编译原理

第三章　词法分析

编译程序首先是在单词级别上来分析和翻译源程序的。词法分析的任务是：从左至右逐个字符地对源程序进行扫描，产生一个个单词符号，把作为字符串的源程序改造成为单词符号串的中间程序。

因此，词法分析是编译的基础。执行词法分析的程序称为**词法分析器**。

3.1对词法分析器的要求

词法分析器功能：输入源程序，输出单词符号。

程序语言的单词符号一般分为五种：关键字，标识符，常数，运算符，界符

词类和属性

1. 关键字(保留字或基本字)：while, if
2. 标识符：用来表示各种名字
3. 字面常数：256，3 .14，true, ‘abc’
4. 运算符：如，＋、－、*、/ 等等
5. 分界符：如逗号，分号，冒号等

词法分析器输出的单词符号常常表示为二元式：
（单词种别，单词符号的属性值）

词类编码原则：

界符和运算符：一符一码。

关键字可分成一类，也可以一个关键字分成一类。一字一码。

常数可统归一类，也可按类型（整型、实型、布尔型等），每个类型的常数划分成一类。一类型一码。

所有的标识符分为一类。一类一码。

对于关键字、界符、运算符来说，它们的词类编码就可以表示其完整的信息，

故对于这类单词，其单词自身的属性值通常为空

而对于标识符，词类编码所反映的信息不够充分，标识符的具体特性还要通过单词自身的属性进行互相区分。

标识符的单词自身的属性常用其在符号表中的入口指针来表示

对于常数， 其单词自身的属性常用其在常数表中的入口指针来表示

作为例子考虑下述C++代码段：

```
while (i>=j) i- -;
```

经词法分析器处理后，它将转换的单词符号序列：

< while, - >

< (, - >

< id ,指向i的符号表项的指针 >

< >= , - >

< id ,指向j的符号表项的指针>

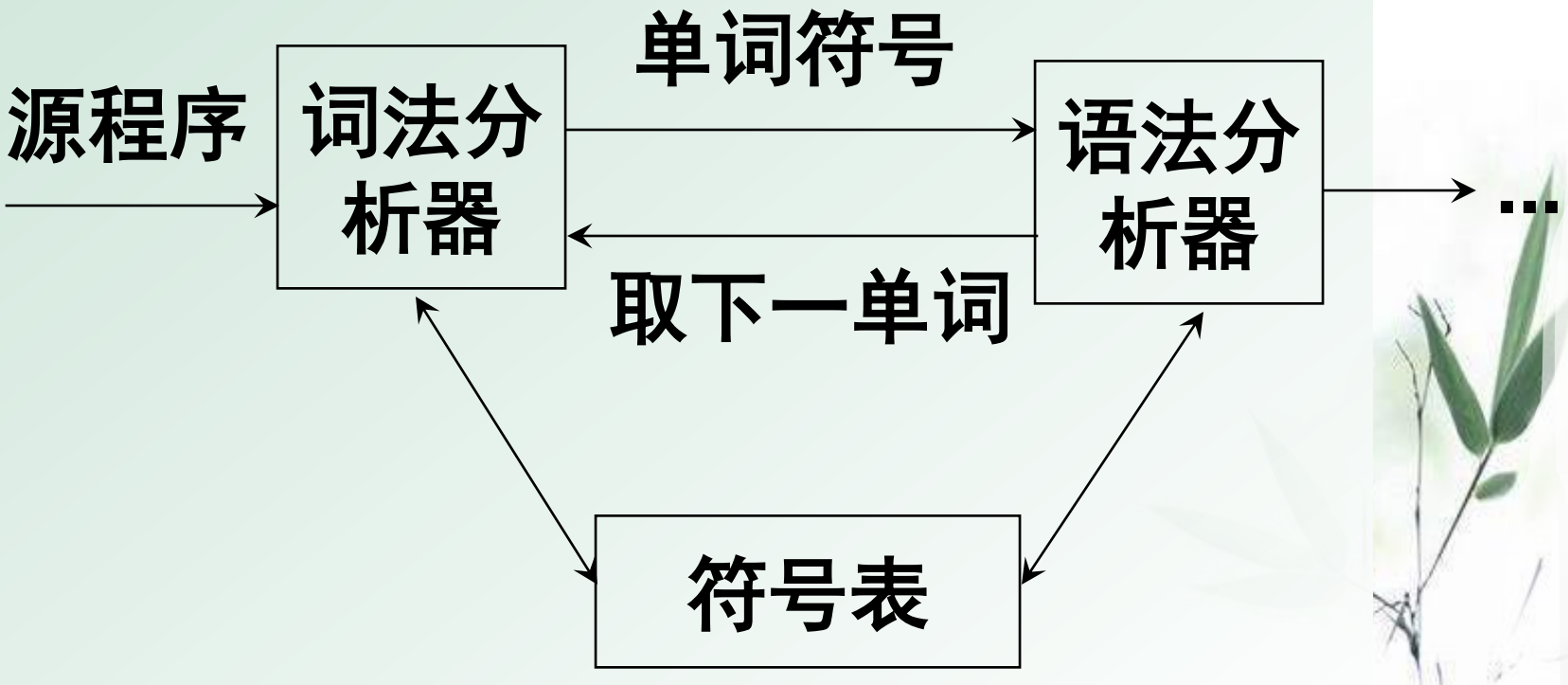
<) , - >

< id ,指向i的符号表项的指针 >

< - - , - >

< ; , - >

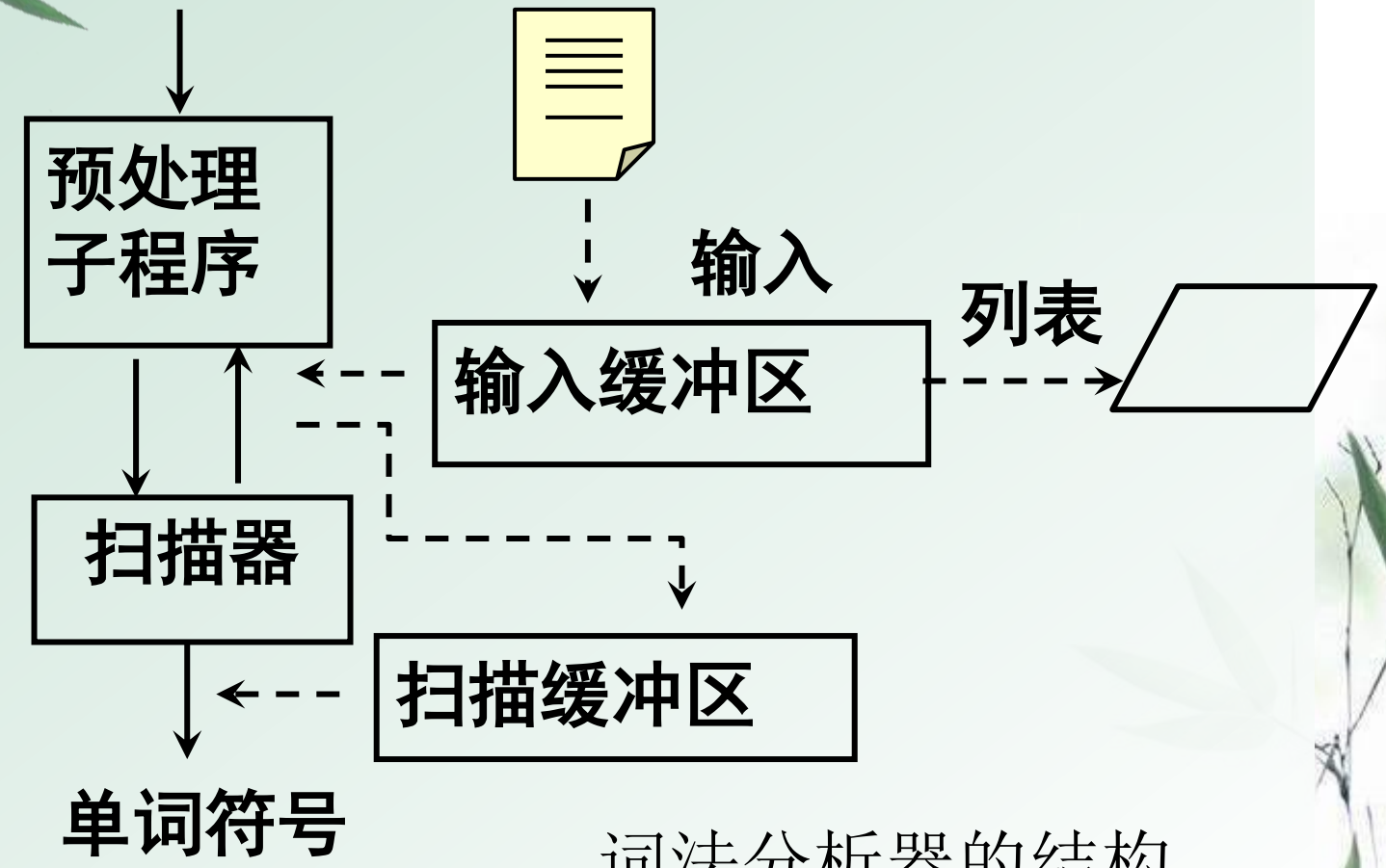
词法分析器



词法分析器作为独立子程序

我们把词法分析器安排成一个独立子程序，每当语法分析器需要一个单词符号时就调用这个子程序。每一次调用，词法分析器就从符号串中识别出一个单词符号，把它交给语法分析器。这样，把词法分析器安排成一个子程序似乎比较自然。在后面的讨论中，我们假定词法分析器是按这种方式进行工作的。

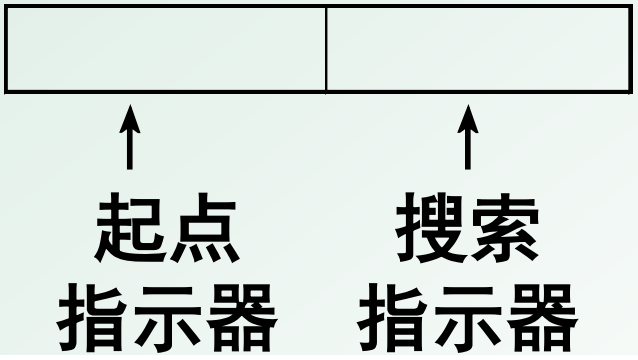
3.2 词法分析器的设计



词法分析器的结构

3.2.1 输入、预处理

- 输入串放在输入缓冲区中。
- 预处理子程序：剔除无用的空白、跳格、回车和换行等编辑性字符;区分标号区、捻接续行和给出句末符等
- 扫描缓冲区



3.2.2 单词符号的识别:超前搜索

1 关键字识别:

例如:

DO99K=1，10　　DO 99 K = 1，10

IF(5.EQ.M)GOTO55　IF (5.EQ.M) GOTO 55

DO99K=1.10

IF(5)=55

- 需要超前搜索才能确定哪些是关键字

2 **标识符**识别:

- 字母开头的字母数字串，后跟界符或算符

3 **常数**识别:

- 识别出算术常数并将其转变为二进制内码表示。有些也要超前搜索。

  5.EQ.M

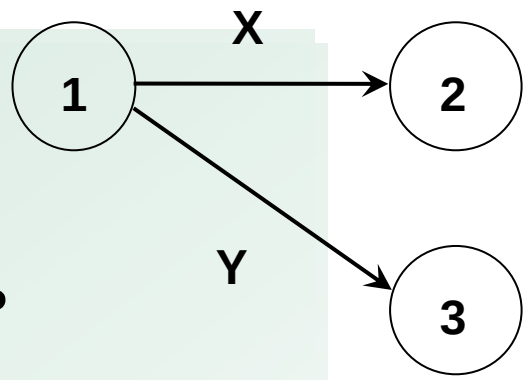
  5.E08

4 **算符和界符**的识别

- 把多个字符符合而成的算符和界符拼合成一个单一单词符号。

  :=，　**，　.EQ.　，　++，--，>=

3.2.3 状态转换图

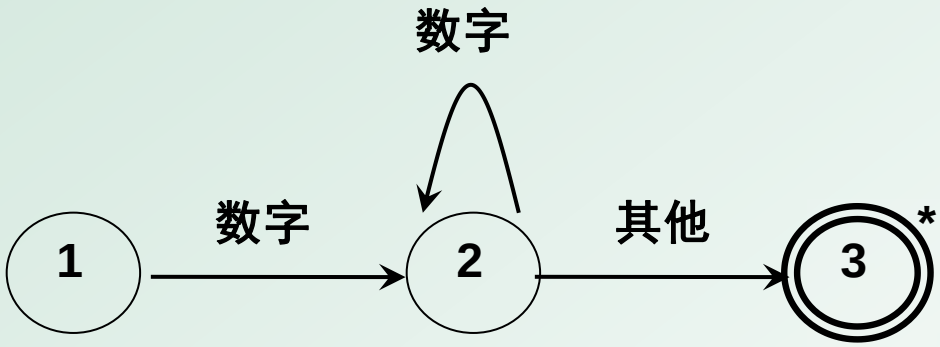


1 概念

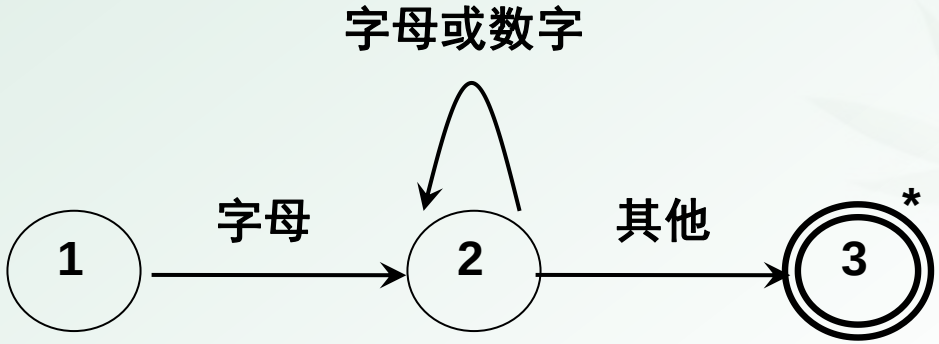
- 状态转换图是一张有限方向图。
  - 结点代表状态，用圆圈表示。
  - 状态之间用箭弧连结，箭弧上的标记(字符)代表射出结状态下可能出现的输入字符或字符类。
  - 一张转换图只包含有限个状态，其中有一个为初态，至少要有一个终态。

- 一个状态转换图可用于识别(或接受)一定的字符串。

- 终态结上打星号表示多读进一个不属于标识符的字符，应退回给输入串。



识别整常数的状态转换图



识别标识符的状态转换图

- 几点重要限制——不必使用超前搜索

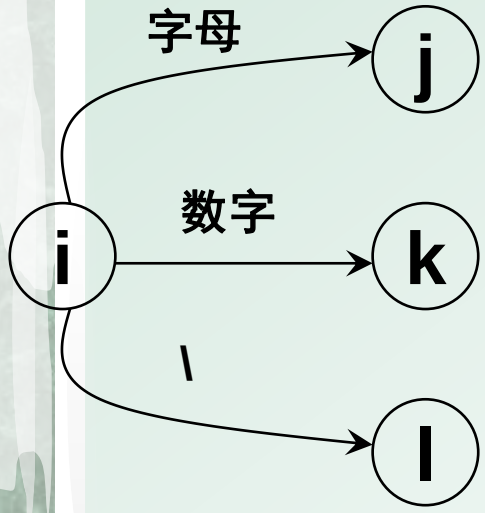
  - 所有基本字都是保留字;用户不能用它们作自己的标识符
  - 基本字作为特殊的标识符来处理;不用特殊的状态图来识别，只要查保留字表。
  - 如果基本字、标识符和常数(或标号)之间没有确定的运算符或界符作间隔，则必须使用一个空白符作间隔。

    DO99K=1，10
    要写成　　DO 99 K=1，10

3.2.4 状态转换图的实现

- 思想：每个状态结对应一小段程序。
- 做法：

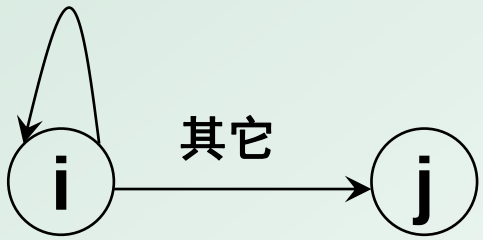
  1）对不含回路的分叉结，可用一个CASE语句或一组IF-THEN-ELSE语句实现




```
GetChar( );
if (IsLetter( )) {…状态j的对应程序段…;}
else if (IsDigit( )) {…状态k的对应程序段…;}
else if (ch=‘/’) {…状态l的对应程序段…;}
else {…错误处理…;}
```

2)对含回路的状态结，可对应一段由WHILE结构和IF语句构成的程序.

字母或数字




```
GetChar( );
while (IsLetter( ) or IsDigit( ))
      GetChar( );
…状态j的对应程序段…
```

3)终态结表示识别出某种单词符号，因此，对应语句为

RETURN (C, VAL)

其中，C为单词种别，VAL为单词自身值.

3.3 正规表达式与有限自动机

- 几个概念:
  - 考虑一个有穷 字母表Σ 字符集
  - 其中每一个元素称为一个字符
  - Σ上的字(也叫字符串) 是指由Σ中的字符所构成的一个有穷序列
  - 不包含任何字符的序列称为空字，记为 ε
  - 用Σ*表示Σ上的所有字的全体,包含空字 ε

例如: 设 Σ={a, b}，则

Σ*={ ε , a, b, aa, ab, ba, bb, aaa, . . . }

- Σ*的子集U和V的连接（积）定义为

$$UV=\{\ \alpha\beta \mid \alpha\in U\ \&\ \beta\in V\ \}$$

V自身的 n次积记为

$$V^n=VV\cdots V$$

- 规定$V^0=\{\varepsilon\}$，令

$$V^*=V^0\cup V^1\cup V^2\cup V^3\cup\cdots$$

称V^*是V的闭包；

- 记 $V^+=VV^*$ ，称V^+是V的正规闭包。

3.3.1 正规式和正规集

- 正规集可以用正规表达式（简称正规式）表示。
- 正规表达式是表示正规集一种方法。一个字集合是正规集当且仅当它能用正规式表示。

- **正规式和正规集的递归定义：**

  对给定的字母表Σ

  1）ε 和∅都是Σ上的正规式，它们所表示的正规集为{ε}和∅；

  2）任何a∈Σ，a是Σ上的正规式，它所表示的正规集为{a}；

3）假定e_1和e_2都是Σ上的正规式，它们所表示的正规集为$L(e_1)$和$L(e_2)$，则

i） $(e_1|e_2)$为正规式，它所表示的正规集为$L(e_1)\cup L(e_2)$，

ii） $(e_1.e_2)$为正规式，它所表示的正规集为$L(e_1)L(e_2)$，

iii） $(e_1)^*$为正规式，它所表示的正规集为$(L(e_1))^*$，

仅由有限次使用上述三步骤而定义的表达式才是Σ上的正规式，仅由这些正规式表示的字集才是Σ上的正规集。

- 所有词法结构一般都可以用正规式描述。
- 若两个正规式所表示的正规集相同，则称这两个正规式等价。如

$b(ab)^*=(ba)^*b$　　$(a^*b^*)^*=(a|b)^*$

$L(b(ab)^*)$
$= L(b)L((ab)^*)$
$= L(b)\ (L(ab))^*$
$= L(b)\ (L(a)L(b))^*$
$=\{b\}\ \{ab\}^*$
$= \{b\}\ \{\varepsilon,\ ab,\ abab,\ ababab,\ \cdots\}$
$= \{b,\ bab,\ babab,\ bababab,\ \cdots\}$

$L(\ (ba)^*b)$
$= L((ba)^*)\ L(b)$
$= (L(ba))^*L(b)$
$= (L(b)L(a))^*\ L(b)$
$= \{ba\}^*\ \{b\}$
$= \{\varepsilon,\ ba,\ baba,\ bababa,\ \cdots\}\ \{b\}$
$= \{b,\ bab,\ babab,\ bababab,\ \cdots\}$

$\because\ L(b(ab)^*)=L(\ (ba)^*b)$　　$\therefore b(ab)^*=(ba)^*b$

- 对正规式，下列等价成立：
  - $e_1|e_2 = e_2|e_1$ 交换律
  - $e_1|(e_2|e_3) = (e_1|e_2)|e_3$ 结合律
  - $e_1(e_2e_3) = (e_1e_2)e_3$ 结合律
  - $e_1(e_2|e_3) = e_1e_2|e_1e_3$ 分配律
  - $(e_2|e_3)e_1 = e_2e_1|e_3\ e_1$ 分配律
  - $e\varepsilon = \varepsilon\ e = e$

$L(e_1|e_2)$

$= L(e_1) \cup L(e_2)$

$= L(e_2) \cup L(e_1)$

$= L(e_2|e_1)$

3.3.2 有限自动机

不确定的有限自动机（简称NFA）

一个数学模型，它包括:

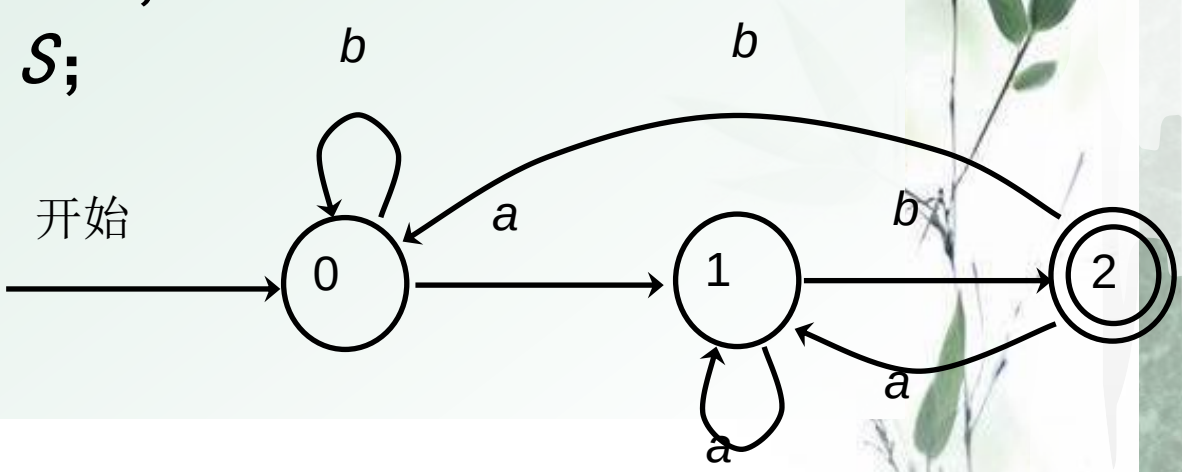
- 状态集合S;
- 输入符号集合Σ;
- 转换函数$move$: $S \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(S)$;
- 状态s_0是开始状态;
- $F \subseteq S$是接受状态集合。

确定的有限自动机（简称DFA）

一个数学模型，包括：

- 状态集合S;
- 输入字母表Σ;
- 转换函数$move : S \times \Sigma \rightarrow S$;
- 唯一的初态 $s \in S$;
- 终态集合$F \subseteq S$;

识别语言
$(a|b)^*ab$
的DFA



- 例如：DFA M=({0, 1, 2, 3}, {a, b}, f, 0, {3}), 其中：f定义如下：

f(0, a)=1　　f(0, b)=2

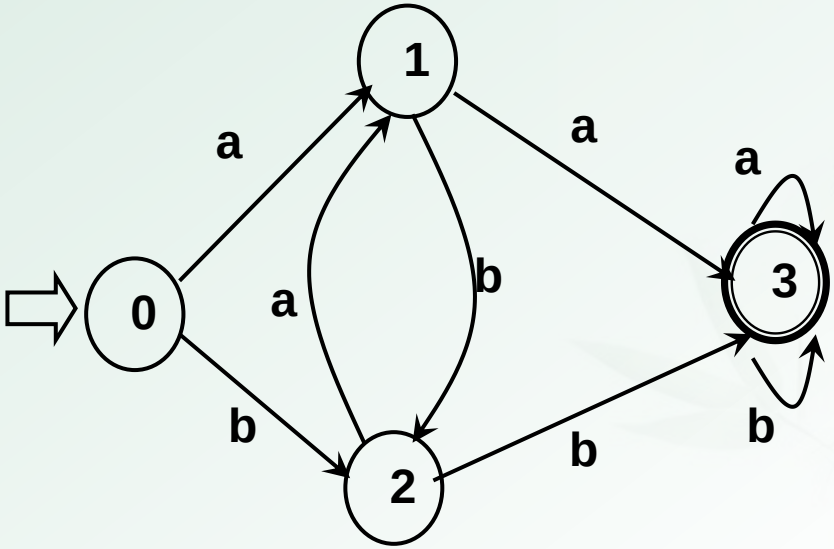
f(1, a)=3　　f(1, b)=2

f(2, a)=1　　f(2, b)=3

f(3, a)=3　　f(3, b)=3

	a	b
0	1	2
1	3	2
2	1	3
3	3	3

状态转换矩阵



状态转换图

- DFA可以表示为状态转换图。假定DFA M含有m个状态和n个输入字符，那么，这个图含有m个状态结点，每个结点顶多含有n条箭弧射出，且每条箭弧用Σ上的不同的输入字符来作标记。

- 对于Σ^*中的任何字α，若存在一条从初态到某一终态的道路，且这条路上所有弧上的标记符连接成的字等于α，则称α为DFA M所识别(接收)
- DFA M所识别的字的全体记为L(M)。
- 可以证明：Σ上的字集$V \subseteq \Sigma^*$是正规集，当且仅当存在上的DFA M，使得$V=L(M)$。

- 从状态图中看NFA 和DFA的区别：

  1 弧上的标记可以是Σ^*中的一个字，而不一定是单个字符；

  2 同一个字可能出现在同状态射出的多条弧上。

- DFA是NFA的特例。

- 定义：对于任何两个有限自动机M和M’，如果L(M)=L(M’)，则称M与M’等价。
- 自动机理论中一个重要的结论：判定两个自动机等价性的算法是存在的。
- 对于每个NFA M存在一个DFA M’，使得L(M)=L(M’)。亦即DFA与NFA描述能力相同。

证明:

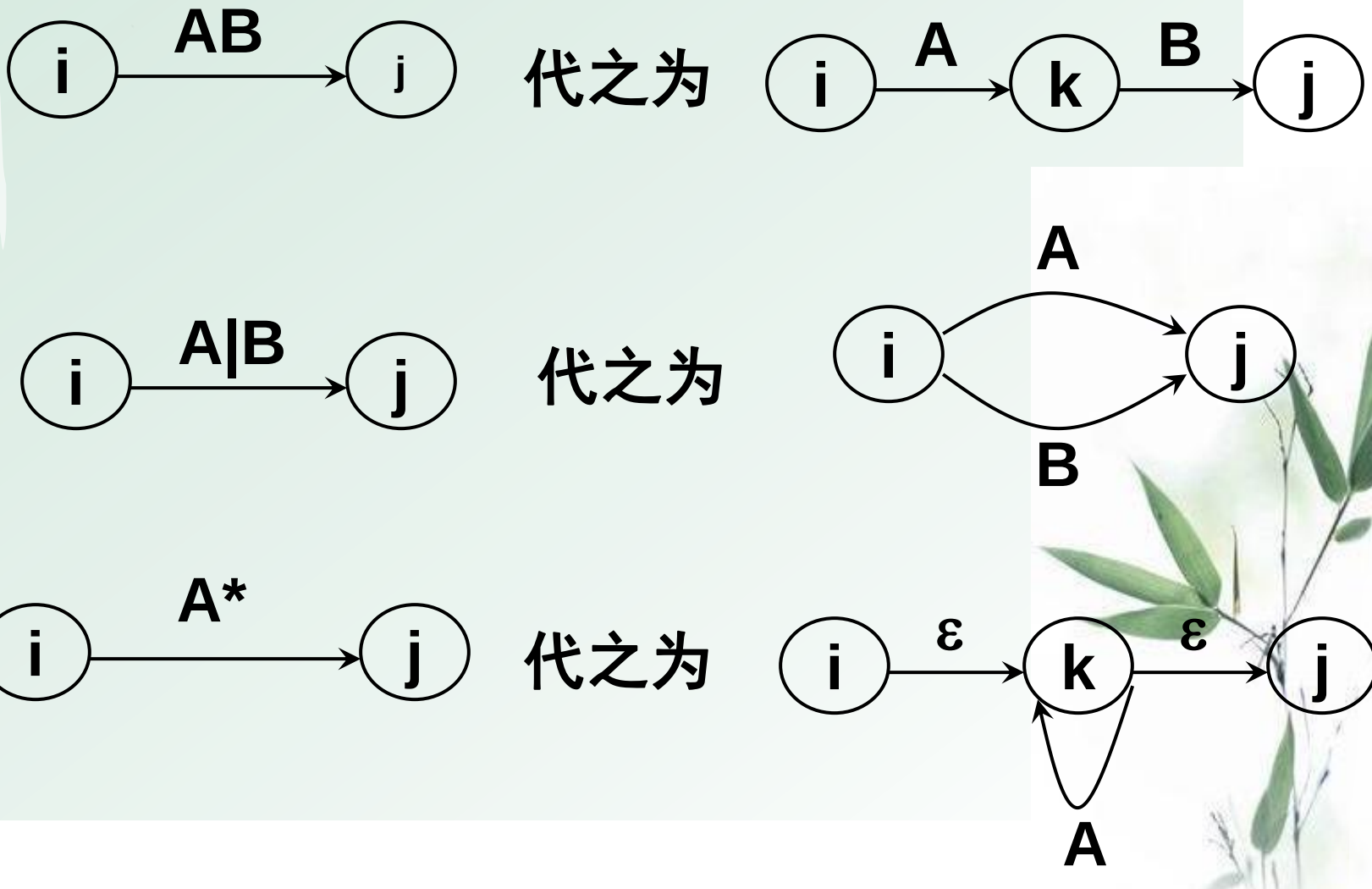
1. 假定NFA $M=\langle S, \Sigma, \delta, S_0, F\rangle$，我们对M的状态转换图进行以下改造：

   1）引进新的初态结点X和终态结点Y，$X, Y \notin S$，

   从X到S_0中任意状态结点连一条ε箭弧，从F中任意状态结点连一条ε箭弧到Y。

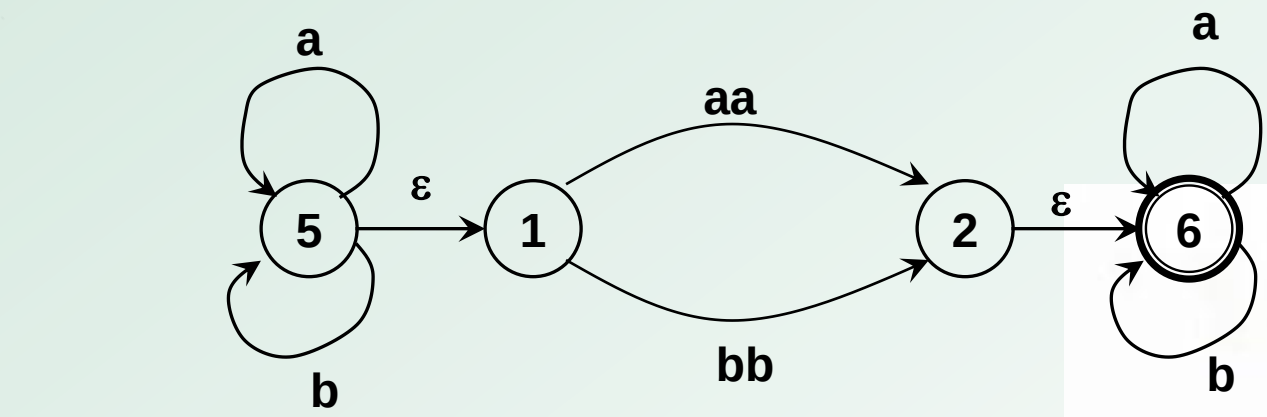
   2）对M的状态转换图进一步施行替换，其中k是新引入的状态。

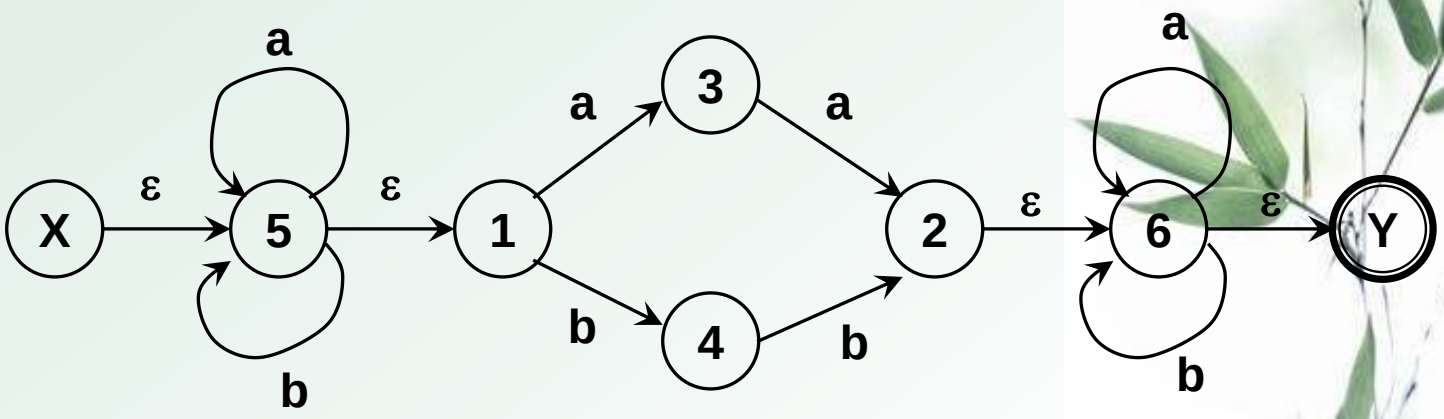
按下面的三条规则对V进行分裂：



- 逐步把这个图转变为每条弧只标记为Σ上的一个字符或ε，最后得到一个NFA M’，显然L(M’)=L(M)

- 识别所有含相继两个a或相继两个b的字





NFA变DFA：确定化

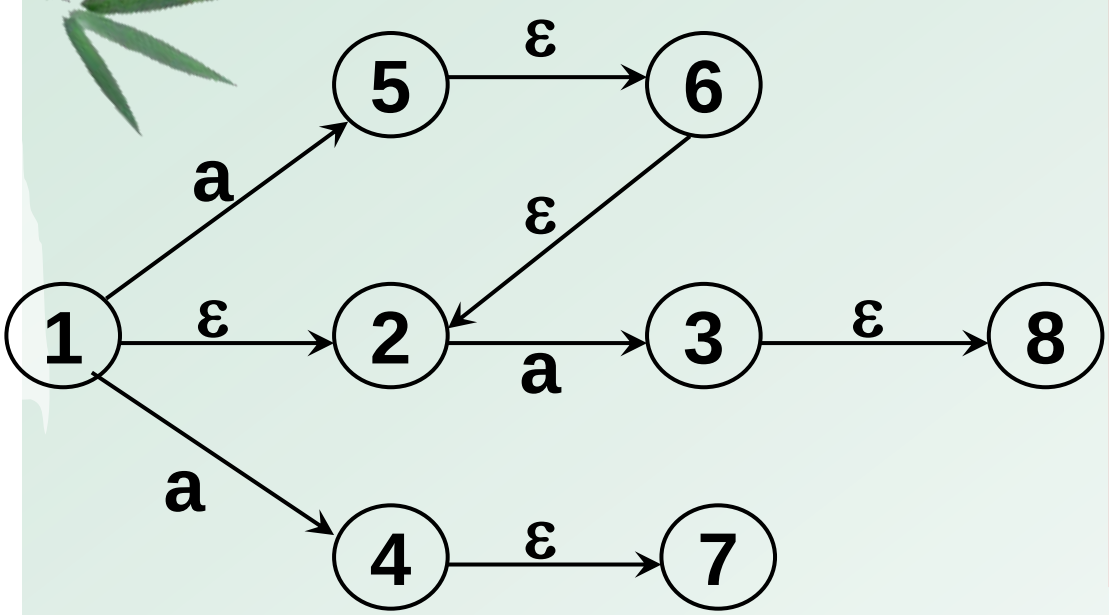
设I是的状态集的一个子集，定义I的ε-闭包ε-closure(I)为:

i）若s∈I，则s∈ε-closure(I)；

ii）若s∈I，则从s出发经过任意条ε弧而能到达的任何状态s’都属于ε-closure(I)

即

ε-closure(I)=I∪{s’|从某个s∈I出发经过任意条ε弧能到达s’}



- 设a是Σ中的一个字符，定义

  I_a= ε-closure(J)

  其中，J为I中的某个状态出发经过一条a弧而到达的状态集合。

- ε-closure({1})={1，2}=I

  J={5，4，3}

  I_a= ε-closure(J)= ε-closure({5，4，3})

  ={5，4，3，6，2，7，8}

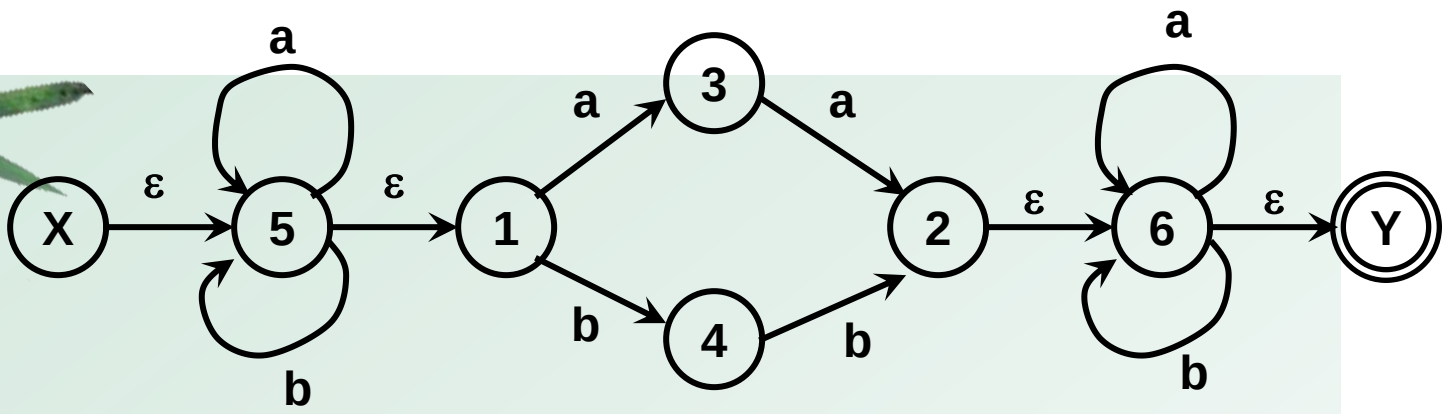
- 确定化的过程：
  设字母表只包含两个a和b，我们构造一张表：

I	I_a	I_b
ε-Closure({X})		

首先，置第1行第1列为ε-closure({X})
求出这一列的I_a，I_b；

然后，检查这两个I_a，I_b，看它们是否
已在表中的第一列中出现，把未曾出现
的填入后面的空行的第1列上，求出每行
第2，3列上的集合...

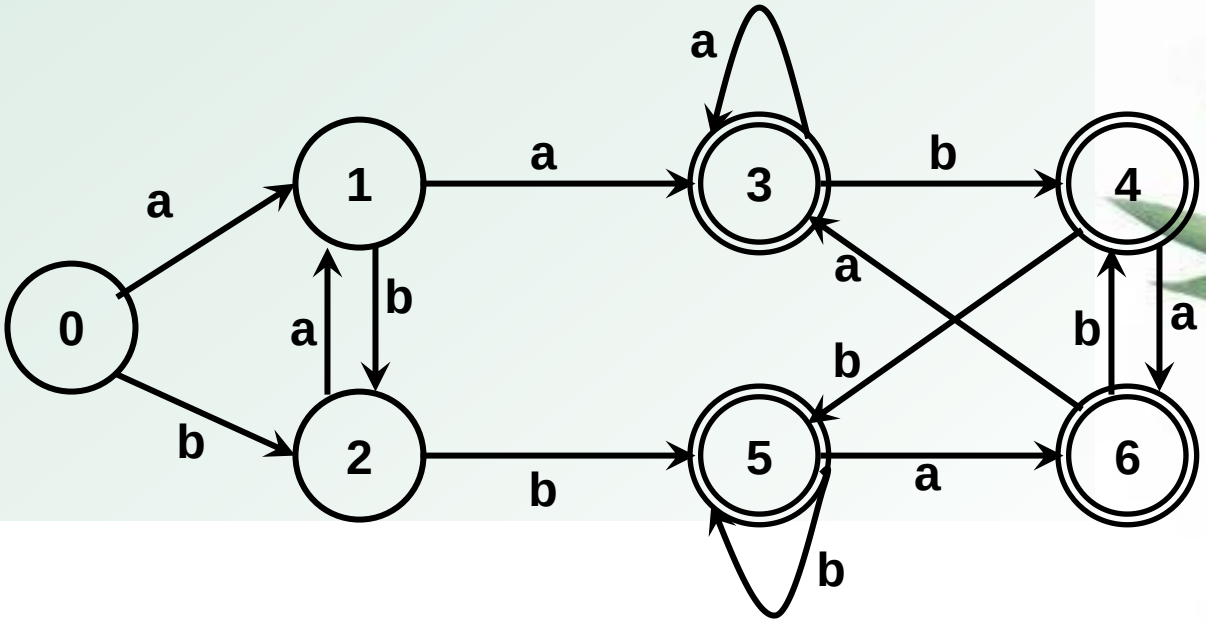
重复上述过程，知道所有第2，3列子集
全部出现在第一列为止。



I	I_a	I_b
{X,5,1}	{5,3,1}	{5,4,1}
{5,3,1}	{5,2,3,1,6,Y}	{5,4,1}
{5,4,1}	{5,3,1}	{5,2,4,1,6,Y}
{5,2,3,1,6,Y}	{5,2,3,1,6,Y}	{5,4,6,1,Y}
{5,4,6,1,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,2,4,1,6,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,3,6,1,Y}	{5,2,3,1,6,Y}	{5,4,6,1,Y}

- 现在把这张表看成一个状态转换矩阵，把其中的每个子集看成一个状态。
- 这张表唯一刻划了一个确定的有限自动机M，它的初态是ε-closure({X})，它的终态是含有原终态Y的子集。
- 不难看出，这个DFA M与M’等价。

I	a	b
0	1	2
1	3	2
2	1	4
3	3	4
4	6	5
5	6	5
6	3	4



3.3.4 正规文法与有限自动机的等价性

- 对于正规文法G和有限自动机M，如果L(G)＝L(M)，则称G和M是等价的。关于正规文法和有限自动机的等价性，有以下结论：

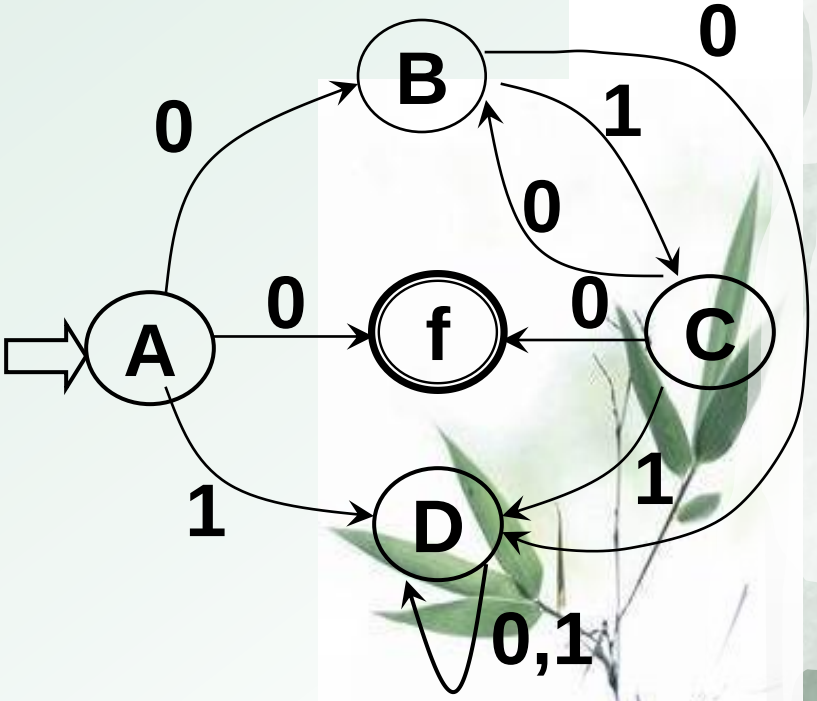
- 定理：

1. 对每一个右线性正规文法G或左线性正规文法G，都存在一个有限自动机(FA) M，使得L(M)＝L(G)。

2. 对每一个FA M，都存在一个右线性正规文法G_R和左线性正规文法G_L，使得L(M)＝L(G_R)＝L(G_L)。

例:

- $G_R(A)$:
  A→0 | 0B | 1D
  B→0D | 1C
  C→0 | 0B | 1D
  D→0D | 1D
- 从G_R出发构造NFA M = <{A, B, C, D, f}, {0, 1}, δ′, A, {f}>, M的状态转换图如右图所示。
- 显然 L(M) = L(G_R)。



例：

- L(M) = 0(10)*
- G_R = <{0, 1}, {A, B, C, D}, A, P>，其中P由下列产生式组成：

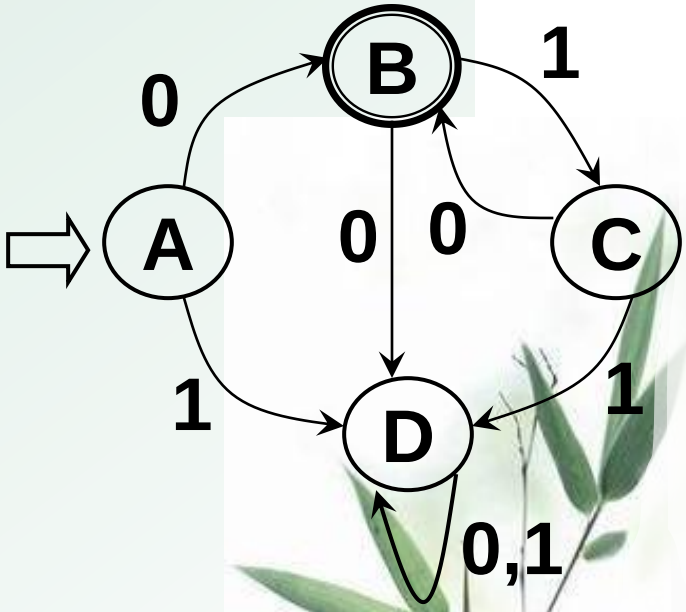
  A→0 | 0B | 1D

  B→0D | 1C

  C→0 | 0B | 1D

  D→0D | 1D

$L(G_R) = L(M) = 0(10)^*$



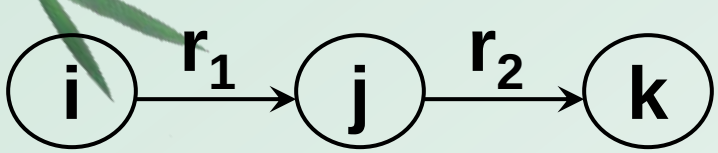
3.3.5 正规式与有限自动机的等价性

- 定理：

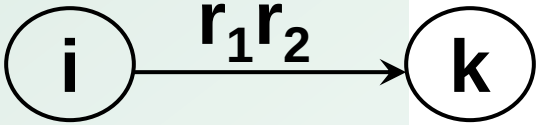
1. 对任何**FA M**，都存在一个正规式**r**，使得**L(r)=L(M)**。

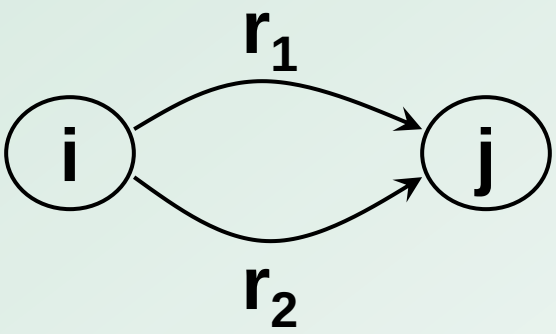
2. 对任何正规式**r**，都存在一个**FA M**，使得**L(M)=L(r)**。

- 对转换图概念拓广，令每条弧可用一个正规式作标记。(对一类输入符号)

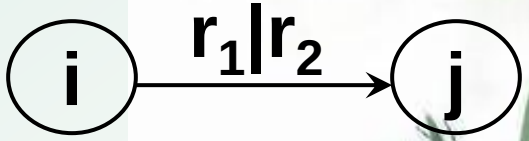


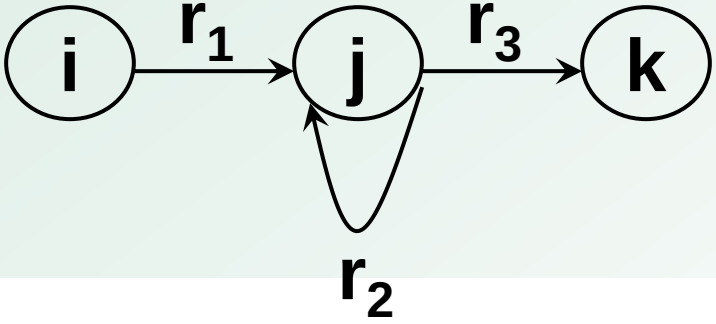
代之为





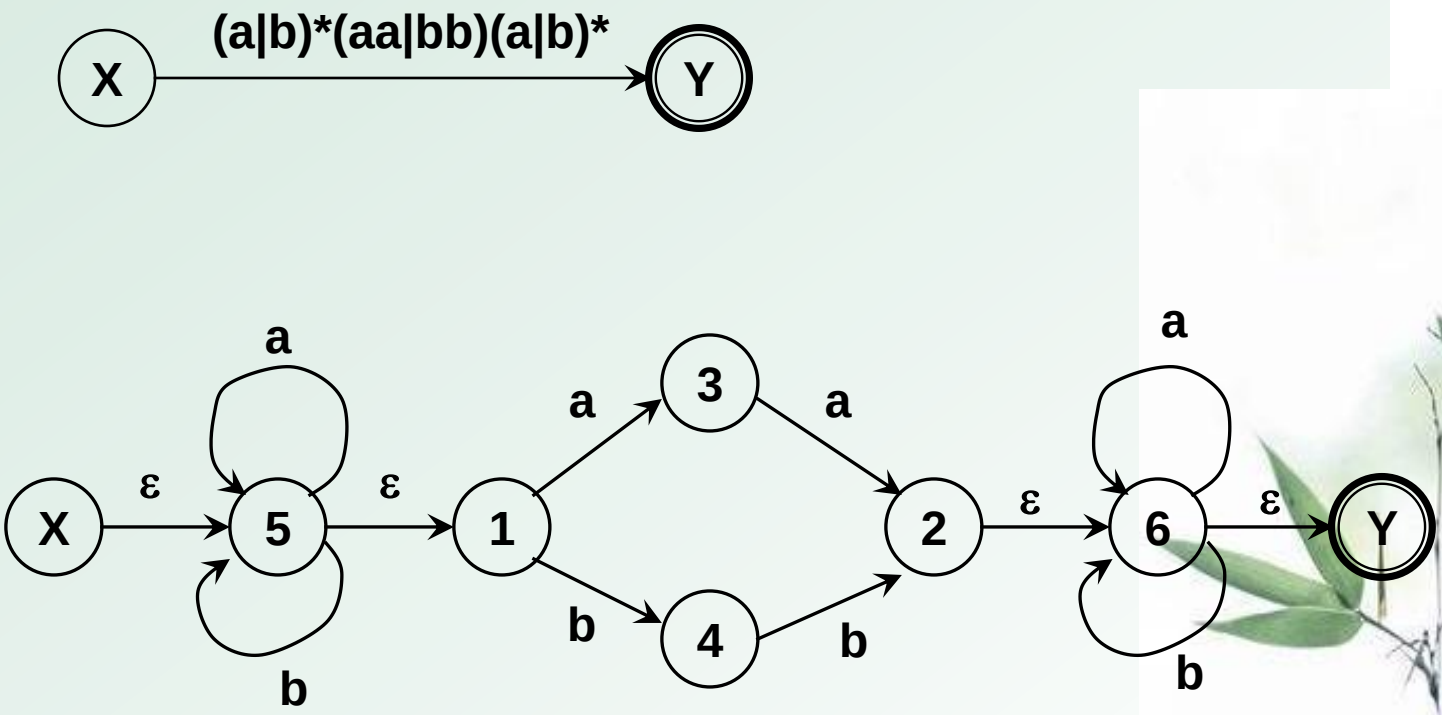
代之为

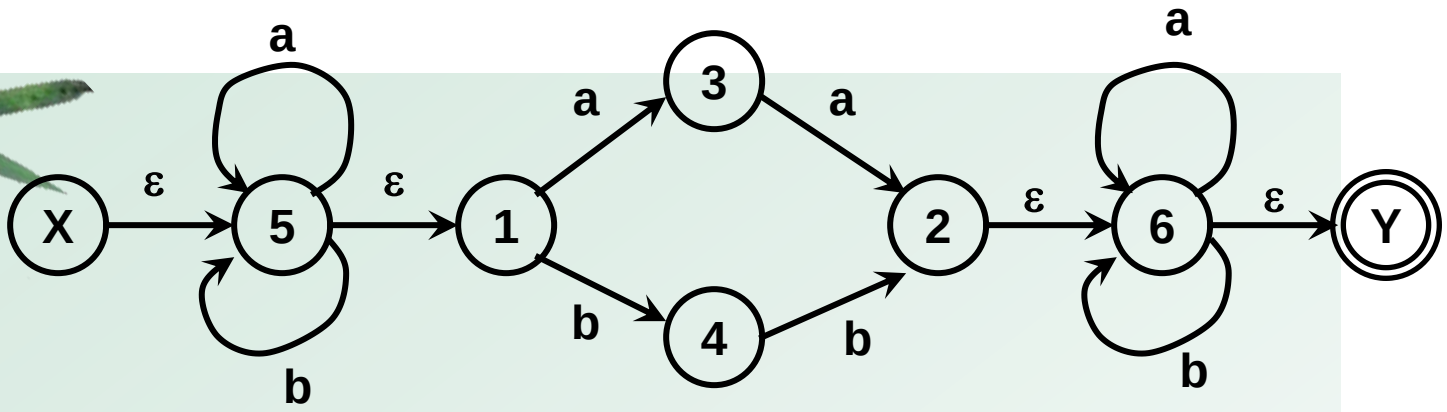




代之为

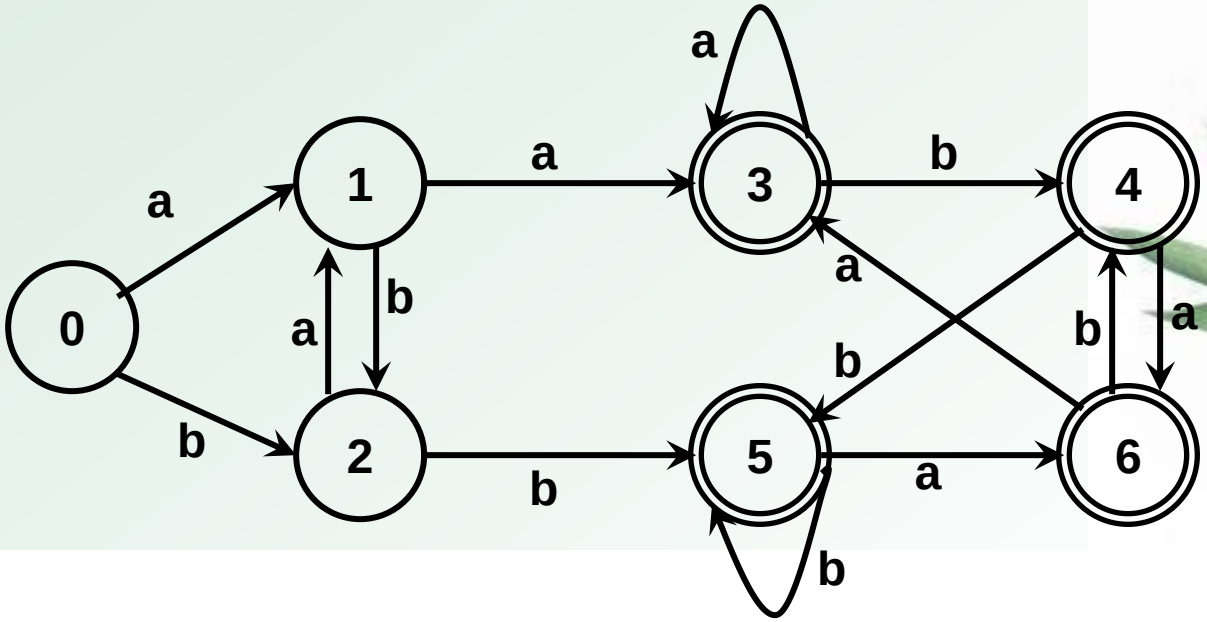
- (a|b)*(aa|bb)(a|b)*





I	I_a	I_b
{X,5,1}	{5,3,1}	{5,4,1}
{5,3,1}	{5,2,3,1,6,Y}	{5,4,1}
{5,4,1}	{5,3,1}	{5,2,4,1,6,Y}
{5,2,3,1,6,Y}	{5,2,3,1,6,Y}	{5,4,6,1,Y}
{5,4,6,1,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,2,4,1,6,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,3,6,1,Y}	{5,2,3,1,6,Y}	{5,4,6,1,Y}

I	a	b
0	1	2
1	3	2
2	1	4
3	3	4
4	6	5
5	6	5
6	3	4



3.3.6 确定有限自动机的化简

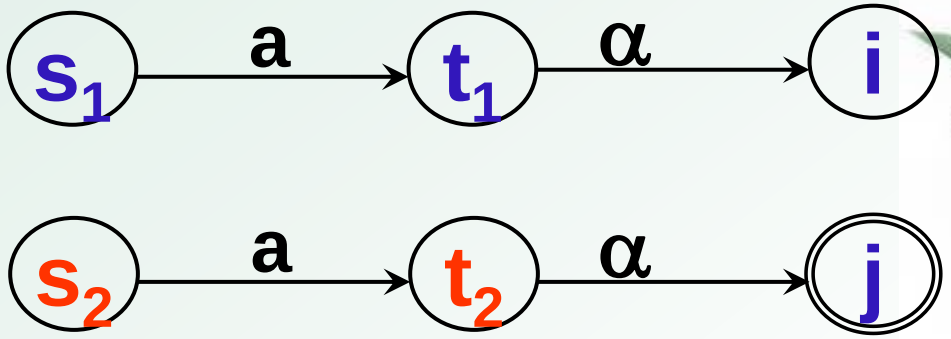
- 对DFA M的化简:寻找一个状态数比M少的DFA M’，使得L(M)=L(M’)
- 假设s和t为M的两个状态，称s和t等价:如果从状态s出发能读出某个字α而停止于终态，那么同样，从t出发也能读出α而停止于终态；反之亦然。
- 两个状态不等价，则称它们是可区别的。

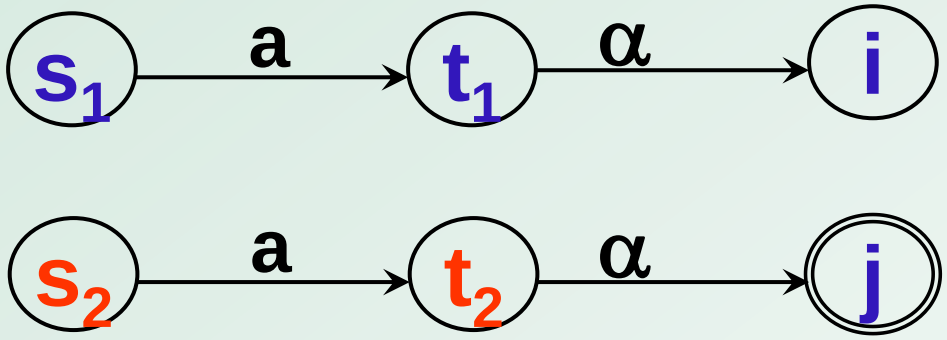
- 对一个DFA M最少化的基本思想:

  把M的状态集划分为一些不相交的子集，使得任何两个不同子集的状态是可区别的，而同一子集的任何两个状态是等价的。最后，让每个子集选出一个代表，同时消去其他状态。

- 具体做法：对M的状态集进行划分
  - 首先，把S划分为终态和非终态两个子集，形成基本划分Π。
  - 假定到某个时候，Π已含m个子集，记为Π={$I^{(1)}$，$I^{(2)}$，…，$I^{(m)}$}，检查Π中的每个子集看是否能进一步划分：
    - 对某个$I^{(i)}$，令$I^{(i)}$={s_1, s_2，…, s_k}，若存在一个输入字符a使得$I_a^{(i)}$ 不会包含在现行Π的某个子集$I^{(j)}$中，则至少应把$I^{(i)}$分为两个部分。

- 例如，假定状态s_1和s_2经a弧分别到达t_1和t_2，而t_1和t_2属于现行∏中的两个不同子集，说明有一个字α，t_1读出α后到达终态，而t_2读出α后不能到达终态，或者反之，那么对于字aα，s_1读出aα后到达终态，而s_2读出aα不能到达终态，或者反之，所以s_1和s_2不等价。





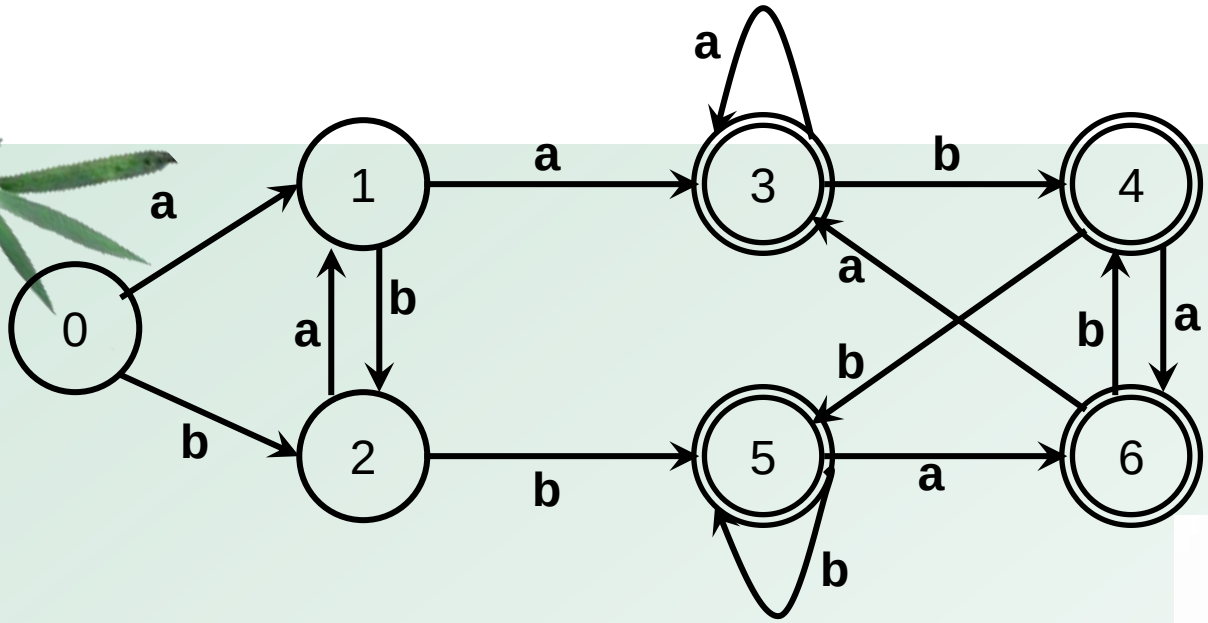
- 则将$I^{(i)}$分成两半，使得一半含有s_1：

  $I^{(i1)}=\{s \mid s\in I^{(i)}$且s经a弧到达t，

  且t与t_1属于现行∏中的同一子集$\}$

  另一半含有s_2：$I^{(i2)}=I^{(i)}-I^{(i1)}$

- 一般地，对某个a和$I^{(i)}$，若$I_a^{(i)}$落入现行$\prod$中 N个不同子集，则应把$I^{(i)}$划分成N个不相交的组，使得每个组J的J_a都落入的$\prod$同一子集。这样构成新的划分。
- 重复上述过程，直到$\prod$所含子集数不再增长。
- 对于上述最后划分$\prod$中的每个子集，我们选取每个子集I中的一个状态代表其他状态，则可得到化简后的DFA M’ 。
- 若I含有原来的初态，则其代表为新的初态，若I含有原来的终态，则其代表为新的终态。



$I^{(1)}=\{0, 1, 2\}$ $I^{(2)}=\{3, 4, 5, 6\}$

$I_a^{(1)}=\{1, 3\}$

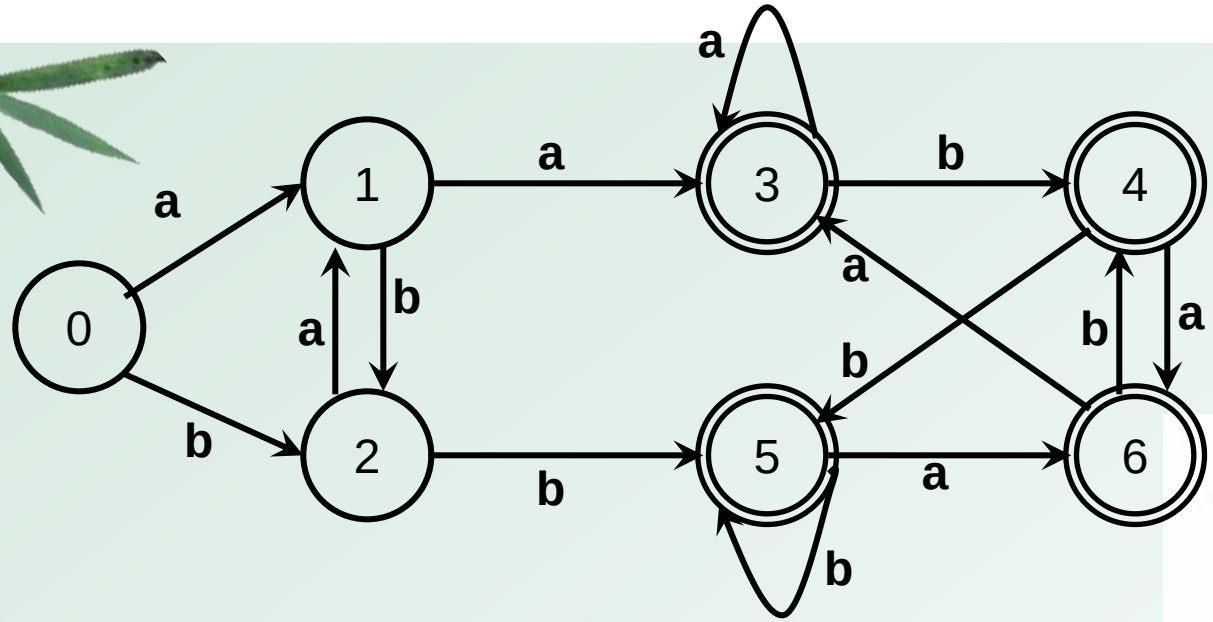
$I^{(11)}=\{0, 2\}$ $I^{(12)}=\{1\}$ $I^{(2)}=\{3, 4, 5, 6\}$

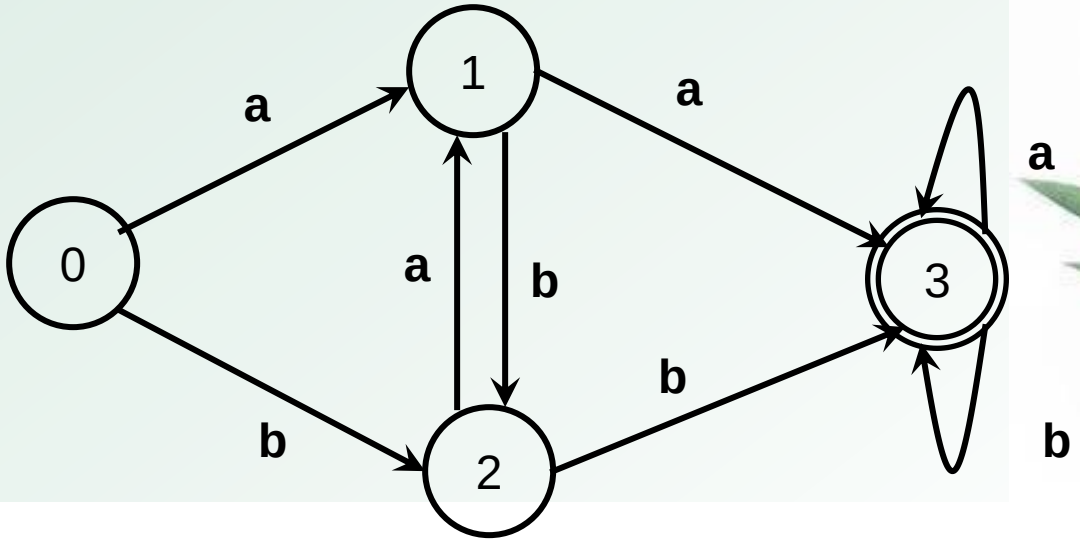
$I^{(11)}=\{0, 2\}$

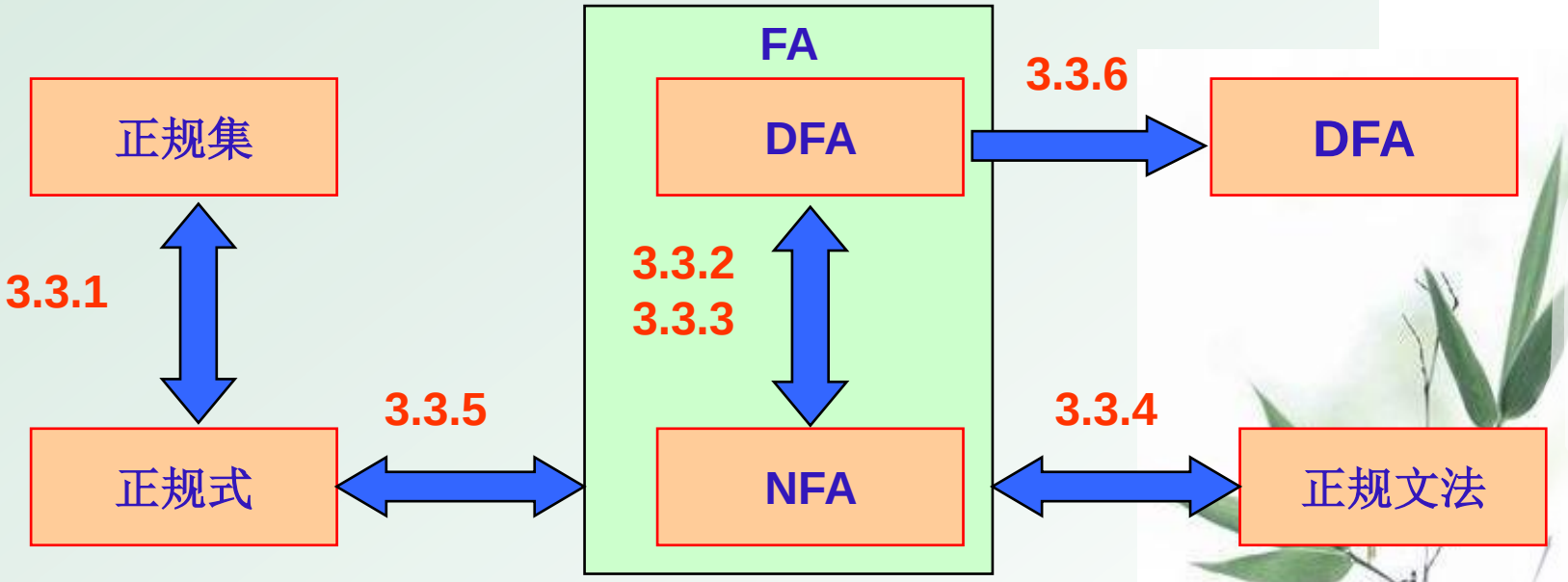
$I_a^{(11)}=\{1\}$ $I_b^{(11)}=\{2, 5\}$

$I^{(111)}=\{0\}$ $I^{(112)}=\{2\}$ $I^{(12)}=\{1\}$ $I^{(2)}=\{3, 4, 5, 6\}$

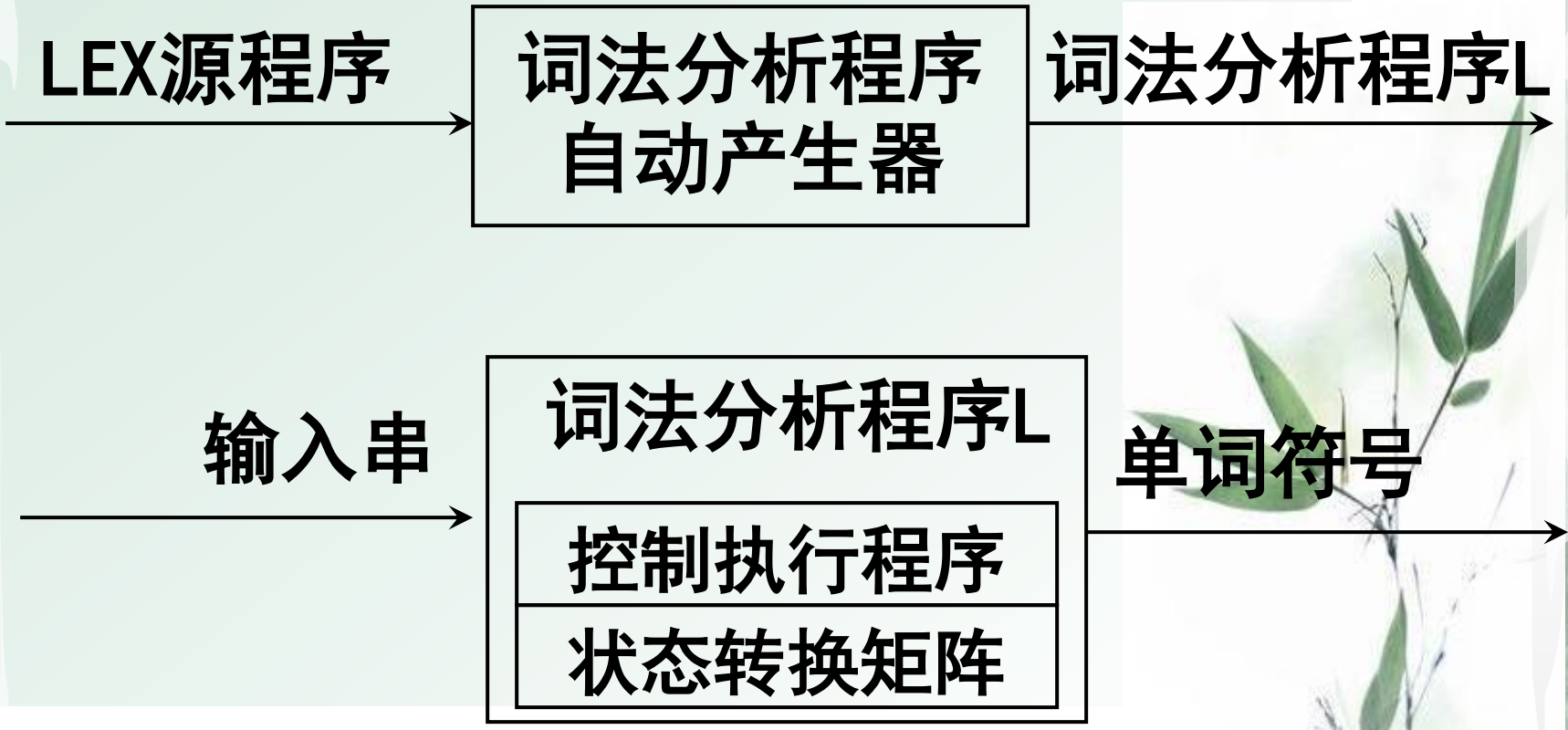
$I_a^{(2)}=\{3, 6\}$ $I_a^{(2)}=\{4, 5\}$


0
1
2
3
4
5
6
a
b


0
1
2
3
a
b


FA
正规集
DFA
DFA
3.3.6
3.3.1
3.3.2
3.3.3
3.3.5
3.3.4
正规式
NFA
正规文法

3.4 词法分析器的自动产生--LEX



AUXILIARY DEFINITION

letter→A|B|...|Z

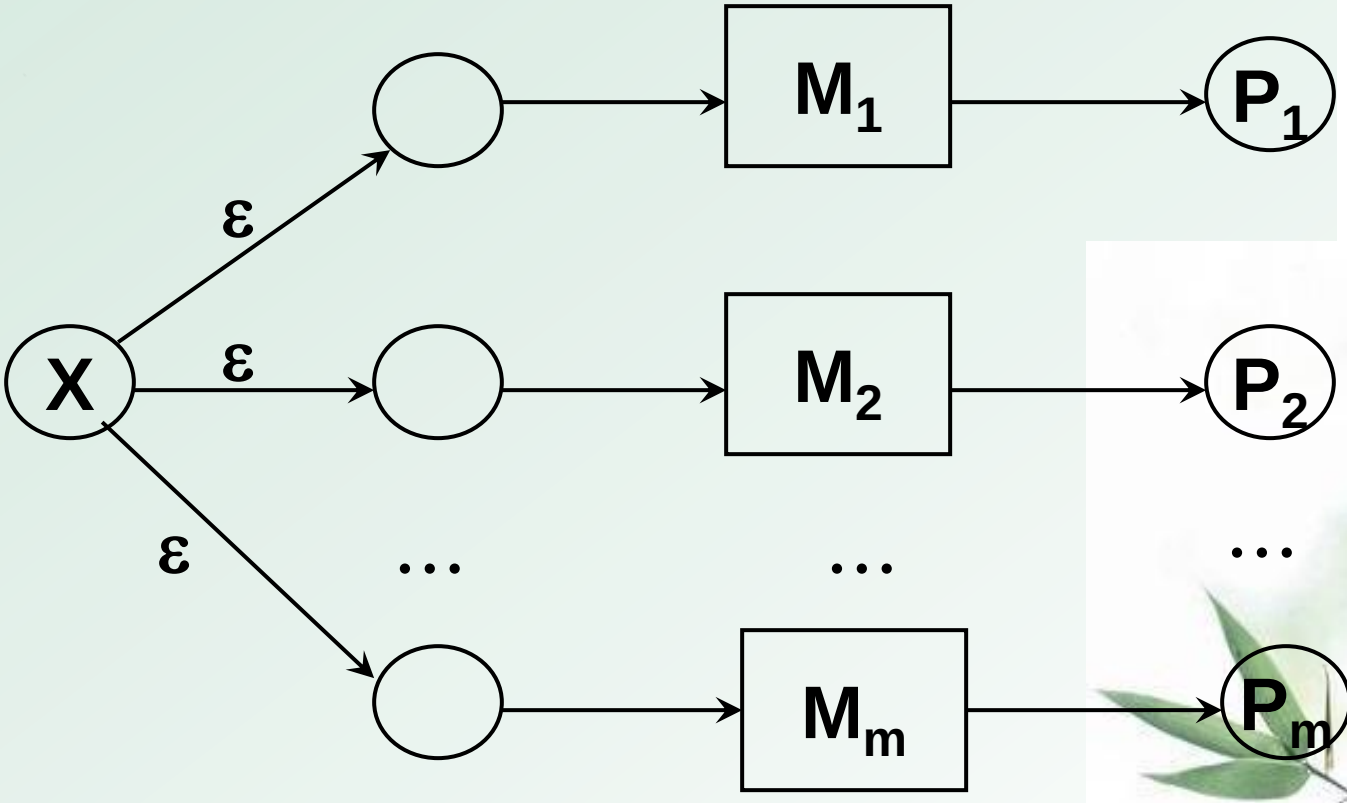
digit →0|1|...|9

正规式

RECOGNITION RULES

1	DIM	{ RETURN (1,-) }
2	IF	{ RETURN (2,-) }
3	DO	{ RETURN (3,-) }
4	STOP	{ RETURN (4,-) }
5	END	{ RETURN (5,-) }
6	letter(letter|digit) *	{ RETURN (6, TOKEN) }
7	digit(digit)*	{ RETURN (7, DTB) }
8	=	{ RETURN (8, -) }
9	+	{ RETURN (9,-) }
10	*	{ RETURN (10,-) }
11	**	{ RETURN (11,-) }
12	,	{ RETURN (12,-) }
13	(	{ RETURN (13,-) }
14	)	{ RETURN (14,-) }

- LEX的工作过程：
  - 首先，对每条识别规则P_i构造一个相应的非确定有限自动机M_i；
  - 然后，引进一个新初态X，通过ε弧，将这些自动机连接成一个新的NFA；
  - 最后，把M确定化、最小化，生成该DFA的状态转换表和控制执行程序


X
ε
ε
ε
M1
M2
Mm
P1
P2
Pm
…
…
…