同济大学
TONGJI UNIVERSITY

编译原理

第三章 词法分析

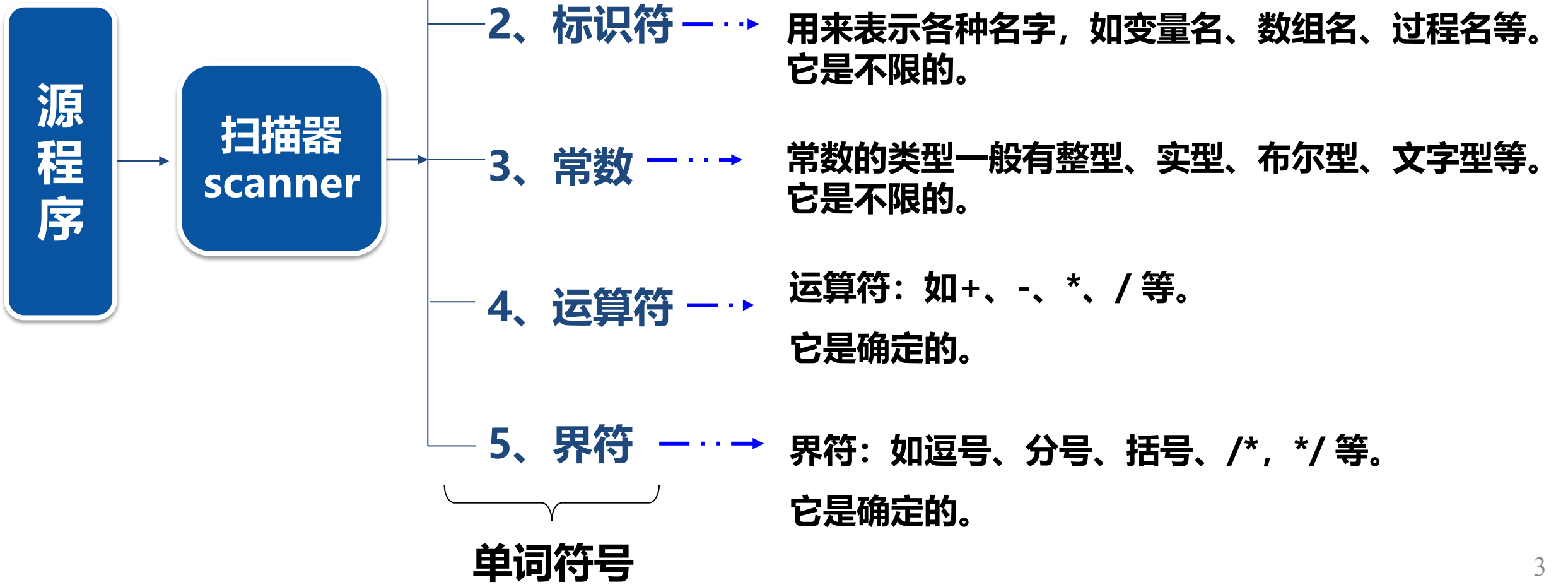
丁志军

dingzj@tongji.edu.cn

内容线索

1. 对于词法分析器的要求
2. 词法分析器的设计
3. 正规表达式与有限自动机
4. 词法分析器的自动生成

词法分析器的功能


源程序
扫描器 scanner
1、关键字
由程序语言定义的具有固定意义的标识符。也可称为保留字或基本字。例如：C中的int，while，if等。它是确定的。
2、标识符
用来表示各种名字，如变量名、数组名、过程名等。它是不限的。
3、常数
常数的类型一般有整型、实型、布尔型、文字型等。它是不限的。
4、运算符
运算符：如+、-、*、/ 等。
它是确定的。
5、界符
界符：如逗号、分号、括号、/*，*/ 等。
它是确定的。
单词符号

单词符号表示形式

- 词法分析器输出的单词符号常表示成二元组：
- （单词种别，单词符号的属性值）
  - 单词种别是语法分析需要的信息
  - 单词符号属性值则是编译其它阶段需要的信息,简称单词值。

示例

语句const i=25,yes=1，其中，单词25和1的类别都是常数，其值分别为25和1；

分类方法

- 单词种别:通常用整数编码。
- 一个语言的单词符号如何分类，分成几类，怎样编码取决于处理上的方便。
  - 标识符一般统归为一种。
  - 常数则宜按类型（整、实、布尔等）分种。
  - 关键字可视其全体为一种，也可以一字一种。采用一字一种的分法实际处理起来较为方便。
  - 运算符可采用一符一种的分法，但也可以把具有一定共性的运算符视为一类。
  - 界符一般用一符一种的分法。

单词符号的属性

- 单词符号的属性是指单词符号的特征或特性。属性值则是反映特性或特征的值。
  - 标识符的属性值是存放它符号表项的指针或内部字符串；
  - 常数的属性值是存放它的常数表项的指针或二进制形式；
  - 关键字、运算符和界符是一符一种，不需给出其自身的值。

词法分析示例

```
void main(int b,int c)
{
    int a;
    a=b+(c-2);
}#
```

词法分析后返回 (如右图)：

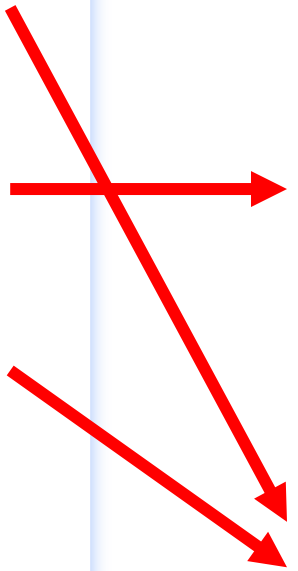
单词值	单词类型
1: void	VOID
1: main	MAIN
1: (	LPAREN
1: int	INT
1: b	ID
1: ,	COMMA
1: int	INT
1: c	ID
1:)	RPAREN
2: {	LBBRACKET
3: int	INT
3: a	ID

单词值	单词类型
3: ;	SEMI
4: a	ID
4: =	ASSIGN
4: b	ID
4: +	PLUS
4: (	LPAREN
4: c	ID
4: -	SUB
4: 2	NUM
4:)	RPAREN
4: ;	SEMI
5: }	RBBRACKET
5: #	ENDFILE

示例

代码段 while (i>=j) i--; 词法分析结果

<while , - >
< (, - >
< id , ptr-i>
< >= , - >
< id , ptr-j>
<) , - >
< id , ptr-i>
< - - , - >
< ; , - >



符号表				
No	ID	Addr	type	••• •••
224	j	AF80	INT	
227	i	DF88	INT	

词法分析程序的实现方式

- **完全独立方式：**词法分析程序作为单独一遍来实现。词法分析程序读入整个源程序，它的输出作为语法分析程序的输入。
  - 编译程序结构简洁、清晰和条理化

- **相对独立方式：**把词法分析程序作为语法分析程序的一个独立子程序。语法分析程序需要新符号时调用这个子程序。
  - 优点：避免了中间文件生成，可以提高效率。

内容线索

√. 对于词法分析器的要求

2. 词法分析器的设计

3. 正规表达式与有限自动机

4. 词法分析器的自动生成

词法分析器的结构

预处理工作包括对空白符、跳格符、回车符和换行符等编辑性字符的处理，及删除注解等。

源程序

预处理子程序

输入缓冲区

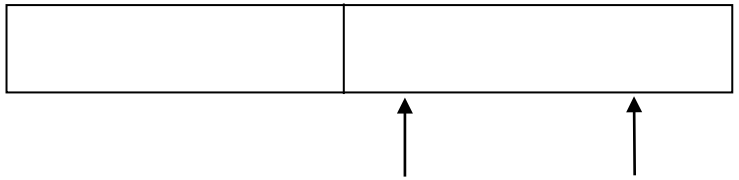
输入源程序文本。输入串一般放在一个缓冲区中，这个缓冲区称输入缓冲区。

扫描器

扫描缓冲区

单词符号

设定两个指示器将缓冲区一分为二



起点指示器　　搜索指示器

单词符号的识别：超前搜索

■ 关键字识别

示例 在标准FORTRAN中四个合法句子：

1、DO99K = 1,10

2、IF(5.EQ.M)I = 10

3、DO99K = 1.10

4、IF(5) = 55

其中的DO、IF为关键字

其中的DO、IF为标识符的一部分

单词符号的识别：超前搜索

- **标识符的识别**
  - 多数语言的标识符是字母开头的“字母/数字”串，而且在程序中标识符的出现后都跟着算符或界符。因此，不难识别。
- **常数的识别**
  - 对于某些语言的常数的识别也需要使用超前搜索。
  - FORTRAN中，5.E08和5. EQ.M都是合法的
- **算符和界符的识别**
  - 对于诸如C++语言中的“+ +”、“- -”，这种复合成的算符，需要超前搜索。

状态转换图

- 大多数程序设计语言中单词符号的**词法规则**可以用**正规文法**描述。

  如：

  <标识符>→ 字母|<标识符>字母|<标识符>数字

  <整数>→数字|<整数>数字

  <运算符>→+| - |×|÷…

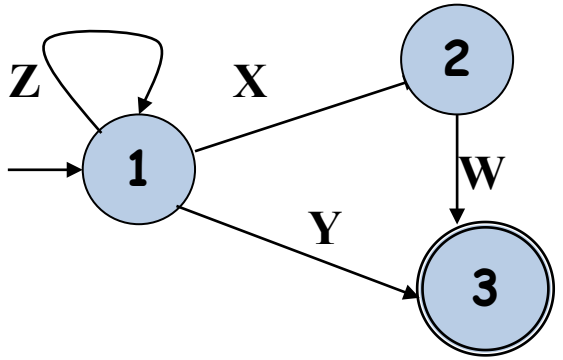
  <界符>→; |, |(|)|…、

- 利用这些规则识别单词符号的过程可用一张称为**状态转换图**的有限方向图来表示，而状态转换图识别单词符号的过程又可以方便地用程序实现。

状态转换图定义

- **转换图：是一个有限方向图。**
  - **结点代表状态，用圆圈表示。**
    - **初态：一张转换图的启动条件，通常有一个,用圆圈表示。**
    - **终态：一张转换图的结束条件，至少有一个，用双圈表示。**
  - **状态之间用方向弧连接。弧上的标记（字符）代表在出射结点状态下可能出现的输入字符或字符类。**
- **状态转换图中只包含有限个状态（结点）**

在状态1下，若输入字符为X，则读进X并转换到状态2；若输入字符为Y则读进Y并转换到状态3，输入字符Z，状态仍为1。

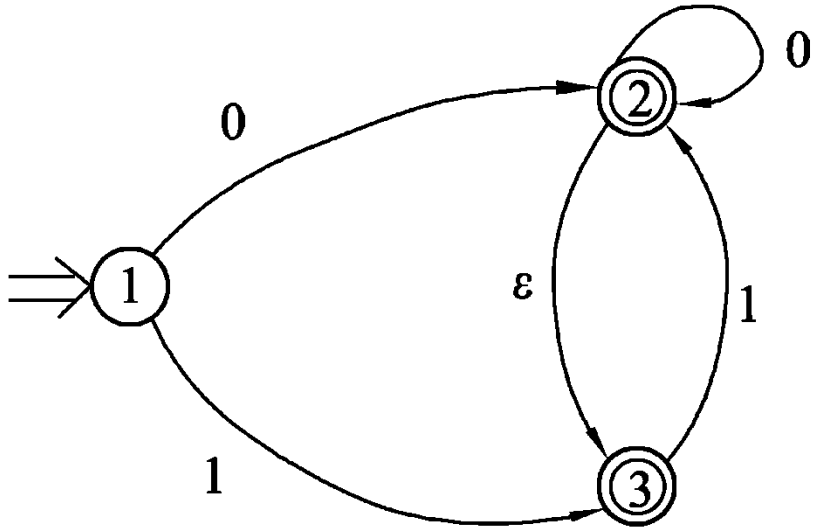


状态转换图的作用

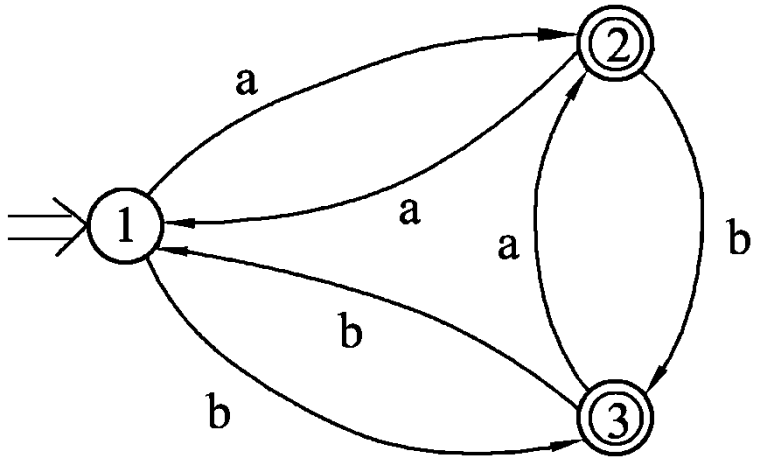
- 一个状态转换图可用于接受（或识别）一定的符号串。
- 路: 在状态转换图中从初始状态到某一终止状态的弧上的标记序列。
- 对于某一符号串β，在状态转换图中，若存在一条路产生β，则称状态转换图接受（或识别）该符号串β，否则称符号串β不能被接受。

状态转换图所能识别的语言

- 能被状态转换图TG接受的符号串的集合记为L(TG)，称它为**状态转换图所能识别的语言**。



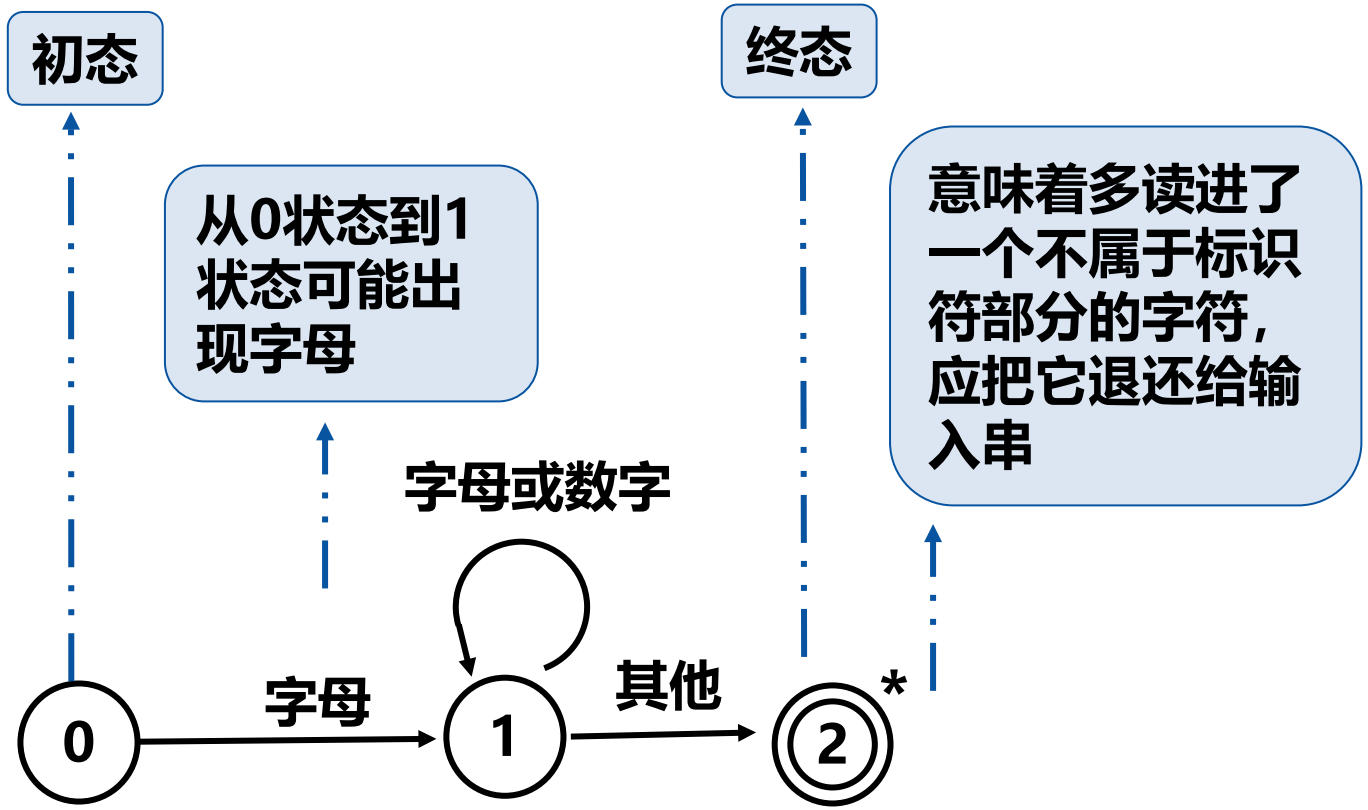
L(TG)={ 0，1，00，01，
11，001，010，...}



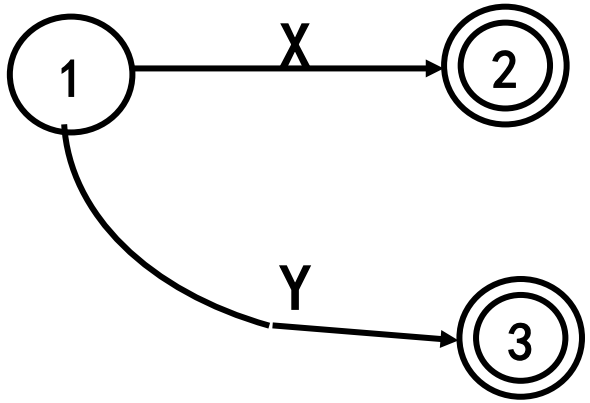
L(TG)={ a，b，ab，ba，
aaa，bbb，aab，bba，...}

状态转换图示例

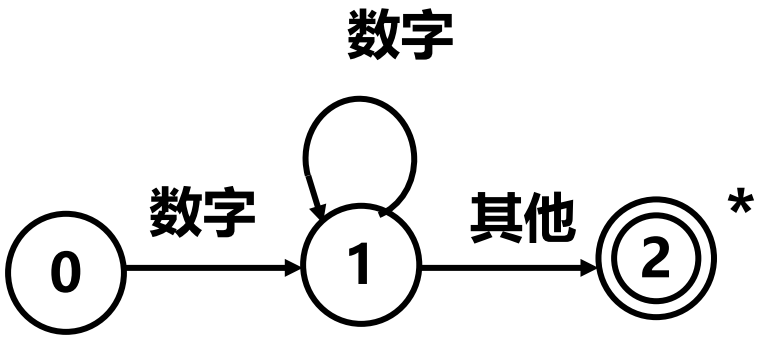
- 大多数程序语言的单词符号都可以用状态转换图予以识别。



（b）识别标识符的转换图



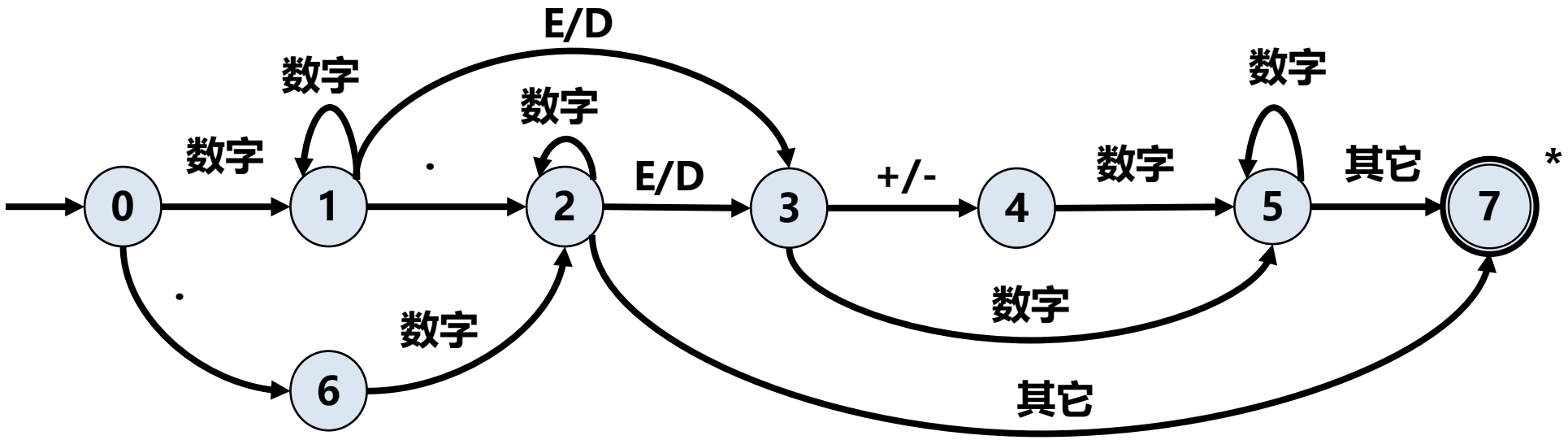
(a)单个符号的转换图示例



（c）识别整数的转换图

a .b E (或D)±d
(a,b,d 为整数常数)

- a.
- .b
- a.b
- a.E±d
- .b E±d
- a.b±E d
- aE±d



(d) 识别FORTRAN实型常数的转换图

状态转换图识别单词符号的过程

- Step1. 从初态开始；
- Step2. 从输入串中读一个字符；
- Step3. 判明读入字符与从当前状态出发的哪条弧上 的标记相匹配，便转到相应匹配的那条弧所指向的状态；
- Step4. 重复Step3，均不匹配时便告失败；到达终态时便识别出一个单词符号。

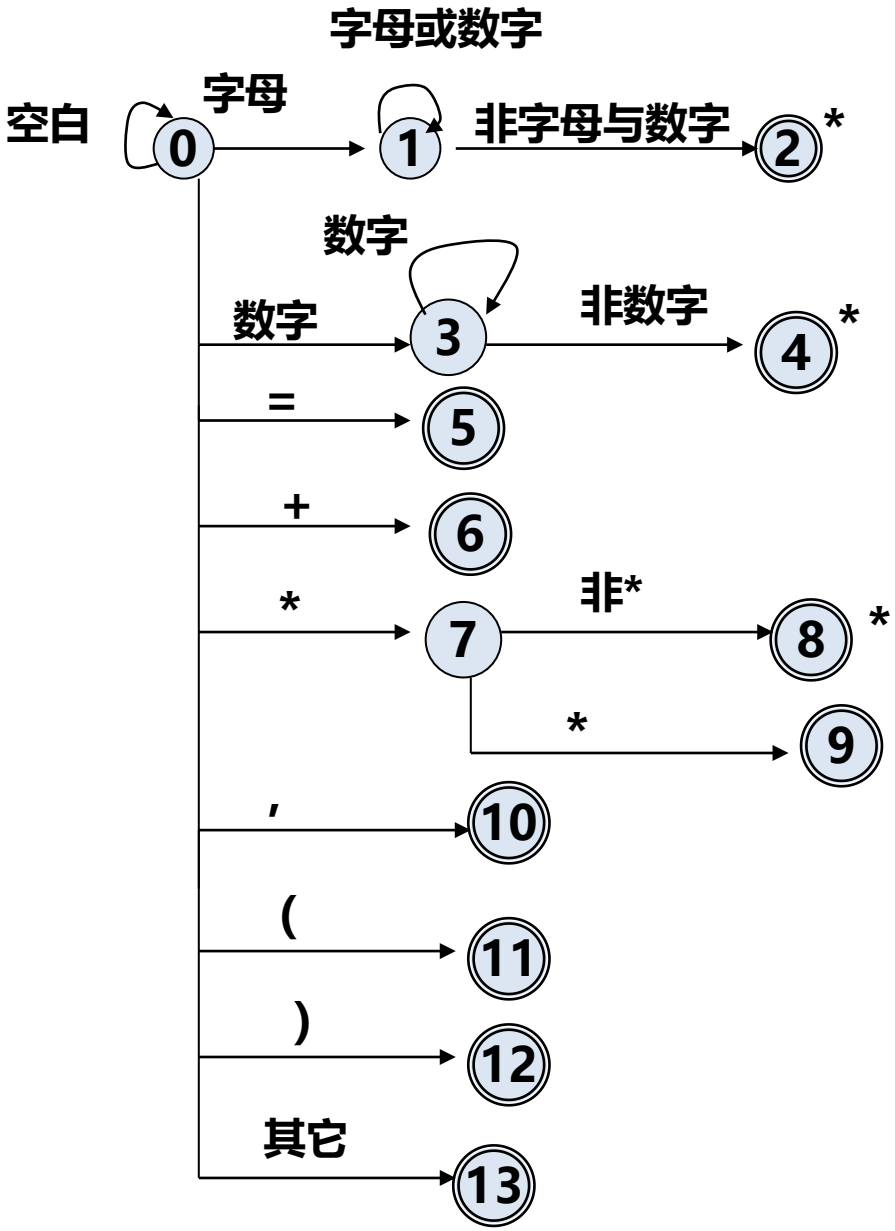
示例

- 设一小语言所有单词符号及其内部表示形式

单词符号	种别编码	助忆符	内码值
DIM	1	$DIM	-
IF	2	$IF	-
DO	3	$DO	-
STOP	4	$STOP	-
END	5	$END	-
标识符	6	$ID	内部字符串
整常数	7	$INT	标准二进制形式
=	8	$ASSIGN	-
+	9	$PLUS	-
*	10	$STAR	-
**	11	$POWER	-
.	12	$COMMA	-
(	13	$LPAR	-
)	14	$RPAR	-

■ 能识别小语言所有单词的状态转换图

- ■ 约定（限制）：
  - ➢ 关键字为保留字;
  - ➢ 保留字作为标识符处理,并使用保留字表识别;
  - ➢ 关键字、标识符、常数间若无运算符或界限符则加一空格



状态转换图实现中的变量和过程

ch：字符变量

功能：存放当前读入字符

strToken：字符数组

功能：存放单词的字符串

GetChar：取字符过程

功能：取下一字符到ch；

搜索指针+1

GetBC：滤除空字符过程

功能：判ch =空? 若是,则调用GetChar

Concat：子程序过程

功能：把ch中的字符拼入strToken

IsLetter, IsDigit：布尔函数

功能： ch中为字母、数字时返回.T.

Reserve：整型函数

功能：按strToken中字符串查保留字表；

查到返回保留字编码;否则返回0

Retract：子程序过程

功能：搜索指针回退一字符

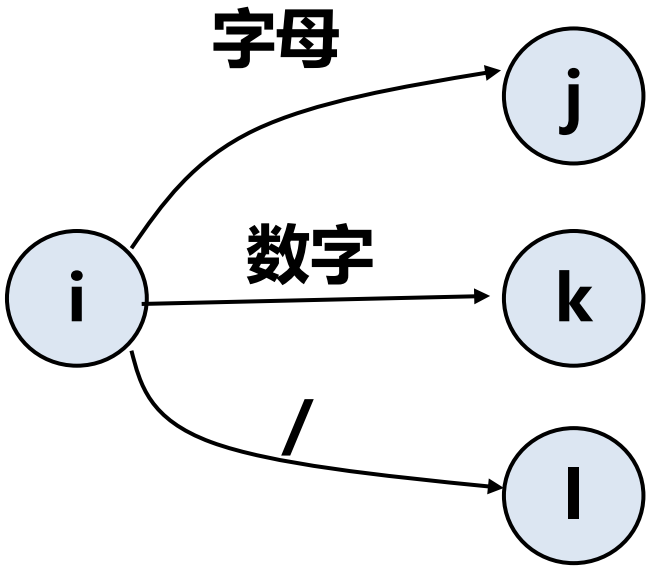
InsertId：函数

功能：将标识符插入符号表，返回符号表指针

InsertConst函数

功能： 将常数插入常数表，返回常数表指针

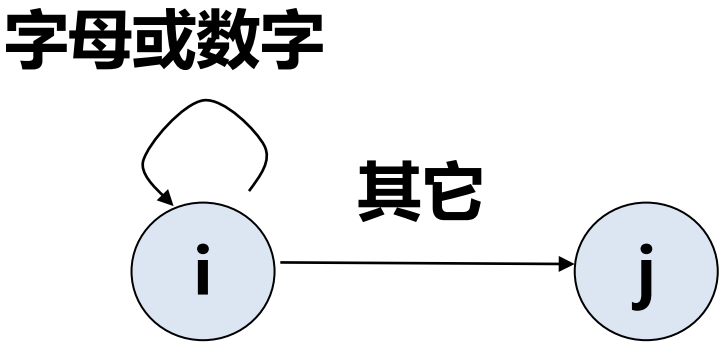
程序段



- 不含回路的分叉结点对应的程序段可表示为

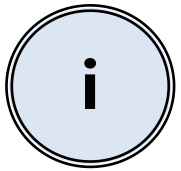
```
GetChar();
 if (IsLetter()) {…状态j的对应程序段…}
 else if (IsDigit()){…状态k的对应程序段…}
 else if(ch= ‘/’ ) {…状态l的对应程序段…}
 else{…错误处理…}
```

程序段



- 含回路的状态结点对应的程序段可表示为

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



- 终态结点对应一条语句

```
return(code,value);
```

扫描器总控程序

```
int code,value;
strToken= “” ;
GetChar();GetBC();
If  (IsLetter())
   { while(IsLetter() or IsDigit())
          {Concat();GetChar();}
     Retract();
     code=Reserve();
    if(code==0)
      { value=InsertId(strToken);
        return($ID,value);}
     else return(code,-);}
else if(IsDigit())
     { while(IsDigit()) {Concat(); GetChar();}
      Retract();
      value=InsertConst(strToken);
      return($INT,value);}
```

```
else if (ch==  ‘=  ‘) return($ASSIGN,-);
else if (ch==  ‘+  ‘) return($PLUS,-);
else if(ch== “*” )
       { Getchar();
         if(ch==  ‘*’  )
             return($POWER,-);
         Retract();return($STAR,-);}
else if(ch==  ‘:’  ) return($SEMICOLON,-);
else if(ch==  ‘(’  ) return($LPAR,-);
else if(ch==  ‘)’  ) return($RPAR,-);
else if(ch==  ‘{’  ) return($LBRACE,-);
else if(ch==  ‘}’  ) return($RBRACE,-);
else ProcError();
```

内容线索

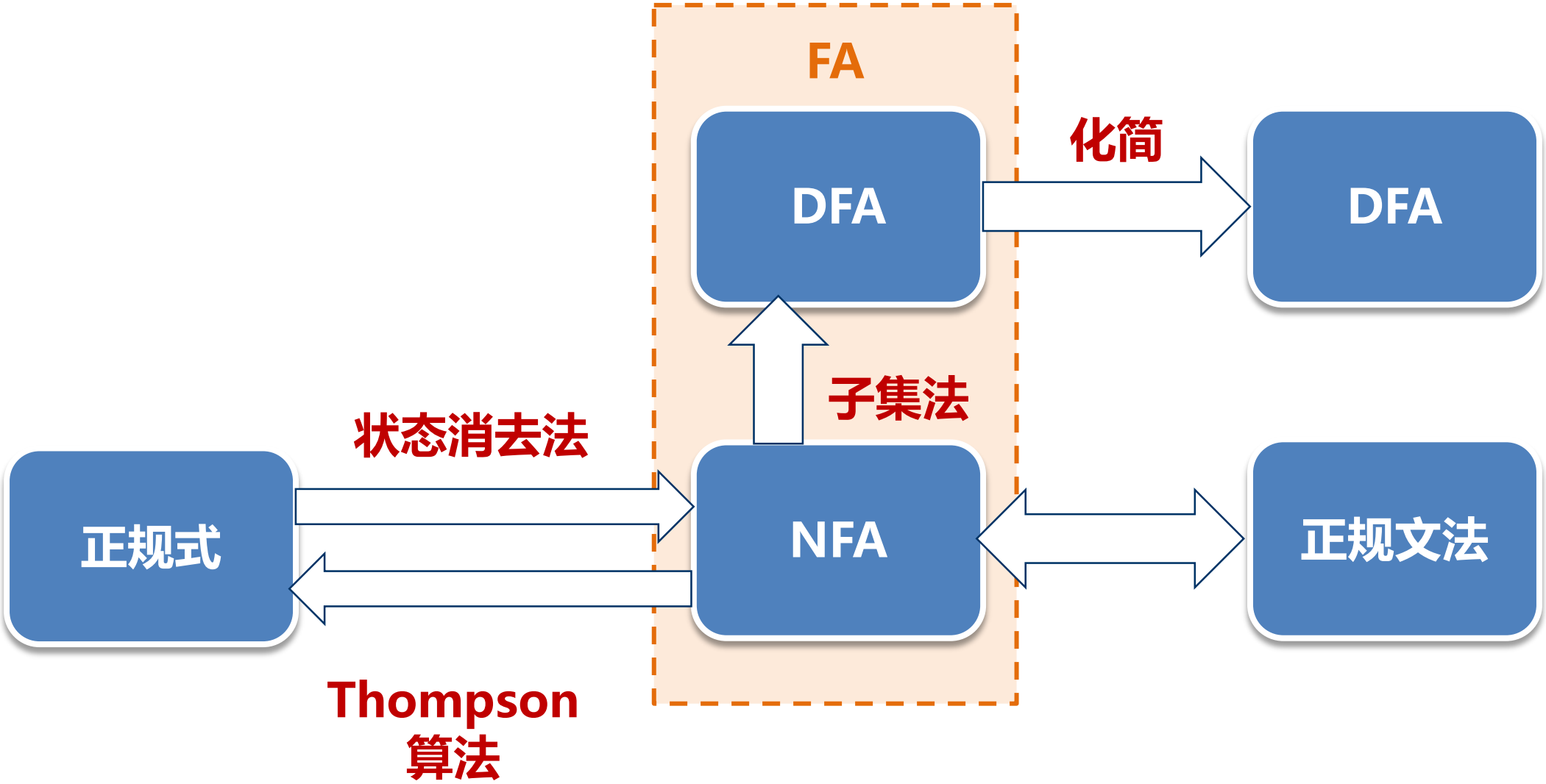
√. 对于词法分析器的要求

√. 词法分析器的设计

3. 正规表达式与有限自动机

4. 词法分析器的自动生成

正规表达式与有限自动机



正规式与正规集

- 字母表Σ上的**正规式**和**正规集**递归定义如下：

  (1) ε和ϕ都是Σ上的正规式，它们所表示的正规集分别为{ε}和 ϕ 。
  其中：ε为空字符串， ϕ为空集；

  (2) 任意元素a∈ Σ ，a是Σ上的一个正规式，它所表示的正规集是{a}；

  (3) 假定U和V都是Σ上的正规式，它们所表示的正规集 记为L(U)和L(V)，
  那么，（U|V），（U•V）和(U)*都是正规式，
  他们所表示的正规集分别记为L(U)∪L(V)，L(U)L(V)和(L(U))*。

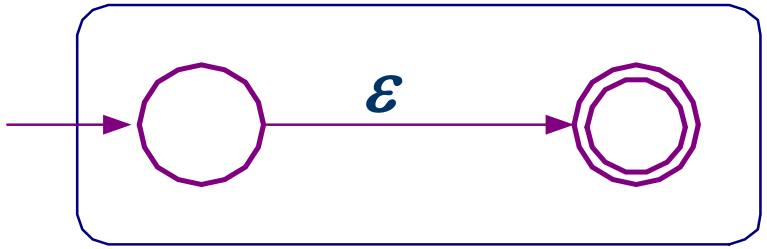
  (4) 仅由有限次使用上述三步而得到的表达式才是Σ上的正规式
  它们所表示的字集才是Σ上的正规集。

从正规式构造等价的NFA

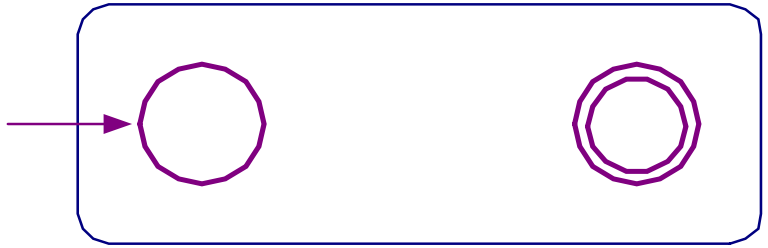
Thompson算法

- 基础

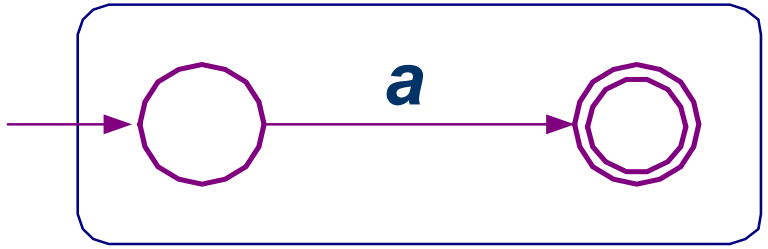
1 对于 ε，构造为



2 对于ϕ，构造为



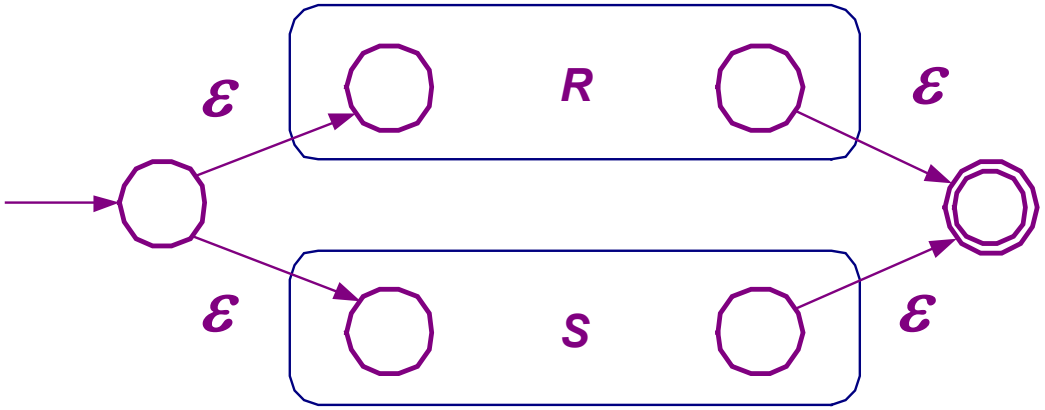
3 对于a，构造为



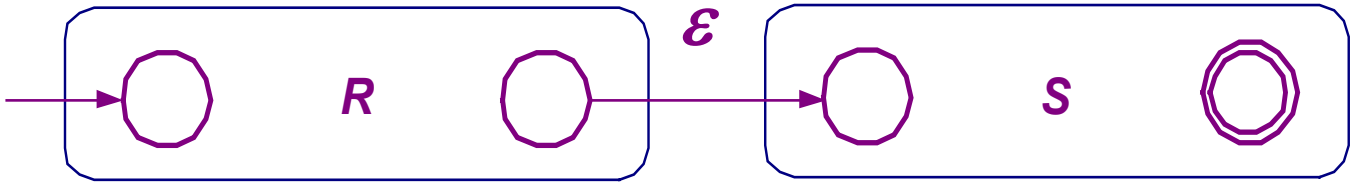
从正规式构造等价的NFA

- 归纳

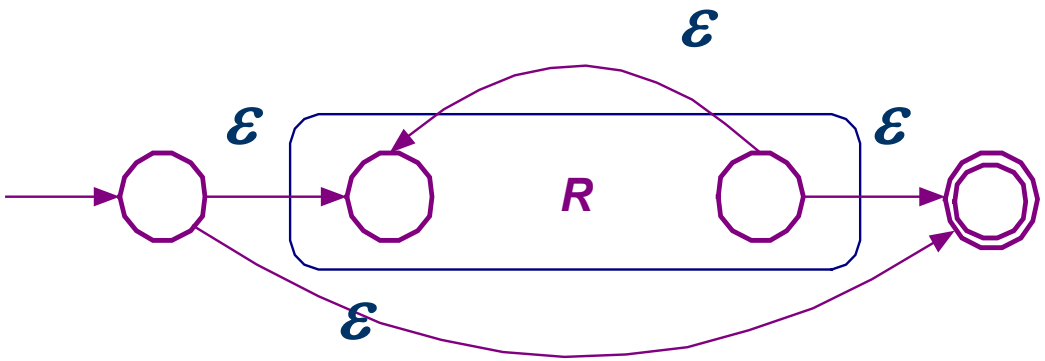
1 对于 R/S，构造为



2 对于 RS，构造为



3 对于 R*，构造为

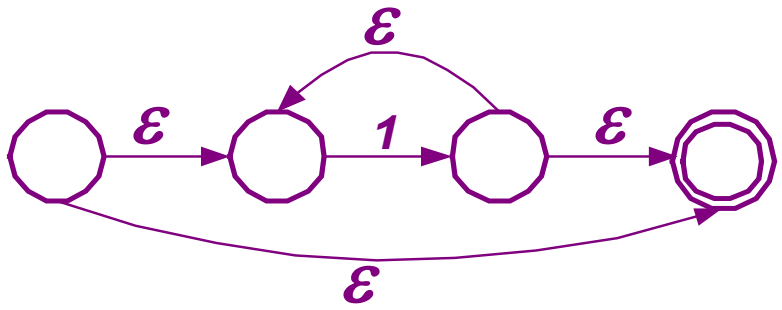


从正规式构造等价的NFA

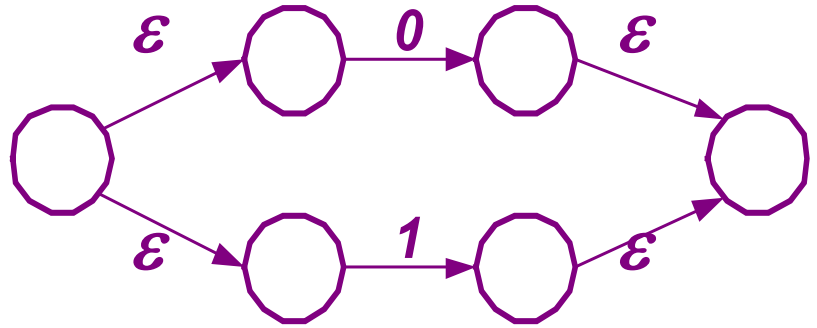
示例

设正则表达式 1*0(0|1)*, 构造等价的NFA.

*1**



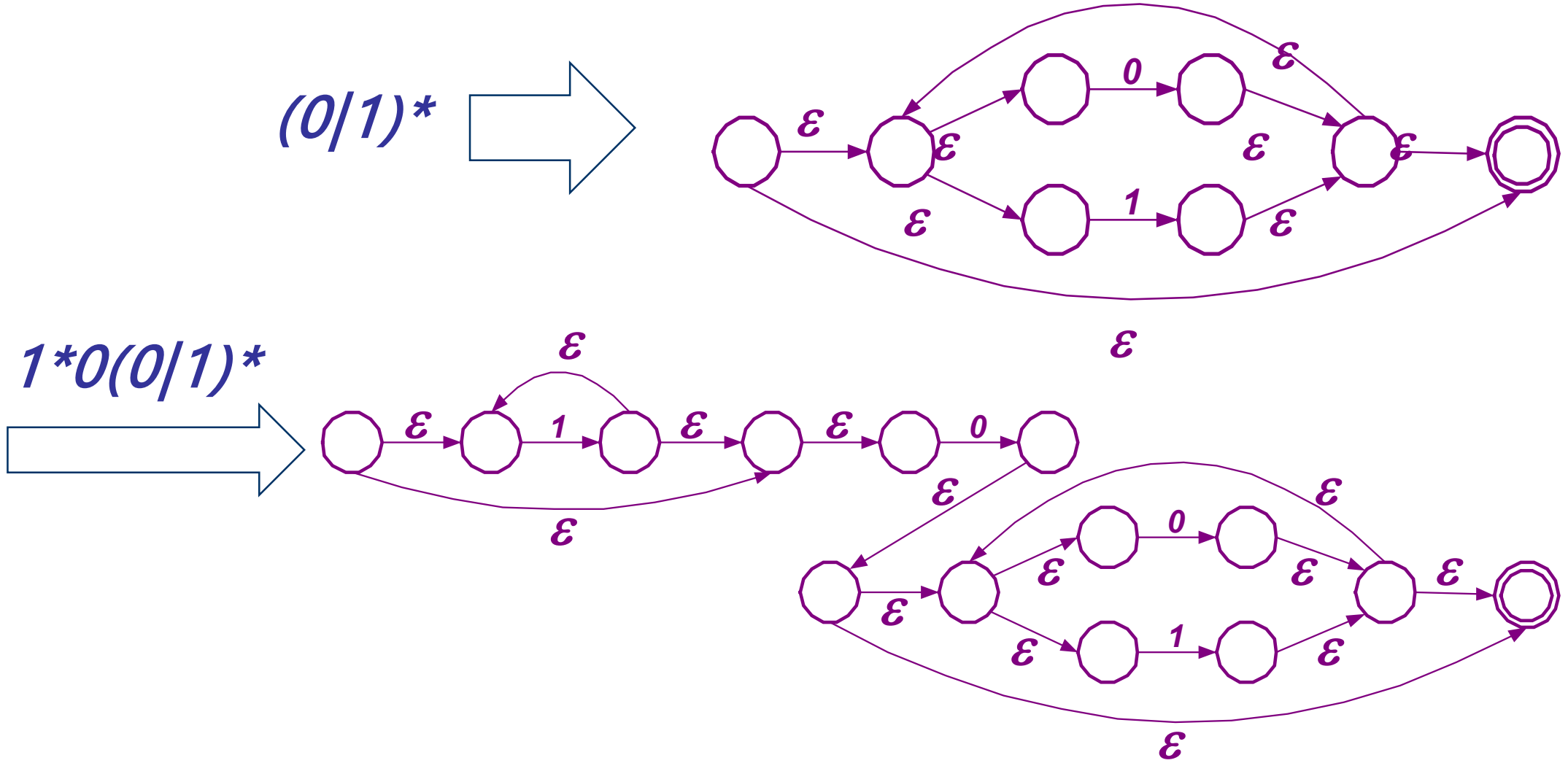
0|1



从正规式构造等价的NFA

示例

设正则表达式 1*0(0|1)*, 构造等价的NFA.



确定有限自动机（DFA）

- DFA是一个五元组 $M=(S, \Sigma, \delta, s_0, F)$
  - S: 有限的状态集合，每个元素称为一个**状态**；
  - Σ : 有限的输入字母表，每个元素称为一个**输入字符**；
  - δ: 转换函数(状态转移集合): $S \times \Sigma \rightarrow S$ ；
  - s_0: 初始状态， $s_0 \in S$ ；
  - F: 终止状态集，$F \subseteq S$ ；

非确定的有限自动机

- 一个非确定的有限自动机（NFA）M 是一个五元组:

  $M = (S, \Sigma, \delta, S_0, F)$，其中:

  (1) S和 Σ的定义同前;

  (2) $\delta: S \times \Sigma \cup \{\varepsilon\} \rightarrow 2^S$（状态子集）；

  对于某个状态$s \in S$ 和一个输入字母a：$\delta(s, a)=S' \subseteq S$

  (3) $S_0 \subseteq S$为非空初态集;

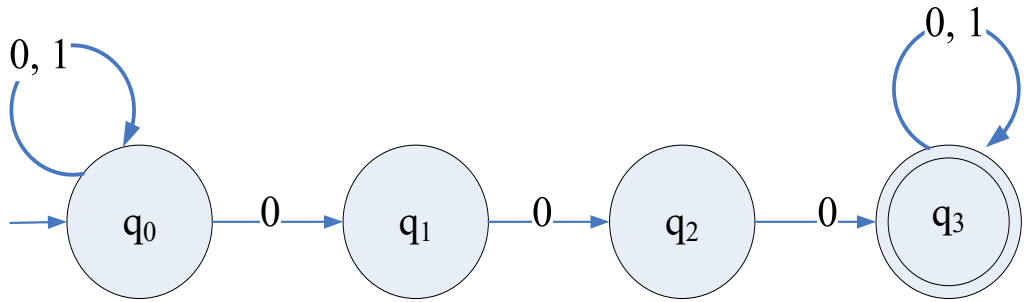
  (4) $F \subseteq S$为终态集

NFA和DFA的比较

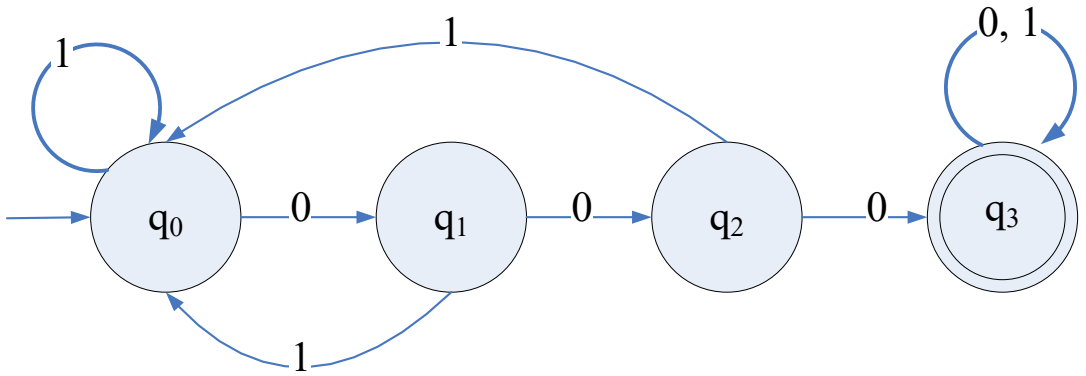
示例

构造NFA，可识别{0,1}上的语言

$$L=\{x000y \mid x,y\in\{0.1\}^*\}。$$



比较对应的DFA



NFA和DFA的比较

- 初态
  - DFA有且仅有一个初态
  - NFA可以有多个初态
- 输入字母
  - DFA的每一个状态对于字母表中的每一个符号都有一个转移函数。
  - 在NFA中，一个状态对于字母表中的每一个符号可能不存在转移函数或者存在空转换。
- 转移状态
  - DFA中的下一状态是确定的，即唯一的
  - NFA中的下一状态是不确定的，可能存在多个转移状态。

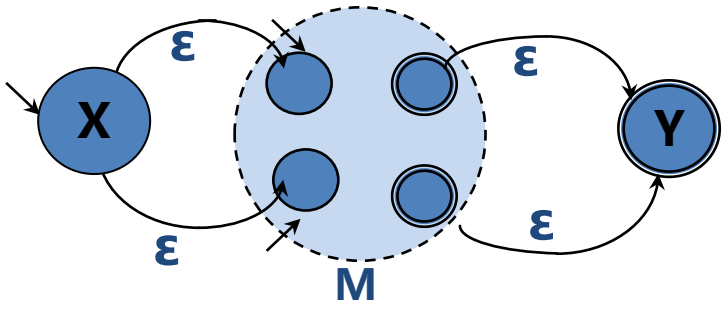
NFA和DFA的等价性

- DFA是NFA的特例，所以NFA必然能接收DFA能接收的语言。
- 一个NFA所能接收的语言能被另一个DFA所接收?
- 设一个NFA接受语言L，那么存在一个DFA接受L。
  - 证明策略：对于任意一个NFA，构造一个接收它所能接收语言的DFA，这个DFA的状态对应了NFA的状态集合。

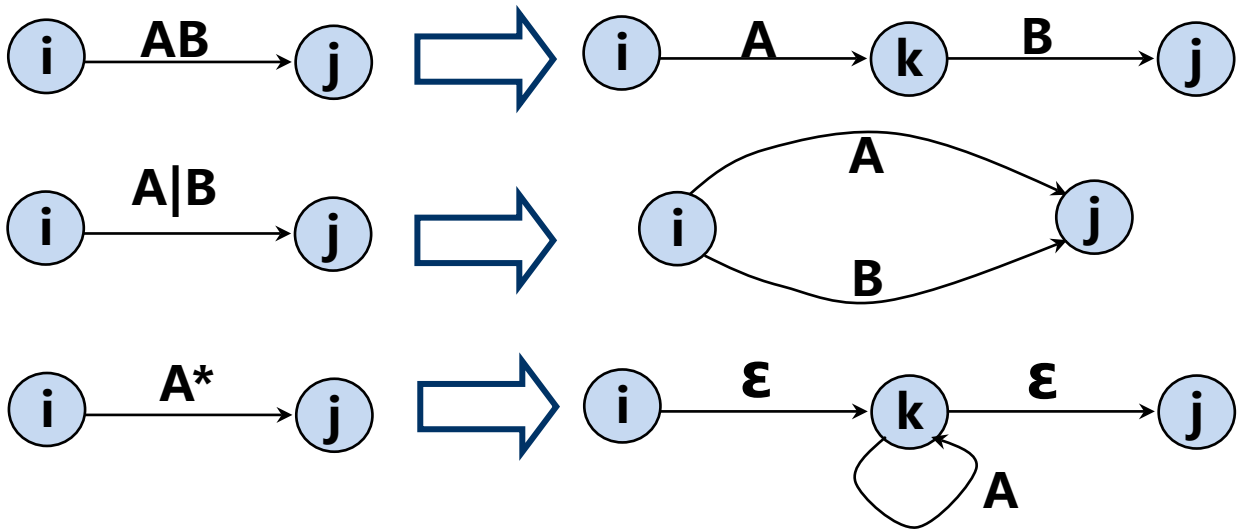
NFA和DFA等价性证明

（1）对NFA M的状态转换图进行改造，得到M′

① 引进新的初始结点X和终态结点Y，X,Y ∉S



② 按以下规则扩展结点、加边



NFA和DFA等价性证明

(2) 将M′进一步变换为DFA

- 设 I 是NFA M状态集的子集， I 的ε-闭包 ε- CLOSURE(I) 为:
  1) 若q∈I，则q∈ε- CLOSURE(I)；
  2) 若q∈I，则从 q出发经过任意条ε弧可到达的任何状态 q′ ∈ ε- CLOSURE(I)。

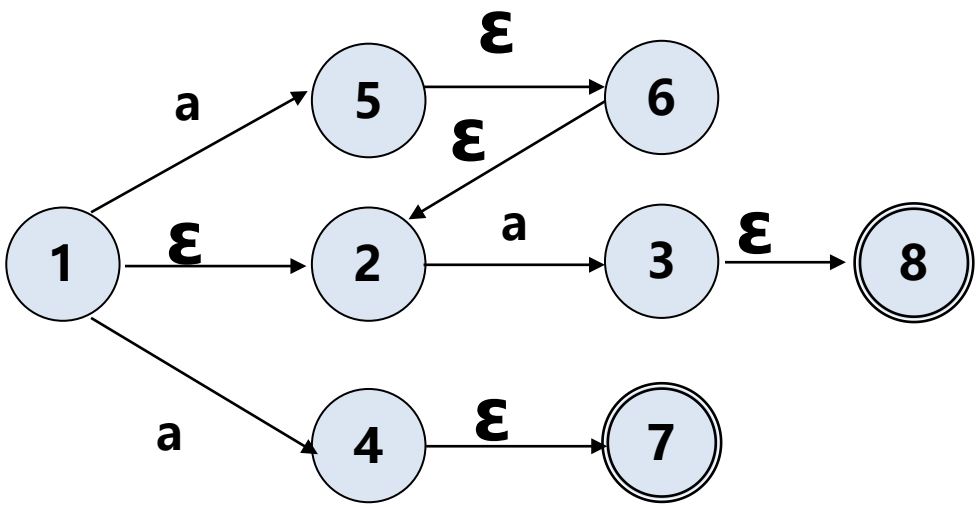
- 设 I 是NFA M状态集的子集，a∈∑ ,定义

  I_a = ε- CLOSURE (J)

  其中：J为从I中某一个状态结点出发，经过一条 a 弧到达的状态结点的全体。

示例

如图所示的NFA状态转换图



设 I={1}，　ε_CLOSURE(I)=

设 I={1, 2 }，I_a=ε_CLOSURE (J)

=

注：实际上，I_a 是从子集I中任一状态出发经a弧（向后可跳过ε弧）而到达的状态集合。

(2) 将M′进一步变换为DFA（续）

① 构造状态转换矩阵；

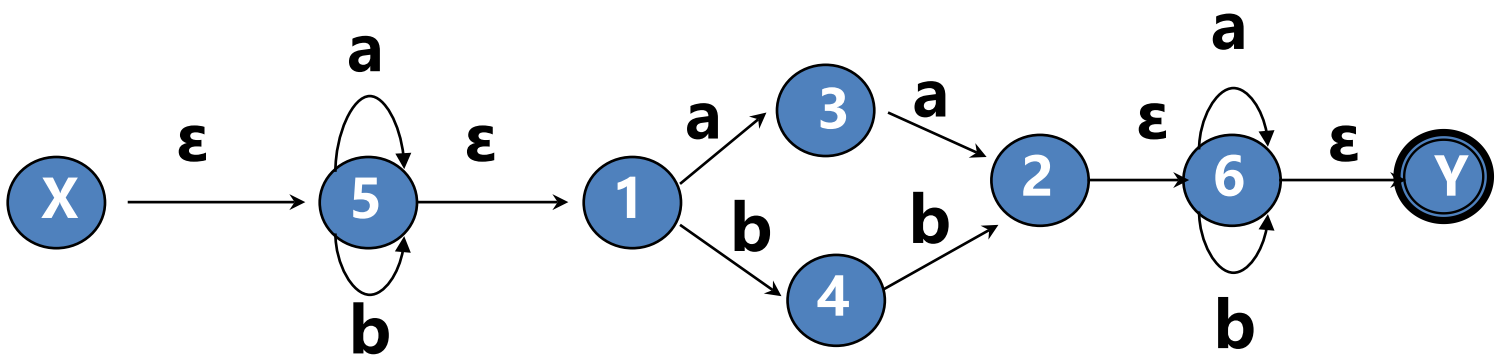
设∑={a,b},构造一张表，表的形式为：

I	I_a	I_b
ε_CLOSURE({X})		

② 把表中第一列的每个子集看做一个新的状态，重新命名，其中，第一行第一列的子集对应的状态是DFA的初态，含有原终态Y的子集是DFA的终态

③ 画出新的 DFA

例1 正规式V = (a | b)*(aa | bb) (a | b)*的NFA状态转换图为



① 利用子集法构造状态转换矩阵

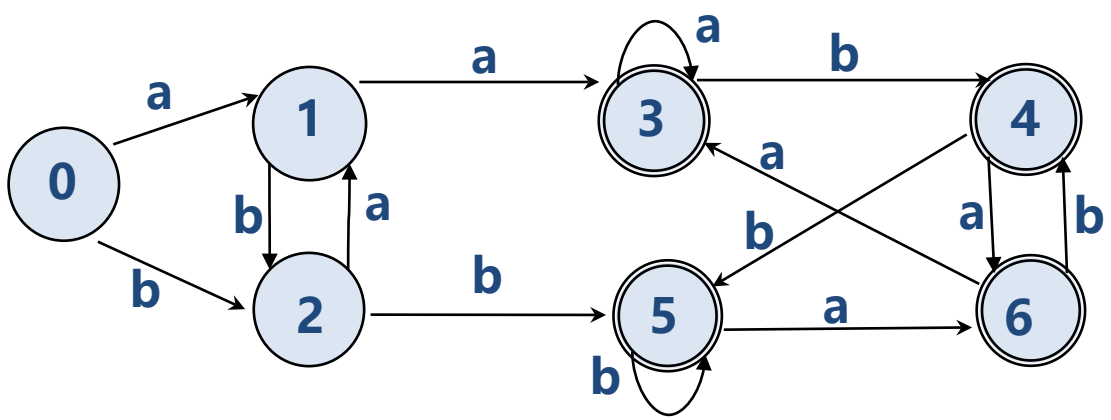
I	I_a	I_b
{X, 5, 1 }	{5, 3, 1 }	{5, 4, 1 }
{5, 3, 1 }	{5, 3, 1, 2, 6, Y }	{5, 4, 1 }
{5, 4,1 }	{5, 3, 1 }	{5, 4, 1, 2, 6, Y }
{5, 3, 1, 2, 6, Y }	{5, 3, 1, 2, 6, Y }	{5, 4, 1, 6, Y }
{5, 4, 1, 2, 6, Y}	{5, 3, 1, 6, Y }	{5, 4, 1, 2, 6, Y }
{5, 4, 1, 6, Y}	{5, 3, 1, 6, Y }	{5, 4, 1, 2, 6, Y }
{5, 3, 1,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

② 对状态子集重新命名得新状态转换矩阵

I		I_a	I_b
{X, 5, 1 }	0	{5, 3, 1 }	{5, 4, 1 }
{5, 3, 1 }	1	{5, 3, 1, 2, 6, Y }	{5, 4, 1 }
{5, 4,1 }	2	{5, 3, 1 }	{5, 4, 1, 2, 6, Y }
{5, 3, 1, 2, 6, Y }	3	{5, 3, 1, 2, 6, Y }	{5, 4, 1, 6, Y }
{5, 4, 1, 6, Y }	4	{5, 3, 1, 6, Y }	{5, 4, 1, 2, 6, Y }
{5, 4, 1, 2, 6, Y}	5	{5, 3, 1, 6, Y }	{5, 4, 1, 2, 6, Y }
{5, 3, 1, 6, Y }	6	{5, 3, 1, 2, 6, Y }	{5, 4, 1, 6, Y }

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

③ 画出状态转换图



DFA的化简

- DFA M 的化简是指寻找一个状态数比 M 少的 DFA M′

  使 L(M) = L(M′)。

- 术语
  - **状态s和t等价:** 若从状态s出发能读出字α停于终态, 则从t出发也能读出α而停于终态；反之,若从状态t出发能读出字α停于终态, 则从s出发也能读出α而停于终态
  - **状态s和t可区别:** 状态s和t不等价。
  - 例如：终态与非终态是可区别的。

DFA化简的思路

- 将 DFA M 的状态集划分为不相交的子集，使不同的两个子集的状态可区别，同一个子集的状态都等价。

DFA化简的步骤

- 把状态集S划分为两个子集，得初始划分

  $\prod = \{ I^{(1)}, I^{(2)} \}$，其中 $I^{(1)}$为**终态集**， $I^{(2)}$为**非终态集**；

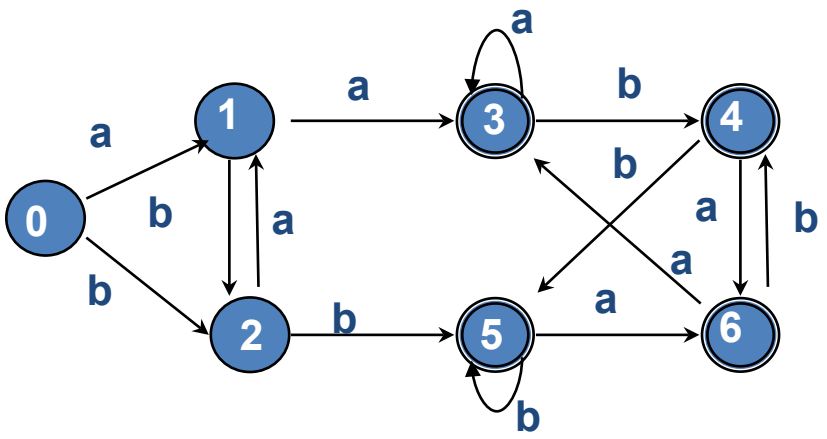
- 设当前 $\prod = \{I^{(1)}, I^{(2)}, \ldots, I^{(m)} \}$, 检查$\prod$中每个$I^{(k)}$是否可以再分

  依据为：如果存在一个输入字符a，使得$I^{(k)}{}_a$不全包含在现行$\prod$的子集中，就将$I^{(k)}$进行划分。

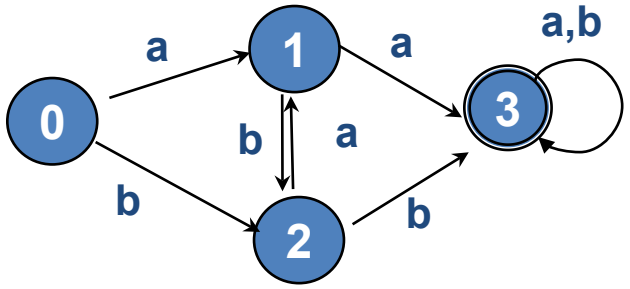
- 一般地，如果$I^{(k)}{}_a$落入现行$\prod$的N个子集中，则应将$I^{(k)}$划分成N个不相交的组，使得每个组$I^{(ki)}$ 的$I^{(ki)}{}_a$都落入$\prod$的同一子集。

- 重复第二步，直至$\prod$中子集数不再增长为止。

化简前



化简后



初始划分$\prod_0$= { $I^{(1)}$, $I^{(2)}$ }, $I^{(1)}$ = {3, 4, 5, 6 }, $I^{(2)}$={0, 1, 2 }

考察$I^{(1)}_a$ = { 3, 6 } 包含于 {3, 4, 5, 6 }

$I^{(1)}_b$ = { 4, 5 } 包含于 {3, 4, 5, 6 }

$I^{(1)}$不可再分, $\prod_0$不变.

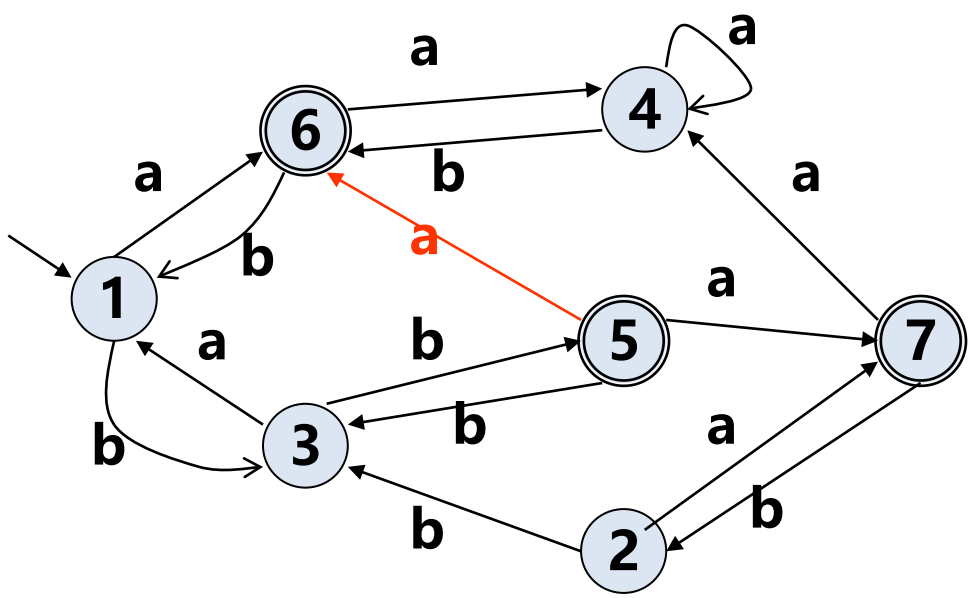
考察 $I^{(2)}_a$ = {1, 3 }，其中$\{1\}_a$=$\{3\}$,$\{0,2\}_a$=$\{1\}$，所以

{ 0, 1, 2 }可分为{ 1 }, {0, 2 }得 $\prod_1$= {{1}, {0, 2 }, {3, 4, 5, 6 }}

考察 $\{0, 2\}_b$ ={2, 5 }, {0, 2 }可分为{0}, {2}

得$\prod_2$={{0}, {1}, {2}, {3, 4, 5, 6}}

令状态3代表{3, 4, 5, 6}, 画出化简后的DFA



Π_0={{1,2,3,4},{5,6,7}}

因为$\{1,2,3,4\}_a$={6,7,1,4}不全包含在Π_0的子集中，需划分。又因为

$\{1,2\}_a$={6,7}落在{5,6,7}集合中，

$\{3,4\}_a$={1,4}落在{1,2,3,4}集合中，

所以得Π_1={{1,2},{3,4},{5,6,7}}

因为$\{3,4\}_a$={1,4}，不全包含在Π_1的子集中,需划分为{3}，{4}得:

Π_2={{1,2},{3},{4},{5,6,7}}

因为$\{5,6,7\}_a$={7,4}，所以

Π_3={{1,2},{3},{4},{5},{6,7}}

内容线索

√. 对于词法分析器的要求

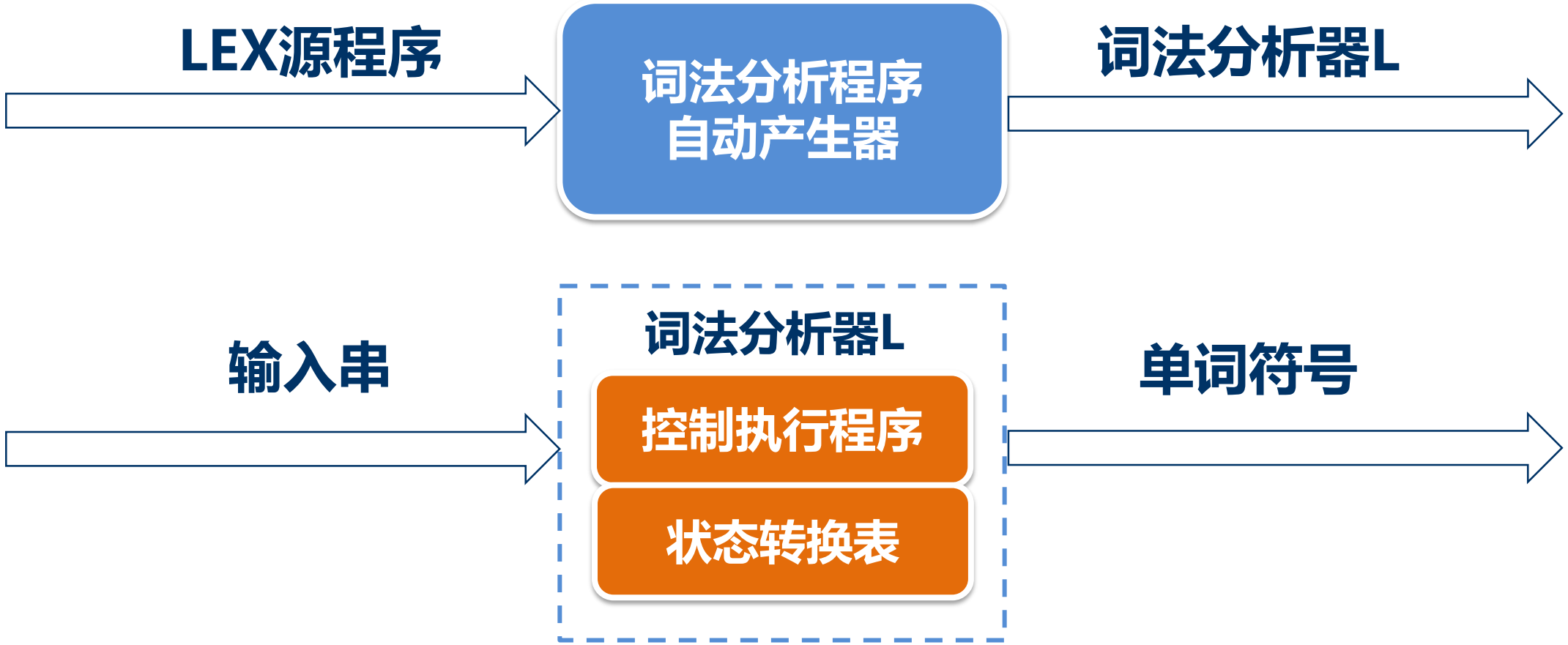
√. 词法分析器的设计

√. 正规表达式与有限自动机

4. 词法分析器的自动生成

词法分析器的自动产生——LEX

- LEX程序由一组正规式以及与每个正规式相应的动作组成。
  - 动作本身是一小段程序代码，它指出了当按正规式识别出一个单词符号时应采取的行动。



语言LEX的一般描述

(1) 正规式辅助定义式

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

...

$d_n \rightarrow r_n$

r_i 为正规式, d_i为该正规式的简名,

r_i中只允许出现 ∑中的字符和已定义的简名

$d_1, d_2, \ldots, d_{i-1}$

LEX源程序包括:

{辅助定义部分}

识别规则部分

{用户子程序部分}

(2) 识别规则：是一串下述形式的LEX语句

P_1 $\{A_1\}$

P_2 $\{A_2\}$

...

P_m $\{A_m\}$

P_i为∑∪{$d_1, d_2, \ldots, d_n$}上的正规式;

A_i 为识别出词形 P_i后应采取的动作，是一小段程序代码。

示例

■ 正规式辅助定义式

letter →A | B | … | Z

digit → 0 | 1 | … | 9

标识符: iden →letter (letter | digit)*

整常数: integer→ digit(digit)*

sign →+ | - | ε

signedinteger →sign integer

不带指数部分的实常数:

decimal →signedinteger . integer

| signedinteger . | sign . Integer

带指数部分的实常数:

exponential →(decimal

| signedinteger) E signedinteger

识别小语言单词符号的 LEX 程序

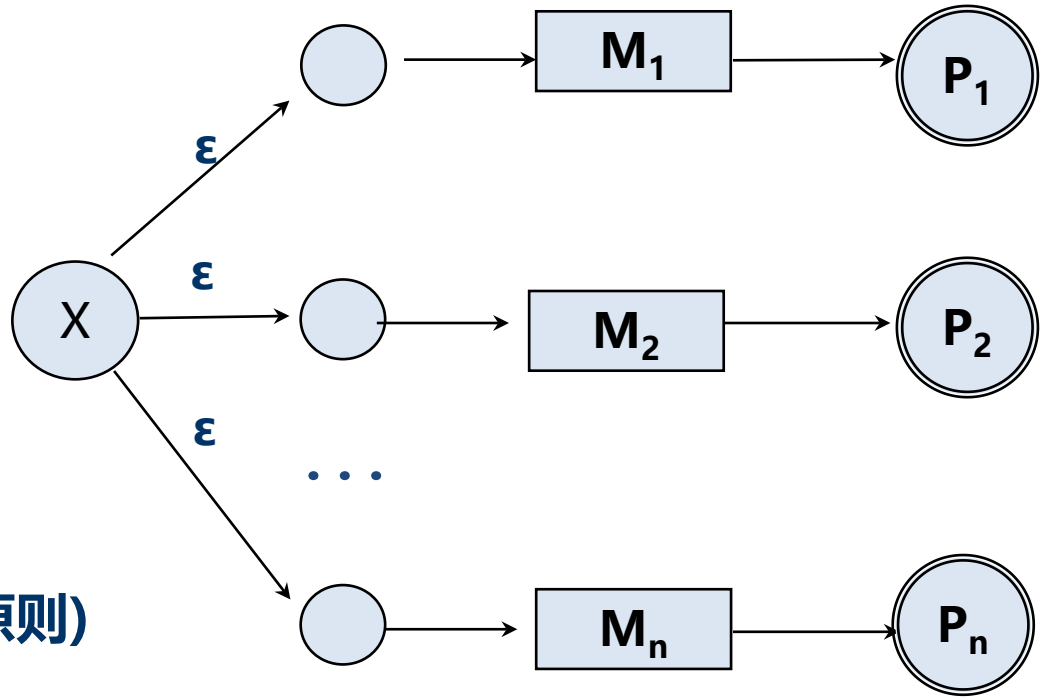
示例

```
AUXILIARY DEFINITIONS   /*  辅助定义  */
                letter→ A | B | . . . | Z
                digit→ 0 | 1 | . . . | 9
RECOGNITION RULES  /*  识别规则  */
1          DIM                    {RETURN (1, _ )}
2          IF                     {RETURN (2, _ )}
3          DO                     {RETURN (3, _ )}
4          STOP                   {RETURN (4, _ )}
5          END                    {RETURN (5, _ )}
6          letter(letter | digit)*  {RETURN (6, getSymbolTableEntryPoint() )}
7          digit (digit)*         {RETURN (7, getConstTableEntryPoint() )}
8          =                      {RETURN (8, _ )}
9          +                      {RETURN (9, _ )}
10         *                      {RETURN (10, _ )}
11         **                     {RETURN (11, _ )}
12         ,                      {RETURN (12, _ )}
13         (                      {RETURN (13, _ )}
14         )                      {RETURN (14, _ )}
```

正规式

LEX 的实现

- 方法
  - 由LEX 编译程序将 LEX 源程序改造为一个词法分析器，即构造相应的 DFA
- 步骤
  - 对每条识别规则Pi构造一个相应的 NFA Mi
  - 引入一个新的初态X, 连接成 NFA M
  - 用子集法将其确定化并化简
  - 将 DFA 转换为词法分析程序
- 注意
  - 匹配最长子串(最长匹配原则)
  - 多个最长子串匹配Pi, 以前面的Pi为准(优先匹配原则)

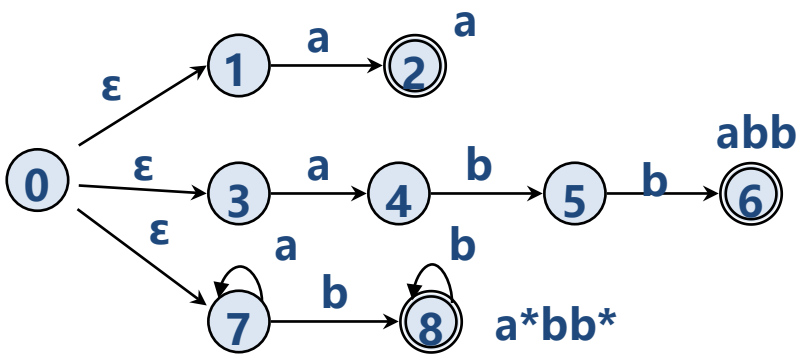


示例

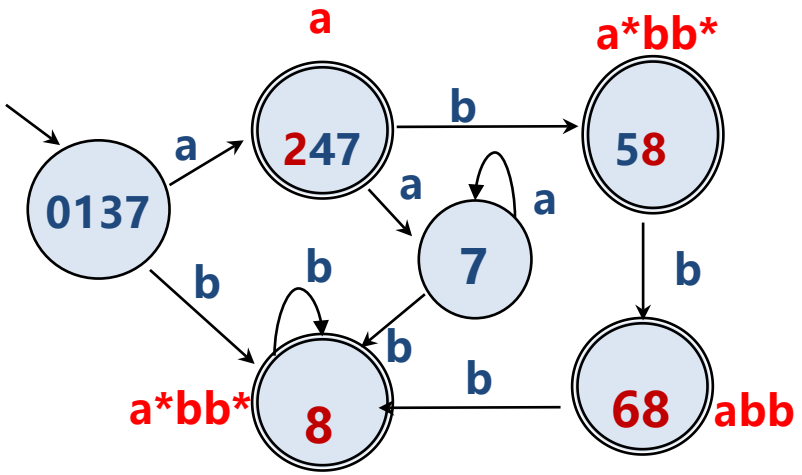
LEX 程序:

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

NFA M:



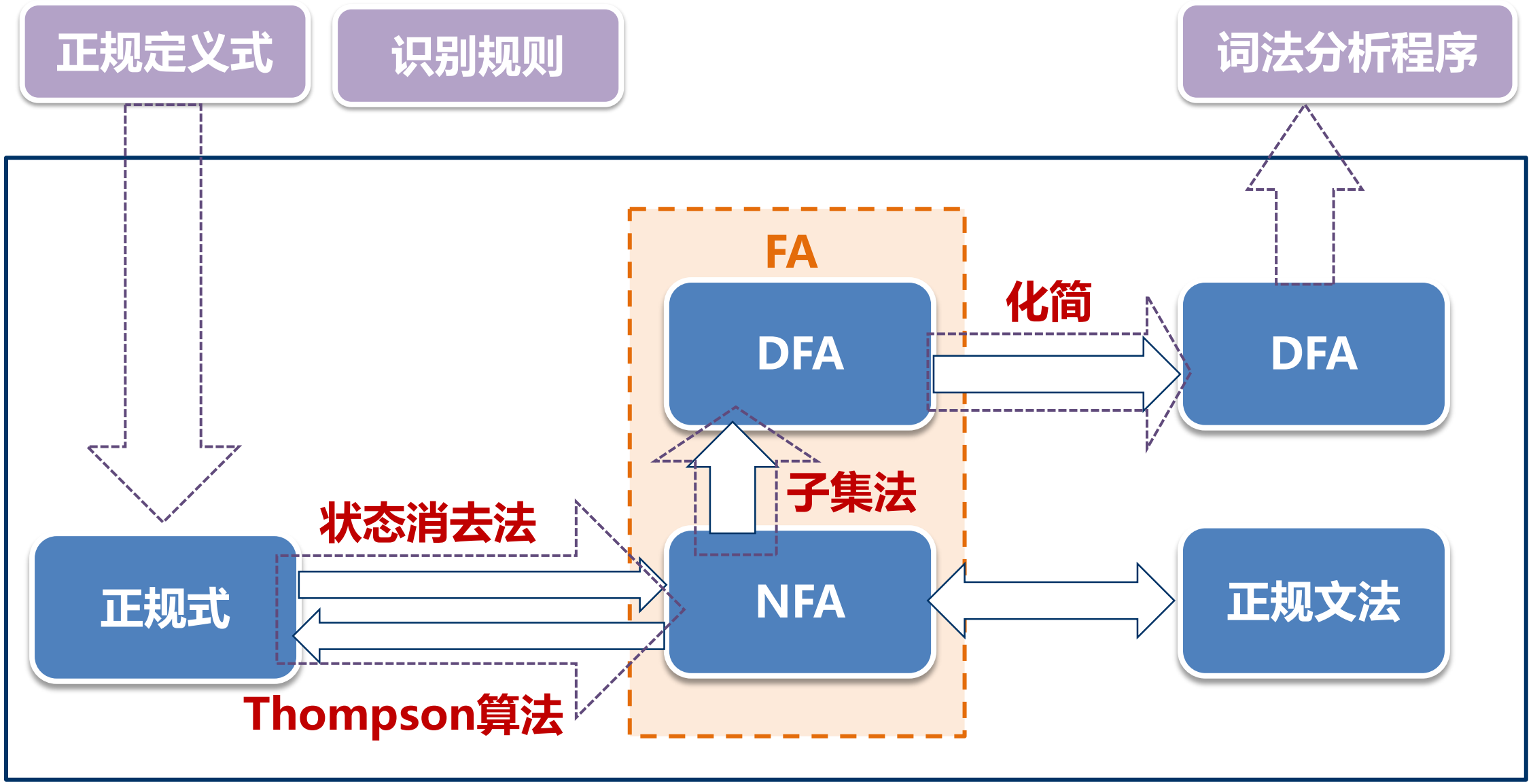
状态	a	b	识别单词
0 1 3 7	2 4 7	8	
2 4 7	7	5 8	a
8		8	a*bb*
7	7	8	
5 8		6 8	a*bb*
6 8		8	abb



输入：abbbabb

输出：abbb　abb

总结


正规定义式
识别规则
词法分析程序
FA
DFA
化简
DFA
子集法
状态消去法
正规式
NFA
正规文法
Thompson算法

作业

- P63
  - 6 (5)
  - 8 (1) (2)
  - 12