第3章　词法分析

1　词法分析程序的输出形式。

2　单词的描述工具

3　有穷自动机

4　正规式和有穷自动机的等价性

5　词法分析程序的编写

本章要点

进入

本章概述及主要概念

词法分析是编译的第一个阶段，它的主要任务是从左至右逐个字符地对源程序进行扫描，产生一个个单词序列，用以语法分析。本章主要介绍词法分析程序的设计原则，与词法分析相关的形式化描述法和分析方法。

符号串

4类文法

规范推导

二义文法

自下而上分析法

句柄

3.1 词法分析程序的输出形式

一般情况，常将词法分析程序设计成一个子程序。

单词符号（TOKEN）是一个程序设计语言的基本语法符号。
程序设计语言的单词符号一般可分成下列5种：

1．基本字，也称关键字，如PASCAL语言中的begin，end，if，while和var等。

2．标识符，用来表示各种名字，如常量名、变量名和过程名等。

3．常数，各种类型的常数，如25，3.1415，TRUE和"ABC"等。

4．运算符，如+，*，<= 等。

5．界符，如逗点，分号，括号等。

上一页 下一页

词法分析程序所输出的单词符号常常采用下二元式表示：

（单词种

（1）一类单词统一用一个种别表示，如1代表基本字，2代表标识符等。

（2）每一个单词用一个种别表示，如：1代表BEGIN，2代表END等。

1．单词

单词的种别是词 析需要的信息。可用整数码或助记符等表示。

如何划分种别？

2. 单词自身的值

是编译其它阶段需要的信息。一般用单词自身表示，若单词类别仅表示一个单词，可不用单词自身的值。

上一页 下一页

例： if i=5 then x:=y 经词法分析可以表示如下二元式形式：

(ifsym,)
(ident, 'i')
(eql,)
(number, 数值)
(thensym,)
(ident, 'x')
(becomes,)
(ident, 'y')

上一页 下一页

3.2 单词的描述工具

- 程序设计语言中的单词(TOKEN)是基本语法符号。
- 单词符号的语法可以用有效的工具加以描述。

上一页 下一页

3.2.1 正规文法

正规文法也是前面2.4节介绍过的3型文法，它所描述的是V_T*上的正规集。**例:**

程序设计语言中的几类单词可用下述规则描述：

<标识符>→L | L <字母数学>

<字母数字>→ L | D | L <字母数字>| d<字母数字>

<无符号整数>→ D | D <无符号整数>

<运算符>→+| - |*|/|=|<<等号>|><等号>……

<等号>→=

<界符>→，|；|（|）|……

其中L表示a~z中的任何一英文字母，D表示0~9中的任一数字。

3.2.2 正规式

正规式和它所表示的正规集的递归定义如下。
设字母表为Σ，辅助字母表Σ ={ |, ·, *, (,) }

1. 和Φ都是Σ上的正规式，它们所表示的正规集分别为{ }和Φ；
2. 任何a ,a是Σ上的一个正规式，它所表示的正规集为{a}；
3. 假定e1和e2都是Σ上的正规式，它们所表示的正规集分别为L(e1)和 L(e2) 那么，（e1），e1|e2，e1·e2和e2*也都是正规式，它们所表示的正规集分别为

   $L(e_1)$，$L(e_1) \cup L(e_2)$，$L(e_1) \cdot L(e_2)$和

   $(L(e_1))^*$。

3.2.2 正规式

4．仅由有限次使用上述三步骤而定义的表达式才是∑上的正规式，仅由这些正规式所表示的字集才是∑上的正规集。

其中的"|"读为"或"（也有使用'+'代替'|'的）；"·"读为"连接"；"*"读为"闭包"（即，任意有限次的自重复连接）。在不致混淆时，括号可省去，但规定算符的优先顺序为先'*'，再"·"最后"|"。连接符"·"一般可省略不写。"*"、和"|"都是左结合的。

例: 令∑={a, b}，列出一些∑上正规式和相应的正规集。

规正式	正规集
a	{a}
b	{b}
a|b	{a,b}
ab	{ab}
(a|b)*	**{ε, a, b, aa, ab, ...所有a, b组成的串}**
(a|b)* aa|bb{a|b}*	**∑*上所有所有含有两个相继a或两个相继b的串**

上一页 下一页

3.3 有穷自动机

有穷自动机（也称有限自动机）作为一种识别装置，它能准确地识别正规集，即识别正规文法所定义的语言和正规式所表示的集合，引入有穷自动机这个理论，正是为词法分析程序的自动构造寻找特殊的方法和工具。

有穷自动机分为两类

- 确定的有穷自动机DFA（Deterministic Fintie Automata）
- 不确定的有穷自动机NFA（Nondeterministic Fintie Automata）

下面我们逐一介绍这两类有穷自动机

上一页 下一页

3.3.1 确定的有穷自动机（DFA）

1. DFA的定义：一确定的有穷自动机（DFA）M是一个五元组：　M=（K，∑，f, S, Z）　其中:

☆ 1．K是一个有穷集，它的每个元素称为一个状态；

☆ 2．∑是一个有穷字母表，它的每个元素称为一个输入字符，所以也称为输入符号字母表；

☆ 3．f是转换函数，是在K×∑→K上的映像，即，如f(ki,a)=kj, 就意味着，当前状态为ki，输入字符为a时，将转换到下一状态kj，我们把kj称为ki的一个后继状态；

☆ 4. S 是唯一的一个初态；

☆ 5．Z 是一个终态集，终态也称可接受状态或结束状态。

例: DFA M=（{S，U，V，Q}，{a, b}, f, S, {Q}）

其中 f 定义为：

f (S, a)=U　f (V, a)=U

f (S, b)=V　f (V, b)=Q

f (U, a)=Q　f (Q, a)=Q

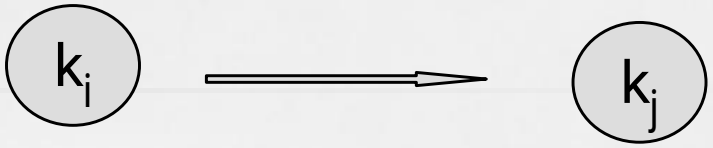
f (U, b)=V　f (Q, b)=Q

上一页 下一页

2、DFA的其它表示形式

(1)状态转换图表示

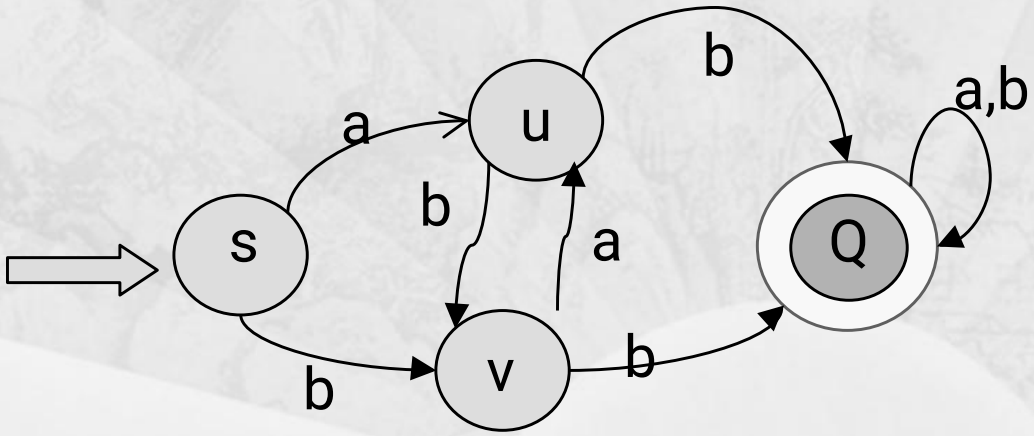
若f (Ki, a)=Kj 则画



初态结冠以==>，如：==>

终态结点用双圈表示, 如：

上例的状态图表示如下：



（2）矩阵表示

行表示状态，列表示输入字符，矩阵元素表示相应的f值。如前例，DFA的矩阵表示如下：

状态＼字符	a	b	
S	U	V	0
U	Q	V	0
V	U	Q	0
(Q)	Q	Q	1

其中圆圈中的状态为终态。

3、DFA的作用

识别字符串

对于∑*中的任何字符串t，若存在一条从初态到某一终态结的道路，且这条路上所有弧的标记符连接成的字符串等于t，则称t可为DFA M所接受，若M的初态结同时又是终态结，则空字可为M所识别。

DFA M所能接受的字符串的全体记为L（M）。

3.3.2 不确定的有穷自动机（NFA）

1.定义

一个不确定的有穷自动机（NFA）M是一个五元组，

M=（K，∑，f，S，Z）。

其中:

☆ 1.K是一个有穷集，它的每个元素称为一个状态

☆ 2.∑是一个有穷字母表，它的每个元素称为一个输入字符

☆ 3.f是一个从K×∑*到K的幂集的映象

☆ 4.S⊂K，是一个非空初态集

☆ 5.Z ⊂K，是一个终态集

NFA也可以用状态转换图，状态转换矩阵表示。

2、DFA与NFA的区别

（1）反映在转换函数上。

DFA：f（Ki, a）= Kj

↑ ↑

a是一个字符 Kj是一个状态

NFA：f (Ki, α) = Kj

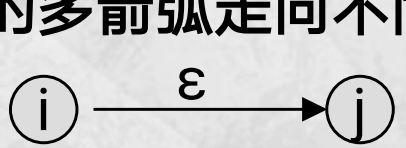
↑ ↑

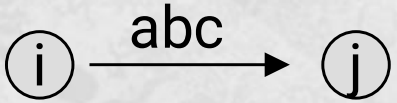
α∈ ∑* Kj是状态子集，可含多个状态

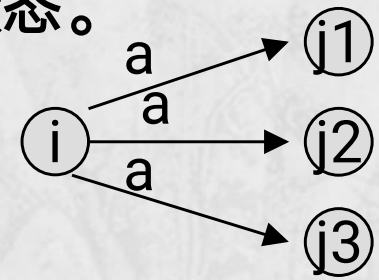
(2) 反映在转换图上。

NFA：箭弧上可标字符或字符串；从同一状态出发，可由标记相同的多箭弧走向不同状态。

如：







DFA：不允许上面情况。

DFA：只有唯一初态。NFA：有初态集。

3、DFA与NFA的联系。

（1）DFA是NFA的特例。

（2）对于每个NFA M，存在一个DFA M’，
使得L（M）=L（M’）

4、自动机的等价

对于任何两个有穷自动机M和M’，

如果L（M）=L（M’），则称M与M’是等价的。

3.3.3 NFA→DFA的转换(确定化)

1、有关运算

（1）状态集合I的 ε- 闭包，表示为 ε--CLOSURE(I)。

定义:

①若S∈I，则S∈ ε--CLOSURE (I)
②若S∈I，则从S出发，经过任意条ε 弧而能到达的任何状态都属于ε--CLOSURE(I)

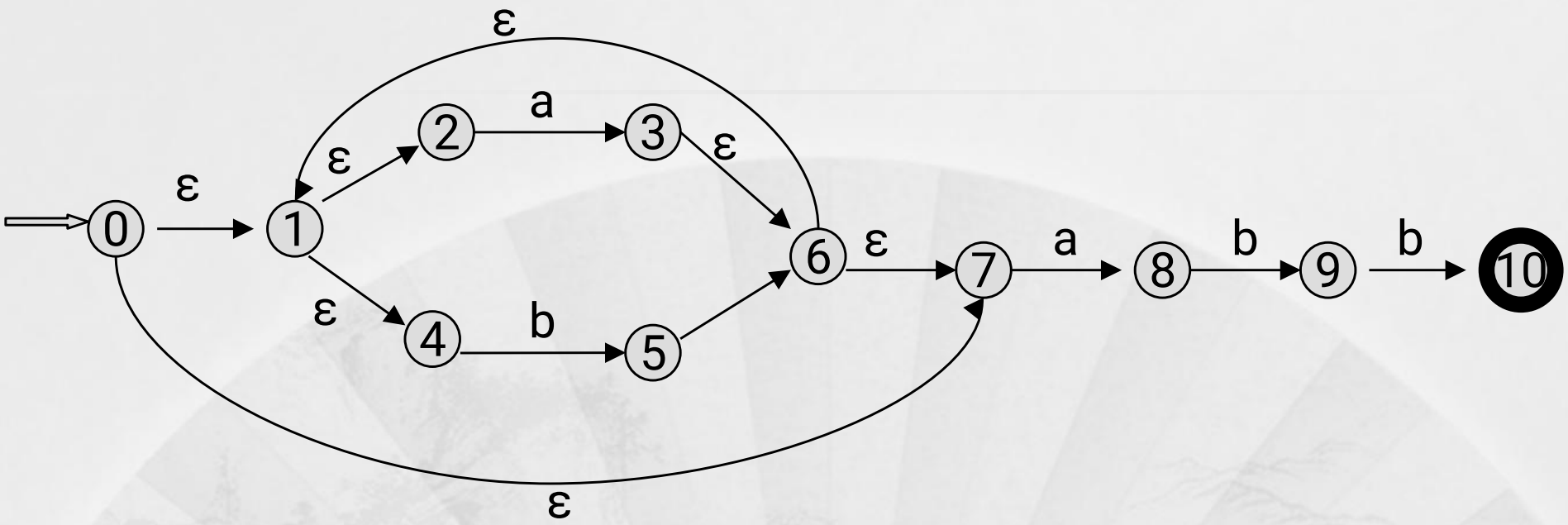
（2）求Ia子集，其中I是状态子集，a是一个输入字符。

定义:

Ia= ε --CLOSURE(J)
其中J是那些可以从I中状态出发经过一条a弧而到达的状态的全体。

通过求Ia，可合并从I出发，经过a弧走向的不同状态，并消去ε 弧。

例，已知一个NFA如下：



求 ε --CLOSURE(0)

ε --CLOSURE(0) = {0, 1, 2, 4, 7}

设 I= { 0, 1, 2, 4, 7 }

求 Ia = ε -CLOSURE({3, 8}) ={1, 2, 3, 4, 6, 7, 8}

上一页 下一页

设I={S_1, S_2, …, S_j}

则 Ia= ε --CLOSURE (f (S_1, a)U f (S_2, a)U …U f (S_j, a)

= ε --CLOSURE (f (S_1, a)) U ε --CLOSURE (f (S_2, a))

U ……U ε --CLOSURE (f (S_j, a))

若先求出NFA N中任一状态S**与任一输入字符a的**ε-LOSURE (f (s,a))，则对任何子集I，求Ia只需求并集**即可，由此得到求Ia**的简捷方法。

（3）使用子集法将NFA N**确定化为**DFA M的过程：设NFA N含有一个初态、终态，每条箭弧上标记为ε或一个字符。

为方便起见，令∑={a, b}，**构造一张状态转换**矩阵表，表有三列，依次化为I, Ia, Ib。

确定化过程：

<1> 将ε-CLOSURE ({x}) 作为新初态置表的第一行第一列(设x是NFA的初态)

即：

I	Ia	Ib
ε−closure({x})		

<2> 若某一行的第一列的状态子集已确定下来,则求出这一行的Ia, Ib, 并填入表。

<3> 若Ia或Ib未在第一列中出现，则填入后面空行的第一列位置上。

<4> 重复（2），（3），直至第一列的所有行都求出了该行的Ia, Ib, 且Ia, Ib全都在第一列中出现了为止。

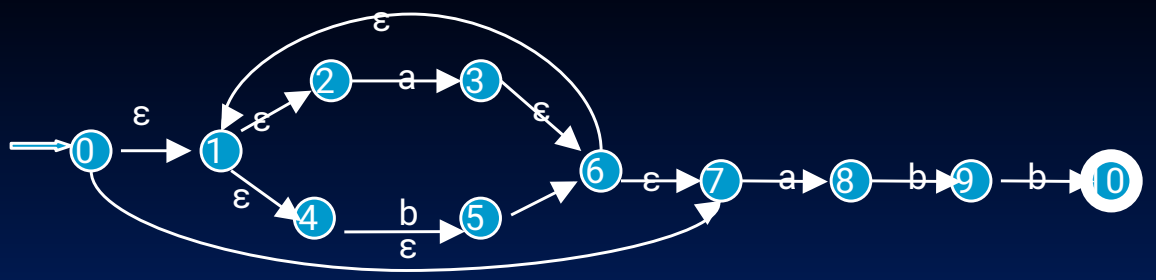
<5> 将表中的每个子集看成一个状态。

新初态是ε -CLOUSRE ({x})

新终态是那些含有原终态的子集。

则得到一个DFA M且L（N）=L（M）

例：将下面NFA N确定化。



求解过程

状态s	ε --CLOSURE (f (S, a))	ε --CLOSURE (f (S, b))
0	Φ	Φ
1	Φ	Φ
2	{1,2,3,4,6,7}	Φ
3	Φ	Φ
4	Φ	{1,2,4,5,6,7}
5	Φ	Φ
6	Φ	Φ
7	{8}	Φ
8	Φ	{9}
9	Φ	{10}
10	Φ	Φ

转下页继续

回节
上一页
下一页

得

过程

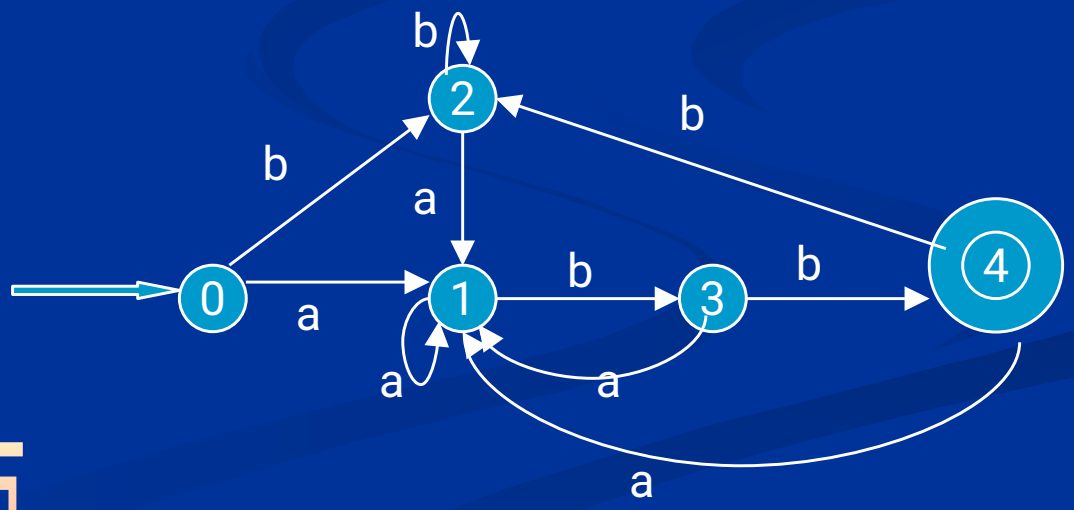
	状态子集 I	Ia	Ib
0	{0,1,2,4,7}	{1,2,3,4,6,7,8,}	{1,2,4,5,6,7}
1	{1,2,3,4,6,7,8}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7,9}
2	{1,2,4,5,6,7}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}
3	{1,2,4,5,6,7,9}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7,10}
4	{1,2,4,5,6,7,10}	{1,2,3,4,6,7,8}	{1,2,4,5,6,7}

得 DFA M

过程

状态I	f(I,a)	f(I,b)
0	1	2
1	1	3
2	1	2
3	1	4
4	1	2

相应的状态转换图



结束

回节 上一页 下一页

3.3.4 确定有穷自动机的化简（最小化）

化简：即消除多余状态，合并等价状态

等价状态定义:

假设DFA M中有状态s和t，如果从s出发能识别某一个字符串而停于终态，则从t出发也能识别此字符串而停于终态，反之亦然，则称s和t是等价的。

如果s和 t 不等价，则称s和 t 是可区别的。

显然，终态和非终态是可区别的。

使用分割法进行化简。分割法的基本思想：

把DFA M的状态分成一些互不相交的子集，使得任何不同的两子集的状态都是可区别的，而在同一子集中的任何两个状态都是等价的。每个子集只取一个状态作代表而删去其它等价状态，则得状态个数最少的DFA。

化简步骤：

（1）初始分划π =（终态集，非终态集）

（2）对π构造新的分划π new

for π中每个子集G DO

begin

把G分成若干子集使得G中两个状态S和 t 属于同一新子集，当且仅当对任何输入符号a，f (s, a)与f (t, a) 都属于同一个子集。

end

（3）如果π new = π， 让π find := π ,执行步骤（4），否则，令π := π new，转（2）

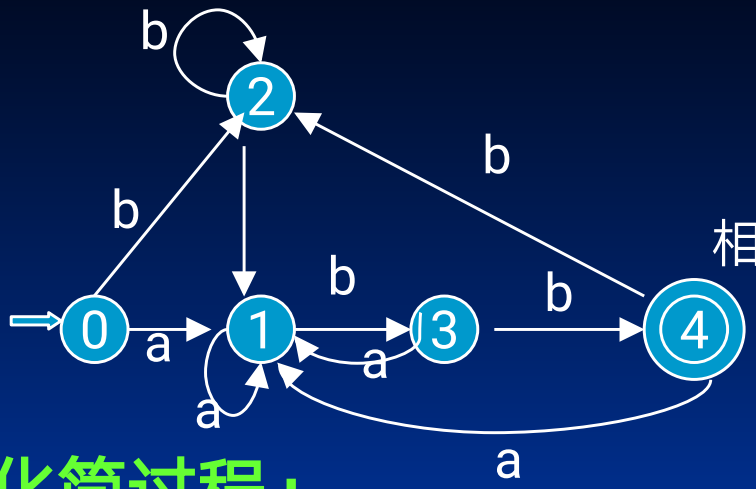
（4）对π find中的每个子集，取其中一个状态作代表，凡是转向该子集的状态，一律改成转向该子集的代表。

含初态的状态子集为新初态。

含终态的状态子集为新终态。

（5）删除无用状态。

例：对下面DFA M化简



状态 \ 字符	a	b
→ 0	1	2
1	1	3
2	1	2
3	1	4
(4)	1	0

化简过程：

（1）初始分划 =（{0，1，2，3}，{4}） 考察{0，1，2，3}$_a$ = {1, 1, 1, 1} <{0, 1, 2, 3}

{0, 1, 2, 3}$_b$ = {2, 3, 2, 4}不在 同一子集中。分成两个子集：{0，1，2}，{3}

得新的分划 πnew=({0，1，2}，{3}，{4}),因 πnew≠π,以π new作为π,转(2) 。

（2）考察{0，1，2}$_a$={1, 1, 1}< {0, 1, 2} {0, 1, 2}$_b$={2, 3, 2}不在同一子集中。

分成两个子集: {0，2}，{1}。得π new=({0, 2}, {1}, {3}, {4}) 因π new ≠π，以πnew作为π 转（2）。

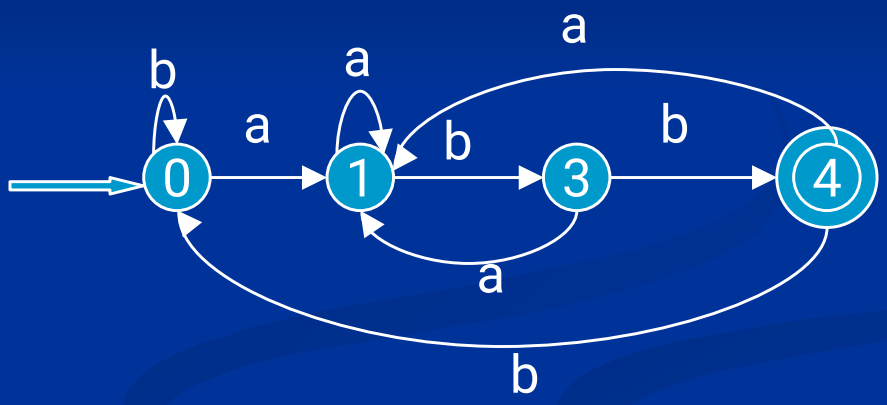
（3）考察：{0，2}a={1, 1} {0, 2}b={2, 2} 不能再分。

此时π_{new} = π ，分划结束。

（4）选状态0作为{0，2}的代表。 得化简的DFA M ‘如下：

回节 上一页 下一页

	a	b
0	1	0
1	1	3
3	1	4
4	1	0



3.4 正规式和有穷自动机的等价性

正规式和有穷自动机的等价性由以下两点说明。

1、对于∑上的每个正规式R，可以构造一个∑上的NFA M，使得L（M）=L（R）。

2、对于∑上的NFA M，可以构造一个∑上的正规式R，使得L（R）=L（M）。

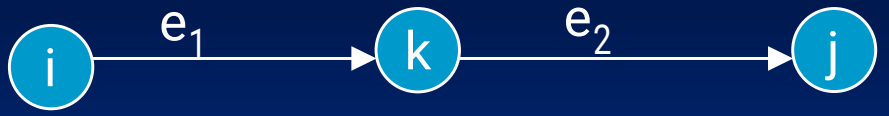
1. 对于∑上的每个正规式R，如何构造NFA M。

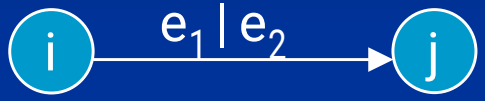
第一步，把正规式R表示成拓广转换图（即允许箭弧上标记正规式）

第二步，反复使用转换规则（加结规则），逐步把图变成每条弧标记为一个字符或ε，得到一个NFA M。

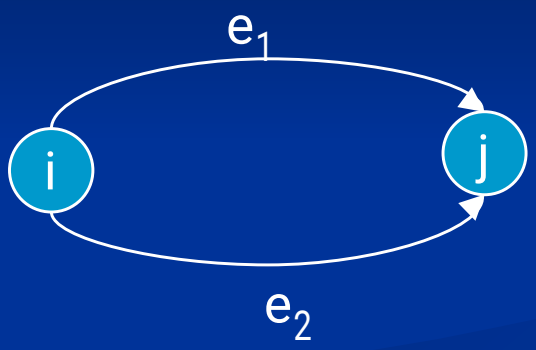
转换规则（即加结规则）：设e1, e2, e是 ∑上的正规式。

代之以

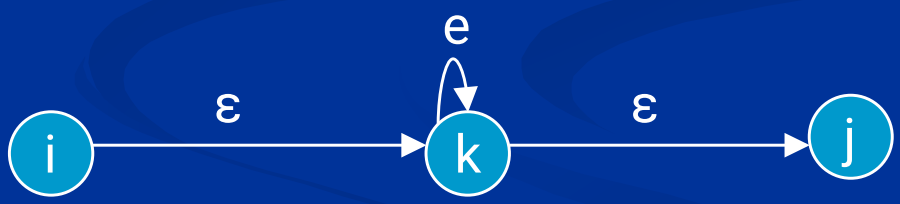




代之以

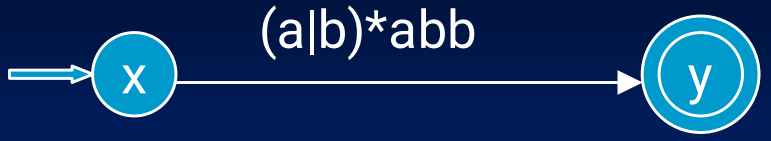


代之以

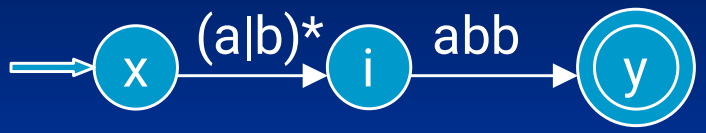


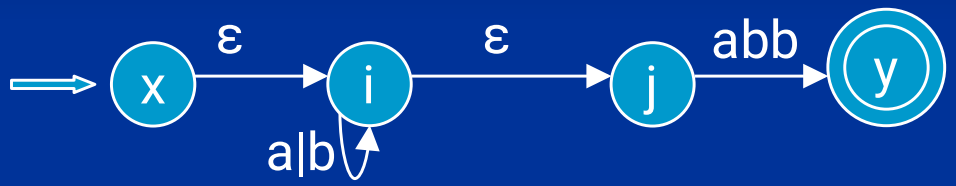
例：为R=(a|b)*abb构造NFA N使得L（N）=L（R）

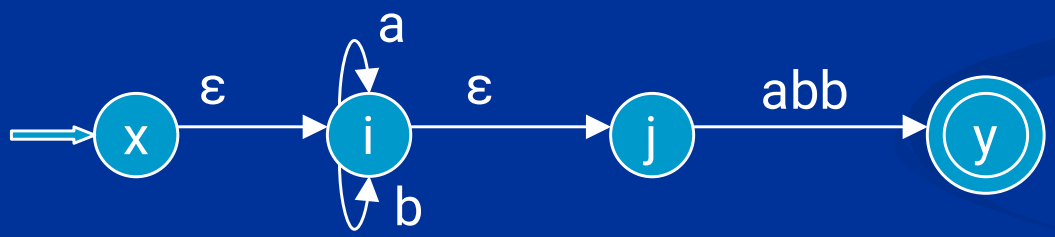
首先

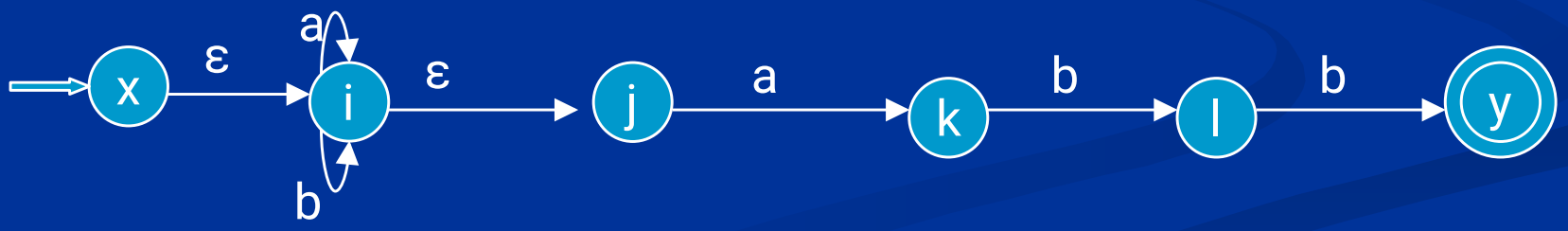


然后









构造出了NFA N

2. 对∑ 上的NFA M，构造 ∑上的正规式

对于∑ 上的NFA M，构造 ∑上的正规式R，可为上述方法的逆过程，通过不断地消结而得到。

应用正规式与有穷自动机的等价性。使用正规式描述单词的结构，而把正规式转换为NFA，进而转换为相应的DFA，再加预先编好的总控程序，就得到了词法分析程序。

如LEX就是基于这种方法自动构造词法分析程序的工具。

3.5 词法分析程序的编写

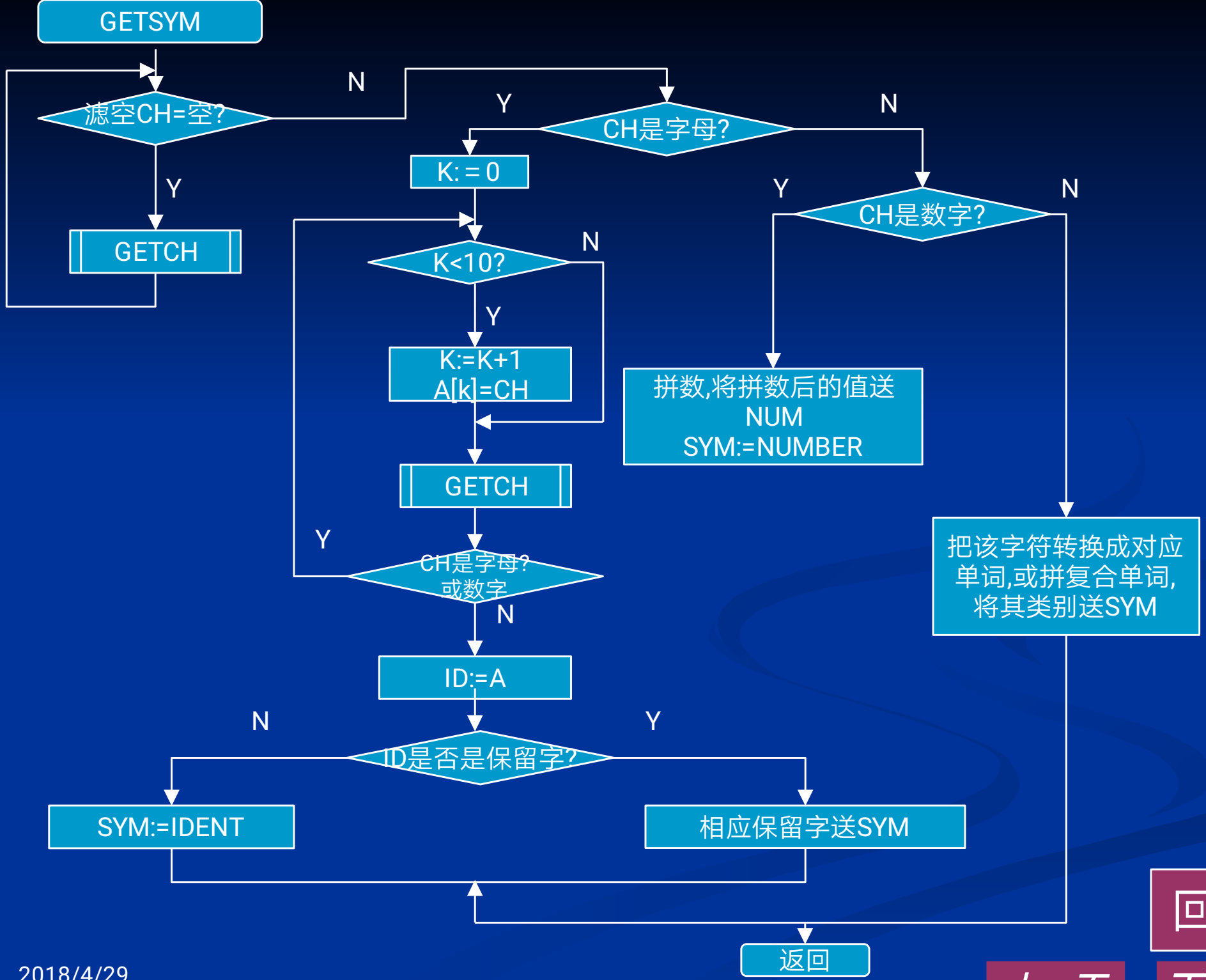
第一步 根据语言文法的BNF描述，列出语言中可能出现的各种单词。

第二步 作识别单词符号的状态转换图或程序流程图。

第三步 根据状态转换图或程序流程图编写程序。

例：PL/0 语言词法分析程序。

词法分析过程GETSYM的流程图。


GETSYM
滤空CH=空?
N
Y
CH是字母?
Y
N
K: = 0
Y
CH是数字?
N
GETCH
K<10?
N
Y
K:=K+1
A[k]=CH
拼数,将拼数后的值送
NUM
SYM:=NUMBER
GETCH
把该字符转换成对应
单词,或拼复合单词,
将其类别送SYM
Y
CH是字母?
或数字
N
ID:=A
N
Y
ID是否是保留字?
SYM:=IDENT
相应保留字送SYM
返回

回节
上一页
下一页

因此词法分析程序GETSYM将完成下列任务：

（1）滤空格：空格在词法分析时是一种不可缺少的界符，而在语法分析时则是无用的，所以必须滤掉。

（2）识别保留字：设有一张保留字表。对每个字母打头的字母、数字字符串要查此表。若查着则为保留字。将对应的类别放在SYM中。如IF对应值IFSYM， THEN对应为THENSYM。若查不着，则认为是用户定义的标识符。

（3）识别标识符：对用户定义的标识符将IDENT放在SYM中，标识符本身的值放在ID中。

（4）拼数：当所取单词是数字时，将数的类型NUMBER放在SYM中，数值本身的值存放在NUM中。

（5）拼复合词：对两个字符组成的算符。

如：>=、:=、<=等单词，识别后将类别送SYM中。

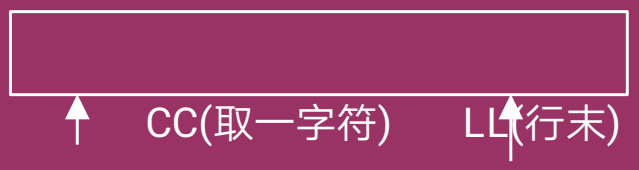
框图对应的词法分析程序如下：

注释

```
procedure getsym;
    var i, j, k: integer;
     procedure getch,
      begin
       if cc=LL
        then
         begin
          if eof (fi
           then
            begin
             write ('program
incomplete')
              goto 99
            end;
           LL:=0;
           CC:=0;
           write (cx: 4, ' ');
```

读入一字符

缓冲区LINE:




```
while not eoln (fin) do
          begin
           ll:=ll+l;
           read (fin, ch);
           write (ch);
           write (fal, ch);
           line [ll]:=ch
         end;
        writeln;
        ll:=ll+l;
        read (fin, line [ll]);
        writeln (fal);
       end;
      cc:=cc+1;
      ch:=line [cc]
    end (* getsym *)
```

读入一行到LINE缓冲区中

注释

```
begin (* getsym *)
 while ch=''do getch;
 if ch in [ a .. z ]
  then
   begin
    k:=0;
    repeat
     if k<al
      then
       begin
        k:=k+1;
        a[k]:=ch
       end;
      getch
    until not (ch in [ a .. z ,  0 .. 9 ])
;
```

```
if k>=kk
   then kk:=k
   else
    repeat
     a [kk]:=''  ;
     kk:=kk-1
    until kk=k;
   id:=a;
   i:=1;
   j:=norw;
   repeat
    k:=(i+j) div 2;
    if id<=word [k]
     then j:=k-1;
     if id > = word [k]
```

id or 保留字放数组a中

二分查找保留字

注释

```
I f id > = word [k]
 then i:=k+1
     until i>j;
   if i-1>j
     then sym:=wsym[k]
     else sym:=ident
   end
  else
  if ch in [ 0 .. 9 ]
   then
    begin (* number *)
     k:=0;
     num:=0;
     sym:=number;
```

number

```
repeat
                    num:
=10*num+(ord(ch)-          ord(
                  0 ));
     k:=k+1;
     getch
   until not(ch in [ 0 .. 9 ]);
   if k> nmax
    then error (30)
  end
 else
  if ch=' : '
   then
    begin
     getch;
```

注释

```
if ch=' = '
        then
         begin
          sym:=becomes;
          ge
          en
        else sym:=nul
       end
      else
      if ch= <
       then
        begin
         getch;
```

```
if ch= '= '
        then
        begin
         sym:=leq;
         getch
        end
      else sym:=lss
       end
      else
      if ch= '> '
       then
      begin
       getch,
```

等于种别

是否是赋值号

小于种别

注释

```
if ch= '= '
    then
     begin
      sym:=gel
       getch
      end
     else sym:=grt
   end
   else
   begin
    sym:=ssym[ch]
    getch
   end
end (* getsym *)
```

大于等于种别

大于种别

Ssym数组中存有:
+ - * / ( )
= , . # ;

保留字表:

```
word[1]:= begin     ; word[2]:= call       ;
word[3]:= const     ; word[4]:= do         ;
word[5]:= end     ; word[6]:= if         ;
word[7]:= odd     ; word[8]:= procedure      ;
word[9]:= read      ; word[10]:= then     ;
word[11]:= var      ; word[12]:= while      ;
word[13]:= write      ;
```

各保留字种别:

```
wsym[1]:=beginsym;        wsym[2]:=callsym;
wsym[3]:=constsym;        wsym[4]:=dosym;
wsym[5]:=endsym;   wsym[6]:=ifsym;
wsym[7]:=oddsym;     wsym[8]:=procsym;
wsym[9]:=readsym;    wsym[10]:=thensym;
wsym[11]:=varsym;    wsym[12]:=whilesym;
wsym[13]:=writesym;
```

```
ssym[ '+' ]:=plus;          ssym[ ' - ' ]:=minus;
ssym[ '* ']:=times;       ssym[ '/ ']:=slash;
ssym[' ( ']:=lparen;        ssym[' ) ']:=rparen;
ssym[ '= ']:=eql;         ssym[ ',' ]:=comma;
ssym[ '. ']:=period;        ssym[ '#' ]:=neq;
ssym[ '; ']:=semicolon;
```

如：分析 If ab=0 then x:=350

返回：

if: sym : ifsym
ab: sym ident, id ab
=: sym eql
0 sym number, num 0
x sym ident, id x
:= sym becomes
350 sym number, num 350

本章学习要点：

正规文法和正规式的概念。

确定有穷自动机和非确定有穷自动机的概念。

根据正规式求非确定有穷自动机。

非确定有穷自动机确定化。

确定有穷自动机最小化.

词法分析程序的设计。