编译原理

文法和语言

1. **概念：文法、语言、推导、句型、句子、短语、句柄的定义.**

2. **文法作为程序语言的语法的描述工具,它用规则只能陈述的是：语言的所有句子以什麽样的符号串能出现.请记住文法和语言的形式定义中的 “形式”的含义—只涉及语言的语法不涉及语言的语义.**

3. **符号串集合的表示法、结构及其特性，是程序设计语言语法分析研究的基础。**

文法的概念

语言

语法：是一组规则，定义符号如何排列，排列与符号含义无关。

语义 ：研究语法的含义

静态语义

动态语义

文法是阐述语法的一个工具

一、文法的概念 （写出以下语言的文法）

“你是大学生” 对　　“我是教师”对

“我大学生是”错　　“我学习大学生”对

符号和符号串

为给出语言的形式化定义，先讨论一些有关概念：

1、字母表—符号集：是字母的有穷非空集合。

2、符号串—字母表的符号组成的任何有穷序列。

3、符号串的长度：符号串x有m个符号，则长度就为m，表示|x|=m

4、空符号串：用ε表示，长度为0（不含任何符号）

5、符号串的运算：

（1） 符号串的头和尾

若z=xy，则x是z的头，y是z的尾。

（2）符号串的固有头和固有尾

若z=xy符号串，x非空，则y是固有尾；

若y非空，则x是固有头。

（3）符号串的连接：

（4）符号串的方幂：

- （6）闭包（∑*）
- 字母表∑，用∑* 表示∑上所有有穷长的串集合，∑* 称为∑的闭包。
- 例：字母表∑={0,1}

则∑*==$\sum^0 \cup \sum^1 \cup \sum^2 \cup \ldots \cup \sum^n$

={ε,0,1,00,01,10,11,000,001,010……}

〈7〉正闭包(∑⁺)

$\sum^+ = \sum^1 \cup \sum^2 \cup \ldots \cup \sum^n$

$\sum^+$称∑的正闭包。

显然：$\sum^* = \sum^0 \cup \sum^+$

$\sum^+ = \sum\sum^* = \sum^*\sum$

三、文法和语言的形式定义

（用以上术语对文法的概念进行形式化）

1、规则（重写规则、产生式、生成式）

形如$\alpha \rightarrow \beta$ 或α ::= β

α 称规则的左部，β称规则的右部。

2、文法的定义

定义为四元组（V_N，V_T，P，S）

3、直接推导的定义

<1> $\alpha \rightarrow \beta$ 是文法G=（V_N，V_T，P，S）的规则，γ和δ是V*中的任意符号，若有符号串V，W满足：

$V=\gamma\alpha\delta$，$W=\gamma\beta\delta$

$$\left.\begin{array}{l}\text{则说 V是直接产生W} \\ \text{或 W是V的直接推导} \\ \text{或 W直接规约到V}\end{array}\right\} \text{记作 } V \Rightarrow W$$

(2) 若存在直接推导的序列：

$V=W_0 \Rightarrow W_1 \Rightarrow W_2 \Rightarrow \ldots W_n = W \quad (n>0)$

则 称V推导W （或W规约到V），记 $V \Rightarrow W$

4、句型的定义：

设G[S]是文法，如果符号串x是从识别符号（开始符号）推导出来的（即$S \overset{*}{\Rightarrow} x$）则称x是文法G[S]的句型。

若x仅由终结符号组成（$S \overset{*}{\Rightarrow} x$，$x \in V_T^*$）

则称x为G[S]的句子。

例： S，0S1，000111都是文法G的句型，000111是G的句子。

〖结论〗句子一定是句型，句型不一定是句子。

5、语言的定义： L(G)表示

文法G产生的语言定义为： G产生的句子的集合

$\{ x \mid S \overset{*}{\Rightarrow} x$， S为文法开始符号， $x \in V_T^* \}$

该集合为语言，用L(G)表示。

从定义可知： x是句型且x是文法G的句子。

想： 例1的语言表示为什么？

四、文法的类型：

语言学家Chomsky把文法分成以下四种类型：

文法类型

- 0型 — 短语文法
- 1型 — 上下文有关文法
- 2型 — 上下文无关文法
- 3型 — 正规文法

逐渐增加限制

如果文法是正规文法⇒一定也是上下文无关文法

设$G=(V_N, V_T, P, S)$，如果它的每一个产生式$\alpha \rightarrow \beta$满足：

$\alpha \in (V_N \cup V_T)^*$且至少包含一个非终结符，

$\beta \in (V_N \cup V_T)^*$，则G是0型文法。

1型文法又称为上下文有关文法，它的每一个产生式也可描述为：$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$，其中$\alpha_1 \alpha_2 \beta$都在$(V_N \cup V_T)^*$中，$\beta \neq \varepsilon$，A在$V_N$中。只有A出现在$\alpha_1 \alpha_2$的上下文中，才允许用$\beta$取代A。

1、上下文无关文法：

设G=（V_N，V_T，P，S），若P中的每一个产生式$\alpha \to \beta$满足：

α是一非终结符，$\beta \in (V_N \cup V_T)^*$，则此文法称为上下文无关文法 (2型文法)。（用β 取代α 与α 所在的上下文无关）

例6：G＝({S,A,B},{a,b},P,S)，P的产生式如下：

$$\left.\begin{array}{ll} S\to aB & B\to b \\ S\to bA & B\to bS \\ A\to bAA & B\to aBB \\ A\to a & \\ A\to aS & \end{array}\right\}$$ 上下文无关文法

该文法可以写成：

$$S \to aB \mid bA$$

$$A \to a \mid aS \mid bAA$$

$$B \to b \mid Bs \mid aBB$$

2、正规文法：

设G=（V_N，V_T，P，S），若P中的每一个产生式都是$A \to aB$或$A \to a$，A,B都是非终结符，a是终结符，则G是正规文法。

五、上下文无关文法及其语法树

描述上下文无关文法的句型推导的直观方法 。

1、语法树（推导树）的定义：

给定文法G=（V_N, V_T，P，S），对于G的任何句型都能构造与之关联的语法树，须满足条件为：

- 每个结点都有一个标记，此标记是V的一个符号。
- 根的标记是S。
- 若一结点n至少有一个它自己除外的子孙，并且有标记A，则A肯定在V_N中。
- 如果结点n的直接子孙，从左到右的次序是结点$n_1, n_2, \ldots, n_K$，其标记分别为$A_1, A_2, \ldots, A_K$，那么$A \to A_1A_2\ldots A_K$一定是P中的一个产生式。

➤语法树的理解：

语法树表示了在推导过程中用到什么样的产生式和用到哪些非终结符，并没有表明采用的顺序。

只表示推导的结果，不表示推导的过程。

2、最左推导（或最右推导）：

若在推导中的任何一步 $\alpha \Rightarrow \beta$（α、β是句型），都是对α中的最左（最右）非终结符进行替换，则称为最左（最右）推导。

- 在形式语言中，最右推导常被称为规范推导。
- 由规范推导所得的句型称为规范句型。

3、文法的二义性的定义：

- ✓ 如果一个文法存在某个句子对应两棵不同的语法树，则说这个文法是二义的。
- ✓ 若一个文法中存在某个句子，它有两个不同的最左（最右）推导，则这个文法是二义的。

例10：文法G＝({ E}, {+，×，i，(，) }，P，E)，其中P为：

E→i

E →E+E

E →E×E

E →(E)

则句型i×i+i的推导：

六、句型的分析

❶ 句型分析：是识别一个符号串是否为某文法的句型，是整个推导的构造过程。（为一个符号串构造一个语法树/推导树）

即：识别输入符号串是否为语法上正确的程序的过程。

❷ 分析程序（识别程序）：是完成句型分析的程序 。

分析算法分为

- 自上而下分析法：从文法开始符号出发，反复使用规则，寻找匹配符号串（推导）
- 自下而上分析法：从输入符号开始，逐步进行“规约”，直至文法的开始符号（归约）

4、句柄的定义：（句柄是可归约串的称呼）

令G是一文法，S是文法的开始符号，$\alpha\beta\delta$是文法G的一个句型。（为$\alpha\beta\delta$ 确定可归约串）如果有$S \overset{*}{\Rightarrow} \alpha A\delta$ 且 $A \overset{+}{\Rightarrow} \beta$，则称$\beta$是句型$\alpha\beta\delta$相对于非终结符A的短语。

若有$A \Rightarrow \beta$，则称β是句型 $\alpha\beta\delta$ 相对$A \rightarrow \beta$的直接短语。一个句型的最左直接短语称为该句型的句柄。

句柄是自底向上句法分析中当前时刻需要规约的符号串。如果能够自动计算出当前的句柄，则可执行自动句法分析。

例：文法G[E]：

$$E \rightarrow T \mid E+T$$
$$T \rightarrow F \mid T \times F$$
$$F \rightarrow (E) \mid i$$

求句型i×i＋i的短语，直接短语和句柄。

文法实用中的一些说明
----化简文法

文法中不含有***有害规则***和***多余规则***

有害规则：形如U→U的产生式。会引起文法的二义性

多余规则：指文法中任何句子的推导都不会用到的规则

多余规则也可表示为：文法中不含有不可到达和不可终止的非终结符，分以下两种情况：

1）文法中某些非终结符不在任何规则的右部出现，该非终结符称为不可到达。

2）文法中某些非终结符，由它不能推出终结符号串，该非终结符称为不可终止。

编译原理

第一章　编译程序概述
第二章　PL/0编译程序的实现
第三章　文法和语言
第四章　词法分析
第五章　自顶向下语法分析方法
第六章　自底向上优先分析方法
第七章　LR分析方法
第八章　语法制导翻译和中间代码生成
第九章　符号表
第一〇章　代码优化
第一一章　代码生成

内容提要

- 词法分析程序的设计
- 单词的描述工具
- 有穷自动机
- 正规式和有穷自动机的等价性
- 正规文法和有穷自动机的等价性
- 词法分析程序的自动构造工具

词法分析

词法分析是编译过程的第一个阶段。

主要任务：从左到右逐个字符地对源程序进行扫描，产生一个个单词序列，用以语法分析。

基本思路：

- 将单词符号的语法用有效的工具描述；
- 基于该描述建立单词的识别机制；
- 基于词法分析程序的自动构造原理，设计和实现词法分析程序

词法分析程序的设计

- 词法分析程序与语法分析程序的接口方式
- 词法分析程序的输出
- 词法分析工作分离的考虑

单词符号(Token)的类别

1．关键字 (保留字) (Keyword)
   – begin, end, if, then, for, while …
2．标识符 (Identifier)
   – 用来表示各种名字
3．常数 (Literal)
   – 256，3 .14159，1E+3, true, "abc"
4．运算符 (Operator)
   – 如 +、-、*、/ 等
5．分界符 (Delimiter)
   – 如逗号，分号，冒号等

词法分析器的输出

- **Token的基本输出格式：**

  <单词类别，单词自身的属性值>

  类别提供给语法分析程序使用；词法单元自身的属性值提供给语义分析程序使用。

- **单词类别可以用整数编码表示，具体的分类设计以方便语法分析程序使用为原则。**
  - **标识符：1**
  - **常数：2**
  - **关键字：3**
  - **运算符：4**
  - **界符：5**

词法分析示例-1

```
while  i != j  do
    if  i>j  then  i:=i-j  else  j:=j-i
```

上述源程序经词法分析器的输出：

〈3，“while”〉
〈1，指向i的符号表入口的指针〉
〈4，“!=”〉
〈1，指向j的符号表入口的指针〉
〈3，“do”〉
〈3，“if”〉
〈1，指向i的符号表入口的指针〉
…………
〈1，指向j的符号表入口的指针〉

- 单词的描述工具
  - 正规文法
  - 正规式
- 单词的识别机制
  - 确定有穷自动机
  - 不确定有穷自动机
- 词法分析程序的自动构造原理
  - 正规式和有穷自动机的等价性
  - 词法分析程序的自动构造工具

4.2 单词的描述工具

某种程序设计语言中的所有单词构成一种语言，该语言的语法都能用正规文法表示。正规文法是描述单词的一种工具。

1、正规文法（回顾）

文法$G=(V_N, V_T, P, S)$，P中每一规则有$A \to aB$或$A \to a$ ，$A, B \in V_N$，$a \in V_T^*$，称G(S)是正规文法。

由正规文法产生的语言称为正规集

<标识符> → l|l <字母数字>

<字母数字> → l|d|l<字母数字>| d <字母数字>

<无符号整数> → d|d<无符号整数>

l表示a-z中的任何英文字母，d表示0-9中的任何数字

2、正规式（正则表达式）

也是表示正规集的工具，是用以描述单词符号的工具。

①正规式与正规集的定义：

设字母表为Σ，辅助字母表Σ'={φ，ε，|，·，*，（，）}；

✓ε和φ都是Σ上的正规式，表示的正规集分别为{ε}和φ；

✓任何a∈Σ是Σ上的一个正规式，表示的正规集为{a}；

✓假定e1和e2都是Σ上的正规式，它们所表示的正规集分别为L(e1)和L(e2)，则(e1)，e1|e2，e1·e2和e1*也都是正规式，所表示的正规集分别为L(e1)，L(e1)∪L(e2)，L(e1)L(e2)和(L(e1))*。

✓ 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，仅由这些正规式所表示的集合才是Σ上的正规集。

例：$\Sigma=\{a,b\}$，Σ上的正规式和相应的正规集为：

- ε
- ϕ
- a
- b
- ------------
- (a)
- (b)
- a|b
- ab
- a*
- b*
- ---------------------
- (a|b)*
- a*|b*
- aba*
- $(a|b)^*(aa|bb)(a|b)^*$

一个正规式可以表示若干个符号串，其正规集就是这些符号串的集合

- ε {ε}
- ϕ ϕ
- a {a}
- b {b}
- ------------------
- (a) {a}
- (b) {b}
- a|b {a, b}
- ab {ab}
- a* {ε, a, aa, aaa, aaaa, …. }
- b* {ε, b, bb, bbb, bbbb, …. }
- -------------------------------
- (a|b)* a和b组成的所有串
- a*|b* {ε, a, aa, aaa, aaaa, …. , b, bb, bbb, bbbb, …. }
- aba* 以ab开头后接若干个(包括0个)a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

一个正规式可以表示若干个符号串，其正规集就是这些符号串的集合

例：令∑＝｛**d**，**.**，**e**，＋，－｝，则∑上的正规式 **$d^*(.dd^*|\varepsilon)(e(+|-|\varepsilon)dd^*|\varepsilon)$**表示的是所有无符号数。

其中**d**为**0**～**9**中的数字。

比如以下都是正规式表示集合中的元素：

2

12.59

3.6e2

471.88e-1

设r，s，t为正规式，正规式服从代数规律有：

1、r|s＝s|r 交换律

2、r|(s|t)＝(r|s)|t 结合律

3、(rs)t=r(st)   结合律

4、r(s|t) = rs|rt 分配律

(s|t)r = sr|tr 分配律

5、εr=r 零一律

rε=r 零一律

程序设计语言中的单词既能用正规文法表示，又能用正规式来表示.

正规文法:

〈标识符〉 →l|l 〈字母数字〉

〈字母数字〉 → l|d|l〈字母数字〉| d 〈字母数字〉

〈无符号整数〉 → d|d〈无符号整数〉

l表示a-z中的任何英文字母

d表示0-9中的任何数字

正规式？

程序设计语言中的单词既能用正规文法表示，又能用正规式来表示.

正规文法:

<标识符> →l|l <字母数字>

<字母数字> → l|d|l<字母数字>| d <字母数字>

<无符号整数> → d|d<无符号整数>

l表示a-z中的任何英文字母

d表示0-9中的任何数字

正规式:

标识符:**e1=l(l|d)***

无符号整数:**e2=dd***

Quiz: 选择题

- 下面的正则表达式对应的语言中包含多少个不同的字符串？

$$(0\,|\,1\,|\,\varepsilon)(0\,|\,1\,|\,\varepsilon)(0\,|\,1\,|\,\varepsilon)(0\,|\,1\,|\,\varepsilon)$$

a) 12
b) 16
c) 31 √
d) 32
e) 64
f) 81

3、正规文法和正规式的等价性

一个正规语言可以由正规文法定义，也可以用正规式定义。

对于任意一个正规文法，存在一个定义同一语言的正规式。

对每一个正规式，存在一个生成同一语言的正规文法。

即正规式⇔正规文法

① 正规式⇒正规文法：（把正规式转换为正规文法所要求的规则形式）

将Σ上的一个正规式r转换为一个正规文法G=（V_N, V_T，P，S）的规则：

令V_T=Σ，

对正规式r，选择一个非终结符S生成S→r，S为G的开始符号。不断拆分r直到符合正规文法要求的规则形式：

◆ 若x，y都是正规式，对形如A→xy的产生式，写成A→xB，B→y。其中B∈ V_N

◆ 对形如A→x*y的产生式，重写为：

A→x A

A→y

B为新的非终结符，B∈ V_N

对形如A→x|y的产生式，重写为：

A→x

A→y

不断利用上述规则进行变换即可。

A→x B

A→y

B→x B

B→y

例：将**R＝a(a|d)***变换成正规文法。令**S**是文法开始符号。

例：将R＝a(a|d)*变换成正规文法。令S是文法开始符号。

解：

$S \rightarrow a(a|d)^*$

$$\left\{\begin{array}{l} S \rightarrow aA \\ A \rightarrow (a|d)^* \end{array}\right. \Rightarrow \left\{\begin{array}{l} A \rightarrow (a|d)A \\ A \rightarrow \varepsilon \end{array}\right.$$

最后得到：

$$\begin{cases} S \to aA \\ A \to aA \\ A \to dA \\ A \to \varepsilon \end{cases}$$

② 正规文法⇒正规式：将一个正规文法转换为正规式的规则：

转换规则：

❶ **A→xB，B→y** 正规式为： **A=xy**

❷ **A→xA|y** 正规式为： **A=x*y**

❸**A→x，A→y** 正规式为： **A=x|y**

不断收缩产生式规则，直到剩下一个开始符号定义的正规式

例：文法G[S]：

S→ aA

S→ a

A→ aA

A→ dA

A→ a

A→ d

转换为正规式

$S \rightarrow aA$

$S \rightarrow a$

$S = aA|a$

根据上述规则3

$A \rightarrow x$，$A \rightarrow y$

推出$A=x|y$

$A \rightarrow aA$

$A \rightarrow dA$

$A = aA|dA$

$A \rightarrow a$

$A \rightarrow d$

$A = a|d$

$A = (aA|dA)|(a|d)$ ⇨

$A = (a|d)A|(a|d)$ ⇨

$A = (a|d)^*(a|d)$

将它化为正规文法

变成$A \rightarrow$

$(a|d)A|(a|d)$

再根据上述

规则2转换

$x = y = (a|d)$

将A代入S＝aA|a得到如下：

$S ＝ a((a|d)^*(a|d))|a$

$=a(a|d)^+|a$

$=a((a|d)^+|\varepsilon)= a(a|d)^*$

4.3 有穷自动机

有穷自动机：是一种自动识别装置，能正确识别正规集；

是词法分析程序的工具和方法，可自动识别（且是正确识别）正规集。

分为

- 确定的有穷自动机（**DFA**）
- 不确定的有穷自动机（**NFA**）

（一）确定的有穷自动机DFA　　自动识别装置

一个确定的有穷自动机M是一个五元组：

M=（K，Σ，f，S，Z），其中：

① K是一个有穷集，每个元素表示一个状态；

②Σ是一个有穷字母表，每个元素是一个输入字符；

③ f是转换函数，是在K×Σ→K上的映象，如：

$f(K_i, a) = K_j \qquad (K_i, K_j \in K)$；

④ S是初态，S∈K；

⑤ Z⊂K，是终态集。

含义：当前状态为K_i，输入字符a，转换为K_j状态

状态转换图

- 状态转换图(transition diagram)
  - 状态(state)：表示在识别词素时可能出现的情况
    - 状态看作是已处理部分的总结
    - 某些状态为接受状态或最终状态，表明已找到词素
    - 开始状态（初始状态）：用“开始”边表示
  - 边(edge)：从一个状态指向另一个状态；边的标号是一个或多个符号
    - 当前状态为s，下一个输入符号为a，就从s离开，沿着标号为a的边到达下一个状态

1、用状态图表示DFA：

方法如下：

- 初始态用“⇒”或“－”表示；
- 终态点用“＋”或“◎”表示；
- 若$f(K_i, a) = K_j$，则从状态点K_i到K_j画弧，标记为**a**。

例4.6：DFA的M＝({S,U,V,Q},{a,b},f,S,{Q})

其中f为：

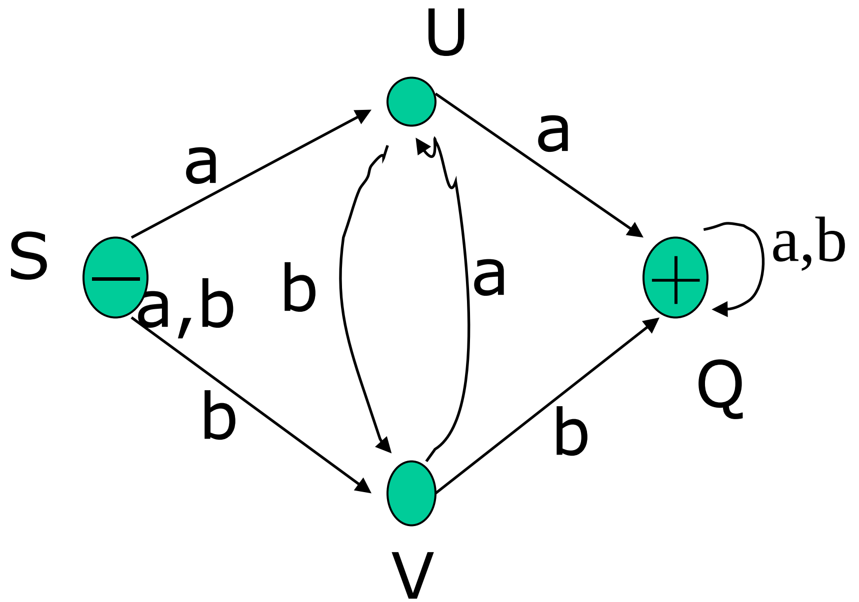
f(S,a)=U, f(S,b)=V, f(U,a)=Q

f(U,b)=V, f(V,a)=U, f(V,b)=Q

f(Q,a)=Q, f(Q,b)=Q

画出状态图

状态图如下：



2、用矩阵表示DFA：

方法：✦ 行表示状态

✦ 列表示输入字符

✦ 元素表示相应状态行和输入字符下的新状态。

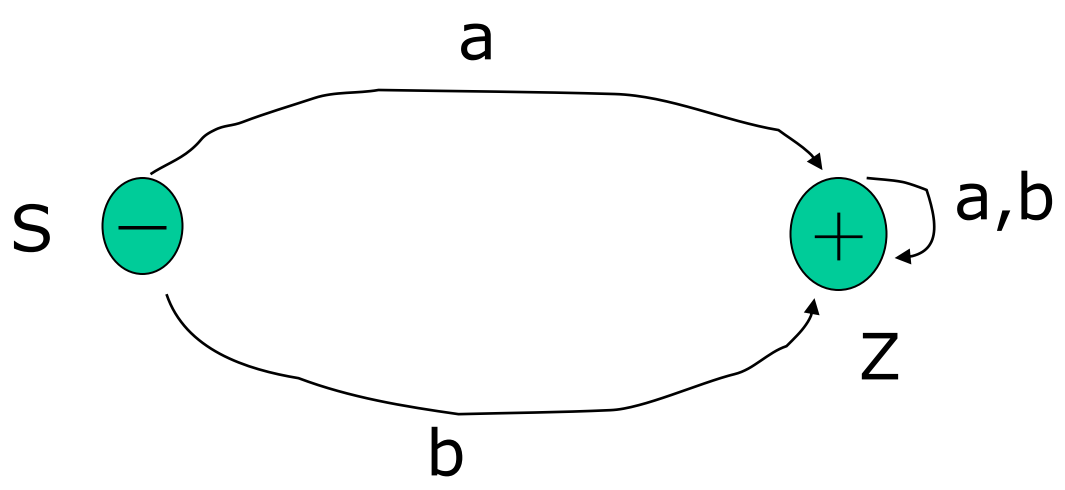
✦ "⇒" 标明初态，默认第一行是初态。

✦ 终态行在表右端标1，非终态标0

上例矩阵表示如下：

状态 \ 字符	a	b	
S	U	V	0
U	Q	V	0
V	U	Q	0
Q	Q	Q	1

例：



表示：$f(S,a)=Z$

$f(S,b)=Z$

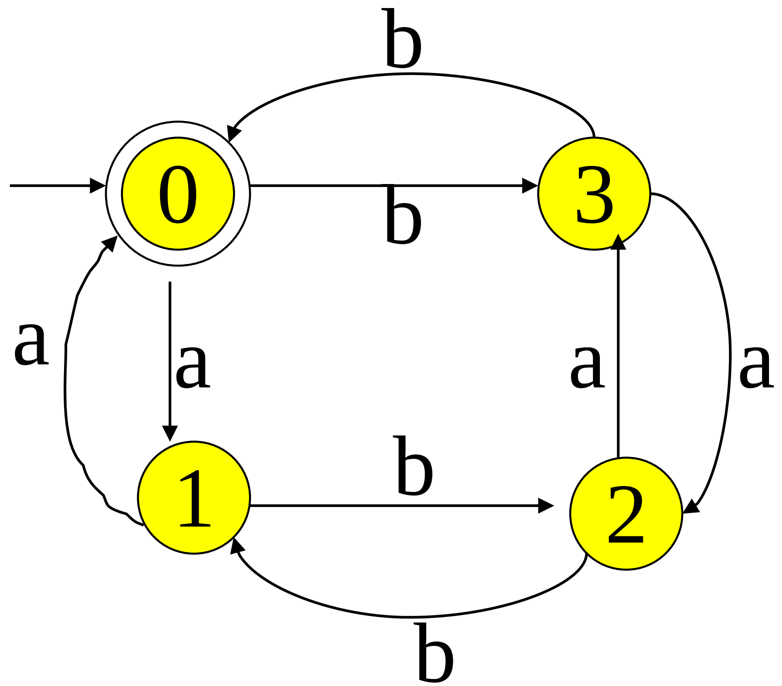
$f(Z,a)=Z$

$f(Z,b)=Z$

	a	b	
S	Z	Z	0
Z	Z	Z	1

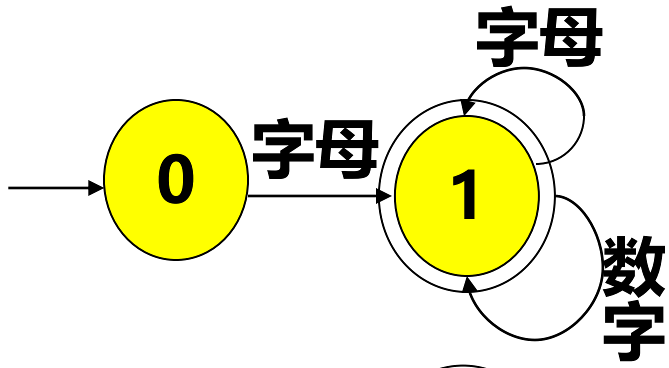
正规式是： $(a|b)(a|b)^*$

从状态转换图看，从开始状态出发，沿任一条路径到达接受状态，这条路径上的弧上的标记符号连接起来构成的符号串被DFA M接受。

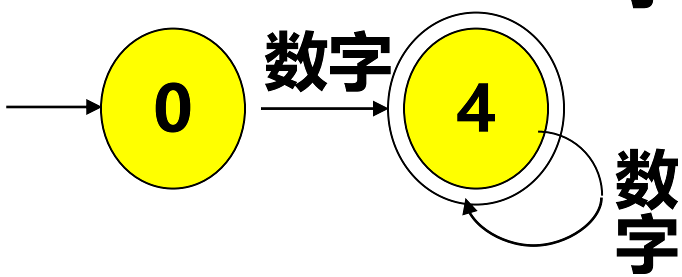


偶数个a，偶数个b的{a,b}串集合

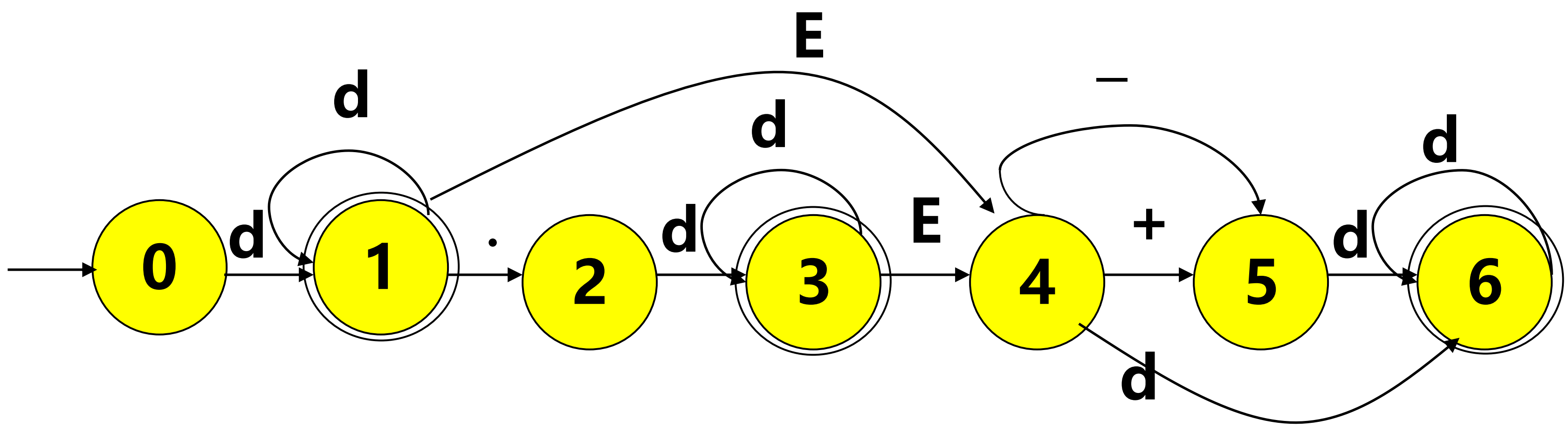
DFA示例-1



Pascal 标识符



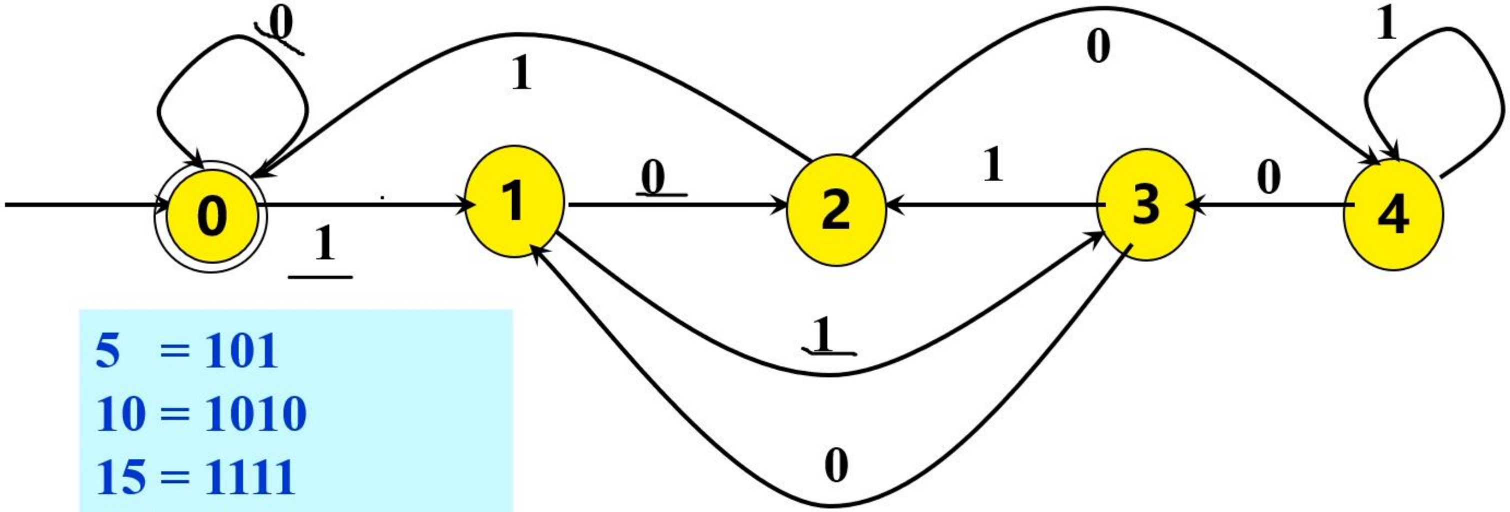
Pascal整数和实数



DFA示例-2

识别Σ ={0,1}上能被5整除的二进制数

按余数跳转

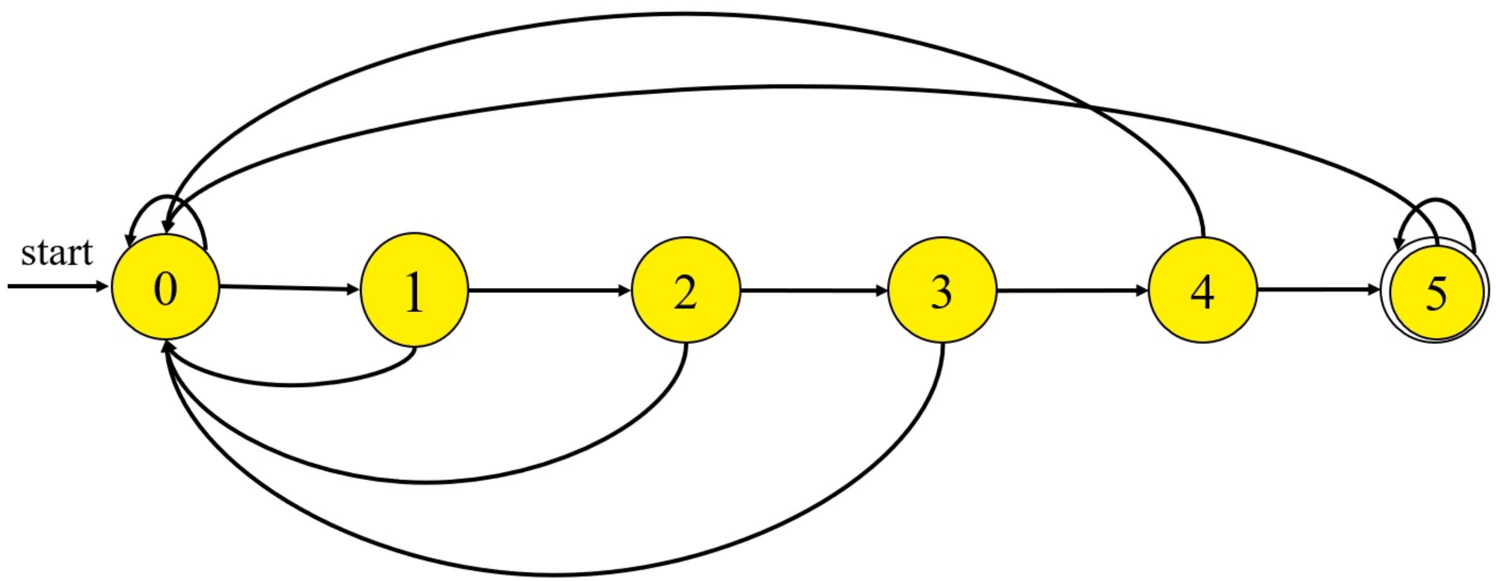


5 = 101
10 = 1010
15 = 1111
65 = 1000001

Quiz

- **画一个DFA，表示所有能被32整除的二进制数。**

末尾0的个数



3、接受（识别）的概念：　　自动识别单词

对于Σ^*中的任何字符串t，若存在一条初态到某一终态的路，且这条路上所有弧的标记符连接成的字符串等于t，则称t可为DFA M所接受。

若M的初态同时又是终态，则空字可为M所接受。

4、DFA M 运行的理解：

① 设$Q \in K$，函数$f(Q,\varepsilon)=Q$，则输入字符串是空串，并停留在原状态上。

② 输入字符串t（t表示成Tt_1形式，$T \in \Sigma$，$t_1 \in \Sigma^*$），在DFA M上运行的定义为：$f(Q, Tt_1)=f(f(Q,T),t1)$，其中$Q \in K$。

例如：**baab字符串被DFA所接受**，**DFA**见上例。

f(S，baab)=f(f(S，b)，aab)=f(V,aab)

＝f(f(V，a)，ab)=(f(U，ab)=f(f(U，a)，b)

=f(Q，b)=Q （**Q**是终态）

③**DFA M**所能接受的字符串的全体记为**L**（**M**）——称为语言（也即句子的集合）

5、DFA的确定性：

当**f**：**k×Σ→K**是一个单值函数，即对任何状态**K∈R**，输入符号**a ∈ K**，**f(k,a)**唯一确定下一状态。

DFA的行为很容易用程序来模拟.

DFA M=（K，Σ，f，S，Z）的行为的模拟程序

```
K:=S;
c:=getchar;
while c<>eof do
    {K:=f(K,c);
      c:=getchar;
    };
if K is in Z then return (‘yes’)
                  else return (‘no’)
```

自动识别单词的方法：

（1）把单词的结构用正规式描述；

（2）把正规式转换为一个NFA；

（3）把NFA转换为相应的DFA；

（4）基于DFA构造词法分析程序。

正则表达式⇒ NFA ⇒DFA ⇒min DFA

（二） 不确定的有穷自动机NFA

一个不确定的有穷自动机NFA M是一个五元组：

M=（K，Σ，f，S，Z），其中：

① K是一个有穷集，每个元素表示一个状态；

②Σ是一个有穷字母表，每个元素是一个输入字符；

③ f是一个从$K \times \Sigma^*$到K上的子集的映射；

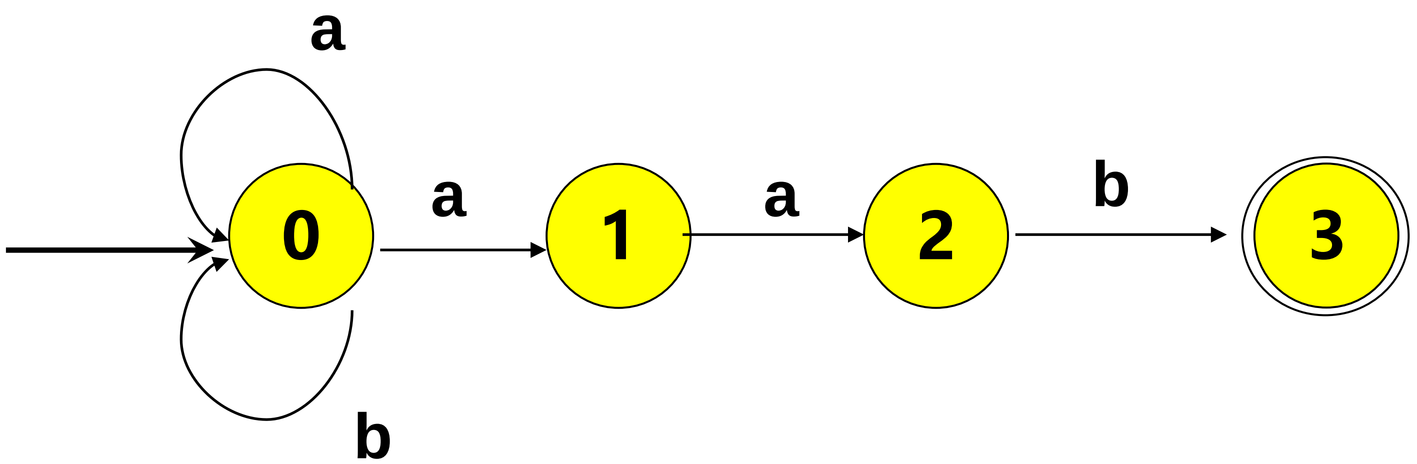
$$f: k \times \Sigma^* \rightarrow 2^k$$

④ $S \subset K$，是一个非空初态集；

⑤ $Z \subset K$，是一个终态集。

与DFA区别：多值函数 f(Ki,a)=Kj f(Ki,a)=Kt；允许输入字符为ε

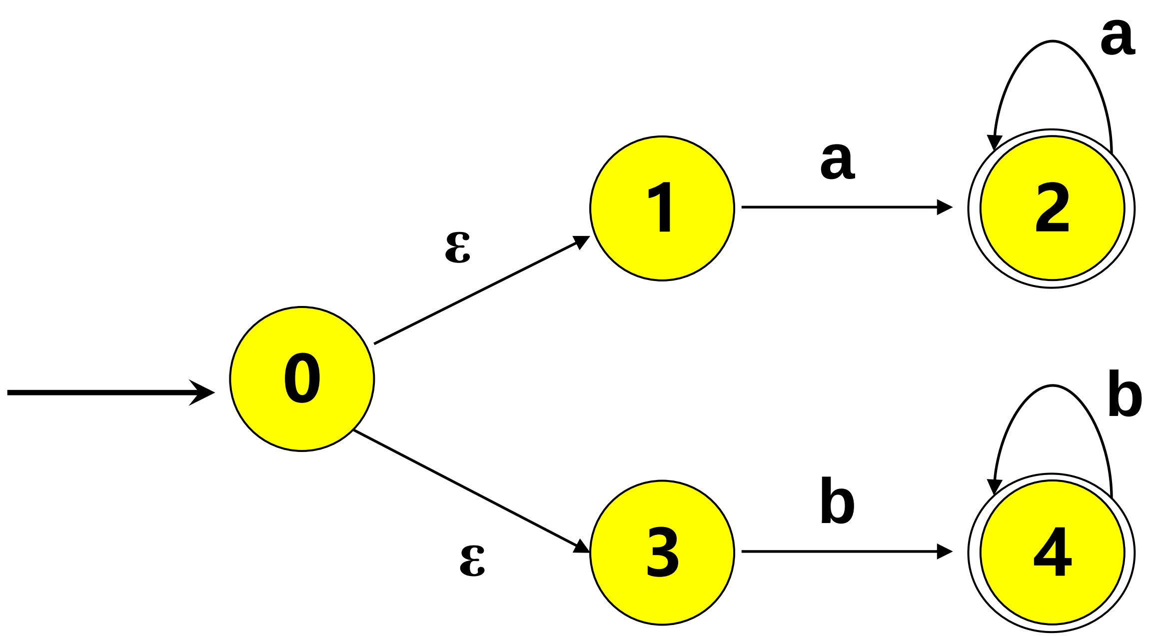
NFA示例-1



NFA 表示的语言：

(a|b)*aab

NFA示例-2



NFA 表示的语言：

aa*|bb*

可不可以用DFA来表示?

例4.7：一个NFA，

M＝({0,1,2,3,4},{a,b},f,{0},{2,4})

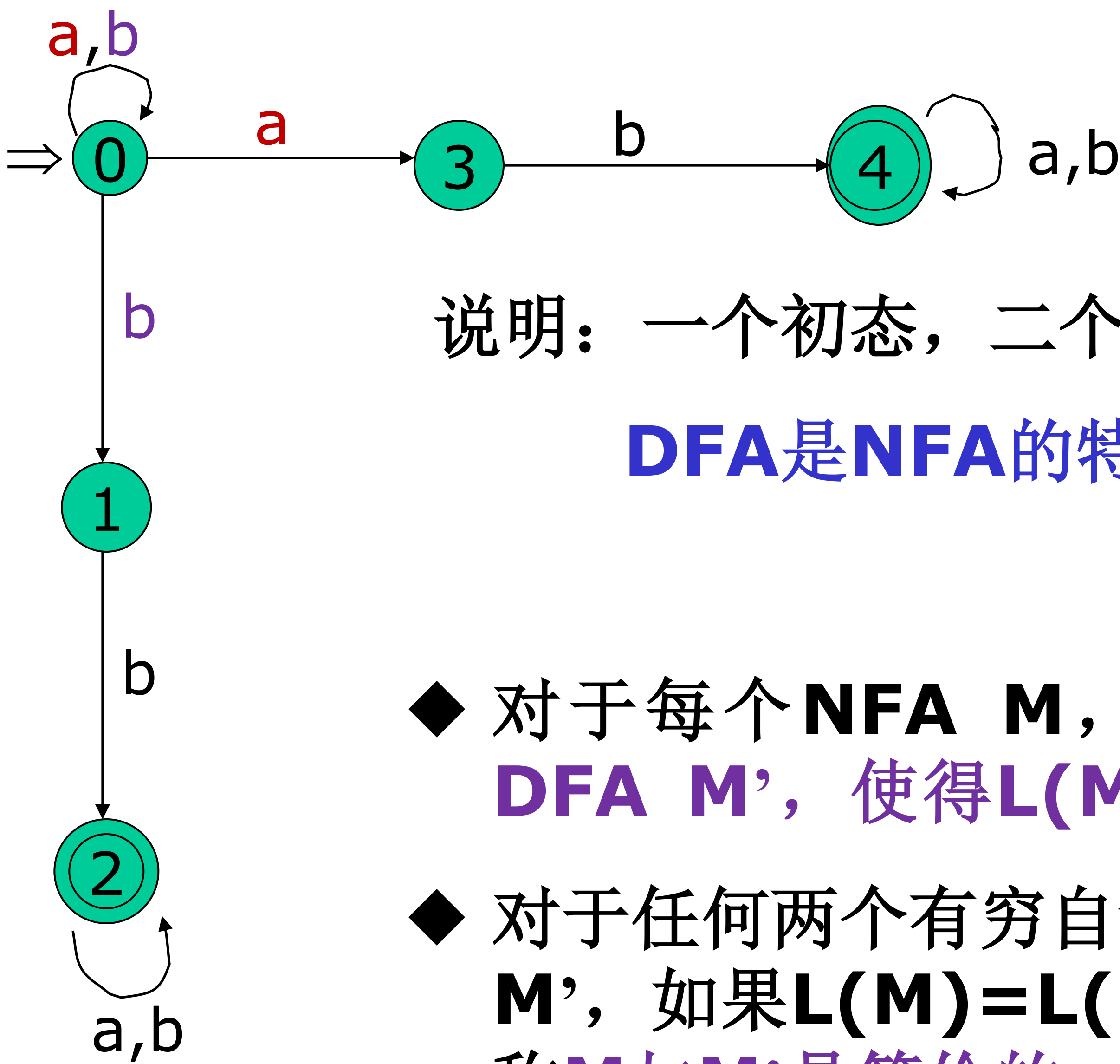
其中：***f(0,a)={0,3}*** **f(2,b)={2}**

f(0,b)={0,1} **f(3,a)={4}**

f(1,b)={2} **f(4,a)={4}**

f(2,a)={2} **f(4,b)={4}**

状态图表示如下：



说明：一个初态，二个终态。

DFA是NFA的特例。

- 对于每个**NFA M**，存在一个**DFA M'**，使得**L(M)=L(M')**。
- 对于任何两个有穷自动机**M**和**M'**，如果**L(M)=L(M')**，则称**M与M'**是等价的。

NFA 转换为等价的DFA

- 从NFA的矩阵表示中可以看出，表项通常是一状态的集合，而在DFA的矩阵表示中，表项是一个状态
- NFA ⇒DFA 基本思路是：DFA的每一个状态对应NFA的一组状态，DFA使用它的状态去记录在NFA读入一个输入符号后可能达到的所有状态

定义对状态集合I的几个有关运算：

1. **状态集合I的ε-闭包**，表示为ε-closure(I)，定义为一状态集，是状态集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的任何状态S都属于ε-closure(I)

2. **状态集合I的a弧转换**，表示为move(I,a)定义为状态集合J，其中J是所有那些可从I中的某一状态经过一条a弧而到达的状态的全体。

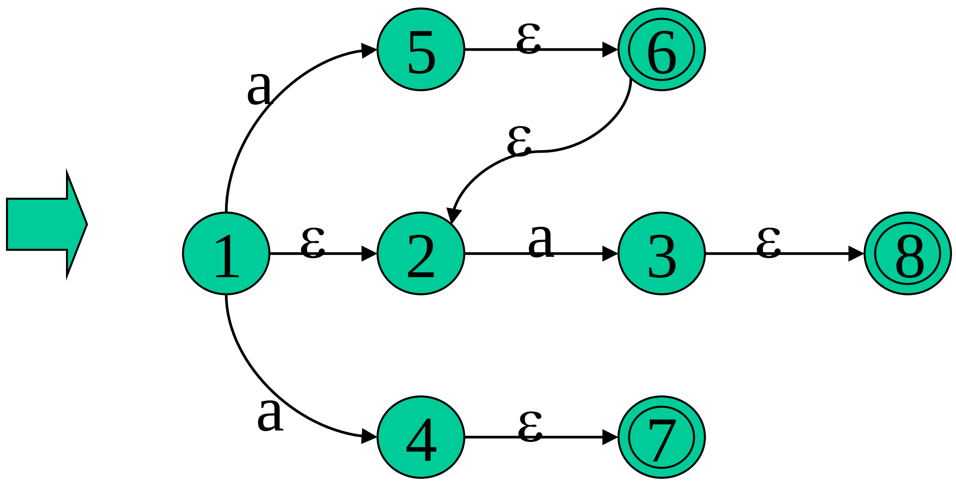
状态集合I的有关运算的例子

I={1}, ε-closure(I)=?；

I={5}, ε-closure(I)=?；

move({1,2},a)=?

ε-closure({5,3,4})=?；



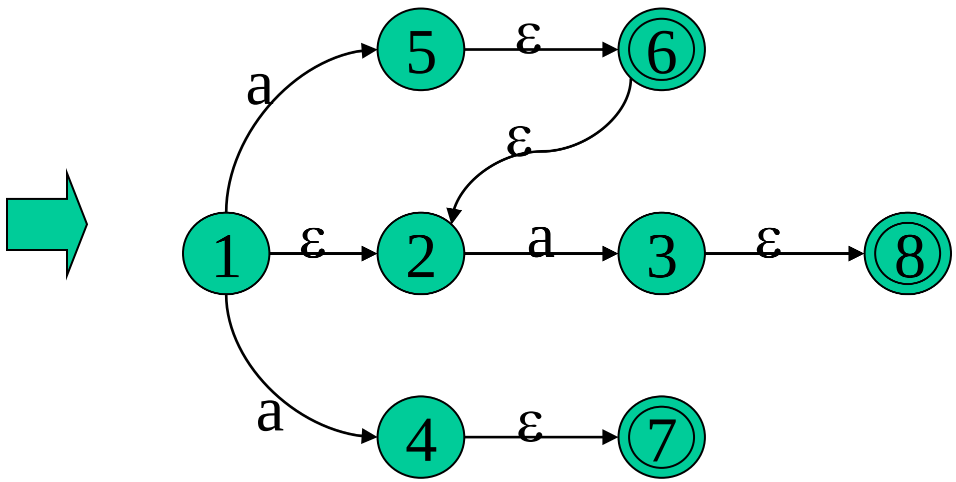
状态集合I的有关运算的例子

I={1}, ε-closure(I)={1,2}；

I={5}, ε-closure(I)={5,6,2}；

move({1,2},a)={5,3,4}

ε-closure({5,3,4})={2,3,4,5,6,7,8}；



NFA确定化算法（子集法）

NFA N=(K, $\sum$,f,K_0,K_t)按如下办法构造一个DFA M=(S, $\sum$,d,S_0,S_t)，使得L(M)=L(N)：

1. 构造DFA M的状态，选择NFA N的状态的一些子集构成：

M的状态集S由K的一些子集组成。用[S_1 S_2… S_j]表示S的元素，其中S_1, S_2,… S_j是K的状态。并且约定，状态S_1, S_2,… S_j是按某种规则排列的，即对于子集{S_1, S_2}={ S_2, S_1}来说，S的状态就是[S_1 S_2]；

2. M和N的输入字母表是相同的，即是Σ；

3. 构造DFA M的转换函数，根据新构造的状态和字母表中的字符构造：

转换函数是这样定义的：

$$d([S_1\ S_{2,}\dots S_j],a)=[R_1R_2\dots R_t]$$

其中 $\{R_{1,}R_{2,}\dots_{,}R_t\} = \varepsilon\text{-closure}(move(\{S_1, S_{2,}\dots S_j\},a))$

4 $S_0=\varepsilon\text{-closure}(K_0)$为M的开始状态；

5 $S_t=\{[S_i\ S_k\dots S_e]$，其中$[S_i\ \ S_k\dots S_e]\in S$且$\{S_i , S_{k,}\dots S_e\}\cap K_t\neq\Phi\}$

构造NFA N的状态K的子集的算法：

把多值转换函数所转换到的多值（多状态）的集合作为一个子集，映射到DFA的一个新的状态

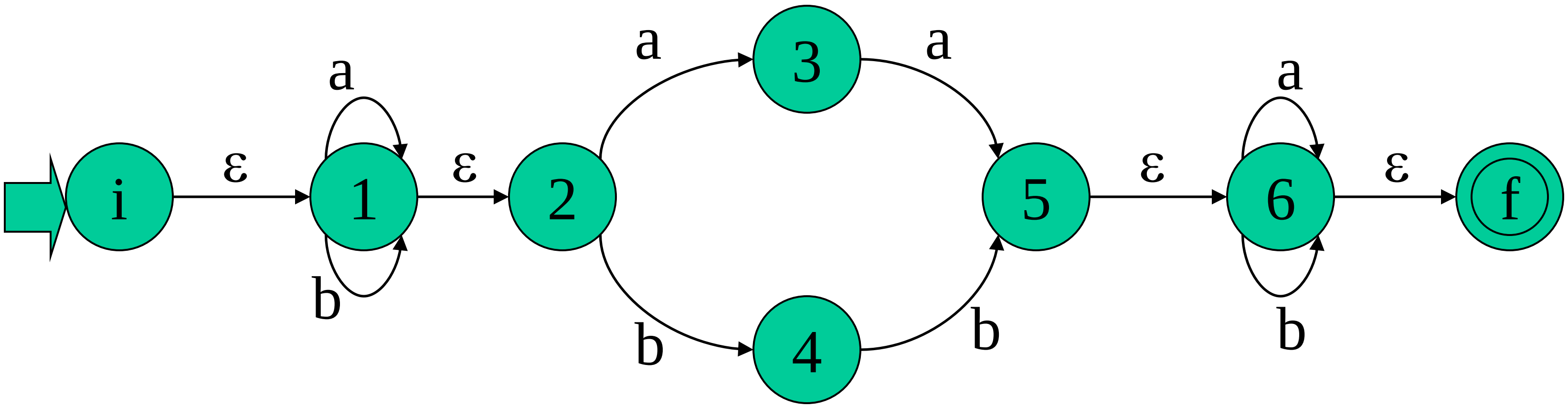
假定所构造的子集族为C，即C= (T_1, T_2,... T_I)，其中T_1, T_2,... T_I为状态K的子集。

1 开始，令ε-closure(K_0)为C中唯一成员，并且它是未被标记的。

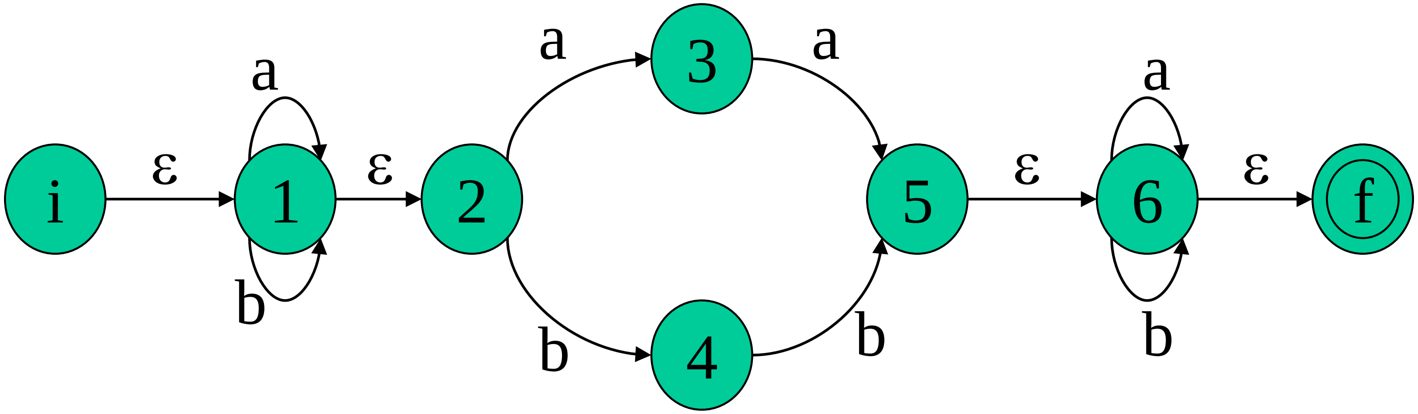
```
2   while （C中存在尚未被标记的子集T）do
  {
   标记T;
    for 每个输入字母a   do
   {
        U:= ε-closure(move(T,a));
        if  U不在C中   then
            将U作为未标记的子集加在C中
   }
  }
```

NFA的确定化

例子

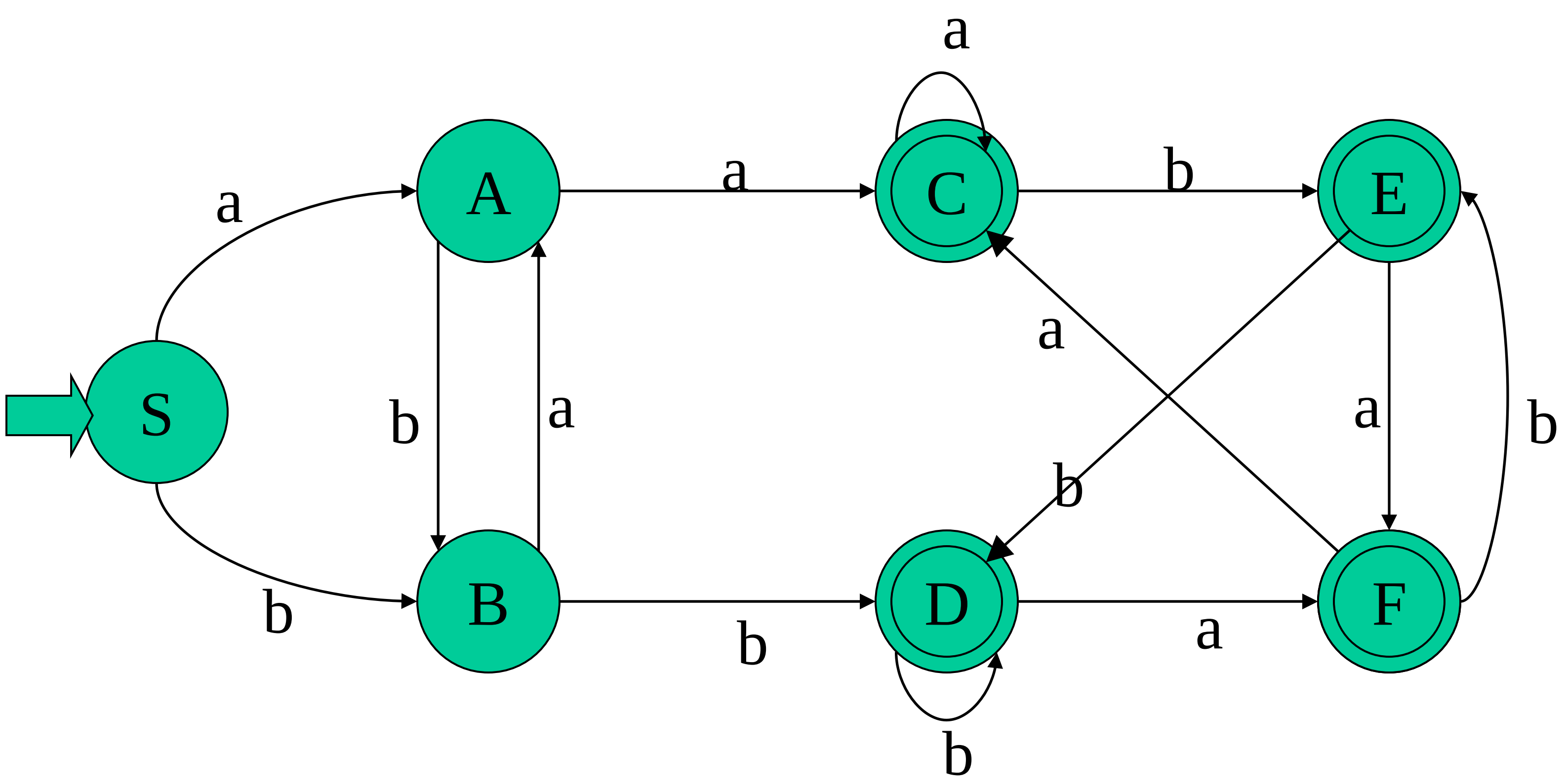


DFA还是NFA？



T		ε-clo (move(T,a))		ε-clo (move(T,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

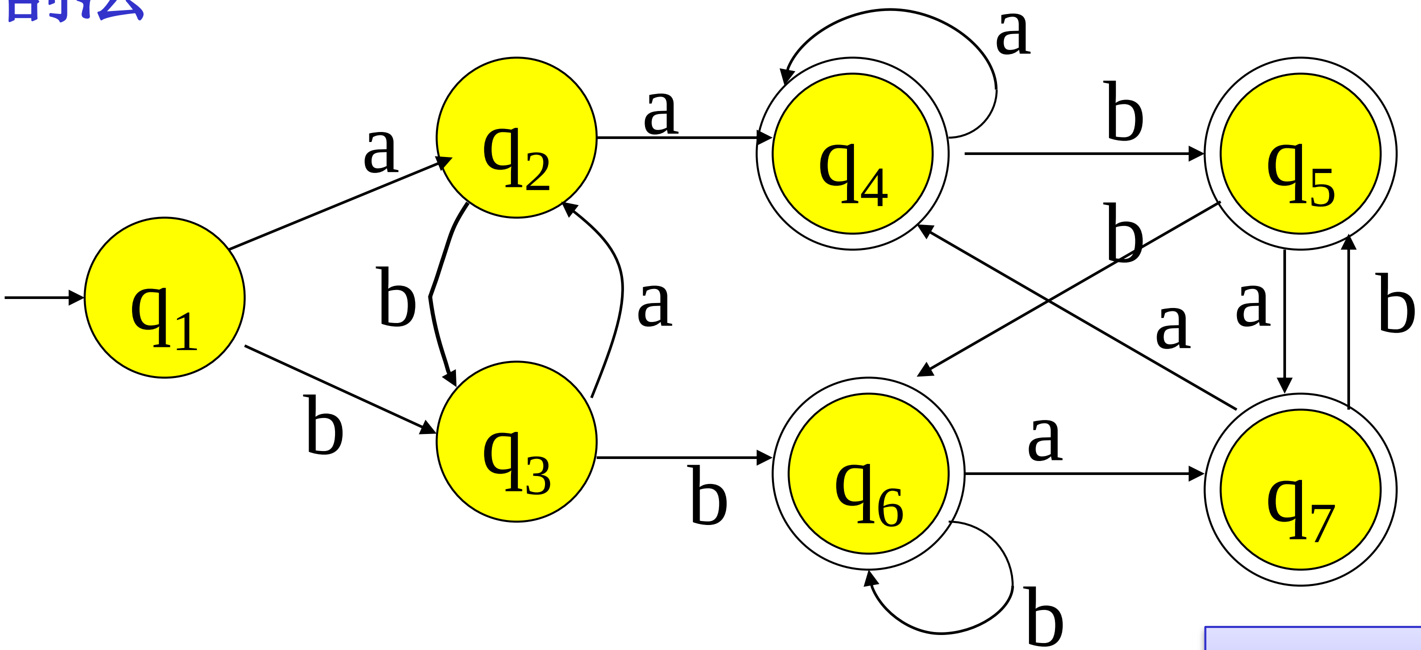
等价的DFA



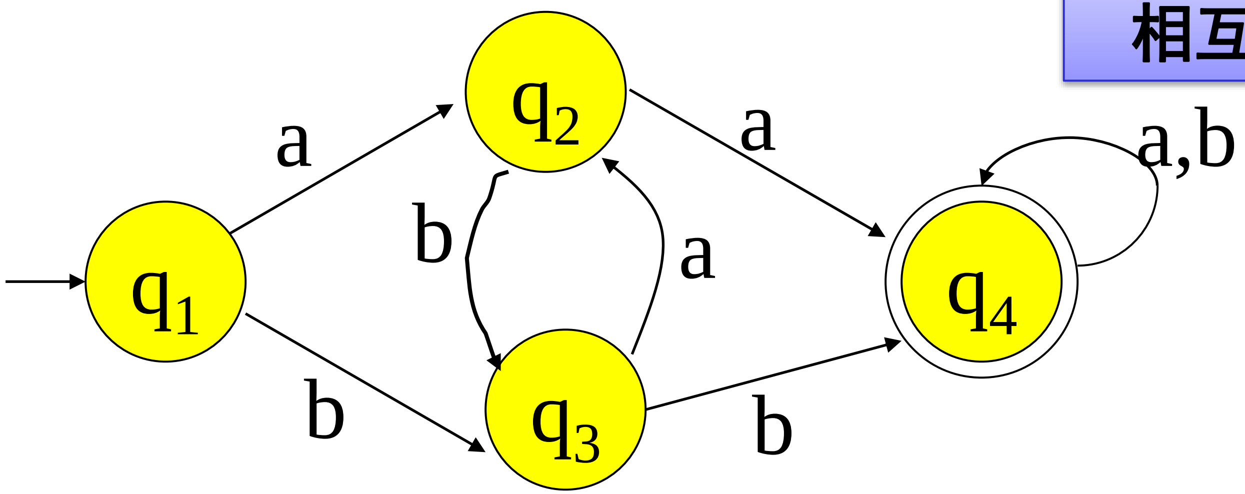
DFA的最小化

- **DFA的化简**
  - 设有DFA M, 寻找一个状态数更少的DFA M ′, 使 L(M′) = L(M)
  - 可以证明，存在一个最少状态的DFA M ′, 使 L(M)=L(M ′)。
- **等价条件**
  - 一致性条件：状态s, t 必须同时为可接受状态或不可接受状态
  - 蔓延性条件：对任意输入符号, 状态s和t必须转换到等价的状态里，否则是**可区别的**。

分割法



q4 q5 q6 q7 相互等价。



内容回顾

- **自动识别单词的方法：**
  - **（1）把单词的结构用正规式描述；**
  - **（2）把正规式转换为一个NFA；**
  - **（3）把NFA转换为相应的DFA；**
  - **（4）基于DFA构造词法分析程序。**

课后作业

- **P72：1（1），2，5**
- **P73：8**