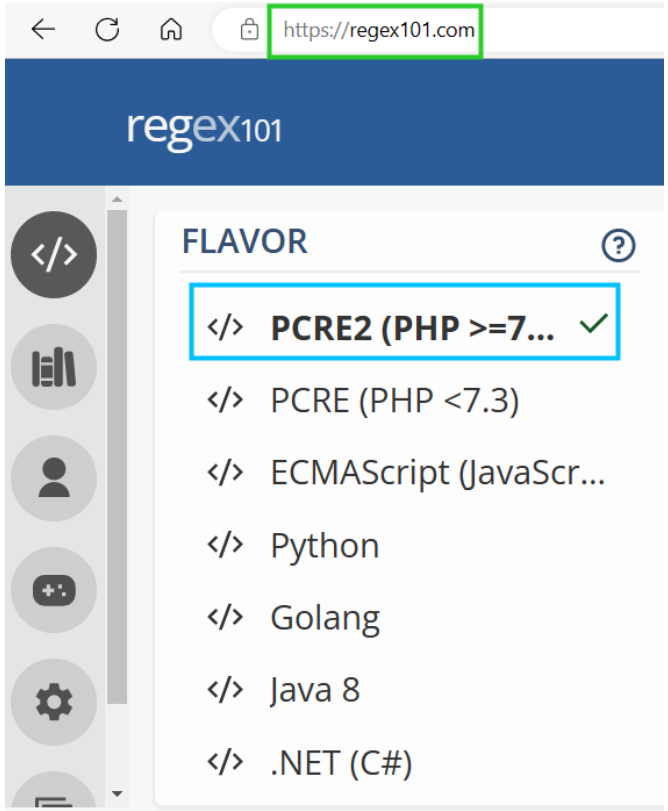
REGULAR EXPRESSION v1

/ ^(0|(1(01*0)*1))*$

TEST STRING

```
0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
10000
10001
10010
10011
10100
```

https://regex101.com/r/C0m3kB/1



3 的倍数 (二进制表示)

```
<body>
  <H1>Welcome to my Homepage</H1>
  Content is divided into two sections:<br/>
  <h2>SQL</h2>
  Information about SQL.
  <h2>RegEx</h2>
  Information about Regular Expressions.
  <h3>This is not a valid HTML</h4>
</body>
```

https://regex101.com/r/PUsCwP/1 (regex/html-head)

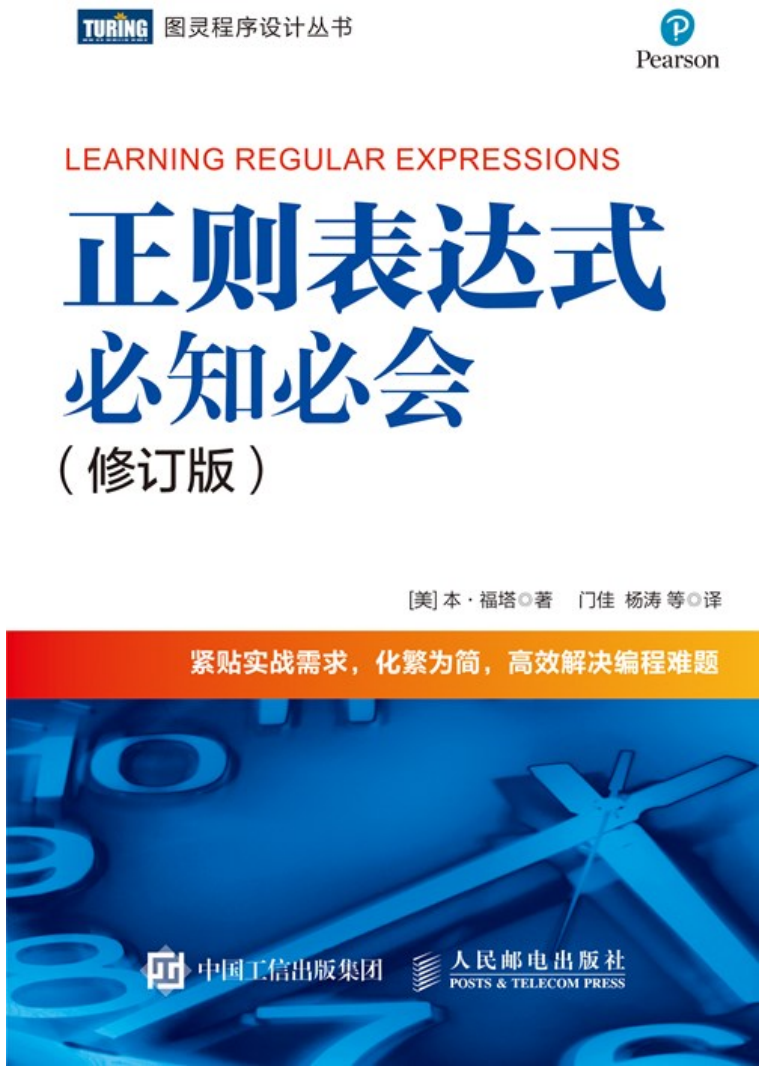
```
<body>
  <H1>Welcome to my Homepage</H1>
  Content is divided into two sections:<br/>
  <h2>SQL</h2>
  Information about SQL.
  <h2>RegEx</h2>
  Information about Regular Expressions.
  <h3>This is not a valid HTML</h4>
</body>
```

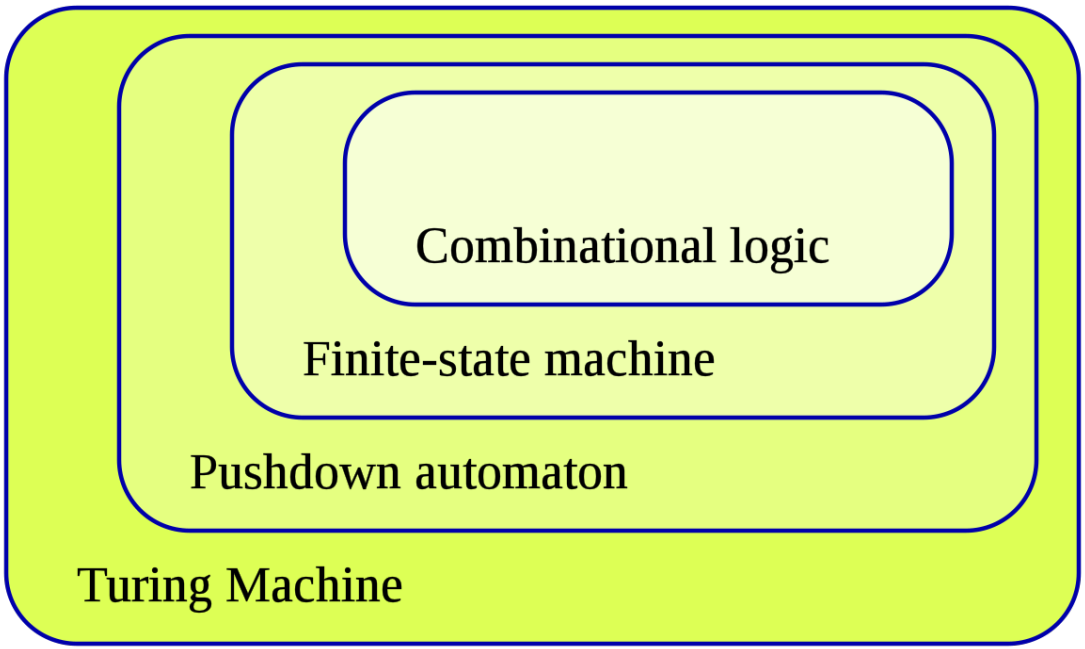
https://regex101.com/r/eXue43/1

(regex/html-head-lookaround)

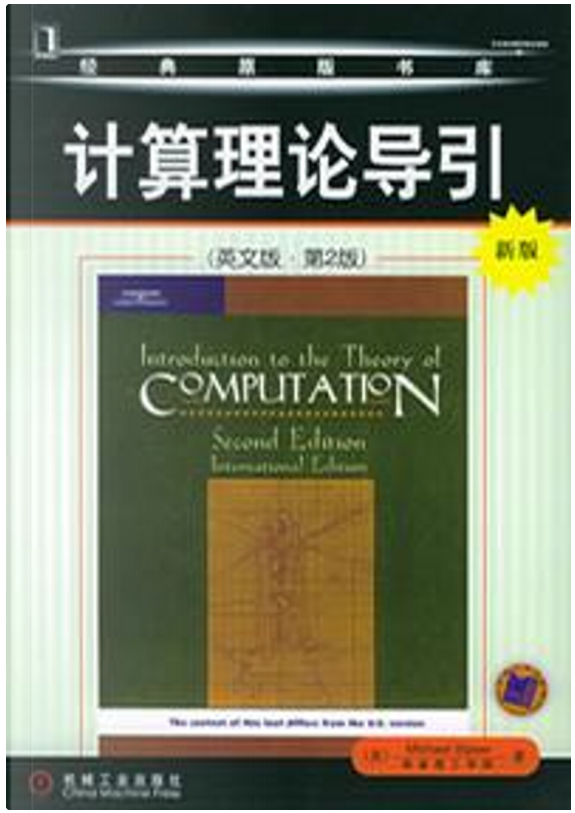
```
<!--·Nav·bar·-->
<div>
··<a·href·=·"/home"><img·src·=·"images/home.gif"></a>
··<img·src·=·"/images/spacer.gif">
··<a·href·=·"search"><img·src·=·"/images/search.gif"></a>
··<img·src·=·"/images/spacer.gif">
··<a·href·=·"help"><img·src·=·"/images/help.gif"></a>
</div>
```

https://regex101.com/r/l07Gpu/1 (regex/html-a-img)

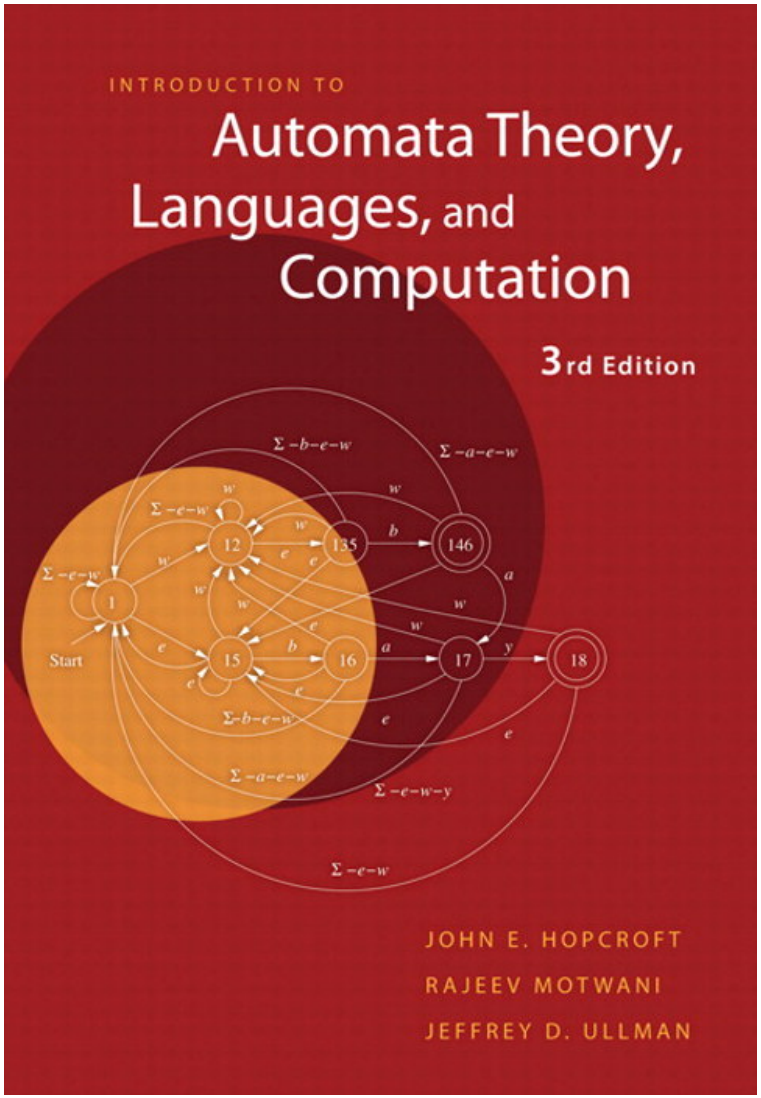

TURING 图灵程序设计丛书
Pearson
LEARNING REGULAR EXPRESSIONS
正则表达式
必知必会
（修订版）
[美] 本·福塔◎著　门佳 杨涛 等◎译
紧贴实战需求，化繁为简，高效解决编程难题
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS



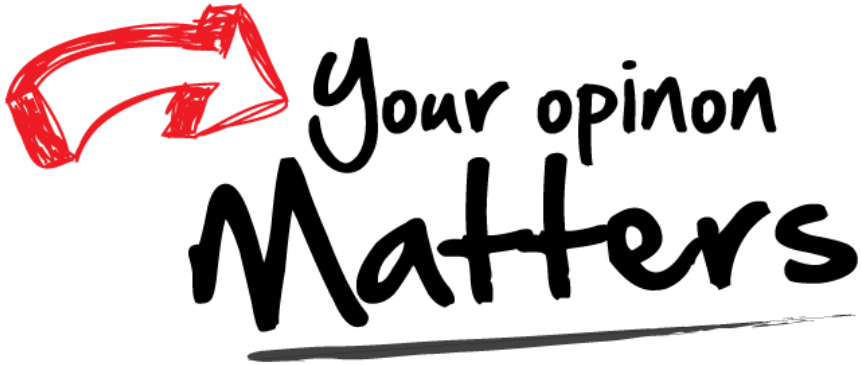
根据**表达/计算能力**的强弱，自动机可以分为不同层次。



https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/


INTRODUCTION TO
Automata Theory, Languages, and Computation
3rd Edition
Start
JOHN E. HOPCROFT
RAJEEV MOTWANI
JEFFREY D. ULLMAN


Thank
You!



Office 926

hfwei@nju.edu.cn