第3章

词 法 分 析

本章介绍词法分析器的设计与实现，并介绍自动机、正规文法这些词法分析的核心工具，以及它们之间、它们与正规式之间的相互转换。

3.1 词法分析器的设计

3.1.1 词法分析器的任务

在1.5.1节编译器框架中，我们已经介绍了词法分析器的输出是单词类别和单词值的二元组序列，或者称为单词串。

单词识别的最直观想法是通过分隔符（如空格、回车换行符等）把单词隔开，分割成一个个子串，然后每个子串根据其构成判断其类别。这种方式编程非常复杂，需要根据不同情况做各种适配，在源程序文件较小时可以适用，但效率极差，源程序规模较大时其运行时间是不可接受的。词法分析器的任务，是期望对源代码字符串进行一次从左到右的扫描，就能识别出所有单词及其类别。

单词的类别一般分为以下5种。

- **标识符**：用来表示各种名字，如变量名、数组名、过程名、标号名等。
- **关键字**：由程序语言定义的有固定意义的标识符，也称为保留字或基本字，如int、while、if等。
- **常数**：一般有整型、实型、布尔型、字符型、字符串型等，如7、3.1415926、true、“Hello world!”。
- **运算符**：如+、−、*、/等。
- **界符**：如逗号、分号、括号、//、/*、*/等。

其中，关键字、运算符和界符都是确定的，一般根据语言的复杂程度不同，有几十个或上百个。标识符和常数的数量一般不加限制，但标识符一般有长度限制。整型、实型、布尔型、字符型常数一般由固定长度的字节表示，字符串型则受数组最大长度限制。

一般来说，为了使用方便，单词类别在程序中使用整型编号表示。一个语言的单词符号如何分类、分几类、怎样编码，是一个技术性问题，主要取决于处理上的方便，大致规则如下。

- 标识符一般统归为一类，因为词法分析时，这个标识符表示的是变量名、数组名、过程名，还是标号名，是无法确定的，适合于统一归类。

- 常数宜按类型分类，如整型、实型、布尔型、字符型、字符串型各一类。
- 关键字可以将全体视为一类，也可以一字一类。
- 运算符可以一符一类，也可以把具有一定共性的运算符视为一类，如加减一类、乘除一类等。注意：有些二义符号，如减号和负号，需要到语义分析阶段才能区分。
- 界符一般一符一类。

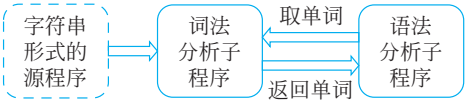
3.1.2 词法分析器设计需要考虑的问题

词法分析器设计需要考虑几个问题，首先考虑的是**把词法分析器作为独立的一遍，还是作为一个子过程**。

作为独立的一遍，即输入原始字符串，输出完整的单词二元组序列，再送给语法分析器，如图3.1(a)所示。这种方式的优点是结构非常清晰，缺点是需要先把词法分析跑完一遍，如果语法分析一开始就发现了错误，那么词法分析这一遍的后续工作都白做了。

把词法分析作为一个子程序，即公开一个过程，语法分析需要一个单词就调用这个过程。词法分析器分析出一个单词就返回，然后等待语法分析器的下一次调用，如图3.1(b)所示。这种方式与作为独立一遍相比，优缺点恰好相反，可以根据语言的特点和喜好选择。

(a) 词法分析器作为独立的一遍



(b) 词法分析器作为一个独立子程序

图 3.1 词法分析器结构

第二个问题是源程序中**注释和空白**的处理问题，这里的空白包括回车符、换行符、跳格符和空格符。最早的编译器把词法分析分为过滤（Screening）和扫描（Scanning）两个阶段[61]，过滤即词法分析前的**预处理**过程，其作用是将注释和空白转换为一个空格符，将连续的注释和空白合并为一个空格；扫描即识别单词的阶段。实际上，注释和空白也可以与单词识别一起处理，读到这些字符直接丢弃即可，在实现部分我们会讨论这种方式。

第三个问题是读入源程序的**缓冲区**问题。目前大部分计算机系统能提供给编译器使用的内存空间，相对于源程序文件大小来说，可以说是无限大的。但同时也无法避免在资源受限机器上跑编译程序的情况。假如某台机器的内存不足以装下整个源程序，就会出现内存中一个完整单词被截断的情况。

当内存受限时，需要设定一个缓冲区长度，也就是一次读入多少字符，假设这个长度是512，而标识符最大长度限定是256。当识别一个单词时，不管缓冲区设置得多大，只要不能完整读入源程序，总会出现如图3.2(a)所示的情况：一个指针记录了标识符的起点，然后搜索指示器向右扫描，未扫描到单词结束位置，缓冲区到末尾了，这样就造成一个完整的单词不在内存里，无法记录下来。一种可行的解决方案是，把从标识符起点到缓冲区末尾的部分移动到缓冲区头部，读入源程序剩余部分，把缓冲区剩下的部分填满，然后继续识别这个单词。这种方式需要在内存中移动部分字符，导致效率下降。

双缓冲策略则可以解决这种问题。如图3.2(b)所示，我们将缓冲区一分为二，每个缓冲区长度不小于标识符最大长度，两个缓冲区形成环状结构，即第2个缓冲区链接在第1个缓冲区尾部，第1个缓冲区也链接在第2个缓冲区尾部。当一个缓冲区在当前识别的单词未结束时到达了缓冲区尾部，就将另一个缓冲区填满，从另一个缓冲区的头部继续搜索。两个缓冲区互补使用，就能保证一个单词总在内存中，直至整个源程序扫描结束。

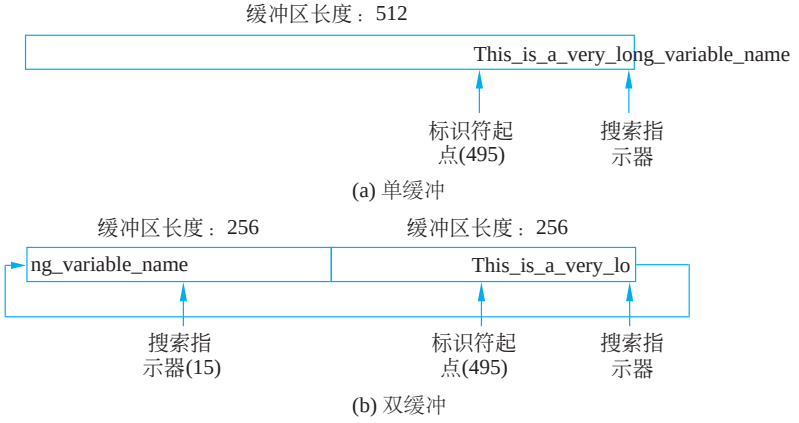


图 3.2　词法分析器缓冲区策略

3.1.3　状态转换图

状态转换图是对具有多种状态的程序进行控制的有效方法，对识别单词也很有效。**状态转换图**（State Transition Diagram，STD）是一个有向图，用结点表示状态，有向边上的**标记**为字符，表示在一个状态下，若输入字符为该标记，则转移到有向边指向的状态。

一个转换图只有有限个状态，即结点有限，如图3.3所示。这些状态中有一个状态称为**初态**，用箭头表示，如状态0；至少有一个**终态**，用双圈表示，如状态2。

设 $\Sigma = \{a, b, \cdots, z, A, B, \cdots, Z, 0, 1, \cdots, 9\}$，图3.3(a)是识别 Σ 上标识符的状态转换图。初始状态为0，当遇到一个字母符号时，就转移到状态1；再遇到字母或数字时，还是转移到状态1；当遇到不是字母和数字的其他符号时，转移到终态2，表示识别出了一个单词。图3.3(b)则是识别 Σ 上整数的转换图，可以做类似的分析。

图 3.3　状态转换图

终态上加星号，表示多读了一个字符，需要把这个字符退回给词法分析器，这个多读符号的策略称为**超前搜索**。像FORTRAN这样的语言，关键字可以用作普通标识符，导致需要超前搜索很多字符才能确定该单词是关键字还是普通标识符。而目前的大部分语言，不允许把关键字作为普通标识符使用，同时除算符和界符外两个单词之间要求至少有一个空白符，这样最多只超前搜索1个字符，就可以识别出一个单词。

可以看出，如果能把识别各类单词的状态转换图组合成一个整体，并且当前状态和输入符号能唯一确定下一个状态时，就可以方便地实现单词识别。一开始可以设置状态为初态0，字符扫描指示器指向第一个字符。每步根据当前状态和当前字符确定下一个状态，状

态转移后扫描指示器读入下一个符号。当到达终态后，就成功识别出一个单词。然后把状态重置为初态，字符扫描指示器回退一个字符，重新开始这个过程去识别下一个单词。

本章后续部分主要研究状态转换图的构造理论，最后用状态转换图实现一个词法分析器。

3.2 有限自动机

当一个问题比较复杂时，通常从以下两个角度切入。

- 从“人”的角度，设计一个人类比较容易理解且容易构造的规则或模型。对单词描述来说，2.3.1节介绍的正规式正是这样一个工具。
- 从“计算机”的角度，设计一个计算机算法比较容易实现的模型。前面所述的状态转换图是一个好的工具，但是需要对其施加一定的限制，后面定义为确定有限自动机。

然后，我们需要寻找一个算法，将人构造的模型转换为计算机能理解的模型。

3.2.1 确定有限自动机

对3.1.3节介绍的状态转换图，只有当一个状态和一个输入符号能唯一确定下一个状态，才能比较方便地编程实现。下面对具有这个特点的状态转换图给出形式化定义。

定义 3.1（确定有限自动机（Deterministic Finite Automata，DFA））

一个确定有限自动机 M 是一个五元组：$M=(S,\Sigma,\delta,s_0,F)$，其中

(1) S 是状态的非空有限集合，$\forall s\in S$，s 称为 M 的一个状态；

(2) Σ 是一个有限字母表，它的每一个元素称为一个输入字符（或符号、字母）；

(3) $\delta: S\times\Sigma\to S$，是状态转换函数（也称状态转移函数）集，对 $\forall(s,a)\in S\times\Sigma$，$\delta(s,a)=t$ 表示 M 在状态 s 读入字符 a，转移到状态 t，并向右移动读入下一个字符；

(4) $s_0\in S$，是唯一的初态；

(5) $F\subseteq S$，是一个终态集，其中的每个元素称为终态（或可接受状态）。

DFA 的最重要特点是 δ，它是一个从 $S\times\Sigma$ 到 S 的单值映射。对 $\delta(s,a)=t$，称 t 为 s 的后继状态，这里 t 是唯一确定的，不会同一个状态有两条标记相同的射出弧导向不同状态。换句话说，如果有 $\delta(s,a)=t_1,\delta(s,a)=t_2$，则 t_1 和 t_2 一定是同一个状态。

DFA 的确定性还体现在，DFA 上没有 ε 弧。如果有，则意味着一个状态不经过任何输入就可以到达另一个状态，这也是不允许的。

初态 s_0 是唯一的，意味着开始状态也是唯一确定的。终态集 F 可空，但没有终态的 DFA 对我们没有意义，因此我们只讨论 F 非空的情形。

例题 3.1DFA 定义 写出图3.4所示状态转换图对应的DFA。

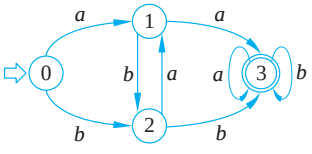


图 3.4 状态转换图

解 DFA $M=\{\{0,1,2,3\},\{a,b\},\boldsymbol{\delta},0,\{3\}\}$，其中 $\boldsymbol{\delta}$：$\delta(0,a)=1,\delta(0,b)=2,\delta(1,a)=$

$3, \delta(1,b)=2,\ \delta(2,a)=1, \delta(2,b)=3, \delta(3,a)=3, \delta(3,b)=3.$

定义 3.2 (状态转换矩阵)

一个DFA的转换函数可以用表格或矩阵表示。表格的行表示状态，列表示输入字符，单元格元素表示$\delta(s,a)$的值，其对应的矩阵称为状态转换矩阵。

例题 3.2 状态转换矩阵 例题3.1的DFA的转换函数用表格表示如表3.1所示。

表 3.1 表格形式的状态转换矩阵

状　态	a	b
0	1	2
1	3	2
2	1	3
3	3	3

用状态转换矩阵表示如式(3.1)所示。

$$\boldsymbol{\delta}=\begin{pmatrix}1 & 2\\ 3 & 2\\ 1 & 3\\ 3 & 3\end{pmatrix} \tag{3.1}$$

状态转换矩阵的元素排列，与状态、输入符号的排列顺序都有关，因此把表格也称作状态转换矩阵。

定义 3.3 (字的识别与语言)

对Σ^*上的任何字α，若存在一条从初态结点到某一终态结点的通路，且这条通路上所有有向弧的标记符连接成的字等于α，则称α可为DFA M所识别（也称读出或接受）。

若M的初态结点同时又是终态结点，则空字ε可为M所识别。

DFA M所能识别的字的全体记为$L(M)$，称为DFA M所能识别的语言。

如果一个DFA M的输入字母表为Σ，也称M是Σ上的一个DFA。

定理 3.1 (正规集与DFA的等价性)

Σ上的字集$V\subseteq\Sigma^*$是正规的，等价于存在Σ上的DFA M，使得$V=L(M)$。

对于这个定理，在本章后续内容通过构造法证明，即对任意的正规式对应的字集V，都能构造DFA M使得$V=L(M)$；反之，对于任意DFA M，都能构造一个正规式，假设它表示的字集是V，则有$V=L(M)$。

下面先看几个DFA分析和构造的例子。

例题 3.3 分析DFA 分析图3.4的DFA所能识别的字（字集）。

解 DFA要识别一个字，必须进入终态，此处终态为3。

要进入终态3，有两条路径：状态1通过a弧进入，或状态2通过b弧进入，此外没有其他路径。

若走状态1的路径，因为状态1只有两条射入的弧，标记都为a，因此该路径要到达状态3，必须先接受1个a到达状态1，再接受一个a到达状态3，故接受包含aa的字。

用同样的分析可得，若走状态2的路径，接受包含bb的字。

故该DFA接受包含aa或bb的字。

例题3.4 设计DFA 设计$\Sigma=\{0,1\}$上的DFA M，使其能识别有偶数个0偶数个1的字。

解 DFA设计问题的关键，是找到状态和转换函数，这两个要素又密切相关。

识别一个字，可以看作从初态到终态状态变化的一个动态过程，每走一步，接受一个字符。

对本问题，可以考虑统计每一步识别出数字中0和1的个数的奇偶性，不同奇偶性组合作为一个状态；从某个状态再输入一个数字0或1，就导致状态改变。

我们对状态做如下编号。

- 状态0：偶数个0偶数个1。此时后面如果读入0（通过0弧转移），则变为奇数个0偶数个1；如果读入1，则变为偶数个0奇数个1。
- 状态1：偶数个0奇数个1。此时后面如果读入0，则变为奇数个0奇数个1；如果读入1，则变为偶数个0偶数个1。
- 状态2：奇数个0偶数个1。此时后面如果读入0，则变为偶数个0偶数个1；如果读入1，则变为奇数个0奇数个1。
- 状态3：奇数个0奇数个1。此时后面如果读入0，则变为偶数个0奇数个1；如果读入1，则变为奇数个0偶数个1。

根据以上分析，得到转换函数：

$\delta(0,0)=2,\delta(0,1)=1,\delta(1,0)=3,\delta(1,1)=0,\delta(2,0)=0,\delta(2,1)=3,\delta(3,0)=1,\delta(3,1)=2$

终态显然是偶数个0偶数个1的状态，即状态0；空字也符合偶数个0偶数个1的要求，所以初态也是状态0。DFA如图3.5所示。

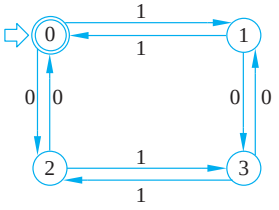


图 3.5 接受偶数个0偶数个1的DFA

例题3.5 设计DFA 设计$\Sigma=\{0,1,\cdots,9\}$上的DFA M，使其接受能被3整除的十进制数字。

解 假设到某个状态已识别出的数字部分记作n，那么经过一个标记为i的弧后，相当于在n后加了一位数字i，这个数字就变为$10n+i$。

可以考虑将数字除以3的余数作为状态，那么后面加上一位数字i后，其除以3的余数也是唯一确定的，即转移到的状态唯一确定。

- 状态0：$n\%3=0$，则$10n\%3=0$

  - $(10n+(0|3|6|9))\%3=0$
  - $(10n+(1|4|7))\%3=1$
  - $(10n+(2|5|8))\%3=2$
- 状态1：$n\%3=1$，则$10n\%3=(9n+n)\%3=n\%3=1$
  - $(10n+(0|3|6|9))\%3=1$
  - $(10n+(1|4|7))\%3=2$
  - $(10n+(2|5|8))\%3=0$
- 状态2：$n\%3=2$，则$10n\%3=(9n+n)\%3=n\%3=2$
  - $(10n+(0|3|6|9))\%3=2$
  - $(10n+(1|4|7))\%3=0$
  - $(10n+(2|5|8))\%3=1$

根据以上分析，DFA如图3.6所示。终态显然是余数为0的状态，即状态0；用排除法可以确定初态为0，但会接受空字。

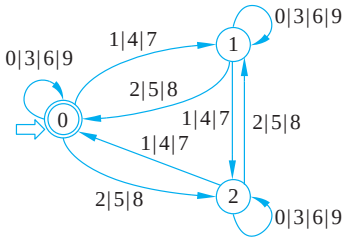
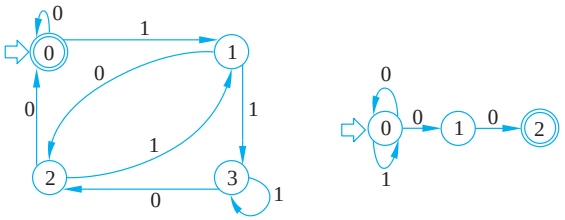


图 3.6　接受能被3整除的十进制数字的DFA

例题3.5的思路可以解决任意进制的余数或整除问题。如构造$\Sigma=\{0,1\}$上能被4整除的二进制数，余数为$0\sim3$共4个状态。到达当前状态识别的数字如果为n，则再读入数字0变为$2n$，读入数字1变为$2n+1$，这样得到的DFA如图3.7(a)所示。



(a) DFA　　(b) 非DFA

图 3.7　接受能被4整除的二进制数字的状态转换图

有时候利用一些规则更容易构造状态转换图，但这样构造时一定要谨慎。如能被4整除的二进制数字，其最后两位必须都是0。根据这个规则，可能构造出图3.7(b)所示的状态转换图。抛开是否接受空串这一点，该状态转换图确实能接受被4整除的二进制数，但它不是DFA，因为同时存在$\delta(0,0)=0$和$\delta(0,0)=1$。

例题3.6 应用DFA　一个人带着狼、羊和白菜要从一条河左岸渡到右岸。有一条船，恰好能装下人和其他三件东西中的一件。如果没有人照看，狼会吃羊，羊会吃白菜。用DFA找出渡河方案。

解　把人和各种东西存在左岸和右岸的情况作为状态，如表3.2所示。初态是人、狼、羊、白菜都在左岸，终态是人、狼、羊、白菜都在右岸。

表 3.2 状态编码

状 态	左 岸	右 岸	说 明
0	人、狼、羊、白菜	∅	初态
1	人、狼、羊	白菜	
2	人、狼、白菜	羊	
3	人、羊、白菜	狼	
—	狼、羊、白菜	人	左岸狼吃羊，羊吃白菜
—	人、狼	羊、白菜	右岸羊吃白菜
4	人、羊	狼、白菜	
—	人、白菜	狼、羊	右岸狼吃羊
—	狼、羊	人、白菜	左岸狼吃羊
5	狼、白菜	人、羊	
—	羊、白菜	人、狼	左岸羊吃白菜
—	人	狼、羊、白菜	右岸狼吃羊，羊吃白菜
6	狼	人、羊、白菜	
7	羊	人、狼、白菜	
8	白菜	人、狼、羊	
9	∅	人、狼、羊、白菜	终态

有向弧上标记的符号，用$G(x)$和$R(x)$分别表示人带x从左岸到右岸和从右岸返回，G和R分别表示人自己到右岸和从右岸返回，则可构造转换矩阵如表3.3所示。

表 3.3 状态转移矩阵

状态	左 岸	G(狼)	G(羊)	G(白菜)	G	R(狼)	R(羊)	R(白菜)	R
0	人、狼、羊、白菜	—	5	—	—	—	—	—	—
1	人、狼、羊	7	6	—	—	—	—	—	—
2	人、狼、白菜	8	—	6	5	—	—	—	—
3	人、羊、白菜	—	8	7	—	—	—	—	—
4	人、羊	—	9	—	7	—	—	—	—
5	狼、白菜	—	—	—	—	—	0	—	2
6	狼	—	—	—	—	—	1	2	—
7	羊	—	—	—	—	1	—	3	4
8	白菜	—	—	—	—	2	3	—	—
9	∅	—	—	—	—	—	4	—	—

从DFA中找到从状态0到状态9的一条通路就是一个渡河方案，最短通路就是最佳渡河方案。状态转换图比较凌乱，可以通过状态转换矩阵寻找通路。

我们找到一条最短路径：

$$0 \xrightarrow{G(羊)} 5 \xrightarrow{R} 2 \xrightarrow{G(狼)} 8 \xrightarrow{G(羊)} 3 \xrightarrow{G(白菜)} 7 \xrightarrow{R} 4 \xrightarrow{G(羊)} 9$$

即渡河方案为①人带羊到右岸；②人自己返回；③人带狼到右岸；④人带羊返回；⑤人带白菜到右岸；⑥人自己返回；⑦人带羊到右岸。

通过前面的例题可以看出，DFA虽然易于编程实现，但是构造非常困难。人类容易构造的，往往是“非确定”的有限自动机。如构造能被4整除的二进制数字时，显然图3.7(b)比图3.7(a)更容易想到。又如，构造接受aa或bb的DFA，构造图3.4是非常困难的，但构造“非确定”的图3.8却很容易。因此，我们放松DFA中确定性的限制，给出非确定有限自动机的概念。

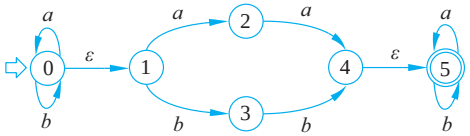


图 3.8　包含aa或bb的NFA

3.2.2　非确定有限自动机

定义 3.4 (非确定有限自动机（Non-Deterministic Finite Automata，NFA）)

一个非确定有限自动机M是一个五元组：$M=(S,\Sigma,\hat{\delta},S_0,F)$，其中

(1) S是状态的非空有限集合；

(2) Σ是一个有限字母表；

(3) $\hat{\delta}:S\times\Sigma^*\to 2^S$，是扩充的状态转换函数；

(4) $S_0\subseteq S$，是非空初态集；

(5) $F\subseteq S$，是可空终态集。

其中扩充的状态转换函数$\hat{\delta}$定义如下。

定义 3.5 (扩充的状态转换函数)

$\hat{\delta}:S\times\Sigma^*\to 2^S$由$\delta$定义扩充而来，主要扩充规则为：

(1) $\hat{\delta}(s,\varepsilon)=\{s\}$；

(2) $\hat{\delta}(s,wa)=\{t\mid\exists r\in\hat{\delta}(s,w)\wedge t=\delta(r,a)\}$，其中$a\in\Sigma,w\in\Sigma^*$。

其中第(1)条规则定义了通过空符号到达自身；第(2)条规则指明状态s通过符号串w到达状态r（可能不止一个），r再通过a到达t，那么s通过wa到达t。

当$(s,a)\in S\times\Sigma$时，δ和$\hat{\delta}$是一致的，证明见式(3.2)。

$$\begin{aligned}\hat{\delta}(s,a) &= \hat{\delta}(s,\varepsilon a)\\ &= \{t\mid\exists r\in\hat{\delta}(s,\varepsilon)\wedge t=\delta(r,a)\}\\ &= \{t\mid\exists r\in\{s\}\wedge t=\delta(r,a)\}\\ &= \{t\mid t=\delta(s,a)\}\\ &= \{\delta(s,a)\}\end{aligned}\tag{3.2}$$

NFA一般是**带空移动**的，即含有ε弧，可以经过空符号进行移动，需要区分时这种NFA称为ε-NFA。在ε-NFA中，δ和$\hat{\delta}$是不同的，我们只考虑$\delta/\hat{\delta}:S\times(\Sigma\cup\{\varepsilon\})\to 2^S$的情况：

- 当$a\neq\varepsilon$时，$t=\delta(s,a)$表示s上有一条a弧到达t；而$t\in\hat{\delta}(s,a)$表示s可能经过了

很多条箭弧才到达t，其中一条必须是a弧，其他都是ε弧。

- 当s到自身有ε弧时，$s=\delta(s,\varepsilon)$；而s到自身没有ε弧时，$s\neq\delta(s,\varepsilon)$。但不管$s$到自身有没有$\varepsilon$弧，$s\in\hat{\delta}(s,\varepsilon)$总是成立的。

例题3.7DFA与NFA 图3.8所示的ε-NFA，其状态转换函数δ和扩充状态转换函数$\hat{\delta}$的比较如表3.4所示。

表 3.4 状态转换函数δ和扩充状态转换函数$\hat{\delta}$的比较

状 态	δ			$\hat{\delta}$		
	ε	a	b	ε	a	b
0	1	0	0	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
1	—	2	3	$\{1\}$	$\{2\}$	$\{3\}$
2	—	4	—	$\{2\}$	$\{4,5\}$	$\varnothing$
3	—	—	4	$\{3\}$	$\varnothing$	$\{4,5\}$
4	5	—	—	$\{4,5\}$	$\{5\}$	$\{5\}$
5	—	5	5	$\{5\}$	$\{5\}$	$\{5\}$

有时为了书写简洁，会用δ表示$\hat{\delta}$，可以通过上下文进行区分。

由于NFA的弧上允许为Σ^*，因此只要能写出正规式，就可以构造NFA为只有一个初态和一个终态，把正规式标记到初态到终态的弧上，如包含aa或bb的NFA，可以构造NFA如图3.9所示。

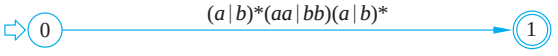


图 3.9 包含aa或bb的NFA

目前，我们已经找到了构造NFA的简单方法，它的难度和正规式构造是等价的，只要能构造出正规式，就能构造出NFA。那么，如果NFA能转换成DFA，则解决了DFA构造难的问题，下面讨论NFA确定化为DFA的问题。

3.2.3 非确定有限自动机确定化

显然，DFA是NFA的一个特例，如果每个NFA也对应一个DFA，则证明了DFA和NFA等价。

定理 3.2 (NFA与DFA的等价性)

对于每个NFA M，存在一个DFA M'，使得$L(M)=L(M')$；即NFA与DFA是等价的。 ♡

这个定理可以使用构造法进行证明，即对每个NFA，都可以构造一个等价的DFA。这里我们不对这个定理进行严格的数学证明，只给出NFA构造DFA的算法。

NFA确定化算法

NFA和DFA的区别，一是NFA有多个初态，而DFA只有一个初态；二是状态转换函

数不同。对前者，先将NFA的初态化为唯一，为统一起见，也使终态唯一。对后者，分两步进行：把映射 $S \times \Sigma^* \to 2^S$ 转换为 $S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$，再进一步转换为 $S \times \Sigma \to S$。

算法3.1为NFA确定化过程，输入为NFA M，输出为确定化后的DFA M'。算法共分3步。

(1) 第1行的makeSingleStartAndEndState()函数，将NFA M 的初态和终态唯一化。

(2) 第2行的splitArrows()函数，将NFA M' 的每个箭弧分裂为只有单个符号或ε，即将映射 $S \times \Sigma^* \to 2^S$ 转换为 $S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$，称为箭弧单符化。

(3) 第3行的determineNFA()函数，将NFA M' 确定化，即将映射 $S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ 转换为 $S \times \Sigma \to S$。

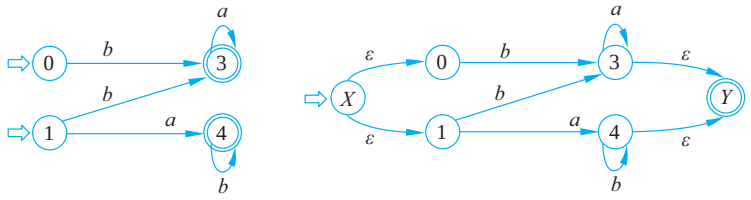
算法 3.1 NFA确定化

输入： NFA $M = (S, \Sigma, \delta, S_0, F)$
输出： DFA M'，使得 $L(M) = L(M')$
1 $M' = \text{makeSingleStartAndEndState}(M)$;
2 $M' = \text{splitArrows}(M')$;
3 $M' = \text{determineNFA}(M')$;

下面分别介绍这3个函数的相关算法，最后举例说明其执行过程。

1. 初态和终态唯一化

使NFA的初态和终态都唯一，可以引入新初态 X，从 X 向原初态引ε弧；引入新终态 Y，从原终态向 Y 引ε弧。这样转换后，初态和终态唯一，且与原NFA等价。如图3.10(a)的NFA有两个初态和两个终态，经变换后得到图3.10(b)的NFA，其只有一个初态 X 和一个终态 Y。



(a) 原NFA　　(b) 初态和终态唯一后的NFA

图 3.10　使初态和终态唯一

算法3.2将NFA初态和终态化为唯一。算法输入为具有多个初态和终态的NFA M，输出的NFA M' 初态和终态均唯一。

算法 3.2 使NFA初态和终态唯一

输入： NFA $M = (S, \Sigma, \delta, S_0, F)$
输出： NFA $M' = (S', \Sigma', \delta', S_0', F')$
1 **NFA** makeSingleStartAndEndState(M)**:**
2 　$S' = S \cup \{X, Y\}, \Sigma' = \Sigma, \delta' = \delta, S_0' = \{X\}, F' = \{Y\}$;
3 　**foreach** $s \in S_0$ **do**
4 　　$\delta' \cup = \{\delta(X, \varepsilon) = s\}$;
5 　**end**
6 　**foreach** $s \in F_0$ **do**
7 　　$\delta' \cup = \{\delta(s, \varepsilon) = Y\}$;
8 　**end**

算法 3.2 续

```
9   |  return M';
10  end makeSingleStartAndEndState
```

第2行初始化，状态集 S' 在原状态集 S 基础上增加两个新的状态 X 和 Y，字母表和状态转换函数取原NFA M 的相应集合；新的初态集 S'_0 中只有一个初态 X，新的终态集 F' 中只有一个终态 Y。第3~5行从 X 向原初态引 ε 弧，第6~8行从原终态向 Y 引 ε 弧。最后将得到的 M' 返回（第9行）。

2. 箭弧单符化

使每条箭弧上或者是单个字符，或者是 ε，简称箭弧单符化。这个过程的本质是把映射 $S \times \Sigma^* \to 2^S$ 转换为 $S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$。对NFA反复施行图3.11的变换，直至每个有向边或者标记为单个字符，或者标记为 ε。图3.11中，k 为新插入的状态结点。

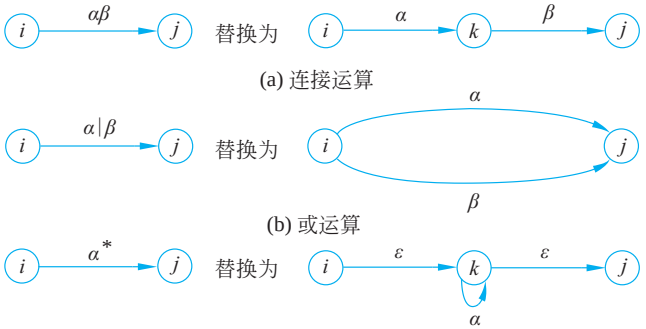


图 3.11 使每个有向边或者标记为单个字符，或者标记为 ε

箭弧上的正规式有多个运算符时（包括省略的连接运算符），要从优先级最低的运算符开始分裂。

例题 3.8 箭弧分裂 选择分裂的运算符。

- $\alpha\beta^*$，省略的连接运算符优先级最低，因此可以看作 α 与 β^* 的连接，选择图3.11(a)进行替换。
- $\alpha^*|\beta\gamma$，或运算优先级最低，因此可以看作 α^* 与 $\beta\gamma$ 的或，选择图3.11(b)进行替换。
- $\alpha^*\beta|\gamma\delta$，或运算优先级最低，因此可以看作 $\alpha^*\beta$ 与 $\gamma\delta$ 的或，选择图3.11(b)进行替换。
- $(\alpha|\beta)(\gamma|\delta)$，由于括号优先级最高，因此它们之间的连接运算优先级最低，可以看作 $(\alpha|\beta)$ 与 $(\gamma|\delta)$ 的连接，选择图3.11(a)进行替换。
- 由于闭包优先级高，因此只有 α^* 这种形式时，才会用到图3.11(c)进行替换。

目前阶段，构造自动分裂算法还是不太可能的。在学习到语法制导翻译，直到7.2.3节，我们才给出一个完美的箭弧分裂算法：通过扫描整个正规式，同时构造出NFA，使得每条边或者是单个字符或者是 ε。目前splitArrows()这个过程，我们只能采用手工分裂方式进行替代。

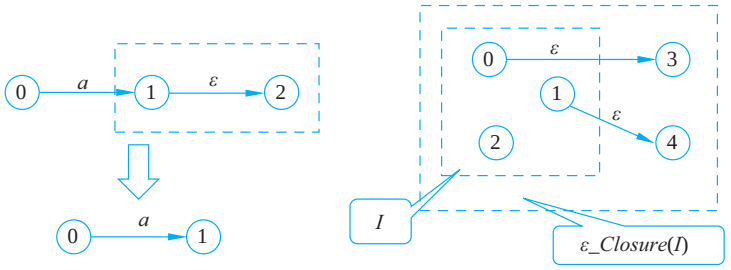
3. NFA 确定化

NFA确定化即把映射 $S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ 转换为 $S \times \Sigma \to S$。NFA确定化比较复杂，下面一步步推导出算法思路。

如果两个状态由 ε 弧连在一起，如图3.12(a)中的状态1和状态2，这样的状态应合并成

一个状态。因为如果从初态 s_0 有 $\hat{\delta}(s_0,w)=\{s\},\delta(s,\varepsilon)=t$，则有 $\hat{\delta}(s_0,w)=\hat{\delta}(s_0,w\varepsilon)$，因此有 $\delta(\hat{\delta}(s_0,w),\varepsilon)\in\hat{\delta}(s_0,w)$。而 $\delta(\hat{\delta}(s_0,w),\varepsilon)=\delta(s,\varepsilon)=t$，因此 $t\in\hat{\delta}(s_0,w)$。根据以上结论，①将 s 和 t 合并是自洽的，这种合并对到达这些状态时识别的字符串没有影响；②这种合并也是必须的，否则会导致 $\hat{\delta}(s_0,w)$ 的不确定性。

我们把这个情况拓广到集合的情况，如图3.12(b)所示。假设有集合 $I=\{s_1,s_2,\cdots,s_m\}$，现有集合 $J=\{t_j\mid(\exists s_i)\delta(s_i,\varepsilon)=t_j\}$，也就是 I 中元素通过 ε 弧到达的状态组成了集合 J。如果集合 I 中的状态能合并成一个状态，那么由于 $\delta(s_i,\varepsilon)=t_j$ 确定的 s_i 和 t_j 也要合并，因此 I 和 J 也能合并，即 $I\cup J$ 能合并。我们把 $I\cup J$ 定义为集合 I 的 ε-闭包。



(a) 有空符连接的状态需要合并　　(b) ε-闭包

图 3.12　ε 弧到达

定义 3.6 (集合 I 的 ε-闭包)

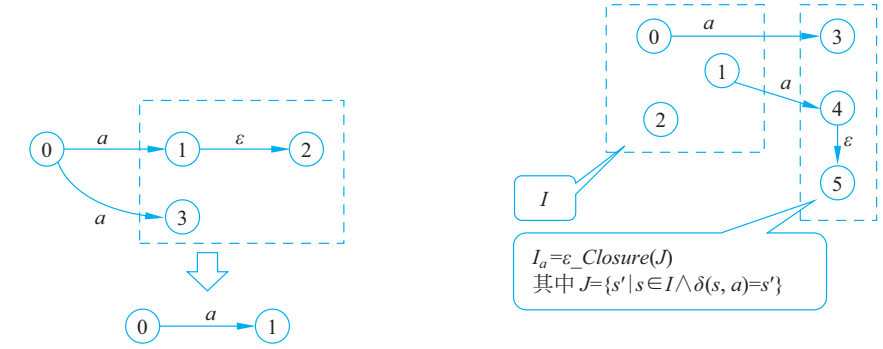
集合 I 的 ε-闭包，记作 $\varepsilon\text{-}Closure(I)$：

(1) 若 $q\in I$，则 $q\in\varepsilon\text{-}Closure(I)$；

(2) 若 $q\in I,\delta(q,\varepsilon)=q'$，则 $q'\in\varepsilon\text{-}Closure(I)$。

$\varepsilon\text{-}Closure(I)$ 即 I 自身及其通过 ε 弧到达的状态的集合。如果 I 中的状态能合并，那么 $\varepsilon\text{-}Closure(I)$ 状态就能合并。剩下的问题就是找出能合并的那些状态集 I。

如图3.13(a)所示，由同一个状态0，通过 a 弧到达状态1和3应该合并为一个状态。首先，如果这两个状态不合并，就无法消除不确定性。其次，合并后接受的符号串无影响。假设 $\delta(s_i,a)=t_1,\delta(s_i,a)=t_2,\hat{\delta}(t_1,w_1)=\{Y\},\hat{\delta}(t_2,w_2)=\{Y\}$，我们将 t_1 和 t_2 合并为 t，只要保持 $\delta(s_i,a)=t,\hat{\delta}(t,w_1)=\{Y\},\hat{\delta}(t,w_2)=\{Y\}$ 即可。再结合 ε-闭包的结论，如果 t_1,t_2 能合并，那么 $\varepsilon\text{-}Closure(\{t_1,t_2\})$ 也能合并。



(a) 同一状态同一符号到达的状态需合并　　(b) I_a 转移闭包

图 3.13　a 弧到达

由于目前我们的NFA有唯一初态X，所以可合并状态$\varepsilon\text{-}Closure(\{X\})$也是唯一的。我们把这个集合看作一个状态，又可以通过a弧转移找到下一组可以合并的状态集。以此类推，就可以找到所有可合并状态。先定义集合I的a转移。

定义 3.7 (集合I的a转移)

$I_a = Go(I,a) = \varepsilon\text{-}Closure(J)$，其中 $J = \{s' \mid s \in I \wedge \delta(s,a) = s'\}$。

这个定义中，J是I的元素通过a弧能直接到达的状态集合，I_a则是J再加上J中元素通过ε弧能到达的元素。最后的NFA确定化过程如算法3.3所示。

算法 3.3 NFA确定化

输入： NFA $M = (S, \{a_1, a_2, \cdots, a_k\}, \delta, \{X\}, \{Y\})$，其中映射$\delta : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$
输出： 等价的DFA M'，其中映射$\delta' : S \times \Sigma \to S$

```
DFA determineNFA(M):
    构造具有 k+1 列的表，记作第 0,1,2,···,k 列;
    首行首列（第0列）置为 ε-Closure({X});
    do
        如果某一行的第0列已确定，记为 I，则该行第 i 列填入 I_{a_i};
        检查该行上的所有状态子集 I_{a_i}，如果未出现在第0列，则填充到后面空行的第0列;
    while 所有行都计算完毕;
    每个状态子集视为新的状态，首行首列为初态，包含原终态 Y 的状态子集为新终态，得到DFA M';
    return M';
end determineNFA
```

4. 确定化示例

NFA确定化示例

例题 3.9NFA确定化 构造正规式$(a|b)^*(aa|bb)(a|b)^*$的DFA。

解 【步骤一】构造NFA，使其初态和终态唯一。由于给出的是正规式，因此可以直接给出初态X和终态Y，将正规式放到X到Y的箭弧，如图3.14所示。

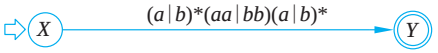


图 3.14 构造初态和终态唯一的NFA

【步骤二】分裂，使箭弧上标记为$\Sigma \cup \{\varepsilon\}$，如图3.15所示。

- 分裂图3.14中的第一个连接运算，得到图3.15(a)。
- 分裂图3.15(a)的$0 \xrightarrow{(aa|bb)(a|b)^*} Y$上的连接运算，得到图3.15(b)。
- 分裂图3.15(b)的$X \xrightarrow{(a|b)^*} 0$的闭包、$0 \xrightarrow{aa|bb} 1$的或、$1 \xrightarrow{(a|b)^*} Y$的闭包运算，得到图3.15(c)。
- 分裂图3.15(c)的$2 \xrightarrow{a|b} 2$的或、$0 \xrightarrow{aa} 1$的连接、$0 \xrightarrow{bb} 1$的连接、$3 \xrightarrow{a|b} 3$的或运算，

得到图3.15(d)。

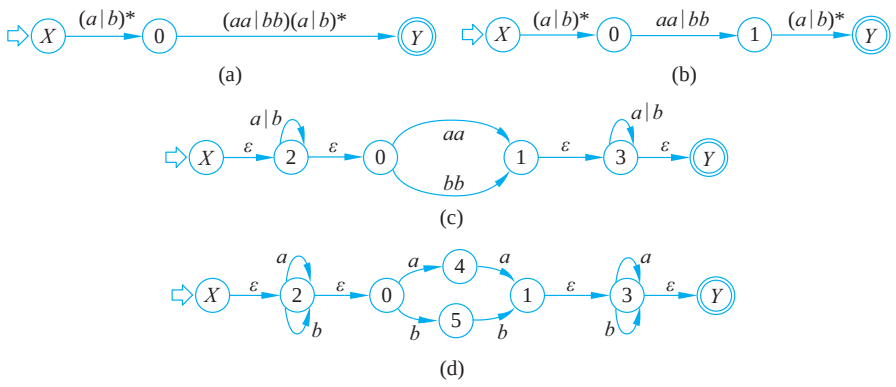


图 3.15　NFA 分裂过程

【步骤三】确定化

利用算法3.3，计算NFA可合并状态子集如表3.5所示，以下是各步骤的解释。

- 首行首列：此处为$\varepsilon\text{-}Closure(\{X\})$，首先$X \in \varepsilon\text{-}Closure(\{X\})$，由$\delta(X,\varepsilon)=2$得到$2 \in \varepsilon\text{-}Closure(\{X\})$，再由$\delta(2,\varepsilon)=0$得到$0 \in \varepsilon\text{-}Closure(\{X\})$，最终得到$\varepsilon\text{-}Closure(\{X\})=\{X,0,2\}$。
- 第0行：$I=\{X,0,2\}$
  - 第1列I_a：由$\delta(2,a)=2,\delta(0,a)=4$，得到$\{2,4\}\subseteq I_a$；再由$\delta(2,\varepsilon)=0$，得到$0 \in I_a$；最终得到$I_a=\{0,2,4\}$。$I_a$未出现在第0列，因此填入第1行第0列。
  - 第2列I_b：由$\delta(2,b)=2,\delta(0,b)=5,\delta(2,\varepsilon)=0$，得到$I_b=\{0,2,5\}$。$I_b$未出现在第0列，因此填入第2行第0列。
- 第1行：$I=\{0,2,4\}$
  - 第1列I_a：由$\delta(2,a)=2,\delta(0,a)=4,\delta(4,a)=1$，再由$\delta(2,\varepsilon)=0,\delta(1,\varepsilon)=3,\delta(3,\varepsilon)=Y$，得到$I_a=\{0,1,2,3,4,Y\}$。$I_a$未出现在第0列，因此填入第3行第0列。
  - 第2列I_b：由$\delta(2,b)=2,\delta(0,b)=5,\delta(2,\varepsilon)=0$，得到$I_b=\{0,2,5\}$。$I_b$已出现在第0列，因此不再填入。
- 第2行：$I=\{0,2,5\}$
  - 第1列I_a：由$\delta(0,a)=4,\delta(2,a)=2$，以及$\delta(2,\varepsilon)=0$，有$I_a=\{0,2,4\}$。$I_a$已在第0列出现，因此不再填入。
  - 第2列I_b：由$\delta(0,b)=5,\delta(2,b)=2,\delta(5,b)=1$，以及$\delta(1,\varepsilon)=3,\delta(3,\varepsilon)=Y,\delta(2,\varepsilon)=0$，有$I_b=\{0,1,2,3,5,Y\}$。$I_b$未在第0列出现，因此$I_b$填入第4行第0列。
- 第3行：$I=\{0,1,2,3,4,Y\}$
  - 第1列I_a：由$\delta(0,a)=4,\delta(2,a)=2,\delta(3,a)=3,\delta(4,a)=1$，以及$\delta(1,\varepsilon)=3,\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有$I_a=\{0,1,2,3,4,Y\}$。$I_a$已在第0列出现，因此不再填入。
  - 第2列I_b：由$\delta(0,b)=5,\delta(2,b)=2,\delta(3,b)=3$，以及$\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有$I_b=\{0,2,3,5,Y\}$。$I_b$未出现在第0列，因此$I_b$填入第5行第0列。

- 第 4 行：$I=\{0,1,2,3,5,Y\}$
  - 第 1 列 I_a：由 $\delta(0,a)=4,\delta(2,a)=2,\delta(3,a)=3$，以及 $\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有 $I_a=\{0,2,3,4,Y\}$。I_a 未出现在第 0 列，因此 I_a 填入第 6 行第 0 列。
  - 第 2 列 I_b：由 $\delta(0,b)=5,\delta(2,b)=2,\delta(3,b)=3,\delta(5,b)=1$，以及 $\delta(1,\varepsilon)=3,\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有 $I_b=\{0,1,2,3,5,Y\}$。I_b 已在第 0 列出现，因此不再填入。
- 第 5 行：$I=\{0,2,3,5,Y\}$
  - 第 1 列 I_a：由 $\delta(0,a)=4,\delta(2,a)=2,\delta(3,a)=3$，以及 $\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有 $I_a=\{0,2,3,4,Y\}$。I_a 已在第 0 列出现，因此不再填入。
  - 第 2 列 I_b：由 $\delta(0,b)=5,\delta(2,b)=2,\delta(3,b)=3,\delta(5,b)=1$，以及 $\delta(1,\varepsilon)=3,\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有 $I_b=\{0,1,2,3,5,Y\}$。I_b 已在第 0 列出现，因此不再填入。
- 第 6 行：$I=\{0,2,3,4,Y\}$
  - 第 1 列 I_a：由 $\delta(0,a)=4,\delta(2,a)=2,\delta(3,a)=3,\delta(4,a)=1$，以及 $\delta(1,\varepsilon)=3,\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有 $I_a=\{0,1,2,3,4,Y\}$。I_a 已在第 0 列出现，因此不再填入。
  - 第 2 列 I_b：由 $\delta(0,b)=5,\delta(2,b)=2,\delta(3,b)=3$，以及 $\delta(2,\varepsilon)=0,\delta(3,\varepsilon)=Y$，有 $I_b=\{0,2,3,5,Y\}$。I_b 已在第 0 列出现，因此不再填入。

表 3.5 NFA 可合并状态子集

行号/列号	0	1	2
	I	I_a	I_b
0	$\{X,0,2\}$	$\{0,2,4\}$	$\{0,2,5\}$
1	$\{0,2,4\}$	$\{0,1,2,3,4,Y\}$	$\{0,2,5\}$
2	$\{0,2,5\}$	$\{0,2,4\}$	$\{0,1,2,3,5,Y\}$
3	$\{0,1,2,3,4,Y\}$	$\{0,1,2,3,4,Y\}$	$\{0,2,3,5,Y\}$
4	$\{0,1,2,3,5,Y\}$	$\{0,2,3,4,Y\}$	$\{0,1,2,3,5,Y\}$
5	$\{0,2,3,5,Y\}$	$\{0,2,3,4,Y\}$	$\{0,1,2,3,5,Y\}$
6	$\{0,2,3,4,Y\}$	$\{0,1,2,3,4,Y\}$	$\{0,2,3,5,Y\}$

然后用行号作为状态编号，替换表中的状态子集，得到DFA状态转移矩阵，如表3.6所示。

表 3.6 DFA 状态转移矩阵

I	I_a	I_b
0	1	2
1	3	2
2	1	4
3	3	5
4	6	4

续表

I	I_a	I_b
5	6	4
6	3	5

画出状态转换图，首行首列状态0为初态，含有原终态Y的状态3、4、5、6为终态，得到图3.16所示的DFA。

例题3.10 NFA确定化 例题2.24设计了能接受偶数个0偶数个1的正规式((01|10)(00|11)*(01|10)|00|11)*，试求其DFA。

解 【步骤一】构造NFA，使其初态和终态唯一，如图3.17所示。

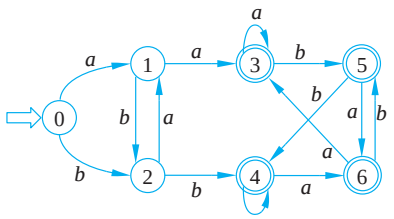


图 3.16 确定化后的DFA

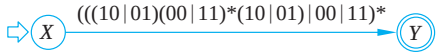


图 3.17 构造初态和终态唯一的NFA

【步骤二】分裂，使边上标记为$\Sigma \cup \{\varepsilon\}$，如图3.18所示。

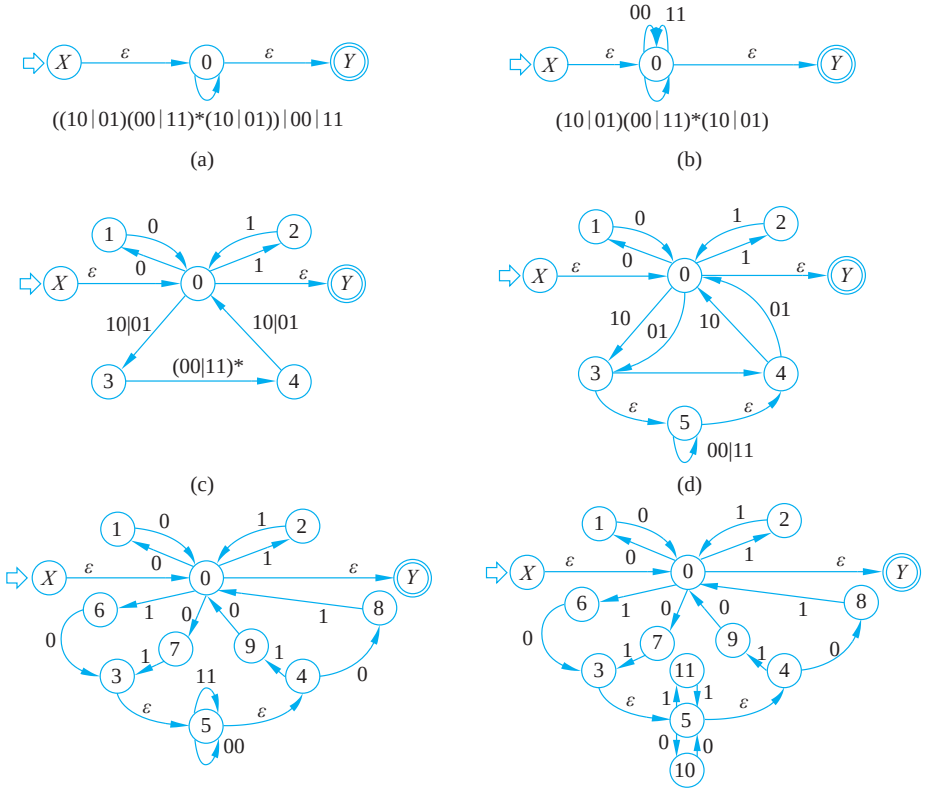


图 3.18 NFA分裂过程

- 分裂图3.17中的闭包运算，得到图3.18(a)。

- 分裂图3.18(a)的0 $\xrightarrow{((10|01)(00|11)^*(10|01))|00|11}$ 0上的或运算，得到图3.18(b)。
- 分裂图3.18(b)的0 $\xrightarrow{00}$ 0的连接、0 $\xrightarrow{11}$ 0的连接、0 $\xrightarrow{(10|01)(00|11)^*(10|01)}$ 0的连接运算，得到图3.18(c)。
- 分裂图3.18(b)的0 $\xrightarrow{10|01}$ 3的或、4 $\xrightarrow{10|01}$ 0的或、3 $\xrightarrow{(00|11)^*}$ 4的闭包运算，得到图3.18(d)。
- 分裂图3.18(d)的0 $\xrightarrow{10}$ 3的连接、0 $\xrightarrow{01}$ 3的连接、4 $\xrightarrow{10}$ 0的连接、4 $\xrightarrow{01}$ 0的连接、5 $\xrightarrow{00|11}$ 5的或运算，得到图3.18(e)。
- 分裂图3.18(e)的5 $\xrightarrow{00}$ 5的连接、5 $\xrightarrow{11}$ 5的连接运算，得到图3.18(f)。

【步骤三】确定化，按照前述步骤逐步填写状态子集表，如表3.7所示。

表 3.7 NFA 可合并状态子集

行号/列号	0	1	2
	I	I_0	I_1
0	$\{X,0,Y\}$	$\{1,7\}$	$\{2,6\}$
1	$\{1,7\}$	$\{0,Y\}$	$\{3,4,5\}$
2	$\{2,6\}$	$\{3,4,5\}$	$\{0,Y\}$
3	$\{0,Y\}$	$\{1,7\}$	$\{2,6\}$
4	$\{3,4,5\}$	$\{8,10\}$	$\{9,11\}$
5	$\{8,10\}$	$\{4,5\}$	$\{0,Y\}$
6	$\{9,11\}$	$\{0,Y\}$	$\{4,5\}$
7	$\{4,5\}$	$\{8,10\}$	$\{9,11\}$

- 首行首列：由$\delta(X,\varepsilon)=0,\delta(0,\varepsilon)=Y$，得到$\varepsilon\text{-}Closure(\{X\})=\{X,0,Y\}$，填入第1行第0列。
- 第0行：$I=\{X,0,Y\}$
  - 第1列I_0：由$\delta(0,0)=1,\delta(0,0)=7$，有$I_0=\{1,7\}$，填入第1行第0列。
  - 第2列I_1：由$\delta(0,1)=2,\delta(0,1)=6$，有$I_1=\{2,6\}$，填入第2行第0列。
- 第1行：$I=\{1,7\}$
  - 第1列I_0：由$\delta(1,0)=0,\delta(0,\varepsilon)=Y$，有$I_0=\{0,Y\}$，填入第3行第0列。
  - 第2列I_1：由$\delta(7,1)=3,\delta(3,\varepsilon)=5,\delta(5,\varepsilon)=4$，有$I_1=\{3,4,5\}$，填入第4行第0列。
- 第2行：$I=\{2,6\}$
  - 第1列I_0：由$\delta(6,0)=3,\delta(3,\varepsilon)=5,\delta(5,\varepsilon)=4$，有$I_0=\{3,4,5\}$，已在第4行第0列出现。
  - 第2列I_1：由$\delta(2,1)=0,\delta(0,\varepsilon)=Y$，有$I_1=\{0,Y\}$，已在第3行第0列出现。
- 第3行：$I=\{0,Y\}$
  - 第1列I_0：由$\delta(0,0)=1,\delta(0,0)=7$，有$I_0=\{1,7\}$，已在第1行第0列出现。
  - 第2列I_1：由$\delta(0,1)=2,\delta(0,1)=6$，有$I_1=\{2,6\}$，已在第2行第0列出现。
- 第4行：$I=\{3,4,5\}$
  - 第1列I_0：由$\delta(4,0)=8,\delta(5,0)=10$，有$I_0=\{8,10\}$，填入第5行第0列。

– 第2列I_1：由$\delta(4,1)=9,\delta(5,1)=11$，有$I_1=\{9,11\}$，填入第6行第0列。

- 第5行：$I=\{8,10\}$
  – 第1列I_0：由$\delta(10,0)=5,\delta(5,\varepsilon)=4$，有$I_0=\{4,5\}$，填入第7行第0列。
  – 第2列I_1：由$\delta(8,1)=0,\delta(0,\varepsilon)=Y$，有$I_1=\{0,Y\}$，已在第3行第0列出现。
- 第6行：$I=\{9,11\}$
  – 第1列I_0：由$\delta(9,0)=0,\delta(0,\varepsilon)=Y$，有$I_0=\{0,Y\}$，已在第3行第0列出现。
  – 第2列I_1：由$\delta(11,1)=5,\delta(5,\varepsilon)=4$，有$I_1=\{4,5\}$，已在第7行第0列出现。
- 第7行：$I=\{4,5\}$
  – 第1列I_0：由$\delta(4,0)=8,\delta(5,0)=10$，有$I_0=\{8,10\}$，已在第5行第0列出现。
  – 第2列I_1：由$\delta(4,1)=9,\delta(5,1)=11$，有$I_1=\{9,11\}$，已在第6行第0列出现。

用行号作为状态编号，替换表中的状态子集，得到DFA状态转移矩阵，如表3.8所示。

表 3.8　DFA状态转移矩阵

I	I_0	I_1
0	1	2
1	3	4
2	4	3
3	1	2
4	5	6
5	7	3
6	3	7
7	5	6

画出状态转换图，首行首列状态0为初态，含有原终态Y的状态0、3为终态，得到图3.19所示的DFA。

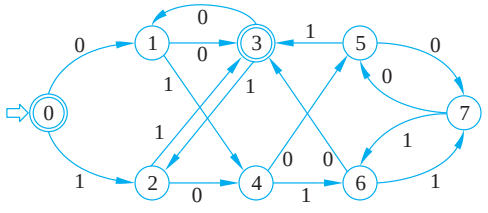


图 3.19　确定化后的DFA

例题3.9得到的图3.16的DFA，与图3.8中的DFA，都接受aa或bb的字符串，但例题3.9得到的DFA复杂很多。同样，例题3.10得到的图3.19的DFA，与图3.5中的DFA，都接受偶数个0偶数个1的字符串，但例题3.10得到的DFA也复杂很多。这就引出了DFA的化简问题。

3.2.4　确定有限自动机化简

DFA化简

DFA M 化简指寻找一个状态数比 M 少的DFA M'，使得 $L(M)=L(M')$；最终目标是找到状态数最少那个 M'。

减少状态的方法就是合并一些状态。状态合并有两个思路：

- 以终态为基准，如果两个状态到达终态识别的字符串相同，就把它们合并起来，如图3.20(a)所示。
- 以初态为基准，如果初态到达两个状态识别的字符串相同，就把它们合并起来，如图3.20(b)所示。

(a) 以终态为基准寻找可合并状态

(b) 以初态为基准寻找可合并状态

图 3.20 DFA 合并状态

这两个条件，满足其中之一即可，我们把可以合并的状态定义为等价状态。但是后续分析会看到，无遗漏地寻找所有等价状态是比较困难的，但是寻找不等价状态相对来说简单。而终态与其他状态有明显区别，所以我们选择以终态为基准定义等价状态。

定义 3.8 (等价状态与可区别状态)

如果状态 s 和 t 同时满足以下条件，就称它们是等价状态：

- 如果从 s 出发能读出某个字 w 停在终态，那么从 t 出发也能读出字 w 停在终态；
- 如果从 t 出发能读出某个字 w 停在终态，那么从 s 出发也能读出字 w 停在终态。

如果状态 s 和 t 不是等价状态，则称它们是可区别的。 ♣

定理 3.3 (状态等价传递定理)

若 $\delta(u,a)=s, \delta(v,a)=t$，且 s,t 等价，则 u,v 等价。 ♡

证明 如图3.21所示，我们把所有终态都看作 Y。

- 若有 $\hat{\delta}(u,aw)=\{Y\}$，则 $\hat{\delta}(s,w)=\{Y\}$；因为 s,t 等价，故 $\hat{\delta}(t,w)=\{Y\}$，因此 $\hat{\delta}(v,aw)=\{Y\}$。
- 若有 $\hat{\delta}(v,aw)=\{Y\}$，则 $\hat{\delta}(t,w)=\{Y\}$；因为 s,t 等价，故 $\hat{\delta}(s,w)=\{Y\}$，因此 $\hat{\delta}(s,aw)=\{Y\}$。

根据等价状态的定义知 u,v 等价。

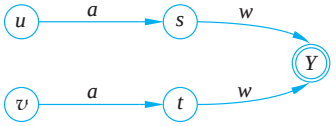


图 3.21 等价状态

用这种方法找等价状态比较困难，但是寻找可区别状态却是容易的。首先，终态和非终态是可区别的，因为终态可以接受空串，但非终态不能接受空串。所以，一开始我们可以把终态和非终态进行区分，形成两个初始子集。每个时刻的每个子集内部的状态可能等价，也可能不等价，但两个不同子集肯定是可区别的。

检查每个子集，看里面的状态是否可区别，如果可区别，就再划分成不同的子集。对一个子集的状态u和v，如果$\delta(u,a)=s$，考虑到DFA中状态转移是唯一确定的，则v有如下两种情况可与u区别：

- $\delta(v,a)=t$，s和t属于不同子集，则u,v可区别。这是因为s和t可区别，必然存在一个字w或者s可接受t不可接受，或者t可接受s不可接受。那么，对字aw，或者u可接受v不可接受，或者v可接受u不可接受，因此u和v可区别。
- 若$\delta(v,a)$不存别，则u,v可区别。这是因为u可接受以a为首的字，而v不可以。

根据以上分析，如果一个子集I中的某些状态，通过某个a弧引入某个子集J，而I中的另一些状态没有a弧引入子集J，那么这两部分状态就是可区别的，可以划分为不同子集，这样就得到了DFA化简的状态子集划分方法，如算法3.4所示。

算法 3.4 DFA化简的状态子集划分算法

输入： DFA $M=(S,\Sigma,\delta,S_0,F)$

输出： DFA M'，M'是使得$L(M)=L(M')$成立的状态数最少的DFA

```
1  Π = {S − F, F};
2  do
3      foreach I^(i) ∈ (Π = {I^(1), I^(2), ··· , I^(m)}) do
4          if (∃a ∈ Σ ∧ ∃s1 ∈ I^(i) ∧ ∃s2 ∈ I^(i)) 使得(δ(s1, a) ∈ I^(j) ∧ δ(s2, a) ∉ I^(j))
           then
5              I^(i1) = {s|s ∈ I^(i) ∧ δ(s, a) ∈ I^(j)};
6              I^(i2) = I^(i) − I^(i1);
7              Π − = {I^(i)};
8              Π ∪ = {I^(i1), I^(i2)};
9          end
10     end
11 while Π 不能再划分;
12 合并Π中的状态子集I^(i);
13 含有原初态的状态子集为新初态，含有原终态的状态子集为新终态;
```

算法第1行将状态集S划分为终态集F和非终态集$S-F$两个子集，形成初次划分Π。第2~11行的do···while对Π的每个子集不断划分，直到不能再划分为止（也就是Π内的集合数量不再增加）。

第3~10行的foreach对Π的每个子集进行划分。第4行的if语句条件中，$I^{(i)}$中存在两个状态s_1和s_2，其中一个通过a弧射入某个集合$I^{(j)}$，而另一个状态通过a弧没有射入该集合（包括没有a弧的情况）。这时就可以把$I^{(i)}$一分为二，第5行为$I^{(i)}$中有a弧射入$I^{(j)}$的状态子集，构成$I^{(i1)}$；第6行为$I^{(i)}$去掉$I^{(i1)}$部分后剩下的子集，作为$I^{(i2)}$。第7行从Π中去掉原来的$I^{(i)}$，第8行把$I^{(i)}$分裂成的$I^{(i1)}$和$I^{(i2)}$并入Π。

划分完成后，把每个集合$I^{(i)}$中的所有状态合并成一个状态（第12行），最后确定新初态和新终态（第13行）。

例题 3.11DFA 化简 化简图3.16所示的DFA。

解 (1) 初次划分，将终态和非终态分开：$\Pi_0 = \{\{0,1,2\},\{3,4,5,6\}\}$

(2) 考察子集$\{0,1,2\}$：$\delta(0,a) = 1 \in \{0,1,2\}, \delta(1,a) = 3 \in \{3,4,5,6\}, \delta(2,a) = 1 \in \{0,1,2\}$

根据达到子集不同进行划分：$\Pi_1 = \{\{0,2\},\{1\},\{3,4,5,6\}\}$

(3) 考察子集$\{0,2\}$：$\delta(0,b) = 2 \in \{0,2\}, \delta(2,b) = 4 \in \{3,4,5,6\}$

根据达到子集不同进行划分：$\Pi_2 = \{\{0\},\{2\},\{1\},\{3,4,5,6\}\}$

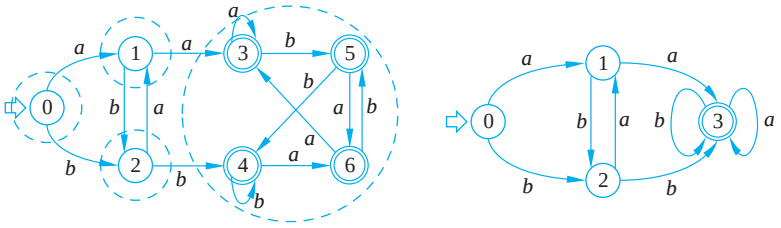
(4) 考察子集$\{3,4,5,6\}$：

$\delta(3,a) = 3 \in \{3,4,5,6\}, \delta(4,a) = 6 \in \{3,4,5,6\}, \delta(5,a) = 6 \in \{3,4,5,6\}, \delta(6,a) = 3 \in \{3,4,5,6\}$,

$\delta(3,b) = 5 \in \{3,4,5,6\}, \delta(4,b) = 4 \in \{3,4,5,6\}, \delta(5,b) = 4 \in \{3,4,5,6\}, \delta(6,b) = 5 \in \{3,4,5,6\}$

不能再划分，因此最终划分为$\Pi = \{\{0\},\{2\},\{1\},\{3,4,5,6\}\}$

将Π中状态子集合并，如图3.22(a)中虚线框所示。合并后如图3.22(b)所示，与图3.4完全一致。



(a) 合并状态子集　　(b) 合并后得到最简DFA

图 3.22　DFA 化简

关于状态子集合并后画有向边的原则，对Π的任意两个状态子集$I^{(i)} = \{s_1^{(i)}, s_2^{(i)}, \cdots, s_m^{(i)}\}$和$I^{(j)} = \{s_1^{(j)}, s_2^{(j)}, \cdots, s_n^{(j)}\}$，这里仍然用$I^{(i)}$和$I^{(j)}$表示合并后的状态：

- 如果所有$s_u^{(i)} \in I^{(i)}$到部分或全部$s_v^{(j)} \in I^{(j)}$有a弧，则有$\delta(I^{(i)}, a) = I^{(j)}$（有$a$弧）；
- 如果所有$s_u^{(i)} \in I^{(i)}$到全部$s_v^{(j)} \in I^{(j)}$没有a弧，则$\delta(I^{(i)}, a) \neq I^{(j)}$（没有$a$弧）；
- 如果部分$s_u^{(i)} \in I^{(i)}$到部分或全部$s_v^{(j)} \in I^{(j)}$有a弧，部分$s_u^{(i)} \in I^{(i)}$没有a弧，则说明$I^{(i)}$的状态是可区别的，还可以再划分。

以上方法对$i = j$的情况也成立。

例题 3.12 DFA 化简 化简图3.19所示的DFA。

解 (1) 初次划分，将终态和非终态分开：$\Pi_0 = \{\{1,2,4,5,6,7\},\{0,3\}\}$

(2) 考察子集$\{1,2,4,5,6,7\}$：

$\delta(1,0) = 3 \in \{0,3\}, \delta(2,0) = 4 \in \{1,2,4,5,6,7\}, \delta(4,0) = 5 \in \{1,2,4,5,6,7\}$,

$\delta(5,0) = 7 \in \{1,2,4,5,6,7\}, \delta(6,0) = 3 \in \{0,3\}, \delta(7,0) = 5 \in \{1,2,4,5,6,7\}$,

根据达到子集不同进行划分：$\Pi_1 = \{\{2,4,5,7\},\{1,6\},\{0,3\}\}$

(3) 考察子集$\{2,4,5,7\}$：

$\delta(2,1) = 3 \in \{0,3\}, \delta(4,1) = 6 \in \{1,6\}$, $\delta(5,1) = 3 \in \{0,3\}, \delta(7,1) = 6 \in \{1,6\}$,

根据达到子集不同进行划分：$\Pi_2 = \{\{2,5\},\{4,7\},\{1,6\},\{0,3\}\}$

(4) 考察子集$\{2,5\}$：

$\delta(2,0)=4\in\{4,7\},\delta(5,0)=7\in\{4,7\},\delta(2,1)=3\in\{0,3\},\delta(5,1)=3\in\{0,3\}$，不能再划分。

(5) 考察子集$\{4,7\}$：

$\delta(4,0)=5\in\{2,5\},\delta(7,0)=5\in\{2,5\},\delta(4,1)=6\in\{1,6\},\delta(7,1)=6\in\{1,6\}$，不能再划分。

(6) 考察子集$\{1,6\}$：

$\delta(1,0)=3\in\{0,3\},\delta(6,0)=3\in\{0,3\},\delta(1,1)=4\in\{4,7\},\delta(6,1)=7\in\{4,7\}$，不能再划分。

(7) 考察子集$\{0,3\}$：

$\delta(0,0)=1\in\{1,6\},\delta(3,0)=1\in\{1,6\},\delta(0,1)=2\in\{2,5\},\delta(3,1)=2\in\{2,5\}$，不能再划分。

综上，得到最终划分为$\Pi=\{\{2,5\},\{4,7\},\{1,6\},\{0,3\}\}$

根据最终划分Π及图3.19中的状态转移，得到状态转移矩阵，如表3.9所示。根据表3.9画出状态转换图，与图3.5完全相同。

表 3.9　合并状态后的状态转换表

状 态 子 集	新状态编号 I	I_0	I_1
$\{0,3\}$	0	1	2
$\{1,6\}$	1	0	3
$\{2,5\}$	2	3	0
$\{4,7\}$	3	2	1

3.2.5　正规式与有限自动机的等价性

我们已经知道DFA和NFA在接受语言的能力上是等价的，把DFA和NFA合称FA（有限自动机），它们与正规式也是等价的。

定理 3.4 (正规式与FA的等价性)

- 对任何FA M，都存在一个正规式r，使得$L(r)=L(M)$；
- 对任何正规式r，都存在一个FA M，使得$L(M)=L(r)$。

♡

证明　对任何FA M，构造正规式r，使得$L(r)=L(M)$。

首先使初态和终态唯一。增加新的初态X和终态Y，由X向所有原初态引ε弧，由原终态向Y引ε弧。显然，对新得到的NFA M'，有$L(M')=L(M)$。

然后，按图3.23的规则，逐步合并有向边，直至只剩下状态X、Y，以及从X到Y的一条有向边，则有向边上的表达式即为所求r。

证明过程中的方法虽然直观，但是并不能直接用于从FA到正规式的构造，因为FA中的有向边往往带有循环，难以呈现图3.23所示的清晰结构。如果有兴趣，可以试一下图3.4，会发现很难构造出$(a|b)^*(aa|bb)(a|b)^*$这样的正规式。FA转换为正规式，一般采用3.3节介绍的正规文法过渡。

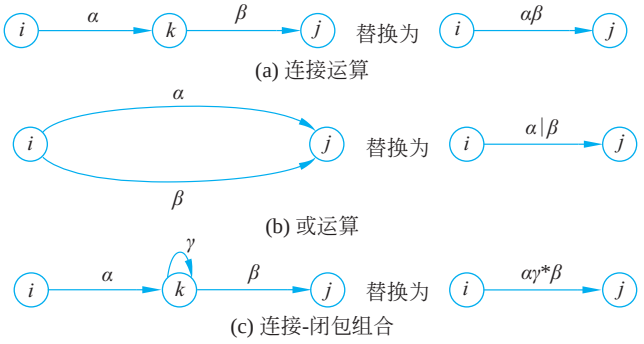


图 3.23 有向边合并规则

证明 对任何正规式r，构造NFA M，使得$L(M)=L(r)$。

使用r中运算符的数目（或、连接、闭包）的数学归纳法证明。

(1) r中有0个运算符，则有$r=\varepsilon, r=\varnothing, r=a$ 3种情况，其中$a\in\Sigma$，如图3.24所示构造3个NFA，显然满足要求。

图 3.24 有0个运算符的正规式构造NFA

(2) 假设当r的运算符少于k $(k\geqslant 1)$个时，当r有k个运算符时有3种情况。

情况1：$r=r_1|r_2$，由归纳假设，对$r_i, \exists M_i=(S_i,\Sigma_i,\delta_i,\{q_i\},\{f_i\})$，使得$L(M_i)=L(r_i)$，其中$i=1,2$。

如图3.25(a)，新增两个状态$q_0\notin S_1\cup S_2, f_0\notin S_1\cup S_2$，构造NFA $M=\{S_1\cup S_2\cup\{q_0,f_0\},\Sigma_1\cup\Sigma_2,\delta,\{q_0\},\{f_0\}\}$，其中$\delta$为：

- $\delta=\delta_1\cup\delta_2$，该步将原$M_1$和$M_2$的有向边加入$\delta$；
- 增加有向边$\delta(q_0,\varepsilon)=q_1, \delta(q_0,\varepsilon)=q_2$，该步从$q_0$向原初态引$\varepsilon$弧；
- 增加有向边$\delta(f_1,\varepsilon)=f_0, \delta(f_2,\varepsilon)=f_0$，该步从原终态向$f_0$引$\varepsilon$弧。

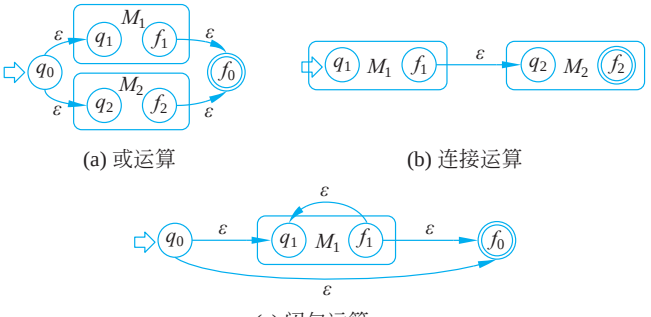


图 3.25 有k个运算符的正规式构造NFA

情况2：$r=r_1r_2$，由归纳假设，对$r_i, \exists M_i=(S_i,\Sigma_i,\delta_i,\{q_i\},\{f_i\})$，使得$L(M_i)=L(r_i)$，其中$i=1,2$。

如图3.25(b)，构造NFA $M=\{S_1\cup S_2,\Sigma_1\cup\Sigma_2,\delta,\{q_1\},\{f_2\}\}$，其中$\delta$为：

- $\delta=\delta_1\cup\delta_2$，该步将原$M_1$和$M_2$的有向边加入$\delta$；
- 增加有向边$\delta(f_1,\varepsilon)=q_2$，该步从$M_1$原终态向$M_2$原初态引$\varepsilon$弧。

情况3：$r = r_1^*$，由归纳假设，对$r_1, \exists M_1 = (S_1, \Sigma_1, \delta_1, \{q_1\}, \{f_1\})$，使得$L(M_1) = L(r_i)$。

如图3.25(c)，新增两个状态$q_0 \notin S_1 \cup S_2, f_0 \notin S_1 \cup S_2$，构造NFA $M = \{S_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, \{q_0\}, \{f_0\}\}$，其中$\delta$为：

- $\delta = \delta_1$，该步将原M_1的有向边加入δ；
- 增加有向边$\delta(q_0, \varepsilon) = q_1, \delta(q_0, \varepsilon) = f_0, \delta(f_1, \varepsilon) = f_0, \delta(f_1, \varepsilon) = q_1$，该步构造闭包运算。

3.3 正规文法

正规文法又称3型文法，定义2.15已给出其定义，它有两种形式：

- 右线性正规文法：产生式形式为$A \to \alpha B$或$A \to \beta$；
- 左线性正规文法：产生式形式为$A \to B\alpha$或$A \to \beta$；

其中$\alpha \in V_T, \beta \in V_T \cup \{\varepsilon\}, A \in V_N, B \in V_N$。

定理 3.5 (正规文法与FA的等价性)

- 对每一个正规文法G，都存在一个FA M，使得$L(M) = L(G)$；
- 对每一个FA M，都存在一个右线性文法G_R和一个左线性文法G_L，使得$L(G_R) = L(G_L) = L(M)$。

♡

由于正规文法分左、右线性文法，因此下面对其分别进行讨论。

3.3.1 右线性文法转有限自动机

为了导出右线性文法转有限自动机的算法，先看一个右线性文法的推导过程。

例题 3.13 右线性文法推导 文法$G[A]: A \to aA|bB|a, B \to aA$

推导句子$aabaa$：$A \Rightarrow aA \Rightarrow aaA \Rightarrow aabB \Rightarrow aabaA \Rightarrow aabaa$

从该例可以看出，右线性文法在推导时，只有最右边的符号是非终结符，那么只能使用以这个符号为左部的产生式。而且每用形如$A \to \alpha B$的产生式推导一步，相当于识别出一个符号α。如果用最右边符号作为状态，相当于使用$A \to \alpha B$一次，识别出一个α，状态由A转移到B。一旦使用$A \to \beta$这样的产生式，推导结束，进入终态，由此得到右线性文法转有限自动机的过程如算法3.5所示。

算法 3.5 右线性文法转有限自动机

输入： 右线性文法 $G_R = (V_N, V_T, P, S)$
输出： FA M，使得$L(M) = L(G_R)$

```
1 令 M = (V_N ∪ {f}, V_T, δ, {S}, {f})，其中 f ∉ V_N;
2 foreach p ∈ P do
3 |   if p形如 A → αB then δ(A, α) = B;
4 |   if p形如 A → β then δ(A, β) = f;
5 end
```

该算法将非终结符作为状态，开始符号为初态，并且增加了一个状态f作为终态。对形如$A \to \alpha B$的产生式，从A向B引α弧；对形如$A \to \beta$的产生式，从A向终态f引β弧。

例题 3.14 右线性文法转FA 文法$G[A]: A \to aA|bB|a, B \to aA$，构造与其等价的FA。

解 将非终结符 A, B 作为状态，文法开始符号 A 为初态，增加一个新的状态 f 作为终态，如图3.26所示。

- 对产生式 $A \to aA$，由 A 向 A 引 a 弧；
- 对产生式 $A \to bB$，由 A 向 B 引 b 弧；
- 对产生式 $A \to a$，由 A 向 f 引 a 弧；
- 对产生式 $B \to aA$，由 B 向 A 引 a 弧。

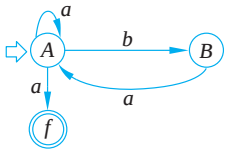


图 3.26 右线性文法转 FA

例题 3.15 右线性文法转 FA 文法 $G[S]: S \to aA|bB|\varepsilon, A \to aB|bA, B \to aS|bA|\varepsilon$，构造与其等价的FA。

解 将非终结符 S, A, B 作为状态，文法开始符号 S 为初态，增加一个新的状态 f 作为终态，如图3.27所示。

- 对产生式 $S \to aA$，由 S 向 A 引 a 弧；
- 对产生式 $S \to bB$，由 S 向 B 引 b 弧；
- 对产生式 $S \to \varepsilon$，由 S 向 f 引 ε 弧；
- 对产生式 $A \to aB$，由 A 向 B 引 a 弧；
- 对产生式 $A \to bA$，由 A 向 A 引 b 弧；
- 对产生式 $B \to aS$，由 B 向 S 引 a 弧；
- 对产生式 $B \to bA$，由 B 向 A 引 b 弧；
- 对产生式 $B \to \varepsilon$，由 B 向 f 引 ε 弧。

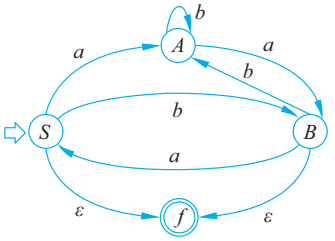


图 3.27 右线性文法转 FA

3.3.2 左线性文法转有限自动机

例题 3.16 左线性文法推导 文法 $G[Z]: Z \to U0|V1, U \to Z1|1, V \to Z0|0$

推导句子1010：$Z \Rightarrow U0 \Rightarrow Z10 \Rightarrow U010 \Rightarrow 1010$

左线性文法的推导过程中，最左边为非终结符，我们还是以非终结符作为状态。但从推导过程可以看出，左线性文法的终结符，是从右向左逐个出现的；但FA识别符号串是从左向右逐符号识别，两个过程的方向相反。该例中第一步使用 $Z \to U0$ 时，识别出最右边一个字符0，对FA来说应该是它识别的最后一个字符，因此应通过0弧到达终态；推导最后一步使用 $U \to 1$ 时，识别出最左边符号1，对FA来说是识别出第一个字符1，应从开始符号通过字符1到达状态 U。所以我们应该把左线性文法的推导过程反过来看，如果还以

推导过程的非终结符号作为状态，那么状态变换是按推导过程从右向左的，最终到达的状态是文法开始符号 Z。

算法3.6将左线性文法转换为FA。该算法将非终结符作为状态，开始符号为终态，并且增加了一个状态 s_0 作为初态。对形如 $A \to B\alpha$ 的产生式，从 B 向 A 引 α 弧；对形如 $A \to \beta$ 的产生式，从初态 s_0 向终态 A 引 β 弧。

算法 3.6 左线性文法转有限自动机

输入： 左线性文法 $G_L = (V_N, V_T, P, S)$
输出： FA M，使得 $L(M) = L(G_L)$

```
1 令 M = (V_N ∪ {s_0}, V_T, δ, {s_0}, {S})，其中 s_0 ∉ V_N;
2 foreach p ∈ P do
3 |   if p形如 A → Bα then δ(B, α) = A;
4 |   if p形如 A → β then δ(s_0, β) = A;
5 end
```

例题 3.17 左线性文法转 FA 文法 $G[Z]: Z \to U0|V1, U \to Z1|1, V \to Z0|0$，构造与其等价的FA。

解 将非终结符 Z, U, V 作为状态，文法开始符号 Z 为终态，并增加一个新的状态 s_0 作为初态，如图3.28所示。

- 对产生式 $Z \to U0$，由 U 向 Z 引 0 弧；
- 对产生式 $Z \to V1$，由 V 向 Z 引 1 弧；
- 对产生式 $U \to Z1$，由 Z 向 U 引 1 弧；
- 对产生式 $U \to 1$，由 s_0 向 U 引 1 弧；
- 对产生式 $V \to Z0$，由 Z 向 V 引 0 弧；
- 对产生式 $V \to 0$，由 s_0 向 V 引 0 弧。

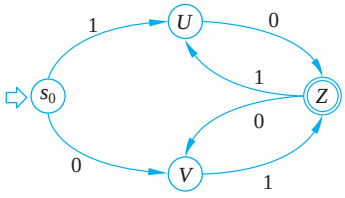


图 3.28 左线性文法转 FA

3.3.3 有限自动机转右线性文法

有限自动机转右线性文法是右线性文法转有限自动机的逆过程，如算法3.7所示。

算法 3.7 有限自动机转右线性文法

输入： FA $M = (S, \Sigma, \delta, \{S_0\}, F)$
输出： 右线性文法 G_R，使得 $L(G_R) = L(M)$

```
1 令 G_R = (S, Σ, P, S_0), P = ∅;
2 foreach (δ(S_i, a) = S_j) ∈ δ do
3 |   P ∪= {S_i → aS_j};
4 |   if S_j ∈ F then P ∪= {S_i → a};
5 end
6 if S_0 ∈ F then  P ∪= {S_0 → ε} ;
```

该算法的核心是三方面：

- 如果 S_i 到 S_j 有 a 弧，则增加产生式 $S_i \to aS_j$；
- 如果 S_j 是终态，再增加一个产生式 $S_i \to a$；
- 如果初态 S_0 同时又是终态，则增加一个空符产生式 $S_0 \to \varepsilon$。

例题 3.18FA 转右线性文法 将图3.29(a) 和图3.29(b) 所示的 FA 转为右线性文法 G_R。

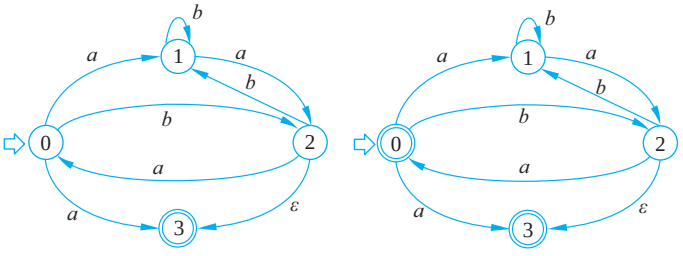


图 3.29 FA 转右线性文法

解 对图3.29(a) 的 FA，

- 由 $\delta(0,a)=1$ 得到 $S_0 \to aS_1$；
- 由 $\delta(0,a)=3$ 得到 $S_0 \to a|aS_3$；
- 由 $\delta(0,b)=2$ 得到 $S_0 \to bS_2$；
- 由 $\delta(1,a)=2$ 得到 $S_1 \to aS_2$；
- 由 $\delta(1,b)=1$ 得到 $S_1 \to bS_1$；
- 由 $\delta(2,a)=0$ 得到 $S_2 \to aS_0$；
- 由 $\delta(2,b)=1$ 得到 $S_2 \to bS_1$；
- 由 $\delta(2,\varepsilon)=3$ 得到 $S_2 \to \varepsilon|S_3$；

整理得 $G_R=(\{S_0,S_1,S_2\},\{a,b\},P,S_0)$，

其中 P：$S_0 \to aS_1|bS_2|aS_3|a, S_1 \to aS_2|bS_1, S_2 \to aS_0|bS_1|S_3|\varepsilon$。

求得的文法中出现了无用符号 S_3，以及单非产生式 $S_1 \to S_3$。利用 2.4.1 节消除无用符号和无用产生式的相关算法，得到：

$G_R=(\{S_0,S_1,S_2\},\{a,b\},P,S_0)$，其中 P：$S_0 \to aS_1|bS_2|a, S_1 \to aS_2|bS_1, S_2 \to aS_0|bS_1|\varepsilon$。

此时单非产生式也一并消除了，如果没有消除，还需要用 2.4.2 节的算法删除单非产生式。

对图3.29(b) 的 FA，与图3.29(a) 相比，初态 0 同时也是终态，所以增加一个空符产生式 $S_0 \to \varepsilon$，得到：

$G_R=(\{S_0,S_1,S_2\},\{a,b\},P,S_0)$，其中 P：$S_0 \to aS_1|bS_2|a|\varepsilon, S_1 \to aS_2|bS_1, S_2 \to aS_0|bS_1|\varepsilon$。

3.3.4 有限自动机转左线性文法

有限自动机转左线性文法是左线性文法转有限自动机的逆过程。由于 FA 的终态转换为左线性文法后为开始符号，为简便起见，应让终态唯一。如果 FA 终态不唯一，则增加新终态，由原终态向新终态引 ε 弧即可。这步操作不再赘述，因此我们只考虑终态唯一的情况。

算法3.8将终态唯一的FA转换为左线性文法，核心包括两方面：

- 如果S_i到S_j有a弧，则增加产生式$S_j \to S_i a$；
- 如果S_i是初态，再增加一个产生式$S_j \to a$；

算法 3.8 有限自动机转左线性文法

输入： FA $M=(S,\Sigma,\delta,\{S_0\},\{S_f\})$
输出： 左线性文法G_L，使得$L(G_L)=L(M)$

```
if S0有射入弧 then G_L=(S,Σ,P,S_f);
else G_L=(S-{S0},Σ,P,S_f);
P=∅;
foreach (δ(S_i,a)=S_j)∈δ do
    P∪={S_j→S_i a};
    if S_i=S0 then P∪={S_j→a};
end
```

例题 3.19 FA转左线性文法 将图3.29(a)所示的FA转为左线性文法G_L。

解 初态有射入弧，因此构造$G_L=(\{S_0,S_1,S_2,S_3\},\{a,b\},P,S_3)$

- 由$\delta(0,a)=1$得到$S_1 \to S_0 a|a$；
- 由$\delta(0,a)=3$得到$S_3 \to S_0 a|a$；
- 由$\delta(0,b)=2$得到$S_2 \to S_0 b|b$；
- 由$\delta(1,a)=2$得到$S_2 \to S_1 a$；
- 由$\delta(1,b)=1$得到$S_1 \to S_1 b$；
- 由$\delta(2,a)=0$得到$S_0 \to S_2 a$；
- 由$\delta(2,b)=1$得到$S_1 \to S_2 b$；
- 由$\delta(2,\varepsilon)=3$得到$S_3 \to S_2$；

P整理得$S_0 \to S_2 a, S_1 \to S_0 a|S_1 b|S_2 b|a, S_2 \to S_0 b|S_1 a|b, S_3 \to S_0 a|S_2|a$。

求得的文法中出现了单非产生式$S_3 \to S_2$，用2.4.2节的算法删除单非产生式，得到P：

$S_0 \to S_2 a, S_1 \to S_0 a|S_1 b|S_2 b|a, S_2 \to S_0 b|S_1 a|b, S_3 \to S_0 a|S_1 a|S_0 b|a|b$。

3.3.5 正规式转右线性文法

正规式转右线性文法如算法3.9所示。

算法 3.9 正规式转右线性文法

输入： 正规式r
输出： 右线性文法$G_R=(V_N,V_T,P,S)$，使得$L(G_R)=L(r)$

```
P={S→r};
while 存在产生式p，其右部不是右线性文法形式 do
    if p形如A→αβ then p替换为A→αB,B→β;
    if p形如A→α|β then p替换为A→α,A→β;
    if p形如A→α*β then p替换为A→αA,A→β;
end
根据得到的产生式P确定V_N,V_T;
```

例题 3.20 正规式转右线性文法 将正规式$r=(a|b)^*(aa|bb)(a|b)^*$转换为右线性文法G_R，使$L(G_R)=L(r)$。

解 转换过程如下：

- $P=\{S\to(a|b)^*(aa|bb)(a|b)^*\}$
- $S\to(a|b)^*(aa|bb)(a|b)^*$ 利用 $A\to\alpha^*\beta$ 进行分解，得到 $P=\{S\to(a|b)S, S\to(aa|bb)(a|b)^*\}$
- $S\to(a|b)S$ 分解得到 $S\to aS, S\to bS$
  $S\to(aa|bb)(a|b)^*$ 分解得到 $S\to(aa|bb)S_1, S_1\to(a|b)^*$
  因此，$P=\{S\to aS, S\to bS, S\to(aa|bb)S_1, S_1\to(a|b)^*\}$
- $S\to(aa|bb)S_1$ 分解为 $S\to aaS_1, S\to bbS_1$
  $S_1\to(a|b)^*$ 利用 $A\to\alpha^*\beta$ 进行分解，其中 $\beta=\varepsilon$，得到 $S_1\to(a|b)S_1, S_1\to\varepsilon$
  因此，$P=\{S\to aS, S\to bS, S\to aaS_1, S\to bbS_1, S_1\to(a|b)S_1, S_1\to\varepsilon\}$
- 继续分解，得到 $P=\{S\to aS, S\to bS, S\to aS_2, S_2\to aS_1, S\to bS_3, S_3\to bS_1, S_1\to aS_1, S_1\to bS_1, S_1\to\varepsilon\}$

整理得 $G_R=(\{S,S_1,S_2,S_3\},\{a,b\},P,S)$；
其中 $P=\{S\to aS|bS|aS_2|bS_3, S_1\to aS_1|bS_1|\varepsilon, S_2\to aS_1, S_3\to bS_1\}$

3.3.6 正规式转左线性文法

正规式转左线性文法如算法3.10所示。

算法 3.10 正规式转左线性文法

输入： 正规式 r
输出： 左线性文法 $G_L=(V_N,V_T,P,S)$，使得 $L(G_L)=L(r)$

```
1 P={S→r};
2 while 存在产生式p，其右部不是左线性文法形式 do
3     if p形如A→αβ then p替换为A→Bβ, B→α;
4     if p形如A→α|β then p替换为A→α, A→β;
5     if p形如A→αβ* then p替换为A→α, A→Aβ;
6 end
7 根据得到的产生式P确定V_N, V_T;
```

例题 3.21 正规式转左线性文法 将正规式 $r=(a|b)^*(aa|bb)(a|b)^*$ 转换为左线性文法 G_L，使得 $L(G_L)=L(r)$。

解 转换过程如下：

- $P=\{S\to(a|b)^*(aa|bb)(a|b)^*\}$
- $S\to(a|b)^*(aa|bb)(a|b)^*$ 利用 $A\to\alpha\beta^*$ 进行分解，得到 $P=\{S\to(a|b)^*(aa|bb), S\to S(a|b)\}$
- $S\to(a|b)^*(aa|bb)$ 分解得到 $S\to S_1(aa|bb), S_1\to(a|b)^*$
  $S\to S(a|b)$ 分解得到 $S\to Sa, S\to Sb$
  因此，$P=\{S\to S_1(aa|bb), S_1\to(a|b)^*, S\to Sa, S\to Sb\}$
- $S\to S_1(aa|bb)$ 分解为 $S\to S_1aa, S\to S_1bb$
  $S_1\to(a|b)^*$ 利用 $A\to\alpha\beta^*$ 进行分解，其中 $\alpha=\varepsilon$，得到 $S_1\to S_1(a|b), S_1\to\varepsilon$
  因此，$P=\{S\to S_1aa, S\to S_1bb, S_1\to S_1(a|b), S_1\to\varepsilon, S\to Sa, S\to Sb\}$
- 继续分解，得到 $P=\{S\to S_2a, S_2\to S_1a, S\to S_3b, S_3\to S_1b, S_1\to S_1a, S_1\to$

$S_1b, S_1 \to \varepsilon, S \to Sa, S \to Sb\}$

整理得 $G_L = (\{S, S_1, S_2, S_3\}, \{a, b\}, P, S)$；

其中 $P = \{S \to Sa|Sb|S_2a|S_3b, S_1 \to S_1a|S_1b|\varepsilon, S_2 \to S_1a, S_3 \to S_1b\}$

3.3.7 正规文法转正规式

正规文法转正规式，反复利用以下规则合并文法的产生式，最后只剩下一个开始符号定义的产生式，并且产生式右部不含非终结符合，那么产生式右部即为所求正规式。

- 右线性文法
  - $A \to \alpha B, B \to \beta$，替换为 $A \to \alpha\beta$
  - $A \to \alpha_1, A \to \alpha_2$，替换为 $A \to \alpha_1|\alpha_2$
  - $A \to \alpha A, A \to \beta$，替换为 $A \to \alpha^*\beta$
- 左线性文法
  - $A \to B\alpha, B \to \beta$，替换为 $A \to \beta\alpha$
  - $A \to \alpha_1, A \to \alpha_2$，替换为 $A \to \alpha_1|\alpha_2$
  - $A \to A\alpha, A \to \beta$，替换为 $A \to \beta\alpha^*$

例题 3.22 右线性文法转正规式 将右线性文法 $G[S]: S \to aS|bS|aB|bC, A \to aA|bA|\varepsilon, B \to aA, C \to bA$ 转为正规式。

解 转换过程如下：

- 由 $S \to aS|bS|aB|bC$ 得到 $S \to (a|b)S|(aB|bC)$，进一步得到 $S \to (a|b)^*(aB|bC)$
- 将 $B \to aA, C \to bA$ 代入上式，得到 $S \to (a|b)^*(aaA|bbA)$，即 $S \to (a|b)^*(aa|bb)A$
- 由 $A \to aA|bA|\varepsilon$ 得到 $A \to (a|b)A|\varepsilon$，进一步得到 $A \to (a|b)^*$
- 将第 (3) 步结果代入第 (2) 步，得到 $S \to (a|b)^*(aa|bb)(a|b)^*$

最终得到正规式：$(a|b)^*(aa|bb)(a|b)^*$

例题 3.23 左线性文法转正规式 将左线性文法 $G[S]: S \to Ba|Cb|Sa|Sb, A \to Aa|Ab|\varepsilon, B \to Aa, C \to Ab$ 转为正规式。

解 转换过程如下：

- 由 $S \to Ba|Cb|Sa|Sb$ 得到 $S \to (Ba|Cb)|S(a|b)$，进一步得到 $S \to (Ba|Cb)(a|b)^*$
- 将 $B \to Aa, C \to Ab$ 代入上式，得到 $S \to (Aaa|Abb)(a|b)^*$，即 $S \to A(aa|bb)(a|b)^*$
- 由 $A \to Aa|Ab|\varepsilon$ 得到 $A \to A(a|b)|\varepsilon$，进一步得到 $A \to (a|b)^*$
- 将第 (3) 步结果代入第 (2) 步，得到 $S \to (a|b)^*(aa|bb)(a|b)^*$

最终得到正规式：$(a|b)^*(aa|bb)(a|b)^*$

3.3.8 3种工具的转换

到目前为止，我们已经详细介绍了FA、正规式、正规文法3种用于词法分析的工具，它们是完全等价的。其中正规式人类容易设计，DFA容易编程实现，正规文法则适合做数学推导以及作为正规式和FA之间转换的桥梁，它们之间的关系如图3.30所示。

这3者之间及内部转换总结如下。

- FA内部，NFA和DFA可以互转，DFA可以进一步化简为最简DFA；
- 正规文法内部，左、右线性文法互转并不那么直接，一般先转换为正规式或FA，再进一步转换为另一种正规文法；

- FA 和正规文法之间容易互转；
- 正规文法和正规式之间容易互转；
- 正规式容易转换为FA，但是FA转换为正规式并不容易，一般需要借助正规文法完成转换。

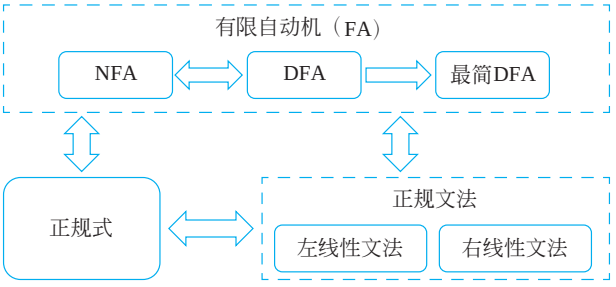


图 3.30 FA、正规式、正规文法之间的转换

3.3.9 有限自动机转正规式

有限自动机直接转正规式，虽然在3.2.5节的正规式与有限自动机的等价性证明中给出了一种直观的构造方法，但实际操作中这种转换非常不容易，主要原因在于FA中有很多循环，导致转换中无法厘清适用的状态转移路径，因此常使用正规文法作中介。

下面的例子用来说明如何将FA转换为正规式。这个例子相当复杂，只是为了让读者了解这种转换过程，以及一个看似简单问题的复杂程度。

例题 3.24 FA 转正规式 写出能被3整除十进制数的正规式。

解 图3.6中已给出接受能被3整除十进制数的DFA，在此基础上，先转换为正规文法，再转正规式。

【步骤一】DFA转右线性文法

为书写简洁，记$\alpha \to 0|3|6|9, \beta \to 1|4|7, \gamma \to 2|5|8$

(1) 由状态0射出的弧，有 $S_0 \to \alpha S_0|\beta S_1|\gamma S_2|\alpha$

(2) 由状态1射出的弧，有 $S_1 \to \alpha S_1|\beta S_2|\gamma S_0|\gamma$

(3) 由状态2射出的弧，有 $S_2 \to \alpha S_2|\beta S_0|\gamma S_1|\beta$

(4) S_0既为初态又是终态，可以接受ε，为简便，我们把空符号单列，写成$S_0' \to S_0|\varepsilon$的形式，其中S_0'为新的开始符号，这样我们只需推导出S_0，然后与ε做或运算即可得到最终的正规式。

【步骤二】右线性文法转正规式

为进一步书写简洁，记$\lambda \to \alpha^*\alpha, \mu \to \alpha^*\beta, \nu \to \alpha^*\gamma$

(1) 由1(1)：$S_0 \to \alpha^*(\beta S_1|\gamma S_2|\alpha)$，即$S_0 \to \mu S_1|\nu S_2|\lambda$

(2) 由1(2)：$S_1 \to \alpha^*(\beta S_2|\gamma S_0|\gamma)$，即$S_1 \to \mu S_2|\nu S_0|\nu$

(3) 由1(3)：$S_2 \to \alpha^*(\beta S_0|\gamma S_1|\beta)$，即$S_2 \to \mu S_0|\nu S_1|\mu$

(4) 将2(3)代入2(2)：$S_1 \to \mu(\mu S_0|\nu S_1|\mu)|\nu S_0|\nu$，即$S_1 \to (\mu\mu|\nu)S_0|\mu\nu S_1|\mu\mu|\nu$

(5) 将2(3)代入2(1)：$S_0 \to \mu S_1|\nu(\mu S_0|\nu S_1|\mu)|\lambda$，即$S_0 \to \nu\mu S_0|(\nu\nu|\mu)S_1|\nu\mu|\lambda$

(6) 由2(4)，消掉右部递归的S_1，得到$S_1 \to (\mu\nu)^*((\mu\mu|\nu)S_0|\mu\mu|\nu)$

(7) 将2(6)代入2(5)：$S_0 \to \nu\mu S_0|(\nu\nu|\mu)(\mu\nu)^*((\mu\mu|\nu)S_0|\mu\mu|\nu)|\nu\mu|\lambda$

即$S_0 \to (\nu\mu|(\nu\nu|\mu)(\mu\nu)^*(\mu\mu|\nu)S_0)|(\nu\nu|\mu)(\mu\nu)^*\mu\mu|(\nu\nu|\mu)(\mu\nu)^*\nu|\nu\mu|\lambda$

(8) 由2(7)：$S_0 \to (\nu\mu|(\nu\nu|\mu)(\mu\nu)^*(\mu\mu|\nu))^*((\nu\nu|\mu)(\mu\nu)^*\mu\mu|(\nu\nu|\mu)(\mu\nu)^*\nu|\nu\mu|\lambda)$

(9) 2(8)的右部，与ε或运算，即为最终结果。

3.4 词法分析器的实现

我们已经介绍了词法分析的相应数学工具，本节采用人工设计正规式，使用DFA实现单词识别。

3.4.1 词法分析器边界

在实现词法分析器前，需要先明确词法分析器的边界，确定哪些功能由词法分析器实现，哪些功能放到后续部分实现，以便更好地实现词法分析器。

词法分析的一个误区是识别出所有单词并发现所有错误。实际上，词法分析是基于单词的形式，也就是词法构成规则的，这些规则不涉及单词的上下文信息。因此，有些看上去似乎属于词法构成的规则，只要涉及上下文信息，就需要放到语法甚至语义分析中识别，这是词法分析需要注意的一个重要问题。编译器设计的原则是，能提前处理的信息要提前处理，但不属于该阶段的任务，需要留到后续模块完成。

要识别出单词，词法分析程序需要至少预读一个符号。如词法分析中，已经扫描了字符串“if”，后面遇到空白或左括号之类，才能确定这是关键字，此时多读了一个符号，需要退给词法分析程序。而“if”后如果是字母、数字或下画线，则可以确定这是一个标识符，但是只有读到非字母、数字、下画线的符号，才能确定这个标识符结束，这时也多读了一个符号。我们词法分析器的设计，期望只预读一个符号就可以识别出所有单词及其类别。

关于词法分析器的边界说明如下。

- 连续两个标识符，如“x␣y”，两个标识符x和y之间有一个空格，只预读一个符号时，词法分析器应识别出两个标识符。两个标识符连在一起属于语法错误，需要到语法分析时才能发现。
- 标识符与数字连接在一起，如“x␣100”，标识符x和数字100之间有一个空格，此时词法分析应识别出一个标识符和一个数字，词法分析无法发现这种语法错误。
- 数字和标识符连在一起，如“2x”，在只预读一个字符的前提下，当数字和标识符之间没有空格时，是能发现这种错误的；而如果有空格，则无法发现这种错误。但这种错误是否放到词法分析部分处理值得商榷，如果放到词法分析部分处理，就需要区分数字后面跟的是一般字母还是空白、算符、界符等；如果放到语法分析处理，数字后面遇到任何非数字都可以认为识别出一个数字，显然后者更容易处理。
- 代码中混合的注释，如“x/*comment*/y”，大部分预处理作为独立一遍的编译器，都会把注释转换为空格，识别出来两个标识符x和y。实际上，把注释删除也是可以的，这样得到一个标识符xy，根据设计的方便性选择其中一个作为语言标准即可。
- 数字常数的符号，不在词法分析时确定。因为对“−5”之类的表达式，不根据上下文无法确定是负5还是减5，词法分析识别出一个减号和一个数字5，到语法分析和语义分析时再确定减号和负号。
- 幂运算符“**”由两个乘号组成，左移运算符“<<”由两个小于号组成，自增“++”

和自减“−−”分别由两个加号和两个减号组成，此类字符在词法分析中是可以和乘号、小于号、加号、减号区分的，规则是较长字符串优先。当然，这类运算符也可以识别为两个符号，遗留到语法分析中再进行处理。

- 三元运算符“?:”由一个问号和一个冒号组成，可看作两个单词，到语法分析和语义分析时再进行组合。

3.4.2 单词正规式

首先定义一些公共字符集，如表3.10所示。

表 3.10 字符集

类 别	正 规 式	说 明
< 字母 >	a|b|··· |z|A|B|··· |Z	大小写字母
< 数字 >	0|1|2|··· |9	一位数字
< 任意 >	Σ	任意字符
< 其他 >		射出弧标记外的任意字符
<H 数字 >	0|1|2|··· |9|A|B|C|D|E|F|a|b|c|d|e|f	十六进制数字
< 空白 >	\t|\n|\r|␣	␣表示空格
< 标首 >	< 字母 > | _	标识符首字母
< 标中 >	< 字母 > | < 数字 > | _	标识符非首字母

然后定义常用单词的正规式，主要是标识符、常量、注释，如表3.11所示。很多语言的常量数据类型相当复杂，如浮点数小数点左右的数字可以省略一个但不能同时省略、指数形式的输入，等等。这些复杂的输入形式只是增加了设计的复杂程度，并不影响词法分析器实现的本质问题。作为一个示例，我们采用相对简洁的单词形式，有兴趣的读者可以自行加入其他形式的单词形式。

表 3.11 常用单词的正规式

类 别	正 规 式
< 标识符 >	< 标首 >< 标中 >*
< 整数 >	< 数字 >+
<H 整数 >	0x< H 数字 >+
< 实数 >	< 数字 >+ . < 数字 >+
< 字符 >	‘< 任意 >’
< 字符串 >	“< 任意 >*”
< 单行注释 >	//< 任意 >*(\r|\n)
< 多行注释 >	/*< 任意 >**/

其中 < 字符 > 和 < 字符串 > 支持转义字符，包括：

- \r，回车符
- \n，换行符
- \t，跳格符（水平制表符）

- \'，单引号
- \"，双引号
- \\，反斜杠
- \<整数>，十进制ASCII码（注意，这个与C语言不同）
- \x<H数字>$^+$，十六进制ASCII码

界符包括：逗号、分号、小括号、花括号。

关键字包括：byte、ubyte、char、bool、short、ushort、int、uint、float、double、var、true、false、if、else、while、for、switch、case、default、goto、foreach、void、main。

运算符包括：=、+、−、*、/、%、**、<、<=、>、>=、==、!=、!、&&、||、&、|、~、^、<<、>>、?、:。

界符、关键字和运算符都采用一字（符）一类。关键字可以先按标识符识别，再查表得到；也可以直接用DFA识别，我们选择后者。

3.4.3 识别单词的DFA

词法分析程序需要将这些正规式整合成一个完整DFA。目前为止，正规式转DFA的过程还无法采用算法自动实现，主要是NFA单符化的过程需要手工完成，因此这部分我们展示一个手工设计的DFA。当学习了语法制导翻译后，可以实现正规式自动转换为DFA的算法。

由于篇幅限制，一页A4纸难以展示这个完整的DFA，并且其原理是重复的，因此我们选取一部分子集做示例，剩余部分读者可以自行添加。

这个展示的示例包括了所有标识符、常量、注释、界符，以及部分关键字和运算符。关键字我们选择int、float、if、else、while等少数几个，其他关键字类似处理；也可以不把关键字加入DFA，识别完标识符后通过查表确定是标识符还是关键字。运算符包括=、+、−、*、/、%、**，其中展示了*和**的区分方法。

关于DFA的几种状态，做一下特殊说明。

- 状态0：是唯一的初态。
- 状态−1：两个终态之一，为识别一个单词成功的状态。
- 状态−2：两个终态之一，为出错状态。
- 状态−1的前驱状态：用于区分识别出的单词类别，可以用这个状态作为单词类别的编码。
- 其他状态，属于正常内部状态。

DFA识别出一个单词时，总是预读了一个字符，需要把这个字符退回给词法分析器，也就是到达状态−1后，词法分析器根据前一个状态确定单词类别，并根据当前位置记录识别出的单词，然后将扫描器指针回退一个字符，并把状态置为初态0，开始下一个单词的识别。

进入终态−2表示出错。DFA中可以显式或隐式跳转到这个状态，显式跳转指DFA中有射入该状态的弧，通过该弧跳转；隐式跳转指当前输入字符在DFA当前状态下查询不到下一个状态，则默认跳转到状态−2。

DFA在初态，如果遇到空白符（如空格、回车、换行、跳格等）则空转，也就是把这些空白符忽略。

如果程序末尾没有空白符，比如以}结尾，会导致缺少预读字符而无法到达终态–1。为解决这个问题，从文件中读取程序到文件末尾时，需要自动在后面追加一个空白符。

图3.31为识别关键字和标识符部分的DFA。先看标识符识别，int、float、if、else和while这5个关键字的首字母包括i、f、e、w，当状态0遇到的字符不是这些字母，且为标首时（标首指字母和下画线，标中指字母、数字和下画线，见表3.10定义），转到状态19（最下面一行）。状态19遇到标中，就转移到本身19，遇到其他字符转到终态–1，表示识别出一个标识符。

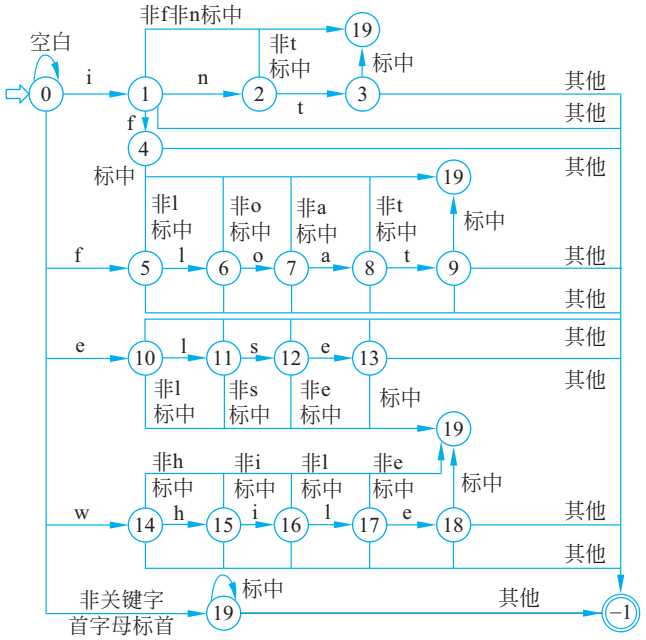


图 3.31 识别单词的DFA之1/4

识别关键字int的路径为$0 \xrightarrow{i} 1 \xrightarrow{n} 2 \xrightarrow{t} 3 \xrightarrow{其他} -1$，识别关键字if的路径为$0 \xrightarrow{i} 1 \xrightarrow{f} 4 \xrightarrow{其他} -1$。这两个关键字都通过i进入状态1，然后根据下一个字符进入不同分支：n、f分别进入int和if识别，而非f非n的标中，说明这是一个普通标识符，转移到状态19进入标识符识别。图3.31中多次出现状态19，是因为将所有有向边引到同一结点会比较凌乱，因此将结点19画在多处。这些状态19是同一结点，其射入和射出的弧取并集即可。

状态1射出的n弧、f弧、非f非n标中弧，三者的并集是标中。状态1上射出的另外一条弧，标记为其他，即非标中（包括空白、界符、算符），“其他”这条路径说明这个i自身就是一个完整的标识符，所以转移到状态–1，表示已经识别出一个单词。

状态2识别出t后进入状态3，表示还在int关键字的分支。而非t标中表示这是一个标识符，转到状态19。除这两种情况外，说明in是一个标识符，转到状态–1。其他状态转移类似，请读者自行分析。

如前所述，状态–1的前一个状态代表了单词类别，比如状态19代表标识符，状态3代表关键字int，状态4代表if，等等。但是，状态1也可以转移到–1，这表示识别出了标识符i，所以状态1也表示标识符。这些表示特殊标识符的状态，应该与状态19合并，它们表示的是同一类单词。

另外还需注意一点，在目前我们的DFA中，状态2是表示标识符的，它代表识别出了

标识符 in。但如果包含了 foreach 语句，in 就会成为关键字，此时状态 2 不再表示标识符，而是代表关键字 in 的类别。所以，每个状态代表的单词类别，需要根据关键字集合确定。

图3.32为识别数字和算符、界符部分的DFA。0开头的，可能是数字，也可能是十六进制数字。状态0遇到字符0进入状态20，如果再遇到x，即为十六进制数字，进入状态21；如果遇到数字，则是十进制数字，进入状态23；遇到小数点，则是小数，进入状态24；其他则是单独的一位数字0，进入状态 –1。状态21后面必须是十六进制数字，否则就是错误，错误的情况进入状态 –2。

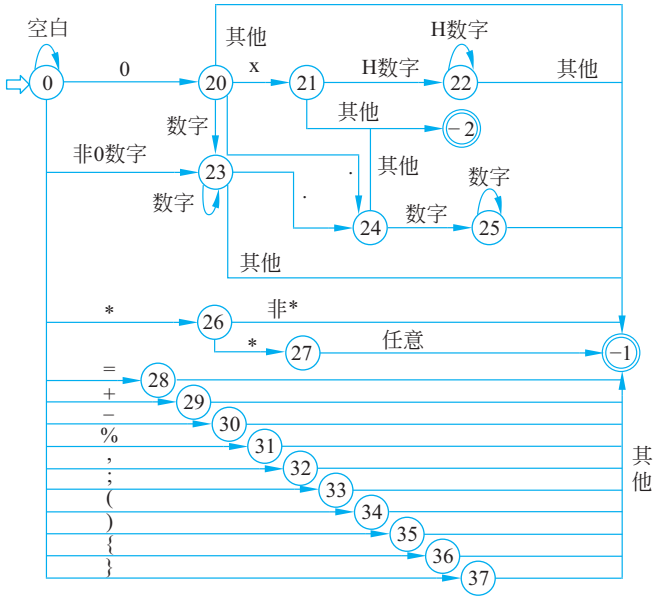


图 3.32　识别单词的 DFA 之 2/4

非0数字进入状态23，此后可以接受多位数字，均在状态23。如果遇到小数点，则为小数，进入状态24；其他字符则是数字结束，进入状态 –1。小数点后至少一位数字，如果没有数字，则出错，进入状态 –2。

状态26和状态27分别为识别乘号 * 和幂 ** 的状态，状态28~37则为各种算符、界符的识别，但不包括除号 (/)。

图3.33为识别字符和字符串常量的DFA。遇到单引号进入状态38，开始单个字符的识别。如果是转义字符，即遇到反斜杠进入状态41，分十六进制数字和十进制数字两种情况。识别转义字符之所以要特殊处理，是因为要把它们转换成内码表示，比如换行符 \n 要转换成 ASCII 码 0A 存储，而不是分 \ 和 n 两个字符存储，完成识别时需要匹配一个动作进行转换。再次遇到单引号表示已经识别出这个字符，但为了预读一个字符与前面保持一致，这里并没有直接进入状态 –1，而是读一个字符再进入状态 –1，这样就可以与前面一样回退一个字符。

字符串的转义字符识别略有不同，主要体现在数字和十六进制数字作为转义字符输入时。这是因为字符常量对数字的结束符一定是单引号，可以把单引号之前的数字进行转义。而字符串常量的数字结束符不确定是什么，所以为了方便处理，识别出数字或十六进制数字转义后，应回退一个字符，然后继续处理字符串常量。

图3.34为识别注释的DFA。单行注释以//开头，多行注释包含在/*··· */之间。状态0

遇到/时，移到状态51。此时如果后面跟的不是/，也不是*，则不是注释，而是除号，直接转状态−1。状态51遇到/进入单行注释识别，遇到回车\r或换行符\n结束（Windows下先遇到回车符，Linux下只遇到换行符）。多行注释的结束，可能会写多个*才写/，即以类似“*********/”的形式作为多行注释的结束，这是设计中需要考虑的一个问题。

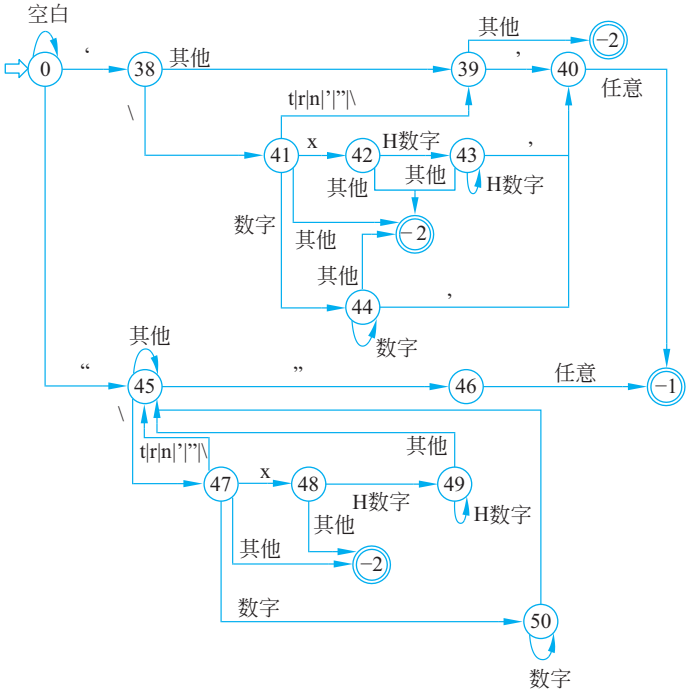


图 3.33 识别单词的DFA之3/4

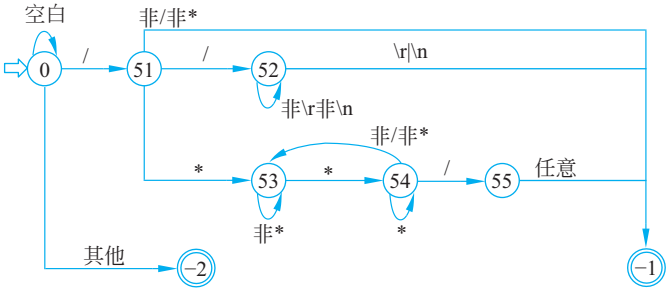


图 3.34 识别单词的DFA之4/4

与其他单词识别不同的是，注释识别出来后不需要做记录，直接丢弃即可。

3.4.4 单词识别算法

本部分只介绍主体算法框架，需要特殊处理的情形，在3.4.3节DFA设计中已做分析，不再赘述。

用到的几个函数或属性如下。

- $buffer.length$，指数组$buffer$的长度。
- $buffer[i:j]$，取数组的i到j之间的元素，含i和j。
- $getNextState(state, symbol)$，根据当前状态和当前符号，返回转移的状态。

- $tokens.add(type, value)$，将单词类型和单词值作为二元组加入$tokens$记录下来，注意标识符的单词类型需要整合。

算法3.11为单词识别的主体框架，输入一个文件，将单词类别和单词值按二元组组成序列输出。

算法 3.11 单词识别

输入： 文件名称

输出： 单词序列$tokens$，其每个结点为单词类别和单词值的二元组

```
1  根据文件名称将源程序读入字符数组buffer，并在最后加一个空白符号;
2  当前状态state = 0, 前一个状态preState = −1;
3  当前符号指针pCur = 0，开始位置指针pStart = −1;
4  while pCur < buffer.length do
5      pStart = pCur;
6      while state ≠ −1 do
7          preState = state;
8          state = getNextState(preState, buffer[pCur]);
9          if state = −2 then 报错退出;
10         判断该状态是否需要特殊处理;
11         pCur++;
12     end
       // 目前识别出了一个单词
13     if PreState ≠ 52 ∧ PreState ≠ 55 then
14         tokens.add(preState, buffer[pStart : pCur − 1]);
15     end
16     pCur − −;
17     state = 0;
18 end
```

第1行将源程序读入缓冲区$buffer$，并在最后追加一个空白符号。第2行将状态$state$初始化为0，$state$的前一个状态$preState$初始化为-1，由$preState$确定单词类别。第3行初始化输入符号指针，其中识别一个单词时，$pStart$总是指向这个单词的起始位置，而$pCur$逐个字符向后扫描；发现单词结尾时，$pStart$和$pCur$之间的符号串为一个单词，其中最后一个符号是预读的符号。

第4~18行开始逐字符扫描源程序符号串，结束条件是$pCur$到达了缓冲区$buffer$末尾。第5行将$pStart$指向$pCur$位置，这是一个单词的开始符号位置。第6~12行由$pCur$逐字符向后扫描，直至$state = -1$到达一个单词的结尾。首先将$preState$置为当前状态$state$（第7行），而DFA根据当前状态和当前符号确定下一个状态，赋值给$state$（第8行）。如果$state = -2$，则报错退出（第9行）。每转换一个新状态，需要判断是否为转义字符，如果是，则进行转义处理（第10行），然后$pCur$读取下一个符号（第11行），当$state = -1$时结束一个单词的识别（第6行）。

识别一个单词后，状态为52或55为注释；非注释时（第13行），将前一个状态$preState$作为单词类别，$pStart$到$pCur - 1$之间为单词值，存入$tokens$（第14行）。之后，回退预读的字符（第16行），并将状态重新置初态0（第17行），进入下一个单词的识别。

第3章 词法分析 内容小结

- 词法分析器可以作为独立的一遍，也可以作为一个独立子程序由语法分析器调用。
- DFA由当前状态和当前符号唯一确定下一个状态，容易编程实现，用于单词识别。
- NFA容易设计，但不适合编程实现，可以通过确定化算法转换为DFA。
- 正规文法、正规式和FA是等价的，可以互相转换。

第3章 习题

第3章 习题

请扫描二维码查看第3章习题。