第2章　形式语言与自动机基础

重庆大学 葛亮

知识点：文法的形式定义

上下文无关文法、正规文法

推导、短语、分析树、二义性

有限自动机的形式定义

自动机、文法、表达式等价性

NFA的确定化、DFA的最小化

形式语言与自动机基础

2.1 语言和文法

2.2 有限自动机

2.3 正规文法与有限自动机的等价性

2.4 正规表达式与有限自动机的等价性

2.5 正规表达式与正规文法的等价性

小结

2.1 语言和文法

一、字母表和符号串
二、语言
三、文法及其形式定义
四、推导和短语
五、分析树及二义性
六、文法的变换

一、字母表和符号串

字母表

- 符号的非空有限集合
- 典型的符号是字母、数字、各种标点和运算符等。

符号串

- 定义在某一字母表上
- 由该字母表中的符号组成的有限符号序列
- 同义词：句子、字

符号串有关的几个概念

- 长度
  - 符号串α的长度是指α中出现的符号的个数，记作|α|。
  - 空串的长度为0，常用ε表示。
- 前缀
  - 符号串α的前缀是指从符号串α的末尾删除0个或多个符号后得到的符号串。如：univ 是 university 的前缀
- 后缀
  - 符号串α的后缀是指从符号串α的开头删除0个或多个符号后得到的符号串。如：sity 是 university 的后缀
- 子串
  - 符号串α的子串是指删除了α的前缀和/或后缀后得到的符号串。如：ver 是 university 的子串

符号串有关的几个概念（续）

- **真前缀、真后缀、真子串**
  - 如果非空符号串β是α的前缀、后缀或子串，并且β≠α，则称β是α的真前缀、真后缀、或真子串。
- **子序列**
  - 符号串α的子序列是指从α中删除0个或多个符号(这些符号可以是不连续的)后得到的符号串。如：nvst

符号串运算

■ 连接

- ◆ 符号串α和符号串β的连接αβ是把符号串β加在符号串α之后得到的符号串
- ◆ 若α=ab，β=cd，则αβ=abcd，βα=cdba。
- ◆ 对任何符号串α来说，都有εα=αε=α

■ 幂

- ◆ 若α是符号串，α的n次幂α^n 定义为：

$$\underbrace{\alpha\alpha\ldots\ldots\alpha\alpha}_{n个}$$

  - 当n=0时，α^0是空串ε。
  - 假如α=ab，则有：

$\alpha^0=\varepsilon$　　$\alpha^1=ab$　　$\alpha^2=abab$

……

$$\alpha^n=\underbrace{abab\ldots\ldots ab}_{n个ab}$$

二、语言

语言： 在某一确定字母表上的符号串的集合。

- 空集ϕ，集合$\{\varepsilon\}$也是符合此定义的语言。
- 这个定义并**没有把任何意义赋予语言中的符号串**。

语言的运算： 假设 L 和 M 表示两个语言

- L和M的**并**记作$L \cup M$：$L \cup M=\{s \mid s \in L$ 或 $s \in M\}$
- L和M的**连接**记作LM：$LM=\{st \mid s \in L$ 并且 $t \in M\}$
- L的**闭包**记作L^*：即L的0次或若干次连接。

$$L^*=\bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ \ldots\ldots$$

- L的**正闭包**记作L^+：即L的1次或若干次连接。

$$L^+=\bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup L^3 \cup L^4 \cup \ \ldots\ldots$$

- 把幂运算推广到语言

  $L^0=\{\varepsilon\}$，$L^n=L^{n-1}L$，于是L^n是语言L与其自身的n-1次连接。

- $L=\{A, B, \ldots, Z, a, b, \ldots, z\}$，$D=\{0, 1, \ldots, 9\}$
  - 可以把L和D看作是字母表
  - 可以把L和D看作是语言
- 语言运算举例：

语言	描述
$L \cup D$	全部字母和数字的集合
LD	由一个字母后跟一个数字组成的所有符号串的集合
L^4	由4个字母组成的所有符号串的集合
L^*	由字母组成的所有符号串（包括ε）的集合
$L(L \cup D)^*$	以字母开头，后跟字母、数字组成的所有符号串的集合
D^+	由一个或若干个数字组成的所有符号串的集合

三、文法及其形式定义

- **文法**：所谓文法就是描述语言的语法结构的形式规则。
- 任何一个文法都可以表示为一个**四元组**$G=(V_T, V_N, S, \varphi)$

  V_T是一个非空的有限集合，它的每个元素称为**终结符号**。

  V_N是一个非空的有限集合，它的每个元素称为**非终结符号**。

  $$V_T \cap V_N = \Phi$$

  S是一个特殊的非终结符号，称为文法的**开始符号**。

  φ是一个非空的有限集合，它的每个元素称为**产生式**。
- 产生式的形式为：$\alpha \rightarrow \beta$

  “$\rightarrow$” 表示 “**定义为**”（或“**由.....组成**”）

  α、$\beta \in (V_T \cup V_N)^*$ ，$\alpha \neq \varepsilon$
- 左部相同的产生式$\alpha \rightarrow \beta_1$、$\alpha \rightarrow \beta_2$、……、$\alpha \rightarrow \beta_n$可以缩写

  $\alpha \rightarrow \beta_1 | \beta_2 | \ldots\ldots | \beta_n$

  “|” 表示 “**或**”， 每个β_i (i=1, 2, …, n)称为α的一个**候选式**

文法分类

- 根据对产生式施加的限制不同，定义了四类文法和相应的四种形式语言类。

文法类型	产生式形式的限制	文法产生的语言类
0型文法	$\alpha \to \beta$ 其中 $\alpha, \beta \in (V_T \cup V_N)^*$ $\lvert\alpha\rvert \neq 0$	0型语言
1型文法，即 上下文有关文法	$\alpha \to \beta$ 其中 $\alpha, \beta \in (V_T \cup V_N)^*$ $\lvert\alpha\rvert \leq \lvert\beta\rvert$	1型语言，即 上下文有关语言
2型文法，即 上下文无关文法	$A \to \beta$ 其中 $A \in V_N$，$\beta \in (V_T \cup V_N)^*$	2型语言，即 上下文无关语言
3型文法，即 正规文法 （线性文法）	$A \to a$或$A \to aB$（右线性），或 $A \to a$或$A \to Ba$（左线性） 其中 $A, B \in V_N$，$a \in V_T \cup \{\varepsilon\}$	3型语言，即 正规语言

上下文无关文法及相应的语言

- 所定义的语法单位(或称语法实体)完全独立于这种语法单位可能出现的上下文环境
- 现有程序设计语言中，许多语法单位的结构可以用上下文无关文法来描述。

例：描述算术表达式的文法G：

G=({i, +, -, *, /, (,)}, {<表达式>, <项>, <因子>}, <表达式>, φ)

其中φ：

<表达式>→<表达式>+<项> | <表达式>-<项> | <项>

<项>→<项>*<因子> | <项>/<因子> | <因子>

<因子>→(<表达式>) | i

- 语言L(G)是所有包括加、减、乘、除四则运算的算术表达式的集合。

BNF（Backus-Normal Form）表示法

- **元语言：**

  ::=　　表示　“定义为”　或　“由.....组成”

  <.....>　　表示非终结符号

  |　　表示“或”

- **算术表达式文法的BNF表示：**

  <表达式> ::= <表达式>+<项> | <表达式>-<项> | <项>

  <项> ::= <项>*<因子> | <项>/<因子> | <因子>

  <因子> ::= (<表达式>) | i

文法书写约定

- 终结符号
  - 次序靠前的小写字母，如：a、b、c
  - 运算符号，如：+、-、*、/
  - 各种标点符号，如：括号、逗号、冒号、等于号
  - 数字1、2、...、9
  - 黑体字符串，如：id、begin、if、then

- 非终结符号
  - 次序靠前的大写字母，如：A、B、C
  - 大写字母S常用作文法的开始符号
  - 小写的斜体符号串，如：*expr*、*term*、*factor*、*stmt*

文法书写约定（续）

- 文法符号
  - 次序靠后的大写字母，如：X、Y、Z
- 终结符号串
  - 次序靠后的小写字母，如：u、v、…、z
- 文法符号串
  - 小写的希腊字母，如：α、β、γ、δ
- 可以直接用产生式的集合代替四元组来描述文法，**第一个产生式的左部符号是文法的开始符号**。

四、推导和短语

例：考虑简单算术表达式的文法G：

$G=(\{+, *, (,), i\}, \{E, T, F\}, E, \varphi)$

φ：

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid i$

- 文法所产生的语言

  从文法的开始符号出发，反复连续使用产生式对非终结符号进行替换和展开，就可以得到该文法定义的语言。

推导

假定$A \to \gamma$是一个产生式，α和β是任意的文法符号串，则有：　　$\alpha A \beta \Rightarrow \alpha \gamma \beta$

- “$\Rightarrow$”表示　“一步推导”

  即利用产生式对左边符号串中的一个非终结符号进行替换，得到右边的符号串。

- 称$\alpha A \beta$直接推导出$\alpha \gamma \beta$
- 也可以说$\alpha \gamma \beta$是$\alpha A \beta$的直接推导
- 或说$\alpha \gamma \beta$直接归约到$\alpha A \beta$

如果有直接推导序列：$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$

则说α_1推导出α_n，记作：$\alpha_1 \overset{*}{\Rightarrow} \alpha_n$

称这个序列是从α_1到α_n的**长度为n的推导**

“$\overset{*}{\Rightarrow}$”表示0步或多步推导

从文法开始符号E推导出符号串i+i的详细过程

αAβ	αγβ	α	β	所用产生式	从E到 αγβ 的推导长度
E	E+T	ε	ε	E→E+T	1
E+T	T+T	ε	+T	E→T	2
T+T	F+T	ε	+T	T→F	3
F+T	i+T	ε	+T	F→i	4
i+T	i+F	i+	ε	T→F	5
i+F	i+i	i+	ε	F→i	6

$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$

最左推导、最右推导

最左推导

如果 $\alpha \overset{*}{\Rightarrow} \beta$，并且在每“一步推导”中，都替换α中**最左边**的非终结符号，则称这样的推导为最左推导。记作：$\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$

$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$

最右推导

如果 $\alpha \overset{*}{\Rightarrow} \beta$，并且在每“一步推导”中，都替换α中**最右边**的非终结符号，则称这样的推导为最右推导。记作：$\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$

最右推导也称为**规范推导**

$E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+i \Rightarrow T+i \Rightarrow F+i \Rightarrow i+i$

句型、句子和语言

句型

对于文法G=(V_T, V_N, S, φ)，如果$S \overset{*}{\Rightarrow} \alpha$，则称α是当前文法的一个句型。

若$S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$，则α是当前文法的一个**左句型**，

若$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$，则α是当前文法的一个**右句型**。

句子

仅含有终结符号的句型是文法的一个**句子**。

语言

文法G产生的所有句子组成的集合是文法G所定义的**语言**，记作L(G)。

$$L(G)=\{\ \alpha \mid\ S \overset{+}{\Rightarrow} \alpha，并且\ \alpha \in V_T^*\ \}$$

短语、直接短语和句柄

- 对于文法$G=(V_T, V_N, S, \varphi)$，假定$\alpha\beta\delta$是文法G的一个句型，如果存在：

$$S \overset{*}{\Rightarrow} \alpha A\delta，并且\ A \overset{+}{\Rightarrow} \beta$$

则称β是句型$\alpha\beta\delta$关于非终结符号A的**短语**。

- 如果存在：

$$S \overset{*}{\Rightarrow} \alpha A\delta，并且\ A \Rightarrow \beta$$

则称β是句型$\alpha\beta\delta$关于非终结符号A的**直接短语**。

- 一个句型的最左直接短语称为该句型的**句柄**。

例：

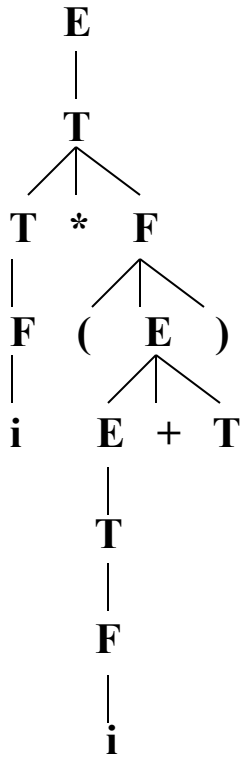
$$\underset{①}{E}\Rightarrow\underset{②}{T}\Rightarrow\underset{③}{T*F}\Rightarrow\underset{④}{T*(E)}\Rightarrow\underset{⑤}{F*(E)}\Rightarrow\underset{⑥}{i*(E)}\Rightarrow\underset{⑦}{i*(E+T)}\Rightarrow\underset{⑧}{i*(T+T)}\Rightarrow\underset{⑨}{i*(F+T)}\Rightarrow\underset{⑩}{i*(i+T)}$$

五、分析树及二义性

- 分析树
- 子树
- 子树与短语之间的关系
- 二义性

分析树

- 推导的图形表示，又称推导树。
- 一棵有序有向树，因此具有树的性质；
- 分析树的特点：每一个结点都有标记。
  - 根结点由文法的开始符号标记；
  - 每个内部结点由非终结符号标记，它的子结点由这个非终结符号的这次推导所用产生式的右部各符号从左到右依次标记；
  - 叶结点由非终结符号或终结符号标记，它们从左到右排列起来，构成句型。



$E \Rightarrow T \Rightarrow T*F \Rightarrow T*(E) \Rightarrow F*(E) \Rightarrow i*(E)$

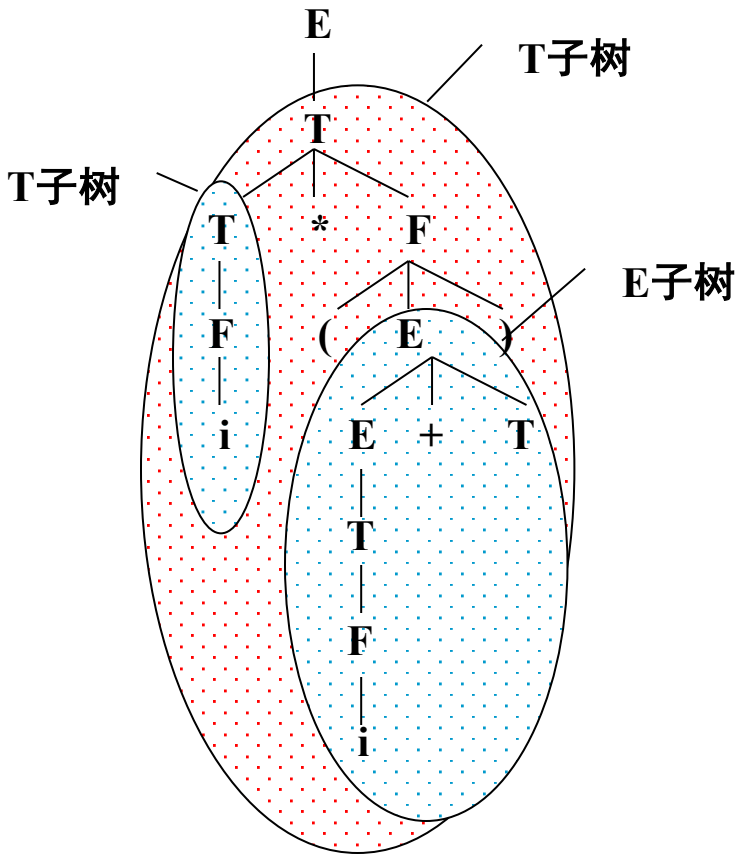
$\Rightarrow i*(E+T) \Rightarrow i*(T+T) \Rightarrow i*(F+T) \Rightarrow i*(i+T)$

$E \Rightarrow T \Rightarrow T*F \Rightarrow F*F \Rightarrow i*F \Rightarrow i*(E)$

$\Rightarrow i*(E+T) \Rightarrow i*(T+T) \Rightarrow i*(F+T) \Rightarrow i*(i+T)$

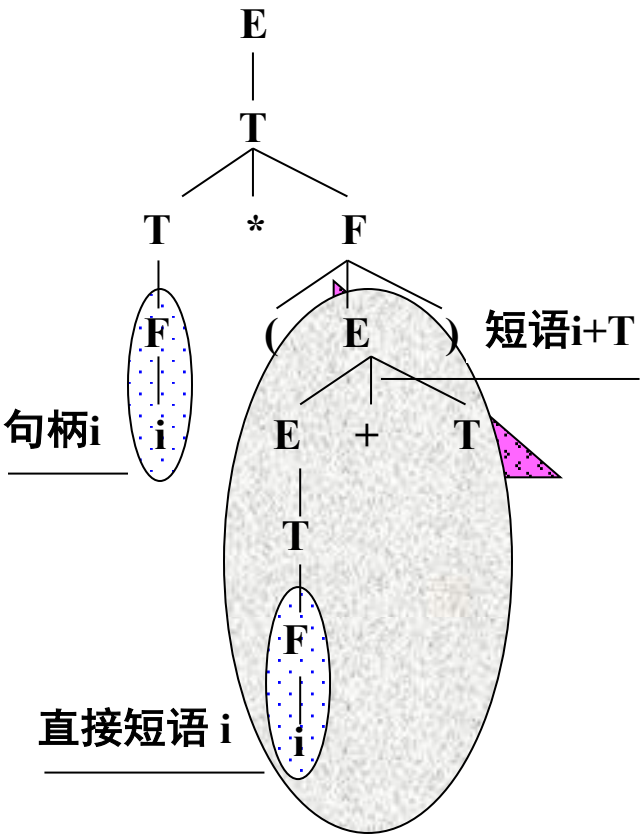
子树

- 分析树中一个特有的结点、连同它的全部后裔结点、连接这些结点的边、以及这些结点的标记。
- 子树的根结点的标记可能不是文法的开始符号。
- 如果子树的根结点标记为非终结符号A，则可称该子树为A-子树。



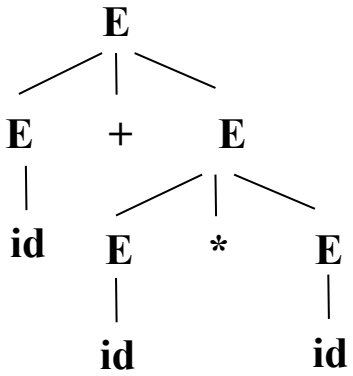
子树与短语的关系

- 一棵子树的所有叶结点自左至右排列起来，形成此句型相对于该子树根的短语；
- 分析树中只有父子两代的子树的所有叶结点自左至右排列起来，形成此句型相对于该子树根的直接短语；
- 分析树中最左边的那棵只有父子两代的子树的所有叶结点自左至右排列起来，就是该句型的句柄。



二义性

- 如果一个文法的某个句子有不止一棵分析树，则这个句子是二义性的。
- 含有二义性句子的文法是二义性的文法。



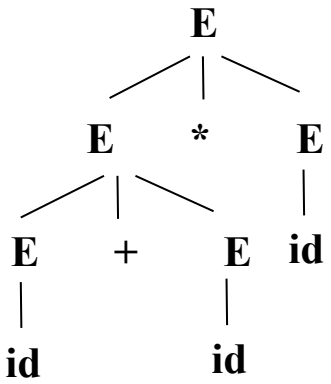
例：考虑文法 $G=(\{+,*,(,),i\},\{E\},E,\varphi)$

φ： $E \rightarrow E+E \mid E*E \mid (E) \mid id$

句子 id+id*id 存在两个不同的最左推导：

$E \Rightarrow E+E \Rightarrow id+E \Rightarrow id+E*E \Rightarrow id+id*E \Rightarrow id+id*id$

$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow id+E*E \Rightarrow id+id*E \Rightarrow id+id*id$



有两棵不同的分析树

文法的二义性和语言的二义性

- 如果两个文法产生的语言相同，即L(G)=L(G′)，则称这两个文法是等价的。
- 有时，一个二义性的文法可以变换为一个等价的、无二义性的文法。
- 有些语言，根本就不存在无二义性的文法，这样的语言称为二义性的语言。
- 二义性问题是不可判定的
  - 不存在一种算法，它能够在有限的步骤内确切地判定出一个文法是否是二义性的。
  - 可以找出一些充分条件（未必是必要条件），当文法满足这些条件时，就可以确信该文法是无二义性的。

六、文法的变换

- 文法二义性的消除
- 左递归的消除
- 提取左因子

文法二义性的消除

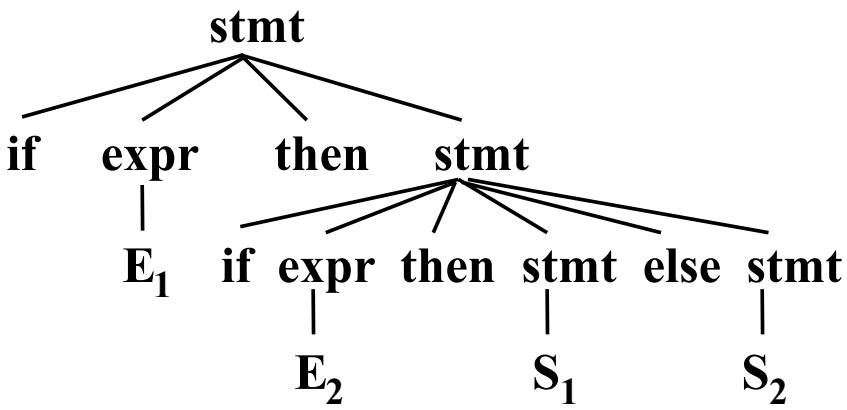
- 映射程序设计语言中IF语句的文法：

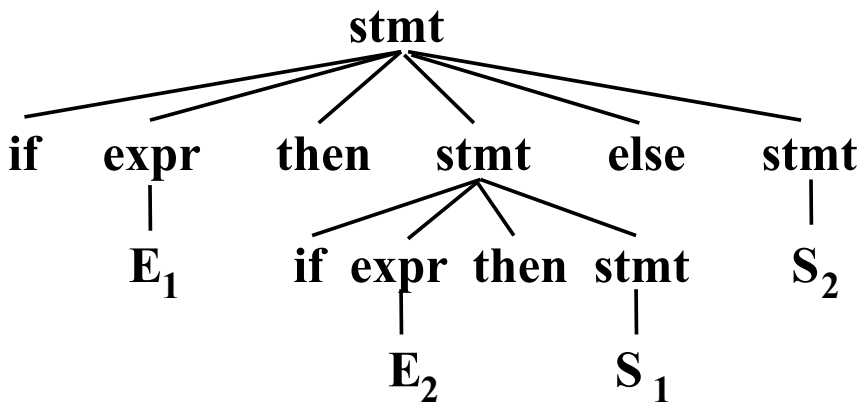
  stmt → if ***expr*** then ***stmt***

  | if ***expr*** then ***stmt*** else ***stmt***

  | other

- 句子if E_1 then if E_2 then S_1 else S_2有两棵不同的分析树：





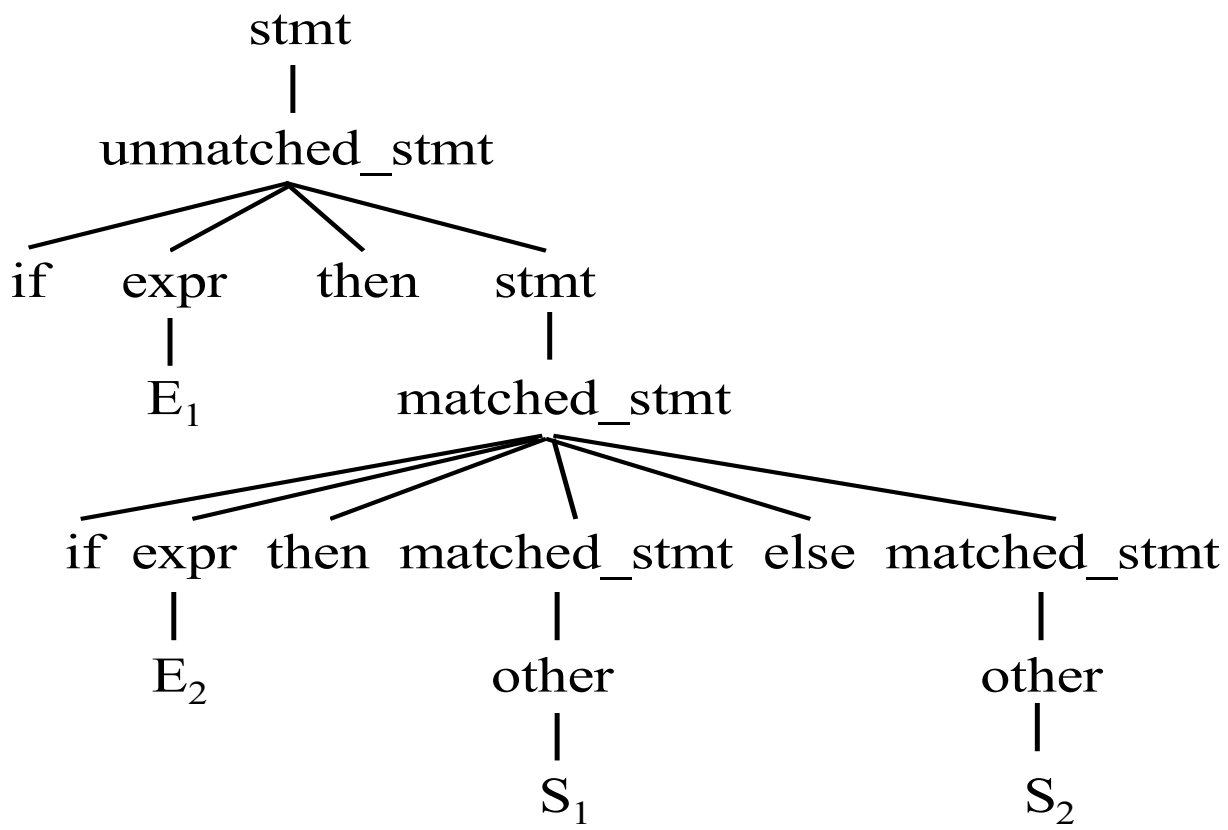
利用“最近最后匹配原则”

else必须匹配离它最近的那个未匹配的then。

- 出现在then和else之间的语句必须是“匹配的”。
- 所谓匹配的语句是不包含不匹配语句的if-then-else语句，或是其他任何非条件语句。
- 改写后的文法：

stmt→*matched_stmt* | *unmatched_stmt*
matched_stmt→if *expr* then *matched_stmt* else *matched_stmt*
| other
unmatched_stmt→ if *expr* then *stmt*
| if *expr* then *matched_stmt* else *unmatched_stmt*

句子if E_1 then if E_2 then S_1 else S_2的分析树



左递归的消除

一个文法是左递归的，如果它有非终结符号A，对某个文法符号串α，存在推导：

$$A \overset{+}{\Rightarrow} A\alpha$$

若存在某个α=ε，则称该文法是有环路的。

- 消除直接左递归的方法：

简单情况：如果文法G有产生式：$A \to A\alpha \mid \beta$

可以把A的这两个产生式改写为：$A \to \beta A'$

$$A' \to \alpha A' \mid \varepsilon$$

这两组产生式是等价的

由于从A推导出的符号串是相同的，即都是βα...α

示例：消除直接左递归

- 例：消除表达式文法中的左递归：

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow (E) \mid id$$

- 利用消除直接左递归的方法，可以改写为：

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E) \mid id$$

消除直接左递归（续）

- **一般情况：** 假定关于A的全部产生式是：

  $A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$

- 产生式可以改写为：

  $A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$

  $A' \rightarrow \alpha_1 A' | \alpha_2\ A' | \ldots | \alpha_m A' | \varepsilon$

消除间接左递归

- 例如有间接左递归文法：

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

算法：消除左递归

输入：无环路、无ε-产生式的文法G

输出：不带有左递归的、与G等价的文法G'

方法：

(1)把文法G的所有非终结符号按某种顺序排列成$A_1,A_2,\ldots,A_n$

(2)for (i=1; i<=n; i++)

for (j=1; j<=i-1; j++)

if ($A_j\rightarrow\delta_1|\delta_2|\ldots|\delta_k$是关于当前$A_j$的所有产生式) {

把每个形如$A_i\rightarrow A_j\gamma$的产生式改写为：$A_i\rightarrow\delta_1\gamma|\delta_2\gamma|\ldots|\delta_k\gamma$;

消除关于A_i的产生式中的直接左递归;

}

(3)化简第(2)步得到的文法，即去除无用的非终结符号和产生式。

这种方法得到的非递归文法可能含有ε-产生式。

为含有ε-产生式的文法$G=(V_T, V_N, S, \varphi)$ 构造不含ε-产生式的文法$G'=(V_T', V_N', S, \varphi')$的方法

- 若产生式$A\to X_1X_2\ldots X_n\in\varphi$
  则把产生式$A\to\alpha_1\alpha_2\ldots\alpha_n$加入$\varphi'$
  其中：X_i、α_i为文法符号，即X_i、$\alpha_i\in(V_T\cup V_N)$

  若X_i不能产生ε，则$\alpha_i=X_i$
  若X_i能产生ε，则$\alpha_i=X_i$ 或 $\alpha_i=\varepsilon$
  注意：不能所有的α_i都取ε
- 文法G′满足：

$$L(G')=\begin{cases} L(G)-\{\varepsilon\} & \text{如果}L(G)\text{中含有}\varepsilon \\ L(G) & \text{如果}L(G)\text{中不含有}\varepsilon \end{cases}$$

- 一个文法是ε-无关的，
  - 如果它没有ε-产生式（即形如$A\to\varepsilon$的产生式），或者
  - 只有一个ε-产生式，即$S\to\varepsilon$，并且文法的开始符号S不出现在任何产生式的右部。

示例：消除下面文法中的左递归

S→Aa|b

A→Ac|Sd|ε

- 首先，必须保证此文法中无环路、无ε-产生式。
- 改写为无ε-产生式的文法：

  S→Aa|a|b

  A→Ac|c|Sd

- 消除其中的左递归：

第一步，把文法的非终结符号排列为S、A；

第二步，由于S不存在直接左递归，所以算法第2步在i=1时不做工作；

在i=2时，把产生式S→Aa|a|b代入A的有关产生式中，得到：

A→Ac|c|Aad|ad|bd

消除A产生式中的直接左递归，得到文法：

S→Aa|a|b

A→cA′|adA′|bdA′

A′→cA′|adA′|ε

注意产生式的书写顺序。
开始符号不能改变。

提取左公因子

- 如有产生式 $A \rightarrow \alpha\beta_1 | \alpha\beta_2$

  提取左公因子α，则原产生式变为：

  $A \rightarrow \alpha A'$

  $A' \rightarrow \beta_1 | \beta_2$

- 若有产生式 $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_n | \gamma$

  可用如下的产生式代替：

  $A \rightarrow \alpha A' | \gamma$

  $A' \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$

示例：映射程序设计语言中IF语句的文法

stmt→ if *expr* then *stmt*
| if *expr* then *stmt* else *stmt*
| a
expr→b

- 左公因子 if *expr* then *stmt*
- 提取左公因子，得到文法：

stmt→if *expr* then *stmt S′* | a
S′→ else *stmt* | ε
expr→b

2.2 有限自动机

- 有限自动机是具有离散输入与输出的系统的一种数学模型
- 系统可处于有限个内部状态的任何一个之中
- 系统的当前状态概括了有关过去输入的信息
- 例：自动电梯的控制机构
- “确定的有限自动机”指，在当前状态下，输入一个符号，有限自动机转换到唯一的下一个状态，称为后继状态。
- “非确定的有限自动机”指，在当前状态下输入一个符号，可能有两种以上可选择的后继状态，并且非确定的有限自动机所对应的状态转换图可以有标记为ε的边。

有限自动机

一、确定的有限自动机（DFA）

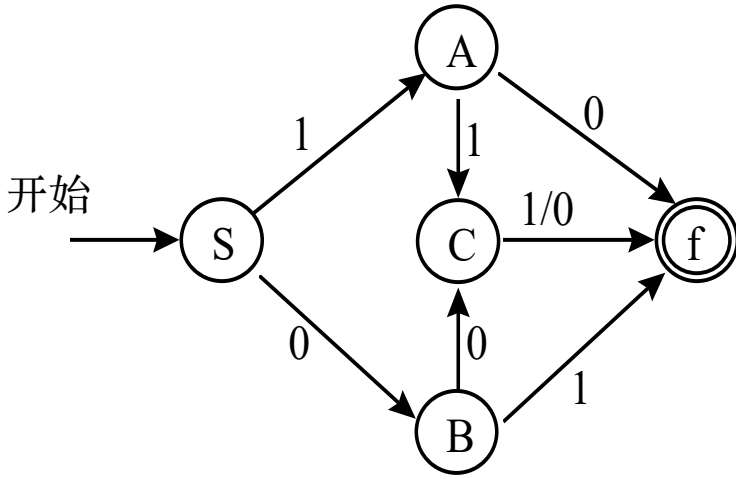
二、非确定的有限自动机（NFA）

三、具有ε-转移的非确定的有限自动机

四、DFA的化简

一、确定的有限自动机（DFA）

■ 状态转换图



- 一张有限的方向图
- 图中结点代表状态，用圆圈表示
- 只含有限个状态，有一个初始状态，可以有若干个终结状态，终态用双圆圈表示。
- 状态之间用有向边连接
- 边上的标记表示在射出结点状态下可能出现的输入符号

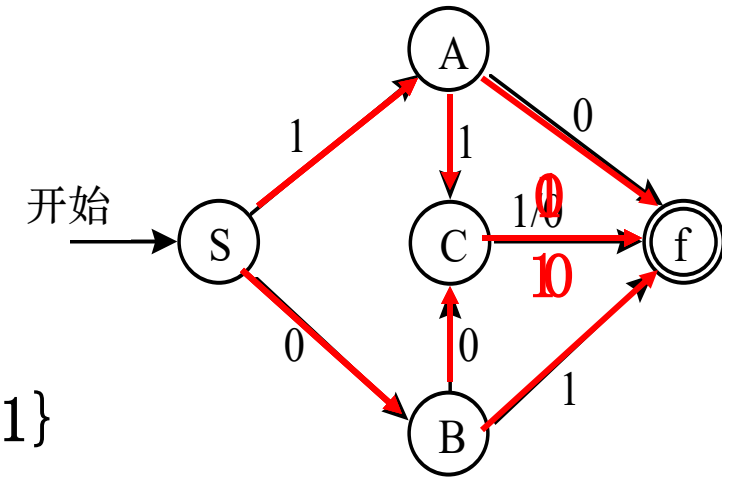
利用状态转换图，识别符号串

- **识别方法：**

(1) 起点为初态S，从ω的最左符号开始，重复步骤(2)，直到达到ω的最右符号为止。

(2) 扫描ω的下一个符号，在当前状态的所有射出边中找出标记为该字符的边，沿此边过度到下一个状态。

- **状态转换图所能识别的符号串的全体称为该状态转换图所识别的语言。**



$L(M)=\{10, 110, 111, 01, 000, 001\}$

确定的有限自动机的定义

- 一个确定的有限自动机M(记作：DFA M)是一个五元组：　M=(∑，Q，q_0，F，δ)

  其中　∑：是一个字母表，它的每个元素称为一个输入符号

  Q：是一个有限的状态集合

  $q_0 \in Q$：q_0称为初始状态

  $F \subseteq Q$：F称为终结状态集合

  δ：是一个从Q×∑到Q的单值映射

- 转换函数δ(q, a)=q′（其中q, q′∈Q, a∈∑）表示当前状态为q，输入符号为a时，自动机将转换到下一个状态q′，q′称为q的一个后继。
- 若Q={q_1, q_2, …, q_n}，∑={a_1, a_2, …, a_m}，则Q×∑=$(\delta(q_i, a_j))_{n\times m}$是一个n行m列的矩阵，它称为DFA M的状态转换矩阵，也称为转换表。

示例： 有 DFA M=({0, 1}, {A, B, C, S, f}, S, {f}, δ)

其中 δ(S, 0)=B　δ(A, 0)=f　δ(B, 0)=C　δ(C, 0)=f

δ(S, 1)=A　δ(A, 1)=C　δ(B, 1)=f　δ(C, 1)=f

- 状态转换矩阵：
  - 5行2列的矩阵
  - 每一行对应M的一个状态
  - 每一列对应M的一个输入符号

	0	1
S	B	A
A	f	C
B	C	f
C	f	f
f	-	-

- DFA M可用一张状态转换图来表示：
  - 若DFA M有n个状态，m个输入符号，则状态转换图有n个状态结点，每个结点最多有m条射出边。
  - 若q、q′∈Q,a∈∑，并且δ(q,a)=q′，则从q到q′有一条标记为a的有向边。
  - 整个图含有唯一的一个初态。

DFA M所识别的语言

- 对∑上的任何符号串ω∈∑*，若存在一条从初态结点到终态结点的路径，该路径上每条边的标记连接成的符号串恰好是ω，则称ω为DFA M所识别。
- DFA M所能识别的符号串的全体记为L(M)，称为 DFA M 所识别的语言。

- 如果我们对所有ω∈∑*，递归地扩张δ的定义：

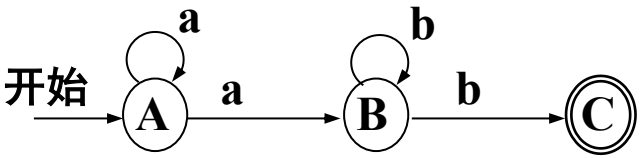
  对任何a∈∑，q∈Q 定义：

  $\delta(q, \varepsilon)=q$

  $\delta(q, \omega a)=\delta(\delta(q, \omega), a)$

$L(M)=\{\ \omega \mid \omega\in\Sigma^*$，并且存在$q\in F$，使$\delta(q_0, \omega)=q\ \}$

二、非确定的有限自动机（NFA）



所识别的语言是 $L(M)=\{a^+b^+\}$

NFA的定义：

一个非确定的有限自动机M(记作：NFA M)是一个五元组

$$M=(\Sigma, Q, q_0, F, \delta)$$

其中 Σ：是一个字母表，它的每个元素称为一个输入符号

Q：是一个有限状态集合

$q_0 \in Q$：q_0称为初始状态

$F \subseteq Q$：F称为终结状态集合

δ：是一个从$Q\times\Sigma$到Q的子集的映射， 即$\delta: Q\times\Sigma \to 2^Q$

其中2^Q是Q的幂集，也就是Q的所有子集组成的集合。

NFA的状态转换图及识别的语言

- 状态转换图
  - NFA M含有n个状态，m个输入符号
  - 图中含有n个状态结点，每个结点可射出若干条边
  - 图中有唯一的一个初态结点
- 对∑上的任何符号串$\omega \in \Sigma^*$，若存在一条从初态结点到终态结点的路径，该路径上每条边的标记连接成的符号串恰好是ω，则称ω为NFA M所识别。
- NFA M所能识别的符号串的全体记为L(M)，称为NFA M所识别的语言。
- 如果$q_0 \in F$
  - 存在一条从初态结点到终态结点的ε-道路
  - 空串ε可为该NFA M所识别，即 $\varepsilon \in L(M)$

NFA示例：

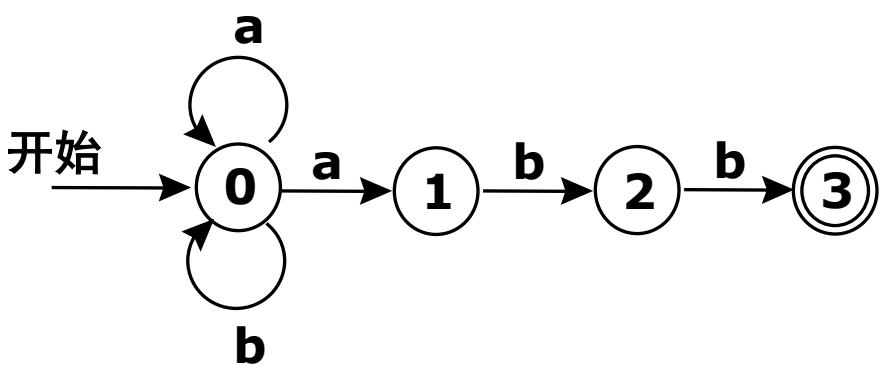
设有 NFA M=({a, b}, {0, 1, 2, 3}, 0, {3}, δ)

其中 δ(0, a)={0, 1}　δ(0, b)={0}　δ(1, b)={2}　δ(2, b)={3}

- 状态转换矩阵：

	a	b
0	{0,1}	{0}
1	-	{2}
2	-	{3}
3	-	-

- 状态转换图：



- NFA M所识别的语言：

$$L(M)=\{(a|b)^*abb\}$$

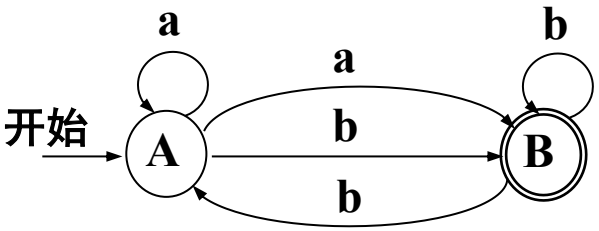
定理：对任何一个NFA M，都存在一个与之等价的DFA D，即L(M)=L(D)。

例：构造与下面的 NFA M 等价的 DFA D

NFA M=（{a, b}，{A, B}，A，{B}，δ）

其中δ：δ(A, a)={A, B}　δ(A, b)={B}　δ(B, b)={A, B}

- **首先，画出该NFA M的状态转换图**



- **假设DFA D=（{a, b}, Q′, q_0', F′, δ′）**

Q′：Q的所有子集组成的集合，即$Q'=2^Q$

Q′={Φ, {A}, {B}, {A. B}}

$q_0'=\{q_0\}$

q_0'={A}

F′：所有含有原NFA M终态的Q的子集组成的集合

F′={{B}，{A，B}}

δ′的构成

$$\delta'(\{q_1, q_2, \cdots, q_k\}, a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \cdots \cup \delta(q_k, a)$$

$\delta'(\Phi, a) = \Phi$

$\delta'(\{A\}, a) = \delta(A, a) = \{A, B\}$

$\delta'(\{B\}, a) = \delta(B, a) = \Phi$

$\delta'(\Phi, b) = \Phi$

$\delta'(\{A\}, b) = \delta(A, b) = \{B\}$

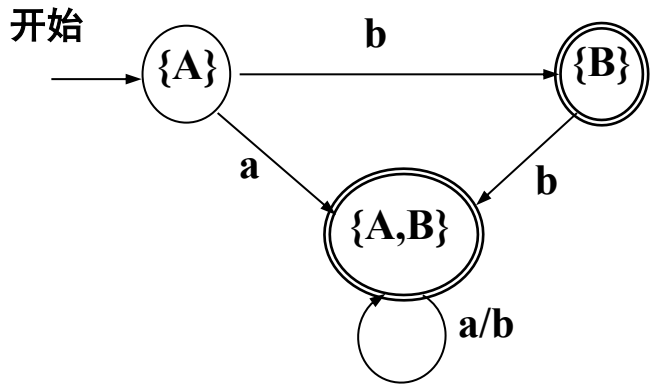
$\delta'(\{B\}, b) = \delta(B, b) = \{A, B\}$

$\delta'(\{A, B\}, a) = \delta(A, a) \cup \delta(B, a) = \{A, B\} \cup \Phi = \{A, B\}$

$\delta'(\{A, B\}, b) = \delta(A, b) \cup \delta(B, b) = \{B\} \cup \{A, B\} = \{A, B\}$

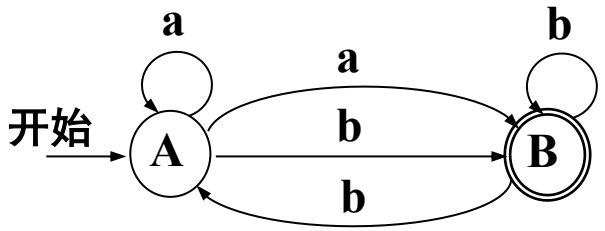
- DFA D的状态转换矩阵和状态转换图

	a	b
{A}	{A, B}	{B}
{B}	–	{A, B}
{A, B}	{A, B}	{A, B}



子集构造法：构造与NFA M等价的DFA D

- 列出NFA M的每个子集及该子集相对于每个输入符号的后继子集
- 对所有子集重新命名，得到DFA D的状态转换矩阵。

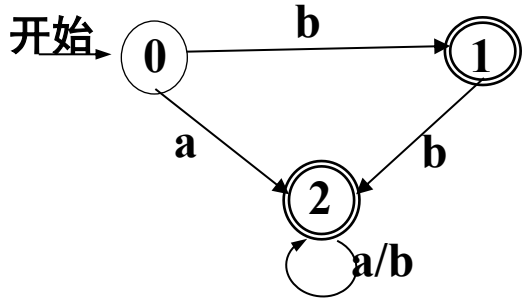


- NFA M的状态转换矩阵

状态子集 \ 输入	a	b
{A}	{A, B}	{B}
{B}	–	{A, B}
{A, B}	{A, B}	{A, B}

- DFA D的状态转换矩阵

状态子集 \ 输入	a	b
0	2	1
1	–	2
2	2	2



三、具有ε-转移的非确定有限自动机

定义：一个具有ε-转移的非确定有限自动机M(记作：NFA M)是一个五元组 $M=(\Sigma, Q, q_0, F, \delta)$

其中 Σ：是一个字母表，它的每个元素称为一个输入符号

Q：是一个有限的状态集合

$q_0 \in Q$：q_0称为初始状态

$F \subseteq Q$：F称为终结状态集合

δ：是一个从$Q\times(\Sigma \cup \{\varepsilon\})$到Q的子集的映射，即$\delta: Q\times(\Sigma \cup \{\varepsilon\}) \to 2^Q$

- 对任何$q \in Q$及$a \in (\Sigma \cup \{\varepsilon\})$，转移函数$\delta$的值具有如下的形式

  $\delta(q, a)=\{q_1, q_2, \ldots, q_k\}$ 其中 $q_i \in Q$ $(i=1, 2, \ldots, k)$

- NFA 的状态转换图
  - 图中可能有标记为ε的边
  - 当$\delta(q, \varepsilon)=\{q_1, q_2, \ldots, q_k\}$时，从q出发有k条标记为ε的边分别指向$q_1, q_2, \ldots, q_k$。

示例：

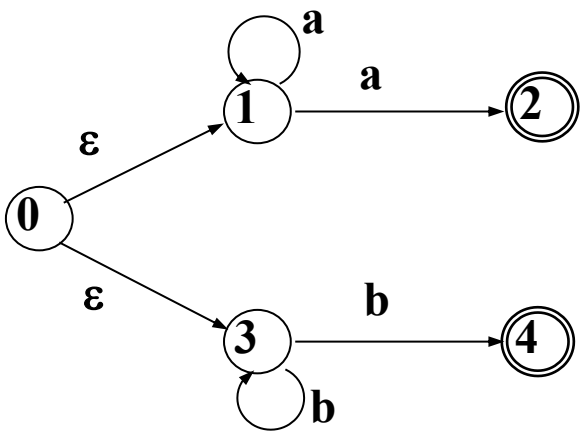
有NFA M=({a, b}, {0, 1, 2, 3, 4}, 0, {2, 4}, δ)

其中 δ(0, ε)={1, 3}　δ(1, a)={1, 2}　δ(3, b)={3, 4}

- NFA M的状态转换矩阵

	ε	a	b
0	{1,3}	–	–
1	–	{1,2}	–
2	–	–	–
3	–	–	{3,4}
4	–	–	–

- NFA M的状态转换图



- NFA M所识别的语言为L(M)={ $a^+|b^+$ }。

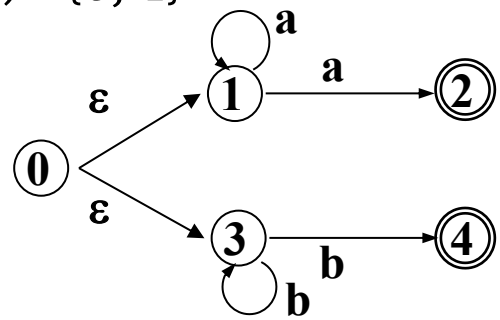
定理：对任何一个具有ε-转移的NFA M，都存在一个等价的不具有ε-转移的NFA N，即L(M)=L(N)。

例：构造与NFA M 等价的不具有ε-转移的NFA N

设 NFA M=({a, b}, {0, 1, 2, 3, 4}, 0, {2, 4}, δ)

其中 δ(0, ε)={1, 3}　δ(1, a)={1, 2}　δ(3, b)={3, 4}

状态转换图如右：



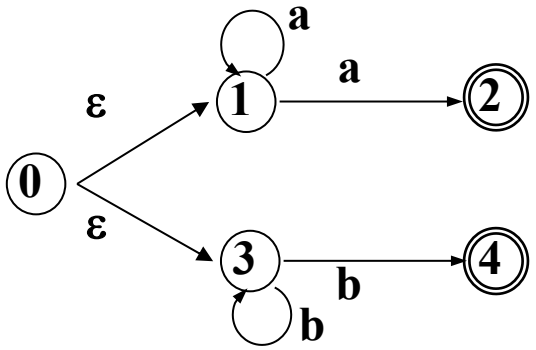
假设 NFA N=({a, b}, {0, 1, 2, 3, 4}, 0, F′, δ′)

- **ε_closure(q)：从状态q出发，经过ε-道路可以到达的所有状态的集合。**
- **F′的构成：**

$$F'=\begin{cases} F\cup\{q_0\} & \text{如果}\varepsilon\_closure(q_0)\text{中包含F的一个终态} \\ F & \text{否则} \end{cases}$$

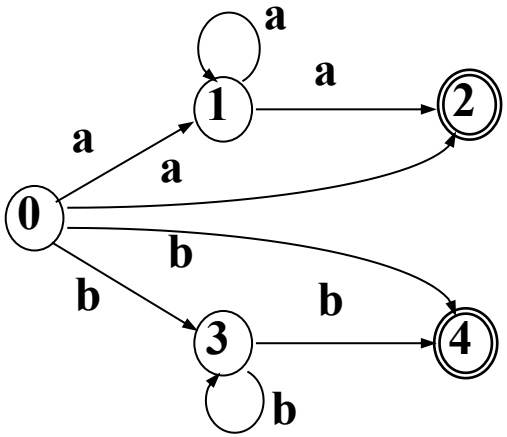
由于ε_closure(0)={0,1,3}，不包含NFA M的终态，因此F′=F={2, 4}

δ′的构成：δ′(q, a)={q′| q′为从q出发，经过标记为a的道路所能到达的状态 }



NFA N的状态转换矩阵和状态转换图如下：

	a	b
0	{1, 2}	{3, 4}
1	{1, 2}	–
2	–	–
3	–	{3, 4}
4	–	–



δ′(0, a)={1, 2}
δ′(1, a)={1, 2}
δ′(2, a)=Φ
δ′(3, a)=Φ
δ′(4, a)=Φ

δ′(0, b)={3, 4}
δ′(1, b)=Φ
δ′(2, b)=Φ
δ′(3, b)={3, 4}
δ′(4, b)=Φ

推论：对于任何一个具有ε-转移的NFA M，都存在一个与之等价的DFA D，即L(M)=L(D)。

- DFA D的每个状态对应NFA M的一个状态子集。
- 对给定的输入符号串，为了使D“并行地”模拟M所能产生的所有可能的转换，令q为NFA M的状态，T为NFA M 的状态子集，引入以下操作：

ε_closure(q)={q′ |从q出发，经过ε-道路可以到达状态q′}

$$\varepsilon\_closure(T)=\bigcup_{i=1}^{n}\varepsilon\_closure(q_i) \quad \text{其中 } q_i\in T$$

从T中任一状态出发，经过ε-道路后可以到达的状态集合。

$$move(T,a)=\{q \mid \delta(q_i,a)=q, \text{ 其中} q_i\in T\}$$

从某个状态$q_i\in T$出发，经过输入符号a之后可到达的状态集合。

算法：计算ε_closure(T)

```
把T中所有状态压入栈;
ε_closure(T)的初值置为T;
while  栈不空
{
    弹出栈顶元素t;
    for  (each  q∈ε_closure(t))
        if  (q∉ε_closure(T)) {
            把q加入ε_closure(T);
            把q压入栈;
         }
}
```

算法：为NFA构造等价的DFA

输入：一个NFA M

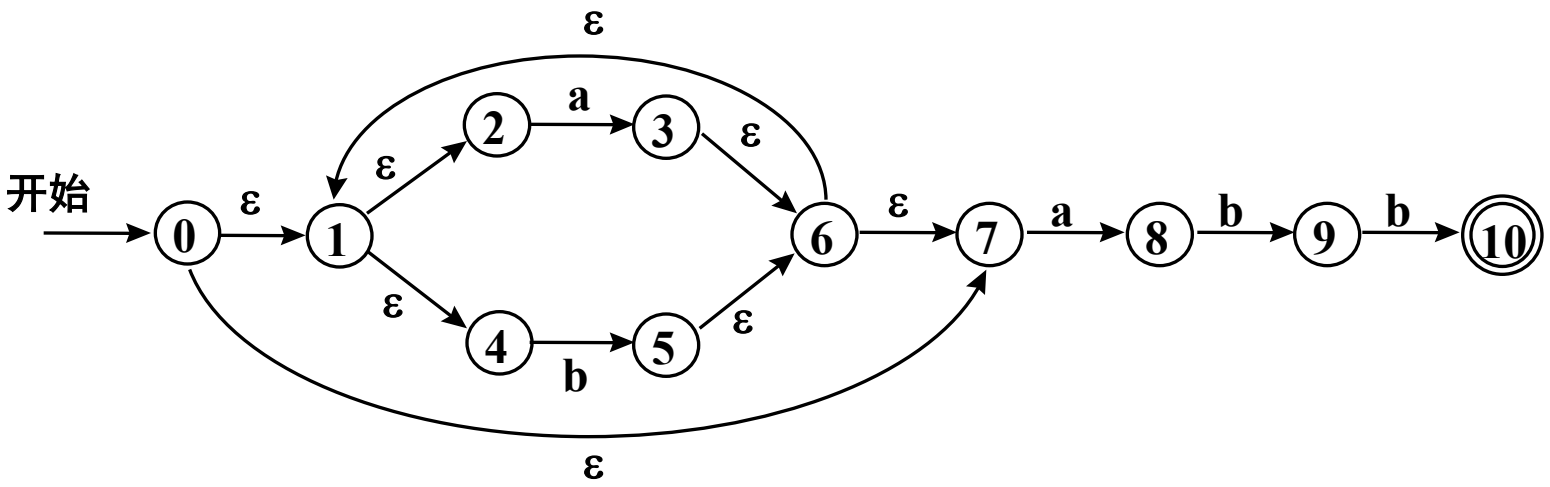
输出：一个与NFA M等价（即接受同样语言）的DFA D

方法：为DFA D构造状态转换表DTT

初态：ε_closure(q_0)是DQ中唯一的状态，且未标记。

```
while（DQ中存在一个未标记的状态T)
{
    标记 T;
     for （each  a∈∑)
     {
         U=ε_closure(move(T,a));
         if （U∉DQ)
             把U做为一个未标记的状态加入DQ;
         DTT[T,a]：=U;
      }
 }
```

示例：构造与下面的NFA M等价的DFA D。



- 字母表∑={a, b}
- 初态为A，A=ε_closure(0)={0, 1, 2, 4, 7}

DTT[A, a]=ε_closure(move(A, a))

=ε_closure(move(0, a) ∪move(1, a) ∪move(2, a) ∪move(4, a) ∪move(7, a))

=ε_closure({3, 8})=ε_closure(3) ∪ε_closure(8)={1, 2, 3, 4, 6, 7, 8}=B

DTT[A, b]=ε-closure(move(A, b))=ε-closure(5)={1, 2, 4, 5, 6, 7}=C

DTT[B, a]=ε-closure(move(B, a))=ε-closure({3, 8})=B

DTT[B, b]=ε-closure(move(B, b))=ε-closure({5, 9})={1, 2, 4, 5, 6, 7, 9}=D

示例（续）

DTT[C,a] = ε-closure(move(C,a)) = ε-closure({3,8}) = B

DTT[C,b] = ε-closure(move(C,b)) = ε-closure(5) = C

DTT[D,a] = ε-closure(move(D,a)) = ε-closure({3,8}) = B

DTT[D,b] = ε-closure(move(D,b)) = ε-closure({5,10})

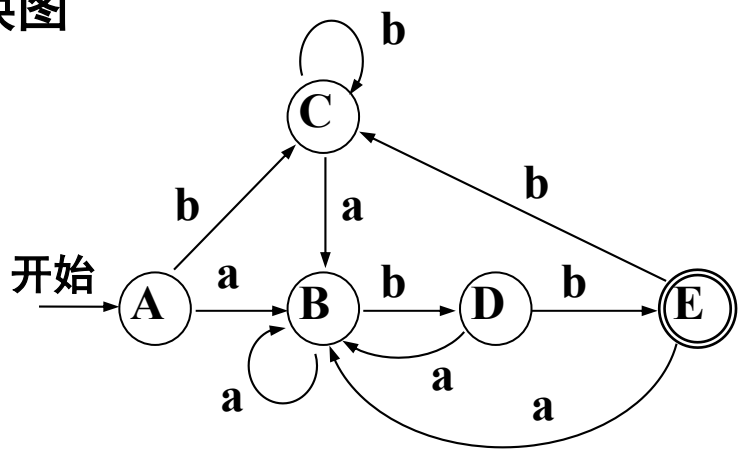
= {1,2,4,5,6,7,10} = E

DTT[E,a] = ε-closure(move(E,a)) = ε-closure({3,8}) = B

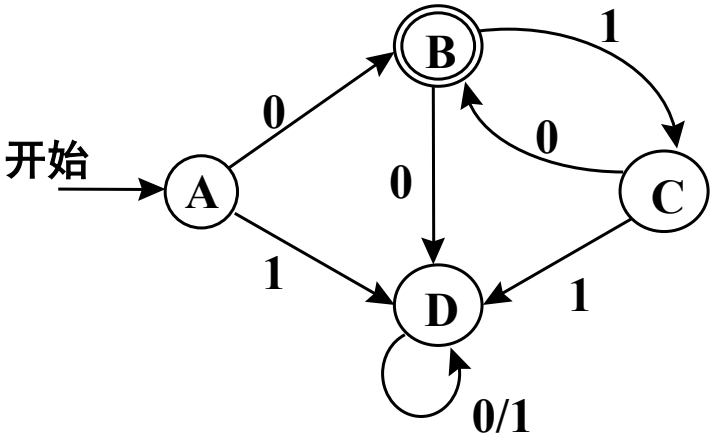
DTT[E,b] = ε-closure(move(E,b)) = ε-closure(5) = C

- DFA D有5个状态，即A、B、C、D、E，
  - 其中A为初态
  - E为终态，因为E的状态集合中包括原NFA M的终态10。
- DFA D的状态转换矩阵和状态转换图

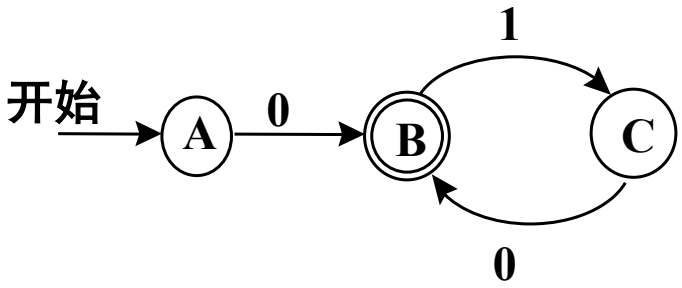
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



四、DFA的化简



- 状态D是一个“死状态”，是一个无用的状态。
- 去掉状态D及与之相连接的边，可以得到等价的状态转换图



- L(M)=0(10)*
- 对于任何一个含有n个状态的DFA，都存在含有m(m>n)个状态的DFA与之等价。
- DFA D的化简，指寻找一个状态数比较少的DFA D′，使L(D)=L(D′)。
- 可以证明，存在一个最少状态的DFA D″，使L(D)=L(D″)，并且这个D″是唯一的。

DFA D的最小化过程

定义： 设s,t∈Q，若对任何ω∈Σ*，

δ(s,ω)∈F 当且仅当 δ(t,ω)∈F，则称状态s和t是等价的。

否则称状态s和t是可区分的。

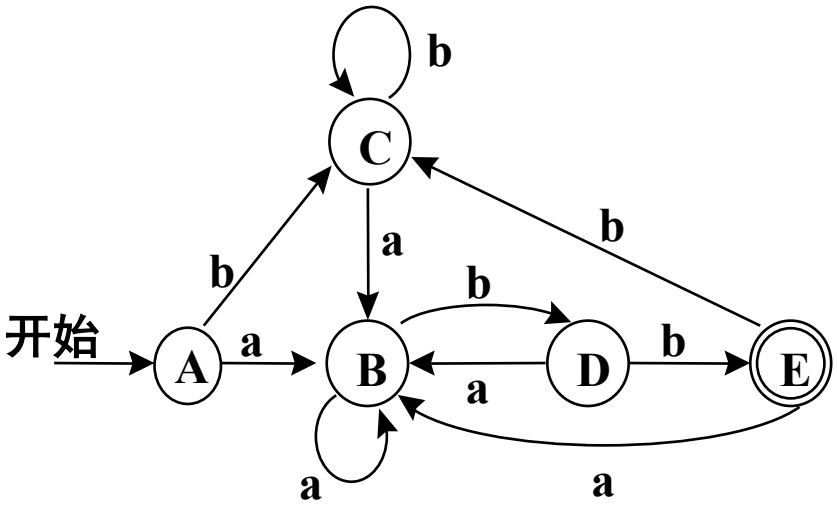
DFA D的最小化过程：

- 首先，把D的状态集合分割成一些互不相交的子集，使每个子集中的任何两个状态是等价的，而任何两个属于不同子集的状态是可区分的。
- 然后，在每个子集中任取一个状态作“代表”，删去该子集中其余的状态，并把射向其它结点的边改为射向“代表”结点。
- 最后，如果得到的DFA中有死状态、或从初态无法到达的状态，则把它们删除。

把状态集合Q分割成满足要求的子集

- 把状态集合Q划分成两个子集：终态子集F和非终态子集G。
- 对每个子集进行划分：
  - 取某个子集A=$\{s_1, s_2, \ldots, s_k\}$
  - 取某个输入符号a，检查A中的每个状态对该输入符号的转换。
  - 如果A中的状态相对于a，转换到不同子集中的状态，则要对A进行划分。使A中能够转换到同一子集的状态作为一个新的子集。
  - 重复上述过程，直到每个子集都不能再划分为止。

示例：对状态转换图所描述的DFA D最小化



第一步：把DFA D的状态集合划分为子集，使每个子集中的状态相互等价，不同子集中的状态可区分。

- 把D的状态集合划分为两个子集：{A, B, C, D}和{E}
- 考察非终态子集{A, B, C, D}
  - 对于a，状态A, B, C, D都转换到状态B，所以对输入符号a而言，该子集不能再划分。
  - 对于b，状态A, B, C都转换到子集{A, B, C, D}中的状态，而状态D则转换到子集{E}中的状态。
  - 应把子集{A, B, C, D}划分成两个新的子集{A, B, C}和{D}。

D的状态集合被划分为：{A, B, C}、{D}和{E}

- 考察子集{A, B, C}
  - 对于a，状态A, B, C都转换到状态B，所以对输入符号a而言，该子集不能再划分。
  - 对于b，状态A, C转换到C，状态B转换到D。状态C和D分属于不同的子集。
  - 应把子集{A, B, C}划分成两个新的子集{A, C}和{B}。

D的状态集合被划分为：{A, C}、{B}、{D}和{E}

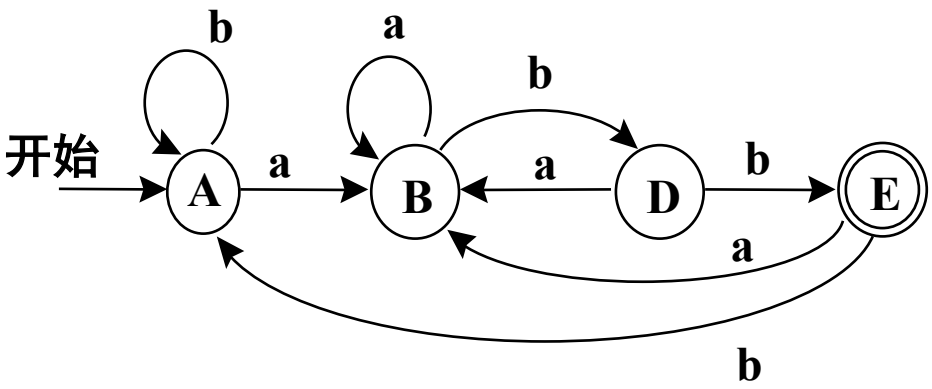
- 考察子集{A, C}
  - 对于a，状态A, C都转换到状态B。
  - 对于b，状态A, C都转换到状态C。
  - 该子集不可再划分。

D的状态集合最终被划分为：{A, C}、{B}、{D}和{E}

构造最小DFA D′

- **第二步：为每个子集选择一个代表状态。**
  - 选择A为子集{A, C}的代表状态

- D′的状态转换图



- D′的状态转换矩阵

状态	输入符号	
	a	b
A	B	A
B	B	D
D	B	E
E	B	A

2.3 正规文法与有限自动机的等价性

- 如果对于某个正规文法G和某个有限自动机M，有L(G)=L(M)，则称G和M是等价的。
- 定理：对每一个右线性文法G或左线性文法G，都存在一个等价的有限自动机M。

证明：首先考虑右线性正规文法

设给定的一个右线性文法G为：$G=(V_T, V_N, S, \varphi)$

与G等价的有限自动机M为：$M=(\Sigma, Q, q_0, F, \delta)$

$\Sigma=V_T$，$q_0=S$，$F=\{f\}$，f为新增加的一个终态符号，$f\notin V_N$，$Q=V_N\cup\{f\}$

δ的定义为：

- 若文法G有产生式$A\to a$，其中$A\in V_N$，$a\in V_T\cup\{\varepsilon\}$，则$\delta(A,a)=f$。
- 若文法G有产生式$A\to aA_1|aA_2|\ldots|aA_k$，

  其中$A, A_i\in V_N, (i=1,2,\ldots,k), a\in V_T\cup\{\varepsilon\}$，则$\delta(A,a)=\{A_1, A_2, \ldots, A_k\}$。

$\omega \in L(G)$的充分必要条件是$\omega \in L(M)$，所以$L(G)=L(M)$

- 在正规文法G中，开始符号S推导出ω的充分必要条件为：在自动机M中，从初态S到终态f有一条路径，该路径上所有边的标记依次连接起来恰好是ω。

现在考虑左线性正规文法

设给定的一个左线性文法G为：$G=(V_T, V_N, S, \varphi)$

与G等价的有限自动机M'为：$M'=(\Sigma, Q, q_0, F, \delta)$

$\Sigma=V_T$，$F=\{S\}$，新增加一个初态符号q_0，$q_0 \notin V_N$，$Q=V_N \cup \{q_0\}$

δ的定义为：

- 若文法G有产生式$A \to a$，其中$A \in V_N$，$a \in V_T \cup \{\varepsilon\}$，则$\delta(q_0, a)=A$。
- 若文法G有产生式$A_1 \to Aa$，$A_2 \to Aa$，…，$A_k \to Aa$，
  其中$A, A_i \in V_N, (i=1,2,\ldots,k), a \in V_T \cup \{\varepsilon\}$，则$\delta(A, a)=\{A_1, A_2, \ldots, A_k\}$

可以证明$L(G)=L(M')$，即

有限自动机M'与左线性文法G是等价的。

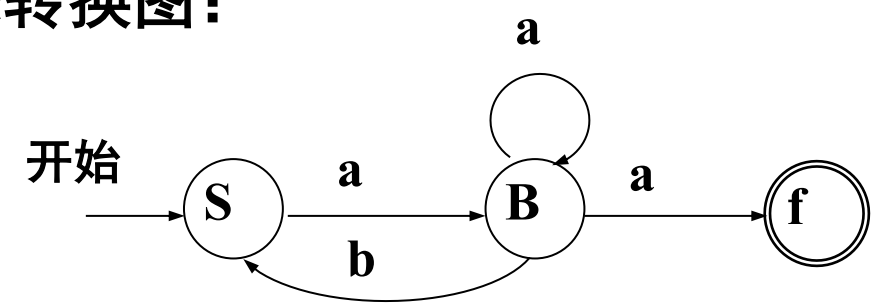
示例：设有右线性文法G=({a, b}， {S, B}， S， φ)，其中φ：

$S \rightarrow aB$

$B \rightarrow aB \mid bS \mid a$

试构造与G等价的有限自动机M。

- 设FA M=(∑， Q， q_0， F， δ)
- ∑={a, b}　q_0=S　F={f}　Q={S, B, f}
- 转换函数δ：
  - 对于产生式$S \rightarrow aB$，有δ(S, a)={B}
  - 对于产生式$B \rightarrow aB$，有δ(B, a)={B}
  - 对于产生式$B \rightarrow bS$，有δ(B, b)={S}
  - 对于产生式$B \rightarrow a$，　有δ(B, a)={f}
- FA M的状态转换图：



定理：对每一个DFA M，都存在一个等价的右线性文法G和一个等价的左线性文法G′。

设DFA M为：$M=(\Sigma, Q, q_0, F, \delta)$

- 构造右线性文法G：$G=(V_T, V_N, S, \varphi)$

  $V_T=\Sigma$、$V_N=Q$、$S=q_0$

  φ的构造：对任何$a\in\Sigma$，及$A、B\in Q$，若存在$\delta(A,a)=B$，则：

  - 如果$B\notin F$，则有$A\to aB$
  - 如果$B\in F$，则有$A\to aB|a$

- 证明$L(M)=L(G)$

  首先证明被DFA M接受的语言可以由右线性文法G产生

  对任何$\omega\in L(M)$，设$\omega=a_1a_2\cdots a_n$，$a_i\in\Sigma$，存在状态序列：$q_0, q_1, \cdots, q_{n-1}, q$

  $q\in F$，有转换函数$\delta(q_0, a_1)=q_1$，$\delta(q_1, a_2)=q_2$，…，$\delta(q_{n-1}, a_n)=q$

  因此在文法G中有产生式：$q_0\to a_1q_1$，$q_1\to a_2q_2$，…，$q_{n-1}\to a_n$

  于是有推导序列：$q_0\Rightarrow a_1q_1\Rightarrow a_1a_2q_2\Rightarrow\ldots\Rightarrow a_1a_2\ldots a_{n-1}q_{n-1}\Rightarrow a_1a_2\ldots a_n$

  因此，$a_1a_2\ldots a_n$是文法G生成的一个句子，即$\omega\in L(G)$，因此$L(M)\subset L(G)$

再证明由文法G产生的语言，能够被DFA M所接受。

对任何$\omega \in L(G)$，设$\omega=a_1a_2\ldots a_n$，其中$a_i \in V_T$，必存在推导序列：

$q_0 \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \ldots \Rightarrow a_1a_2\ldots a_{n-1}q_{n-1} \Rightarrow a_1a_2\ldots a_n$

DFA M中有转换函数：$\delta(q_0, a_1)=q_1, \delta(q_1, a_2)=q_2, \ldots, \delta(q_{n-1}, a_n)=q$，并且$q \in F$

在DFA M中有一条从q_0出发、依次经过状态$q_1, q_2, \ldots, q_{n-1}$再到达终态$q$的道路，路径上有向边的标记依次为$a_1, a_2, \ldots, a_{n-1}, a_n$，这些标记依次连接起来恰好是$\omega$，所以$\omega$被DFA M所接受，即$\omega \in L(M)$，因此$L(G) \subset L(M)$。

- 若$q_0 \in F$，则$\omega=\varepsilon \in L(M)$，但$\varepsilon \notin L(G)$，即：$L(G)=L(M)-\{\varepsilon\}$

进一步改进文法G：增加一个新的非终结符号S′，及相应产生式：$S' \to S \mid \varepsilon$，并用S′代替S作为文法的开始符号。

改进后的文法G仍是右线性文法，并且满足：$L(M)=L(G)$。

推论：对任何一个有限自动机M，都存在一个等价的正规文法G，反之亦然。

推论：对任何一个右线性文法G，都存在一个等价的左线性文法G′，反之亦然。

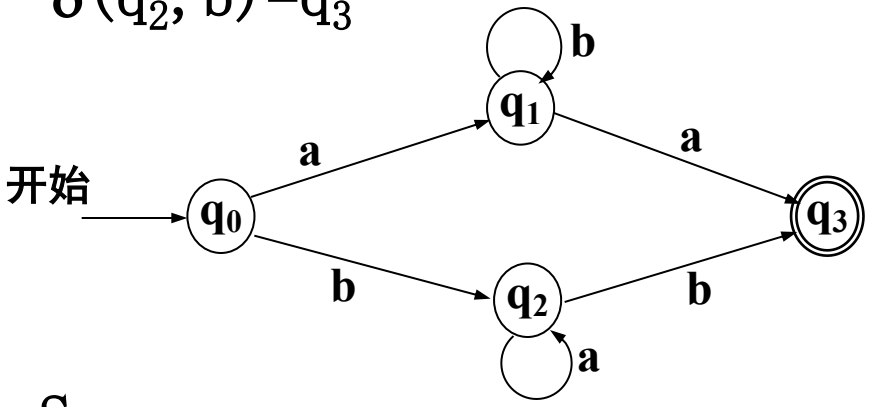
示例：设有DFA $M=(\{a,b\},\ \{q_0,q_1,q_2,q_3\},q_0,\{q_3\},\delta)$

其中转换函数δ如下：

$\delta(q_0,a)=q_1,\quad \delta(q_1,a)=q_3,\quad \delta(q_2,a)=q_2$

$\delta(q_0,b)=q_2,\quad \delta(q_1,b)=q_1,\quad \delta(q_2,b)=q_3$

试构造与之等价的右线性文法G。



- DFA M的状态转换图
- 构造右线性文法$G=(V_T,V_N,S,\varphi)$
- $V_T=\{a,b\}\quad V_N=\{q_0,q_1,q_2,q_3\}\quad S=q_0$
- 产生式集合φ

$\delta(q_0,a)=q_1$，∴ $q_0\to aq_1$

$\delta(q_0,b)=q_2$，∴ $q_0\to bq_2$

$\delta(q_1,a)=q_3$，$q_3\in F$，∴ $q_1\to a\mid aq_3$

$\delta(q_1,b)=q_1$，∴ $q_1\to bq_1$

$\delta(q_2,a)=q_2$，∴ $q_2\to aq_2$

$\delta(q_2,b)=q_3$，$q_3\in F$，∴ $q_2\to b\mid bq_3$

构造的文法G：

$G=(\{a,b\},\{q_0,q_1,q_2\},q_0,\varphi)$

φ：$q_0\to aq_1\mid bq_2$

$q_1\to a\mid bq_1$

$q_2\to aq_2\mid b$

2.4 正规表达式与有限自动机的等价性

- 用正规表达式可以精确地定义集合，

定义Pascal语言标识符的正规表达式：

letter(letter|digit)*

定义：字母表Σ上的正规表达式

(1) ε是正规表达式，它表示的语言是{ε}

(2) 如果a∈Σ，则a是正规表达式，它表示的语言是{a}

(3) 如果r和s都是正规表达式，分别表示语言L(r)和L(s)，则：

1) (r)|(s) 是正规表达式，表示的语言是L(r)∪L(s)

2) (r)(s) 是正规表达式，表示的语言是L(r)L(s)

3) (r)* 是正规表达式，表示的语言是(L(r))*

4) (r) 是正规表达式，表示的语言是L(r)

- 正规表达式表示的语言叫做正规集。

正规表达式的书写约定

- 一元闭包‘*’具有最高优先级，并且遵从左结合
- 连接运算的优先级次之，遵从左结合
- 并运算‘|’的优先级最低，遵从左结合

例：如果Σ={a，b}，则有：

正规表达式 a|b 表示集合 {a，b}

(a|b)(a|b) 表示：{aa，ab，ba，bb}

a* 表示：由0个或多个a组成的所有符号串的集合

a|a*b 表示：a和0个或多个a后跟一个b的所有符号串的集合

(a|b)* 表示：由a和b构成的所有符号串的集合

(a*|b*)*

- 如果两个正规表达式r和s表示同样的语言，即L(r)=L(s)，则称r和s等价，写作r=s。

  如：(a|b)=(b|a)

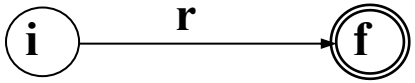
正规表达式遵从的代数定律

定律	说明
r|s=s|r	“并”运算是可交换的
r|(s|t)=(r|s)|t	“并”运算是可结合的
(rs)t=r(st)	连接运算是可结合的
r(s|t)=rs|rt (s|t)r=sr|tr	连接运算对并运算的分配
εr=r，rε=r	对连接运算而言，ε是单位元素
r*=(r|ε)*	*和ε之间的关系
r**=r*	*是等幂的
r*=r+|ε，　r+=rr*	+和*之间的关系

定理：对任何一个正规表达式r，都存在一个FA M，使L(r)=L(M)，反之亦然。

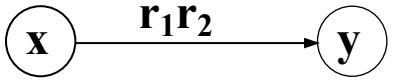
- 证1：设r是∑上的一个正规表达式，则存在一个具有ε-转移的NFA M接受L(r)。

  首先，为正规表达式r构造如下图所示的拓广转换图。

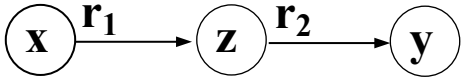


- 然后，按照下面的转换规则，对正规表达式r进行分裂、加入新的结点，直到每条边的标记都为基本符号为止。

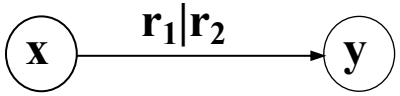
若$r=r_1r_2$，则将：



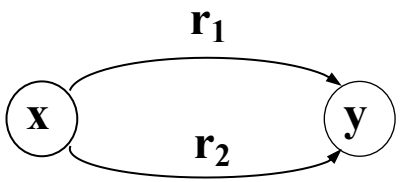
代之以：



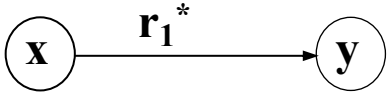
若$r=r_1|r_2$，则将：



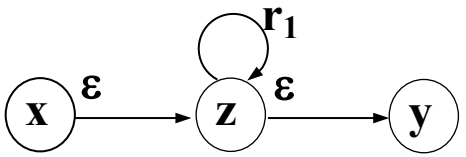
代之以：



若$r=r_1^*$，则将：

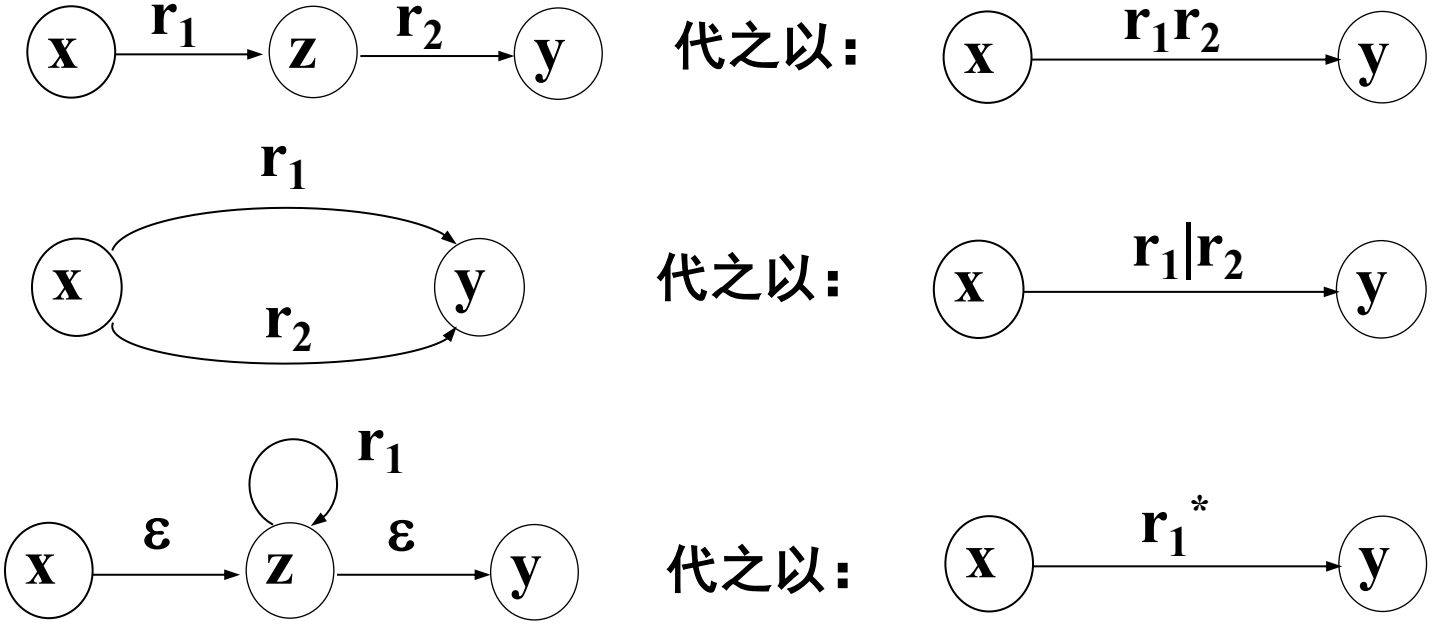


代之以：



证2：设有FA M，则存在一个正规表达式r，它表示的语言即该FA M所识别的语言。

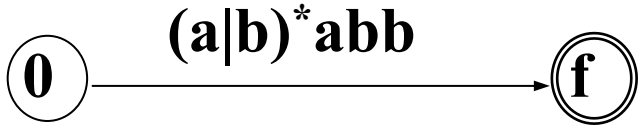
- 首先，在FA M的转换图中增加两个结点i和f，并且增加ε边，将i连接到M的所有初态结点，并将M的所有终态结点连接到f。形成一个新的与M等价的NFA N。
- 然后，反复利用下面的替换规则，逐步消去N中的中间结点，直到只剩下结点i和f为止。



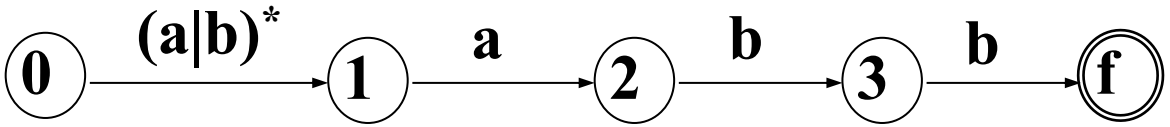
示例：为正规表达式(a|b)*abb，构造等价的NFA。

- 构造过程：

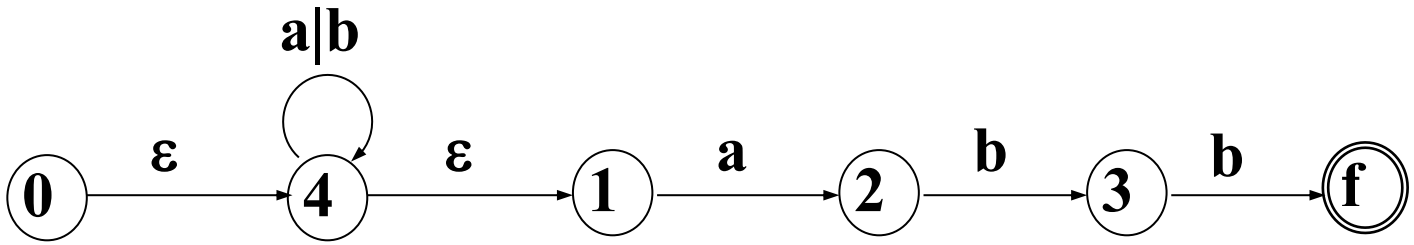
(a)



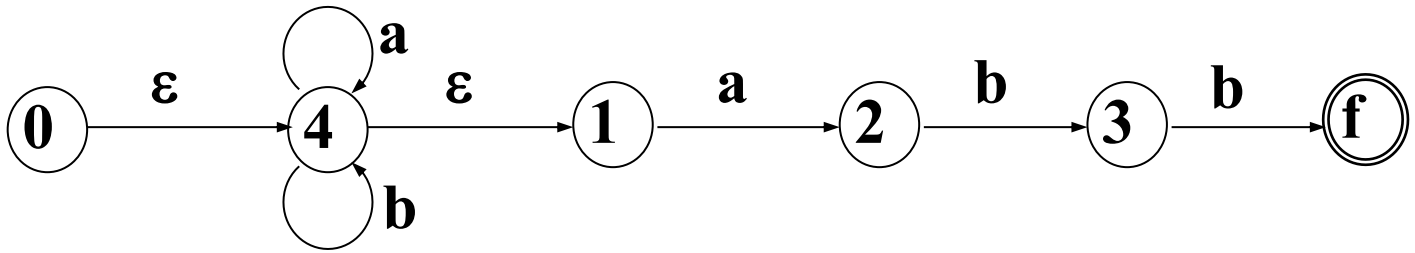
(b)



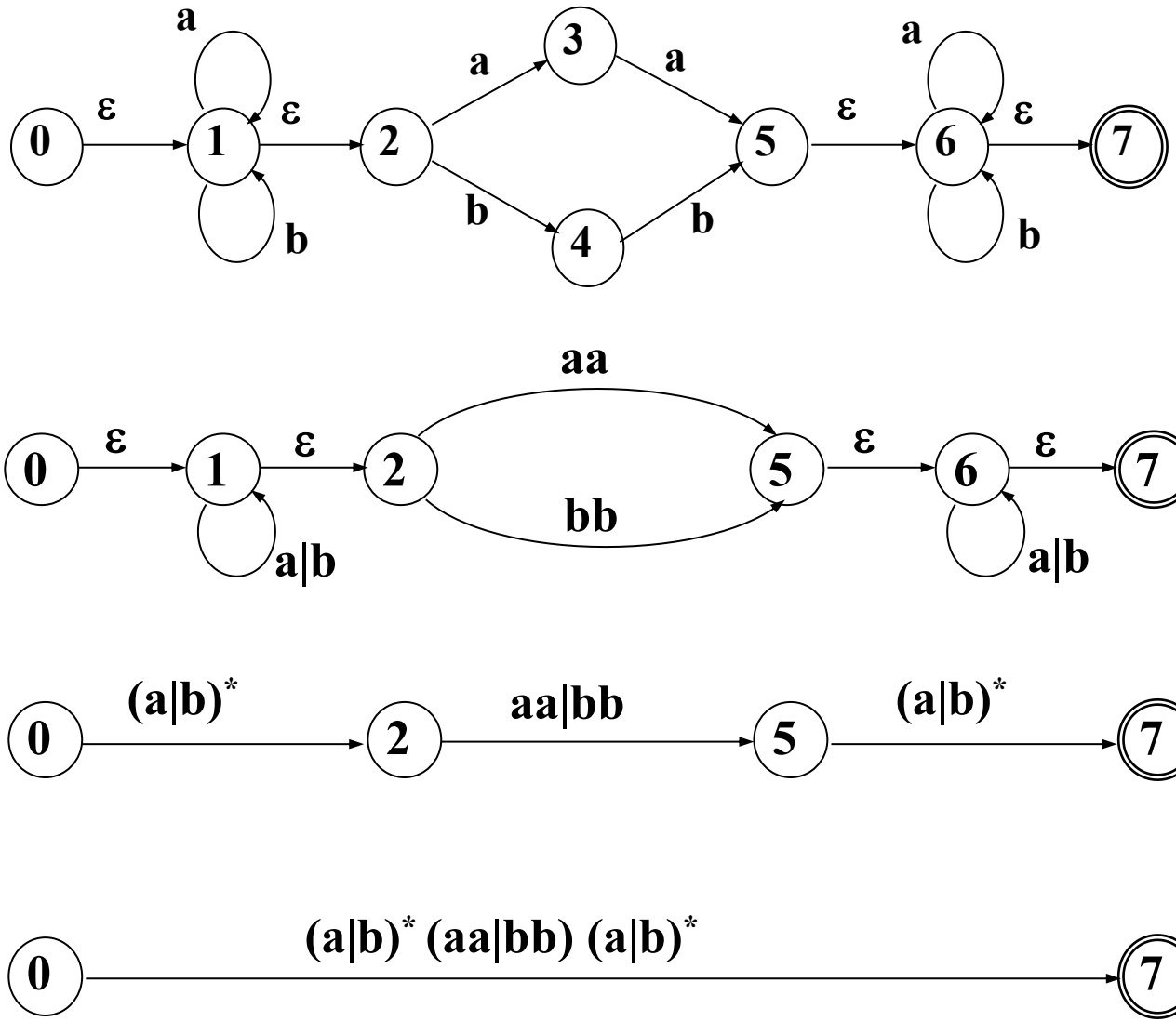
(c)



(d)



示例：构造与如下的NFA M等价的正规表达式r。



2.5 正规表达式与正规文法的等价性

- 正规表达式与正规文法具有同样的表达能力
- 对任何一个正规表达式都可以找到一个正规文法，使这个正规文法所产生的语言（即正规语言）恰好是该正规表达式所表示的语言（即正规集），反之亦然。
- 正规表达式和正规文法都可以用来描述程序设计语言中单词符号的结构
  - 用正规表达式描述，清晰而简洁；
  - 用正规文法描述，易于识别。

正规定义式

定义：令∑是字母表，正规定义式是如下形式的定义序列：

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

...

$d_n \rightarrow r_n$

其中d_i是不同的名字，r_i是$\sum \cup \{d_1, d_2, ..., d_{i-1}\}$上的正规表达式。

例：Pascal语言的无符号数可用如下的正规表达式来描述：

$digit^+(.digit^+|\varepsilon)(E(+|-|\varepsilon)digit^+|\varepsilon)$

正规定义式：

digit → 0|1|...|9

digits → digit digit*

optional_fraction → .digits|ε

optional_exponent → (E(+|-|ε)digits)| ε

num → digits optional_fraction optional_exponent

表示的缩写

- 引入正闭包运算符 ‘+’
  - $r^*=r^+|\varepsilon$、 $r^+=rr^*$
  - digits → $digit^+$
- 引入可选运算符 ‘？’
  - r?=r|ε
  - optional_fraction → (.digits)?
  - optional_exponent → (E(+|-)?digits)?
- 引入表示 ‘[...]’
  - 字符组[abc]，表示正规表达式a|b|c
  - digit → [0-9]
  - 标识符的正规表达式：[A-Za-z][A-Za-z0-9]*

正规表达式转换为等价的正规文法

- 例：Pascal语言标识符的正规表达式：

  letter(letter|digit)*

- 引入名字letter、digit、和id
- 正规定义式：

  letter → A|B|...|Z|a|b|...|z

  digit → 0|1|...|9

  id → letter(letter|digit)*

- 关键：如何把正规定义式转换为相应的正规文法
- 分析：
  - 为子表达式(letter|digit)* 取一个名字rid
  - 展开第三个正规定义

转换为正规文法

(letter|digit)* =ε|(letter|digit)+

=ε|(letter|digit) (letter|digit)*

=ε|letter(letter|digit)* | digit(letter|digit)*

=ε|(A|B|…|Z|a|b|…|z)(letter|digit)* | (0|1|…|9)(letter|digit)*

=ε|A(letter|digit)* | B(letter|digit)* | … | Z(letter|digit)*

|a(letter|digit)* | b(letter|digit)* | … | z(letter|digit)*

|0(letter|digit)* | 1(letter|digit)* | … | 9(letter|digit)*

- id和rid看成是文法的非终结符号，产生式：

  id → A *rid* |B *rid* |…|Z *rid* |a *rid* |b *rid* |…|z *rid*

  rid →ε |A *rid* |B *rid* |…|Z *rid* |a *rid* |b *rid* |…|z *rid*

  |0 *rid* |1 *rid* |…|9 *rid*

- 把letter和digit看作是终结符号，产生式：

  id → letter *rid*

  rid → ε | letter *rid* | digit *rid*

正规文法的产生式和正规定义式中的正规定义

- 两个不同的概念，具有不同的含义。
- **产生式**：左部是一个非终结符号，右部是一个符合特定形式的文法符号串α，α中的非终结符号可以与该产生式左部的非终结符号相同，即允许非终结符号的递归出现。
- **正规定义**：左部是一个名字，右部是一个正规表达式，表达式中出现的名字是有限制的，即只能是此定义之前已经定义过的名字。

小　结

- 字母表和符号串
  - 前缀、后缀、子串、子序列、真前缀、真后缀、真子串
  - 连接、幂
- 语言
  - 语言的运算：并、连接、闭包、正闭包
- 文法
  - 形式定义$G=(V_T, V_N, S, \varphi)$
  - 文法的分类
  - 上下文无关文法（$A\rightarrow\beta$）
  - 正规文法
    - 右线性文法（$A\rightarrow aB$　　$A\rightarrow a$）
    - 左线性文法（$A\rightarrow Ba$　　$A\rightarrow a$）

- 推导
  - 一步推导、直接推导、推导的长度
  - 最左推导、最右推导、规范推导
  - 句型、左句型、右句型、规范句型
  - 句子
- 短语
  - 短语、直接短语、句柄

$S \overset{*}{\Rightarrow} \alpha A \delta$，并且 $A \overset{+}{\Rightarrow} \beta$

$S \overset{*}{\Rightarrow} \alpha A \delta$，并且 $A \Rightarrow \beta$

- 分析树及二义性
  - 分析树、子树
  - 子树与短语之间的关系
    - 子树——短语
    - 只有父子两代的子树——直接短语
    - 最左边的只有父子两代的子树——句柄
  - 句子二义性、文法的二义性、语言的二义性

- 文法的变换
  - 文法的改写
  - 左递归的消除
    - 简单的直接左递归的消除
    - 间接左递归的消除算法
    - 改写文法为无ε-产生式的文法
  - 提取左公因子
- 有限自动机
  - 形式定义$M=(\Sigma, Q, q_0, F, \delta)$
  - DFA： $\delta: Q\times\Sigma\rightarrow Q$
  - NFA： $\delta: Q\times\Sigma\rightarrow 2^Q$
  - 具有ε-转移的NFA： $\delta: Q\times(\Sigma\cup\varepsilon)\rightarrow 2^Q$
- 自动机之间的等价性
  - NFA确定化
  - 具有ε-转移的NFA的确定化

- DFA的最小化
  - 状态等价、状态可区分
  - 将DFA的状态集合划分为等价状态子集
- 正规文法与有限自动机之间的等价性
  - 为右线性文法构造DFA
  - 为DFA构造右线性文法
- 正规表达式与有限自动机之间的等价性
  - 为NFA构造正规表达式
  - 为正规表达式构造NFA
- 正规表达式与正规文法之间的等价性
  - 同等表达能力
  - 利用正规定义式，将正规表达式转换为正规文法