第 3 章　自动机

3.1　正规语言及其描述方法

若文法中每个产生式都限制为 $A \to aB$ 或 $A \to a$，其中 A 和 B 均为单个非终结符（此文法为右线性文法，左线性文法形如 $A \to Ba$ 或 $A \to a$），则称这类文法为正规文法。

【定义】正规语言是指由正规文法定义的语言。

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

正规文法就是正规语言的一种描述形式，正规式可以简单理解为正则表达式，是用计算机语言来描述正规语言的一种方式，这三种都是等价的。

【例 3.1】 $G(Z): A \to A|\varepsilon$　（正规文法）

$\because A \Rightarrow \varepsilon;$

$A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow \ldots \Rightarrow a^n, n \geqslant 0$

$\therefore L(G) = \{a^n \mid n \geqslant 0\}$　（正规语言）

【例 3.2】$L1 = \{a^m b^n \mid m \geqslant 0, n \geqslant 1\}$, 正规语言？

$\because$ 可由正规文法定义：

$G1(S): \quad S \to aS|bA; A \to bA|\varepsilon$

$\therefore L1$ 是正规语言

图 3.1: 两个例子

如果一种语言是正规语言，那就可以找到一个正规文法去描述该语言。

【例 3.3】$L2 = \{(ab)^n \mid n \geqslant 1\}$, 正规语言？

$\because$ 可由正规文法定义：

$G2(S): S \to aA; A \to bB; B \to aA \mid \varepsilon$

$\therefore$L2 是正规语言

【例 3.4】$L3 = \{a^n b^n \mid n \geqslant 0\}$, 正规语言？

∵不能由正规文法定义(正规文法无法描述 a 和 b 数量上不相等!): ∴不是正规语言

用什么样的语言可以描述

【例 3.4】这个文法？这个模型对应的语言需要用下推机来描述。

如果上下文无关文法不含有 ε，则该文法可以被转化为正规文法。

3.1.1 正规语言的正规式表示法

※ 正规式：是指由字母表中的符号，通过以下三种运算(也可以使用圆括号)连接起来构成的表达式 e：连接(.)或(|)闭包(+，*)

正规式	正规式的值
ab.cde	abcde
ab|cde	ab, cde
a^+	$a, aa, aaa, \ldots, a^n, \ldots$
a^*	$\varepsilon, a, aa, aaa, \ldots, a^n, \ldots$

图 3.2: 正规式举例

※ 设 val(e),L(e) 分别表示正规式 e 的值和语言，则：L(e)= x| x=val(e)，即正规式表示的语言是该正规式所有值的集合(正规集)。

※ 正规式表示正规语言示例：

【例】 (1)$e1 = a^*b^+$

$L(e1) = \{a^m b^n \mid m \geqslant 0, n \geqslant 1\};$

$= \{b, ab, bb, aaab, aab, abb, aabb, \ldots\}$

(2) $e2 = (ab)^+$

$L(e2) = \{(ab)^n \mid n \geqslant 1\},$

$= \{ab, abab, ababab, abababab, \ldots\}$

(3)$e3 = ab^*c \mid b^*$

$L(e3) = \{ab^n c, \quad b^n \mid n \geqslant 0\}$

$= \{ac, abc, abbc, abbbc, \ldots; \varepsilon, b, bb, bbb, \ldots\}$

展开:

图 3.3: 正规语言示例

3.1.2 正规语言的有限自动机表示法

有限自动机（FA）是正规语言的描述方法之一，先来看三个示例。

$L1 = \{a^m b^n \mid m \geqslant 0, n \geqslant 1\}$,L1 对应的有限状态自动机为

FA2:

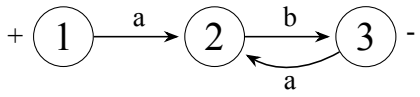


图 3.4: L1 对应的有限状态自动机

$L2 = \{(ab)^n \mid n \geqslant 1\}$,L2 对应的有限状态自动机为

FA2:

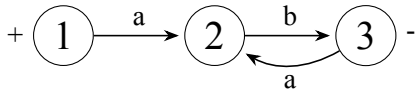


图 3.5: L2 对应的有限状态自动机

$L3 = \{ab^nc, b^n \mid n \geqslant 0\}$,L3 对应的有限状态自动机为

FA3:

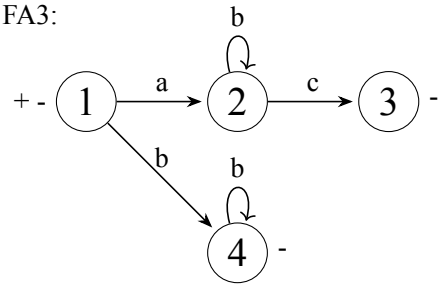


图 3.6: L3 对应的有限状态自动机

※ 初看起来，有限自动机是带标记的有向图！①②③④ 是图上的节点，称之为状态，带箭头的线可以理解为图上的边，这个边是有方向的，称之为有向边，+和-分别对应起始状态和结束状态
我们让大家有一个直观的印象，每一个正规语言都可以用一个有限状态自动机来描述。
※ 有限自动机表示法说明：
$L3 = \{ab^nc, b^n \mid n \geqslant 0\}$ 有限状态自动机由什么组成呢？

FA3:

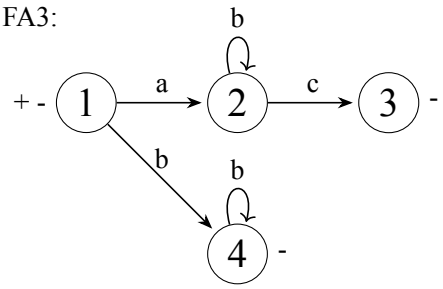


图 3.7: FA3 对应自动机

我们分析一下 FA3
【图符说明】：

{1,2,3,4}–状态集;

其中:+(开始状态);-(结束状态)

{a,b,c}–字母表;

$\delta(1,a)=2$–变换 (或 ① $\xrightarrow{a}$ ②);...

(表示 1 状态遇符号 a 变到 2 状态...);

图 3.8: 符号说明

①②③④ 是状态，状态表示的是事物在某一时刻的表示，+ 和-分别表示开始状态和结束状态，自动机的运行一定要从开始状态开始，最终到达结束状态，① 代表起始状态和结束状态可以是同一个状态。
字母表可以理解为文法中的字母表，自动机需要读入的最基本的元素，δ 表示状态转移，对应有向图中带箭头的边，$\delta(1,a)=2$ 表达状态 1 读入 a 后要跳转到状态 2。
【运行机制】
一个 FA，若存在一条从某开始状态 i 到某结束状态 j 的通路，且这条通路上所有边的标记连成的符号串为 α，则 α 就是一个句子；所有这样的 α 的集合，就是该 FA 表示的语言。
如果一个自动机的起始状态和终止状态写在了一起，则表示这个自动机是能接受空串的。
从自动机的应用上来说，基本的功能是判断一个符号串能否被一个文法所接收，换句话来说，若能找到一个路径恰巧对应这个符号串，则表示该符号串能被接收，如果找不到这样的路径，则表示这个符号串不能被自动机所接收。
※ 正规语言的三种表示法综合示例:

【例 3.5】$L=\{ab^nc,b^n\mid n\geqslant 0\},\square=\{a,b,c\}$

1. 正规文法描述:

$G(S):S\to aA|bB|\varepsilon,A\to bA|c,B\to bB|\varepsilon$

2. 正规式描述:$e=ab^*c\mid b^*$

3. 有限自动机描述:

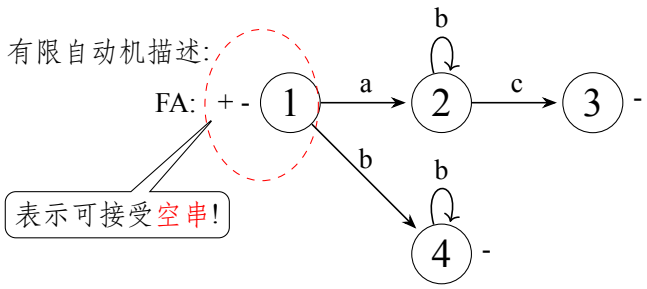


图 3.9: 综合示例

【注】凡是能由上述三种方法表示的语言，一定是正规语言；反之，凡是不能由上述三种方法表示的语言，一定不是正规语言。

3.2 有限自动机的定义与分类

3.2.1 有限自动机的定义

【定义】有限自动机是一种数学模型，用于描述正规语言，可定义为五元组:FA = (Q, Σ, S, F, δ)
上下文无关文法是几元组？四元组：非终结符集，终结符集，起始符号，产生式集合。有限自动机是五元组，
Q(有限状态集);
Σ(字母表);
S(开始状态集,$S \subseteq Q$);
F(结束状态集,$F \subseteq Q$);
δ: 变换 (二元函数):
状态 ①②③④⑤ 用状态 Q 来表示，S 是 Q 的子集, 变换 $\delta(i,a) = j$ 或 ⓘ $\xrightarrow{a}$ ⓙi, j ∈ Q, a ∈ $\sum + \{\varepsilon\}$ 表示状态 i 遇符号 a，变换为状态 j。空串在应用的角度来说不好，给计算机处理带来很大的麻烦，但是有限自动机是可以读入空串的。

3.2.2 有限自动机怎样描述语言

令 L(FA) 为自动机 FA 所描述的正规语言；则 $L(FA) = \left\{x \mid (i \overset{x}{\Rightarrow} j), x \in \Sigma^*, i \in S, j \in F\right\}$
一个有限状态自动机 FA 所对应的语言写作 L(FA)。从起始状态 i 到结束状态 j 的路径上所有符号所拼接起来构成的符号串便是该自动能接受的一个符号串，所有这样符号串构成的集合便是 FA 所对应的语言，记作 L(FA)。i $\overset{x}{\Rightarrow}$ j 表示从起始状态 i 到终止状态 j，其中每一次跳转分别读入的符号为 a1、a2 一直到 a_n，写成这样的形式设 $\mathbf{x} = \mathbf{a}_1\mathbf{a}_2 \ldots \mathbf{a}_n$，　$\mathbf{a}_i \in \Sigma + \{\varepsilon\}$ 则有 $i \overset{a_1}{\Rightarrow} i_1 \overset{a_2}{\Rightarrow} i_2 \ldots \overset{a_n}{\Rightarrow} j$, 每次读入的字符构成了一个符号串 $\mathbf{a}_1\mathbf{a}_2 \ldots \mathbf{a}_n$, 记为 x，则符号串 x 是 L（FA）中的一个句子，所有句子构成的集合就是该有限自动机对应的语言。
FA 存在一条从某初始状态 i 到某结束状态 j 的连续变换，且每次变换 (⇒) 的边标记连成的符号串为 x，则称 x 被 FA 接受，否则拒绝。
所以有限状态自动机最重要的一个功能是判断一个词串是否能被接收，词法分析便是利用有限状态自动机来识别单词串。

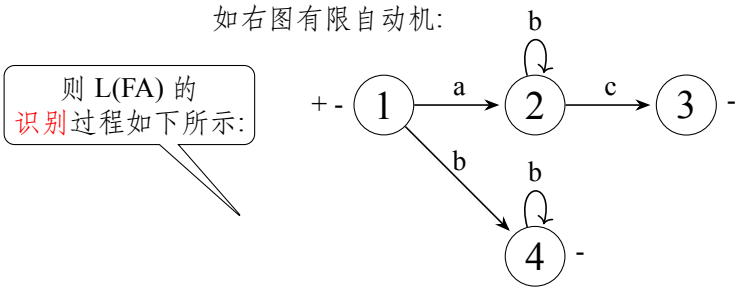


图 3.10: 有限自动机

⁎L(FA) 的生成 (或识别) 过程示例:

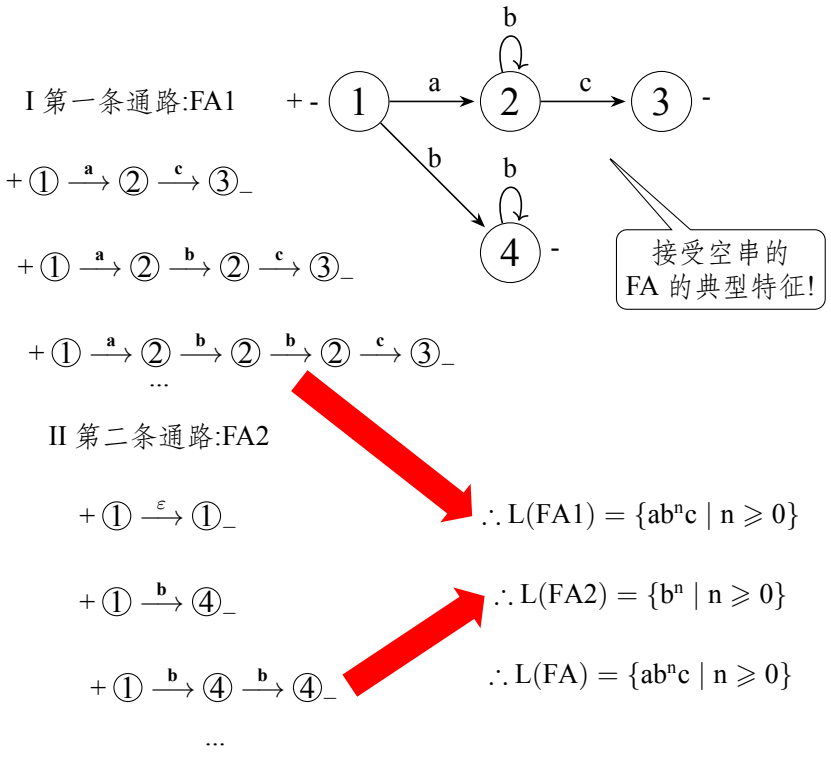


图 3.11: L(FA) 生成 (或识别) 过程示例

起始状态 ① 读入 a 跳转到 ② 号状态，② 号状态读入一个 c 到 ③ 号状态，③ 号状态是一个结束态，因此符号串 ac 是该语言的一个句子。
起始状态 ① 读入 a 跳转到 ② 号状态，② 号状态读入一个 b 仍然跳转到 ② 状态，② 状态读入 c 跳转到 ③ 号状态，③ 状态是个结束态，因此符号串 abc 是该语言的一个句子。
起始状态 ① 读入 a 跳转到 ② 号状态，② 号状态读入一个 b 仍然跳转到 ② 状态，② 号状态又读入一个 b 仍然跳转到 ② 状态，② 状态读入 c 跳转到 ③ 号状态，③ 状态是个结束态，因此符号串 abbc 是该语言的一个句子
综上，上半部分自动机 FA1 对应的语言可表示为 $\therefore L(FA1) = \{ab^n c \mid n \geqslant 0\}$
II. 第二条通路：FA2
起始状态 ① 直接读入空串就到达了结束态。
起始状态 ① 读入 b 到达结束态 ④。
起始状态 ① 读入 b 到达状态 ④，再读入 b 又到了 ④ 状态，这个时候是结束态。
以此类推，下半部分的自动机 FA2

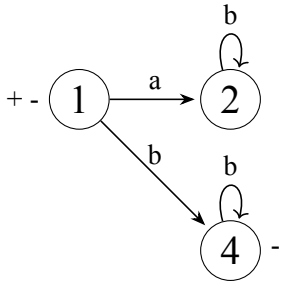


图 3.12: FA 自动机

表达的是 $\therefore L(FA2) = \{b^n \mid n \geqslant 0\}$
因此整个自动机所对应的语言就是 FA1 和 FA2 对应的语言合并在一起，即 $\therefore L(FA) = \{ab^n c \mid n \geqslant 0\}$

3.2.3 有限自动机的两种表现形式

【例 3.6】有限自动机:FA = (Q, Σ, S, F, δ)
其中:Q=1,2,3,4,Σa,b,c,S=1,2,F=3,4
δ:$\delta(1,a)=2;\delta(1,b)=4;\delta(2,b)=2;$
$\delta(2,c)=3;\delta(2,\varepsilon)=3;\varepsilon(4,b)=4;$
我们可以把有限自动机写成状态图或变换表的模式
FA 的两种表现形式:

(1) 状态图:

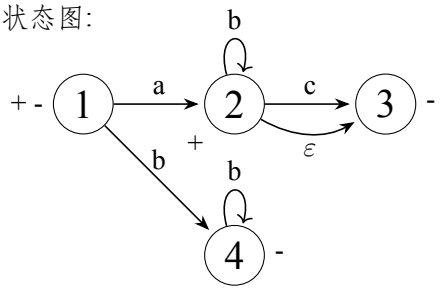


图 3.13: FA 状态图

这个图并不容易让计算机去识别

(2) 变换表:

开始状态
结束状态

	a	b	c	ε	
1	2	4			+
2		2	3	3	+
3					-
4		4			-

图 3.14: FA 变换表

这是计算机存储中最常用的方式，可以把状态转移看成一个表的形式，表中每一行对应一个状态，所以这个表中写了四个状态，每一个列分别对应一个字母，行和列的交叉点表示对应的状态读入相应的字母时应跳转到的状态。

	a
1	2

图 3.15: 状态 1

表达 ① 状态读入 a 跳转到 ② 状态，即 $\delta(1,a)=2$
起始状态和结束状态分别用 “+” 和 “-” 表示。※ 有限自动机的构造示例 1

【例 3.7】$A=\{ab^nc,(ab)^n \mid n \geqslant 0\}$FA 的构造:

方法一: 联合式

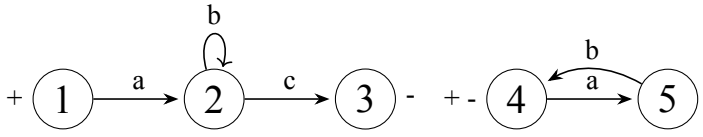


图 3.16: 方法一

对于符号串 ab^nc 可构造出这样一个自动机，开始状态 ① 读入 a 进入 ② 状态，使用循环方式表示 ② 状态可读入无限个 b,② 状态若读入 c 进入 ③ 状态。同样道理第二个串 $(ab)^n$ 可以写成这样的自动机，开始状态 4 读入 a 进入 5 状态，5 状态读入 b 进入结束态 4。注意 4 状态既是起始态又是结束态，表示该自动机可接收一个空串。

方法二:组合式 1

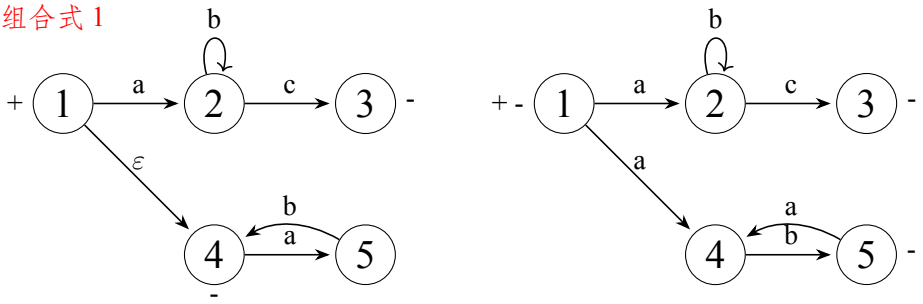


图 3.17: 方法二

④ 状态是一个起始状态，但是在方法二中 ① 状态通过跳转到了 ④ 状态，所以 ④ 状态可以不标注起始状态。
消除 ① 状态到 ④ 状态的 ε 跳转得到

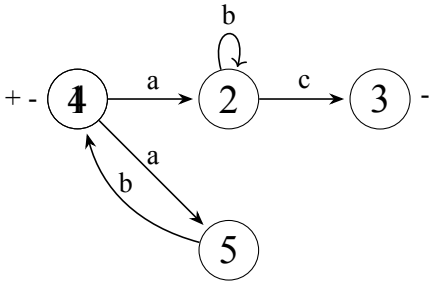


图 3.18: 跳转结果

该自动机与联合式或组合式的自动机是否等价？答案：不等价
正确的自动机如图所示

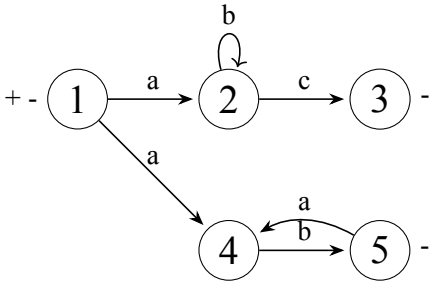
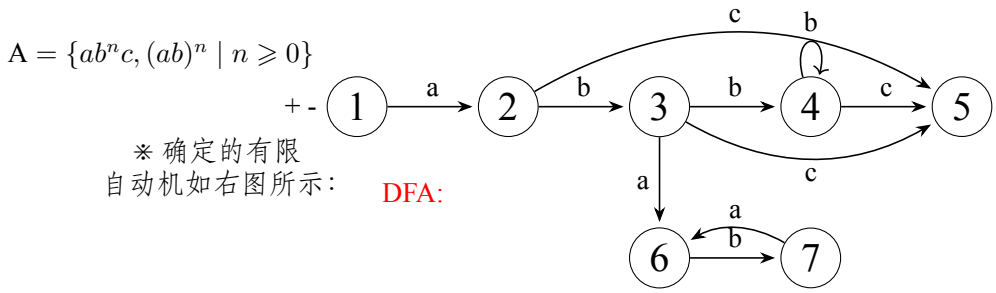


图 3.19: 正确结果

接下来比较各种自动机表示方法的优劣：方法一：开始状态不唯一，不好用！方法二：带有 ε 边，还是不好用！另外，变换函数不单值，如 $\delta(1,a)=(2|4)$ 也不好用！

方法三:确定式

$A = \{ab^n c, (ab)^n \mid n \geqslant 0\}$



※ 确定的有限自动机如右图所示：

DFA:

图 3.20: 方法三

状态唯一，不带有 ε 边，变换函数单值的有限自动机。

最后我们再举一个现在中的例子：奇偶校验

例 3.1 奇偶校验

奇校验

Input：由 0，1 组成的字符串

Output：奇数个 1，则输出 ok，否则拒绝

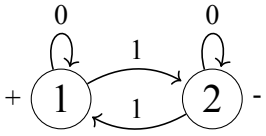


图 3.21: 奇校验

类似的，偶校验的自动机如下：

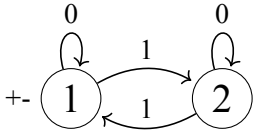


图 3.22: 偶校验

3.2.4 有限自动机的分类

1. 确定的有限自动机（DFA）

特征：①开始状态唯一；②变换函数单值；③不带 ε 边。

2. 非确定的自动机（NFA）– 不能全部具备上述特征者！

(a) 带有 ε 边的非确定的有限自动机，记为 εNFA；

(b) 不带有 ε 边的非确定的有限自动机，记为 $\bar{\varepsilon}$NFA；

※ 有限自动机的分类示例

例 3.2 试分别指出下述有限自动机的分类情况

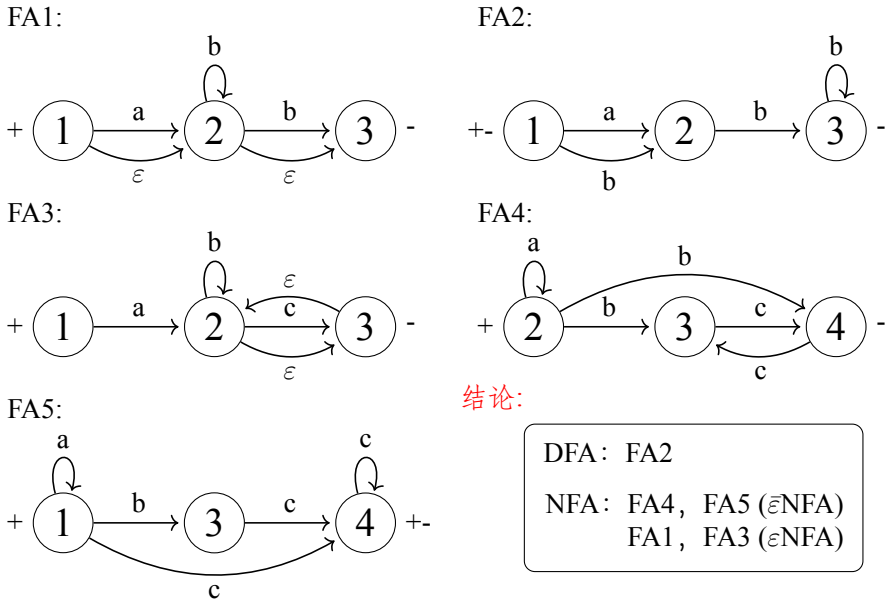


图 3.23: 有限自动机的分类情况

上述五个有限状态自动机中哪些是确定型的？FA2 为 DFA。

FA4 变换函数不单值，FA5 起始状态不唯一，FA1 和 FA3 带有 ε 边。

3.3 有限自动机的等价变换

能不能找到确定型的有限状态自动机，从理论上已经证明任何有限状态自动机都可以转化为确定型的有限状态自动机。有限自动机的等价变换, 是指把 FA1 变换为 FA2, 且有 L(FA1) = L(FA2)。主要包含以下两个内容:

1. 有限自动机的确定化（NFA => DFA）;

找到一个确定型有限状态自动机，同时其接收的语言和原始的自动机接收的语言是一摸一样的，这个过程叫确定化。非确定机（NFA）较易构造，但不易于使用。相反，确定机（DFA）较难构造，但使用上更加方便。

※ 任何一非确定机 NFA, 皆可通过有效算法把其转换为等价的确定自动机 DFA。

2. 有限自动机的化简（最小化）;

找到一个自动机和原自动机是等价的，但其状态数是最少的。

※ 对任何一确定机 DFA1, 皆可通过有效算法将其转换为等价的确定机 DFA2，且 DFA2 的状态最少。

3.3.1 有限自动机的确定化

第一步先把一个含 ε 边的非确定有限状态自动机转化为不含 ε 边的非确定有限状态自动机，第二步把不含 ε 边的非确定有限状态自动机确定化。

1. 消除 ε 边算法（εNFA $\Rightarrow$ $\bar{\varepsilon}$NFA 或 DFA）

1. 若存在 ε 闭路，则把 ε 闭路上各节点合而为一:

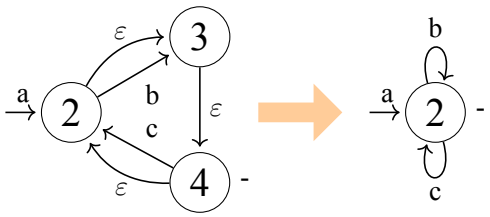


图 3.24: ε 闭路上各节点合而为一

2. 标记隐含的开始态和结束态：

开始态的 ε 通路上的节点：+

结束态逆向 ε 通路上节点：-

从起始状态开始找到所能到达的 ε 通路上的所有节点，将其标注为起始状态。比如起始状态①，通过 ε 跳转到达的任何节点都标注为起始状态。

从结束态逆向标注，如果一个节点通过 ε 跳转能到达结束态，则认为这个节点是结束态逆向通路上的节点，将其标记为结束态。

例如：

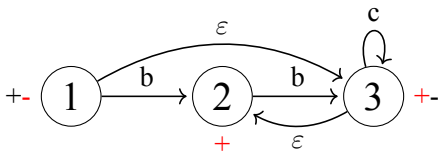


图 3.25: 标记隐含的开始态和结束态

起始状态①通过 ε 跳转到③状态，③状态通过 ε 跳转到②状态，因此③状态和②状态都是起始状态 ε 通路上的节点，将状态②和状态③标记为起始状态。同样道理，③状态是结束态，①状态通过 ε 跳转可以到达③状态，因此将①状态标记为结束状态。

3. 对剩余的 ε 边，逆向逐一进行：

①删除一个 ε 边；同时

②引进新边：凡由原 ε 边终点出发的边，也要由其始点发出。

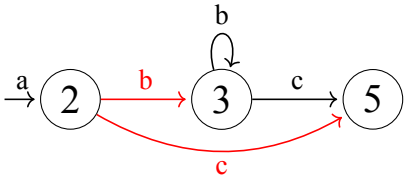


图 3.26: 删除 ε 边，同时引进新边

4. 重复步骤 (3)，直到再无 ε 边为止。例如：

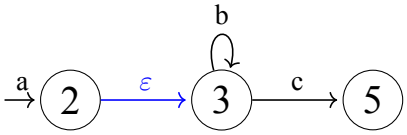


图 3.27: 重复步骤 (3)，直到再无 ε 边

删除状态②到状态③的 ε 边，③状态能到达③状态和⑤状态，所以②状态读入 b 也得到达③状态，②状态读入 c 也能到达⑤状态。

例 3.3 消除 ε 边算法示例：

已知 εNFA 如下，求 L(εNFA) =?

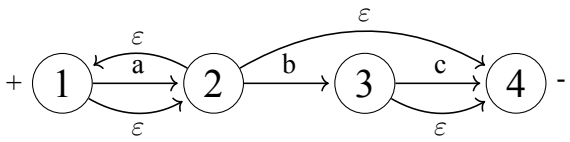


图 3.28: 消除 ε 边算法示例

1. ε 闭路上的节点等价，(① ≡ ②)，可合二为一；得到

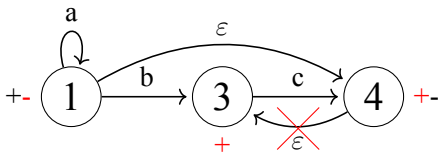


图 3.29: 节点等价

2. 标记隐含的开始态和结束态：+④，+③，-①

从开始状态找 ε 通路，把通路上的节点都标为开始状态，①状态读入 ε 到达④状态，④状态读入 ε 到达③状态。从结束态开始，找 ε 逆向通路上的节点，将其标记为结束态，①状态通过 ε 跳转到达结束态④状态，所以将①状态标记为结束态。

3. 逆序逐一删除 ε 边，同时引进新边：

先删除④状态到③状态的 ε 边

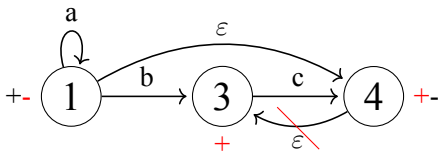


图 3.30: 删除④到③的 ε 边

然后删除①到④的 ε 边

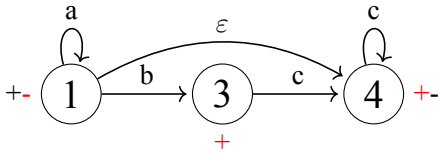


图 3.31: ①到④的 ε 边

③能到达④，那④就能到达④（即③能读入 c 到④，那④就能读入 c 到④）

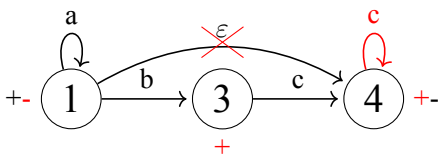


图 3.32: ④读入 c 到④

删除状态①到状态④的 ε 边，□ 状态能到达④状态，所以①状态读入 c 也能到达④状态

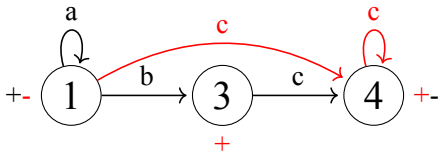


图 3.33: εNFA

$\therefore L(\varepsilon NFA) = \{a^m, a^m bc^n, a^m c^n \mid m \geqslant 0, n \geqslant 1\}$

2.εNFA 的确定化算法（εNFA ⇒ DFA）

1. 构造 DFA 的变换表（框架）：

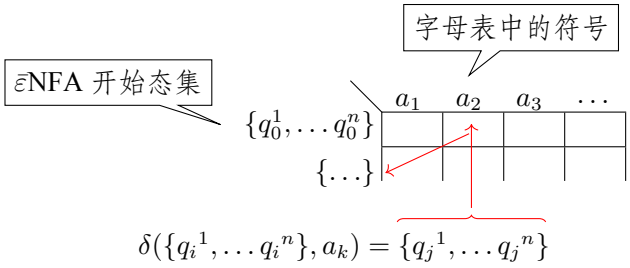


图 3.34: DFA 变换表

2. 按 εNFA 的变换函数实施变换：$\delta(\{q_i{}^1, \ldots q_i{}^n\}, a_k) = \{q_j{}^1, \ldots q_j{}^n\}$
3. 若 $\{q_i{}^1, \ldots q_i{}^n\}$ 未作位状态进行标记，则作新行标记；
4. 重复步骤 (2)(3)，直到不再出现新状态集为止；
5. 标记 DFA 的开始态和结束态：

- 第一行 $\{q_i{}^1, \ldots q_i{}^n\}$，（右侧）标记 +；
- 凡是状态行中含有 εNFA 的结束状态者，（右侧）标记-。

【注】必要时，新产生的 DFA 可用状态图表示。

自动机所有的起始状态构成一个集合 $\{q_i{}^1, \ldots q_i{}^n\}$，该集合作为确定自动机 DFA 的起始状态。DFA 变换表的行由原始自动机中状态或状态集合表示 $\{q_i{}^1, \ldots q_i{}^n\}$，列是字母表上的符号。从 DFA 起始状态集合开始，把集合中任意一个状态读入相应的字母所到达的状态合成一个集合（到达的状态集合），作为 DFA 变换表中相应的表项，注意该表项可能是一个单一状态，也可能是一个状态集合。只要表项中状态或状态集合没有出现在 DFA 变换表的行标记中，则将其作为一个新的行标记，直到没有新的行标记出现为止。经过上述处理后，便可得到原始自动机对应的确定化有限自动机，这个确定化有限自动机的状态可能是是原始自动机中的状态或状态的集合。最后，标注开始态和结束态，新自动机的开始态只有一个，即 $\{q_i{}^1, \ldots q_i{}^n\}$；如果新自动机中行标记包含原始自动机的结束态，就将其都标记为结束态。

⁎εNFA 确定化示例

例 3.4 联合自动机 NFA：

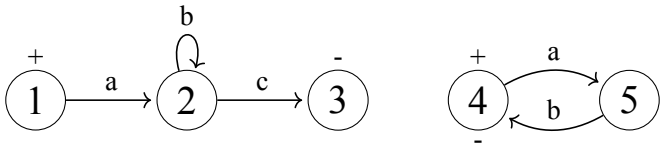


图 3.35: 联合自动机 NFA

不是确定型的自动机

	a	b	c
+ - A{1,4}	B{2,5}		
B{2,5}		C{2,4}	D{3}
- C{2,4}	E{5}	F{2}	D{3}
- D{3}			
E{5}		G{4}	
F{2}		F{2}	D{3}
- G{4}	E{5}		

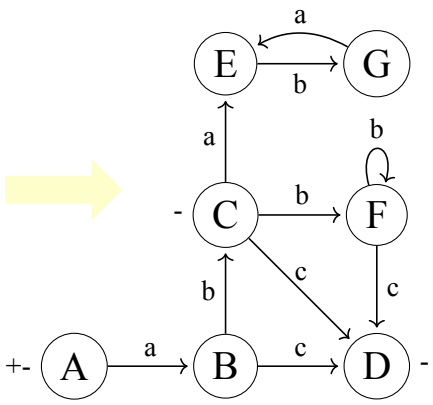


图 3.36: 确定化过程

【注】A,B,C,…状态集的代码

表列是字母表，行对应的是状态，原始自动机的起始状态①④作为确定自动机的起始状态，读入 a 到达②⑤

B{2,5} 读入 b 到达 C{2,4} 读入 c 到达 D{3}；

C{2,4} 读入 a 到达 E{5} 读入 b 到达 F{2} 读入 c 到达 D{3}；

E{5} 读入 b 到达 G{4}；

F{2} 读入 b 到 F{2} 读入 c 到 D{3}；

G{4} 读入 a 到 E{5}；

只要新自动机中行标记包含原始自动机的结束态，就将其都标记为结束态。

根据确定自动机的变换表，便可构造出该确定自动机对应的状态图，不难看出，这两个自动机是等价的。

3.3.2 有限自动机的最小化

有限自动机的最小化, 又称有限自动机的化简，是指对给定的确定机 DFA1, 构造另一个确定机 DFA2, 使得 L(DFA1) = L(DFA2), 且 DFA2 的状态最少。

有限自动机最小化算法，是指构造满足下述条件的确定有限自动机 (称为最小机)：

1. 删除无用状态；
2. 合并等价状态。

第一步是删除无用状态

定义 3.1 无用状态是指由开始态达不到的状态 (不可达) 或者由其出发不能到达结束态的状态 (不终结)。

无用状态是指两种状态，一种是不可达的状态，一种是不终结的状态。这两种状态是不会参与到自动机的运行里的，参与了也没有用，所以这两种状态是不需要的，可以删除掉。第二步是合并等价状态。

如果两个状态是等价的，这两个状态是可以在最小化的操作中被合并的。那什么是等价状态呢？

定义 3.2 等价状态是指这样的两个状态，若分别将其看作开始态，二者接收的符号串集合相同。

下面我们来看一个例子

例 3.5 无用状态

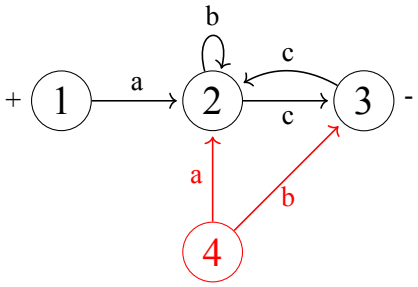


图 3.37: 不可达状态

状态④只有出边没有入边，所以该状态是一个不可达状态。

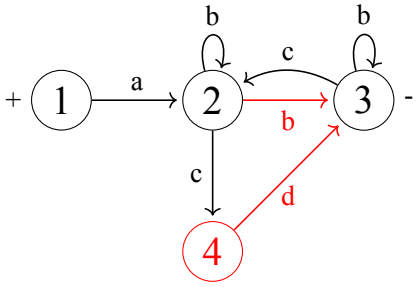


图 3.38: 不终结状态

状态③只有入边，没有出边，所以该状态是一个不终结状态。

不管是不可达也好，还是不终结也好，这两个状态都不是自动机需要的状态。

我们再来看下一个例子:

例 3.6 合并等价节点

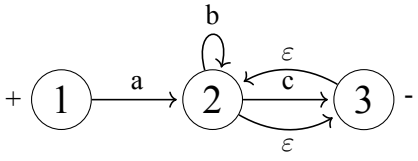


图 3.39: 合并等价节点

状态②和状态③这两个状态可以通过读入空串来互相跳转，②通过读入空串跳转到③，③通过读入空串跳转到②。之前说过，这是一个带空边的闭环，所以在这个闭环上的所有节点都是等价的，因此可以把它们合二为一。

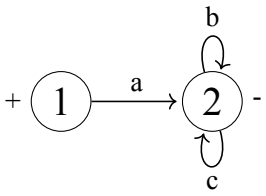


图 3.40: 保留非空跳转

可以看到，合并之后的②状态可以通过读 b 跳转到②，可以通过读 c 跳转到②，这是因为②③合并之后，②和③上的非空跳转仍然需要保留。

接下来看下一个例子，如何去判断等价性。上面例子中的带空边闭环是比较好判断的。

例 3.7 如何判断等价性：

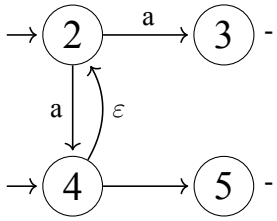


图 3.41: 判断等价性

首先判断④和⑤是否是等价的，我们可以从定义出发：

等价状态是指这样的两个状态,若分别将其看作开始态,二者接收的符号串集合相同。如果同样把④和⑤看做开始态，可以看到④状态可以接收空边到②，②再接收 a 到④，④还可以接收 b 到结束态⑤，即④作为开始态可以接收符号串 ab，但是⑤作为开始态并不可以接收符号串 ab。所以④和⑤是不等价的。

第二个问题：③和⑤是否是等价的？

状态③和⑤都是终止态，这两个状态都不能接收任何符号串，所以状态③和状态⑤是等价的。

第三个问题，②和③是否是等价？

显然②状态能够读入 a 到③状态，而③状态不能接收任何符号串，所以这两个状态是不等价的。

最后一个问题，②和④是否是等价？

④状态能够接收 b 到结束态⑤，而②状态不能接收 b，所以这两个状态也不是等价的。

【算法】

接下来我们看一下如何去执行这两步：删除无用状态以及合并等价状态。首先看删除无用状态的算法。删除无用状态我们在这给了两个算法，第一个是删除不可达状态。

※【删除不可达状态】构造可达状态集 Q_{AR}

1. 设 q_0 为开始态，则令 $q_0 \in Q_{AR}$；
2. 若 $q_i \in Q_{AR}$ 且有 $\delta(q_i, a) = q_j$ 则令 $q_j \in Q_{AR}$；
3. 重复执行 (2)，直到 Q_{AR} 不再增大为止。
4. 从状态集 Q 中，删除不在 Q_{AR} 中的所有状态。

首先要理解什么是不可达状态？就是无法到达的状态。那么为什么不可达呢？这个不太好描述，我们可

以反过来想想，什么是可达的状态。可达状态的定义非常简单，从起始状态开始，只要能找到一条路径到达一个状态，那么这个状态就是可达状态。从这个角度说，是否可以找出一个可达状态的集合呢？这个算法就告诉我们怎么去构造可达状态集合。

首先从起始状态开始，不断的进行搜索，将所有可达的状态都加入到集合当中，最后当这个集合不再增加的时候，就得到了可达状态集合，从状态集中把这些状态删除，就得到了不可达状态的集合。

同样的道理，不终结的状态怎么判断呢？类似的，不好判断一个状态是否是不终结状态，我们可以去想想什么节点能够终结？那就是什么节点能够到达终止状态，如果可以找出所有能够到达终止状态的节点，那么其余状态便是不终结状态。

※【删除不终结状态】构造可终结状态集 Q_{FN}

1. 设 q_i 为结束态，则令 $q_i \in Q_{FN}$；
2. 若 $q_j \in Q_{FN}$ 且有 $\delta(q_i, a) = q_j$ 则令 $q_i \in Q_{FN}$；
3. 重复执行 (2)，直到 Q_{FN} 不再增大为止。
4. 从状态集 Q 中，删除不在 Q_{FN} 中的所有状态。

接下来看一下如何判断等价状态。首先，两个状态 i,j 等价，当且仅当满足下面两个条件：

1. 必须同是结束态，或同不是结束态；
2. 对所有字母表上符号，状态 i,j 必变换到等价状态。

第一个条件，两个状态必须同是结束态，或者同不是结束态，如果这两个状态里有一个是结束态，有一个不是结束态，显然结束态的那个节点可以读空直接停止了，而非结束态的不一定。

第二个条件，对于字母表上的所有符号，两个状态分别读入同一个字符跳转到的两个状态都应该是等价状态。例如，i 状态读入 a 跳转到 I’,j 状态读入 a 跳转到 J’，如果 I’ 和 J’ 等价，那么 i,j 也是等价的。

例 3.8 判断等价状态：

把下述自动机最小化

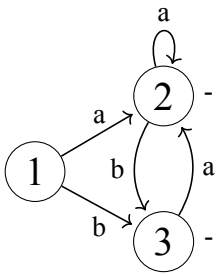


图 3.42: 判断等价状态

1. 初分成两个不等价子集：$Q_1 = \{1, 2\}$，$Q_2 = \{3\}$

将状态分为两个不等价子集，根据等价条件的第一个条件，因为①，②状态都是结束态，而③状态不是结束态，可以划分成两个不等价子集 1，2 和 3。这一步只能说①，②有可能是等价，但是①跟 3 或者②跟③肯定不是等价的。

接下来第二步，根据等价条件的第二个条件，如果两个状态读入相同的字符，都能跳转到等价状态，那么这两个状态是等价的。状态①，②同时读入 a，b 都能跳转到同一个状态，因此①，②等价。

2. 还能分成不等价子集吗?

$\because \delta(\{1, 2\}, a) = 2$

又 $\delta(\{1,2\},\mathrm{b})=3$
$\therefore 1\equiv 2$

所以可以将①，②状态合并在一起，得到了下面的自动机

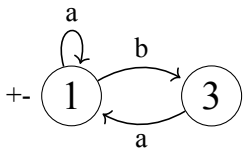


图 3.43: 将①，②状态合并

我们可以将上述步骤总结成一个算法：

【合并等价状态算法】– 划分不等价状态集

1. 初始，把状态集 Q 化分成两个不等价子集：

Q_1（结束状态集），Q_2（非结束状态集）；

2. 把每个 Q_i 再划分成不同的子集，条件是：

同一 Q_i 中两个状态 i，j，若对字母表中的某个符号，变换到已划分的不同的状态集中，则 i，j 应分离，如：$\delta(\mathrm{i},\mathrm{a})\in Q_m,\delta(\mathrm{j},\mathrm{a})\in Q_n$ 且 $m\neq n$

3. 重复步骤 (2)，直到再不能划分为止；

4. 合并最终划分的每个子集中的状态（合而为一）。

例 3.9 ※ 有限自动机化简示例：

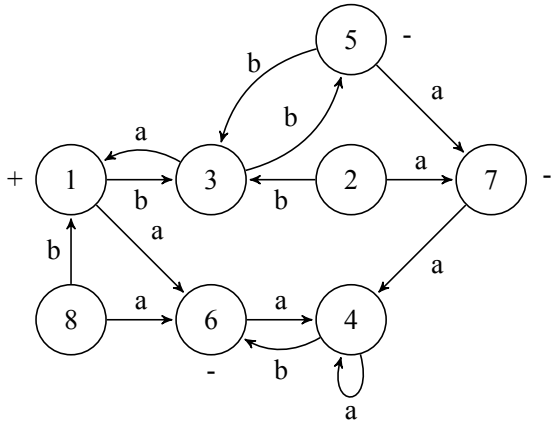


图 3.44: DFA

第一步：删除无用状态

动态构造 DFA 变换表，即从开始状态 1 出发，把变换后的状态填入表项，并同时作为新行标记；如此下去，直到不再出现新状态为止。未出现的状态，就是无用的状态。

		a	b
+	1	6	3
	3	1	5
-	6	4	
-	5	7	3
	4	4	6
-	7	4	

图 3.45: 构造 DFA 变换表

例3.9对应的 DFA 变换表如上图所示，该表中状态便是所有的可达状态，状态 2 和 8 不在里面，所以状态 2 和 8 是不可达状态。在这里我们分析的是不可达状态，不终结状态可以采用类似的方法来分析，最终删除的不可达和不终结状态是 2 状态和 8 状态。

第二步：合并等价状态

		a	b
+	1	6	3
	3	1	5
-	6	4	
-	5	7	3
	4	4	6
-	7	4	

图 3.46: DFA 变换表

1. 先将原始的状态划分为两个状态集，一个是结束状态集，另一个是非结束状态集。从变换表可以看出来结束态是 5，6，7，所以把 5，6，7 划分为一个状态集，剩余的 1，3，4 是非结束状态集。

令 $Q_{NE=1,3,4,5,6,7}$

2. 第二步逐一去访问当前等价状态集合中的任意两个状态，如果它们读入同一个字符后跳转到的状态不在同一个等价状态集合里，则把它们分裂到不同的等价集中。

取 {3,4}：

$$\because \delta(1,a)=6, \delta(3,4,a)=\{1,4\}$$

$$\therefore \text{划分成} Q_1=\{1\}, Q_2=\{3,4\}$$

$$\text{即} Q_{NE}=\{\{1\},\{3,4\},\{5,6,7\}\}$$

首先看 1，3，4 状态集，由于状态 1 遇到 a 跳转到状态 6，3，4 遇到 a 跳转到状态 1,4，其中状态 6 和 1，4 在不同的等价集合中，即 6 在 5，6，7 中，3，4 在 1，3，4 中，所以把 1 和 3，4 分裂成不同的等价状态集。在这一步，我们可知状态 1 和状态 3，状态 4 肯定不等价，但是状态 3，状态 4 是否等价我们还不知道，还得继续去执行算法。

3. 下一步我们看 3，4，来确认状态 3 和状态 4 是否等价。

由于状态 3 遇到 a 变成状态 1，状态 4 遇到 a 变成状态 4，由于状态 1, 状态 4 在不同的等价划分里，所以状态 3 和状态 4 肯定不是等价状态。因此，得到了新的等价集合 3 和 4。

4. 第四步继续对 5，6，7 执行算法。

取 {5,6,7}：同理，可划分成 $Q_1=\{5\}, Q_2=\{6,7\}$

最后：$Q_{NE}=\{\{1\},\{3\},\{4\},\{5\},\{6,7\}\}$

需要注意的是，划分等价状态集算法适用于不含空边的自动机。若自动机带有空边，可以先用之前学过的算法消空边，接下来再执行划分等价状态集算法。

5. 最后得到了状态 6 和状态 7 是等价状态，因此只需要将 6，7 合并，如何合并呢？只需要将 6 替换 7，得到的结果如下所示：

		a	b
+	1	6	3
	3	1	5
-	6	4	
-	5	6	3
	4	4	6

图 3.47: 合并等价状态

6. 最后将转化表变为自动机。

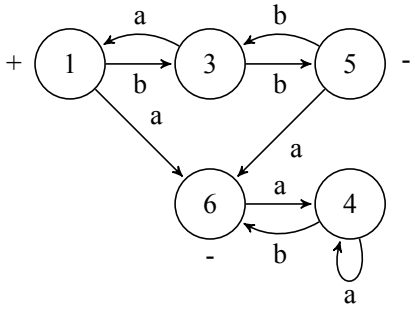


图 3.48: 最小的 DFA

3.4 正规语言描述方法间的相互转换

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

任何一个表示方法都能转化为另外两个，我们先看看正规文法和确定有限状态自动机之间的转换方法。

设 $G(Z)=(V_N,V_T,Z,P)$, $DFA=(Q,\sum,s,F,\delta)$ 正规文法是一个四元组，确定有限状态自动机是一个五元组，可以发现二者之间是有对应关系的，如下图所示：

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

图 3.49: 正规文法与 DFA 之间的对应关系

例 3.10 自动机 ⇒ 正规文法：

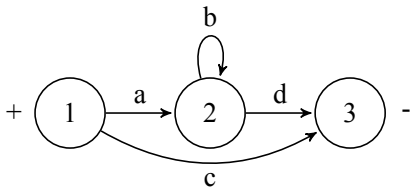


图 3.50: DFA

令 Z=①，A=②,B=③，则有正规文法

$$G(Z): Z \to aA \mid cB, A \to bA \mid dB, B \to \varepsilon$$

根据上面的对应关系，可以基于自动机写出正规文法。

相反，我们也能够将正规文法转为自动机：

例 3.11 正规文法 ⇒ 自动机，并求 L(G)：

文法规则

$$G(Z): Z \to aZ \mid bA \mid \varepsilon, A \to bA \mid d$$

首先第一步，$A \to d$，根据上面的对应规则，我们先做一点点变换：

将 $A \to d$ 变换为 $A \to dB, B \to \varepsilon$。

这一步的目的是什么？是希望让 B 作为终止态，显性体现在自动机里，经过上述变换后得到一个新的等价文法，唯一的区别是新文法中引入了一个非终结符 B。

$$\therefore G'(Z) \text{与 } G(Z) \text{ 等价：}$$

$$Z \to aZ \mid bA \mid \varepsilon, A \to bA \mid dB, B \to \varepsilon$$

之后令状态①表示 Z，状态②表示 A，状态③表示 B，画出自动机。

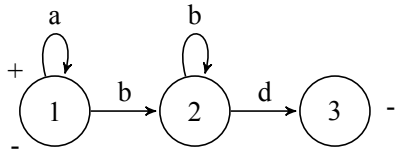


图 3.51: DFA

这个文法对应的语言为：

$$L(G) = \{\varepsilon, a^m b^n d \mid m \geqslant 0, n > 0\}$$

接下来，我们来看一下正规式和有限自动机状态机的转换。

转换的机制如下：

设 e 为正规式，$DFA = (Q, \sum, s, F, \delta)$，转换机制：

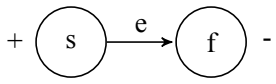


图 3.52: 转换机制

对于正规式 e，可以看作是一个符号串，我们希望有一个自动机，能够从初始状态读入这个符号串，到达终止状态，这就完成了转化的过程。

定义 3.3 符号串转换为 DFA 的过程称为分解，DFA 转换为符号串的过程称为合成。

以下是转换的规则，图中正方向称为分解过程，负方向称为合成过程：

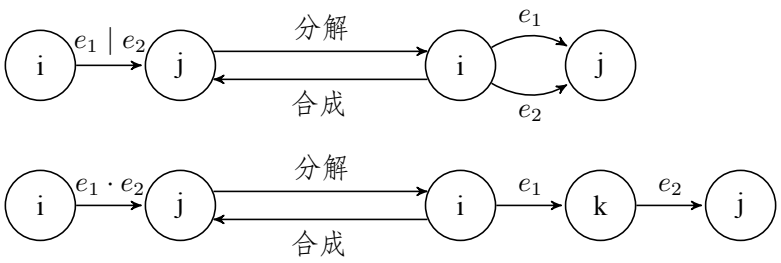


图 3.53: 转换规则

而对于闭包型正规式，有如下 5 种转换方式：

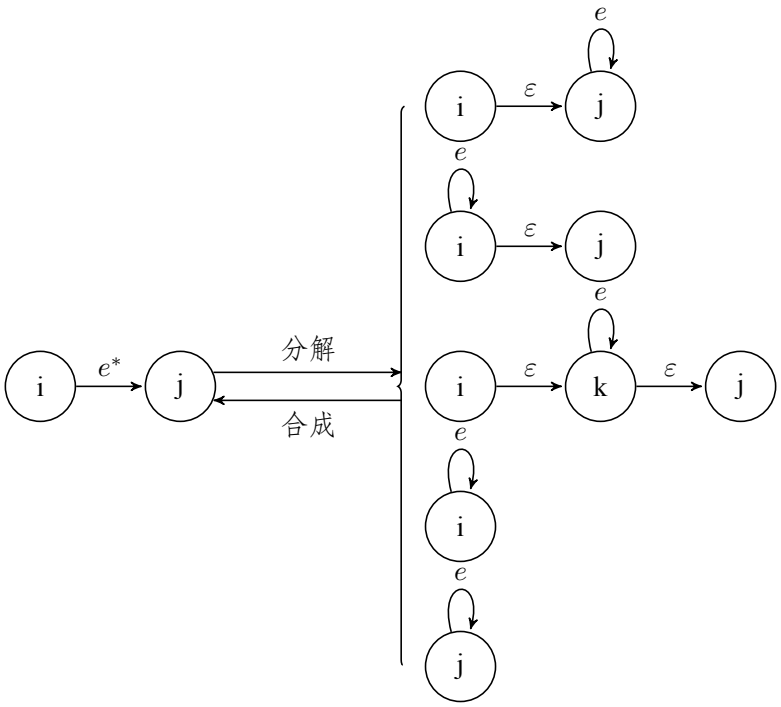


图 3.54: 闭包型正规式转换规则

例 3.12 正规式 ⇒ 自动机：

设 $e = a^*b \mid bc^*$

首先根据上述的转化规则转化为一个初始的自动机：

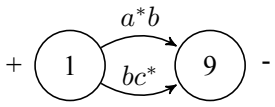


图 3.55: 初始自动机

经过一次分解后自动机变为：

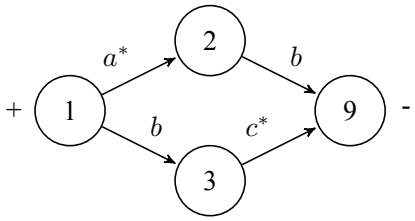


图 3.56: 一次分解后自动机

对闭包型正规式分解后变为：

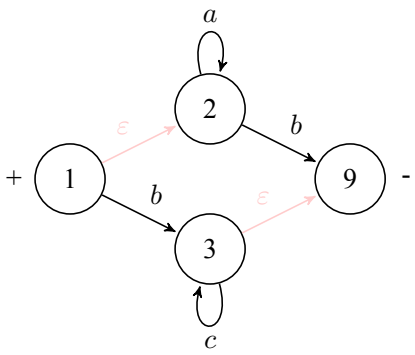


图 3.57: 完全分解后自动机

之后需要对这个带空边的自动机进行确定化，怎么进行确定化呢？先标记起始状态和终止状态，从起始状态开始，经过空边到达的所有节点都标记为起始状态，同样，所有通过空边到达终止状态的节点都标记为终止状态。然后逆向消除空边，第一次消除空边后变成这样：

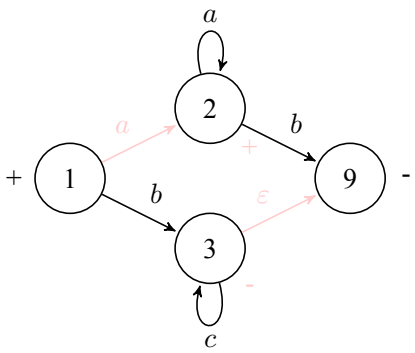


图 3.58: 消除一条空边的自动机

消除第二个空边变成下面这样：

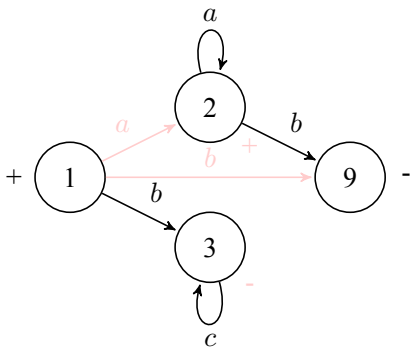


图 3.59: 消除第二条空边的自动机

由于有①②两个起始状态，我们需要进一步进行确定化，用自动机确定化算法，构造出如下表所示的确定自动机的变换表。

		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}

-

图 3.60: 自动机变换表

最终得到了确定化后的自动机：

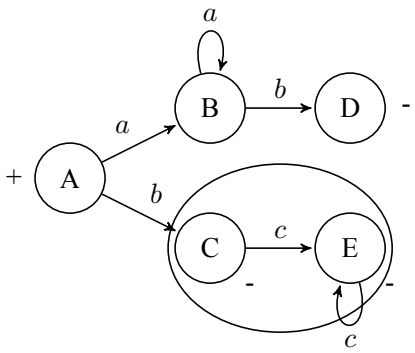


图 3.61: 确定化后的状态机

由变换表3.60可知，C、E 状态等价，最小化后的状态机如下图所示：

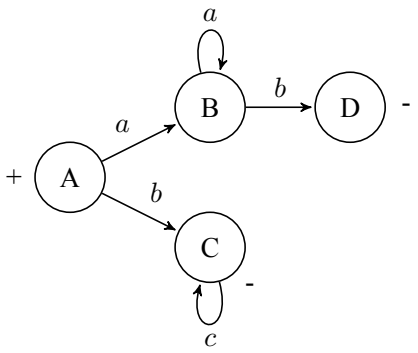


图 3.62: 最小化后的自动机

3.5 有限状态自动机的实现问题

在实际应用时，怎么去设计一个自动机呢？自动机的设计有以下两个说明:

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)。

自动机其实就是这么一个装置，对任意的一个字符串，能接受就是 yes，不能就是 no。

2. 为了便于处理，可令 ∀ 作为待识别的符号串的泛指后继符。

我们将这个符号作为符号串的后继符，这个后继符怎么去理解呢？可以理解成把一个字符串的结尾单独用一个符号来表示。

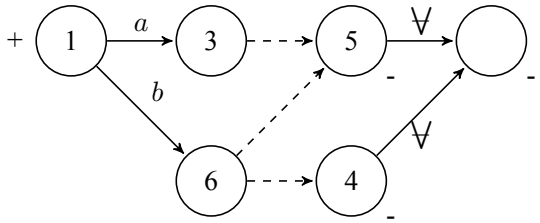


图 3.63: 扩展后的自动机

扩展之后的自动机如上图所示，只有唯一的开始态和唯一的结束态。当然我们可以将自动机表示成一个变换表，其中 no 代表不能接受的后继字符。

		a	b	...	∀
+	1	3	6	...	no
	...	...	...	...	...
-	4	no	no	...	ok
-	5	no	no	...	ok

图 3.64: 自动机变换表

3.5.1 控制程序设计

基于上面的说明，给出一个自动机的控制程序流程图:

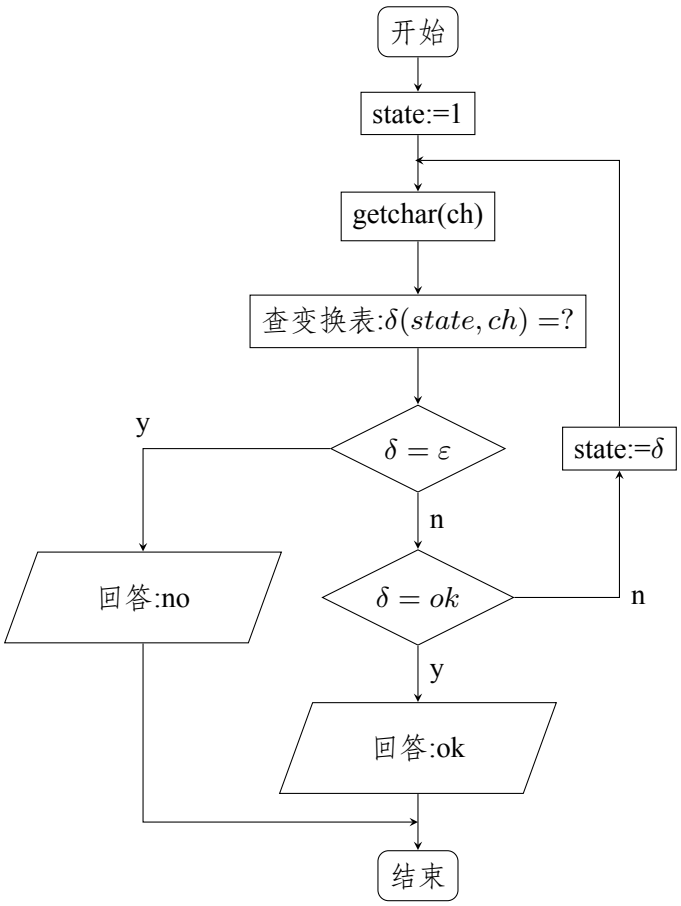


图 3.65: 自动机控制程序流程图

3.5.2 变换表存贮结构设计

下面将解决状态转移表即变换表存储的问题，可以有很多存储状态转移表的方式:

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,…。

二维数组的方式优点是简洁、访问快，提供下标就能直接访问。但是缺点是太占空间。

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,如下图所示,图中 ∀（其他）代表错误符号，左侧蓝色索引表中的序号代表数字，右侧红色表为变换子表:

我们一般用第二种方法，索引表的形式。

例 3.13 ※ 有限自动机实现示例:

有限自动机 DFA:

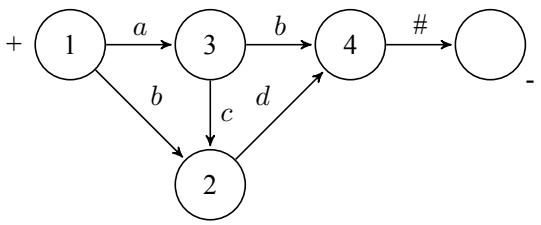


图 3.67: DFA

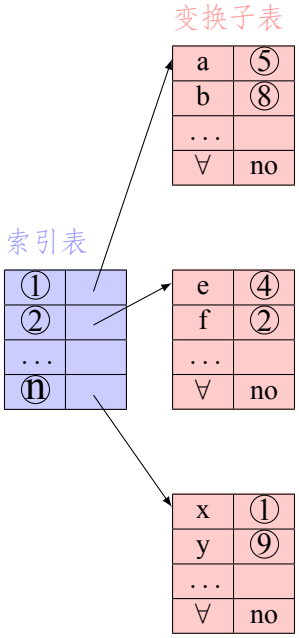


图 3.66: 压缩变换表

根据这个有限状态自动机，我们可以用上述的压缩变换表的形式存储自动机状态转移表。

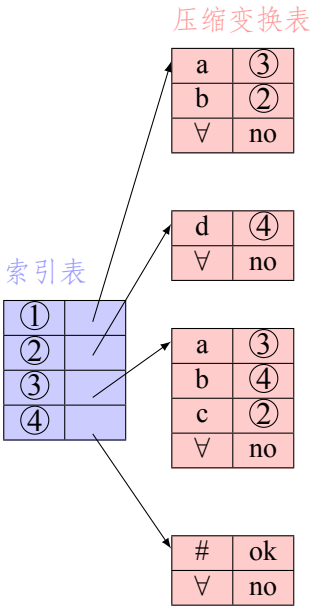


图 3.68: 自动机状态转移表

最后，模拟一下应用压缩变换表的自动机的识别字符串的过程：

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

图 3.69: 字符串识别过程