编译原理

第三章 词法分析

第三章 词法分析

- 对于词法分析器的要求
- 词法分析器的设计
- 正规表达式与有限自动机
- 词法分析器的自动产生 --LEX

第三章 词法分析

- 对于词法分析器的要求
- 词法分析器的设计
- 正规表达式与有限自动机
- 词法分析器的自动产生 --LEX

回顾

DIM,IF, DO,STOP,END
number, name, age
125, 2169
…

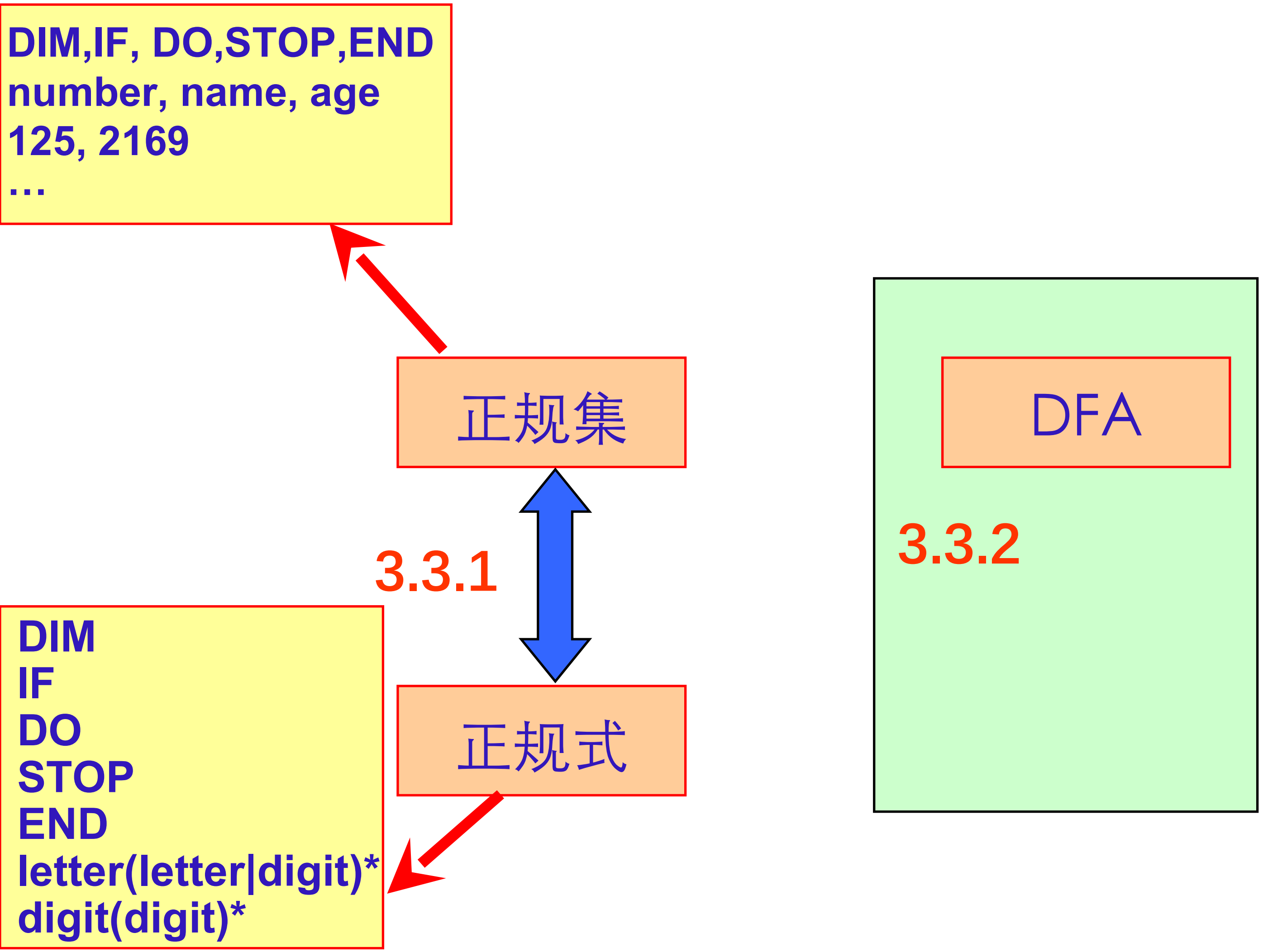
正规集

3.3.1

正规式

DIM
IF
DO
STOP
END
letter(letter|digit)*
digit(digit)*

关系图


DIM,IF, DO,STOP,END
number, name, age
125, 2169
…
正规集
3.3.1
正规式
DIM
IF
DO
STOP
END
letter(letter|digit)*
digit(digit)*
DFA
3.3.2

确定有限自动机(DFA)

- 对状态图进行形式化，则可以下定义：

确定有限自动机(DFA)M 是一个五元式 $M=(S, \Sigma, f, S_0, F)$，其中：

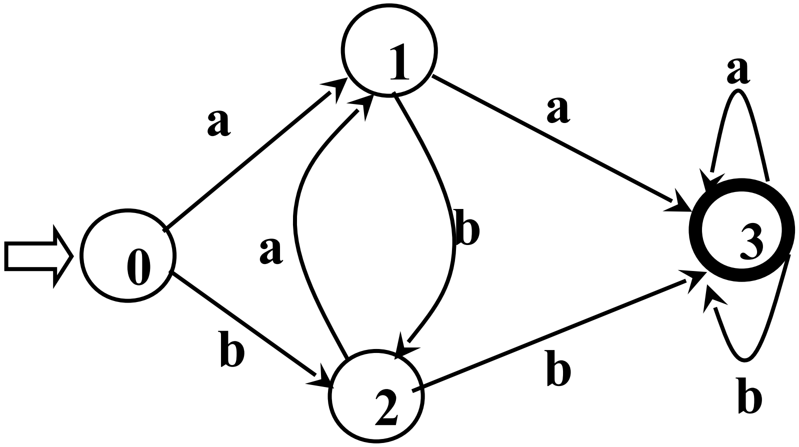
1. S: 有穷状态集
2. Σ： 输入字母表(有穷)
3. f: 状态转换函数，为 $S\times\Sigma\rightarrow S$ 的单值部分映射，$f(s, a)=s'$ 表示：当现行状态为 s，输入字符为 a 时，将状态转换到下一状态 s'，s' 称为 s 的一个后继状态
4. $S_0\in S$ 是唯一的一个初态
5. $F\subseteq S$ ：终态集(可空)

- 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

关系图

```
curState = 初态
GetChar();
while( stateTrans[curState][ch] 有定义 ){
    // 存在后继状态，读入、拼接
    Concat();
    // 转换入下一状态，读入下一字符
   curState= stateTrans[curState][ch];
    if cur_state 是终态 then  返回 strToken 中的单
    GetChar( );
}
```

DIM,IF, DO,STOP,END
number, name, age
125, 2169,
…

正规集

3.3.1

正规式

DIM
IF
DO
STOP
END
letter(letter|digit)*
digit(digit)*

FA

DFA

3.3.2

3.3.3

NFA

3.3.3 非确定有限自动机 (NFA)

- 1976 年图灵奖
  - For their joint paper "Finite Automata and Their Decision Problem," which introduced the idea of nondeterministic machines, which has proved to be an enormously valuable concept. Their (Scott & Rabin) classic paper has been a continuous source of inspiration for subsequent work in this field.

M. O. Rabin*
D. Scott†

Finite Automata and Their Decision Problems‡

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

Introduction

Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an *a priori* upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.

In the last few years the idea of a *finite automaton* has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications.

Many equivalent forms of the idea of finite automata have been published. One of the first of these was the definition of "nerve-nets" given by McCulloch and Pitts.[3] The theory of nerve-nets has been developed by authors too numerous to mention. We have been particularly influenced, however, by the work of S. C. Kleene[2] who proved an important theorem characterizing the possible action of such devices (this is the notion of "regular event" in Kleene's terminology). J. R. Myhill, in some unpublished work, has given a new treatment of Kleene's results and this has been the actual point of departure for the investigations presented in this report. We have not, however, adopted Myhill's use of directed graphs as a method of viewing automata but have retained throughout a machine-like formalism that permits direct comparison with Turing machines. A neat form of the definition of automata has been used by Burks and Wang[1] and by E. F. Moore,[4] and our point of view is closer to theirs than it is to the formalism of nerve-nets. However, we have adopted an even simpler form of the definition by doing away with a complicated output function and having our machines simply give "yes" or "no" answers. This was also used by Myhill, but our generalizations to the "nondeterministic," "two-way," and "many-tape" machines seem to be new.

In Sections 1-6 the definition of the one-tape, one-way automaton is given and its theory fully developed. These machines are considered as "black boxes" having only a finite number of internal states and reacting to their environment in a deterministic fashion.

We center our discussions around the application of automata as devices for defining sets of tapes by giving "yes" or "no" answers to individual tapes fed into them. To each automaton there corresponds the set of those tapes "accepted" by the automaton; such sets will be referred to as *definable sets*. The structure of these sets of tapes, the various operations which we can perform on these sets, and the relationships between automata and defined sets are the broad topics of this paper.

After defining and explaining the basic notions we give, continuing work by Nerode,[5] Myhill, and Shepherdson,[7] an intrinsic mathematical characterization of definable sets. This characterization turns out to be a useful tool for both proving that certain sets are definable by an automaton and for proving that certain other sets are not.

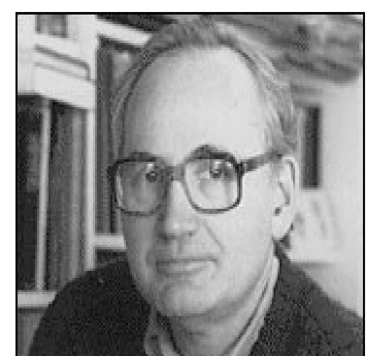
In Section 4 we discuss decision problems concerning automata. We consider the three problems of deciding whether an automaton accepts any tapes, whether it ac-

*Now at the Department of Mathematics, Hebrew University in Jerusalem.
†Now at the Department of Mathematics, University of Chicago.
‡The bulk of this work was done while the authors were associated with the IBM Research Center during the summer of 1957.

114

IBM JOURNAL • APRIL 1959

Michael O. Rabin



Dana S. Scott

3.3.3 非确定有限自动机 (NFA)

- 定义：一个非确定有限自动机 (NFA) M 是一个五元式 $M=(S, \Sigma, f, S_0, F)$，其中：

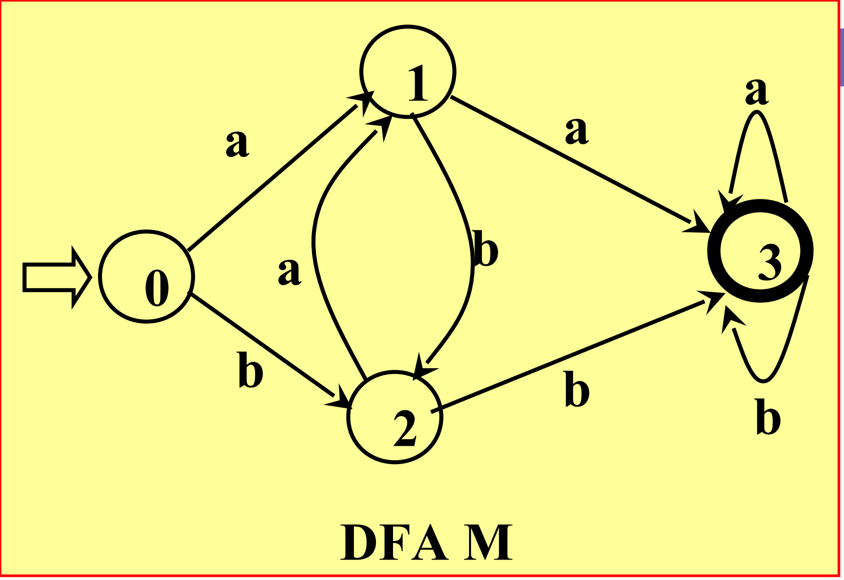
1 S：有穷状态集

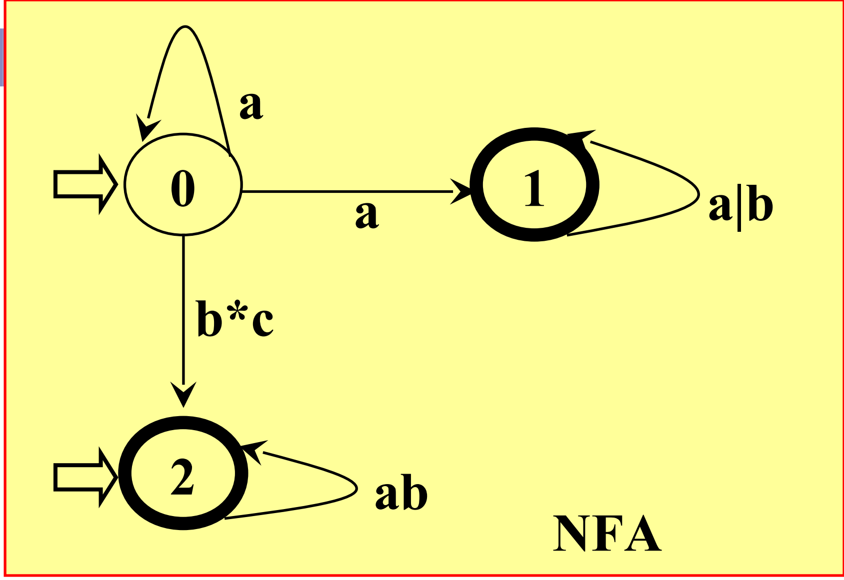
2 Σ ：输入字母表（有穷）

3 f：状态转换函数，为 $S\times\Sigma^* \rightarrow 2^S$ 的部分映射

4 $S_0 \subseteq S$ 是非空的初态集

5 $F \subseteq S$ ：终态集（可空）

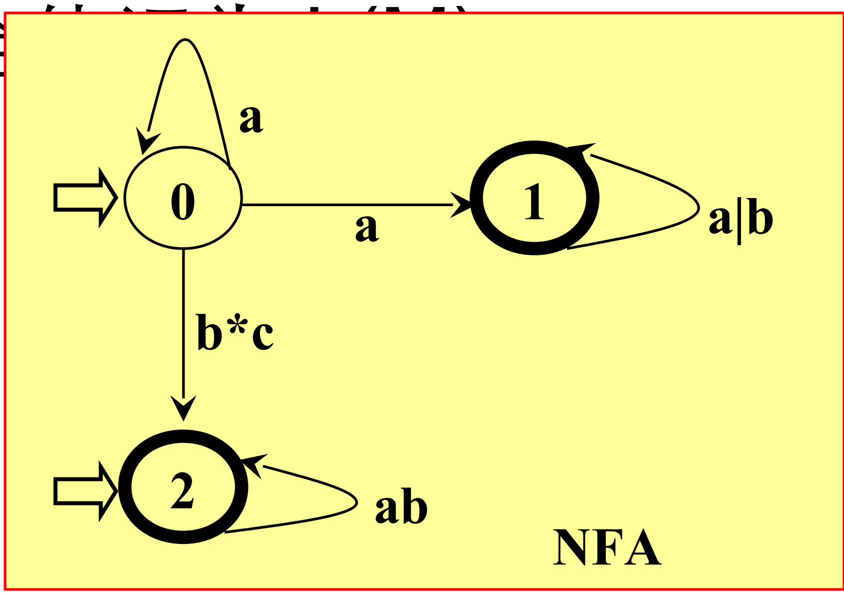




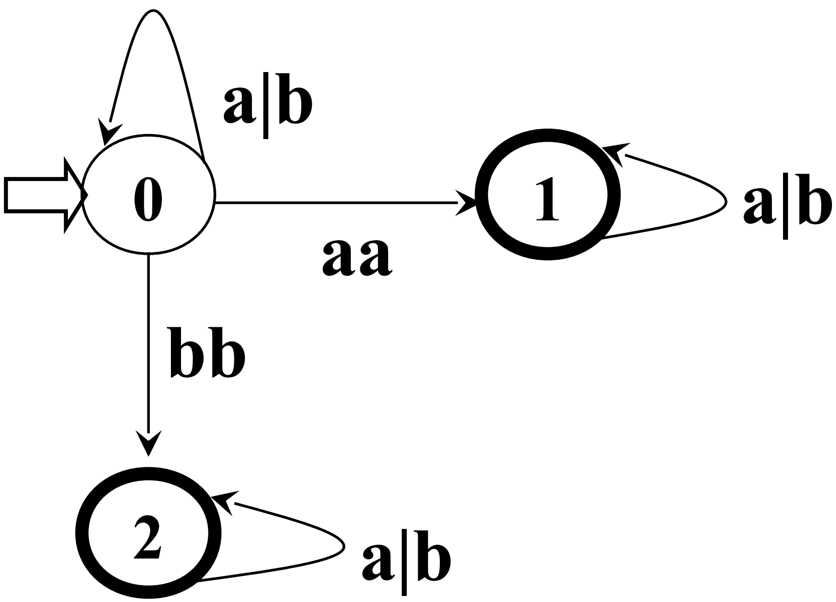
- 从状态图看 NFA 和 DFA 的区别
  - 可以有多个初态
  - 弧上的标记可以是Σ * 中的一个字（甚至可以是一个正规式），而不一定是单个字符
  - 同一个字可能出现在同状态射出的多条弧上
- DFA 是 NFA 的特例

- 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字连接成的字等于α（忽略那些标记为ε的弧），则称α为 NFA M 所识别（接收）

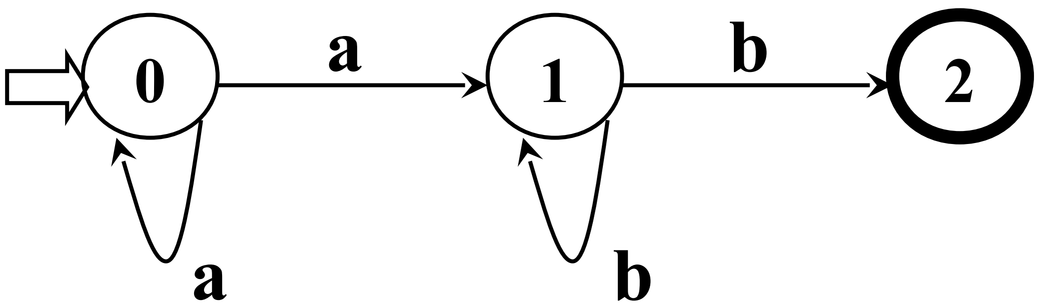
- NFA M 所识别的字的全



NFA 示例



识别所有含相继两个 a 或相继两个 b 的字



$\{a^m b^n \mid m, \quad n \geq 1\}$

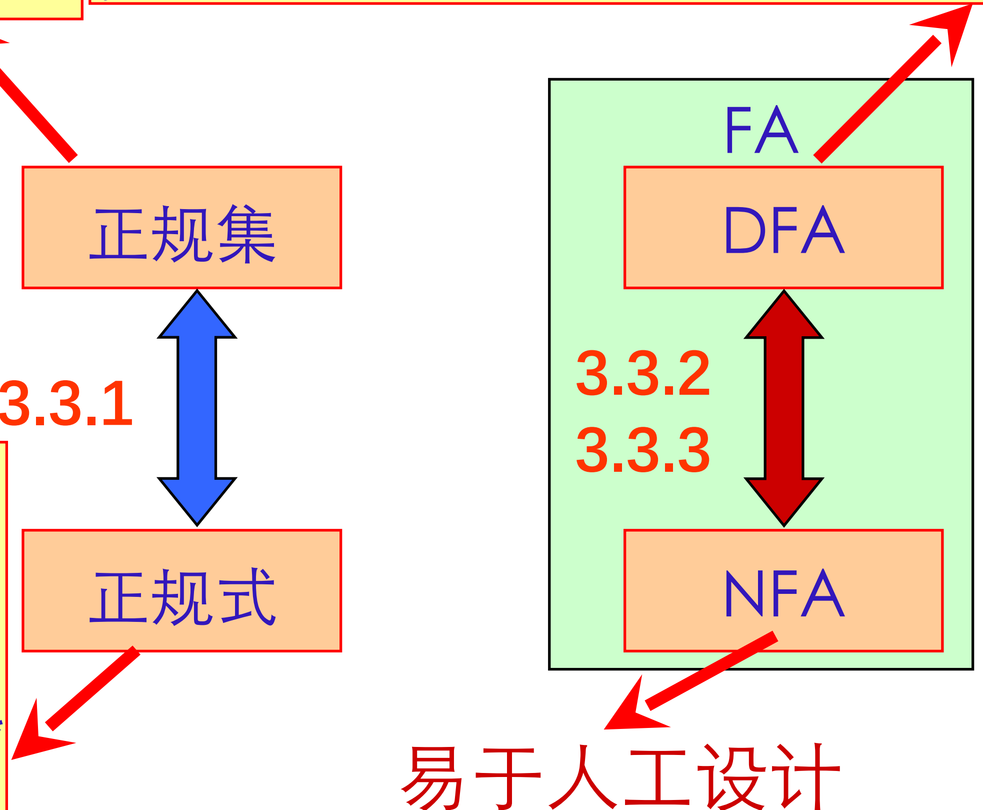
NFA 与 DFA

- 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’)
- DFA 与 NFA 描述能力相同！

关系图

```
curState = 初态
GetChar();
while( stateTrans[curState][ch] 有定义 ){
    // 存在后继状态，读入、拼接
    Concat();
    // 转换入下一状态，读入下一字符
   curState= stateTrans[curState][ch];
    if cur_state 是终态 then  返回 strToken 中的单
    GetChar( );
}
```

DIM,IF, DO,STOP,END
number, name, age
125, 2169
…



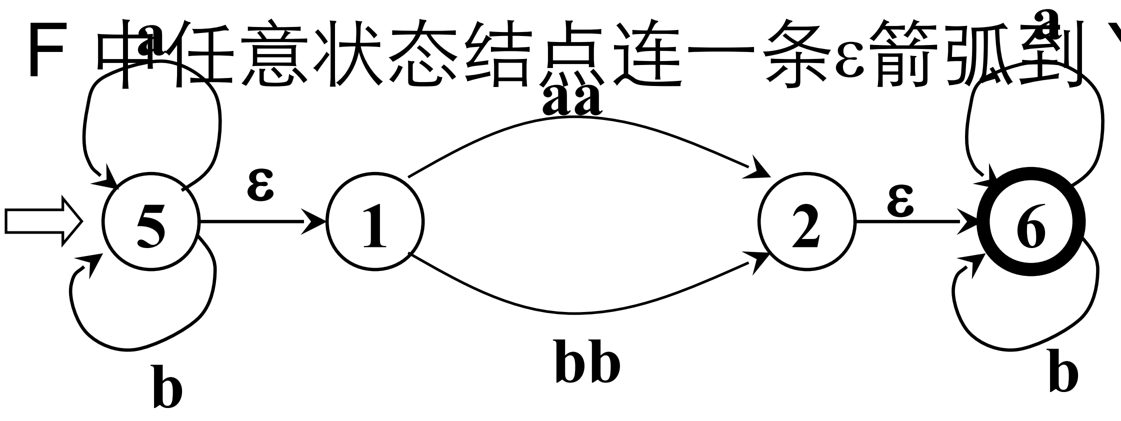
DIM
IF
DO
STOP
END
letter(letter|digit)*
digit(digit)*

证明：

1. 假定 NFA M=<S, Σ, δ, S_0, F>， 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 。

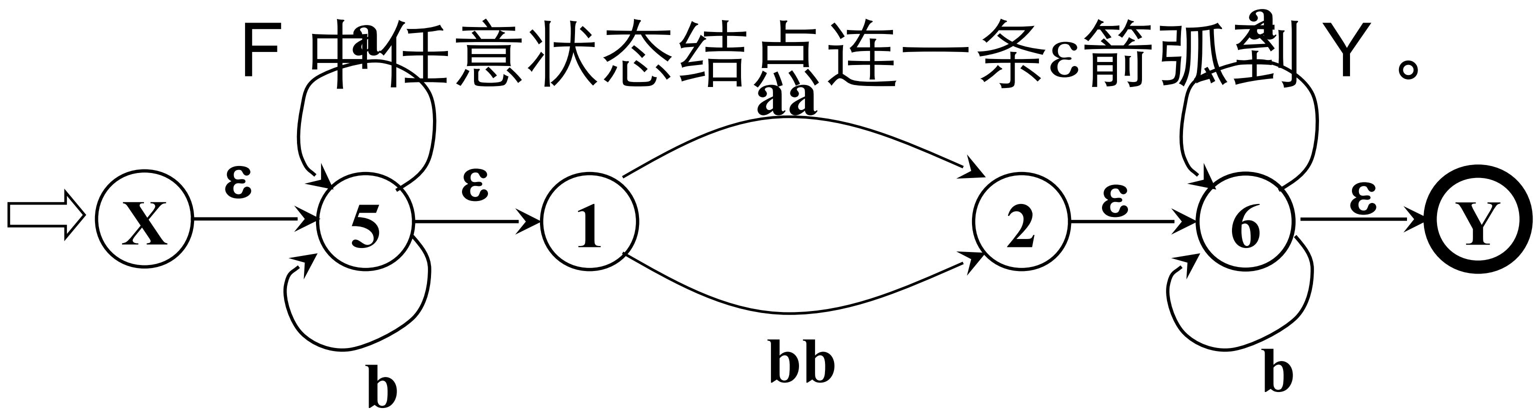


证明：

1. 假定 NFA M=<S, Σ, δ, S_0, F>，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∉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。



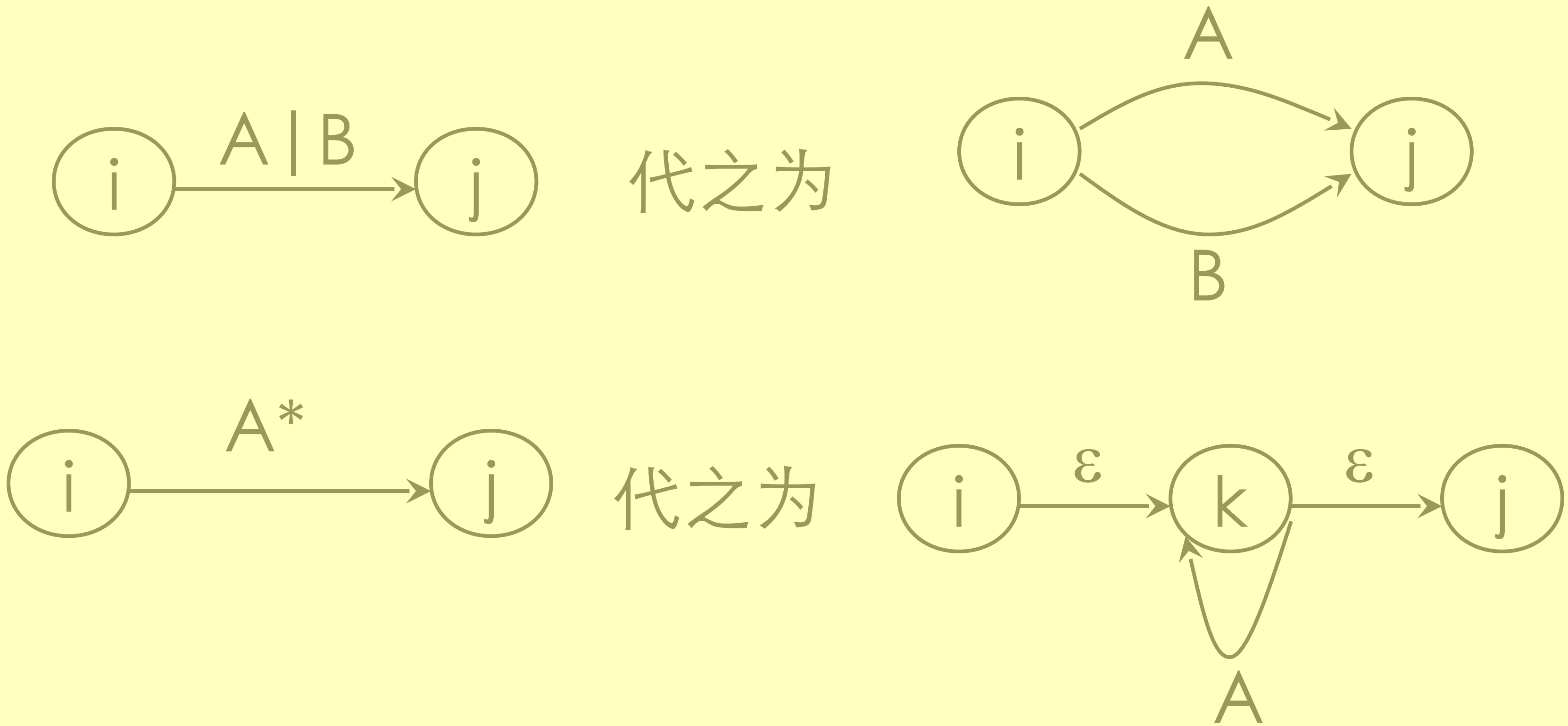
证明：

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

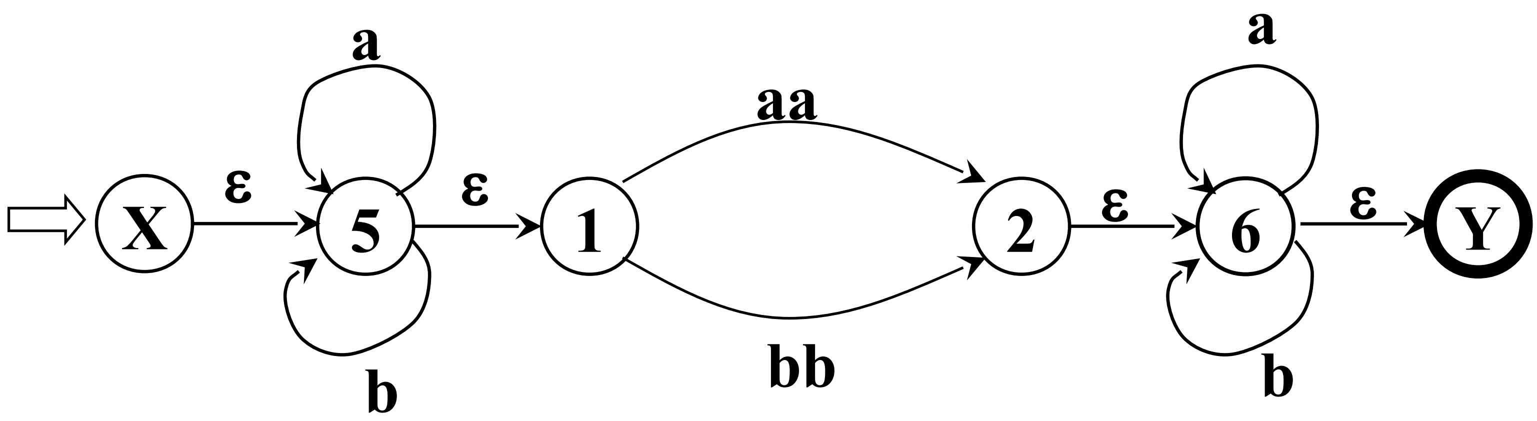
   1) 引进新的初态结点 X 和终态结点 Y， $X,Y \notin S$ ，
      从 X 到 S_0 中任意状态结点连一条ε箭弧，从 F 中任意状态结点连一条ε箭弧到 Y 。

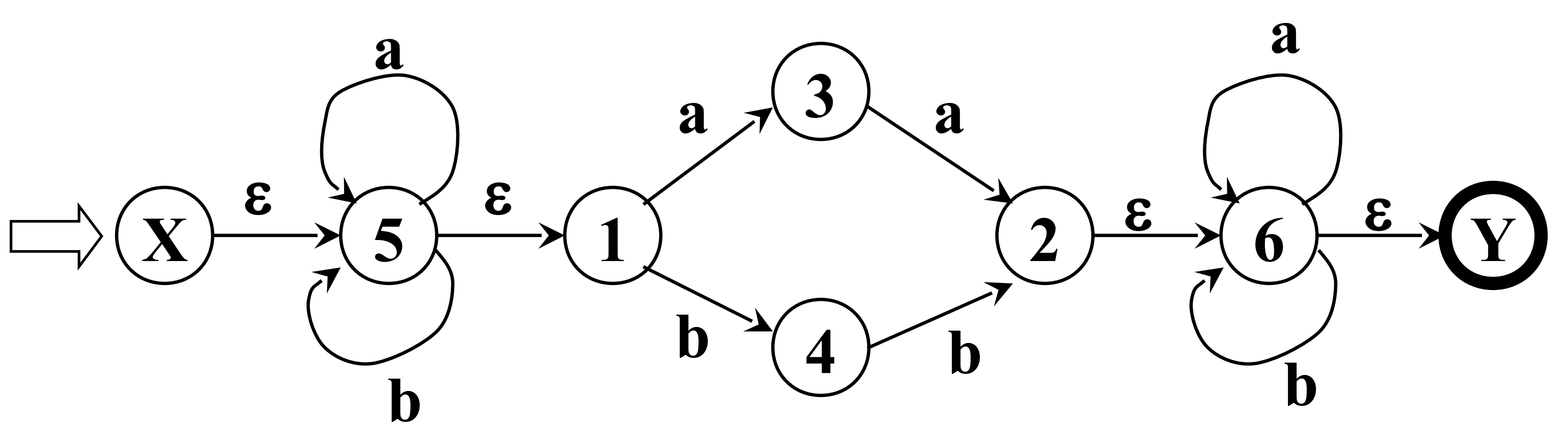
   2) 对 M 的状态转换图进一步施行替换，其中 k 是新引入的状态。

按下面的三条规则对箭弧进行分裂：

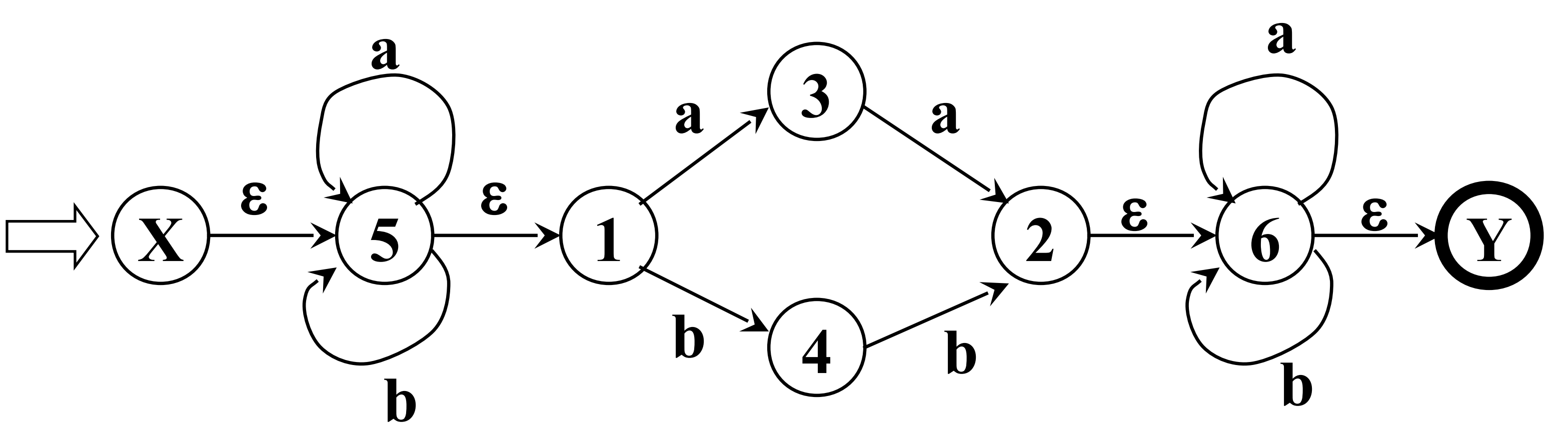


- 识别所有含相继两个 a 或相继两个 b 的字





- 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)



2. 把上述 NFA 确定化——采用子集法

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)

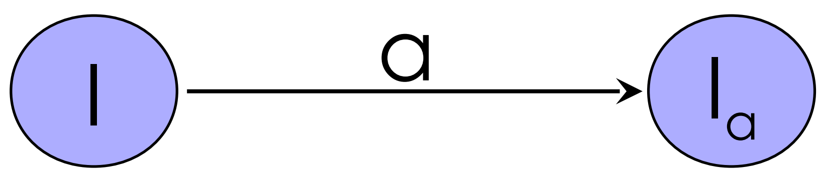
即

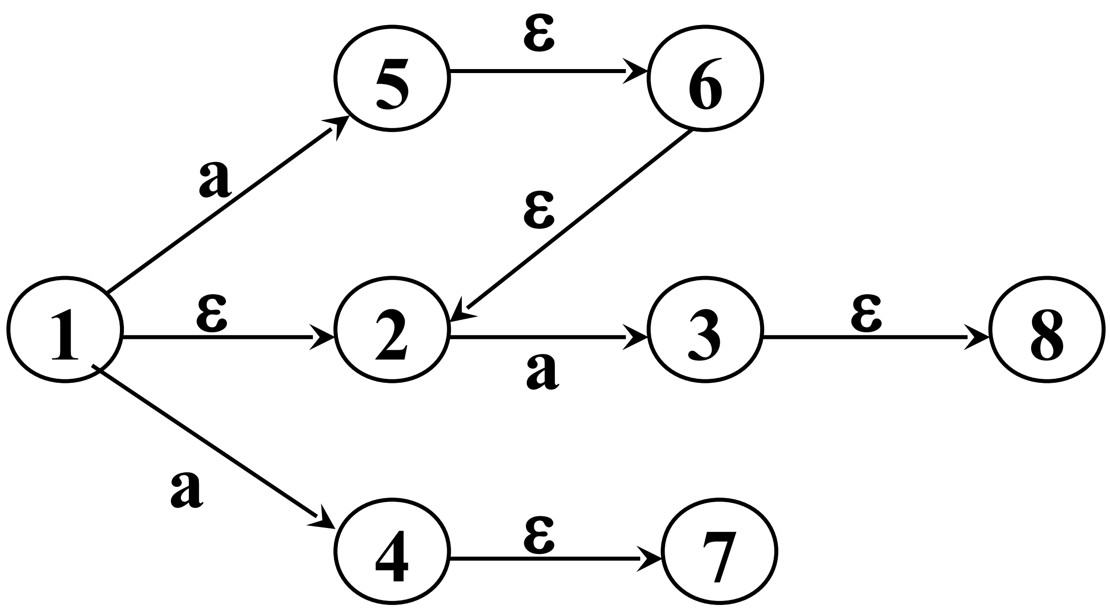
ε-closure(I)=I∪{s' | 从某个 s∈I 出发经过任意条ε弧能到达 s' }

- 设 a 是Σ中的一个字符，定义

$$I_a = \varepsilon\text{-closure}(J)$$

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。





- 设 a 是Σ中的一个字符，定义

  $I_a = \varepsilon\text{-closure}(J)$

  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

I_a?

J={5， 4， 3}

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

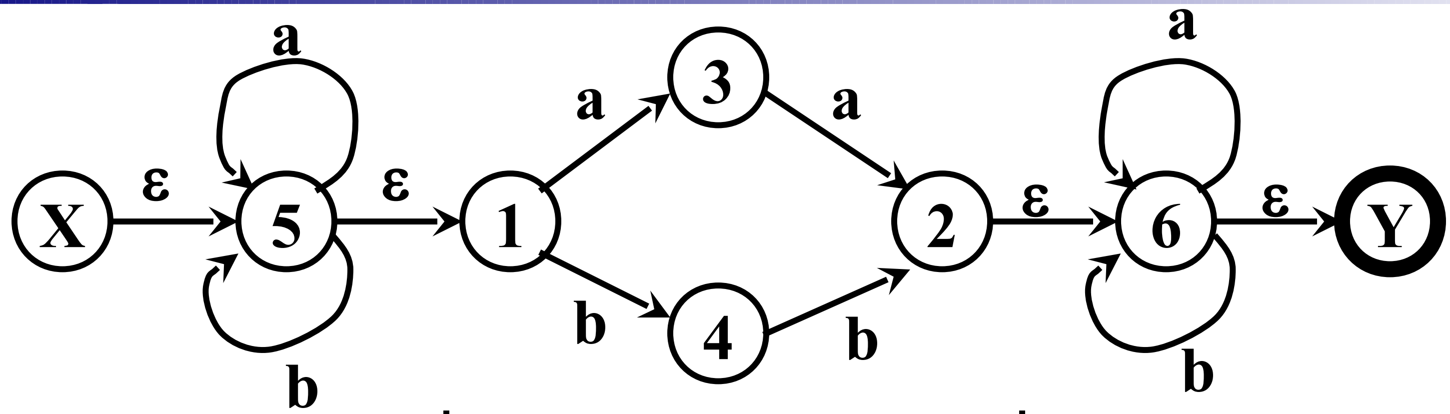
={5， 4， 3， 6， 2， 7， 8}

确定化的过程

- 不失一般性，设字母表只包含两个 *a* 和 *b*，我们构造一张表：

I	I_a	I_b
$\varepsilon-Closure(\{X\})$	{…}	{…}
{…}	{…}	{…}
{…}	{…}	{…}

- 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，直到所有第 2， 3 列子集全部出现在第一列为止



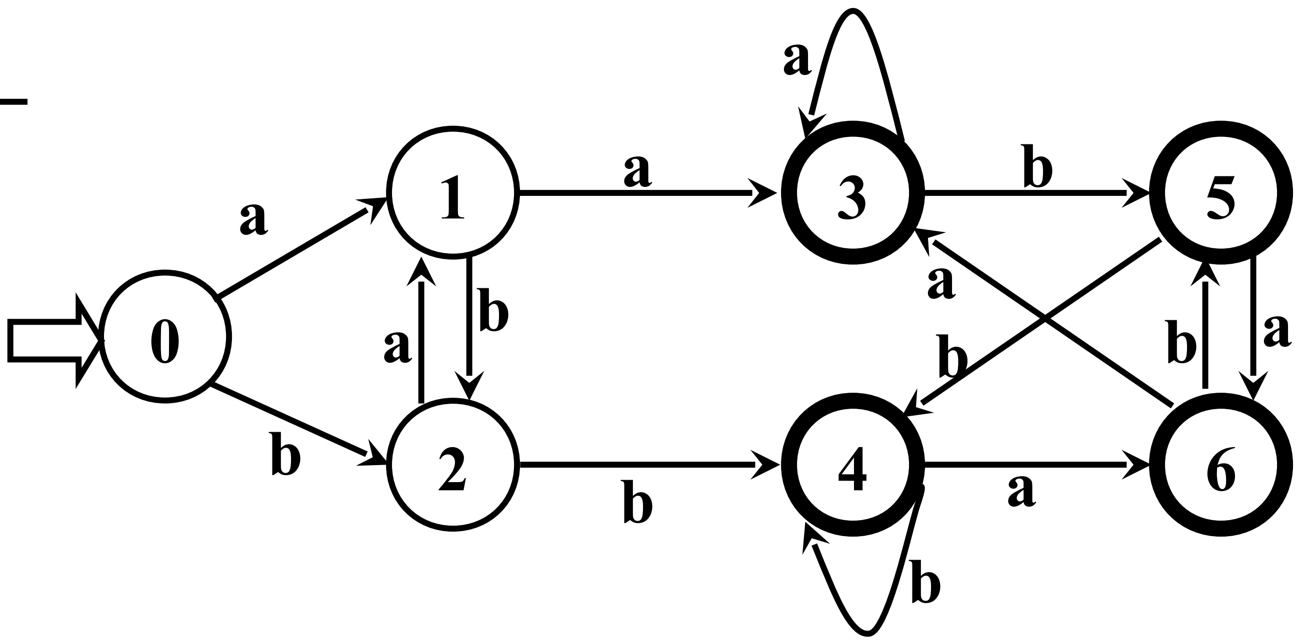
I	I_a	I_b
{X,5,1}	{5,3,1}	{5,4,1}
{5,3,1}	{5,2,3,1,6,Y}	{5,4,1}
{5,4,1}	{5,3,1}	{5,2,4,1,6,Y}
{5,2,3,1,6,Y}	{5,2,3,1,6,Y}	{5,4,6,1,Y}
{5,2,4,1,6,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,4,6,1,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,3,6,1,Y}	{5,2,3,1,6,Y}	{5,4,6,1,Y}

I	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}
{5,2,4,1,6,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,4,6,1,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}

- 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
  - 初态是ε -closure({X})
  - 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	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}
{5,2,4,1,6,Y}	{5,3,6,1,Y}	{5,2,4,1,6,Y}
{5,4,6,1,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	5
4	6	4
5	6	4
6	3	5



小结

```
curState = 初态
GetChar();
while( stateTrans[curState][ch] 有定义 ){
    // 存在后继状态，读入、拼接
    Concat();
    // 转换入下一状态，读入下一字符
   curState= stateTrans[curState][ch];
    if cur_state 是终态 then  返回 strToken 中的单
    GetChar( );
}
```

DIM,IF, DO,STOP,END
number, name, age
125, 2169
…

FA

DFA

3.3.2
3.3.3

NFA

正规集

3.3.1

正规式

3.3.5

DIM
IF
DO
STOP
END
letter(letter|digit)*
digit(digit)*

易于人工设计