3.4 正规语言描述方法间的相互转换

- 众所周知，正规语言有三种等价的表示方法：

（1）正规文法 （2）正规式 （3）有限自动机

- 我们以有限自动机为核心，介绍彼此间的转换关系：

Ⅰ. 正规文法 <=> DFA

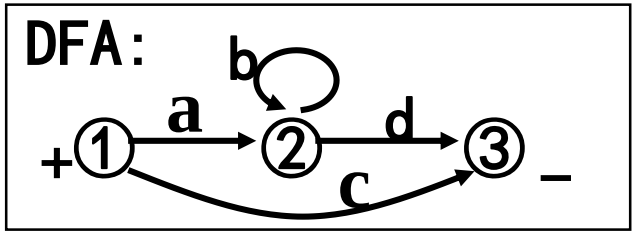
设 $G(Z)=(V_N, V_T, Z, P)$， $DFA=(Q, \Sigma, s, F, \delta)$

则有：

正规文法	DFA
V_N（非终结符集）	Q（状态集）
V_T（终结符集）	Σ（字符集）
Z（开始符号）	S（开始状态）
A -> aB	$\delta(A, a)=B$
A -> a	$\delta(A, a)=B$（结束态）
A -> ε	A（结束态）

※ 正规文法 与 DFA间转换示例:

【例3.16】 自动机 => 正规文法:



令 Z=①, A=②, B=③

则有 正规文法

G(Z): Z -> aA | cB , A ->bA | dB , B->ε

【例3.17】正规文法 => 自动机, 并求 L(G):

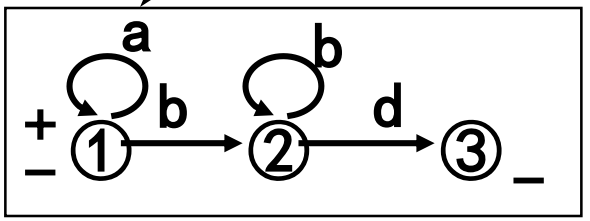
G(Z): Z->aZ|bA|ε ,A->bA|d

∵ A->d 可变换为 A->dB, B->ε

∴ G`(Z) (与 G(Z)等价):

Z->aZ|bA|ε , A->bA|dB ,B->ε

对应的DFA



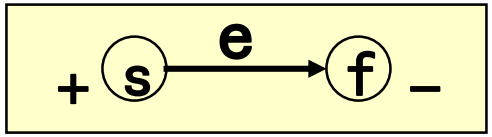
令 ①-Z, ②-A, ③-B

则 L(G)={ε, $a^m b^n d$ | m≥0, n>0}

II. 正规式 <=> DFA

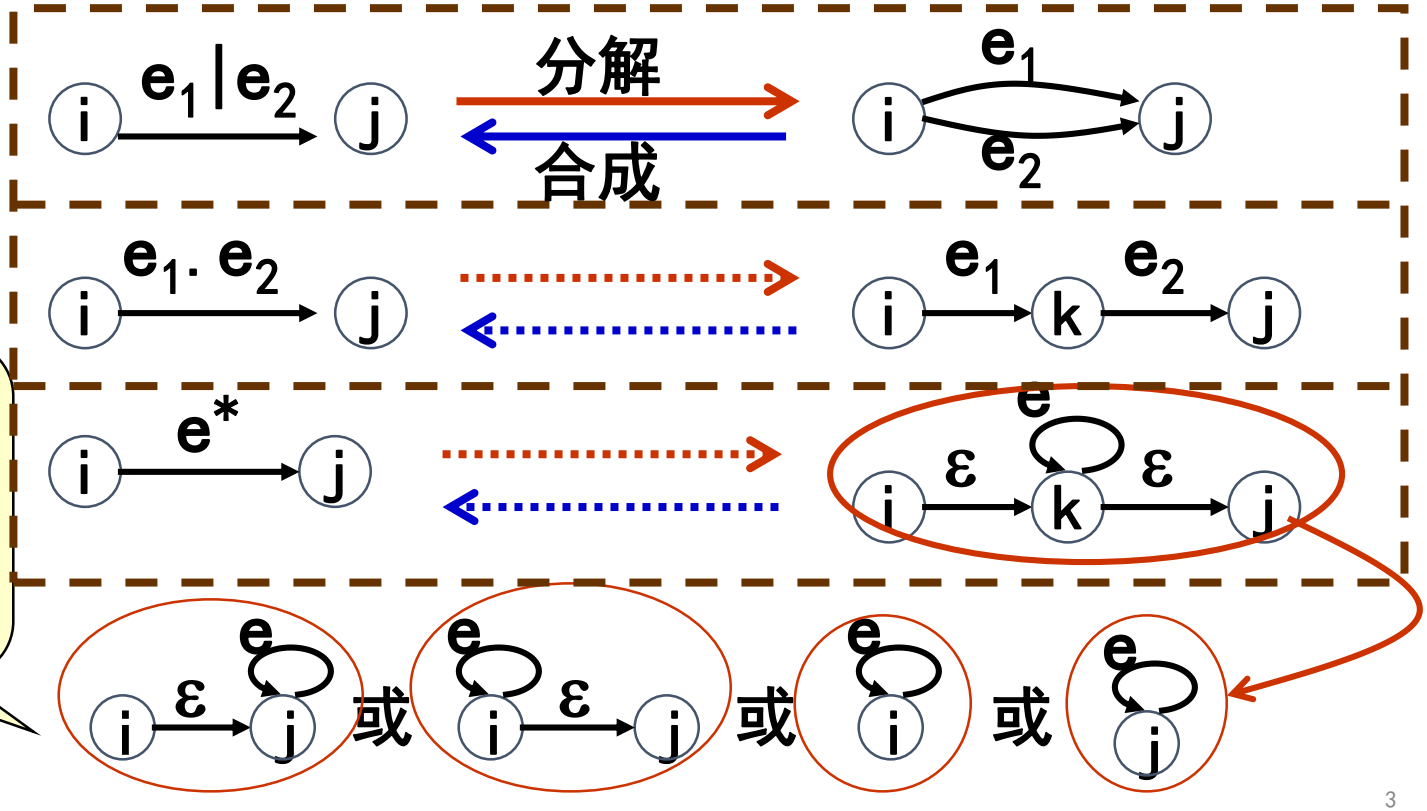
设 e 为正规式 , DFA=(Q, Σ, s, F, δ)

转换机制：



※ e => DFA 分解过程 (其逆过程为合成过程)：

则有：

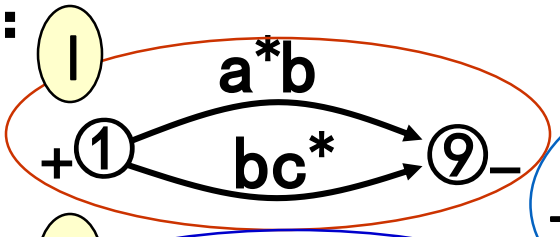


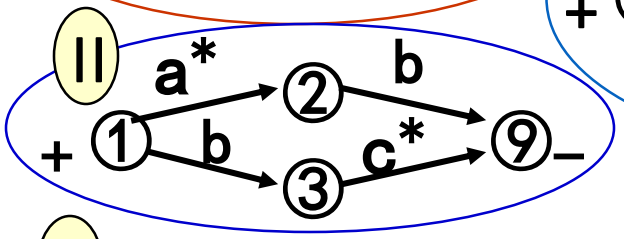
※ 正规式 与 DFA间转换示例:

【例3.18】**正规式 => 自动机**

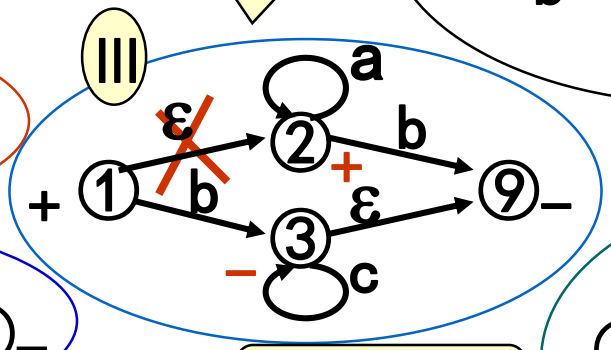
设 e = a*b|bc*

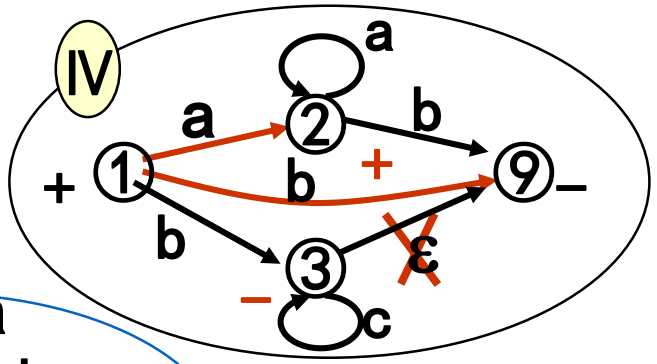
则:

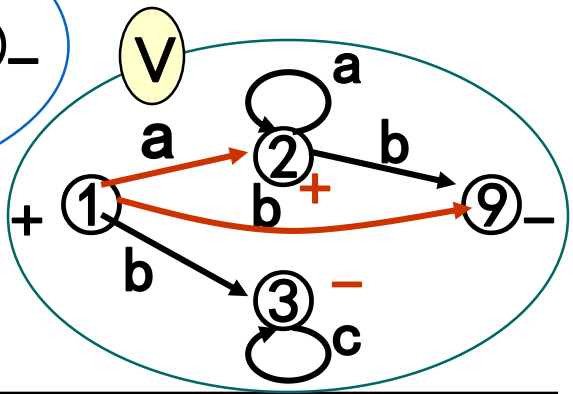




确定化1



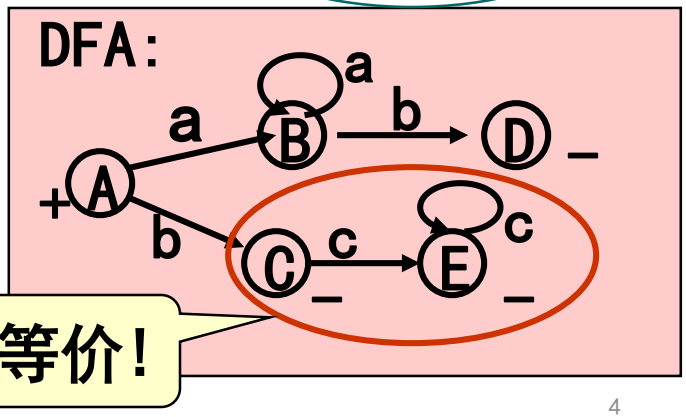




确定化2

VI

		a	b	c
+	A{1, 2}	B{2}	C{3, 9}	
	B{2}	B{2}	D{9}	
–	C{3, 9}			E{3}
–	D{9}			
–	E{3}			E{3}



等价!

3.5 有限自动机的实现问题

用计算机完成有限自动机的功能，其核心是“变换”的实现技术。这里介绍的是把变换表按某种方式存贮起来，作为知识源，实现机制是：

控制程序 ⊕ 变换表

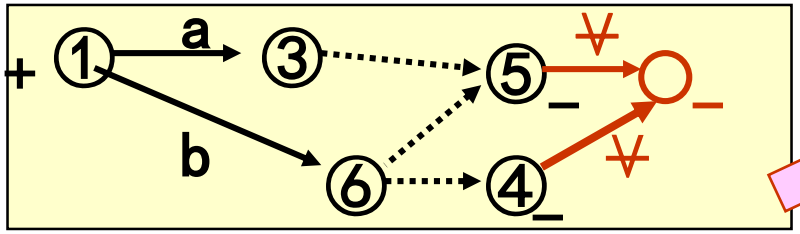
【三点说明】

(1) 假定自动机只作为识别器，即 对待识别的符号串：

仅回答 是(接受)或 否(拒绝)。

(2) 为便于处理，可令 ∀ 作为待识别的符号串的泛指后继符。

为此，扩展自动机如下：

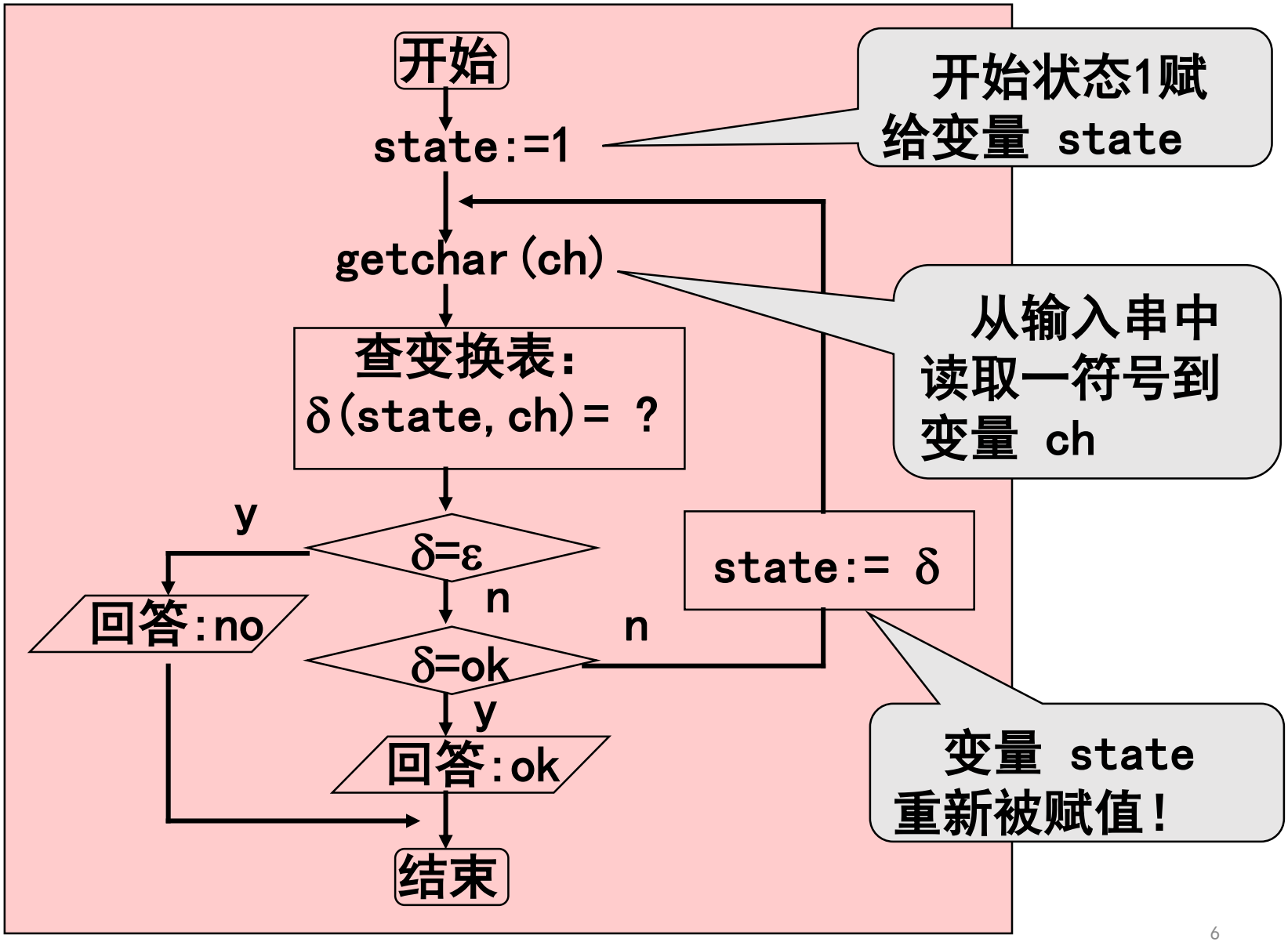


		a	b	…	∀
+	1	3	6		
	…	…	…	…	…
_	4				ok
_	5				ok

(3) 令 getchar(ch) 读符号函数。

‘空’ 则 no

3.5.1 控制程序设计


开始
state:=1
开始状态1赋给变量 state
getchar(ch)
从输入串中读取一符号到变量 ch
查变换表:
δ(state, ch)= ?
y
δ=ε
回答:no
n
δ=ok
n
state:= δ
y
回答:ok
变量 state 重新被赋值!
结束

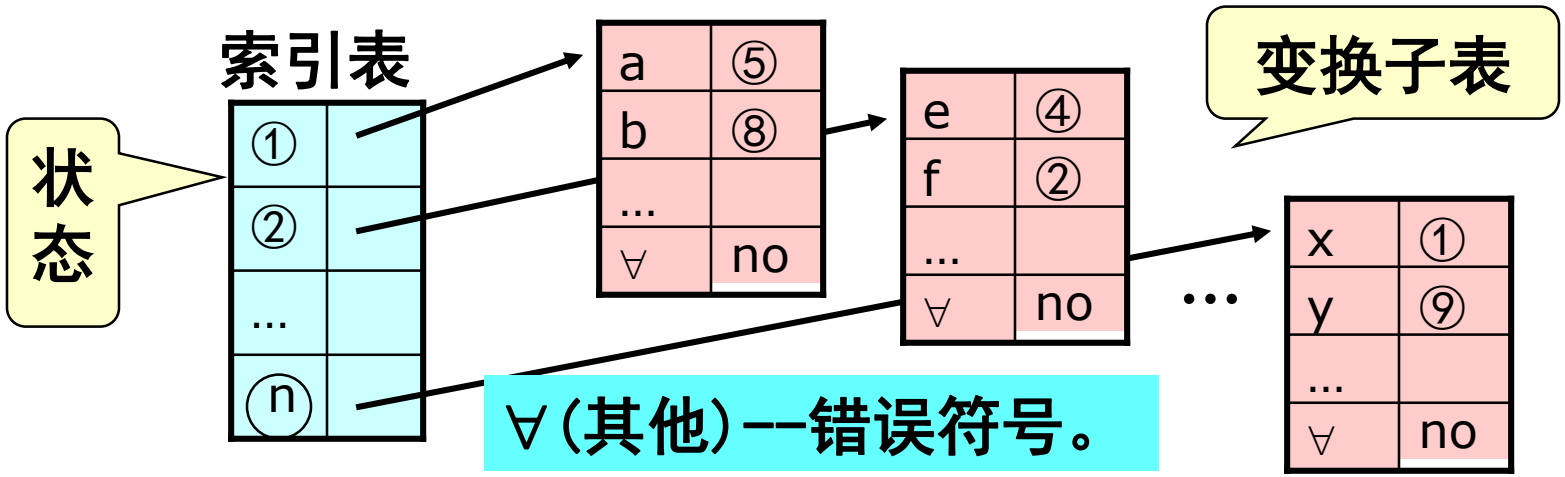
3.5.2 变换表存贮结构设计

变换表的存贮结构可选择下述两种方式之一：

(1) **二维数组**，其下标是(状态，输入符号)；

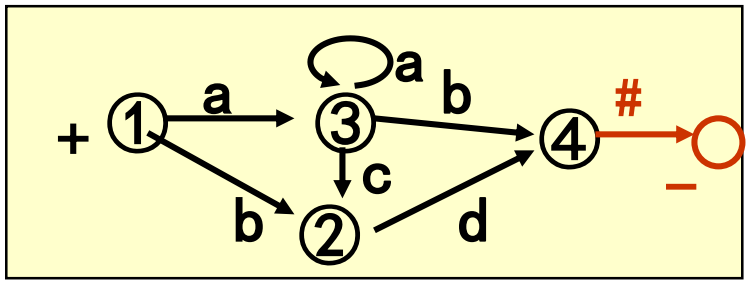
※ 为了适应不同编码语言的需要，状态和输入符号可采取相应的编码形式；通常，使用连续的正整数：0, 1, 2, 3, …。

(2) **压缩变换表**，方法是把每个状态行作为**子表**，**状态为索引**，**并把**错误的输入符号合并在一起，如：



※有限自动机实现示例:

【例3.19】 有限自动机DFA:



索引表

①	
②	
③	
④	

a	③
b	②
∀	no

d	④
∀	no

a	③
b	④
c	②
∀	no

#	ok
∀	no

输入串 aacd# 识别过程:

state	ch	剩余	变换	备注
1	a	acd#	3	
3	a	cd#	3	
3	c	d#	2	
2	d	#	4	
4	#		ok	接受

压缩变换表

谢谢收看！