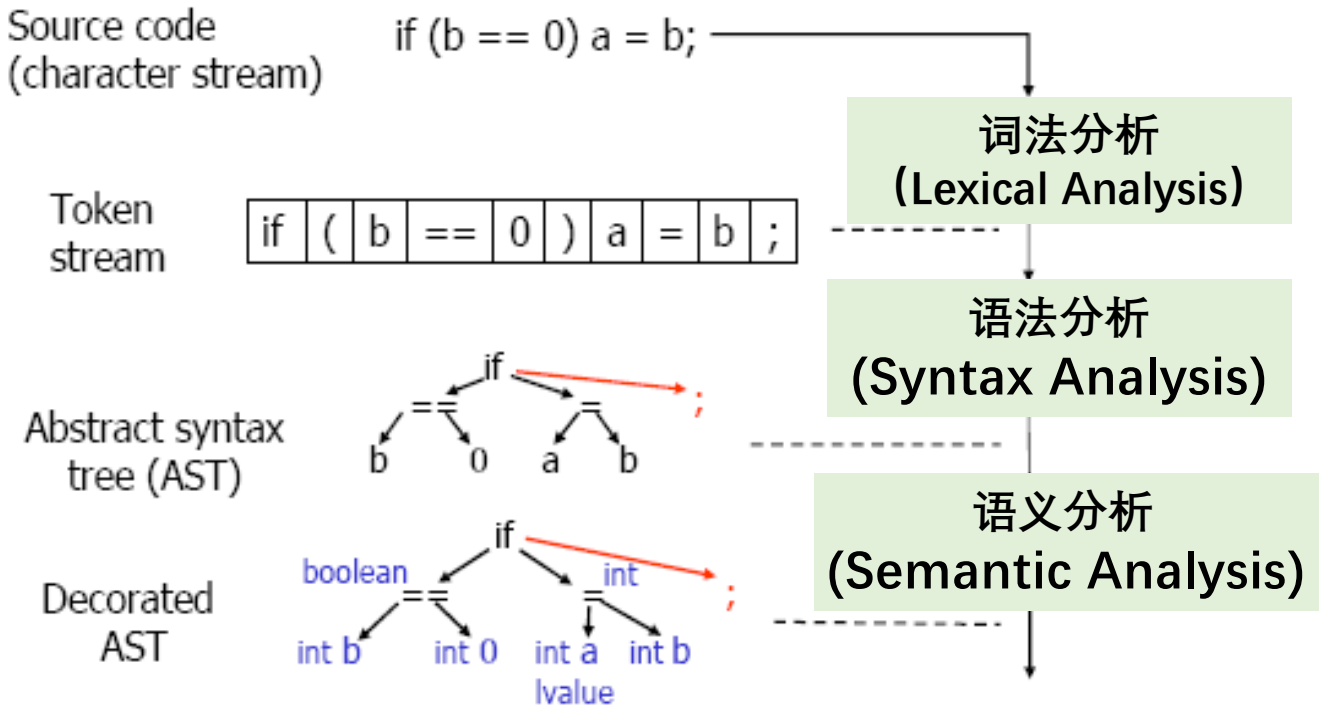
编译原理

--词法分析自动生成

刘爽

中国人民大学信息学院

编译器结构

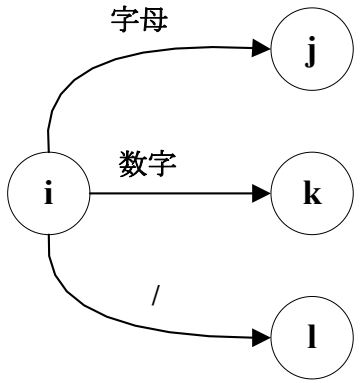

Source code
(character stream)
if (b == 0) a = b;
词法分析
(Lexical Analysis)
Token
stream
if
(
b
==
0
)
a
=
b
;
语法分析
(Syntax Analysis)
Abstract syntax
tree (AST)
if
==
=
;
b
0
a
b
语义分析
(Semantic Analysis)
Decorated
AST
if
boolean
int
==
=
;
int b
int 0
int a
lvalue
int b

状态转换图的实现

- 程序实现:每个状态结点对应一段程序.

1) 不含回路的分叉结点:

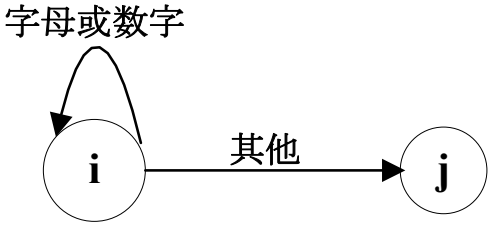
- CASE 或者 IF--THEN—ELSE




```
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) 含回路的分叉结:

- WHILE




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

3) 终点结:

- RETURN(Code, Value): 返回调用者

种别编码，属性值

Outline

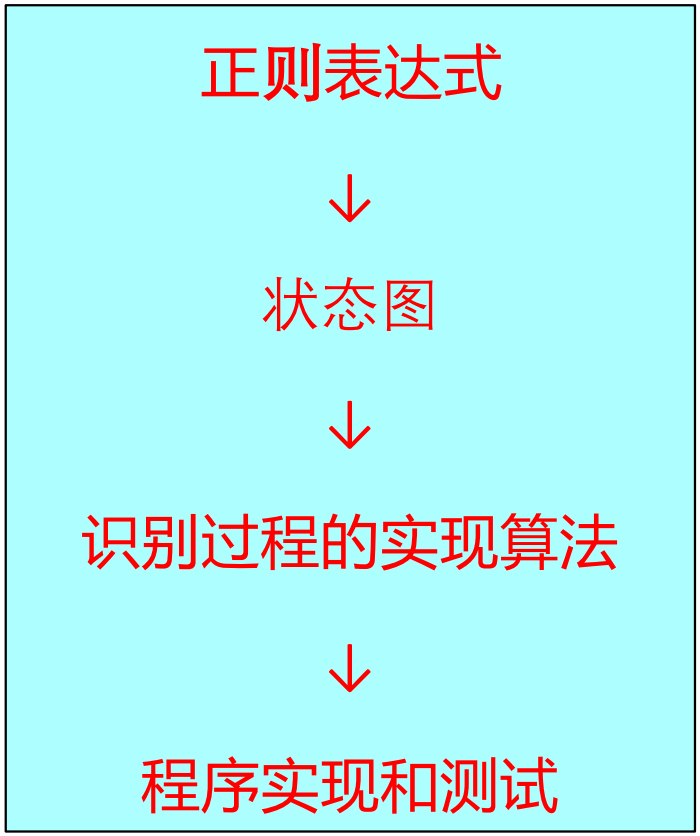
- 词法分析器的作用
- 词法分析程序的设计与实现
  - 状态转换图
- 正则表达式和有限自动机
  - 正规/正则表达式（Regular Expression）
  - 有限自动机 （Finite Automata）
  - 正则文法（Regular Gramma）
- 词法分析程序的自动生成

问题的提出

- 将状态转换图的概念稍加形式化，更好的构造词法分析程序
- 通过正则表达式与有限自动机的相互转换，实现词法分析程序的自动生成器。
- 有助于理解词法分析器的生成器，可以自动产生有效的词法分析器。（例如lex，flex，Jlex）

词法分析器的实现步骤

- 用正则表达式描述词法规则，设置单词种别和属性；
- 按照右图所示步骤，逐步实现词法分析器；



正则表达式与正则集 (Regular Set)

- 正则表达式可以详细说明单词符号的结构，可以精确地定义集合（正则集合）。正则表达式和正则集合的递归定义如下：
  1. ε 和Φ 都是Σ 上的正则式，其正则集为{ε} 和Φ；
  2. 任何**a**∈Σ，　**a**是Σ的一个正则式，正则集为{**a**}；
  3. 假定**U** 和**V** 都是Σ 上的正则式，正则集为**L(U)** 和**L(V)**，那么，

     (U|V), (UV), (U)*也都是**S**上的正则式，正则集为**L(U) U L(V)**，**L(U)L(V)** 和**L(U)***；

  仅由有限次使用上述三步骤而定义的表达式才是Σ上的正则表达式，仅由这些正则表达式所表示的字集才是Σ上的正则集。

正则集合（正规集）：可以用一个正则表达式定义的语言。

正则表达式和正则集的例子

令Σ={a，b},

正则表达式	正则集合集
a	{a}
a|b	{a, b}
ab	{ab}
(a|b)(a|b)	{aa, ab, ba, bb}
a*	{ε ,a, aa, ……任意个a的串}
(a|b)*	{ε, a, b, aa, ab ……所有由a 和b组成的串}
(a|b)*(aa|bb)(a|b)*	{Σ*上所有含有两个相继的a或两个相继的b组成的串}

正则表达式和正则集—练习

- Σ={a,b} 则Σ上的正则表达式
  - $a^*ba^*ba^*ba^*$表示的正则集是？
  - $((\varepsilon|a)b^*)^*$表示的正则集是？

正则表达式的等价性

- 若两个正则表达式所表示的正则集合相同，则认为二者等价。两个等价的正则表达式U和V记为U=V。
  - 例如： U= (a|b)， V = (b|a)
  - 又如： U= b(ab)* ， V =(ba)*b
  - 再如： U= (a|b)* ， V =(a*|b*)*

正则表达式的性质

- 设U、V和W都是正则表达式，则如下关系普遍成立：
  - U|V = V|U　　（交换律）
  - U|(V|W) = (U|V)|W　　（结合律）
  - U(VW) = (UV)W　　（结合律）
  - U(V|W) = UV | UW　　（分配律）
    (V|W)U = VU |WU;
  - εU = Uε=U
  - r|r=r　　r*=ε|r|rr|…　“或”的抽取律

- 运算优先级和结合性：
  - *　高于 "连接" 高于 |
  - |　具有交换律、结合律
  - “连接”　具有结合律、分配律
  - ( )　　指定优先关系

Outline

- 词法分析器的作用
- 词法分析程序的设计与实现
  - 状态转换图
- 正则表达式和有限自动机
  - 正规/正则表达式（Regular Expression）
  - 有限自动机 （Finite Automata）
  - 正则文法（Regular Gramma）
- 词法分析程序的自动生成

有限自动机

有限自动机

- 有限自动机（也称有穷自动机）作为一种识别装置，它能准确地识别正则集合，即识别正则文法所定义的语言与正则表达式所表示的集合，引入有穷自动机这个理论，正是为词法分析程序的自动构造寻找特殊的方法和工具。
- 有限自动机分为两类：
  - 确定有限自动机 Deterministic Finite Automata（DFA）
  - 不确定有限自动机 Nondeterministic Finite Automata（NFA）
- 其中“不确定”的含义是：对于某个输入符号，在同一状态上存在不止一种转换。
- 确定和不确定有限自动机都能而且仅能识别正则集合，即它们能够识别正则表达式所表示的语言。

确定有限自动机

确定有限自动机 (DFA)

- 一个确定的有限自动机(DFA)是一个五元式：$M=(S, \sum, \delta, s_0, F)$ 其中
  - S是一个有限集，它的每个元素称为一个状态。
  - $\sum$是一个有穷字母表，它的每个元素称为一个输入字符
  - $\delta: S\times\sum \rightarrow S$ 是一个单值映射（确定性）。
    - $\delta(s, a)=s'$意味着：当现行状态为s、输入字符为a时，将转换到下一个状态s'。我们称s'为s的一个后继状态。
  - $s_0 \in S$，是唯一的初态。
  - $F \subseteq S$，是一个终态集（可空）

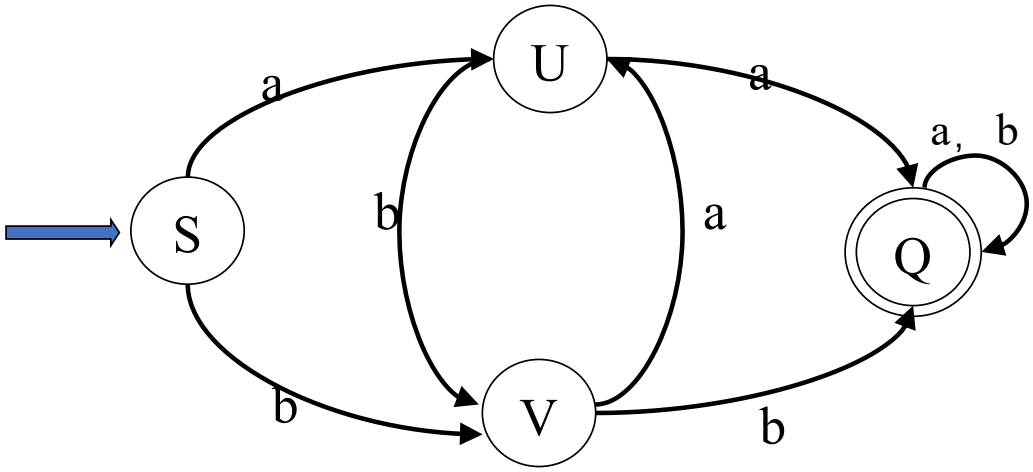
确定的有限自动机—表示方式

DFA M=（{S，U，V，Q}，{a，b}，δ，S，{Q}）其中δ定义为：

δ（S，a）=U　　δ（V，a）=U
δ（S，b）=V　　δ（V，b）=Q
δ（U，a）=Q　　δ（Q，a）=Q
δ（U，b）=V　　δ（Q，b）=Q

状态 \ 字符	a	b
S	U	V
U	Q	V
V	U	Q
Q	Q	Q

图（1）状态转换矩阵



图（2）状态转换图

δ(s,a)值------状态转换矩阵值。如图（1）所示：行--状态，列--输入字符

一个DFA也可以表示为一张（确定的）状态转换图，如图（2）所示

练习

DFA M 可用一张（确定的）状态转换图表示。

- DFA M=({A,B,C,D},{a,b}, δ,A,{D})

$$\delta(A,a)=B \quad \delta(A,b)=A$$
$$\delta(B,a)=B \quad \delta(B,b)=C$$
$$\delta(C,a)=B \quad \delta(C,b)=D$$
$$\delta(D,a)=B \quad \delta(D,b)=A$$

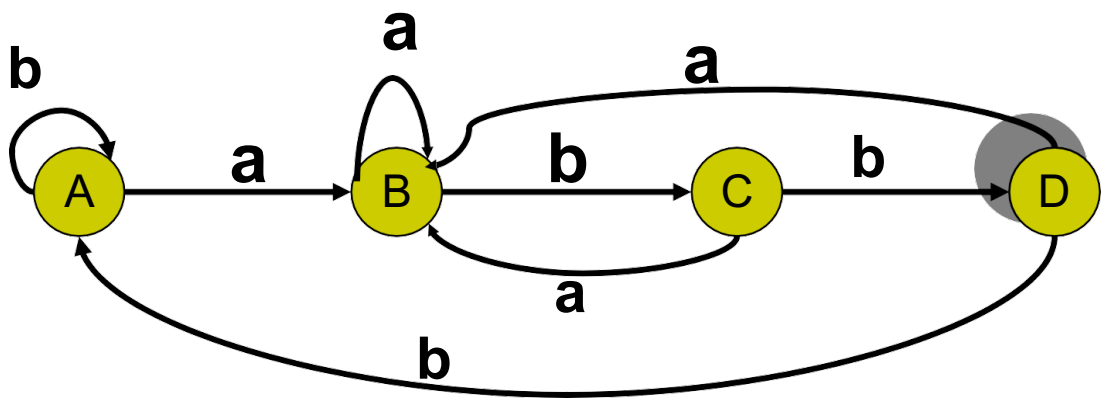
画出其对应的状态转换图和状态转换矩阵

答案

M(A,a)=B　M(A,b)=A
M(B,a)=B　M(B,b)=C
M(C,a)=B　M(C,b)=D
M(D,a)=B　M(D,b)=A

状态 \ 字符	a	b
A	B	A
B	B	C
C	B	D
D	B	A

图（1）状态转换矩阵



图（2）状态转图

确定有限自动机（DFA）接受的字符串

- DFA接受的字符串
  - 对于$\sum^*$中的任何字符串t，若存在一条从初态到某一终态的路径，且这条路径上所有弧的标记字符连接成的字符串等于t, (即若$t=t_1t_2...t_n \in \sum^*$，$\delta(S, t_1)=P_1$，$\delta(P_1, t_2)=P_2$，…, $\delta(P_{n-1}, t_n)=P_n$，其中S为**M**的开始状态, $P_n \in Z$, Z为终态集）则称t可被**DFA M接受(识别)**
  - 若M的初态同时又是终态，则空字符串可为M所识别。
  - 一个**DFA M** 所能识别的所有的字符串的集合记为：**L(M)**
  - Σ上的一个字集**V** ⊆ **Σ*** 是正则的，当且仅当存在Σ上的**DFA M ,**使得**V=L(M)**
- $\sum^*$上的符号串t，(我们将它表示成t_1t_x的形式，其中$t_1 \in \sum$，$t_x \in \sum^*$) 在**DFA M**上运行的定义为：
  - $\delta(Q, t_1t_x)= \delta(\delta(Q, t_1), t_x)$ 其中$Q \in S$ （是**M**的开始状态）

确定有限自动机—例子

- 例：证明t=baab被例中的DFA所接受。

δ (S,baab)

= δ (δ (S,b),aab)

= δ (V,aab)

= δ (δ (V,a),ab)

= δ (U,ab)

= δ (f(U,a),b)

= δ (Q,b)

=Q

DFA M=（{S，U，V，Q}，{a，b}，δ，S，{Q}）其中δ定义为：

δ（S，a）=U　δ（V，a）=U
δ（S，b）=V　δ（V，b）=Q
δ（U，a）=Q　δ（Q，a）=Q
δ（U，b）=V　δ（Q，b）=Q

Q属于终态, 找到了从M的初态到终态的一条路径，该路径上的字符连接成的字符串为t。

非确定有限自动机

非确定有限自动机 （NFA）

- 一个非确定有限自动机(NFA)是一个五元式： $M=(S, \sum, \delta, S_0, F)$ 其中
  - S是一个有限集，它的每个元素称为一个状态。
  - $\sum$是一个有穷字母表，它的每个元素称为一个输入字符
  - $\delta: S\times\sum^* \rightarrow 2^S$ 是一个从$S\times\sum^*$至**S子集**的单值映射。
  - $S_0 \subseteq S$，是一个非空的**初态集**
  - $F \subseteq S$，是一个终态集（可空）

若δ是一个多值函数，且输入可允许为ε，则有穷自动机是不确定的，即在某个状态下，对于某个输入字符存在多个后继状态。

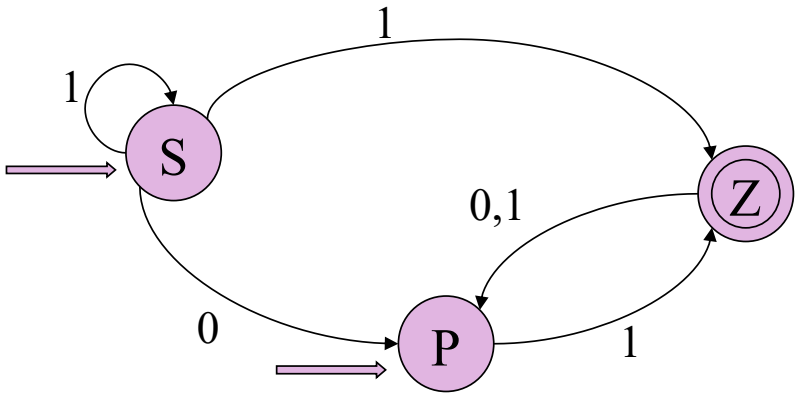
非确定的有限自动机—表示方式

NFA N=({S,P,Z},{0,1},δ,{S,P},{Z})

- δ(S,0)={P}
- δ(S,1)={S, Z}
- δ(P,1)={Z}
- δ(Z,0)={P}
- δ(Z,1)={P}

	0	1
S	{P}	{S,Z}
P	{}	{Z}
Z	{P}	{P}

图（1）状态转换矩阵



图（2）状态转换图

δ (s , a) 值------状态转换矩阵值。如图（1）所示：行--状态，列--输入字符

一个NFA也可以表示为一张（确定的）状态转换图，如图（2）所示

NFA--练习

画出下列NFA的状态转换矩阵和状态转换图

NFA M=({S,Q,U,V,Z},{0,1}, δ,{S},{Z});

δ(S,0)={V,Q} **δ(S,1)={U,Q}**

δ(U,0)=Φ **δ(U,1)={Z}**

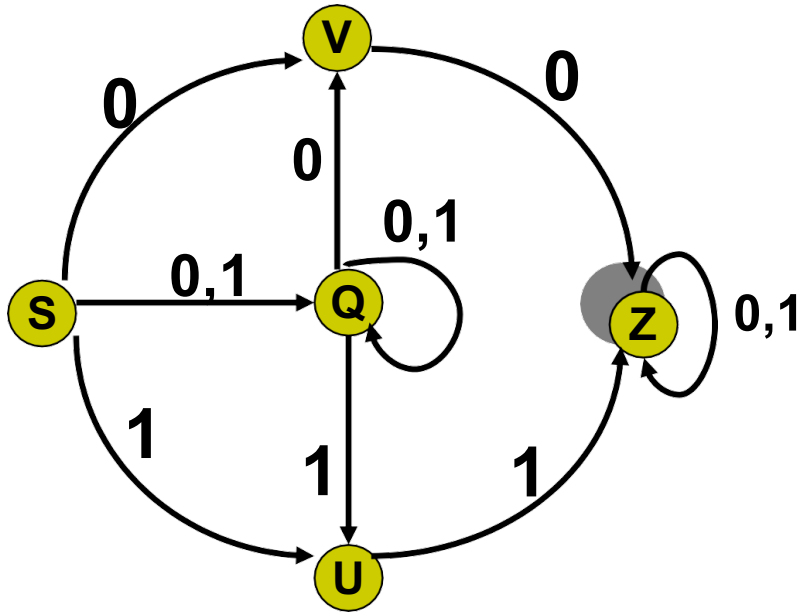
δ(V,0)={Z} **δ(V,1)= Φ**

δ(Q,0)={V,Q} **δ(Q,1)={U,Q}**

δ(Z,0)={Z} **δ(Z,1)={Z}**

答案

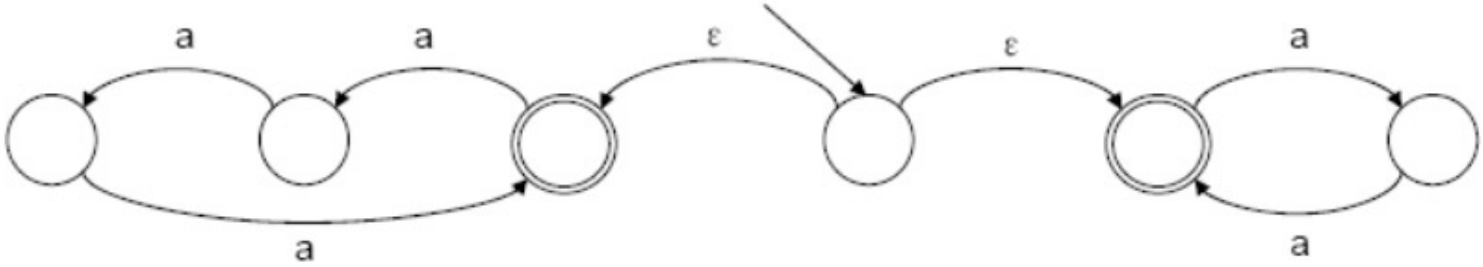
	0	1
S	{V,Q}	{U,Q}
Q	{V,Q}	{U,Q}
U	ϕ	{Z}
V	{Z}	ϕ
Z	{Z}	{Z}



图中某些状态射出两条具有相同标记的弧，如**S,Q.**

非确定有限自动机(NFA)接受的字符串

- 对于∑*中任何字α，若存在一条从某一初态到某一个终态的路径，且这条路径上所有弧的标记字符依序连接成的字符串等于α，则称α可为NFA M所识别（接受）。
- 若M的某些结点既是初态又是终态，或存在一条从某一初态到某一终态的ε路径，那么空字ε可为M所识别。

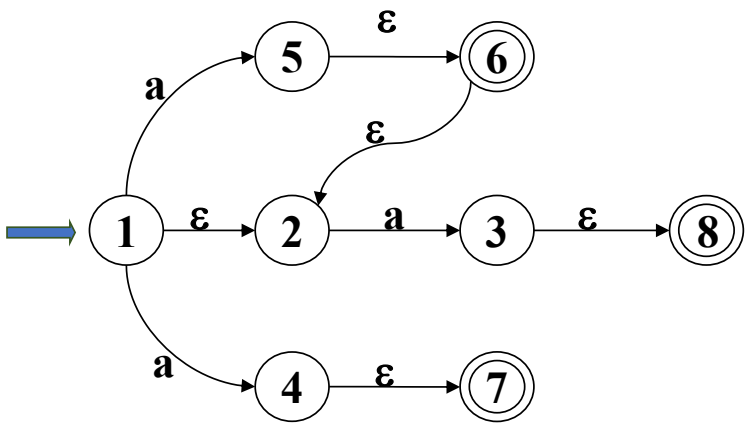


NFA的确定化

NFA的确定化

- 显然，DFA是NFA的特例，但是对于每个NFA M都存在一个DFA M''，使L(M)=L(M'')。
- 定义对状态集合I的几个有关运算：
  - 状态集合I的**ε-闭包**，表示为**ε-closure(I)**，定义为一状态集，包括
    (1)状态集合I的任何状态s都属于**ε-closure(I)**；
    (2)状态集合I中的任何状态s经任意条ε弧而能到达的状态的集合。
  - 状态集合I的**α弧转换** I_a = **ε-closure(move(I, a)) = ε-closure(J)**，其中**move(I, a)** 为状态集合J，其定义是所有那些可从I的某一状态经过一条a弧而到达的状态的全体。

ε-闭包--例子



- I={1}, ε-closure(I)={1,2};
- I={5}, ε-closure(I)={5,6,2};
- I={1,2},
  move({1,2},a)={5,3,4}=J;
  I_a=ε-closure({J})=ε-closure({5,3,4})={2,3,4,5,6,7,8};

不具有ε–转移的NFA转具有ε–转移的NFA

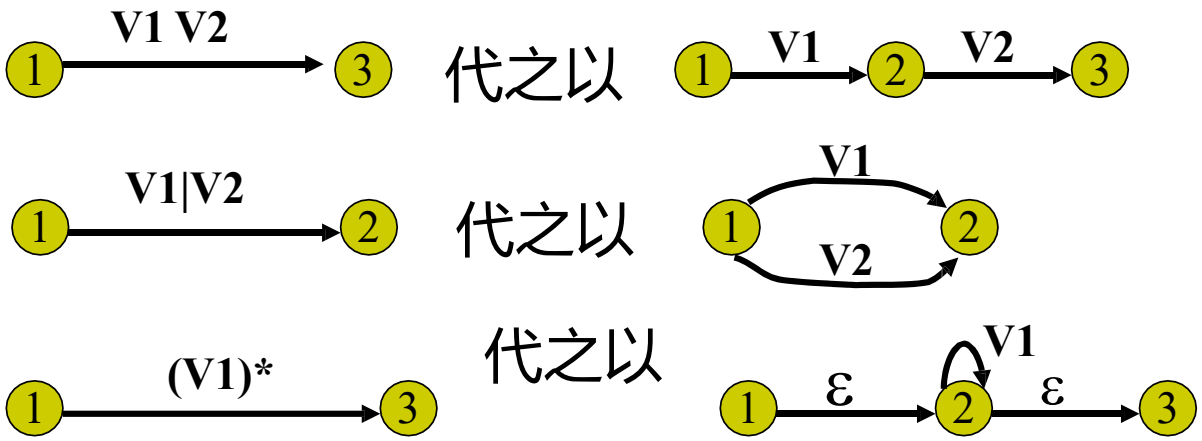
- 定理1：对任何一个不具有ε–转移的NFA M'=(Q, Σ, f', q0, F')，一定存在一个具有ε–转移 的 NFA M, 使
  - L(M) = L(M')

从M'出发构造一个具有ε–转移的 NFA M，使得L(M) = L(M')

- 步骤：
  （1）引进初态X和终态Y，X, Y ∉ S, 从X到S_0中任意状态连接一条ε箭弧，从F中任意状态连接一条ε箭弧到Y。

不具有ε–转移的NFA转具有ε–转移的NFA

（2）对M的状态转换图进一步实施如下替换，其中状态2是新引入状态。重复该过程，直到每条箭弧上标记ε，或者为∑中的单个字符。



将 ε–NFA转化为等价的 DFA

- 定理2：对于字母表Σ上任何一个具有ε–转移的NFA M, 一定存在一个的DFA M', 使得：L(M') = L(M)。

- NFA M =(S,Σ,f,S0,Z') 用构造ε–closure(T) 的方法构造DFA M' = （S',Σ,f',q0,Z')：

- 基本思想：

1) 首先从S0出发，仅经过任意条ε箭弧所能到达的状态所组成的集合I作为M'的初态q0.

2) 分别把从q0（对应于M的状态子集I）出发，经过任意a∈Σ 的a弧转换 I_a 所组成的集合作为M' 的状态，如此继续，直到不再有新的状态为止。

将 ε–NFA转化为等价的 DFA (cont)

- I_a = ε-closure(move(I,a)) = ε-closure(J)

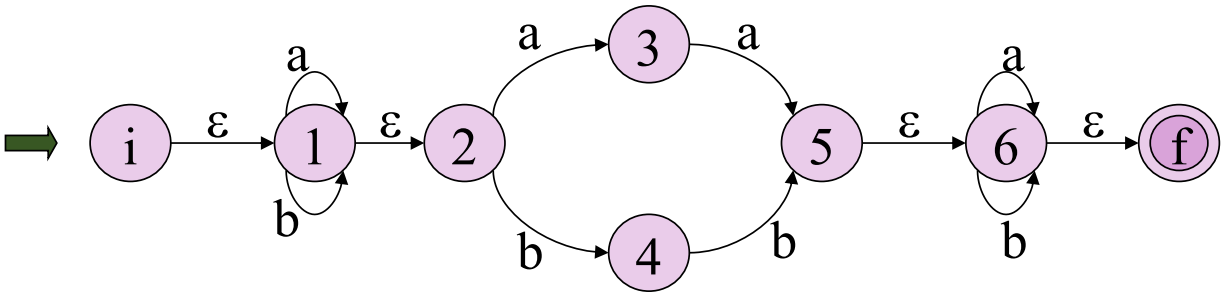
- 假定∑有k个字母。我们构造一张表，有k+1列。
  - 表的每行首列为ε-closure(X)，X ⊂ ∑ ，且未曾出现在表的第一列。
  - 表的第i+1列为I_{ai}, a_i 为∑中的第i个字符。
  - 重复上述过程填表，因为∑子集有限，所以该过程一定会终止。

DFA和NFA的等价性

- 定理3：对任何一个**NFA M** =(S, Σ, f, S_0, F)，都存在一个**DFA M'** =(S', Σ, f', s_0, F')，使得**L(M')=L(M)**。
  - 证明的思想是由**M**出发构造等价的**M'**，办法是让**M'** 的一个状态对应于**M**的一个状态集合。
  - 即若 δ**(s, a)={s1, s2, …, sk},**
  - 我们将集合**{s1, s2, …, sk}**作为一个整体看作**M'**中的一个状态，即**S'**中的一个元素。

NFA确定化的例子

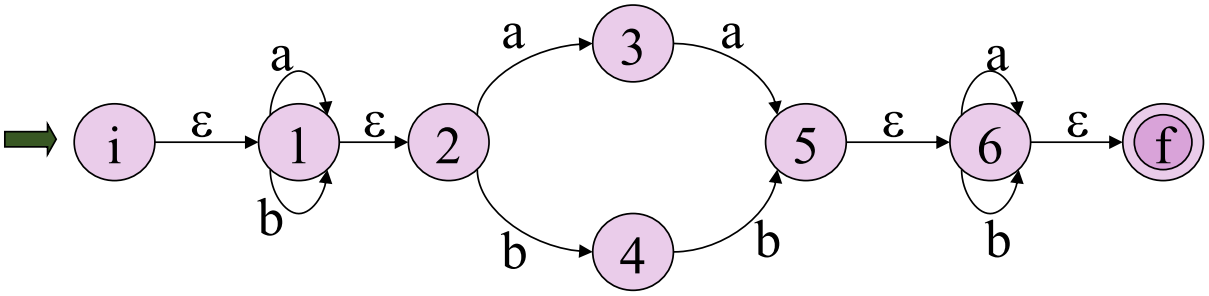
I_a = ε-closure(move(I,a)) = ε-closure(J)



I	Move(I, a)	I_a	Move(I, b)	I_b

NFA确定化的例子

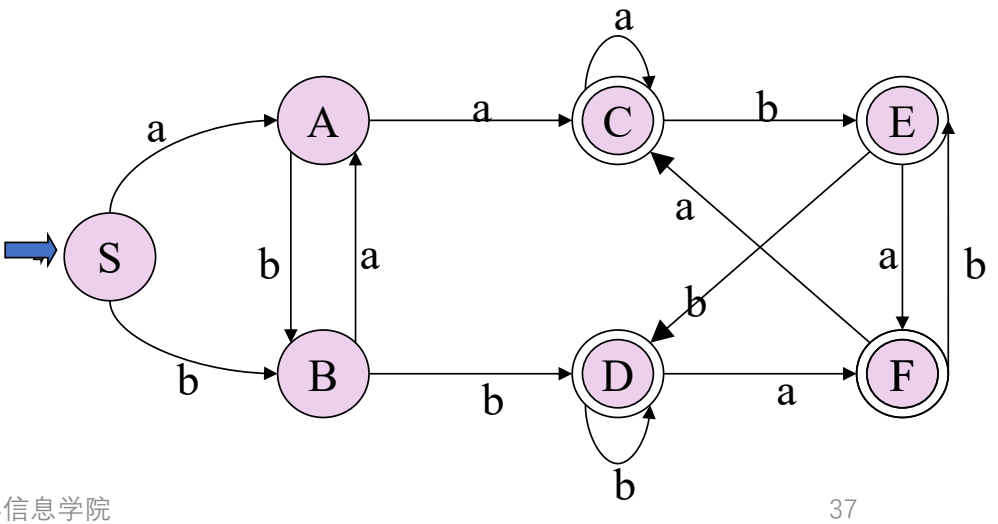
I_a = ε-closure(move(I,a)) = ε-closure(J)



原NFA所有初始状态构成集合的的ε-闭包作为DFA的初态

包含原终止状态f的状态子集为DFA的终态

	I_a	I_b
{i, 1, 2} S	{1, 2, 3} A	{1, 2, 4} B
{1, 2, 3} A	{1, 2, 3, 5, 6, f} C	{1, 2, 4} B
{1, 2, 4} B	{1, 2, 3} A	{1, 2, 4, 5, 6, f} D
{1, 2, 3, 5, 6, f} C	{1, 2, 3, 5, 6, f} C	{1, 2, 4, 6, f} E
{1, 2, 4, 5, 6, f} D	{1, 2, 3, 6, f} F	{1, 2, 4, 5, 6, f} D
{1, 2, 4, 6, f} E	{1, 2, 3, 6, f} F	{1, 2, 4, 5, 6, f} D
{1, 2, 3, 6, f} F	{1, 2, 3, 5, 6, f} C	{1, 2, 4, 6, f} E



NFA→DFA例子

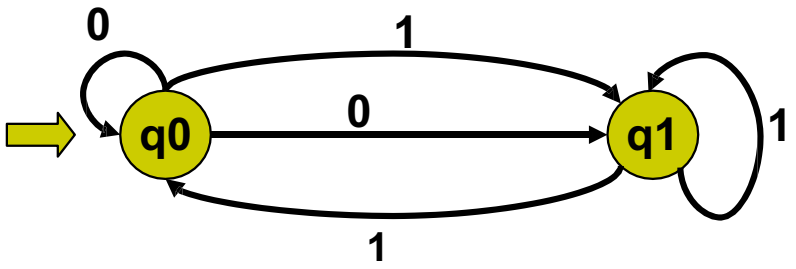
NFA M=({q0,q1}, {0,1}, f, {q0}, {q1}).

f(q0,0)={q0,q1}　f(q0,1)={q1}

f(q1,0)= Φ　f(q1,1)={q0,q1}

构造与M等价的DFA M'.

M:

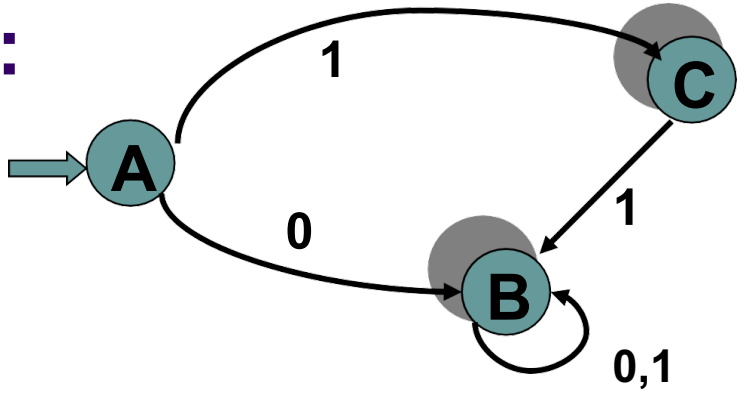


解:

令**DFA M'** = ({0,1},Q',f',q0',F')

	0	1
A {q0}	B {q0,q1}	{q1} C
B {q0,q1}	B {q0,q1}	{q0,q1} B
C {q1}	Φ	{q0,q1} B

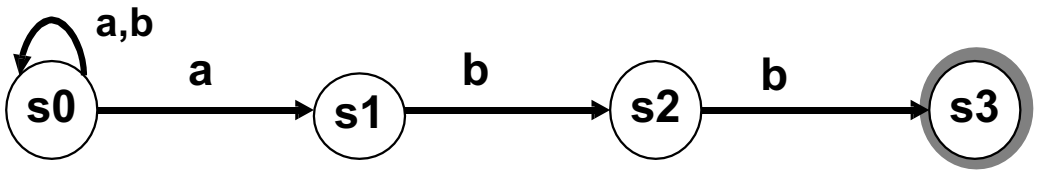
M':



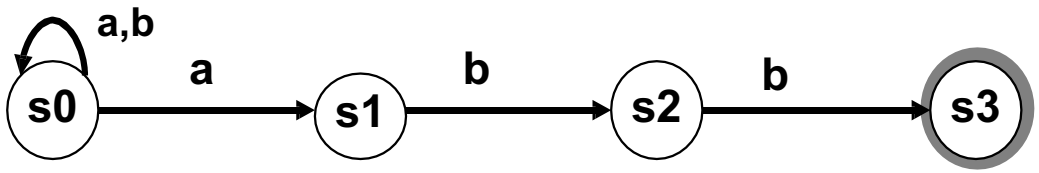
练习

NFA M=({S0,S1,S2,S3},{a,b},{S0},{S3})

将M转换成其等价的DFA



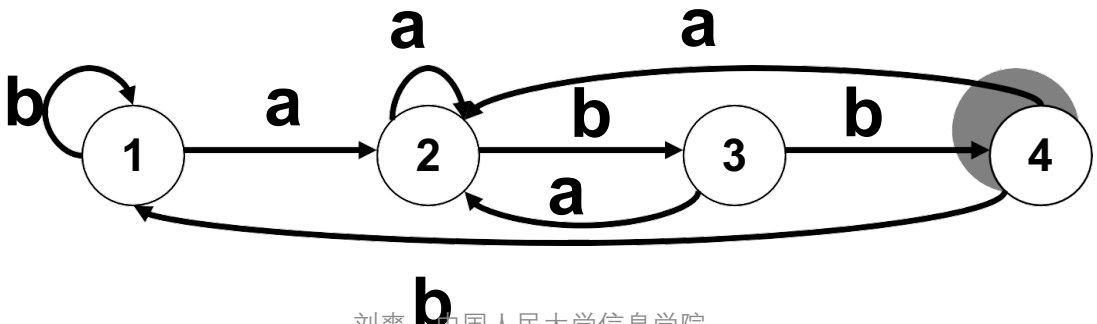
解：



状态转换表：

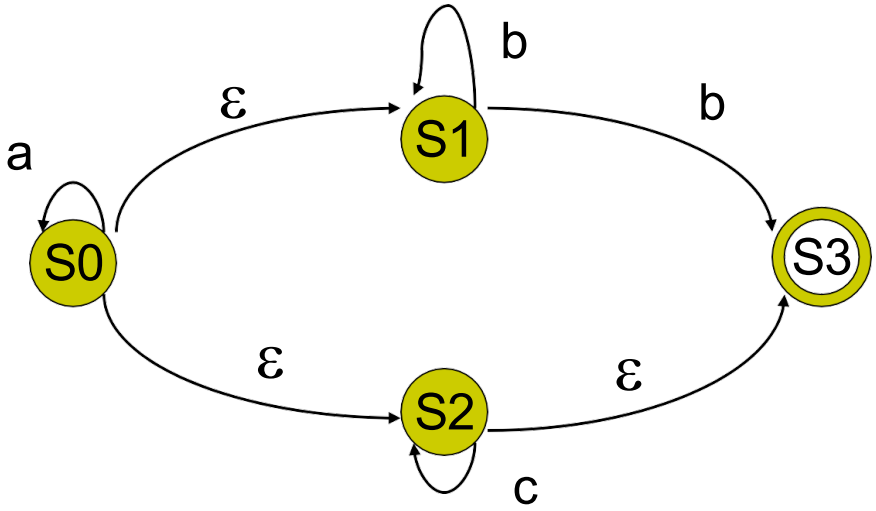
	a	b
1. {s0}	{s0,s1}	{s0}
2. {s0,s1}	{s0,s1}	{s0,s2}
3. {s0,s2}	{s0,s1}	{s0,s3}
4. {s0,s3}	{s0,s1}	{s0}

状态转换图：



练习

- 将下列ε– NFA确定化:



DFA最小化

DFA的最小化

- DFA的最小化(DFA的化简): 一个有穷自动机通过消除多余状态和合并等价状态而转换成一个最小的与之等价的有穷自动机。
  - 寻找一个状态数比M少的 DFA M’，使得L(M) = L(M’)
- 最小状态DFA
  - 不存在多余状态（死状态）
  - 不存在互相等价（不可区别）的两个状态
- 两个状态s和t等价：从s出发可以读出某个字符串w而到达终态，从t出发也可以读出字符串w而到达终态；反之亦然。
- 两个状态s和t可区别：不满足
  - 兼容性（一致性）——同是终态或同是非终态
  - 传播性（蔓延性）——从状态s出发读入某个a ($a \in \Sigma$)和从状态t出发读入某个a到达的状态等价。

DFA的最小化—算法

- 算法的核心
  - 把M的状态集K分成不相交的子集。
- 结论
  - 接受L的最小状态有穷自动机不计同构是唯一的。
- DFA M = （S, ∑, δ, s_0, S_t), 最小状态DFA M’

  1.构造状态的初始划分∏: 终态S_t 和非终态S- S_t两组

  2.对∏施用传播性原则构造新划分Π_{new}

  3. 如Π_{new} == ∏,则令 Π_{final}= ∏ 并继续步骤4，否则∏ := Π_{new}重复2

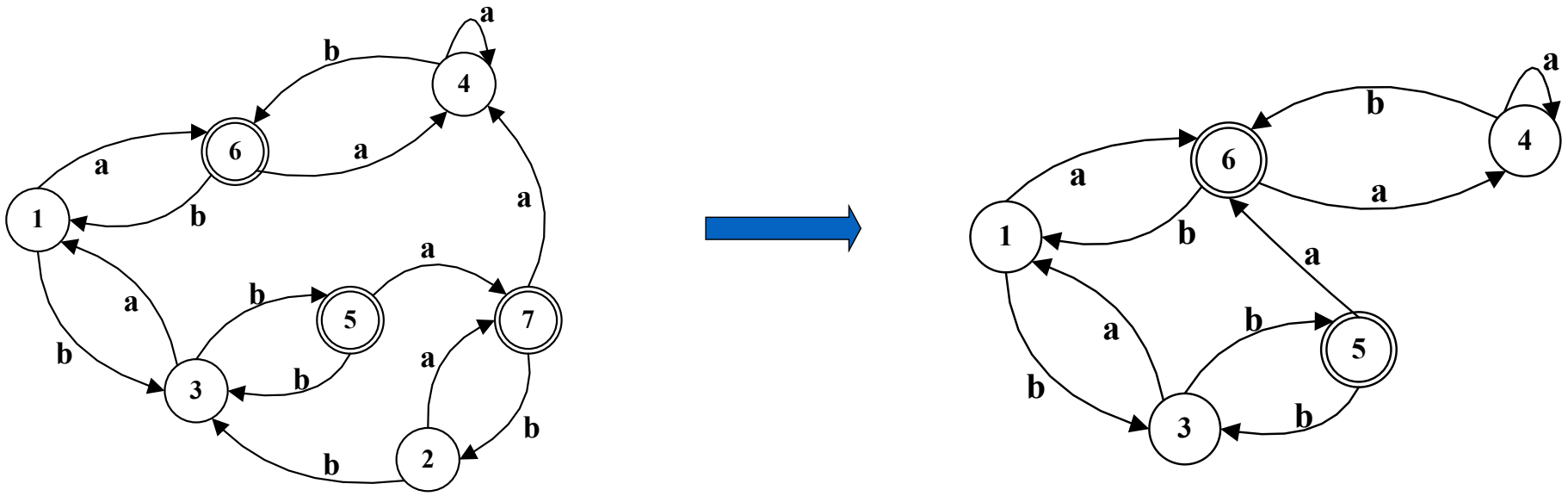
  4.为Π_{final}中的每一组选一代表，这些代表构成M’的状态。若s是一代表,且δ(s,a)=t,令r是t组的代表，则M’中有一转换δ’(s, a)=r。 M’ 的开始状态是含有s_0的那组的代表，M’的终态是含有s_t的那组的代表

  5.去掉M’中的死状态

DFA的最小化—算法（cont）

- 对Π施用传播性原则构造新划分Π_{new}步骤：
  - 假设Π被分成m个子集$\Pi=\{S_1, S_2, ..., S_m\}$, 且属于不同子集的状态是可区别的，检查$\Pi$的每一个子集$S_i$, 看是否能够进一步划分。
  - 对于某个S_i, 令Si={s1, s2, ..., sk}, 若存在数据字符a使得$\mathbf{I_a}$不全包含在现行Π的某一子集S_j中，则将S_i一分为二：即假定状态s1, s2，经过a弧分别达到状态t1, t2, 且t1, t2属于现行Π的两个不同子集，那么将S_i分成两半，一半含有s1: $S_{i1}=\{s \mid s\in S_i$ 且s经a弧到达t1所在子集中的某状态}；
    另一半含有s2: $S_{i2}=S_i-S_{i1}$
  - 由于t1, t2是可区别的，即存在w, 使得t1读出w而停于终态，t2读不出w或读出w却未停于终态。因而aw可以将s1, s2区分开。也就是说S_{i1}和S_{i2}中的状态时可区别的。
  - $\Pi_{new}=\{S_1, S_2, ... S_{i1}, S_{i2}, ..., S_m\}$

DFA的最小化—例子



(1) 初始划分：一个由终态组成，一个由非终态组成 $\prod_0=(\{1,2,3,4\},\{5,6,7\})$

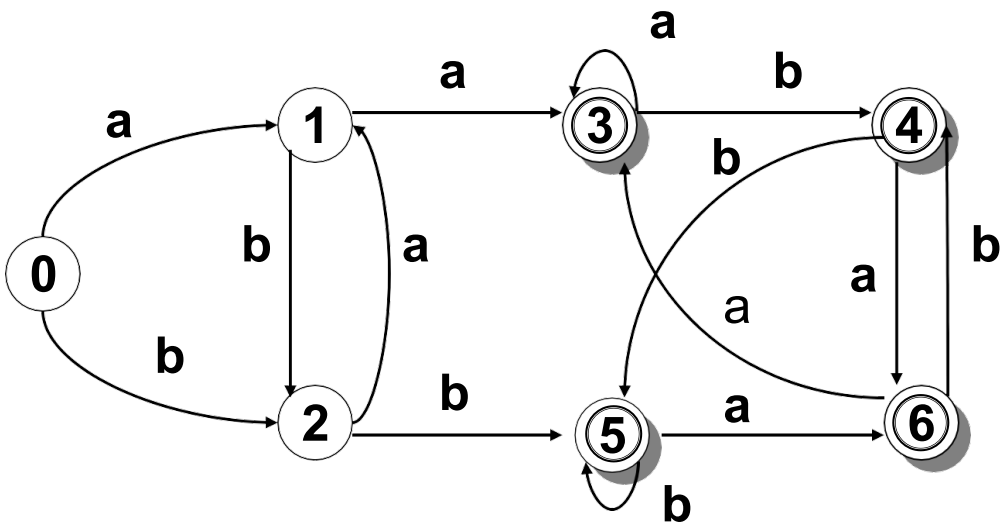
(2) 观察第一个子集，在读入a之后划分为$\prod_1=(\{1,2\},\{3,4\},\{5,6,7\})$

(3) 观察第二个子集，在读入a之后划分为$\prod_2=(\{1,2\},\{3\},\{4\},\{5,6,7\})$

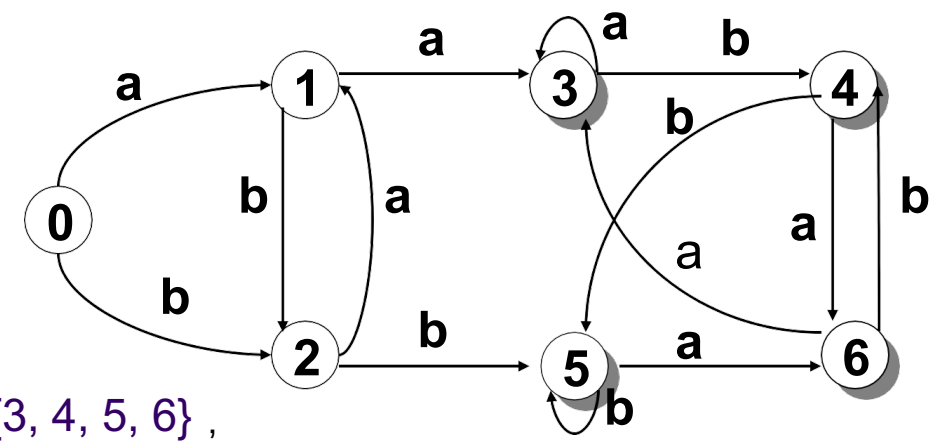
(4) 观察第四个子集，在读入a之后划分为$\prod_3=(\{1,2\},\{3\},\{4\},\{5\},\{6,7\})$

练习

- 简化下面的DFA M



解



1）初始化分Π_0={{3 , 4, 5, 6},{0, 1, 2}}

2) 观察Π_0中的第一个子集: {3, 4, 5, 6} $_a$ ={ 3, 6} ⊆ {3, 4, 5, 6} ，

{3, 4, 5,6} $_b$ ={4, 5} ⊆ {3, 4, 5, 6}不能再分。{0，1，2} $_a$={1,3} {1,3} 没有完 全包含在 {3,4,5,6}和{0,1,2}中，故{0,1,2}一分为二。

1 经a 弧到3，3 ∈{3，4，5，6};

0，2 经a 弧到1，1 ∈{0，1，2}；将{0，1，2}分成{0，2}，{1}

Π_1={{3 , 4, 5, 6},{0，2}，{1}}

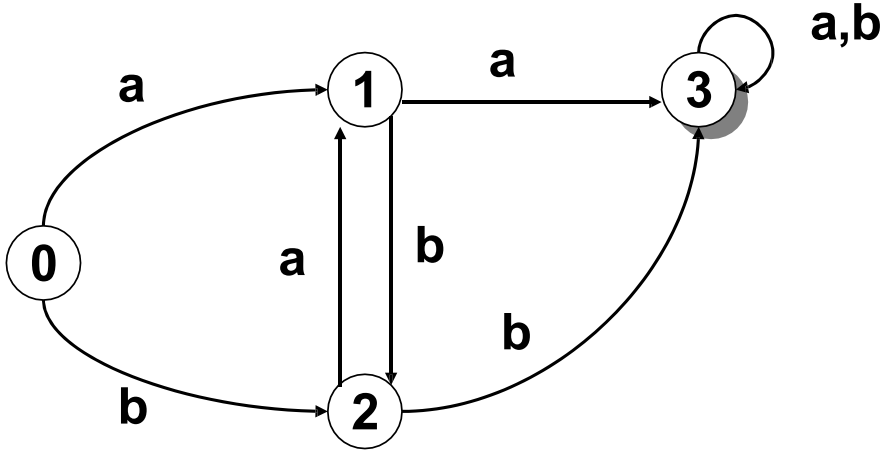
3）检查Π_1中第二个子集{0，2}：{0，2} $_b$ = {2，5}，{2，5}不完全落在{0，2}，{3，4，5}中，故{0,2}再分成{0}，{2} 得到

Π_2={{0}，{1}，{2}，{3，4，5，6}}

4) 检查Π_2中的每一个子集，均不可继续划分，故Π_{final} = Π_2={{0}，{1}，{2}，{3，4，5，6}}

解

- 令：3代表{3，4，5，6}，得到：



M = ({0,1,2,3} , {a,b} , f , 0 , {3})

f(0,a) = 1　　f(0,b)=2

f(1,a) = 3　　f(1,b)=2

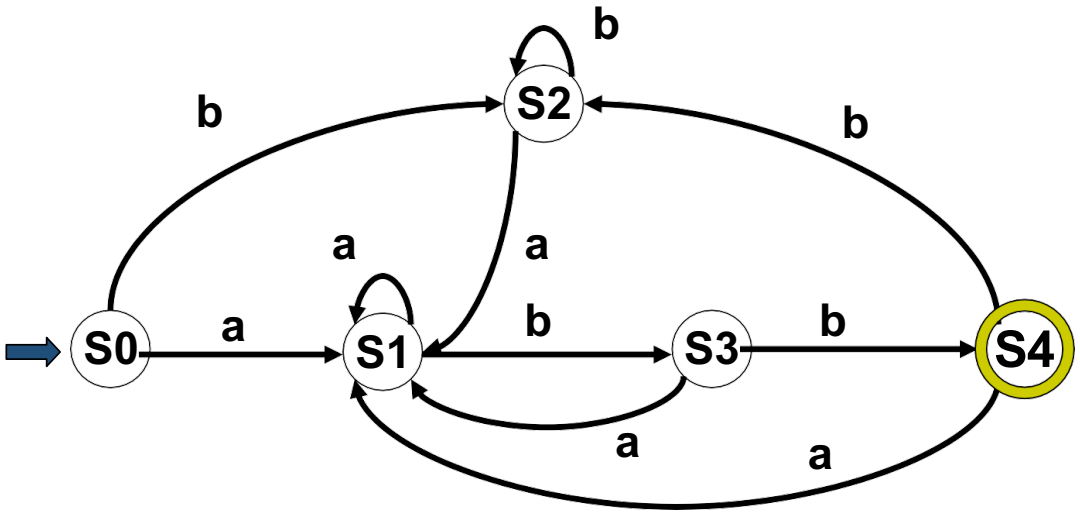
f(2,a) = 1　　f(2,b)=3

f(3,a) = 3　　f(3,b)=3

练习

- 将下面的DFA最小化：M=({S0,S1,S2,S3,S4}, {a,b},f, S0 , {S4})

	a	b
S0	S1	S2
S1	S1	S3
S2	S1	S2
S3	S1	S4
S4	S1	S2



解

1) 初始划分：{S0,S1,S2,S3},{S4}

2) 考察{S0,S1,S2,S3}: {S0,S1,S2,S3}a={S1}⊆ {S0,S1,S2,S3}

{S0,S1,S2}b={S2,S3}, {S3}b={S4} {S0,S1,S2,S3} 不包含在同一子集中。 一分为二:

NEW: {S0,S1,S2},{S3},{S4}

3)考察{S0,S1,S2}， {S0,S1,S2}a={S1} ⊆ {S0,S1,S2}

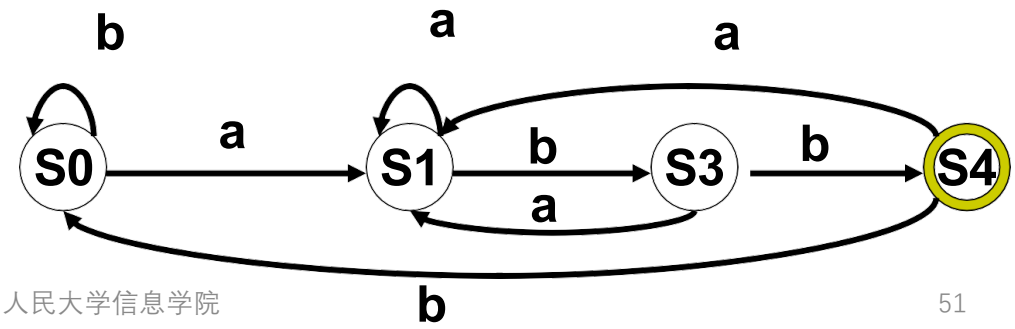
{S0,S2}b={S2}， {S1}b={S3}

{S0,S1,S2} 不包含在同一子集中。一分为二:

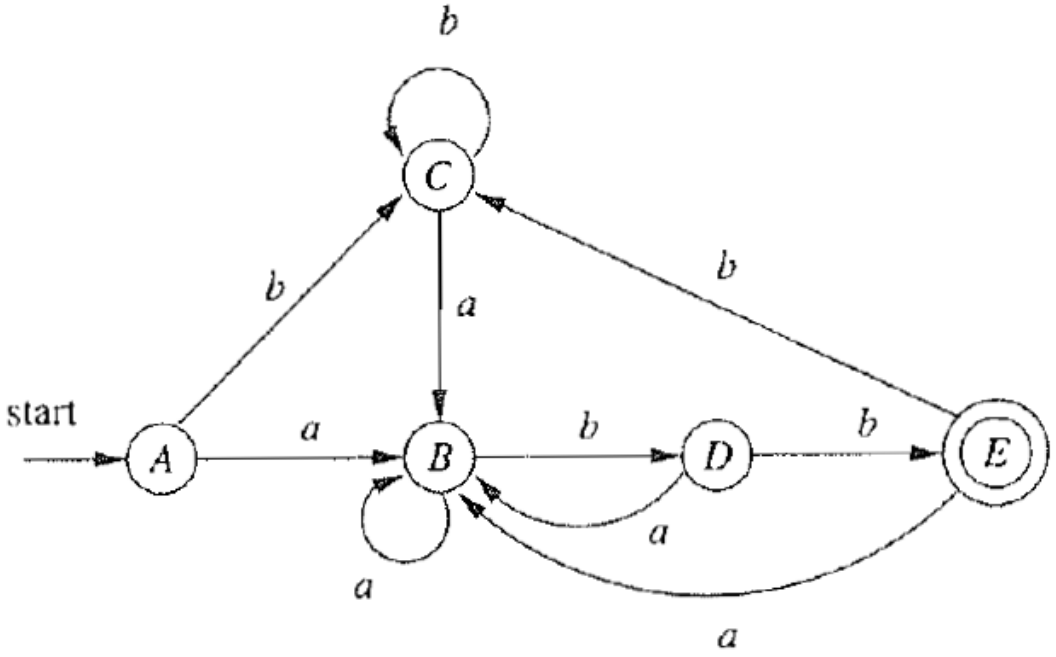
NEW :{S0,S2},{S1},{S3},{S4}

……. 直到NEW不再改变。

4)S0 作为{S0,S2} 的代表。



练习：最小化下述DFA



内容回顾

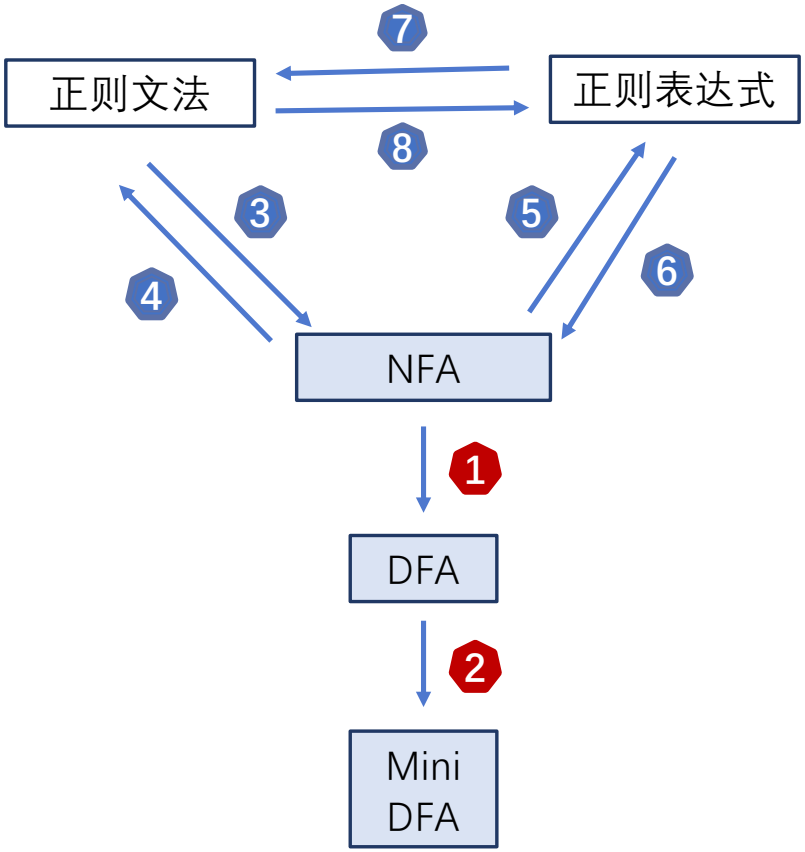
- 词法分析器的作用
- 词法分析程序的设计
  - 状态转换图
- 正则表达式和有限自动机
  - 有限自动机 （Finite Automata）
  - 正则表达式（Regular Expression）
  - 正则文法（Regular Gramma）
- 词法分析程序的自动生成

确定有限自动机 （DFA）
非确定有限自动机 （NFA）
NFA确定化
DFA最小化

内容提要

- 词法分析器的作用
- 词法分析程序的设计
  - 状态转换图
- 正则表达式和有限自动机
  - 有限自动机 （Finite Automata）
  - 正则表达式（Regular Expression）
  - 正则文法（Regular Gramma）
- 词法分析程序的自动生成

Overview

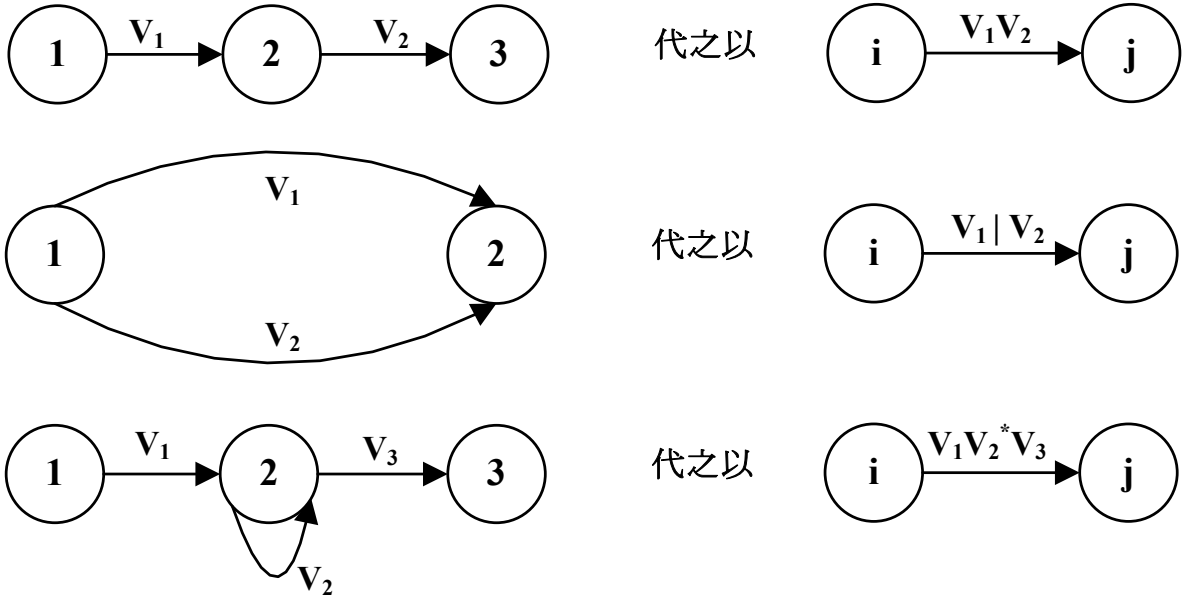


1. NFA→ DFA (NFA 确定化)
2. DFA→mini DFA (DFA最小化)
3. 正则文法→ NFA （自学）
4. DFA→正则文法（自学）
5. NFA→正则表达式
6. 正则表达式→NFA
7. 正则表达式→正则文法（自学）
8. 正则文法→正则表达式 (自学)

正则表达式和有限自动机

正则表达式和有限自动机的等价性
--有限自动机→正则表达式

- 关于正则表达式与自动机的等价性，有如下性质：
  - 对任何FA M，都存在一个正则表达式r，使得L(r)=L(M)。
  - 对任何正则表达式r，都存在一个FA M，使得L(M)=L(r)。



步骤：

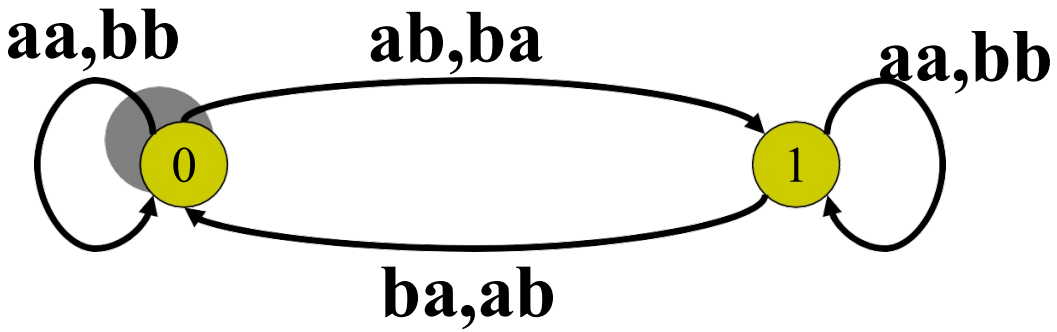
把转换图的意义拓宽,令每条弧上可以标记正则式.

1. 在给定的NFA M上加两个新状态, 一个为初态X,从X用ε弧连接M的所有初态,另一为Y,从M的所有终态用ε弧连到Y, 新的NFA M'与M等价.

2. 对M'用左侧等价规则,消去除X,Y以外的其它结.

例子

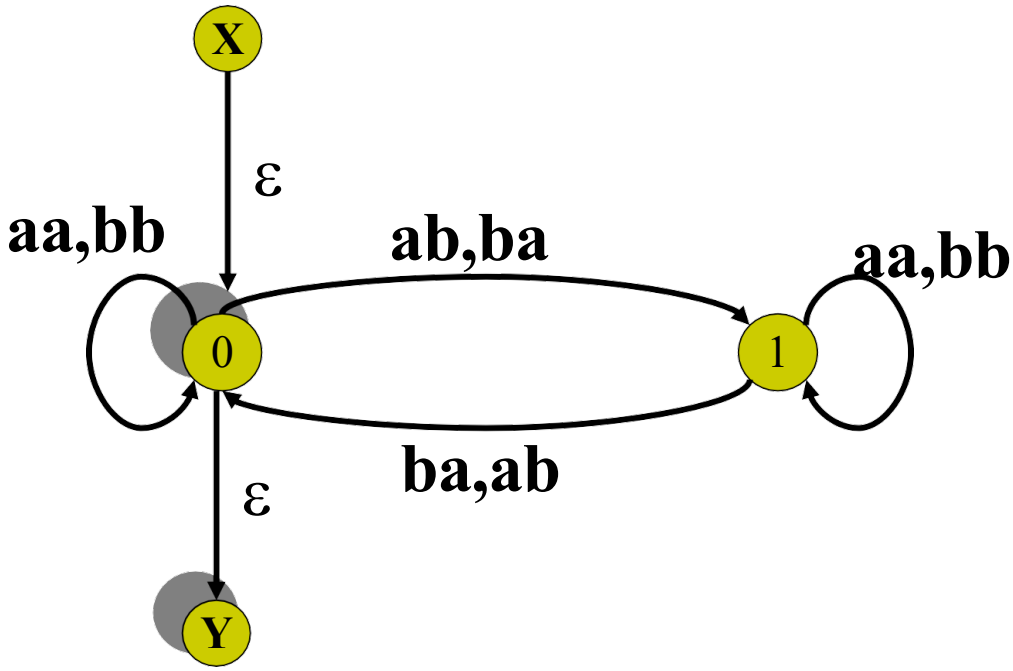
- 把下面的NFA M 转化成等价的正则表达式



NFA M 是识别具有偶数a或偶数个b的非确定有限自动机，则其初始状态是？

解

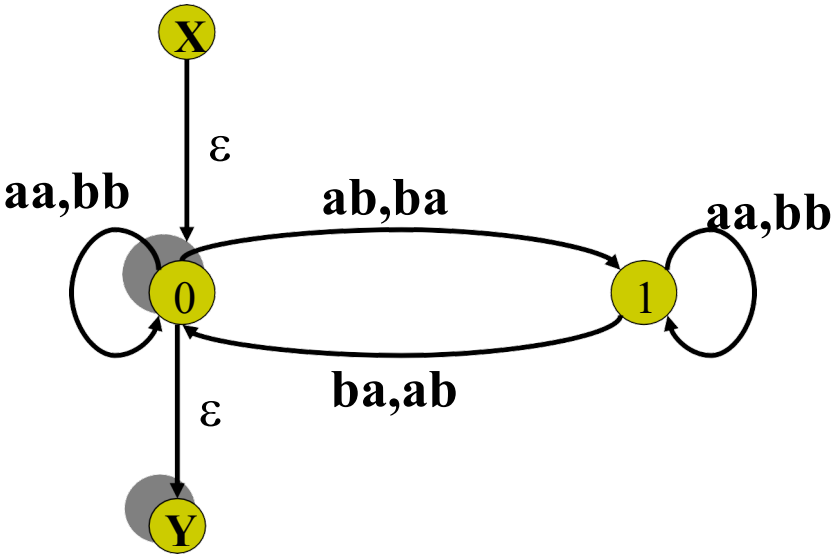
1. 在给定的NFA M上加两个新状态, 一个为初态X,从X用ε弧连接M的所有初态,另一为Y,从M的所有终态用ε弧连到Y, 新的NFA M’与M等价.

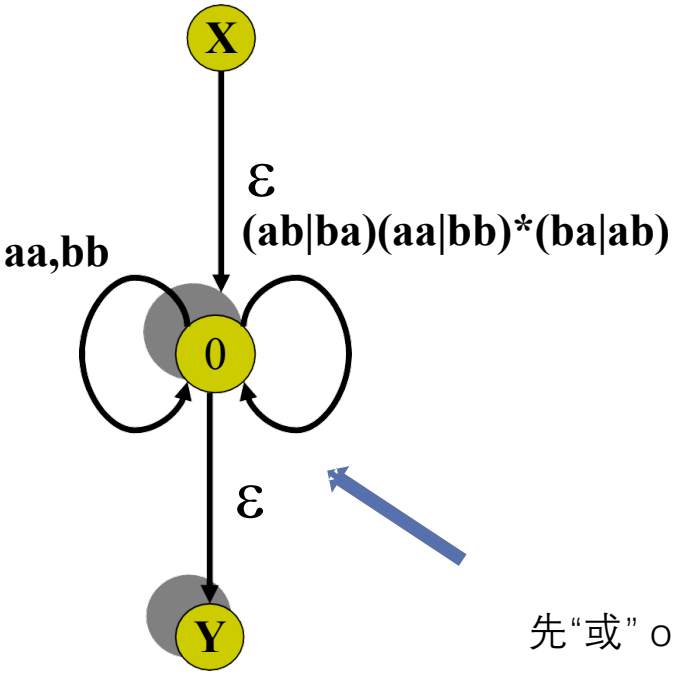


解

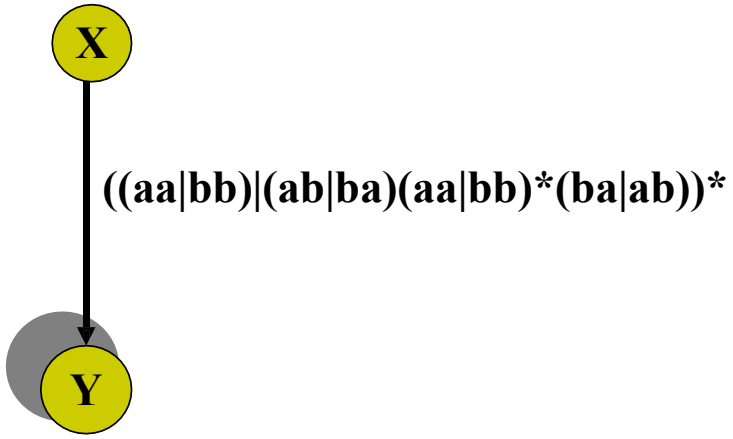
2. 对M'用等价规则,

消去除X,Y以外的其它结点.





先"或" or先 "*"？

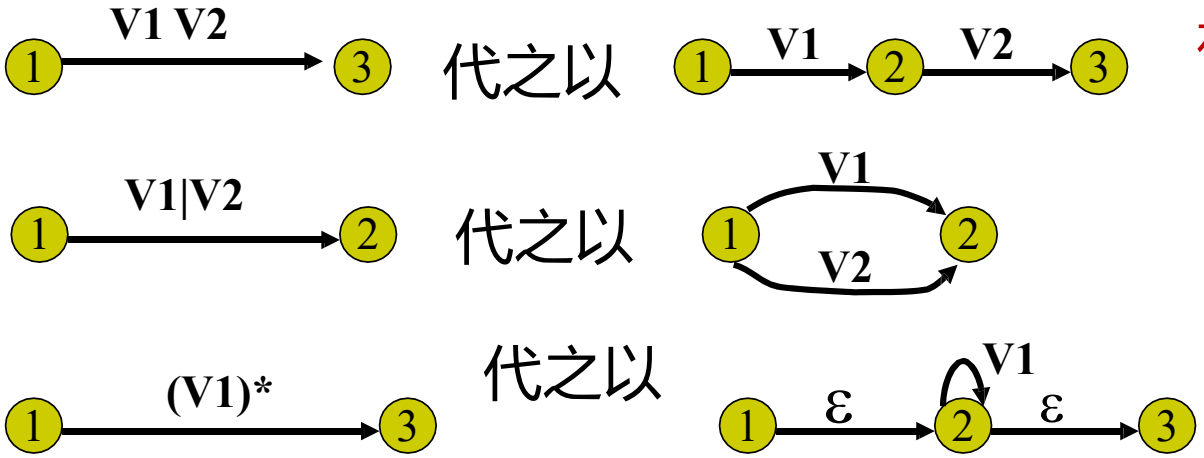


正则表达式和有限自动机的等价性

--正则表达式→有限自动机(I)

1. 对给定的正则表达式构成一个NFA M。 先写出:

2. 用以下规则对V进行分解并加进新节点



在分解过程中，要求:

（1）X,Y始终为唯一的初态和终态。

（2）所加新结点其名字彼此不同。

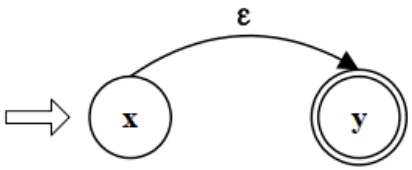
（3）弧上的标记必须是字符或空字。

正则表达式和有限自动机的等价性

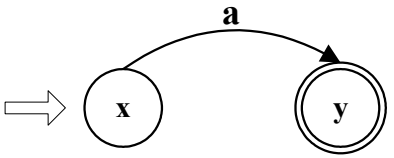
--正则表达式→有限自动机(II)

- 对于简单的正则表达式：
  - 对于正则表达式Φ，所构造的NFA为：

  - 对于正则表达式ε，构造的NFA为：



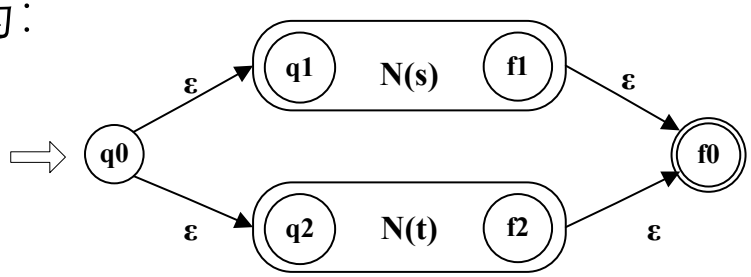
  - 对于正则表达式a，a∈∑，构造的NFA为：



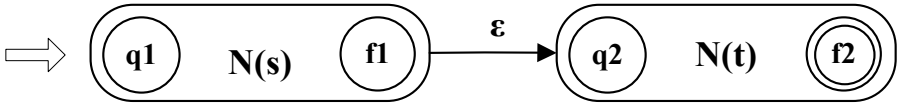
正则表达式和有限自动机的等价性

--正则表达式→有限自动机

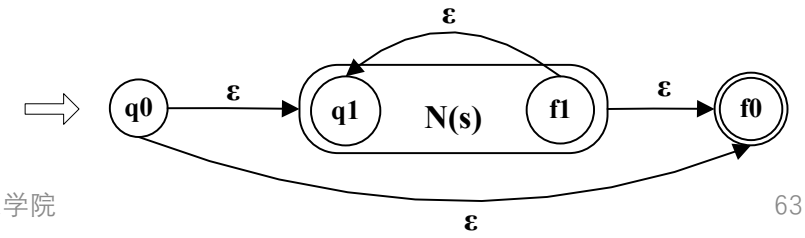
- 若s,t为∑上的正则表达式，相应的NFA分别为N(s)和N(t)，则
  - 对于正则表达式R=s | t，所构造的NFA(R)为：



  - 对于正则表达式R= st，构造的NFA(R)为：

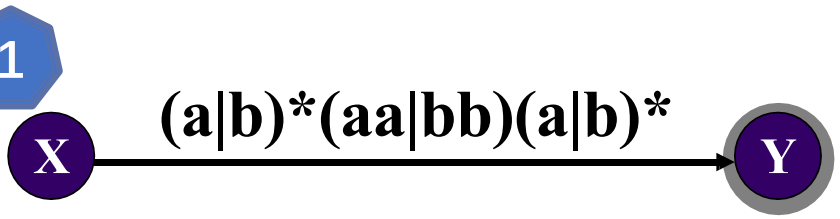


  - 对于正则表达式R=s*，构造的NFA(R)为：

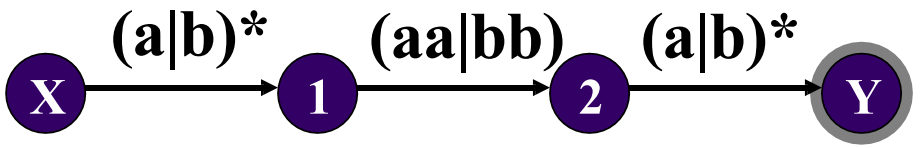


例子 （设V=(a|b)*(aa|bb)(a|b)*）

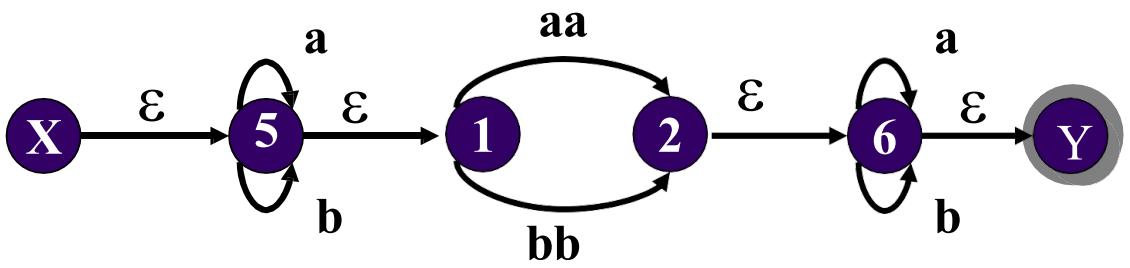
1



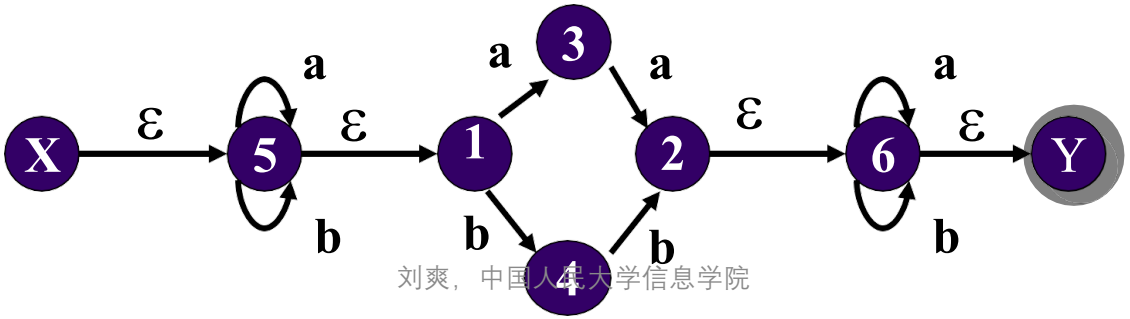
2



3



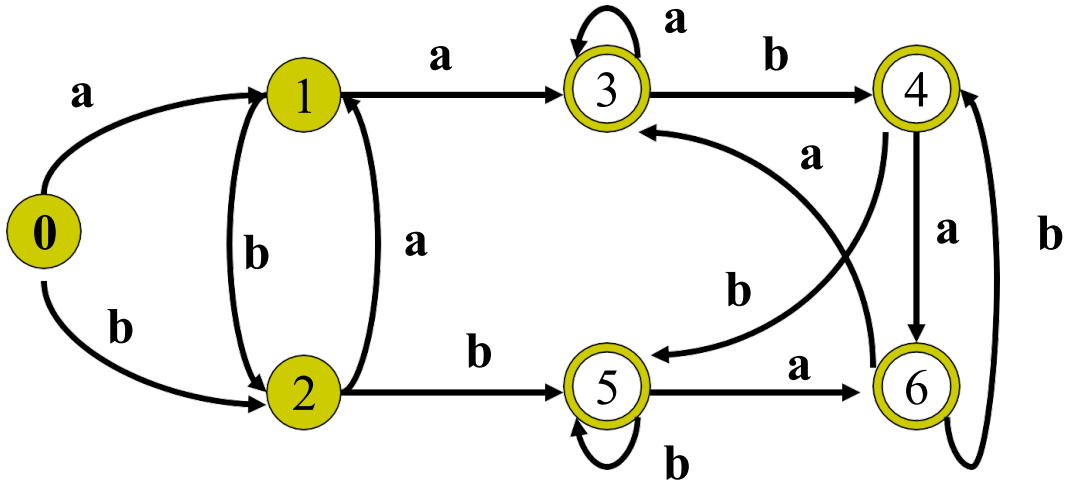
4



解

	I	Ia	Ib
0	{X,5,1}	{5,3,1}	{5,4,1}
1	{5,3,1}	{5,3,1,2,6,Y}	{5,4,1}
2	{5,4,1}	{5,3,1}	{5,3,1,2,6,Y}
3	{5,3,1,2,6,Y}	{5,3,1,2,6,Y}	{5,4,1,6,Y}
4	{5,4,1,6,Y}	{5,3,1,6,Y}	{5,3,1,2,6,Y}
5	{5,4,1,2,6,Y}	{5,3,1,6,Y}	{5,4,1,2,6,Y}
6	{5,3,1,6,Y}	{5,3,1,2,6,Y}	{5,4,1,6,Y}

解

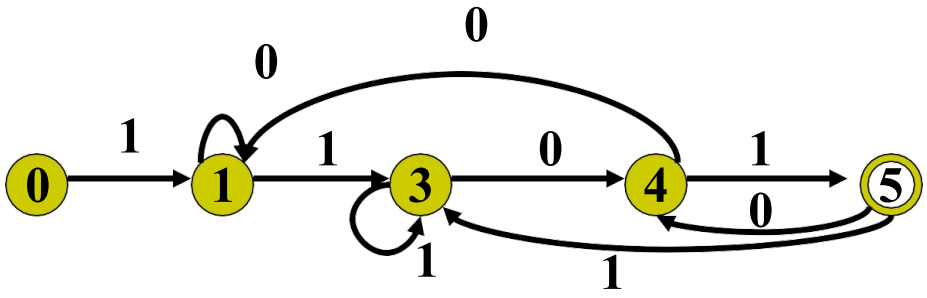


练习

构造下列正则表达式相应的DFA

1(0|1)*101

练习



构造下列正规式相应的DFA

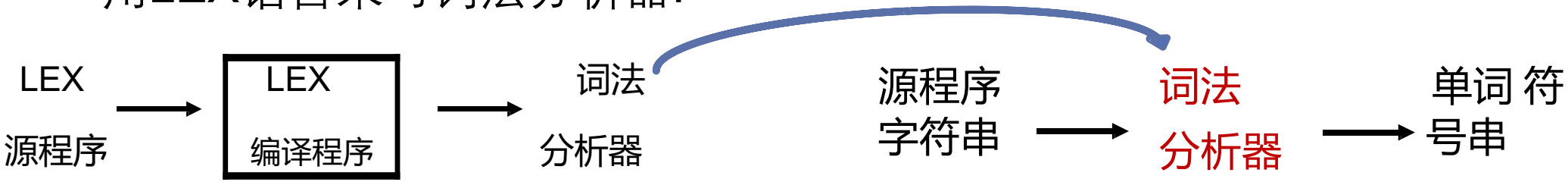
1(0|1)*101

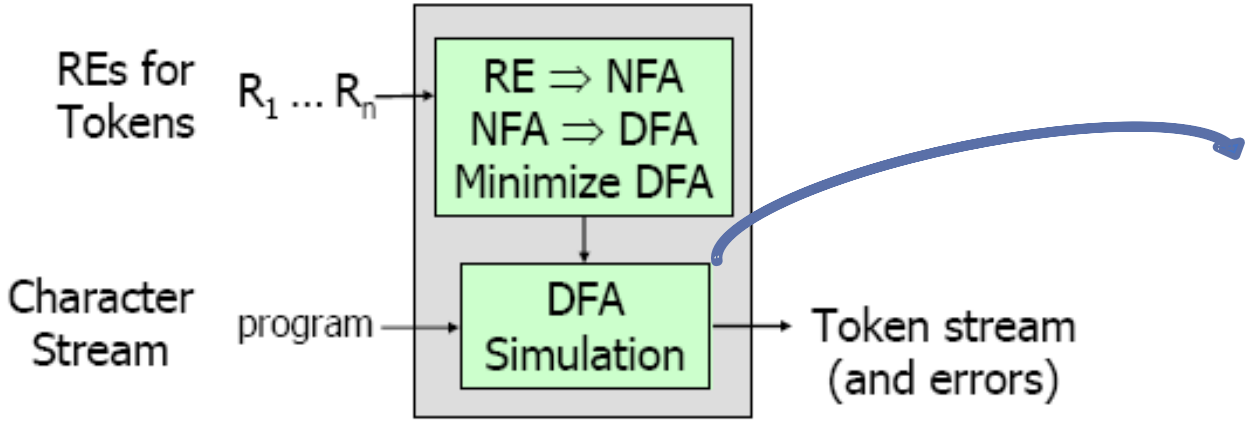
内容提要

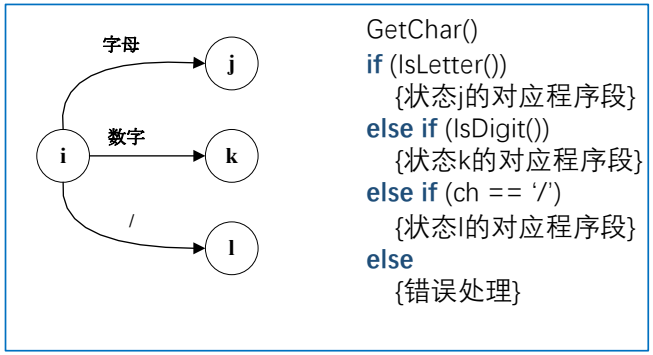
- 词法分析器的作用
- 词法分析程序的设计
  - 状态转换图
- 正规表达式和有限自动机
  - 正规表达式（Regular Expression）
  - 有限自动机 （Finite Automata）
- 词法分析程序的自动生成

词法分析器的自动生成

- 用LEX语言来写词法分析器:

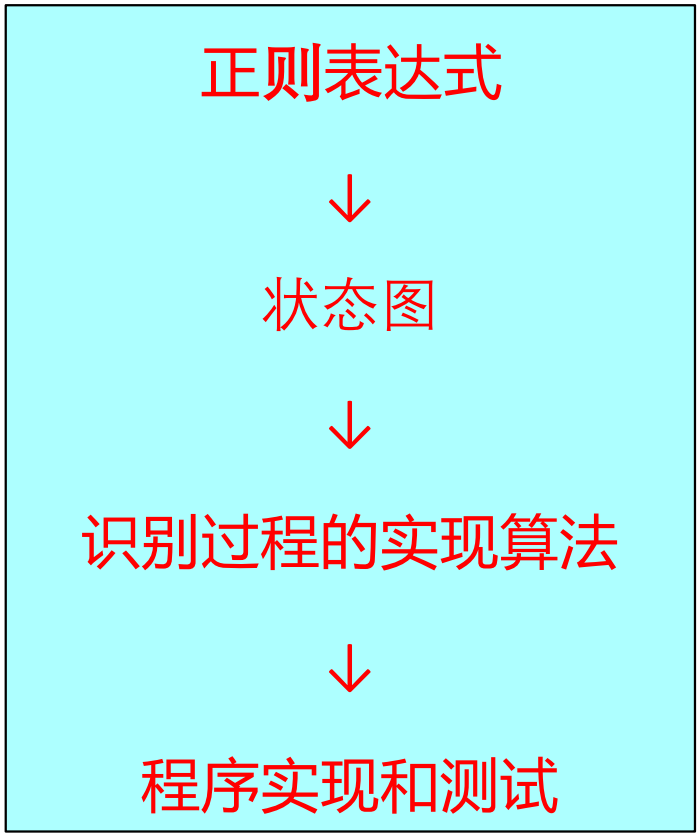






词法分析器的实现步骤

- 用正则表达式描述词法规则，设置单词种别和属性；
- 按照右图所示步骤，逐步实现词法分析器；

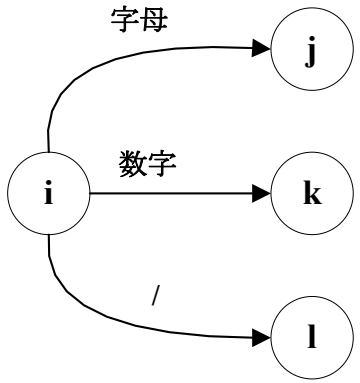


状态转换图的实现

- 程序实现:每个状态结点对应一段程序.

1) 不含回路的分叉结点:

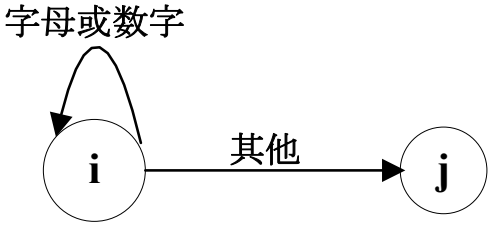
- CASE 或者 IF--THEN—ELSE




```
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) 含回路的分叉结:

- WHILE




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

3) 终点结:

- RETURN(Code, Value): 返回调用者

种别编码，属性值

LEX语言的一般介绍:

LEX程序由两部分生成:

（1）正则式（辅助）定义式

由一些LEX语句组成,形式为:

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

…

$d_m \rightarrow r_m$

其中: r_i为正则式,它定义在Σ U $\{d_1,d_2,\ldots,d_{i-1}\}$. d_i 为其简名

（2）识别规则

LEX语句组成:

P1: {A1}

P2: {A2}

…

Pn: {An}

Pi是词形,由定义在 Σ U $\{d_1,d_2,\ldots,d_n\}$ 上的正则表达式表示.

Ai是动作,当识别出Pi后应做的工作.

例子

letter ⟶ A|B|C|…|Z

digit ⟶ 0|1|2|…|9

iden ⟶ (letter|digit)*

IF： {return(1, -)}

DO： {return(2, -)}

Iden： {return(3, getEntry())}

=： {retrun(4, -)}

…

LEX工作过程

- 首先，使用LEX语言写一个定义词法分析器的源程序lex.l。
- 然后利用LEX编译器将lex.l转换成C语言程序lex.yy.c。它包括从lex.l的正则表达式构造的状态转换表以及使用该表格识别词素的标准子程序。
- 与lex.l中正则表达式相关联的动作是C代码段，这些动作可以直接加入到lex.yy.c中。
- 最后，lex.yy.c通过C编译器生成目标程序，这个目标程序就是把输入流转换成记号序列的词法分析器。

Lex词法分析器如何工作:

P1: {A1}
P2: {A2}
…
Pn: {An}

1) 最长匹配原则:

L扫描输入串,寻找最长的子串匹配某一个Pi; 并把该子串截下放入TOKEN缓冲区; 然后调用动作Ai,把表示Pi对应的二元式送给语法分析器.

2)优先匹配原则:

- 在服从最长匹配的前提下,处于前面的Pi,匹配优先权就越高.
- 解决二义性问题

3)出错处理:

在输入串中找不到与某一个Pi匹配的子串,则要报告出错.

4) Ai返回单词的种别和内部值。在LEX程序中用 RETURN(C, LEXVAL)

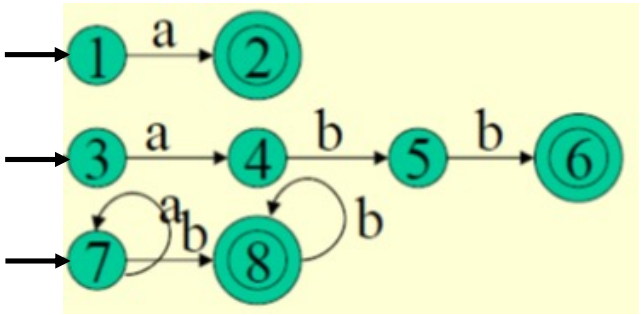
LEX举例

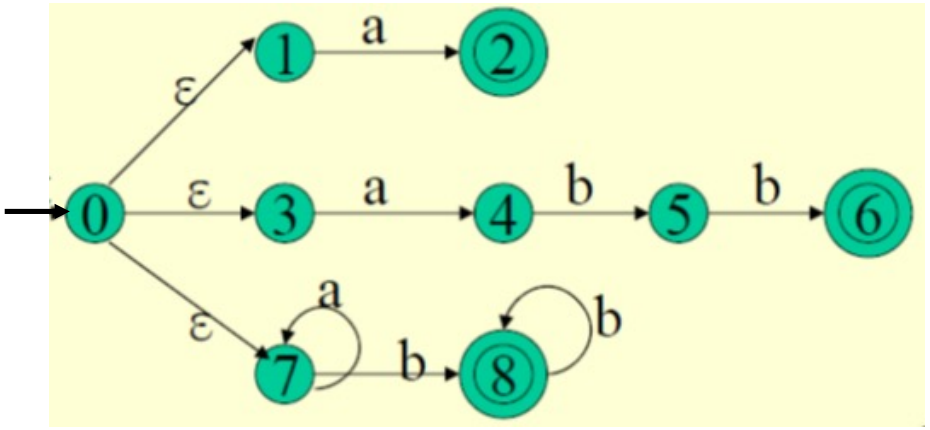
例子:

Lex 源程序

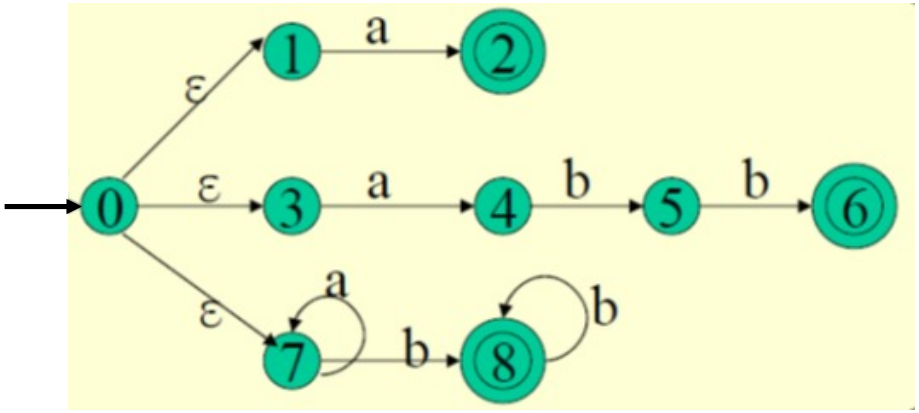
a	{ }
abb	{ }
a*bb*	{ }

读入Lex源程序，生成NFA，合并成一个NFA，确定化最小化





LEX举例



- NFA确定化: 给出状态转换表

状态	a	b	到达终态所识别的单词
0137	247	8	
247	7	58	a
8	-	8	a*bb*
7	7	8	
58	-	68	a*bb*
68	-	8	abb

自学

正则文法和有限自动机

自学

正则文法

文法 $G=(V_T, V_N, P, S)$ V_T 是非空有限集, 每个元素是一个终结符。V_N 是非空有限集, 每个元素是一个非终结符. S是一个非终结符, 是开始符号 (S在产生式的左端必须至少出现一次), P是产生式的集合。

- 如果P中 的每一个产生式的形式为 (其中, A, $B \in V_N$, $a \in V_T \cup \{\varepsilon\}$)

  $A \rightarrow aB$ 或 $A \rightarrow a$, 则称 G 是右线性文法。

- 若文法G中的每一个产生式的形式为

  $A \rightarrow Ba$ 或 $A \rightarrow a$, 则称 G是左线性文法。

右线性文法和左线性文法都称为正则文法（3型文法）, 其所产生的语言都称为正则语言 或3型语言。

自学

正则文法与有限自动机的等价性

- 对于正则文法G和有限自动机M，如果L(G)=L(M)，则称G和M是等价的。关于正则文法和有限自动机的等价问题，有以下结论：
  - 对每一个右线性正则文法G_R或左线性正则文法G_L，都存在一个有限自动机(FA) M，使得L(M) =L(G_R)=L(G_L)。
  - 对每一个有限自动机(FA) M，都存在一个右线性正则文法G_R和左线性正则文法G_L，使得L(M)=L(G_R)=(G_L)。

自学

正则文法和有限自动机的等价性

--右线性文法→有限自动机

- 右线性文法$G = (V_N, V_T, S_G, P)$, 构造 $M = (S, \Sigma, \delta, S_0, F)$:
  - 令$S=V_N \cup \{f\}$，f为一新增非终结符号且f不属于V_N；
  - 令$\Sigma = V_T$；
  - 令$S_0 = \{S_G\}$;
  - 令$F= \{f\}$;
  - 令δ如下:
    - 对于G中每一形如$A\rightarrow aB$的产生式, 其中$A,B \in V_N$ 且$a \in V_T \cup \{\varepsilon\}$, 从结点A引一条箭弧到结点B, 并用符号a标记这条箭弧，即$\delta(A, a)=B$.
    - 对于G中每一形如$A\rightarrow a$的产生式, 其中$A \in V_N$ 且$a \in V_T \cup \{\varepsilon\}$, 从结点A引一条箭弧到终态结点f,并用符号a标记这条箭弧，即$\delta(A, a)=f$.

在G中，$S_G \overset{+}{\Rightarrow} w$的充要条件是：在M中，从状态$S_0$出发到f有一条通路，其上所有箭弧的标记符号依次连接起来恰好等于w, 这就是说$w \in L(G)$当且仅当$w \in L(M)$, 故$L(G)=L(M)$

自学

例题

设给定右线性文法G[S]: $S \longrightarrow aS|bA|b$

$A \longrightarrow aS$

构造与其等价的有限状态自动机

自学

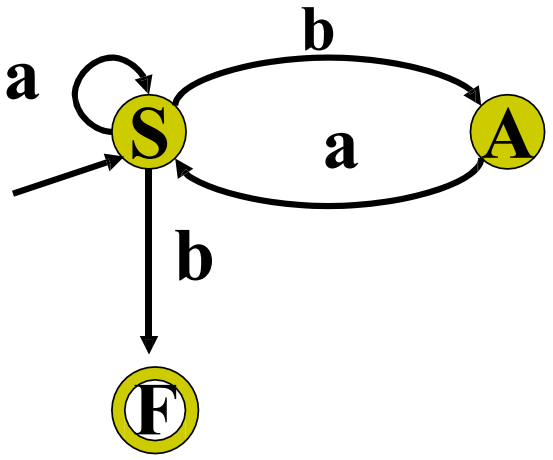
例题

设给定右线性文法G: $S \longrightarrow aS|bA|b$

$A \longrightarrow aS$

构造与其等价的有限状态自动机

M=({S, A, f}, {a, b}, δ, {S}, {f})
δ(S, a)=S,
δ(S, b)=A,
δ(S, b)=f,
δ(A, a)=S



自学

正则文法和有限自动机的等价性
--左线性文法→有限自动机

- G={V_N,V_T, P, S_G)是一个左线性文法,构造的状态转换图M=(S, Σ, δ , S_0, F) 如下:
  - 令S=V_N U { q0 }, q0为一新增非终结符号且q0不属于V_N ;
  - 令Σ = V_T ;
  - 令S_0 = {q0} ;
  - 令F= { S_G };
  - 令δ如下:
    - 对于G中每一形如A→Ba的产生式,其中A,B ∈V_N 且a ∈V_T U {ε}, 从结点B引一条箭弧到结点A,并用符号a标记这条箭弧, 即δ (B, a)=A
    - 对于G中每一形如A → a的产生式,其中A,B ∈V_N 且a ∈V_T U {ε}, 从初态q0引一条箭弧到结点A,并用符号a标记这条箭弧, δ (q0, a)=A

例题

自学

- 对于左线性文法G[S]:（{S，U}，{0，1}，P，S）其中P={S→S1，S→U1，U→U0，U→0}

构造与其等价的有限自动机

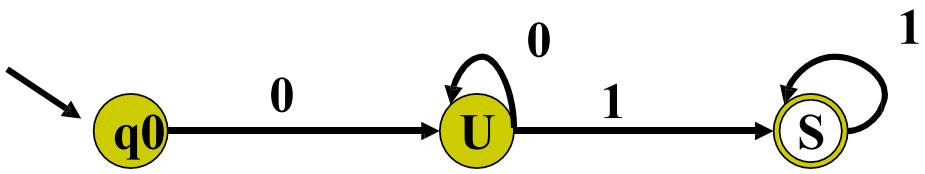
自学

例题

- 对于左线性文法G=（{S，U}，{0，1}，P，S）其中P={S→S1，S→U1，U→U0，U→0}

构造与其等价的有限自动机

M=({S, U, q0}, {0, 1}, δ, {q0}, {S})
δ(S, 1)=S,
δ(U, 1)=S,
δ(U, 0)=U,
δ(q0, 0)=U



自学

正则文法和有限自动机的等价性

--有限自动机→右线性文法

- 由DFA M = (S, Σ , δ , s_0, F)定义右线性文法G = (V_N , V_T , S_G , P):
  - 如果s_0不属于F, 令 V_N=S, V_T= Σ, S_G = s_0 , P是按下面规则定义: 对任何a ∈ Σ且A,B∈ S, 若有δ(A, a)=B
    - 当B不属于F, 令A →aB;
    - 当B ∈ F, 令A→a|aB

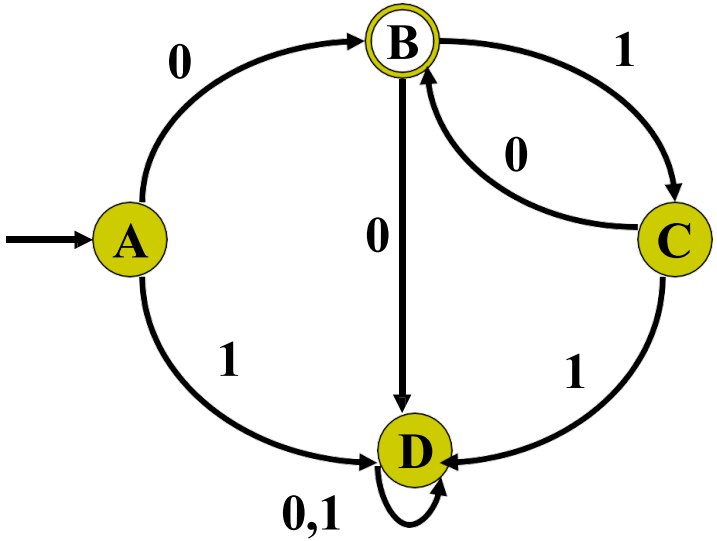
类似的从M出发可构造左线性文法。

  - 如果s_0∈F, 因为δ(s_0, ε)= s_0,所以, ε ∈ L(M), 但ε不属于上面构造的G所产生的语言L(G)。实际上L(G)=L(M)-{ε}， 因而对上面由M出发所构造的右线性正规文法G中添加一个非终结符号S'_0不属于 V_N (即V_N =S US'$_0$)和产生式S'_0→s_0| ε (即P=PU {S'_0→s_0| ε })并用 S_G = S'_0代替s_0作开始符号。这样经过 修正后的G仍是右线性正规文法且 L(G)=L(M).

自学

例题:

设DFA M=({A,B,C,D}, {0,1} ,f, A, {B})



L(M)=0(10)*.

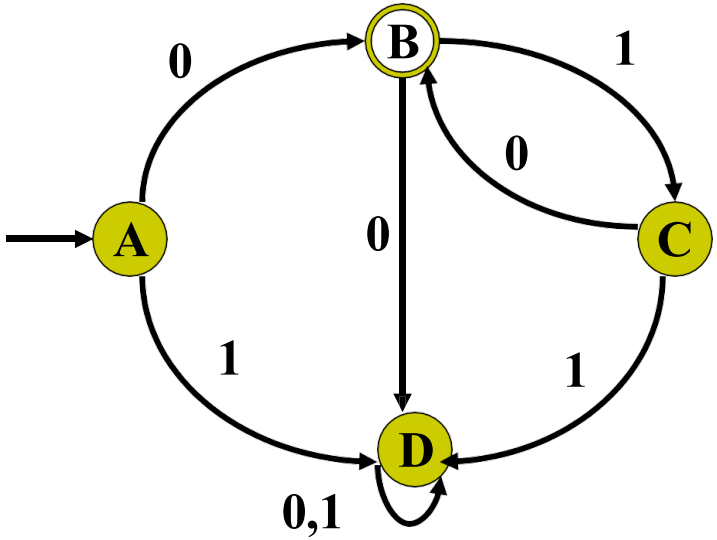
构造与其等价的右线性正规文法:

自学

解

G= ({A，B，C，D}, {0，1}，A，P)
其中P为产生式的集合：

A→0|0B|1D　　B→0D|1C
C→0|0B|1D　　D→0D|1D



L(G)=L(M)=0(10)*

自学

例题：由右线性正规文法G出发构造NFA

A→0|0B|1D　　B→0D|1C

C→0|0B|1D　　D→0D|1D

M’=({A,B,C,D,f}, {0,1}, δ,{A},{f})

其中

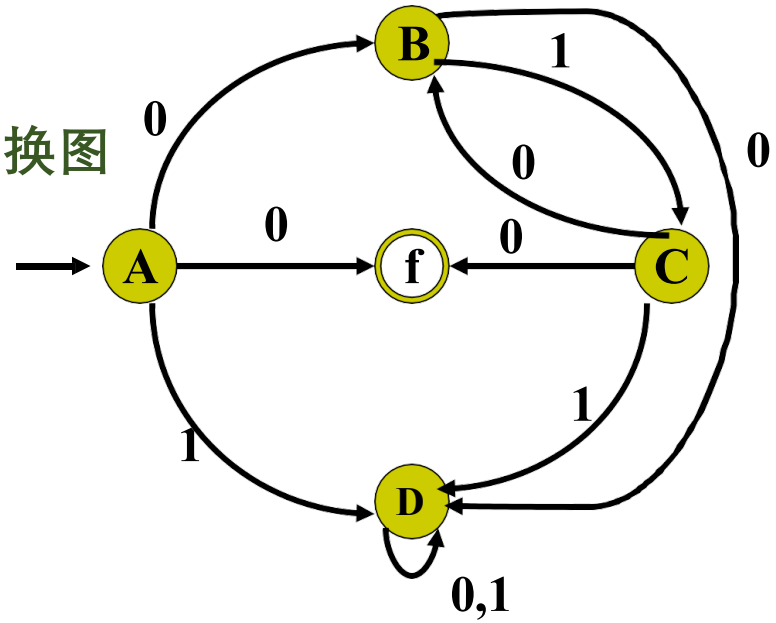
δ(A,0)={f,B}　　δ(A,1)={D}

δ(B,0)={D}　　δ(B,1)={C}

δ(C,0)={f,B}　　δ(C,1)={D}

δ(D,0)={D}　　δ(D,1)={D}

状态转换图



$$L(M') = L(M)$$

自学

例题：由NFA 构造左线性正规文法

- 从M’构造左线性文法G’的产生式P 的方法:
  - 若对任何S, S’ ∈{B, C, D, f}且a∈{0,1}有δ(S, a)=S’, 则令S’→Sa,
  - 若有δ(A, a)=S (A是M’的初态), 则令S→a,

- G’=（{B，C，D，S}，{0，1}，S，P）
- 其中S=f 为下面产生式的集合:

  S→0| C0　　C→B1　　B→0|C0　　D→1|C1|D0|D1|B0

- 有L(G’)=L(M’)=L(G)=L(M)=0(10)*

自学

练习

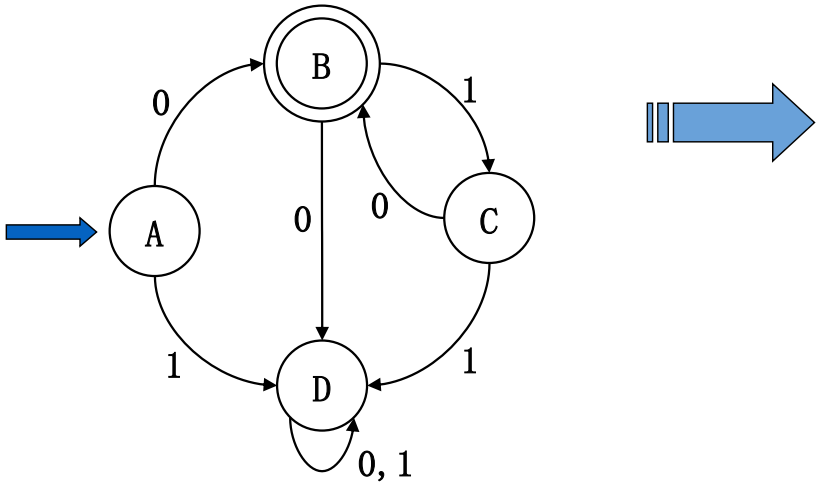


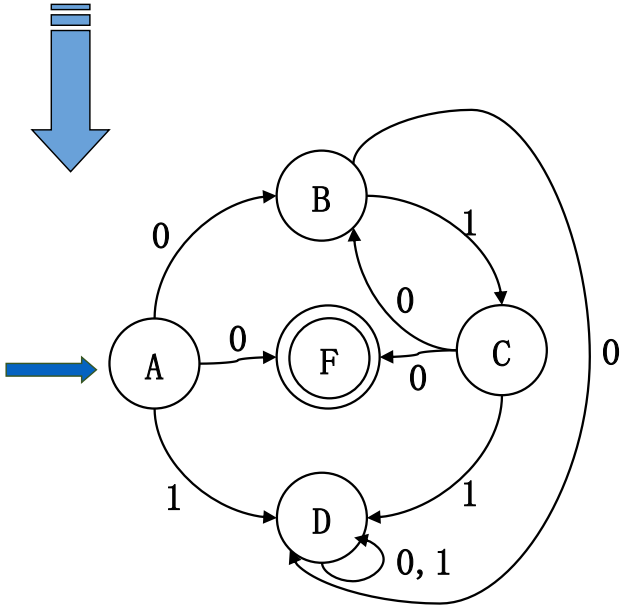
图1 DFA

正则文法

$G_R=<\{0,1\},\{A,B,C,D\},A,P>$其中P由以下产生式构成

A→0|0B|1D　　B→0D|1C

C→0|0B|1D　　D→0D|1D



（1）由图1的DFA构造与其等价的右线性文法，

（2）再从该文法出发构造等价的NFA

自学

正则表达式和正则文法

自学

正则表达式转正则文法（步骤）

- 对∑上的正则表达式U ,存在一个正则文法$G=(V_N,V_T,P,S)$: L(G)=L(U)

步骤：

初始： $V_T=\sum$, $S \in V_N$, 生成正则产生式$S \to U$

(1) 对形如 $A \to r_1 r_2$的正则产生式： $A \to r_1 B$

$B \to r_2$ $B \in V_N$

(2)对形如$A \to r^* r_1$的正则产生式： $A \to rB$

$A \to r_1$

$B \to rB$

$B \to r_1$ $B \in V_N$

(3)对形如$A \to r_1 \mid r_2$的正则产生式: $A \to r_1$

$A \to r_2$

不断应用上述规则做变换，直到每个产生式右端只含一个V_N

自学

正则表达式转正则文法举例

- 例1：r=a(a|d)*
  - V_T={a,d} S→a(a|d)*

(1) S→aA
A→(a|d)*

(2) A→(a|d)A
A→ε

(3) G[s]: S→aA
A→ε
A→aA
A→dA
} P

V_T={a, d}
V_N={S,A}
初始符号为S

初始：$V_T=\Sigma$, $S \in V_N$，生成正则产生式S→U

(1) 对形如 $A \to r_1 r_2$的正则产生式：
$A \to r_1 B$
$B \to r_2$　　$B \in V_N$

(2)对形如$A \to r^* r_1$的正则产生式：
$A \to rA$　　$A \to r_1$

(3)对形如$A \to r_1 | r_2$的正则产生式：
$A \to r_1$　$A \to r_2$

不断应用上述规则做变换，直到每个产生式右端只含一个V_N

自学

正则文法转正则表达式*

- 对G=(V_N,V_T,P,S),存在一个$\Sigma = V_T$上的正则表达式r : L(r)=L(G)

	正则文法	**正则表达式**
规则1	A→xB, B→y	A=xy
规则2	A→xA | y	A=x*y
规则3	A→x, A→y	A=x | y

自学

正则文法转正则表达式举例

- G[s]: S→aA
  A→aA
  A→dA
  S→a
  A→a
  A→d

① S→aA|a
② A→aA|a|dA|d
③ A→(a|d)A|(a|d)
④ A→(a|d)*(a|d)

S=a(a|d)*(a|d)|a =a((a|d)*(a|d)|ε) =a((a|d)+|ε)

规则1 A→xB, B→y A=xy
规则2 A→xA|y A=x*y
规则3 A→x, A→y A=x|y

小结

- 词法分析器的作用
- 词法分析程序的设计
  - 状态转换图
- 正规表达式和有限自动机
  - 有限自动机 （Finite Automata）
  - 正则表达式（Regular Expression）
  - 正则文法（Regular Gramma）
- 词法分析程序的自动生成

阅读材料：《程序设计语言编译原理（第3版）》，陈火旺等编著，国防工业出版社， 2004年----第三章