第2章 词法分析

- 2.1 词法分析概述
- 2.2 正则表达式
- 2.3 有限自动机
- 2.4 词法分析程序的设计与实现

2.1 词法分析概述

- 2.1.1 词法分析的任务
- 2.1.2 单词分类
- 2.1.3 Token结构

2.1.1 词法分析的任务

- 词法分析主要有三个任务:
- （1）将字符串形式的源程序分割成一个一个子串，其中每个子串称为一个“单词”
- 源程序在词法分析前:

```
if (position > 10) rate = 3.14 * initial;
```

- 源程序在词法分析后:

if	(	position	>	10	)	rate	=	3.14	*	initial	;

- 思考：为什么这样分割？是否有其它分割方案?
- 如何进行分割？用空格或空白分割是否可行?

2.1.1 词法分析的任务

- 词法分析主要有三个任务:
- （2）将每个子串表示成便于后续使用的形式—Token(自定义的数据结构)
- 词法分析后得到的单词序列:

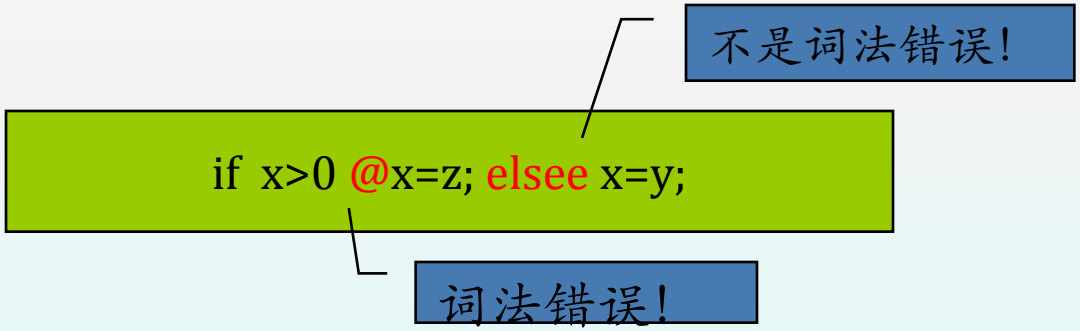
if	(	position	>	10	)	rate	=	3.14	*	initial	;

- 表示成Token形式的单词序列:

(if,-)	((,-)	(id, position)	(gt,-)	(num,10)	( ),-)
(id,-)	(ass,-)	(num, 3.14)	(*,-)	(id, initial)	(;,-)

2.1.1 词法分析的任务

- 词法分析主要有三个任务:
- （3）报告词法错误
- 词法错误的种类很少，常见的一类是源程序中出现了源语言字母表以外的字符



- 思考：编译时发现了词法错误，是立即停止好，还是采取某些措施继续向下分析好?

2.1.2 单词分类

- 对于程序而言，一个运算符、一个标识符、一个数值常量、一个格式符或者一个保留字等，是具有语义且不可分割的组成部分，称为“单词”.单词是最小的语义单位.

```
if (position > 10) rate = 3.14 * initial;
```

- if 是单词　属于保留字
- (　是单词　属于分隔符
- position　是单词　属于标识符
- >　是单词　属于运算符
- 10　是单词　属于整型常数
- 3.14　是单词　属于实型常数
- \n　是单词　属于格式符

2.1.2 单词分类

- 常见单词种类:
- 保留字(关键字):
  if ,while,int,float,include,define,class……
- 标识符： x, y,add,A,person,student
- 运算符： +, -, *, /, ++, >, <, !=, &&,……
- 格式符： \t, \r, \n, 空……
- 分隔符： {,}, [,],(,),;,,,:
- 字面常量： 120, 3.14, ‘a’, “hello!”
- 思考：你使用过的程序语言中有单词不能归入这几类吗？你能想出新的单词种类吗？你能找出每类单词的构成特点吗？每类单词的构成是否相同？可否根据每类单词的构成特点识别单词？

2.1.3 Token结构

- Token是单词的内部表示(数据结构)
- Token结构一般包含两部分：(token-type,attribute-value)
  - token-type:用以记录单词的种类
  - attribute-value: 用以记录单词的语义信息

(if,-)	((,-)	(id, position)	(gt,-)	(num,10)	( ),-)
(id,-)	(ass,-)	(num, 3.14)	(*,-)	(id, initial)	(;,-)

2.1.3 Token结构

- 练习：按照前面给出的Token结构，写出下列单词的Token表示.
- 1. void main() { printf("Hello world!\n"); }
- 2. bool prime(int n){//This is an annotation

```
    int j;
    for (j=n/2; j>=2; j--)
       if (n % j ==0) return false;
    return true;
}
```

思考：制表符、回车、换行、空格等格式符需要表示成Token结构吗？程序中的注释呢？

2.2 正则表达式

- 2.2.1 什么是正则表达式
- 2.2.2 正则表达式的应用
- 2.2.3 正则定义
- 2.2.4 正则表达式的局限性

2.2.1 什么是正则表达式

- 基本概念
- 字母表：符号的非空有穷集合,记为Σ, 其中的符号称为字母或符号.
  - 二进制字母表{0,1}
  - 十进制字母表{0,1,2,3,4,5,6,7,8,9}
  - ASCII
  - Unicode
- 符号串：字母表中的符号组成的有穷序列,常用α,β,γ表示
  - 二进制符号串：1001101, 110, 101100……
  - 十进制符号串：211,985,973, 2016……
  - 英文字符串：This is a test.
  - 中文字符串：今天天气不错!
  - C语言字符串：#include “stdio.h” void main(){printf(“Hello!”);}
  - 特例：空串,不含任何符号的串，记为ε或λ，空串ε不等于空集ϕ

2.2.1 什么是正则表达式

- 符号串的长度：符号串中所包含的符号的个数.若α表示符号串,则|α|表示α串的长度.
  - α=abc, |α| = 3，或记为|abc| = 3
  - α=a, |α| = 1，或记为|a| = 1
  - α=ε, |α| = 0，或记为|ε| = 0
- 符号串的联接：设α,β是符号串,则α•β表示α串与β串相联接，即β串中的字符依次接在α串的尾部. α•β也可直接写作αβ
  - α=abc, β=123, 则α•β=abc123
  - α=abc, β= ε, 则α•β=abc
  - α=abc, β=123, 则β•α=123abc
  - α=abc, β= ε, 则β•α =abc

2.2.1 什么是正则表达式

- 符号串集：符号串的集合,一般记为A,B,C
- 符号串集的乘积：若串集A={α_1, α_2, ……, α_n},串集B= {β_1, β_2, ……, β_m},则乘积AB={$\alpha\bullet\beta$|其中$\alpha\in$A, $\beta\in$B }.
  - A={abc,123}, B={e,f,g}, AB= {abce,abcf,abcg,123e,123f,123g}
  - eabc∈AB? abe ∈AB? 13e ∈AB? No!No!No!
  - A={abc,123}, B={e, ε}, AB= {abce,abc,123e,123},BA={eabc,e123,abc,123}
  - A={abc,123}, B={ε}, AB= {abc,123}
  - A={abc,123}, B={ }, AB= {abc,123}
- 符号串集的方幂：设A是符号串集,i是非负整数,则A^i表示i个A相乘,称为A的i次方幂的.
  - A^0={ε} A^1=A A^2=AA……
  - A={ab} A^0={ε} A^1={ab} A^2=AA={abab} A^3=AAA={ababab}

2.2.1 什么是正则表达式

- 符号串集的正闭包：记为A^+
- $A^+ = A^1 \cup A^2 \cup A^3 \cup \ldots\ldots$
- A={ab}, A^+ ={ab,abab,ababab,……}
- A={a,c},
- A^+ ={a,c,aa,ac,ca,cc,aaa,aac,aca,acc,caa,cac,cca,ccc,……}
- 符号串集的星闭包：记为A^*,又称自反闭包或Kleene闭包
- $A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \ldots\ldots$
- A={ab}, A^* ={ε,ab,abab,ababab,……}
- A={a,c}, A^* ={ε, a,c,aa,ac,ca,cc,aaa,aac,aca,acc,caa,cac,cca,ccc,……}

2.2.1 什么是正则表达式

- 正则表达式的形式定义:

∅是RE

ε是RE

任何c ∈Σ，c是RE

若A,B都是RE,则

(A)是RE

A|B是RE

A•B是RE

A^*是RE

- 正则表达式看似符号串,却不是符号串!

2.2.1 什么是正则表达式

- 正则表达式定义的语言
- 对于给定的字母表Σ,Σ上的一个正则表达式r定义了Σ上的一个字符串集合,这个集合称为r定义的语言，记为$L(r)$. $L(r)$也称为正则集.

r:	$L(r)$:
$\varnothing$	$\varnothing$
ε	$\{\varepsilon\}$
c	$\{c\}$
(A)	$L(A)$
A|B	$L(A) \cup L(B)$
A•B	$L(A)•L(B)$
A^*	$L(A)^*$

2.2.1 什么是正则表达式

- 例如：
- r = ab*, L(r) = {a,ab,abb,abbb,abbbb,……}
- r = a*bc*, L(r) = {b,ab,bc,aab,abc,aabc,abcc,……}
- r = (a|b)*c, L(r) = {c,ac,bc,aac,bbc,abc,bac,……}
- r = (0|1)(0|1)*, L(r) = {0,1,00,01,10,11,000,001,010,011,100,101,110,111,……}
- r = 1(0|1)* | 0,
- L(r) = {0,1,10,11,100,101,110,111,……}

2.2.1 什么是正则表达式

- 正则表达式的等价：若两个正则表达式r,s所定义的语言相同，则说r和s是等价的.
  - (a|b)*和(b|a)*定义的语言相同，因此二者等价
- 正则表达式的性质
  - r | s = s | r
  - r | (s | t) = (r | s) | t
  - r(st) = (rs)t
  - r(s | t) = rs | rt
  - (r | s) | t = rt | st
  - r** = r*
  - r ε = ε r = r

2.2.2 正则表达式的应用

- 日常生活中很多涉及到符号串的地方都可以用正则表达式加以描述：例如网络协议、文件访问规则、财务报表、电子表单等.
- 例：给出以下形式Email地址的正则表达式定义
  - xiaoyang2011@163.com 或
  - xiaoyang2011@jlu.edu.cn或
  - 12345@qq.com
- ∑ = {A,…Z,a,…z,0,…9,@, . }
- Names = (Letter | Digit) (Letter | Digit) *
- Address = Names@(Names.| ε) Names.Names
- 或：　　Names@Names.Names(ε | .Names)

2.2.2 正则表达式的应用

- 应用正则表达式定义程序设计语言的单词
  - 保留字: if, while,include,int, char, void,……
  - 标识符: (A|B|…Z|a|b|…|z)(A|B|…Z|a|b|…|z|0|1|2|3|4|5|6|7|8|9)*
  - 整型常量:(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)* | 0
  - 实型常量: ((1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*|0).(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*
  - 运算符: +,-,*,&&,>,<,>=,……
  - 分隔符: {,},{,},(,),;,:,,,.

2.2.3 正则定义*

- 正则定义: 在必要的时候，可以给正则表达式的一些子表达式起个名字，以减少正则表达式的长度，从而提高正则表达式的可读性
- 例：标识符的正则表达式:
  (A|B|...Z|a|b|...|z)(A|B|...Z|a|b|...|z| 0|1|2|3|4|5|6|7|8|9)*
- 可定义 Letter = A|B|...Z|a|b|...|z
- Digit = 0|1|2|3|4|5|6|7|8|9
- 则标识符的正则表达式可写成：Letter Digit*
- 再比如：整型常量的正则表达式:
  (1|2|3|4|5|6|7|8|9)(0| 1|2|3|4|5|6|7|8|9)*|0
- 可定义 NZ_Digit = 1|2|3|4|5|6|7|8|9
- 则整型常量的正则表达式可写成：NZ_Digit Digit* | 0

2.2.4 正则表达式的局限性

- 正则表达式可以以简洁的数学形式，定义复杂的字符串结构.可认为正则表达式是一个模板，能够定义符合某种特性的语言.
- 但是正则表达式的表达能力是有限的！不能定义下列两种结构的语言：
  - 对称结构: $\mathbf{A = \{a^n b a^n \mid n > 0\}}$,
  - 嵌套结构: **1) n ∈AE 2) (AE) ∈AE 3) AE+AE ∈AE**

2.3 有限自动机

- 2.3.1 非确定有限自动机
- 2.3.2 确定有限自动机
- 2.3.3 有限自动机与正则表达式

2.3.1 非确定有限自动机

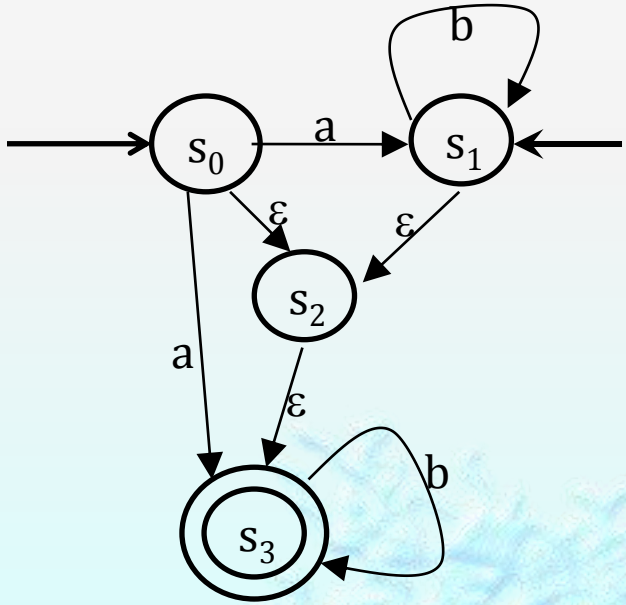
- 定义：非确定有限自动机NFA是一个五元组(S,Σ,f,S_0,Z)，其中：
  - S是有限状态集，不能为空
  - Σ是有穷字母表，可以为空
  - f是状态转换函数，f：$S\times\Sigma\cup\{\varepsilon\}\rightarrow 2^S$
  - S_0是初始状态集，$S_0\subseteq S$，不能为空
  - Z 是终止状态集，$Z\subseteq S$，可以为空

2.3.1 非确定有限自动机

- NFA的例子: $M_1=\{\{s_0,s_1,s_2,s_3\},\Sigma=\{a,b\},f,\{s_0,s_1\},\{s_3\}\}$,其中
- $f(s_0,a)=s_1$, $f(s_0,a)=s_3$, $f(s_0,\varepsilon)=s_2$,
- $f(s_1,b)=s_1$, $f(s_1,\varepsilon)=s_2$,
- $f(s_2,\varepsilon)=s_3$,
- $f(s_3,b)=s_3$

	a	b	ε
s_0^+	$\{s_1, s_3\}$		$\{s_2\}$
s_1^+		$\{s_1\}$	$\{s_2\}$
s_2			$\{s_3\}$
s_3^-		$\{s_3\}$	

M_1的转换表表示



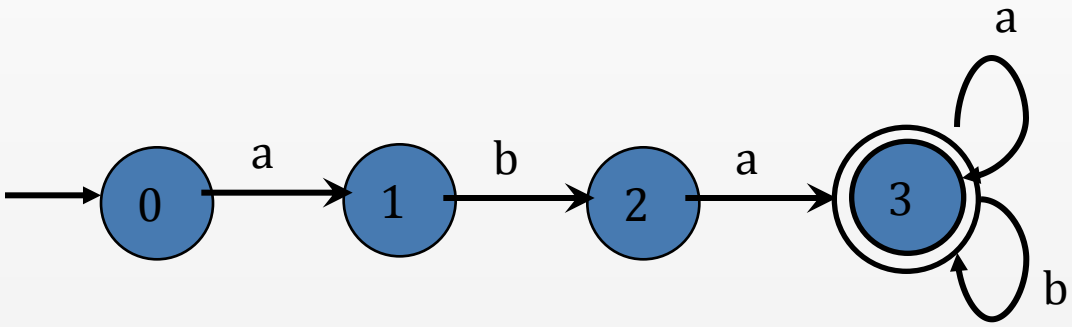
M_1的转换图表示

2.3.1 非确定有限自动机

- FA接受的字符串：若M是一个FA, 对于Σ上的任意一个符号串α，若存在一条从某一初始状态节点到某一终止状态节点的通路，而且这条通路上所有有向边上标记的符号按顺序依次连接而成的符号串为α，则称α为M所接受或识别.
- FA接受的语言：FA M接受的所有符号串的集合称为M定义的语言，记为L(M).

2.3.1 非确定有限自动机

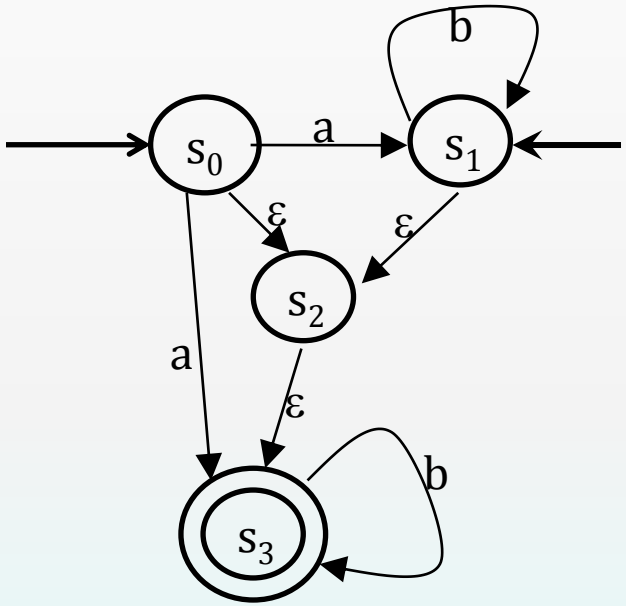
- 例1:



- 该自动机接受的语言是
- L = {aba, abaa, abab, abaab, abaaab, abaabb, ……}
- 等价于正则表达式aba(a|b)*定义的语言

2.3.1 非确定有限自动机

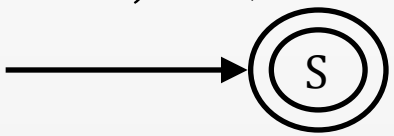
- 例2:



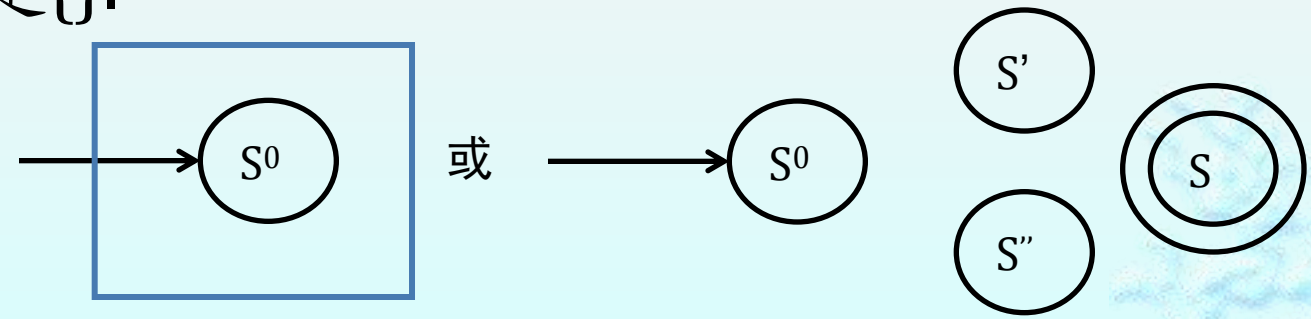
- 该自动机接受的语言是
- L = {ε,a,ab,abb,abbb,abbbb……}
- 等价于正则表达式ε |ab*定义的语言

2.3.1 非确定有限自动机

- 例3:若自动机M只有一个状态，且该状态即是初始状态又是终止状态，则M定义的语言是{ε}.

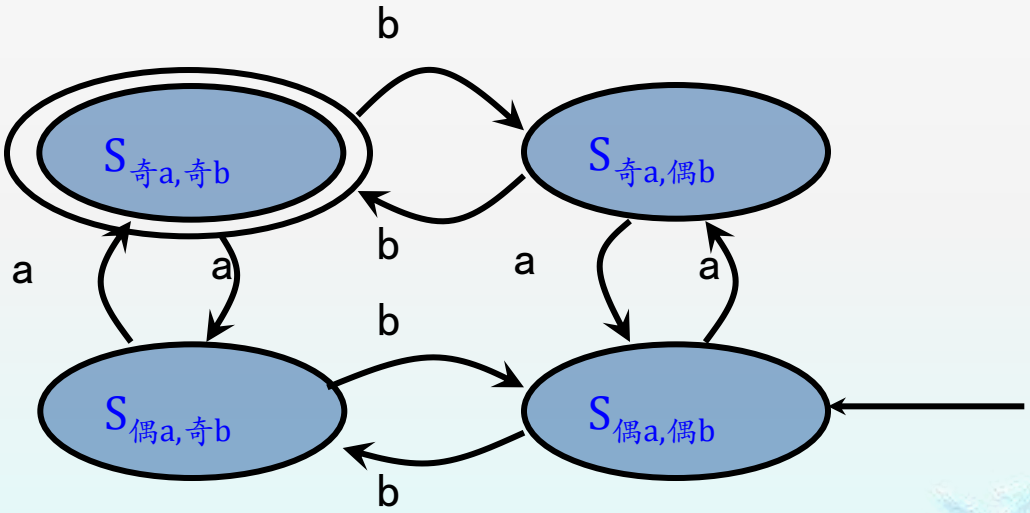


- 例4:若自动机M只有一个状态，且该状态是初始状态，或者M有若干个状态，但是任意初始状态到任意终止状态之间都不存在通路，则M定义的语言是{}.



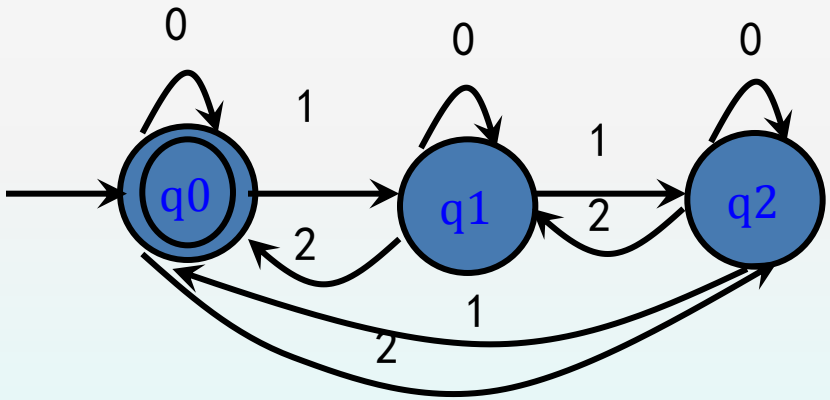
2.3.1 非确定有限自动机

- 例5：Σ = {a,b},构造自动机,识别由所有奇数个a和奇数个b组成的字符串.



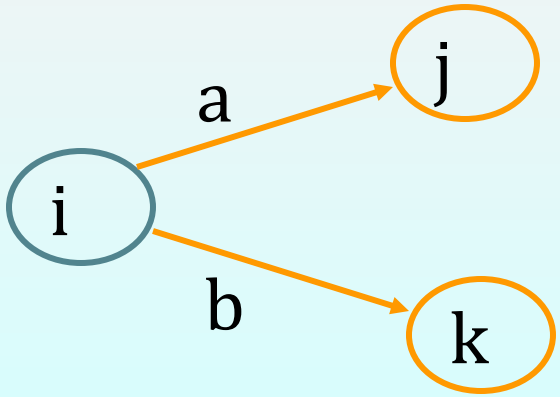
2.3.1 非确定有限自动机

- 例6：设计有限自动机M，识别{0,1,2}上的语言，每个字符串代表的数字能整除3.



2.3.1 非确定有限自动机

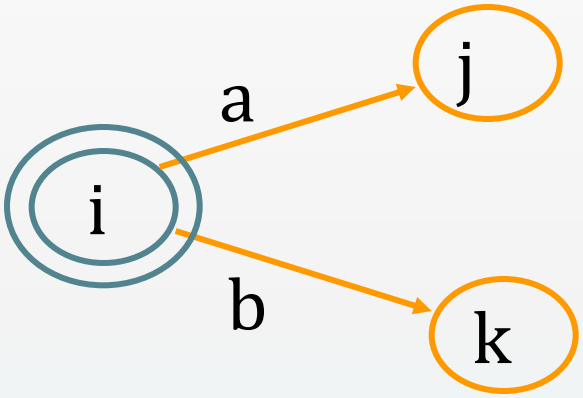
- 基于转换图的实现：思想是将状态转换函数固化到程序中
- 方法: (1)每个状态对应一个带标号的语句;
  - (2):状态遇到符号发生不同的转换用switch语句模拟;
  - (3):状态转换用goto语句模拟;




```
Li: switch CurrentChar of
      case a   : goto Lj;
      case b   : goto Lk;
      default  : return false;
```

2.3.1 非确定有限自动机

- 方法: (4)对于每个终止状态, 增加一个条件分支:若当前字符是输入串的结束符,则接受.




```
Li: switch CurrentChar of
      case  a   : goto Lj
      case  b   : goto Lk
      case  #   :  return true;
      default   : return false;
```

- 思考:初始状态需要什么特殊处理吗?
- 同一个状态遇到相同的符号转向不同的状态，程序如何写?如何执行?
- 状态遇到ε转换边，程序如何写?

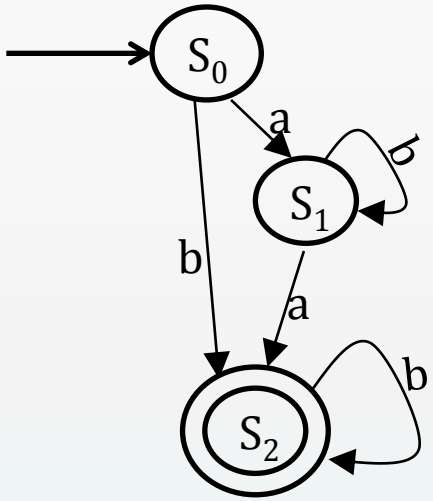
2.3.1 非确定有限自动机

- 基于转换表的实现：将状态转换函数表示成二维表格，作为识别程序的输入，该识别程序的计算逻辑是自动机工作过程的模拟
- 识别程序的算法描述：

1. *State* = InitState;
2. Read(CurrentChar);
3. while *T*(State, CurrentChar) <> error && *CurrentChar* <> '#'
   do
   begin State = *T*(State, CurrentChar);
   Read(CurrentChar); end;
4. if *CurrentChar* = '#' && *State* ∈ *FinalStates*, return *true*;
   otherwise, return *false*.

2.3.1 非确定有限自动机

- 自动机实现的例子



	a	b
s_0 +	s_1	s_2
s_1	s_2	s_1
s_2 *		s_2

```
LS0: Read(CurrentChar);
     switch (CurrentChar) {
      case  a: goto LS1;
      case  b: goto LS2;
      default: return false; }
LS1: Read(CurrentChar);
     switch (CurrentChar) {
     case  a: goto LS2;
     case  b: goto LS1;
     default: return false; }
LS2: Read(CurrentChar);
     switch (CurrentChar) {
     case  b: goto LS2;
     case #: return true;
     default: return false; }
```

符号串 abaa
能否被接受?
符号串abbab?
符号串bb?

2.3.1 非确定有限自动机

- 两种实现方式的比较:
- 基于转换表的实现：实现算法是通用的，对于不同的语言(语言不同，接受它的自动机是不同的)，只需改变输入的转换表，识别程序不需改变;

- 基于转换图的实现：不需要存储转换表(通常转换表是很大的)，但当语言改变即自动机的结构改变时，整个识别程序都需要改变。

2.3.2 确定有限自动机

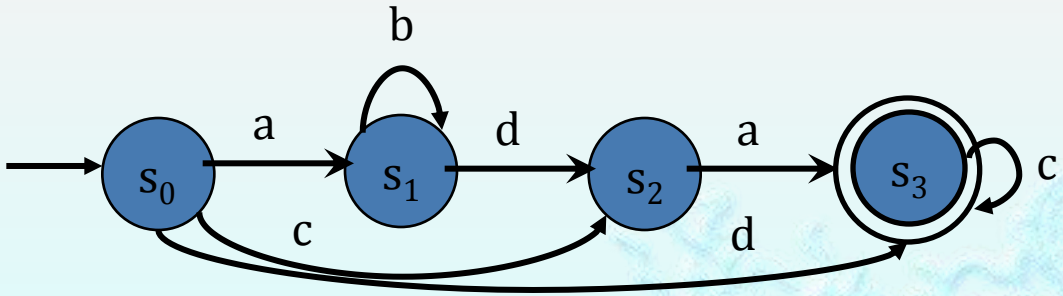
- 定义：确定有限自动机DFA是一个五元组(S,Σ,f,s_0,Z)，其中：
  - S是有限状态集，不能为空
  - Σ是有穷字母表，可以为空
  - f是状态转换函数，f：$S\times\Sigma\rightarrow S$
  - s_0是初始状态，$s_0\in S$
  - Z 是终止状态集，$Z\subseteq S$，可以为空

2.3.2 确定有限自动机

- 例子： 自动机 $M_2 = \{S=\{s_0,s_1,s_2,s_3\},\Sigma=\{a, b, c, d\},f, s_0, Z=\{s_3\}\}$,其中：
- $f(s_0,a) = s_1$, $f(s_0,c) = s_2$, $f(s_0,d) = s_3$,
- $f(s_1,b) = s_1$, $f(s_1,d) = s_2$,
- $f(s_2,a) = s_3$,
- $f(s_3,c) = s_3$

	a	b	c	d
s_0^+	s_1	⊥	s_2	s_3
s_1	⊥	s_1	⊥	s_2
s_2	s_3	⊥	⊥	⊥
s_3^*	⊥	⊥	s_3	⊥

M_2的转换表表示



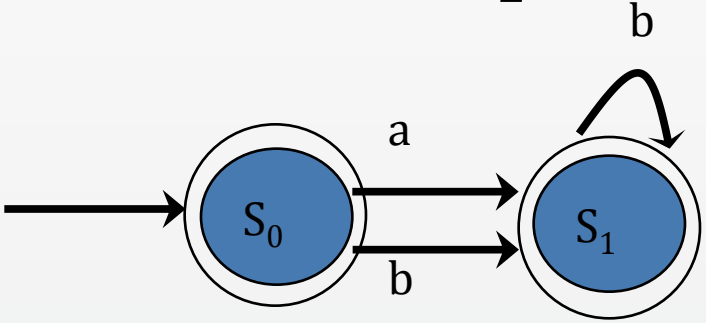
M_2的转换图表示

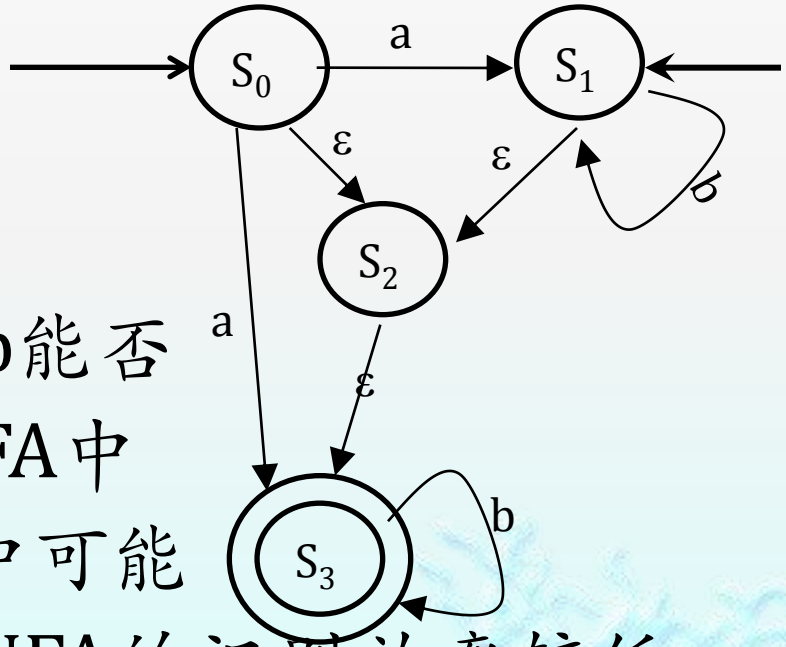
2.3.2 确定有限自动机

- 确定有限自动机接受的字符串、语言的定义与非确定有限自动机相同.
- 例：M_2接受的语言等价于正则表达式 (ab*da | ca | d)c* 定义的语言
- 确定有限自动机的实现方式也是基于转换图和基于转换表，方法相同.
- 对于同一个语言，既可以用确定有限自动机定义，又可以用非确定有限自动机定义.

2.3.2 确定有限自动机

- 例：识别串集: (a|ε)b*的DFA M_1如左图所示，等价的NFA M_2如右图所示：





- 当我们想判断字符串abb能否
- 被M_1或M_2所接受时，DFA中
- 只有一条路可走，NFA中可能
- 要尝试多条路径，因此NFA的识别效率较低

2.3.2 确定有限自动机

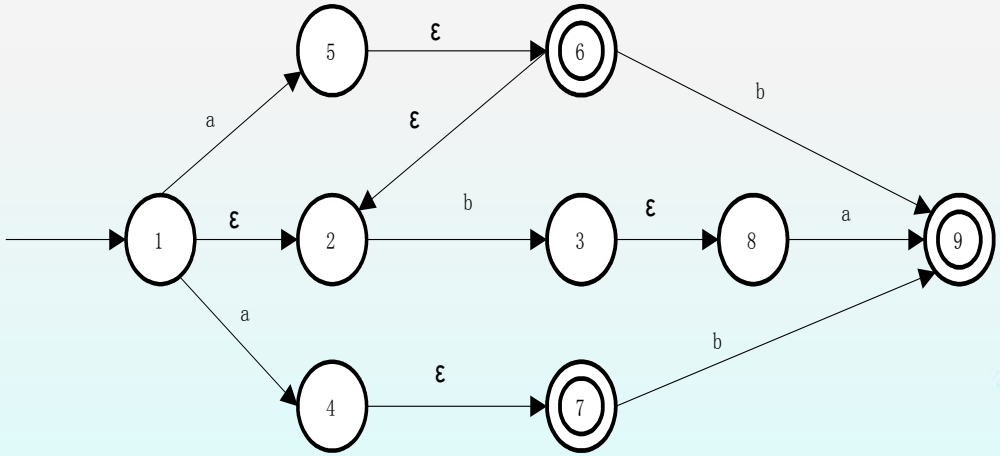
- NFA到DFA的转化：思想消除不确定性
- 方法：(1)合并初始状态
- (2)消除空边 ---计算空闭包
- (3)消除多重定义的边 ---合并多重转换
- NFA确定化的算法---子集法
  - I是NFA的状态子集, ε-Closure(I)是从I中的每个状态出发经过任意条空边所能到达的所有状态的集合
  - I是NFA的状态子集, $a \in \Sigma$, I_a = ε-Closure(J), J是从I中的每个状态出发经过一条a边所能到达的所有状态的集合.

2.3.2 确定有限自动机

- 对给定的NFA按照如下的步骤进行确定化：
- (1) 构造一张表，共有$|\sum|+1$列，第一列表示状态子集I，然后对每个$a\in\sum$分别用一列表示I_a；
- (2) 第二行第一列的状态子集为$\varepsilon_CLOSURE(S_0)$，其中S_0为NFA M的初始状态集；
- (3) 若第一列中的每个状态子集都经本步骤处理过，则转步骤5；否则任取一个未经本步骤处理过的状态子集I，对每个$a\in\sum$求I_a，并记入相应的I_a列，如果它和表格第一列中的所有状态子集均不相同，则此表生成一新行，将它填入新行的第一列中；
- (4) 转步骤3；
- (5) 将第一列中的每个状态子集重命名为新的状态，第二行的状态子集重命名后成为新的DFA的初始状态，含有NFA的终止状态的状态子集重命名后成为新的DFA的终止状态。上述表格便成为新DFA的状态转换矩阵.
- 思考：上述算法是否一定终止?

2.3.2 确定有限自动机

- 例子：设有NFA M = ({1，2，…，9}，{a，b}，f，{1}，{6，7，9})，其中f如下图所示，将该非确定有限自动机确定化。



2.3.2 确定有限自动机

- DFA M的化简是指寻找一个状态数最少的DFA M’，使得L(M’) = L(M).
- 定义：设DFA M 的两个不同状态q_1和q_2，如果对任意输入的符号串α，从q_1和q_2出发，总是同时到达接受状态或拒绝状态中，则称q_1和q_2是等价的. 如果q_1和q_2不等价，则称q_1和q_2是可区分的.
- 定义：从有限自动机的初始状态开始，任何输入序列都不能到达的那些状态称为无关状态。
- 定义：如果DFA M 没有无关状态，也没有彼此等价的状态，则称DFA M 是最小的(或归约的)。

2.3.2 确定有限自动机

- DFA M的化简：状态分离法
- 思想：是将一个DFA(不含无关状态)的状态集划分为一些不相交的子集，使得任何两个不同子集中的状态都是可区分的，而同一子集中的任何两个状态都是等价的；然后，在每一个子集中选一个代表，合并掉其它的等价状态，这样就得到了一个最小的DFA。
- 步骤: (1)去掉 DFA中的无关状态，把状态集中的终止状态和非终止状态分开，生成两个状态子集，形成初始划分I ;
- (2) 任取一个状态集，考察集合内状态是否等价，不等价则进行拆分;
- (3)重复步骤2，直至再没有状态集可拆分;
- (4)对最终得到的分划中的状态集进行命名，并确认初始状态、终止状态和转换函数

2.3.2 确定有限自动机

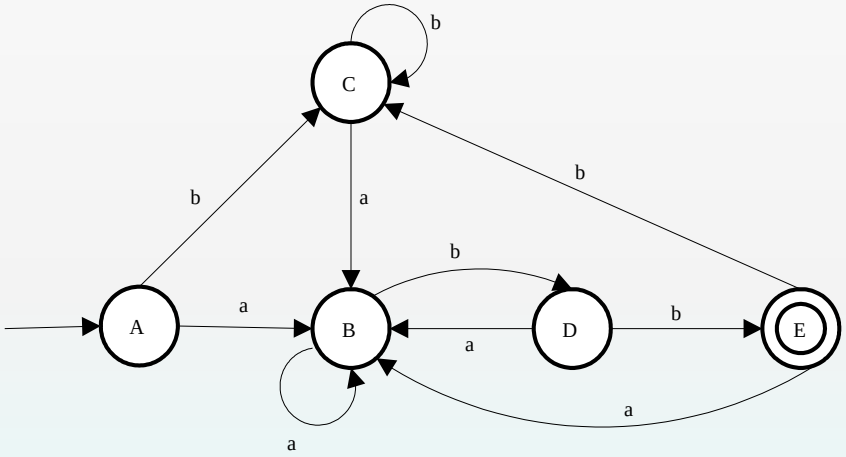
- 步骤2：某一时刻，$I=\{I_1,I_2,...,I_m\}$,任意$I_i=\{q_1,q_2,...q_k\}$是否可拆分呢?
- 方法：对所有$a\in\Sigma$, 若a使得I_i中的各状态遇到a后转移的后继状态不全在某一现存的子集中，则对I_i进行进一步的划分.一般地，若I_i经过a弧落入现行划分中的N个不同子集，则应将I_i划分为N个不相交的子集，使得同一个子集中的所有状态经a弧都落入当前划分的同一个子集中，这样形成新的划分. 重复上述过程，直至划分中所含的子集数不再增长为止。

2.3.2 确定有限自动机

- 步骤4：若最后分划$I = \{I_1, I_2, \ldots, I_m\}$，
- 如果I_i中包含原DFA的初始状态，则I_i是新DFA的初始状态；
- 如果I_i包含原DFA的终止状态，则I_i是新DFA的终止状态；
- 对于子集$I_i = \{q_1, q_2, .., q_n\}$，可选择其中的任意一个，比如$q_1$来代表这个子集，在原来的自动机中，凡是转到$q_2$、$q_3$、…、$q_n$的有向边都改成转到$q_1$.

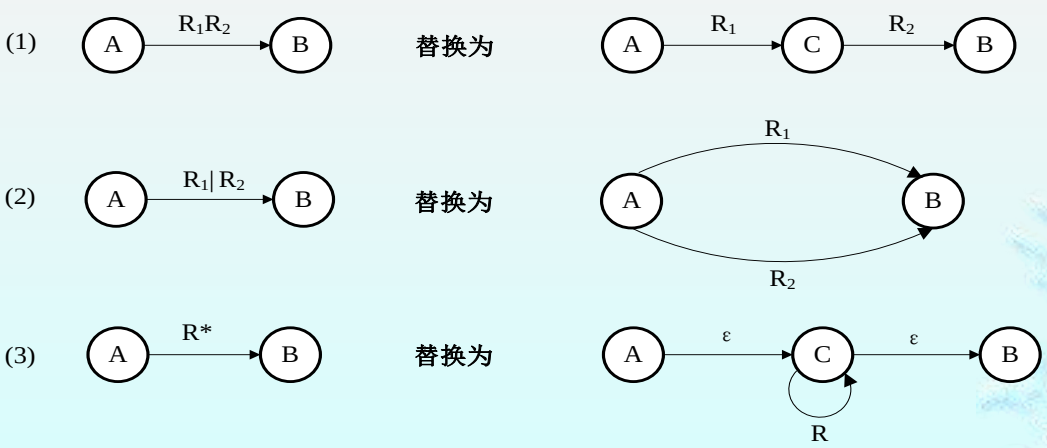
2.3.2 确定有限自动机

- 例子：对下图所示自动机最小化



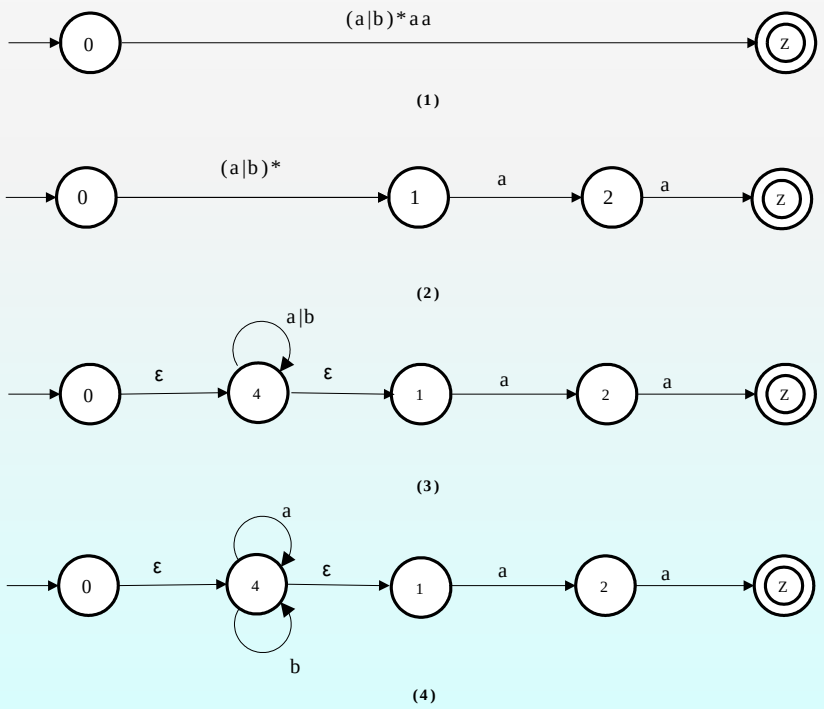
2.3.3 有限自动机与正则表达式

- 有限自动机的表达能力与正则表达式等价，因此相互之间是可以进行转换的.
- 一般地，正则表达式常用来描述字符串，有限自动机常用来识别字符串
- 正则表达式到有限自动机的转换



2.3.3 有限自动机与正则表达式

- 例子：有正则表达式(a|b)*aa，为之构造等价的NFA.



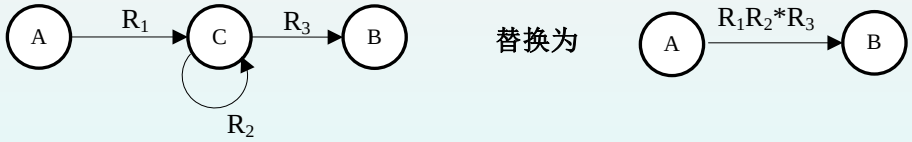
2.3.3 有限自动机与正则表达式

- 有限自动机到正则表达式的转换

(1)

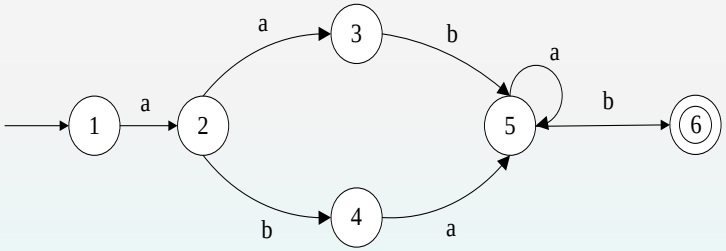
(2)

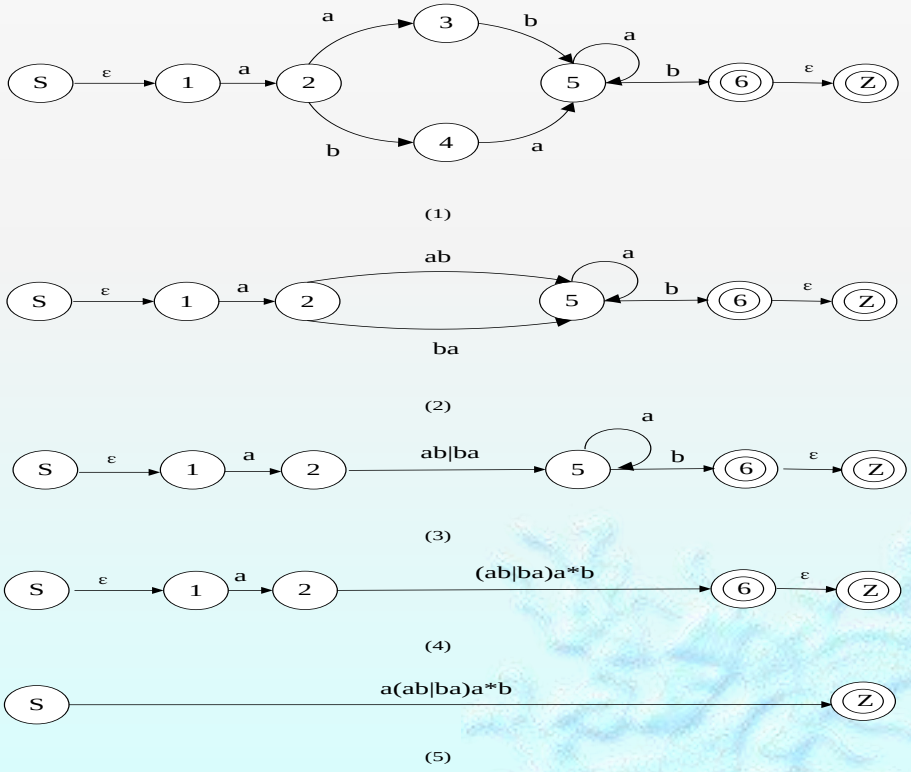
(3)



2.3.3 有限自动机与正则表达式

- 例子：有如下左图所示的NFA, 给出与之等价的正则表达式.





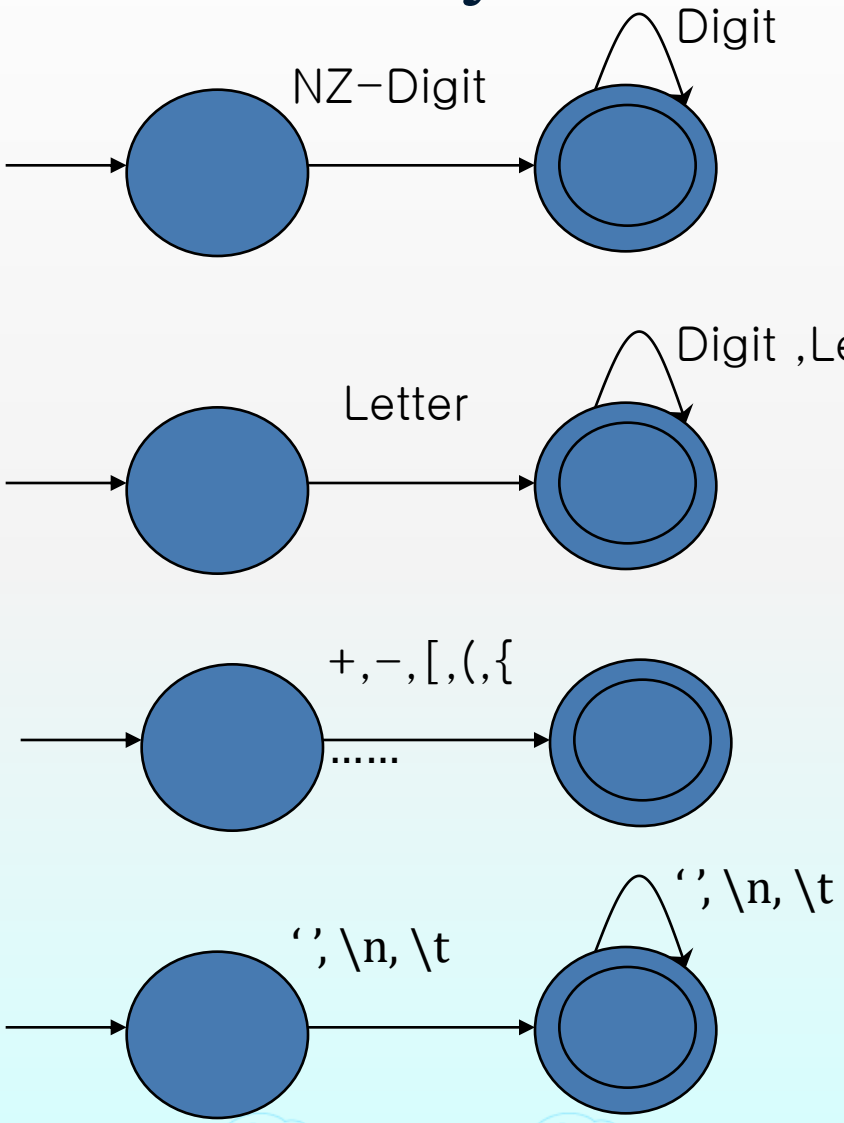
2.4 词法分析程序的设计与实现

- 词法分析程序可以用手工编写，也可用工具自动生成
- 一般地，手工编写的步骤是：先用RE定义单词结构，然后将RE转化成NFA，NFA再转化成DFA，DFA化简并实现得到词法分析程序
- 工具自动生成的步骤是：将语言单词的构成规则用RE定义出来，输入给该工具，该工具运行后就会输出一个词法分析程序

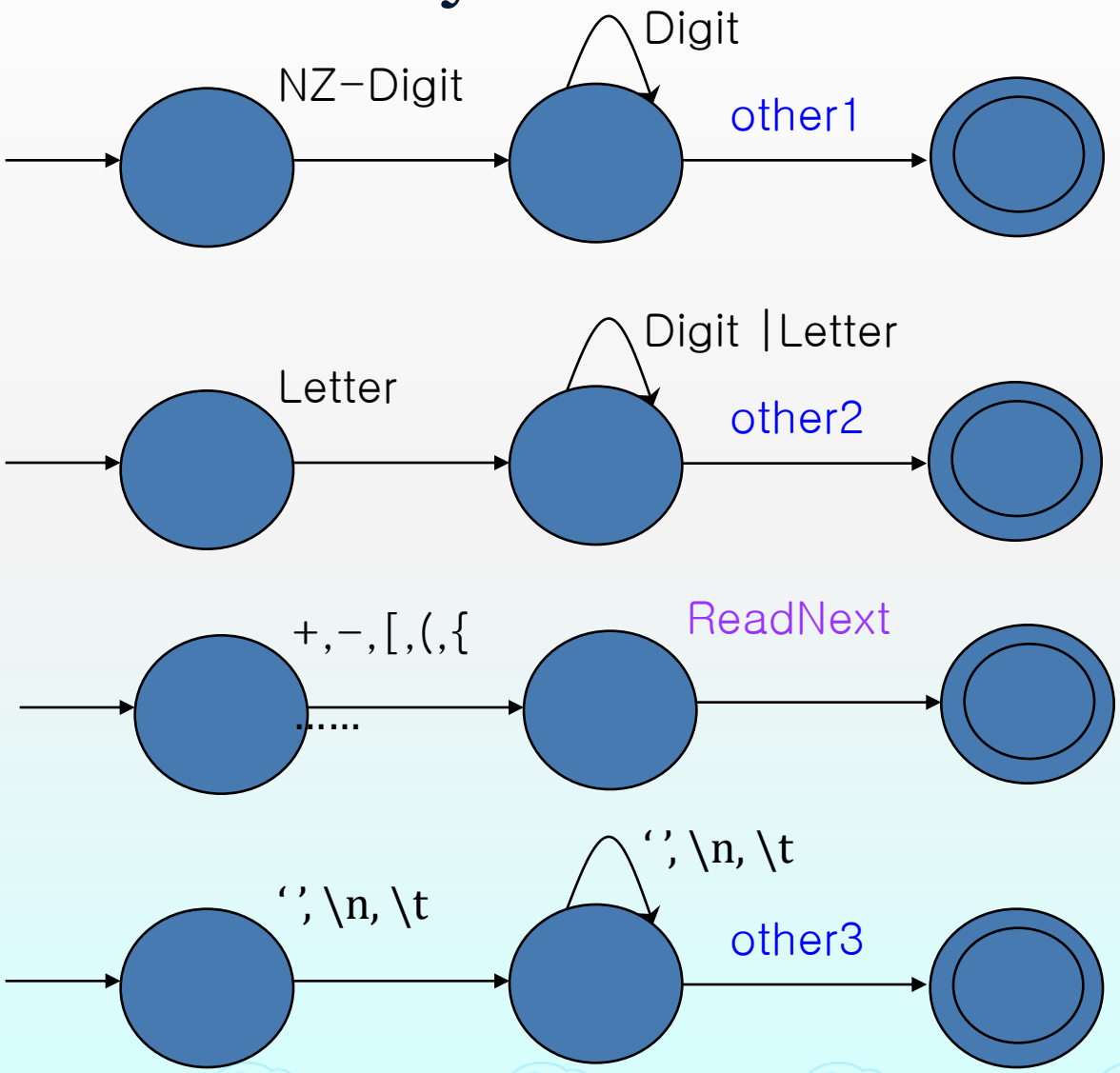
ToyL词法规则的正则定义

- **letter = a|...|z|A|...|Z**
- **digit = 0|...|9**
- **NZ-digit = 1|...|9**
- **Keywords:**
  Keyword = if | else| while| read| write| int
- **Identifiers:**
- **identifier =** letter (letter|digit)*
- **Constant:**
  intC = NZ-digit digit* | 0
- **Other symbols:**
- **syms = +|-| *|/ | > | < | = | : | ; | (|) | { | }**
- **Whitespace:**
- **ws = (‘ ’ | \n | \t)+**
- **Lexical structure:**
  lex = Keyword | identifier | intC | syms | ws

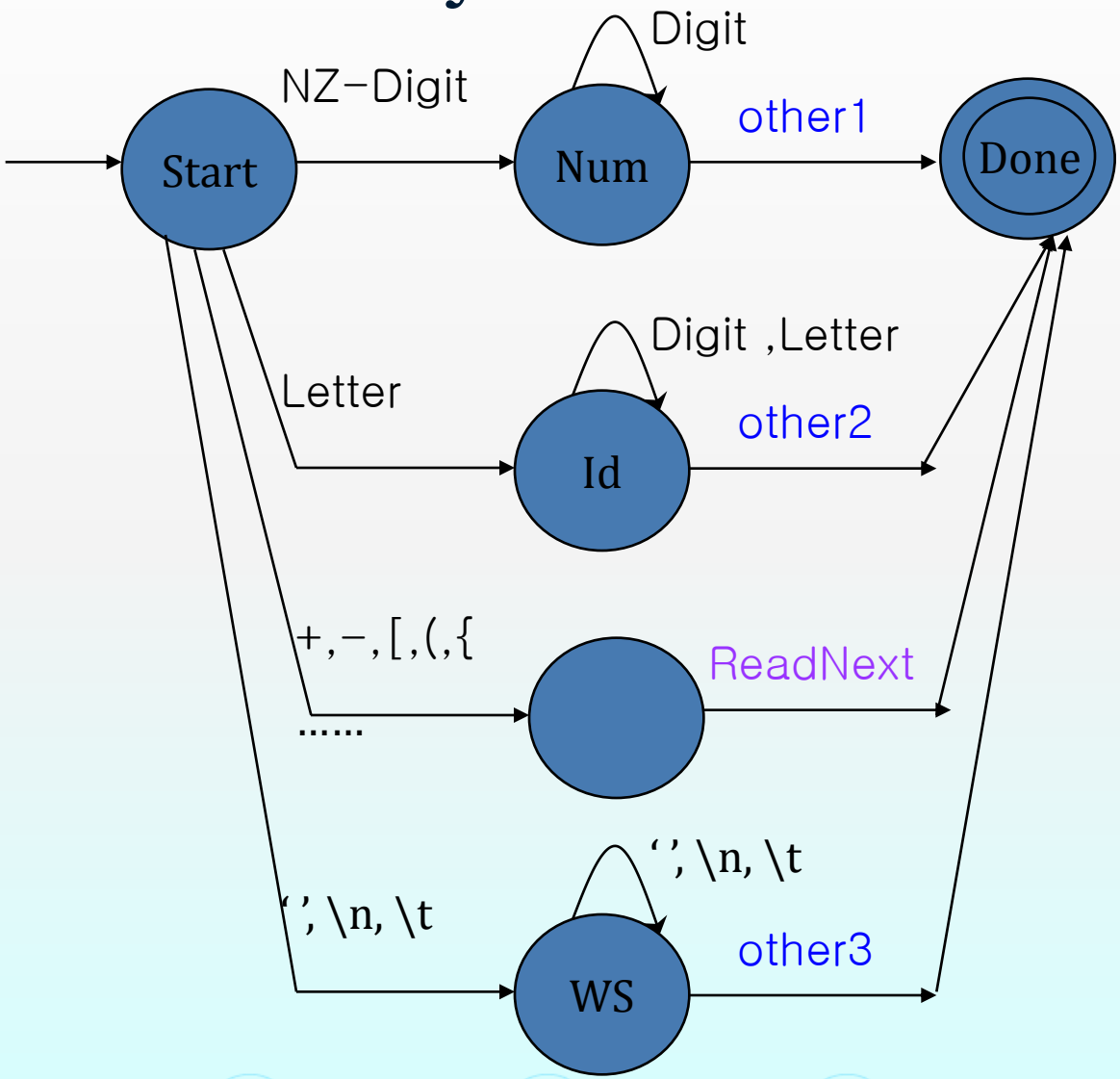
ToyL的有限自动机



ToyL的有限自动机


NZ-Digit
Digit
other1
Letter
Digit |Letter
other2
+,-,[,(,{
.....
ReadNext
‘ ’, \n, \t
‘ ’, \n, \t
other3

ToyL的有限自动机


Digit
NZ-Digit
Start
Num
other1
Done
Digit ,Letter
Letter
Id
other2
+,−,[,(,{
.....
ReadNext
‘ ’, \n, \t
‘ ’, \n, \t
WS
other3

根据 DFA构造词法分析程序

- 输入:
  - 以EOF 作为结束符的符号序列;
- 输出:
  - token序列;
- 数据结构:
- struct Token {TkType type;　　//单词类别
- 　　char　val[50];　//语义信息
- 　　}

根据 DFA构造词法分析程序

单词类别:

```
typedef enum { IDE, NUM,                    //标识符，整数
               PLUS, MINUS, MUL,            // +, -, *,
               DIV, GT, LT, EQ,             // /, >, <, =
               SEMI, LPAREN, RPAREN         // ;, (. )
               LG, RG,                      // {, }
               ELSE, INT ,WHILE, READ, WRITE , IF    //关键字
              } TkType
```

根据 DFA构造词法分析程序

全局变量:

- char str[50]; ----- 存储已经读入的串;
- int len = 0; ----- str的长度, str数组下标
- Token tk; ----- 当前token
- Token TokenList[100]; ---- token序列
- int total = 0; ----- 识别出的token的个数

根据 DFA构造词法分析程序

预定义函数:

- ReadNext() --- 将当前符号读入CurrentChar,
  如果当前符号是 EOF 返回 false;
  否则返回true;
- IsKeyword(str) --- 检查str是否是关键字,如果是关键字,返回关键字号;否则返回-1;

根据 DFA构造词法分析程序

```
if  (!ReadNext()) exit(1);
/*读入一个字符到CurrentChar中，若到输入串结束(CurrentChar = EOF)，则结束分析;否则继续分析。*/
start:   //开始状态的标号
   case CurrentChar of
      “1..9”: str[len++] = CurrentChar; goto Num ;
      “a..z”, “A..Z”: str[len++] = CurrentChar; goto ID;
      “+”, “-”, “>”,”(”,…: tk.type =PLUS; ReadNext(); goto Done;
      “\t”,”\n”,” ”: goto WS;
      other: error();  //出现字母表以外的字符
```

根据 DFA构造词法分析程序

```
Num:                          //数字状态的标号
   ReadNext();
   case CurrentChar of
      "0..9": str[len++] = CurrentChar;  //数字拼接
              goto Num ;
      other:  tk.type = NUM ;     //other1:数字以外的字符
              strcpy(tk.val, str);
              goto Done;          //已经读入下一字符
```

根据 DFA构造词法分析程序

```
ID:                                    //标识符的标号
   ReadNext();
   case CurrentChar of
      “0..9”, “a..z”,“A..Z”：  str[len++] = CurrentChar;
                                       goto ID;
      other: if  IsKeyword(str) //other2:字母,数字以外的字符
                    {tk.type = IsKeyword(str) }
               else  {tk.type = IDE, strcpy(tk.val, str) };
               goto Done;        //已经读入下一字符
```

根据 DFA构造词法分析程序

```
WS:                                    //回车，换行，空格符
      ReadNext();
  case CurrentChar of
       “\t”, “\n”, “ ”: goto WS;
       other: goto Done; //other3:Tab, 换行, 空格以外的字符
```

根据 DFA构造词法分析程序

```
Done:                                  //完成状态的标号
  TokenList[total] = tk;      // 向token表中添加新的token
  total ++;                            //token总数增加一个
  len = 0;                             //准备存储新的token串
  strcpy(str, “ ”);                    // token属性值重置
  if (CurrentChar == EOF) exit(1);   //到达程序结尾退出
  goto start;                          //开始扫描新的token
```

开发词法分析程序应注意的一些问题

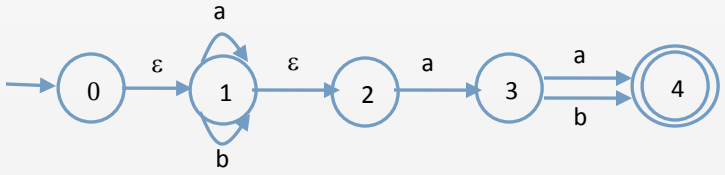
- 去掉注释、空格、制表符等格式控制符号
  - *{…*
  - *x = 0;　　/*变量初始化*/*
  - *x = y * z;　//计算x*
  - *…}*
- 进行宏替换
  - *#define LEN 100*
- 包含文件的拷贝
  - *#include “stdio.h”*

开发词法分析程序应注意的一些问题

- 保留字的处理
  - 单独识别还是与标识符一起识别?
- 复合单词的识别
- 数的转换
- 括号匹配
  - 词法分析时处理还是语法分析时处理?
- 词法分析的结束
  - 特殊符号还是EOF?

作 业

- 1.有限自动机识别字符串：构造一个DFA：使它能够识别出所有能被3整除的二进制数(读数顺序是从高位到低位)
- 2.NFA到DFA转换：



- 3.DFA化简（先转化成DFA再化简）