重庆大学
CHONGQING UNIVERSITY

第二讲 形式语言与自动机基础

重庆大学 计算机学院 张敏

CHONGQING UNIVERSITY

本讲要点

- 语言和文法
- 有限自动机
- 自动机、文法、表达式等价性

讨论

要开发一个编译器，第一步应该做什么？

定义程序设计语言

思考

如何定义程序设计语言?

如何定义自然语言?

- 汉语--所有符合汉语语法的句子的全体

程序设计语言--所有该语言的程序的全体

语言中哪些部分是有穷的?哪些部分是无穷的?

- 程序是有穷的吗?

如何表示一种语言中的每个成分?

- 枚举?
- 以有限表示无限?

2.1 语言和文法

如果语言是有穷的，可以将句子逐一列出来表示

如果语言是无穷的，找出语言的有穷表示：

- ✓ 生成方式： 文法
- ✓ 识别方式： 自动机

2.1.1 语言定义相关概念

每种语言具有两个可识别的特性：

- 语言的形式
- 与形式相关联的意义

语言的实例相关联的意义：

- 该句子的创立者所想要表示的意义
- 接收者所检验到的意义

程序设计语言

- 每个程序构成的规律
- 每个程序的含义
- 每个程序和使用者的关系

2.1.2 形式语言

语法意义下的语言称作形式语言。
形式语言抽象地定义为一个数学系统。

2.1.2.1 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。

英语字母表：Σ={a,b,c,……x,y,z,A,B,…,Y,Z}

机器语言字母表： Σ={0，1}

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

Σ={a,b}

ε ,a,b,aa,ab,aabba…都是Σ上的符号串

1. 空符号串 ε (没有符号的符号串)是Σ上的符号串
2. 若α是Σ上的符号串,a是Σ的元素,则α a是Σ上的符号串
3. y是Σ上的符号串,当且仅当它可以由1和2导出

2.1.2.1 字母表和符号串

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。

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是university中的子序列

2.1.2.2 语言

语言：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。

2.1.2.3 语言的运算

假设 L 和 M 表示两个语言
L和M的并记作L∪M：L∪M={s|s∈L 或 s∈M}
L和M的连接记作LM：LM={st|s∈L 并且 t∈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。

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

2.1.2.3 语言的运算

L={A,B, … ,Z,a,b, … ,z}，D={0,1, … ,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	描述
L∪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∪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

2.1.2.3 语言的运算

例：$\Sigma=\{a,b\}$

$\Sigma^*=\{\varepsilon,a,b,aa,ab,ba,bb,aaa,aab,...\}$

$\Sigma^+=\{a,b,aa,ab,ba,bb,aaa,aab,...\}$

- 集合$\{ab,aabb,aaabbb,...,a^nb^n,...\}$或$\{w|w\in\Sigma^*且w=a^nb^n,n\geq1\}$为字母表$\Sigma$上的一个语言。
- 集合$\{a,aa,aaa,...\}$或$\{w|w\in\Sigma^*且w=a^n,n\geq1\}$ 为字母表Σ上的一个语言。
- $\{\varepsilon\}$是字母表Σ上的一个语言。 Φ即$\{\ \}$是字母表Σ上的一个语言。

2.1.3 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$G=(V_T,V_N,S,P)$

  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。

  P是一个非空的有限集合，它的每个元素称为产生式。

2.1.3.1 产生式的定义

规则（产生式或生成式），是形如α→β或α::=β的（α，β）有序对，且α∈V^+，β∈V^*。

- α称为规则的左部(或生成式的左部)。
- β称为规则的右部(或生成式的右部)。

左部相同的产生式$\alpha\to\beta_1$、$\alpha\to\beta_2$、……、$\alpha\to\beta_n$可以缩写为 $\alpha\to\beta_1|\beta_2|\ldots\ldots|\beta_n$

“|” 表示 “或”， 每个$\beta_i(i=1,2,\ldots,n)$称为α的一个候选式

BNF（Backus-Naur Form）

::= 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

2.1.3.2 产生式书写约定

非终结符号：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。

示例：

1、文法G=（V_N，V_T，P，S），V_N = { S }， V_T ={ 0，1 }，P={ S→0S1， S→01 }，S为开始符号

2、文法G=（V_N，V_T，P，S），V_N ={标识符，字母，数字}，V_T ={a,b,c,…x,y,z,0,1,…,9}

P={ <标识符>→<字母>
<标识符>→<标识符><字母>
<标识符>→<标识符><数字>
<字母>→a,…，<字母>→z
<数字>→0,…，<数字>→9 }

S=<标识符>

2.1.4 文法推导

文法所产生的语言

从文法的开始符号出发，推导得到的所有字符串的集合，为该文法定义的语言。

推导

假定A→γ是一个产生式，α和β是任意的文法符号串，则：αAβ ⇒ αγβ 称αAβ直接推导出αγβ，也可以说αγβ是αAβ的直接推导 或说αγβ直接归约到αAβ

如果有直接推导序列：α1⇒α2⇒...⇒αn 则说α1推导出αn，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

示例：

<句子>→<名词><动词><名词>

<名词>→狼|羊|草|人|水

<动词>→吃|喝

2.1.4 文法推导

示例：G=({+，*，（，），i},{E，T，F},E,φ)

φ：E→E+T|T

T→T*F|F

F→(E)|i

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 ⇒ E+T ⇒ T+T ⇒ F+T ⇒ i+T ⇒i+F ⇒i+i

2.1.4 文法推导

最左推导

如果 $\alpha \overset{*}{\Rightarrow} \beta$，并且在每“一步推导”中，都替换$\alpha$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$\alpha$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

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

讨论

根据文法对某个句型/句子进行最左推导或者最右推导，推导过程是唯一的吗？什么情况下不是唯一的？（例如表达式 1+2-3）

2.1.5 句子、句型、语言

句型

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

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

若$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$，则$\alpha$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)。

2.1.6 短语、直接短语和句柄

- 对于文法G=(V_T,V_N,S,φ)，假定αβδ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αβδ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αβδ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。

例如:

$$\underline{E}\Rightarrow\underline{T}\Rightarrow\underline{T*F}\Rightarrow\underline{T*(E)}\Rightarrow\underline{F*(E)}\Rightarrow\underline{i*(E)}\Rightarrow\underline{i*(E+T)}\Rightarrow\underline{i*(T+T)}\Rightarrow\underline{i*(F+T)}\Rightarrow\underline{i*(i+T)}$$

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

2.1.7 语言的定义

由文法G生成的语言记为L(G), 是文法G的一切句子的集合:
$L(G)=\{x|S =>x$，其中S为文法的开始符号，且$x \in V_T^*\}$

例1：G[S]： S→0S1， S→01

$$L(G)=\{0^n1^n|n\geqslant 1\}$$

例2：文法G[S]：

（1）S→aSBE
（2）S→aBE
（3）EB→BE
（4）aB→ab
（5）bB→bb
（6）bE→be
（7）eE→ee

$$L（G）=\{ a^nb^ne^n \mid n\geqslant 1 \}$$

2.1.8 文法的等价

若L（G_1）=L（G_2），则称文法G_1和G_2是等价的。

如文法G_1[A]：A→0R | 01 | R→A1 与 G_2[S]：S→0S1 | 01 等价

2.1.9 分析树及二义性

分析树：推导的图形表示即分析树，又称推导树。

分析树的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

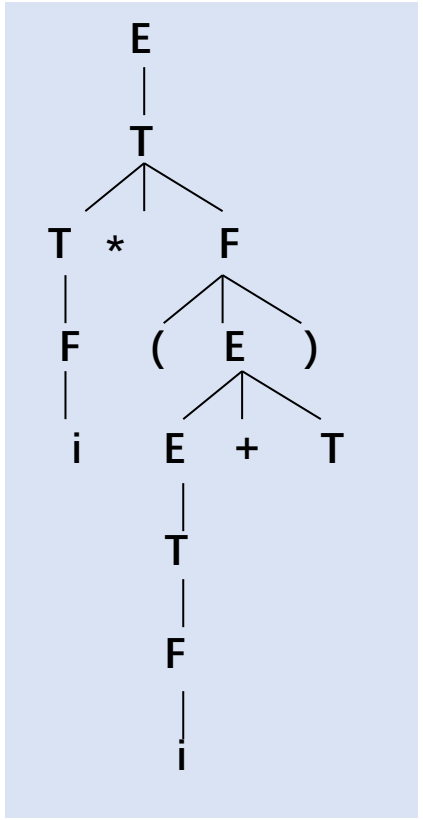
2.1.9 分析树及二义性

G[E]：E→E+T|T
T→T*F|F
F→(E)|i

句型：i*(i+T)

推导过程：

E⇒T⇒T*F⇒T*(E)⇒F*(E)⇒i*(E)
⇒i*(E+T)⇒i*(T+T)⇒i*(F+T)⇒i*(i+T)

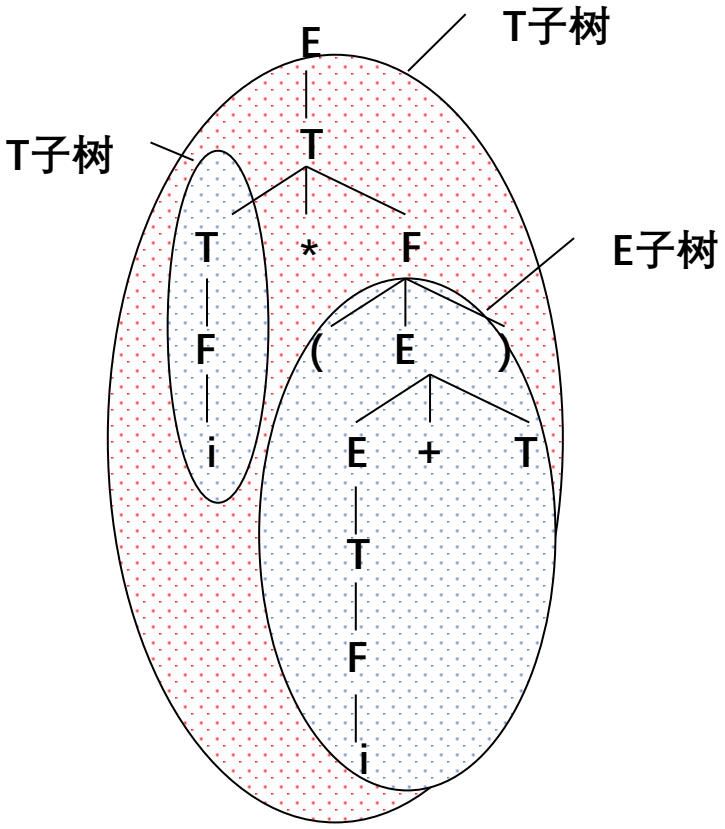


2.1.9 分析树及二义性

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。

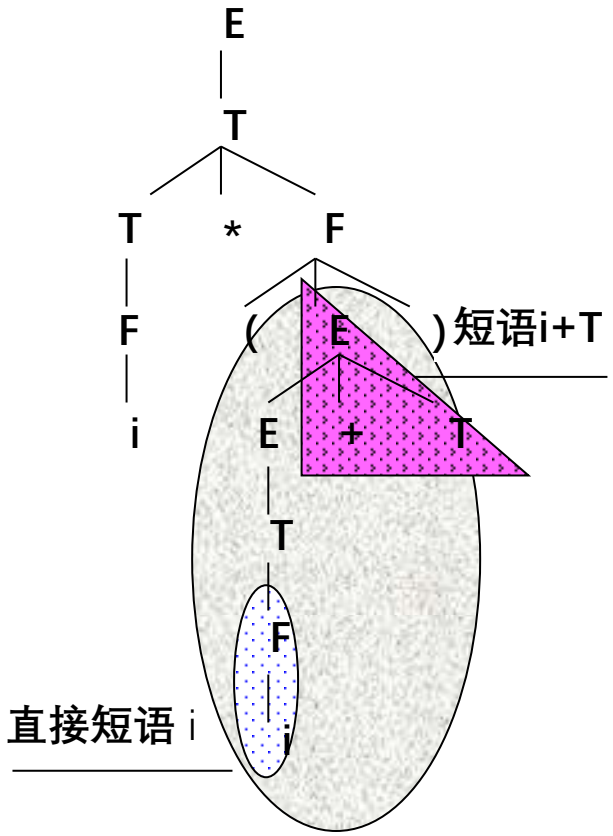
如果子树的根结点标记为非终结符号A，则可称该子树为A-子树。



2.1.9 分析树与二义性

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。



讨论

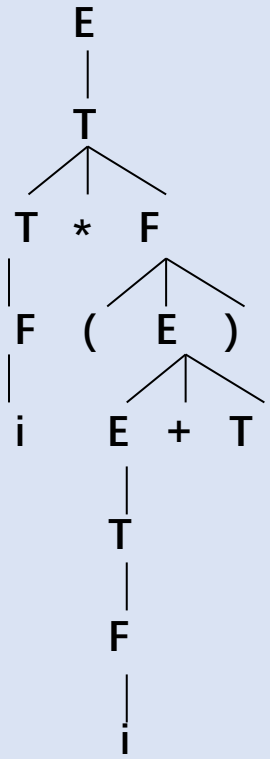
一个句子对应的分析树是唯一的吗?

G[E]：E→E+T|T
　　T→T*F|F
　　F→(E)|I

句型：　i*(i+T)

E⇒T⇒T*F⇒T*(E)⇒F*(E)⇒i*(E)⇒i*(E+T)⇒i*(T+T)⇒i*(F+T)⇒i*(i+T)

E⇒T⇒T*F⇒F*F⇒i*F⇒i*(E)⇒i*(E+T)⇒i*(T+T)⇒i*(F+T)⇒i*(i+T)



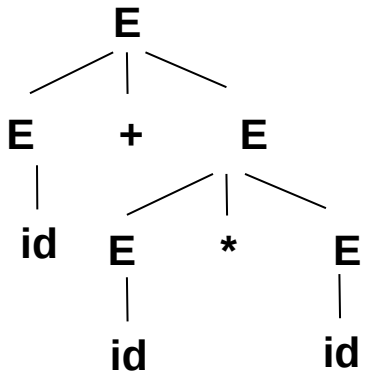
2.1.9 分析树与二义性

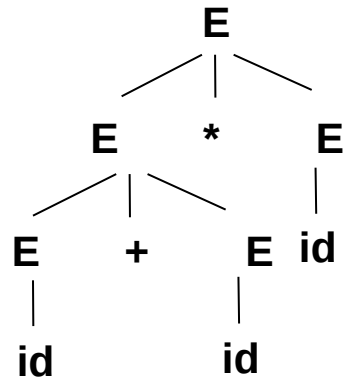
文法的二义性

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,φ)
φ: E→ E+E | E*E | (E) | id

句子 id+id*id 存在两个不同的最左推导：
E⇒E+E⇒id+E⇒id+E*E⇒id+id*E⇒id+id*id
E⇒E*E⇒E+E*E⇒id+E*E⇒id+id*E⇒id+id*id

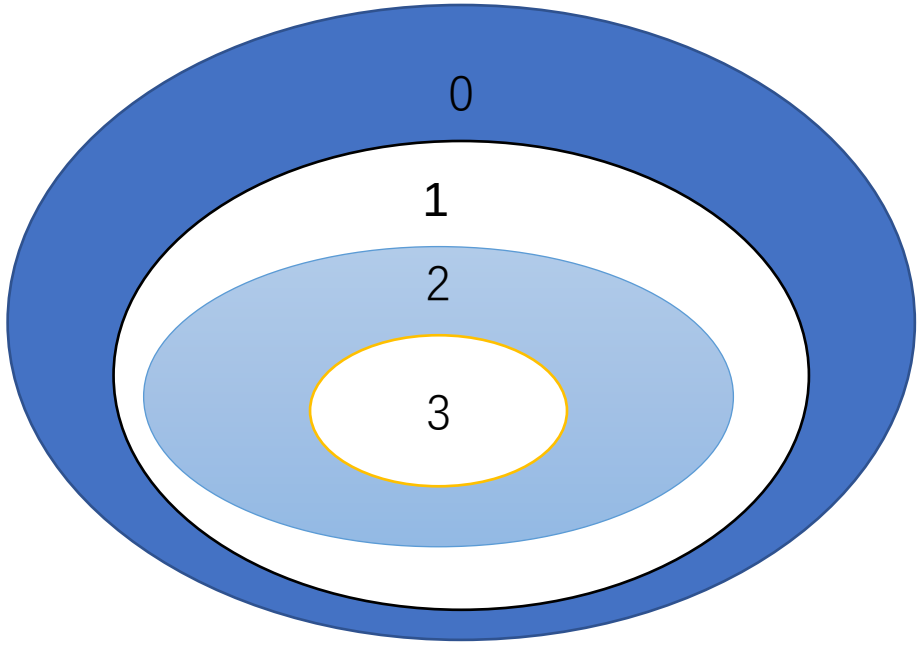




语言的二义性

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。
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。
二义性问题是不可判定的

2.2 Chomsky文法体系



2.2 Chomsky文法体系

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	$\alpha \rightarrow \beta$ 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	$\alpha \rightarrow \beta$ 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	$A \rightarrow \beta$ 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）	$A \rightarrow a$或$A \rightarrow aB$（右线性），或 $A \rightarrow a$或$A \rightarrow Ba$（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

2.2 Chomsky文法体系

1型（上下文有关）文法

文法G[S]：
S→aSBE
S→aBE
EB→BE
aB→ab
bB→bb
bE→be
eE→ee

2型（上下文无关）文法

文法G[S]：
S→aB|bA
A→a|aS|bAA
B→b|bS|aBB

3型（正则）文法

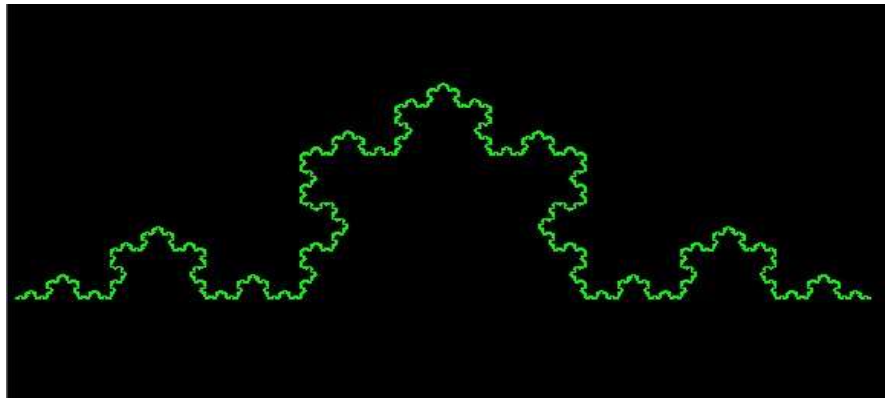
文法G[S]：
S→0A|1B|0
A→0A|1B|0S
B→1B|1|0

讨论

比较几种类型的文法，程序设计语言中不同组成成分，各自适合用哪种文法进行描述？为什么？

拓展：文法应用-分形

分形（frcatals）



http://paulbourke.net/fractals/

拓展：文法应用-分形

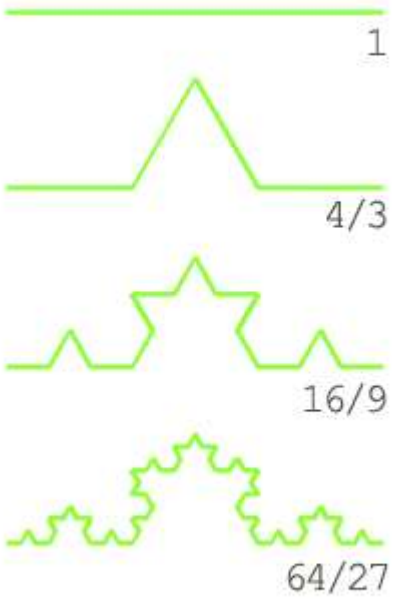
序列1：0
序列2：0 1
序列3：0 1 1 2
序列4：0 1 1 2 1 2 2 3
序列5：0 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4

初始元 ： F

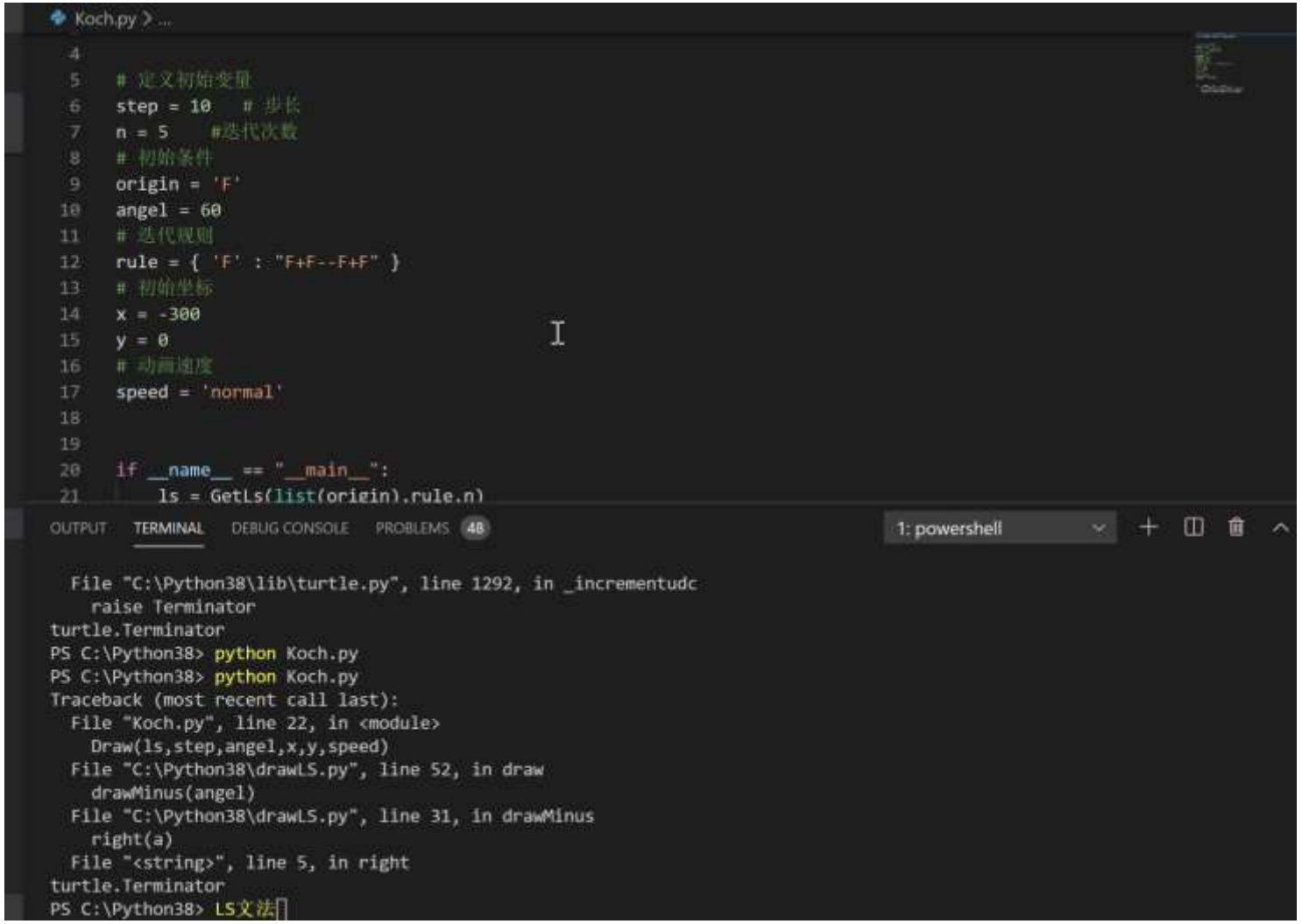
第一级变换： F+F--F+F

第二级变换： (F+F--F+F)+(F+F--F+F)--(F+F--F+F)+(F+F--F+F)

. . .
. . .
. . .

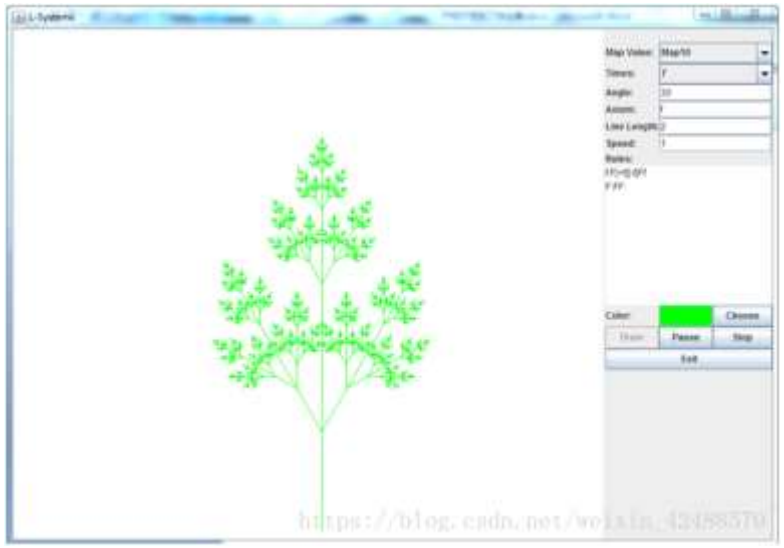


拓展：文法应用-分形


Koch.py > ...
4
5 # 定义初始变量
6 step = 10 # 步长
7 n = 5 #迭代次数
8 # 初始条件
9 origin = 'F'
10 angel = 60
11 # 迭代规则
12 rule = { 'F' : "F+F--F+F" }
13 # 初始坐标
14 x = -300
15 y = 0
16 # 动画速度
17 speed = 'normal'
18
19
20 if __name__ == "__main__":
21 ls = GetLs(list(origin).rule.n)
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 48
1: powershell
File "C:\Python38\lib\turtle.py", line 1292, in _incrementudc
raise Terminator
turtle.Terminator
PS C:\Python38> python Koch.py
PS C:\Python38> python Koch.py
Traceback (most recent call last):
File "Koch.py", line 22, in <module>
Draw(ls,step,angel,x,y,speed)
File "C:\Python38\drawLS.py", line 52, in draw
drawMinus(angel)
File "C:\Python38\drawLS.py", line 31, in drawMinus
right(a)
File "<string>", line 5, in right
turtle.Terminator
PS C:\Python38> LS文法

拓展：

L系统—植物生长建模



拓展：

```
# 定义初始变量
step =10  # 步长
n =3   #迭代次数
# 初始条件
origin = 'F'

angel = 20
# 迭代规则
rule = { 'F' : "F[-F]F[+F]F" }
# 初始坐标
x = 0
y = -250
# 动画速度s
speed = 'fast'
# 初始角度s
head = 90
# 渲染速度
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS        1: powershell

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

PS C:\Python38>
```

2.3 有限自动机

有限自动机(finite automata)

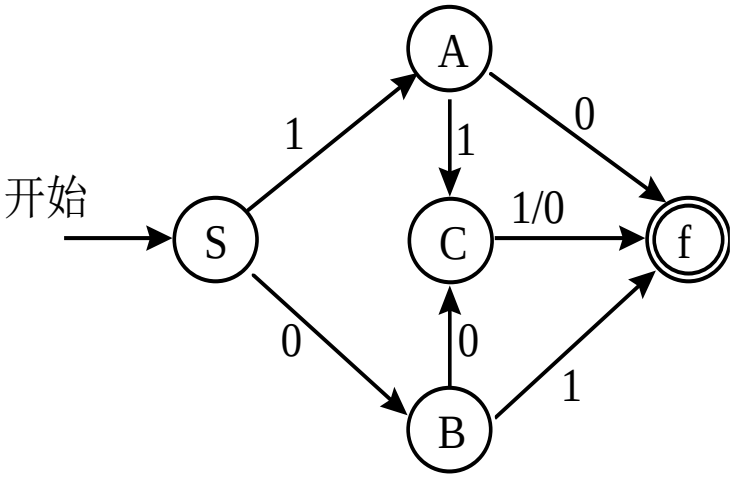
离散数字系统的抽象数学模型，具有离散的输入和输出

有限自动机分类

- 确定的有限自动机(Deterministic Finite Automata,DFA)
- 不确定的有限自动机(Nondeterministic Finite Automata,NFA)

2.3.1确定的有限自动机（DFA）

- 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



状态转换图

利用状态转换图识别符号串

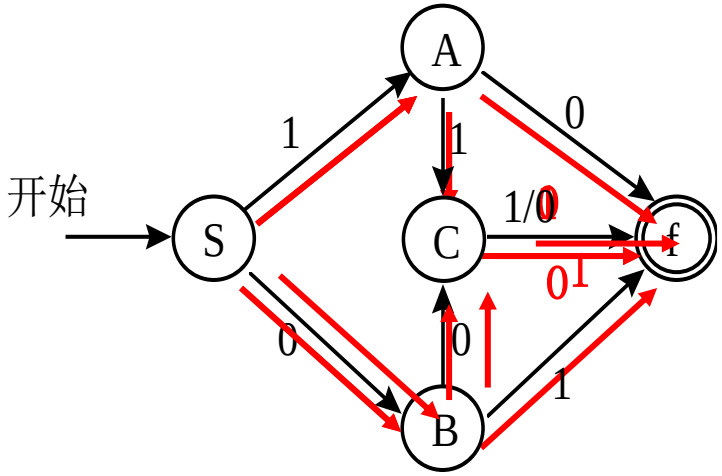
- 识别方法：

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

(2) 扫描w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： $M=(\Sigma, Q, q_0, F, \delta)$

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$Q \times \Sigma$到Q的单值映射

转换函数$\delta(q,a)=q'$（其中$q,q' \in Q, a \in \Sigma$）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$Q=\{q_1,q_2,\cdots,q_n\}$，$\Sigma=\{a_1,a_2,\cdots,a_m\}$，则$Q \times \Sigma=(\delta(q_i,a_j))_{n \times m}$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

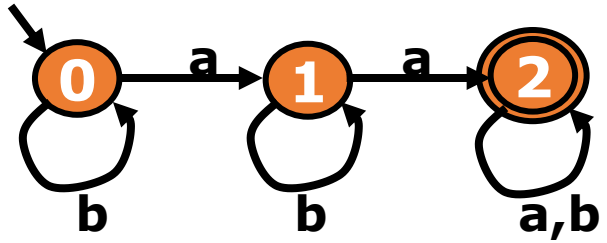
DFAM与状态转换图

DFAM可用一张状态转换图来表示：

- 若q、q′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。
- $\in Q,a\in\sum$，并且$\delta(q,a)=q'$，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。

思考

图中自动机可接受什么样的字符串?



转移函数:

{ (q0,**a**)→q1, (q0,**b**)→q0,
(q1,**a**)→q2, (q1,**b**)→q1,
(q2,**a**)→q2, (q2,**b**)→q2 }

2.3.2 DFA M所识别的语言

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$\omega \in \Sigma^*$，递归地扩张δ的定义：对任何$a \in \Sigma$，$q \in Q$ 定义

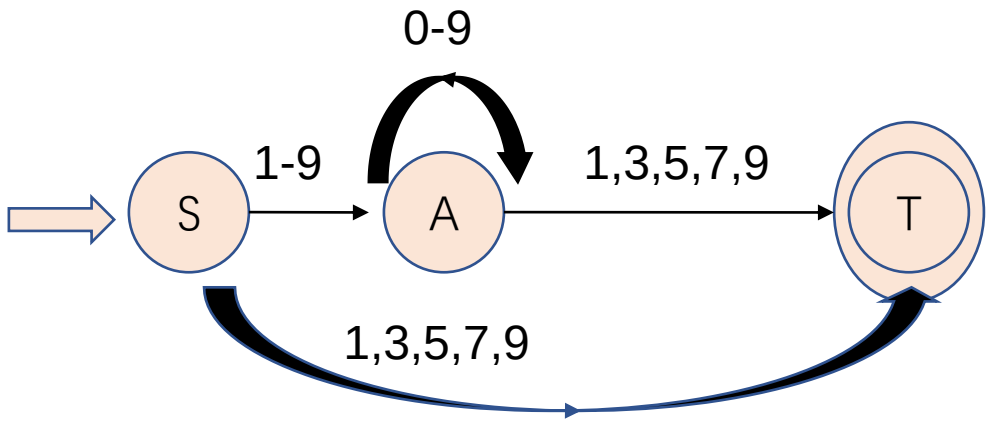
$\delta(q,\varepsilon)=q$

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

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

思考

如下所示的FA，所能识别的语言是什么？



DFA M所识别的语言

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$\omega \in \Sigma^*$，递归地扩张δ的定义：对任何$a \in \Sigma$，$q \in Q$ 定义

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

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

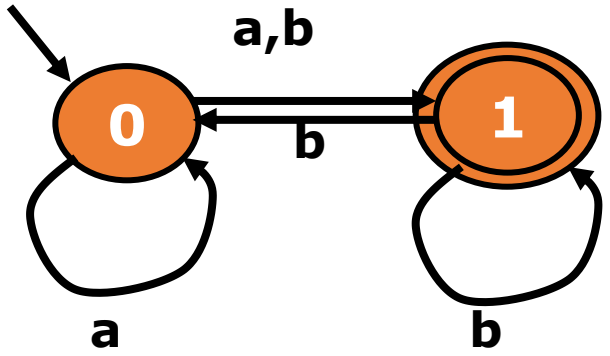
$L(M)=\{\ \omega \mid \omega \in \Sigma^*$，并且存在$q \in F$，使$\delta(q_0,\omega)=q\ \}$

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

转换函数：

- $\{(q0,a)\rightarrow\{q0,q1\},$
  $(q0,b)\rightarrow\{q1\},$
  $(q1,a)\rightarrow\varnothing,$
  $(q1,b)\rightarrow\{q0,q1\}\}$



2.3.3 NFA转化DFA

定理 2.1：

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}

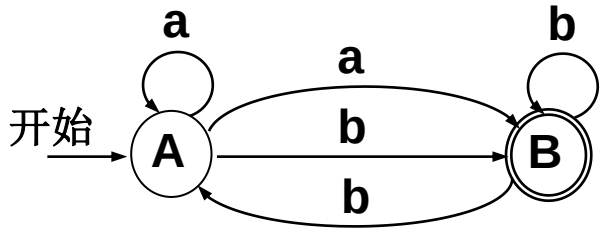
1、转换函数构造法：

首先，画出该NFA M的状态转换图

假设 DFA D=（{a,b},Q′,q0′,F′,δ′），则Q′：Q的所有子集组成的集合，即Q′=$2^{Q,}$ Q′={φ,{A},{B},{A.B}}

q0′={q0},此DFAM中 q0′={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

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

$\delta'(\varphi,a)=\varphi$　　$\delta'(\varphi,b)=\varphi$

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

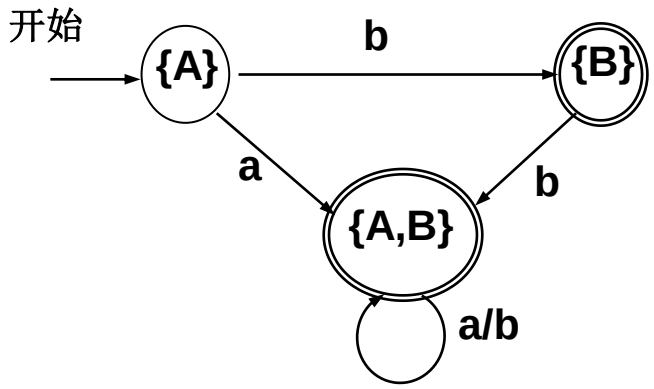
$\delta'(\{B\},a)=\delta(B,a)=\varphi$　　$\delta'(\{B\},b)=\delta(B,b)=\{A,B\}$

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

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

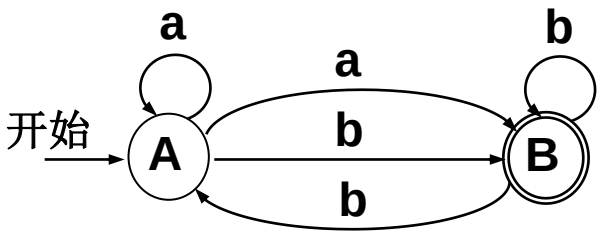
DFA D的状态转换矩阵和状态转换图

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



2、子集构造法：

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。

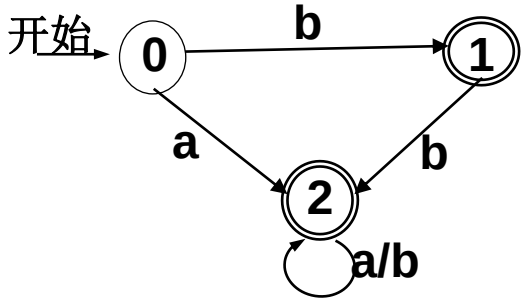


NFA M的状态转换矩阵

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



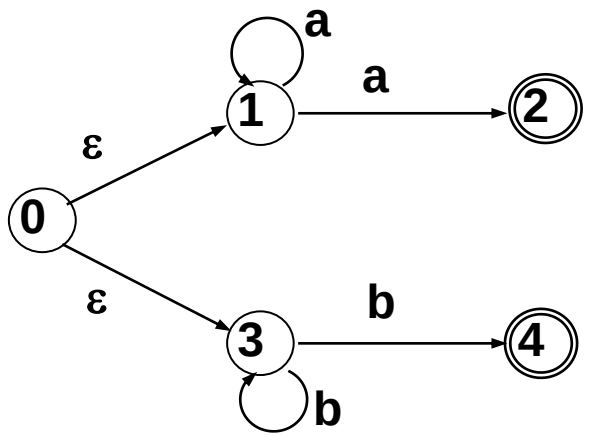
示例

有NFAM=({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)={ $a^+|b^+$ }。

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=(∑，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$Q\times(\sum \cup \{\varepsilon\})$到Q的子集的映射，即$\delta: Q\times(\sum \cup \{\varepsilon\}) \to 2^Q$

对任何$q \in Q$及$a \in (\sum \cup \{\varepsilon\})$，转移函数δ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,...,k)

NFA 的状态转换图

–图中可能有标记为ε的边

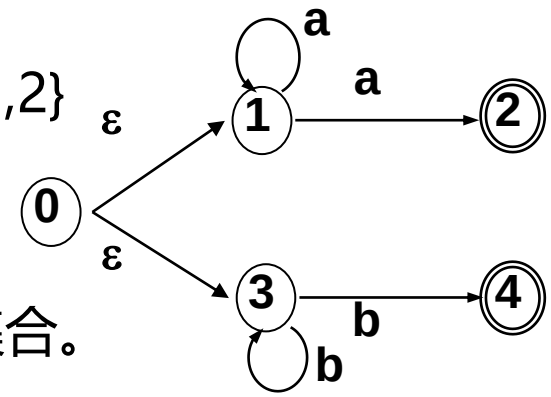
–当$\delta(q,\varepsilon)=\{q_1,q_2,\ldots,q_k\}$时，从q出发有k条标记为ε的边分别指向$q_1,q_2,\ldots,q_k$

定理2.2

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

设NFAM=({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。

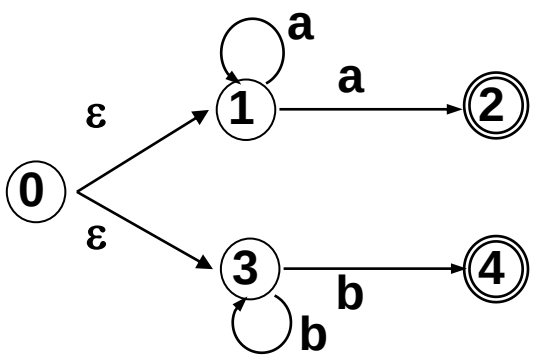
F′的构成：

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

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：

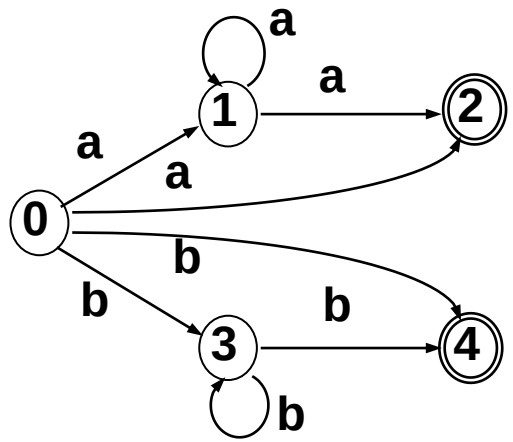
δ′(q,a)={q′| q′为从q出发,经过标记为a 的道路所能到达的状态 }



$$\delta'(0,a)=\{1,2\} \qquad \delta'(0,b)=\{3,4\}$$
$$\delta'(1,a)=\{1,2\} \qquad \delta'(1,b)=\Phi$$
$$\delta'(2,a)=\Phi \qquad \delta'(2,b)=\Phi$$
$$\delta'(3,a)=\Phi \qquad \delta'(3,b)=\{3,4\}$$
$$\delta'(4,a)=\Phi \qquad \delta'(4,b)=\Phi$$

NFA N的状态转换矩阵和状态转换图：

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



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：

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

（2） ε_closure(T)= ε_closure(q_i) 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。

（3）move(T,a)={q | δ(q_i,a)=q，其中$q_i \in T$ }
从某个状态qi∈T出发，经过输入符号a之后可到达的状态集合

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：

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

（2） ε_closure(T)= ε_closure(q_i) 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。

（3）move(T,a)={q | δ(q_i,a)=q，其中$q_i \in T$ }
从某个状态qi∈T出发，经过输入符号a之后可到达的状态集合

算法：由NFA构造等价的DFA

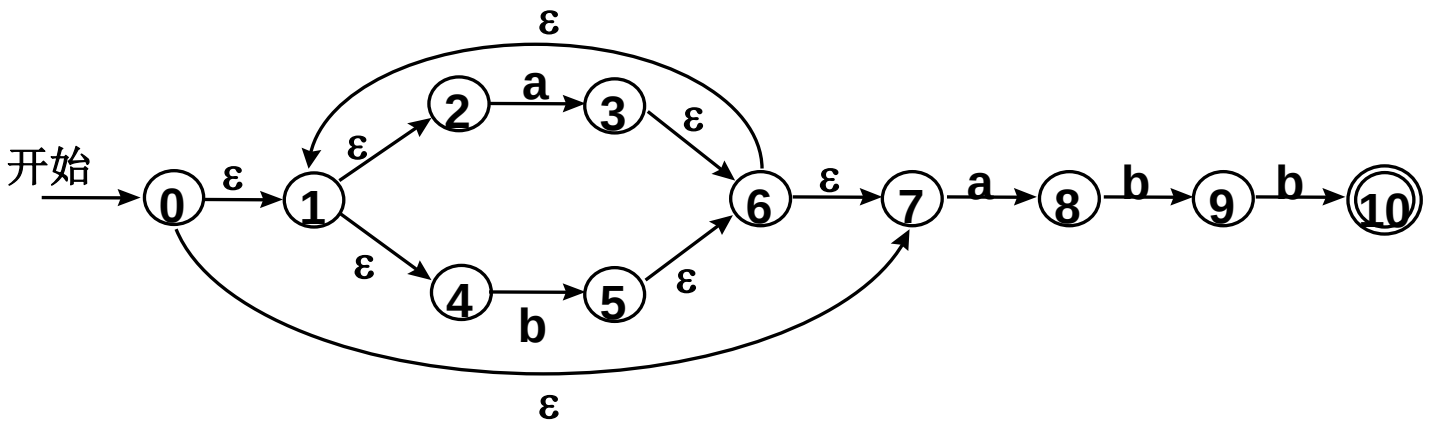
输入：一个NFA M
输出：一个与NFA M等价（即接受同样语言）的DFA D
方法：为DFA D构造状态转换矩阵DTT

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;
    }
}

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;
           }
}
```

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

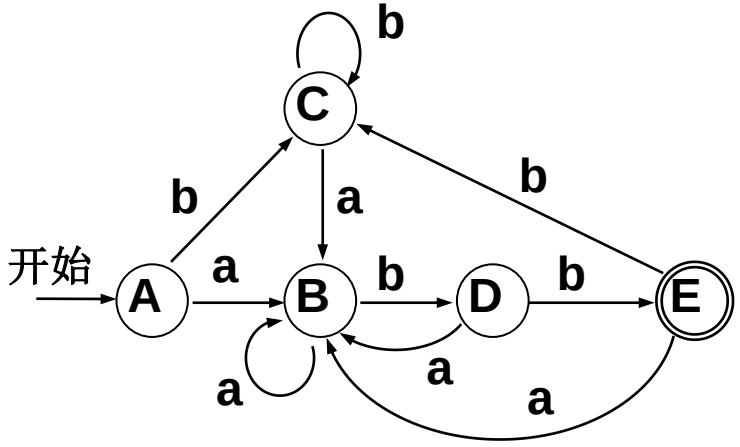
DTT[B,b]=ε-closure(move(B,b))=ε-closure({5,9})={1,2,4,5,6,7,9}=D

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



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;
           }
}
```

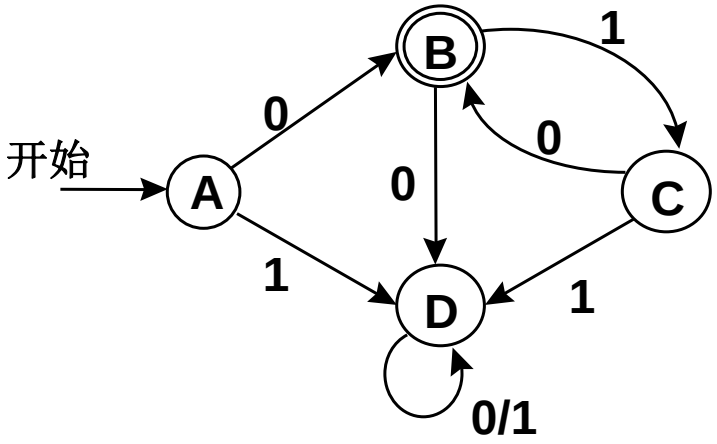
思考

DFA/NFA 如何实现?

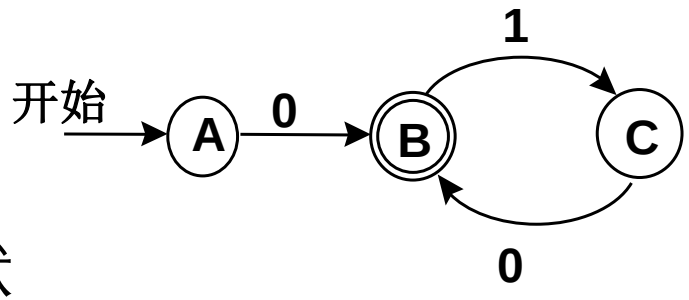
2.3.4 DFA的化简

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



- 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



2.3.4 DFA的化简

状态的区分

定义：设$s,t\in Q$，若对任何$\omega\in\Sigma^*$，$\delta(s,\omega)\in F$ 当且仅当 $\delta(t,\omega)\in F$，则称状态s和t是等价的。否则称状态s和t是可区分的。

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。

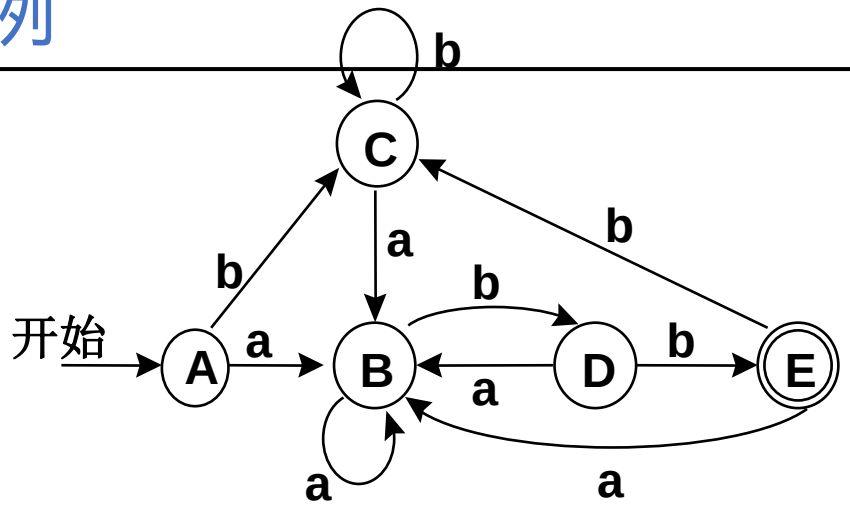
Hopcroft算法

- 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S=$\{s_1, s_2, \cdots, s_k\}$
  - 取某个输入符号c，检查S中的每个状态对该输入符号的转换。
  - 如果S中的状态相对于c，转换到不同子集中的状态，则要对S进行划分。使S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。

```
split(S)
  foreach ( character c )
   if ( c can split S )
   split S into T1, …, Tk

hopcroft ( )
   split all nodes into F, G
   while ( set is still changes ) split(S)
```

示例



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}。

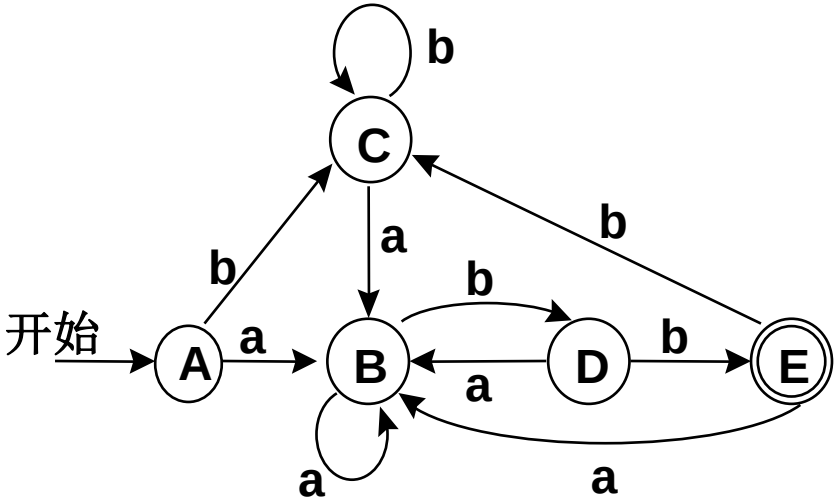
示例

- 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。

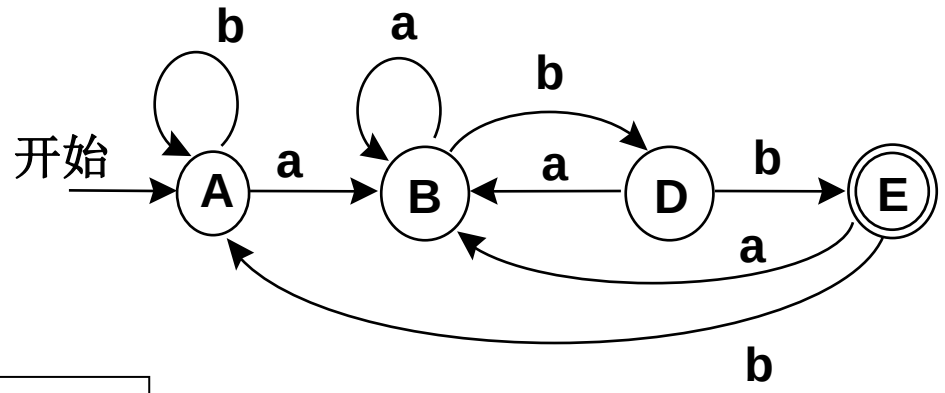
D的状态集合最终被划分为：{A,C}、{B}、{D}和{E}

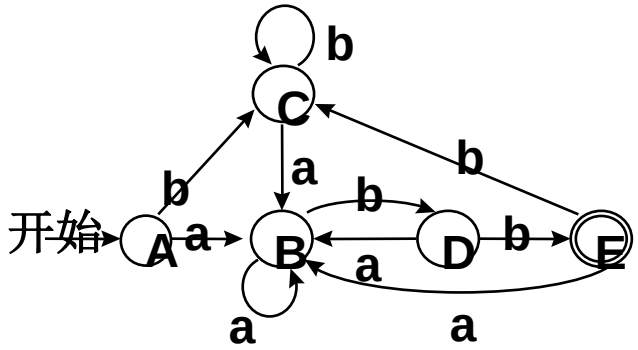


示例

- 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



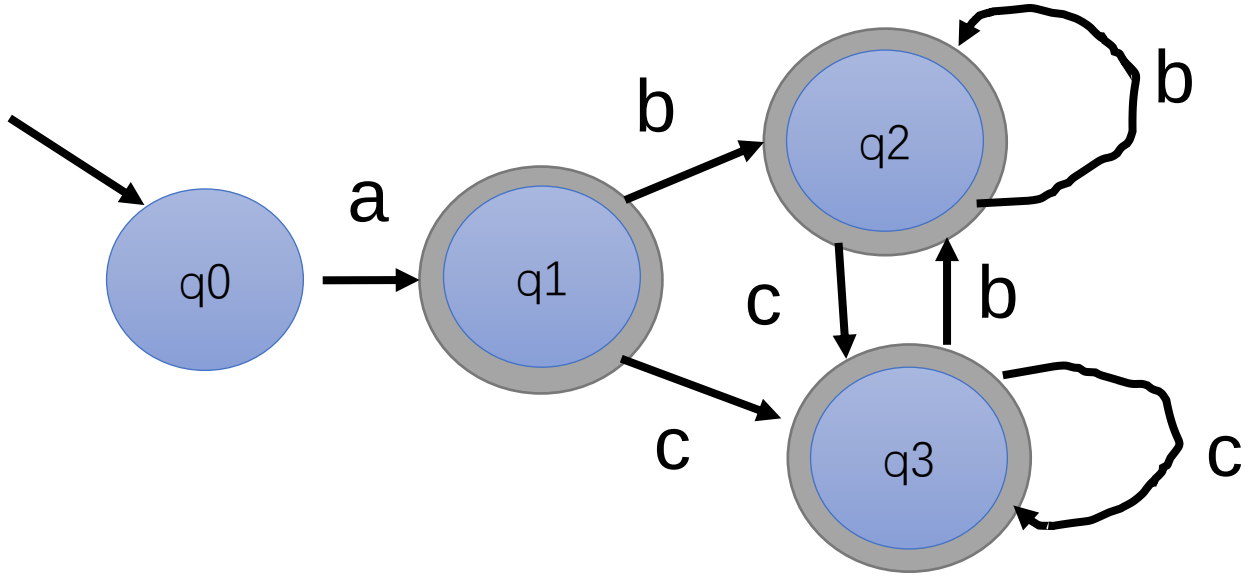


状态	输入符号	
	a	b
A	B	A
B	B	D
D	B	E
E	B	A

课堂练习

请将图示中的DFA最小化

请将图示中的DFA最小化



2.4 文法与自动机

请输入内容

- 0型文法（短语文法）的能力相当于图灵机，可以表征任何递归可枚举集，而且任何0型语言都是递归可枚举的
- 1型文法（上下文有关文法）：产生式的形式为$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$，即只有A出现在$\alpha_1$和$\alpha_2$的上下文中时，才允许β取代A。其识别系统是线性界限自动机。
- 2型文法（上下文无关文法、CFG）：产生式的形式为$A \rightarrow \beta$，β取代A时与A的上下文无关。其识别系统是不确定的下推自动机。
- 3型文法（正规文法、右线性文法）：产生的语言是有穷自动机（FA）所接受的集合

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。

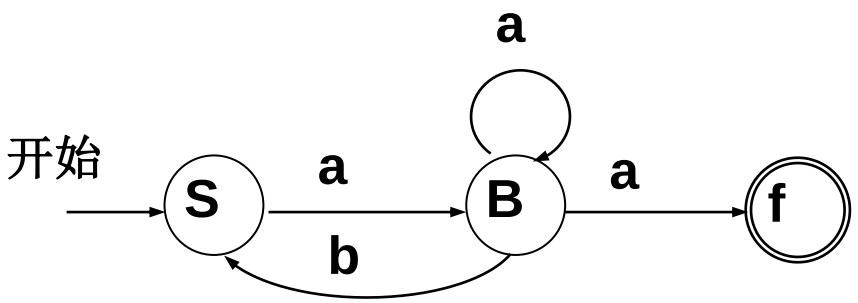
定理：对每一个右线性文法G或左线性文法G，都存在一个等价的有限自动机M。

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′。

正规文法与有限自动机的等价性

示例：
设有右线性文法G=({a,b}, {S,B}, S, φ)，
其中φ：S→aB,B→aB|bS|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→aB，有δ(S,a)={B}
  - 对于产生式B→aB，有δ(B,a)={B}
  - 对于产生式B→bS，有δ(B,b)={S}
  - 对于产生式B→a，　有δ(B,a)={f}
- FA M的状态转换图：



正规文法与有限自动机的等价性

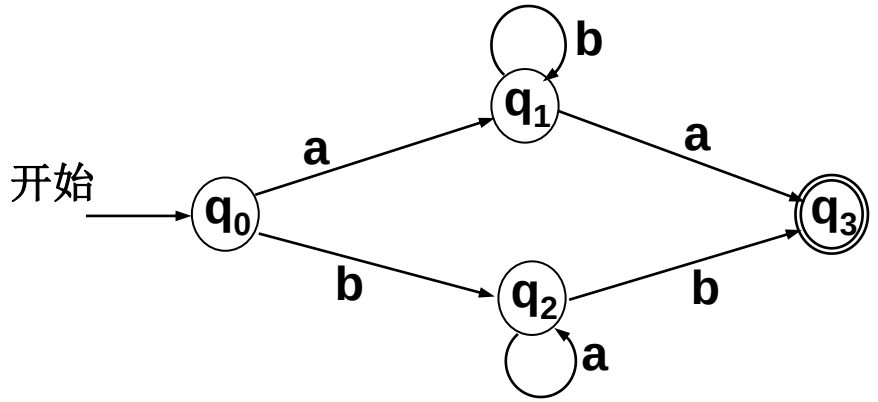
示例：设有DFA M=({a,b}, {q_0,q_1,q_2,q_3},q_0,{q_3},δ)

其中转换函数δ如下：

δ(q_0,a)=q_1， δ(q_1,a)=q_3， δ(q_2,a)=q_2

δ(q_0,b)=q_2， δ(q_1,b)=q_1， δ(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,φ)

V_T={a,b}　　V_N={q_0,q_1,q_2,q_3}　　S=q_0

产生式集合φ

δ(q_0,a)=q_1，∴ q_0→aq_1

δ(q_0,b)=q_2，∴ q_0→bq_2

δ(q_1,a)=q_3，q_3∈F，∴ q_1→a|aq_3

δ(q_1,b)=q_1，∴ q_1→bq_1

δ(q_2,a)=q_2，∴ q_2→aq_2

δ(q_2,b)=q_3，q_3∈F，∴ q_2→b|bq_3

构造的文法G：

G=({a,b},{q_0,q_1,q_2},q_0,φ)

φ：q_0→aq_0|bq_2

q_1→a|bq_1

q_2→aq_2|b

2.5 正规表达式与有限自动机的等价性

2.5.1 正规表达式

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

正规表达式表示的语言叫做正规集。

(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：

(r)|(s) 是正规表达式，表示的语言是L(r)∪L(s)

(r)(s) 是正规表达式，表示的语言是L(r)L(s)

(r)* 是正规表达式，表示的语言是(L(r))*

2.5 正规表达式与有限自动机的等价性

2.5.1 正规表达式

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

正规表达式表示的语言叫做正规集。

(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：

(r)|(s) 是正规表达式，表示的语言是L(r)∪L(s)

(r)(s) 是正规表达式，表示的语言是L(r)L(s)

(r)* 是正规表达式，表示的语言是(L(r))*

2.5.1 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

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)

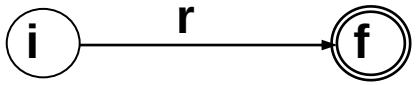
2.5.2 正规表达式遵从的代数定律

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

2.5.3 正规式与有限自动机

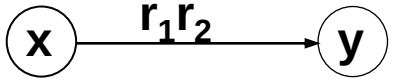
定理：对任何一个正规表达式r，都存在一个FA M，使L(r)=L(M)，反之亦然。

- 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。

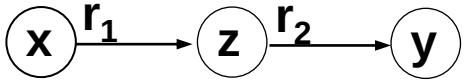


然后，按照下面的转换规则，对正规表达式r进行分裂、加入新的结点，直到每条边的标记都为基本符号为止。

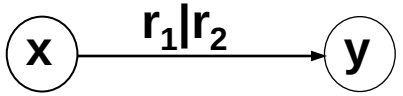
若$r=r_1r_2$，则将：



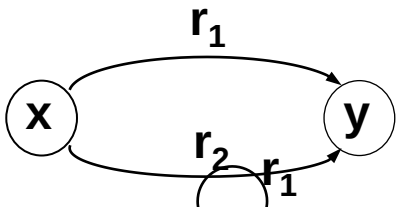
代之以：



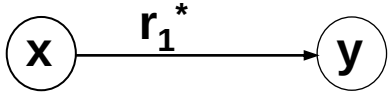
若$r=r_1|r_2$，则将：



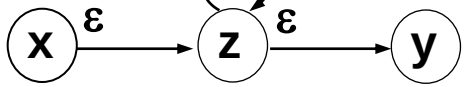
代之以：



若$r=r_1^*$，则将：



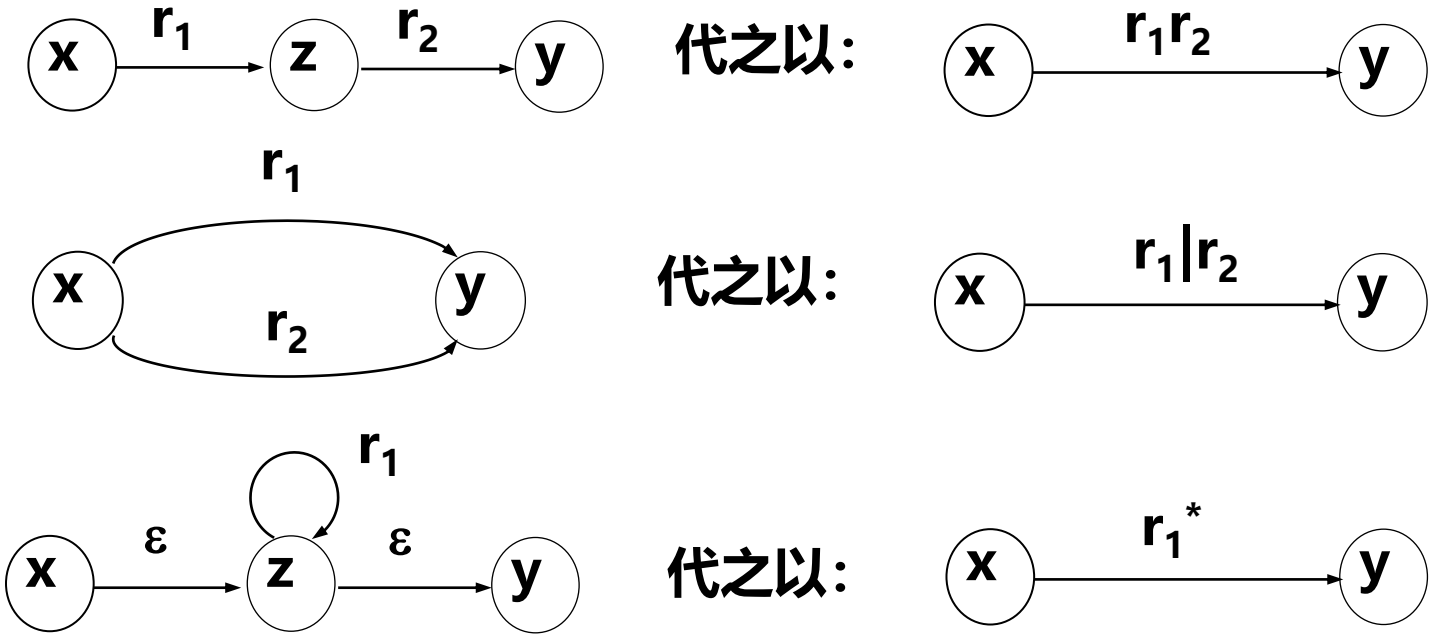
代之以：



2.5.3 正规式与有限自动机

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

- 首先，在FAM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。



2.5.4 RE到NFA——Thompson算法

- 对RE的结构进行归纳

- 对基本的RE直接构造
- 对复合的RE递归构造

2.5.4 RE到NFA——Thompson算法

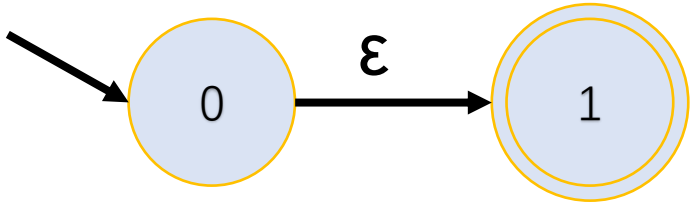
e → ε

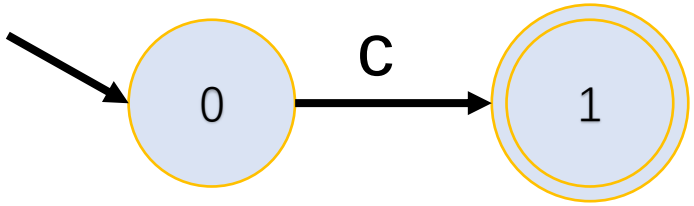
e → c

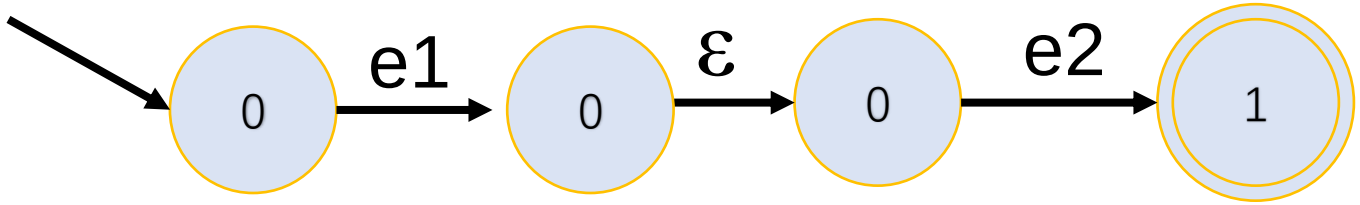
e → e1 e2

e → e1 | e2

*e → e1**







2.5.4 RE到NFA——Thompson算法

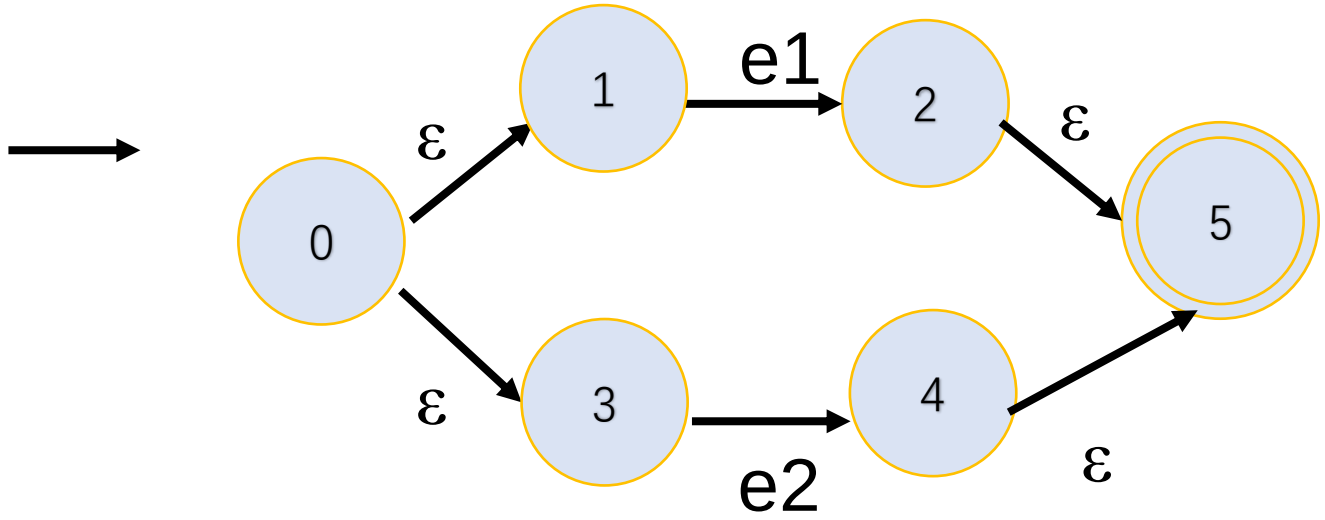
e → ε

e → c

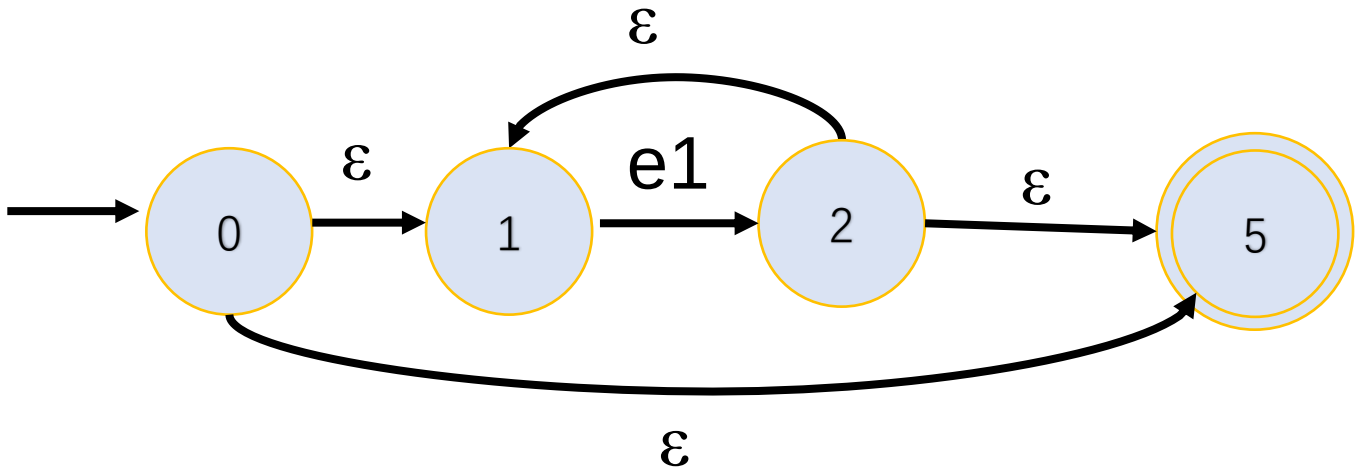
e → e1 e2

e → e1 | e2

e → e1*

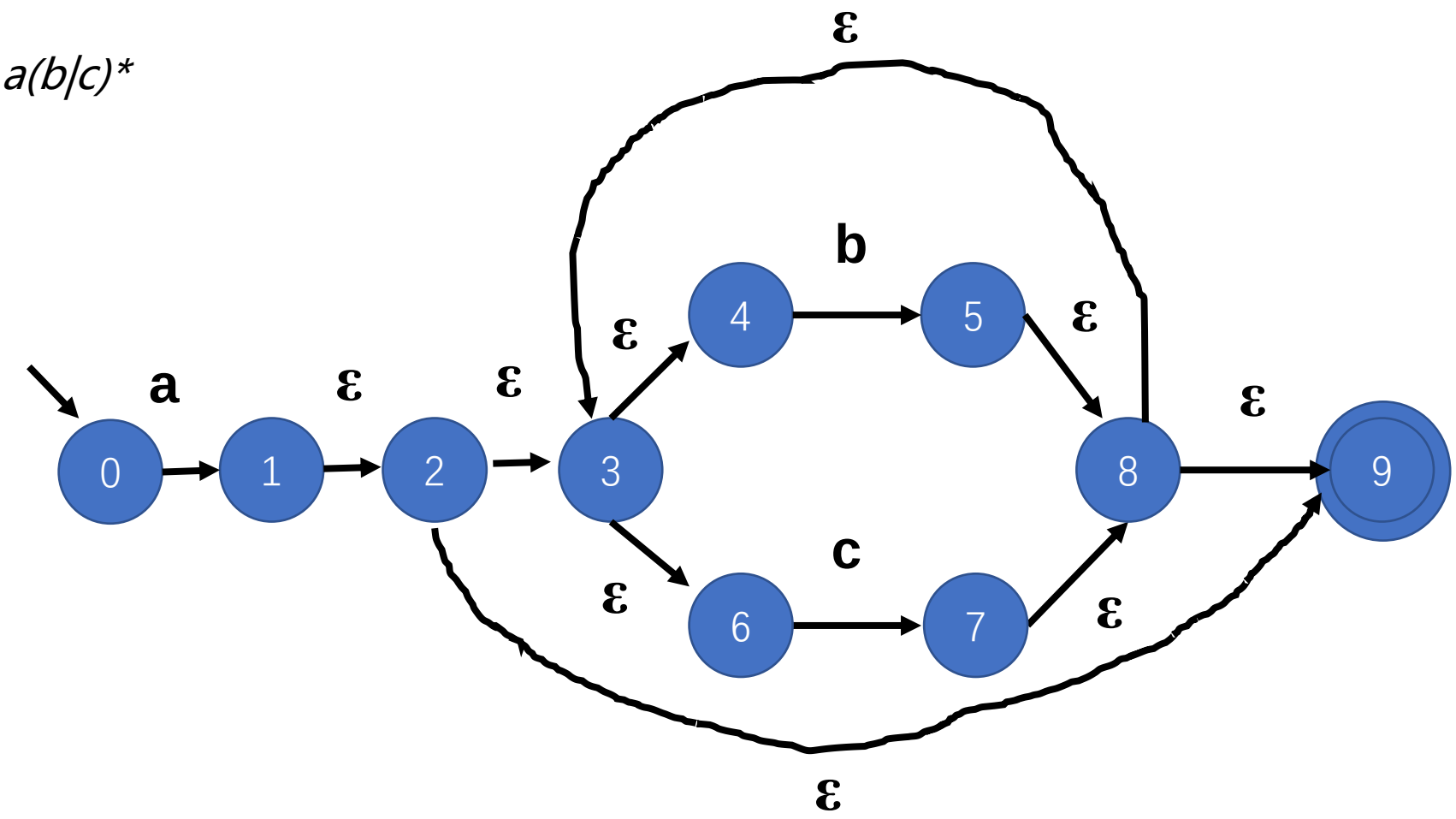


?

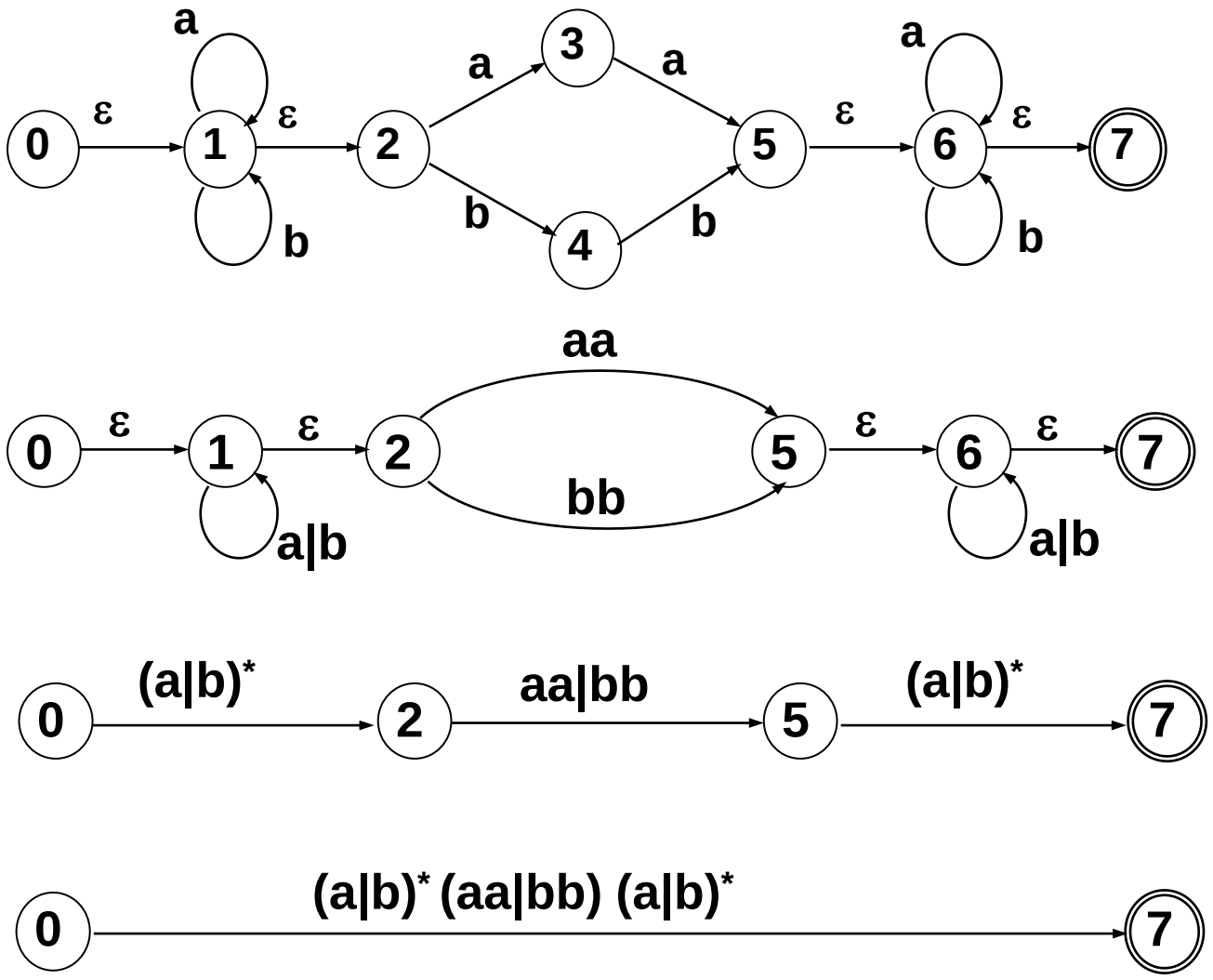


2.5.4 RE到NFA——Thompson算法

*a(b|c)**



示例：构造与如下的NFAM等价的正规表达式



2.5.5 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。

2.5.5.1 正规定义式

定义：令Σ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$\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$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*

2.5.5.1 正规定义式

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 ‘?’

$r?=r|\varepsilon$

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]*

2.5.5.1 正规定义式

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。

2.5.5.1 正规定义式

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)*

正规定义式：

letter → A|B|...|Z|a|b|...|z

digit → 0|1|...|9

id → letter(letter|digit)*

2.5.5.1 正规定义式

分析：

为子表达式(letter|digit)* 取一个名字rid，展开第三个正规定义

(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：

$id \rightarrow$ A rid |B rid |…|Z rid |a rid |b rid |…|z rid

$rid \rightarrow \varepsilon$ |A rid |B rid |…|Z rid |a rid |b rid |…|z rid
|0 rid |1 rid |…|9 rid

2.6 DFAM的实现及应用

2.6.1 DFA表示方式

- 状态转换表
- 状态转换图
- 映射函数集合
- 状态转换矩阵

2.6 DFAM的实现及应用

2.6.2 DFA实现方式

实际中，有不同的DFA代码表示

转移表（类似于邻接矩阵）

哈希表

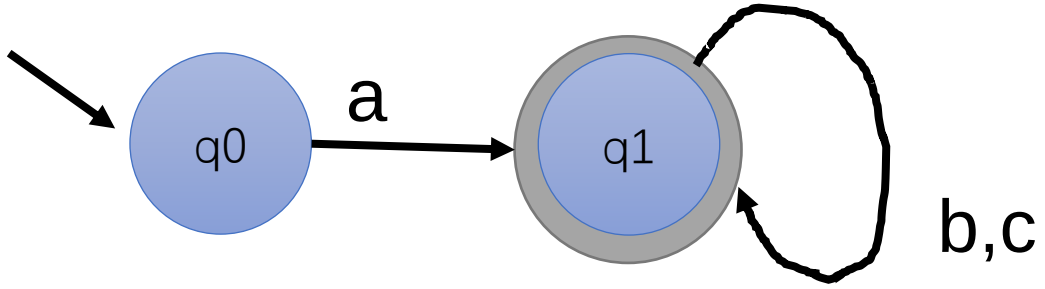
跳转表

…

取决于在实际实现中，对时间空间的权衡

2.6 DFAM的实现及应用

转移表实现



状态/字符	a	b	c
0	1		
1		1	1

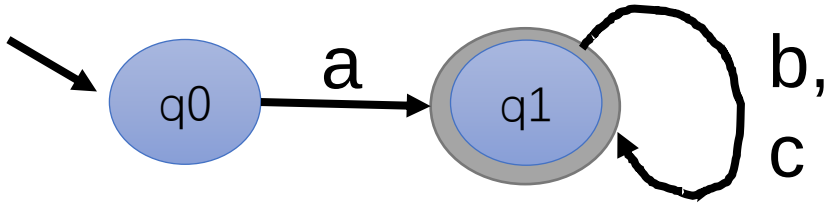
```
char table[M][N];

table[0]['a']=1;
table[1]['b']=1;
table[1]['c']=1;

  // other table entries
```

2.6 DFAM的实现及应用

跳转表实现



状态/字符	a	b	c
0	1		
1		1	1

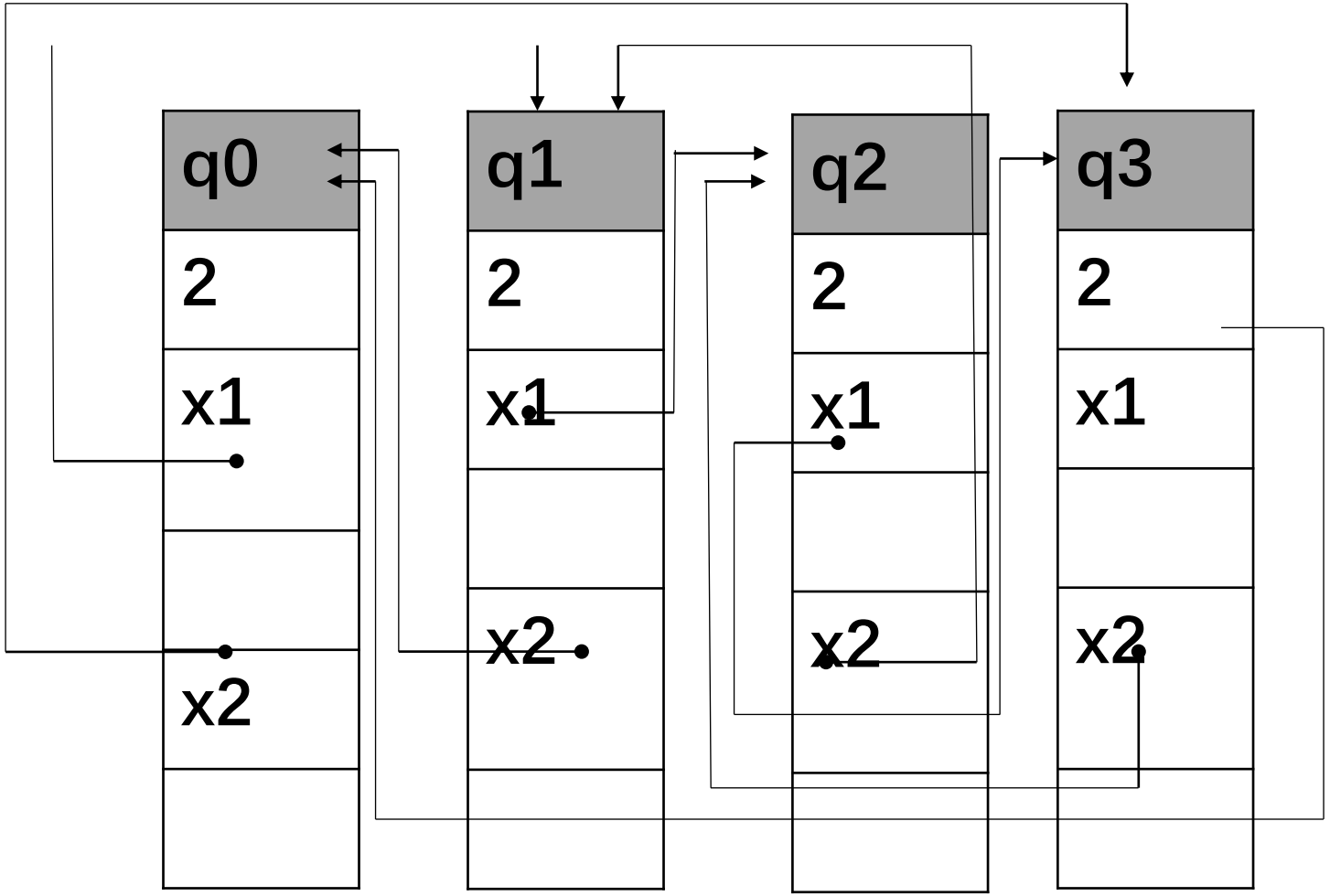
```
nextToken()
  state = 0
  stack = []
  goto q0
```

```
q0:
 c = getChar()
 if (state is ACCEPT)
    clear (stack)
 push (state)
 if (c==‘a’) goto q1:
```

```
q1:
 c = getChar()
 if (state is   ACCEPT)
     clear (stack)
 push (state)
 if (c==‘b’||c==‘c’)
     goto q1
```

2.6 DFAM的实现及应用

链表实现



2.6.3 DFA应用

示例

某公共汽车始发站，在始发站有乘客的条件下，每隔一固定时间段发出一班车。车站发出一班车后，如果在下一发车时刻车站没有乘客，则停一班次，当再等到下一发车时刻时，不管始发站有没有乘客，则车站必发出一班车 。

用有限自动机刻画这一过程 ： 设 x0 ， x1 分别表示始发站没有乘客和有乘客 ，为自动机的输入信息 ； 自动机的状态有两个 ， 分别为上一发车时刻没有发车和发车两个状态 ，我们分别用 q0 ， q1 表示这两个状态， q0为初始状态。

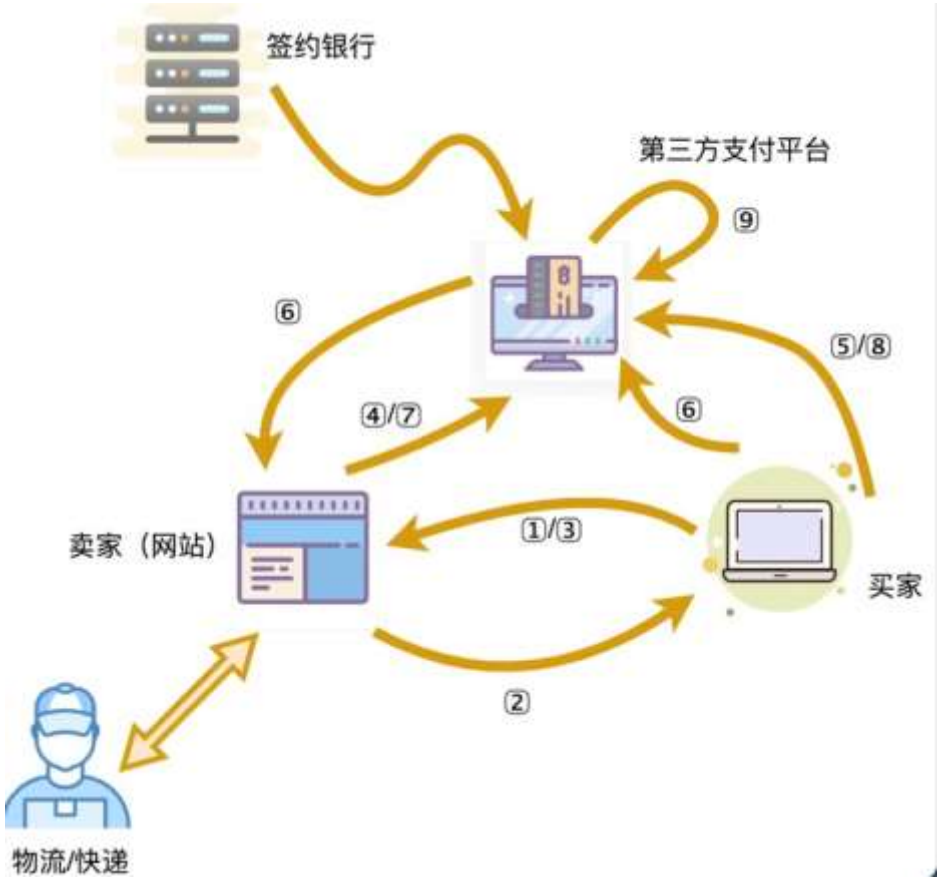
则有限自动机M=（Q, Σ, δ,q0,F）可表述如下:

Q={ q0,q1}, Σ={x0,x1}, F⊆ Q

δ ={ δ(q0,x0)=q1, δ(q0,x1)=q1, δ(q1,x0)=q0, δ(q1,x1)=q1}

2.6.3 DFA应用

DFA在网上购物平台中的应用



交易具体步骤：

1. 商品浏览
2. 商品推送
3. 订单确认
4. 订单提交
5. 贷款支付
6. 支付响应
7. 发货确认
8. 收货确认
9. 交易完成

课堂练习

请给出描述上述网上购物流程的DFA。

操作	动作描述
e1	买家下单
e2	买家付款
e3	买家取消订单
e4	买家确认收货
e5	买家退货
e6	卖家发货
e7	卖家确认退款
e8	卖家取消订单
e9	其他操作

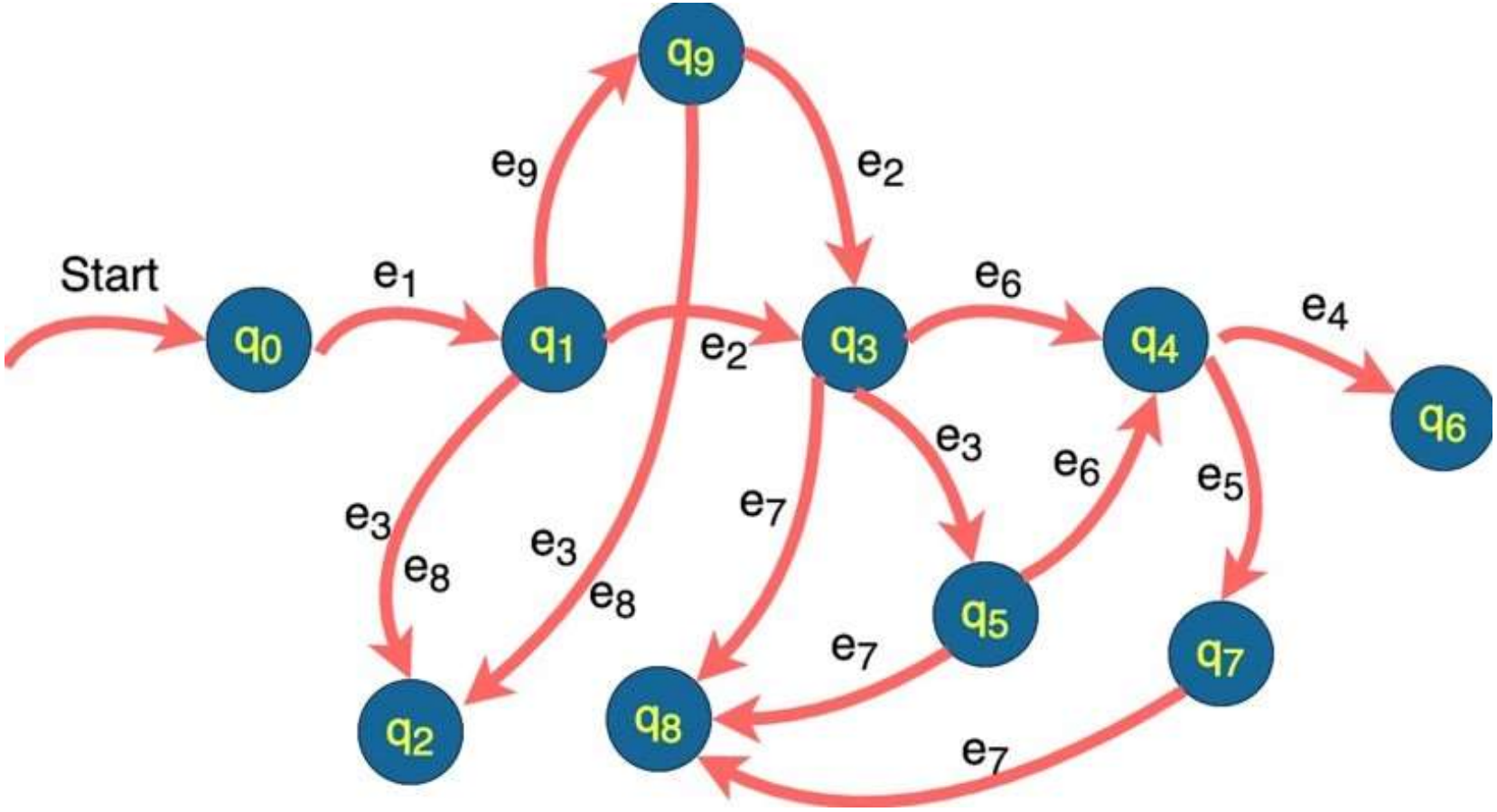
交易操作描述/输入

状态	状态描述
q0	开始状态，等待订单
q1	已下单状态
q2	终结状态1，订单取消
q3	已付款状态
q4	已发货状态
q5	申请退款状态
q6	终结状态2，已收货状态
q7	已退货状态
q8	终结状态3，确认退款状态
q9	其他状态

DFA状态描述

2.6.3 DFA应用

购物流程DFA状态图



2.6.3 DFA应用

实际应用

设备管理：车辆管理
业务流程管理：航班调度
嵌入式菜单：嵌入式HTML解析器
自然语言识别：微博舆情监控

作业 1

1、已知语言描述，写出产生该语言的文法。

语言由0、1符号串组成，串中0和1的个数相同，构造描述该语言的文法。

2、已知文法，写出所产生语言的描述。

G：S→a

S→aB

B→aS

作业 2

根据文法规则，扩展编程作业3，实现正整数运算（加、减、乘、除）的解释器。

输入：正整数四则运算表达式

输出：运算结果

作业 3

构造识别输入串 （a|b)*a(a|b)* 的DFA

作业 4

构造识别输入串 （a|b)*a(a|b)* 的DFA

作业 5

美团共享电动车
快递收发柜
扫地机器人
无人驾驶汽车
…
请给出一个应用实例，并用DFA进行描述。

课后扩展 1

ABNF文法是扩展的BNF，在互联网领域应用广泛，请阅读相关资料。

http://oss.org.cn/man/develop/rfc/RFC2234.txt).

科大讯飞ABNF文法规范

科大讯飞翻译器

课后扩展 2

课后请查阅资料并思考：文法还能应用于哪些领域？你能给出一些实例吗？

课后扩展 3

课后请阅读：文法实验解读。