第三章 词法分析

重点： 词法分析器的输入、输出，
用于识别符号的状态转移图的构造，
根据状态转移图实现词法分析器。

难点： 词法的正规文法表示、正规表达式表示、
状态转移图表示，它们之间的转换。

第3章 词法分析

3.1 词法分析器的功能

3.2 单词的描述

3.3 单词的识别

3.4 词法分析程序的自动生成

3.5 本章小结

3.1 词法分析器的功能

- 功能：输入源程序，输出单词符号(token)。
  即：把构成源程序的字符串转换成“等价的”单词(记号)序列
  - 根据词法规则识别及组合单词，进行词法检查
  - 对数字常数完成数字字符串到二进制数值的转换
  - 删去空格字符和注释

3.1.1 单词的分类与表示 & 3.1.2 词法分析器的输出

一、单词的种类

1. 关键字:也称基本字，begin、end、for、do...

2. 标识符:由用户定义，表示各种名字

3. 常数:整常数、实常数、布尔常数、字符串常数等

4. 运算符:算术运算符+、-、*、/等；逻辑运算符not、or与and等；关系运算符=、<>、>=、<=、>和<等

5. 分界符:，、；、（、）...

二、单

表示单词的种类，可用整数编码或记忆符表示

不同的单词不同的值

种别	属性值

几种常用的单词内部形式：

1、按单词种类分类

2、保留字和分界符采用一符一类

3、标识符和常数的单词值又为指示字(指针值)

1、按单词种类分类

单词名称	类别编码	单词值
标识符	1	内部字符串
无符号常数(整)	2	整数值
无符号浮点数	3	数值
布尔常数	4	0 或 1
字符串常数	5	内部字符串
保留字	6	保留字或内部编码
分界符	7	分界符或内部编码

•*

2、保留字和分界符采用一符一类

单词名称	类别编码	单词值
标识符	1	内部字符串
无符号常数(整)	2	整数值
无符号浮点数	3	数值
布尔常数	4	0 或 1
字符串常数	5	内部字符串
BEGIN	6	-
END	7	-
FOR	8	-
DO	9	-
……	……	……
:	20	-
+	21	-
*	22	-
,	23	-
(	……	-

例3.1 语句if count>7 then result := 3.14 的单词符号序列

(IF，0)

(ID，指向count 的符号表入口)

(GT，0)

(INT，7)

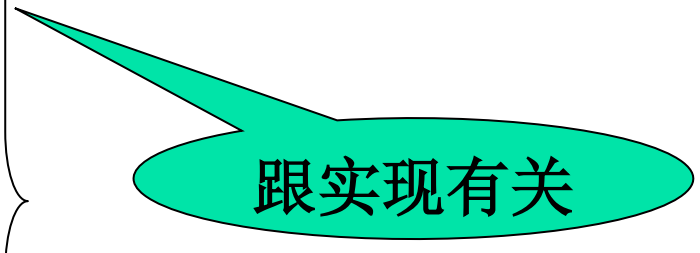
(THEN，0)

(ID，指向result的符号表入口)

(ASSIGN，0)

(REAL，3.14)

(SEMIC，0)

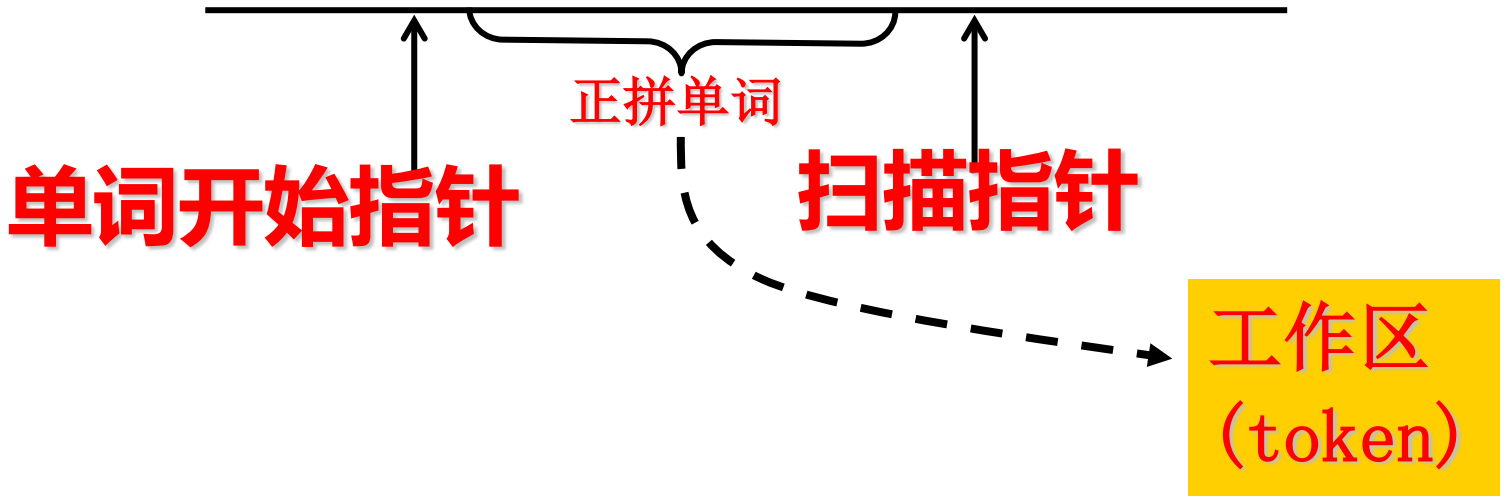


3.1.3 源程序的输入缓冲与预处理

- 超前搜索和回退
  - 双字符运算符（**, /*,:=,...）
  - DO 90 k=1,10
  - DO 90 k=1.10
- 缓冲区
  - 假定源程序存储在磁盘上，这样每读一个字符就需要访问一次磁盘，效率显然是很低的。
- 空白字符的剔除
  - 剔除源程序中的无用符号、空格、换行、注释等

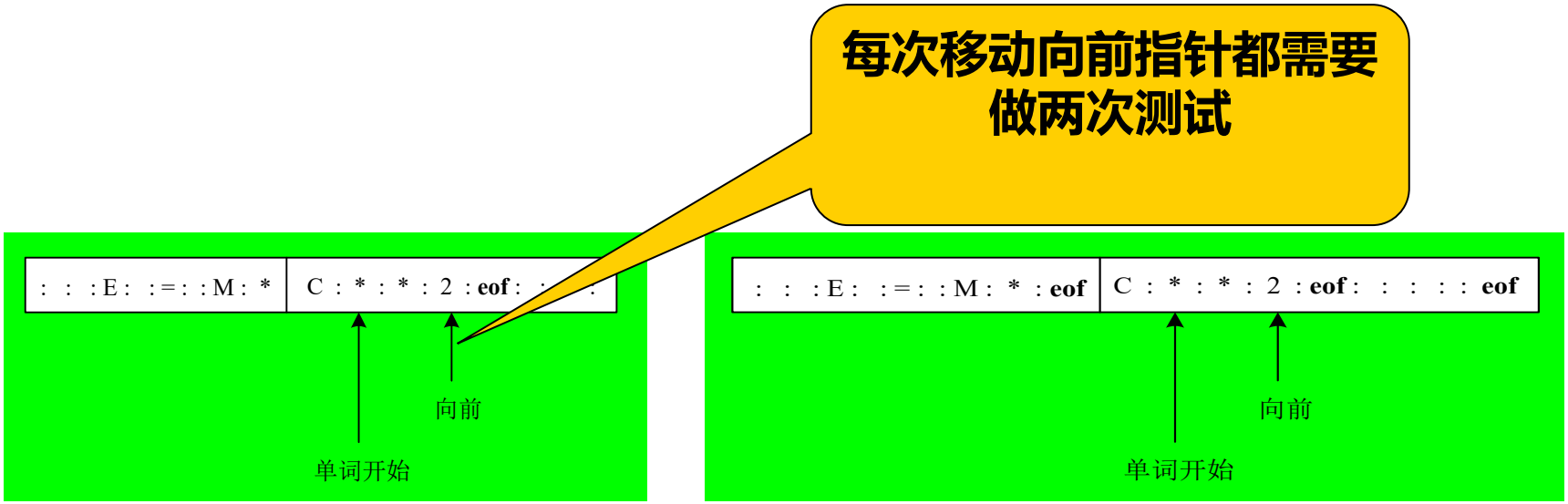
3.1.3 源程序的输入缓冲与预处理(续)

- 输入缓冲区



3.1.3 源程序的输入缓冲与预处理(续)

双缓冲区问题__超前扫描导致的效率问题



3.1.4 词法分析阶段的错误处理

1．非法字符检查

2．关键字拼写错误检查

3．不封闭错误检查

4．重复说明检查

5．错误恢复与续编译

紧急方式恢复(panic-mode recovery)

反复删掉剩余输入最前面的字符，直到词法分析器能发现一个正确的单词为止。

3.1.5 词法分析器的位置

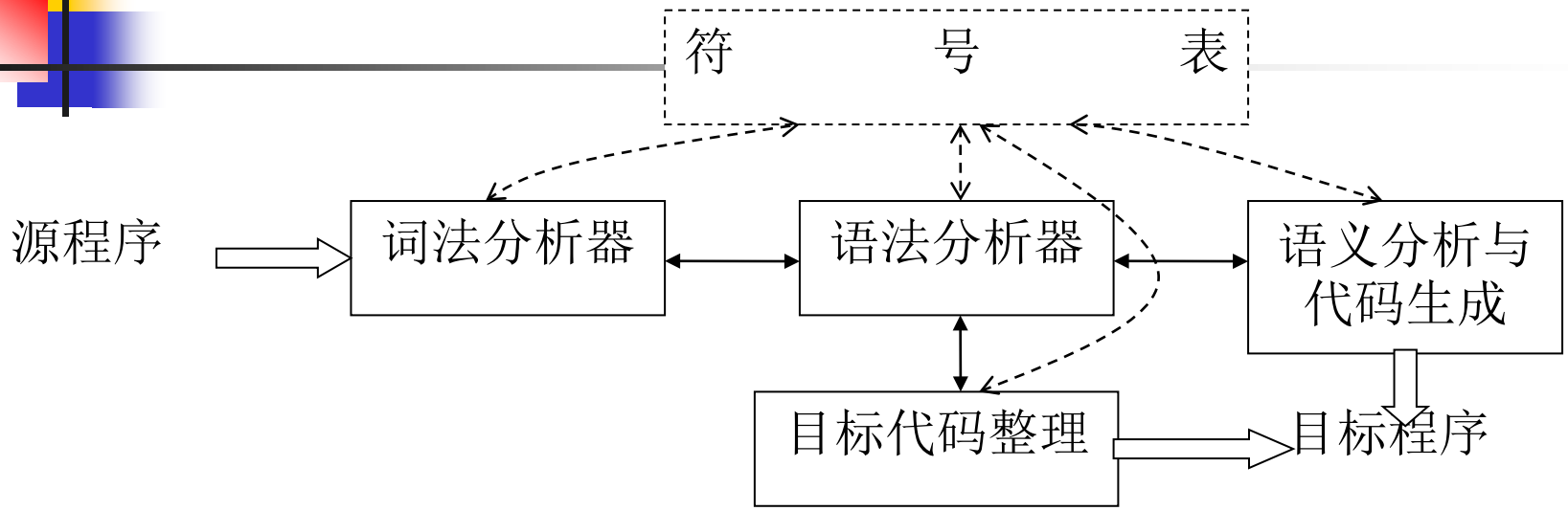


图3.4　以语法分析器为中心

- **以词法分析作为独立的阶段：**
  - **简化编译器的设计。**
  - **提高编译器的效率。**
  - **增强编译器的可移植性。**

3.2 单词的描述
3.2.1 正则文法

- 正则文法$G=(V，T，P，S)$中，对$\forall\alpha\to\beta\in P$，$\alpha\to\beta$均具有形式$A\to w$或$A\to wB(A\to w$或$A\to Bw)$，其中$A$，$B\in V$，$w\in T^{+}$。
- 例3.2 标识符的文法
  - 约定：用digit表示数字：0,1,2,…,9;
    用letter表示字母：A,B,…,Z,a,b,…,z
  - <id> → <letter> | <id><digit> | <id><letter>
  - <letter> → *A* | *B* | … | *Z* | *a* | *b* | … | *z*
  - <digit> → 0 | 1 | 2 | … | 9

3.2.2 正则表达式

- 例3.2：标识符的另一种表示
  - letter(letter|digit)*
  - | 表示"或"
  - * 表示Kleene闭包
  - + 表示正闭包
  - ？ 表示“0或1个”
  - letter和(letter|digit)*的并列表示两者的连接
- 正则式r表示正则集,相应的正则集记为 L(r)

3.2.2 正则表达式(续)—RE

定义3.1 设$\sum$是一个字母表，则$\sum$上的正则表达式及其所表示的正则语言可递归地定义如下：

(1) $\varnothing$是$\sum$上的一个正则表达式，它表示空集；

(2) ε是$\sum$上的一个正则表达式，它表示语言$\{\varepsilon\}$；

(3) 对于$\forall a(a \in \sum)$，a是$\sum$上的一个正则表达式，它表示的正则语言是$\{a\}$；

(4) 假设r和s都是$\sum$上的正则表达式，它们表示的语言分别为$L(r)$和$L(s)$，则：

① (r)也是$\sum$上的正则表达式，它表示的语言为$L(r)$；

② $(r|s)$也是$\sum$上的正则表达式，它表示的语言为$L(r) \cup L(s)$；

③ $(r \bullet s)$也是$\sum$上的正则表达式，它表示的语言为$L(r)L(s)$；

④ (r^*)也是$\sum$上的正则表达式，它表示的语言为$(L(r))^*$；

(5) 只有经有限次使用上述规则构造的表达式才是$\sum$上的正则表达式。

正则表达式中的运算优先级

运算优先级和结合性:

- *高于“连接”和|，“连接”高于|
- |具有交换律、结合律
- “连接” 具有结合律、和对|的分配律
- ( )指定优先关系

意义清楚时，括号可以省略

例:

1. $L((a|b)^*)=L((a^*|b^*)^*)=\{x|x$是$a$和$b$构成的符号串，包括$\varepsilon\}$
2. $L(a|a^*b)=\{a, b, ab, aab, aaab, aaaab,...\}$

3.2.3 正则表达式与正则文法的等价性

- 正则表达式与正则文法等价
  - 对任意一个正则表达式，存在一个定义同一语言的正则文法
  - 对任意一个正则文法，存在一个定义同一语言的正则表达式

•*

根据正则文法构造等价的正则表达式

- **问题：给定正则文法G，构造一个正则表达式r，使得$L(r) = L(G)$**
- **基本思路**
  - **为正则文法的每个产生式构造一个正则表达式方程式，这些方程式中的变量是文法G中的语法变量，各变量的系数是正则表达式，简称为方程式。从而得到一个联立方程组。**
  - **用代入消元法消去联立方程组中除开始符号外的其他变量，最后得到关于开始符号S的解：$S = r$，r即为所求的正则表达式。**

根据正则文法构造等价的正则表达式

- **具体步骤**

(1) 根据正则文法G构造正则表达式联立方程组。

假设正则文法G是右线性的，其每个产生式的右部只含有一个终结符，则有如下方程式构造规则：

① 对形如$A\rightarrow a_1|a_2|\ldots|a_m$的产生式，构造方程式$A=a_1|a_2|\ldots|a_m$。其中可以有形如$A\rightarrow\varepsilon$的产生式；

② 对形如$A\rightarrow a_1A|a_2A|\ldots|a_mA$的产生式，构造方程式$A=(a_1|a_2|\ldots|a_m)^*A$；

③ 对形如$A\rightarrow a_1B|a_2B|\ldots|a_mB$的产生式，构造方程式$A=(a_1|a_2|\ldots|a_m)B$，其中$B\neq A$。

•*

根据正则文法构造等价的正则表达式

(2)解联立方程组，求等价的正则表达式r

用代入消元法逐个消去方程组中除开始符号S外的其他变量，最后即可得到关于开始符号S的解。代入消元规则如下：

① 如果有$A=r_1B|r_2B|\ldots|r_nB$，则用$A=(r_1|r_2|\ldots|r_n)B$替换之，其中$B\neq A$；

② 如果有$A=t_1A|t_2A|\ldots|t_mA$，则用$A=(t_1|t_2|\ldots|t_m)^*A$替换之；

•*

根据正则文法构造等价的正则表达式

③ 如果有$A=(r_1|r_2|\ldots|r_n)B$，$B=(t_1|t_2|\ldots|t_m)C$，则用$A=(r_1|r_2|\ldots|r_n)\ (t_1|t_2|\ldots|t_m)C$替换之，其中$B\neq A$；

如果有$A=(r_1|r_2|\ldots|r_n)B$，$B=(t_1|t_2|\ldots|t_m)$，则用$A=(r_1|r_2|\ldots|r_n)\ (t_1|t_2|\ldots|t_m)$替换之，其中$B\neq A$；

④ 对$A=(t_1|t_2|\ldots|t_m)^*A$且$A=(r_1|r_2|\ldots|r_n)B$，其中$B\neq A$，则用$A=(t_1|t_2|\ldots|t_m)^*\ (r_1|r_2|\ldots|r_n)B$替换之；

对$A=(t_1|t_2|\ldots|t_m)^*A$且$A=r_1|r_2|\ldots|r_n$则用$A=(t_1|t_2|\ldots|t_m)^*\ (r_1|r_2|\ldots|r_n)$替换之；

根据正则文法构造等价的正则表达式

⑤ 如果有$A=\beta_1$、$A=\beta_2$、…、$A=\beta_h$，则用$A=\beta_1|\beta_2|\ldots|\beta_h$代替之。

如果最后得到的关于S的方程式为如下形式，

$$S=\alpha_1|\alpha_2|\ldots|\alpha_h$$

则将方程式右边所有其中仍然含有语法变量的$\alpha_i(1\le i\le n)$删除，得到的结果就是与G等价的正则表达式；如果任意的$\alpha_i(1\le i\le n)$均含有语法变量，则∅就是与G等价的正则表达式。

•*

根据正则文法构造等价的正则表达式

- 例3.6 将如下文法G转换成相应的正则表达式

$S \rightarrow aS|aB$

$B \rightarrow bB|bC|aB|bS$

$C \rightarrow cC|c$

1.列方程组

- $S=a^*S \quad S=aB$
- $B=(a|b)^*B \quad B=bC \quad B=bS$
- $C=c^*C \quad C=c$

2.代入法解方程组

- $C=c^*c$
- $B=(a|b)^*bC|(a|b)^*bS=(a|b)^*(bc^*c)|(a|b)^*bS$
- $S=a^*aB$
- $S=a^*a(a|b)^*(bc^*c)|a^*a(a|b)^*bS$
- $S=(a^*a(a|b)^*b)^*a^*a(a|b)^*(bc^*c)$
- 如果用正闭包表示，则为$(a^+(a|b)^*b)^*a^+(a|b)^*(bc^+)$

根据正则文法构造等价的正则表达式

- **练习：将如下文法*G*转换成相应的正则表达式**

G: S→aS
S→aA
A→bB
B→aB
B→a

$B = a^*a$
$A = ba^*a$
$S = a^*S \quad S = aba^*a$
$S = a^*aba^*a$
$S = a^+ba^+$

•*

将正则表达式转换成等价的正则文法

- **问题：给定∑上的一个正则表达式r，根据r构造正则文法G，使得L(G)=L(r)**
- **定义3.3 设字母表为∑，{*A*、*B*、…、*C*}为语法变量集合，对于∑上的任意正则表达式*r*，形如*A*→*r*的式子称为正则定义式；如果r是∑中的字母和用正则定义式定义的变量组成的正则表达式，则形如A→r的式子称为正则定义式。**

将正则表达式转换成等价的正则文法

- 按如下方法构造正则定义式，并逐步将其转换成正则文法
- 引入开始符号S，从如下正则定义式开始
  - $S \rightarrow r$
- 按如下规则将$S \rightarrow r$分解为新的正则定义式，在分解过程中根据需要引入新的语法变量

•*

将正则表达式转换成等价的正则文法

- $A \to r$是正则定义式，则对$A \to r$的分解规则如下：

  (1) 如果$r=r_1 r_2$，则将$A \to r$分解为$A \to r_1 B$，$B \to r_2$，$B \in V$；

  (2) 如果$r=r_1^* r_2$，则将$A \to r$分解为$A \to r_1 A$，$A \to r_2$；

  (3) 如果$r=r_1 | r_2$，则将$A \to r$分解为$A \to r_1$，$A \to r_2$。

  不断应用分解规则(1)到(3)对各个正则定义式进行分解，直到每个正则定义式右端只含一个语法变量(即符合正则文法产生式的形式)为止。

例3.9 正则表达式到正则文法的转换

- 将正则表达式$a(a|b)^*$转换为正则文法
- $S \rightarrow a(a|b)^*$

$S \rightarrow aA$　　$A \rightarrow (a|b)^*$

$A \rightarrow (a|b)A$　　$A \rightarrow \varepsilon$

$A \rightarrow aA|bA$

等价正则文法：

$S \rightarrow aA$

$A \rightarrow aA|bA|\varepsilon$

例3.9 正则表达式到正则文法的转换

- $a(a|b)^*(\varepsilon|((.|\_)(a|b)(a|b)^*))$

$S\rightarrow aA|aB \quad A\rightarrow aA|bA|\varepsilon$

$B\rightarrow aB|bB|.C|\_C$

$C\rightarrow aA|bA$

$S\rightarrow a(a|b)^*(\varepsilon|((.|\_)(a|b)(a|b)^*))$

$S\rightarrow a(a|b)^*$ $\quad$ $S\rightarrow a(a|b)^*(.|\_)(a(a|b)^*|b(a|b)^*)$

S→ aA $\quad$ A→(a|b)*

A→aA|bA|ε

S→ aB $\quad$ B→(a|b)*(.|_)(a(a|b)*|b(a|b)*)

B→(a|b)B $\quad$ B→(.|_)(a(a|b)*|b(a|b)*)

B→(.|_)C $\quad$ C→a(a|b)*|b(a|b)*

B→.C|_C $\quad$ C→aA|bA

例 3.10 标识符定义的转换

S=letter (letter|digit)*

- 引入 S

  $S \rightarrow$ letter (letter|digit)*

- 分解为

  $S \rightarrow$ letter A

  $A \rightarrow$ (letter|digit)A|ε

- 执行连接对|的分配律

  $S \rightarrow$ letter A

  $A \rightarrow$ letter A|digit A|ε

例 3.10 标识符定义的转换

- 练习：将(a|b)*a(a|b)(a|b) 转换成正则文法

S→(a|b)*a(a|b)(a|b)

S→(a|b)S　　S→a(a|b)(a|b)

S→aS|bS　　S→aA

A→(a|b)(a|b)

A→(a|b)B　　B→(a|b)

A→aB|bB　　B→a|b

S→aS|bS
S→aA
A→aB|bB
B→a|b

高级语言词法的简单描述

- 词法
  - 单词符号的文法，用来描述高级语言中的：标识符、常数、运算符、分界符、关键字
- 参考教材P73-77，了解如何定义高级语言中的整数、实数……等的相应正则文法。

例 3.7 某简易语言的词法
——正则定义式

词法规则	单词种别	属性
<标识符>→<字母>(<字母>|<数字>)*	IDN	符号表入口
<无符号整数>→ <数字> (<数字>)*	NUM	数值
<赋值符>→ :=	ASG	无
其它单词→字符本身	单词名称	无

变换为正规文法

<标识符>→letter<标识符尾>

<标识符尾>→ε|letter<标识符尾>|digit<标识符尾>

<整数>→digit <整数尾>

<整数尾>→ε| digit<整数尾>

<赋值号>→:<赋值尾>

<赋值尾> →=

<加号>→+

<等号>→=

…

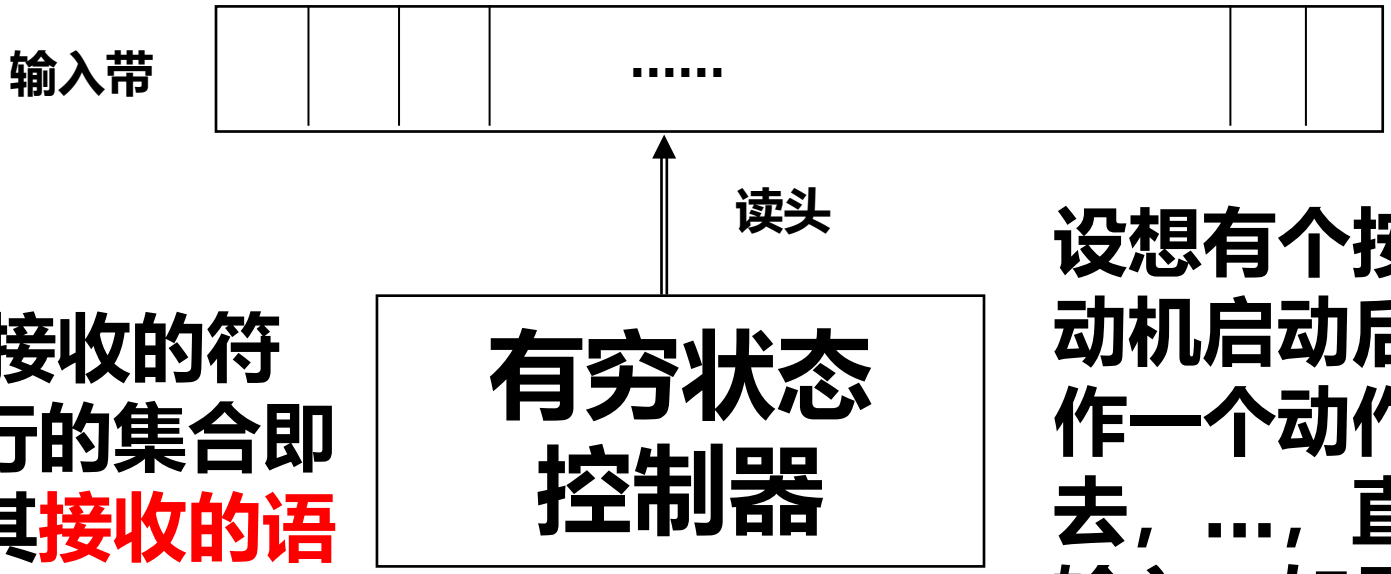
(其它：实数、算术运算符、关系运算符、分号、括号等)

问题：如何识别记号?

3.2.4 有穷状态自动机

- 具有离散输入输出的系统的数学模型
- 具有有穷个内部状态
- 系统只需根据当前所处的状态和面临的输入就能确定后继的行为，处理完当前输入后系统的状态将发生变化
- 具有初始状态和终止状态
- 例：文本编辑程序、词法分析程序……

有穷自动机的物理模型



FA接收的符号行的集合即为其**接收的语言**

设想有个按钮，自动机启动后一个动作一个动作地做下去，…，直到没有输入。如果停在终态，接收；如果停在非终态，不接收

一个动作：

[p，a]→q，读头前进一格

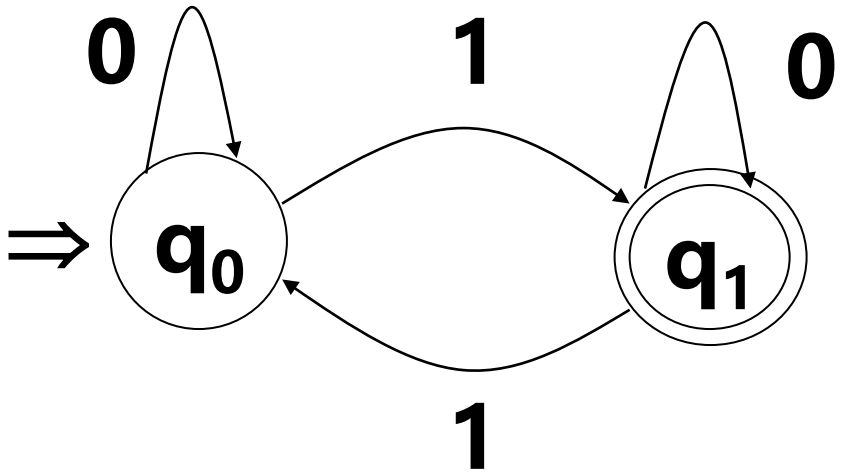
有穷自动机的用处

- 有穷自动机是许多重要类型的硬件和软件的有用模型
  - 数字电路的设计和检查软件
  - 典型编译器的词法分析器
  - 扫描大量文本来发现单词、短语或其他模式的出现的软件
  - 所有只有有穷个不同状态的系统（如通信协议或安全交换信息的协议）的验证软件

•*

例： 一个奇偶校验器

测试输入中1的个数的奇偶性，并且只接收含有奇数个1的那些输入串。



注意：状态有记忆功能，记住输入串的部分特征。

问题：有穷自动机的形式描述?

关键是如何描述动作?

确定的有穷自动机的形式定义

定义3.4 一个**确定的有穷自动机** M（记作DFA M）是一个五元组$M=(Q, \Sigma, \delta, q_0, F)$，其中

① Q是一个有穷状态集合。

② Σ是一个字母表，它的每个元素称输入符号。

③ $q_0 \in Q$，q_0 称为初始状态。

④ $F \in Q$，F称为终止状态集合。

⑤ δ是一个从 $Q \times \Sigma$到 Q的单值映射

$\delta(p, a) = q \quad (p, q \in Q, a \in \Sigma)$

表示当前状态为p,输入符号为a时，自动机将转换到下一个状态q，q称为p的一个后继。

•*

DFA的表示

例 设DFA $M=(\{0,1,2,3\},\{a,b\},\delta,0,\{3\})$

其中:

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

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

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

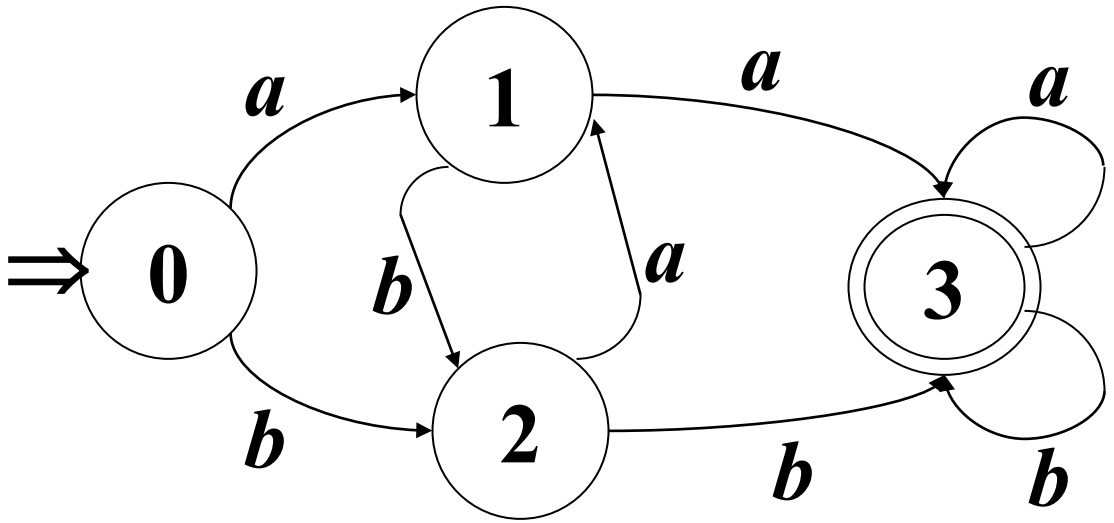
$\delta(2,b)=3,\ \delta(3,b)=3$

- ✓ 一个DFA有三种表示:
  (1) 转换函数;
  (2) 转移矩阵;
  (3) 状态转换图。

转移矩阵

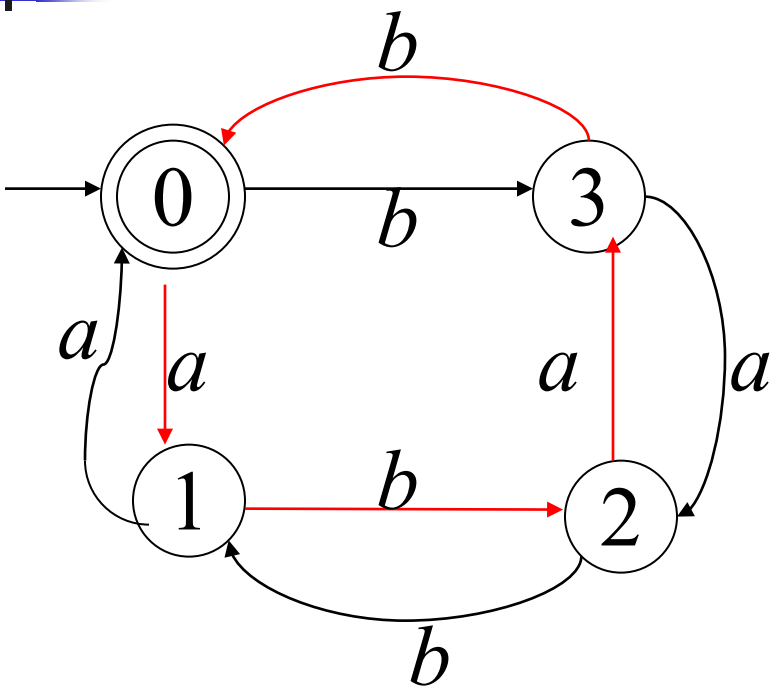
状态转换图

	a	*b*
0	1	2
1	3	2
2	1	3
3	3	3



易存储

DFA *M*接受的语言



从状态转换图看，从初态出发，沿任一条路径到达接受状态，这条路径上的弧上的标记符号连接起来构成的符号串被接受。

如：*abab*

问题：如何形式描述DFA接收的语言?

DFA M接受的语言

如果对所有$w\in\Sigma^*$，$a\in\Sigma$，$q\in Q$以下述方式递归地扩展δ的定义$\hat{\delta}(q,\varepsilon)=q$，

$$\hat{\delta}(q,wa)=\delta(\hat{\delta}(q,w),a)$$

则M所接收的语言为：

$$L(M)=\{w \mid w\in\Sigma^*, \text{且}\delta(q_0,w)\in F\}$$

对于上页例中的DFA M和$w=baa$，

$$\delta(0,baa)=\delta(2,aa)=\delta(1,a)=3$$

非确定的有穷自动机NFA M

定义3.6 非确定的有穷自动机M是一个五元组

$$M = (Q, \Sigma, \delta, q_0, F)$$

其中Q，Σ，q_0，F的意义和DFA的定义一样，而δ是一个从$Q\times\Sigma\cup\{\varepsilon\}$到$Q$的子集的映射，即$\delta$：$Q\times S\rightarrow 2^Q$，其中$S=\Sigma\cup\{\varepsilon\}$。

类似于DFA，NFA M亦可用状态转换图表示，同样也可以定义NFA M接受的语言。

DFA M的模拟算法

输入：以eof结尾的串x，DFA $M = (Q, \Sigma, \delta, q_0, F)$；

输出：如果M接受x则输出“yes”，如果M不接受x则输出“no”

步骤：

```
1 s = q0;
2 c = getchar(x);
3 while (c != eof) {
4     s = move(s, c);
5     c = getchar(x);
6 }
7 if s∈F return “yes”
8 else return “no”;
```


•*

3.2.5 状态转换图

- 定义3.7 设$M=(Q, \sum, \delta, q_0, F)$为一个有穷状态自动机，满足如下条件的有向图被称为$M$的状态转换图(transition diagram)：

(1) $q \in Q \Leftrightarrow q$是该有向图中的一个顶点；

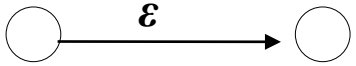
(2) $\delta(q, a)=p \Leftrightarrow$ 图中有一条从顶点q到顶点p的标记为a的弧；

(3) $q \in F \Leftrightarrow$ 标记为q的顶点被用双层圈标出；

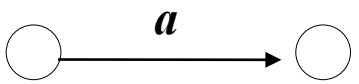
(4) 用标有start的箭头指出M的开始状态。

3.2.6 正则表达式转换为状态转换图

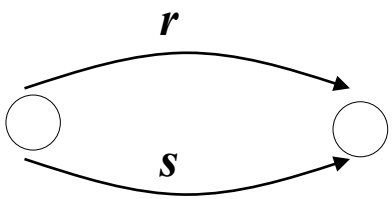
(a) ∅对应的状态转换图



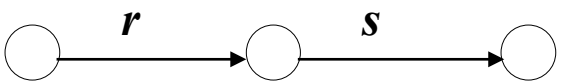
(b) ε对应的状态转换图



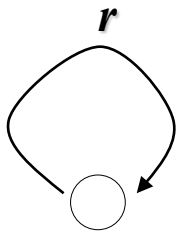
(c) *a*对应的状态转换图



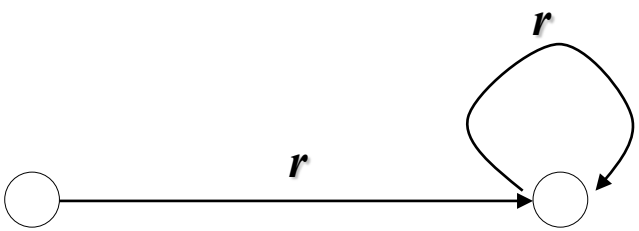
(d) *r* | *s*对应的状态转换图



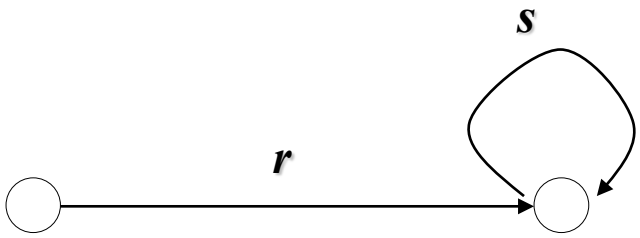
(e) *rs*对应的状态转换图



(f) r^*对应的状态转换图



(g) r^+对应的状态转换图



(h) rs^*对应的状态转换图

图3.8 典型正则表达式对应的状态转换图

3.2.6 正则表达式转换为状态转换图

- 转换过程如下：
  - 设置一个开始状态和一个终止状态，从开始状态到终止状态引一条标记为待转换表达式的边；
  - 检查图中边的标记，如果相应的标记不是字符、∅、ε或用“|”连接的字符和ε，则根据规则(1)-进行替换，直到图中不再存在不满足要求的边。按照习惯，如果一条边上标记的是∅，这个边就不用画出来。

3.3 单词的识别

3.3.1有穷状态自动机与单词识别的关系

- 有穷状态自动机和正则文法等价，考虑到状态转换图的直观性，我们从状态转换图出发来考虑词法分析器的设计。
- 允许在状态转换图的边上标记像digit、 letter这样意义明确的符号，other表示例外情况

3.3.1有穷状态自动机与单词识别的关系

- 考虑到在识别单词的过程中需要执行一些动作，所以将这些动作标记标在基本的状态转换图上。
- 如果到达终止状态，则意味着读入了一个与当前单词无关的字符，由于这个无关字符是下一个单词的开始符号，所以必须回退一个字符。状态上的*表示向前指针必须回退一个字符。

例 3.14 不同进制无符号整数的识别

八进制数:　　(OCT，值)

- **oct→0(0|1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)***

十进制数:　　(DEC，值)

- **dec→(1|...|9)(0|...|9)* |0**

十六进制数:　(HEX，值)

- **hex→0x(0|1|...|9|a|...|f|A|…|F)(0|...|9|a|...|f |A|…|F)***

识别不同进制数的状态图

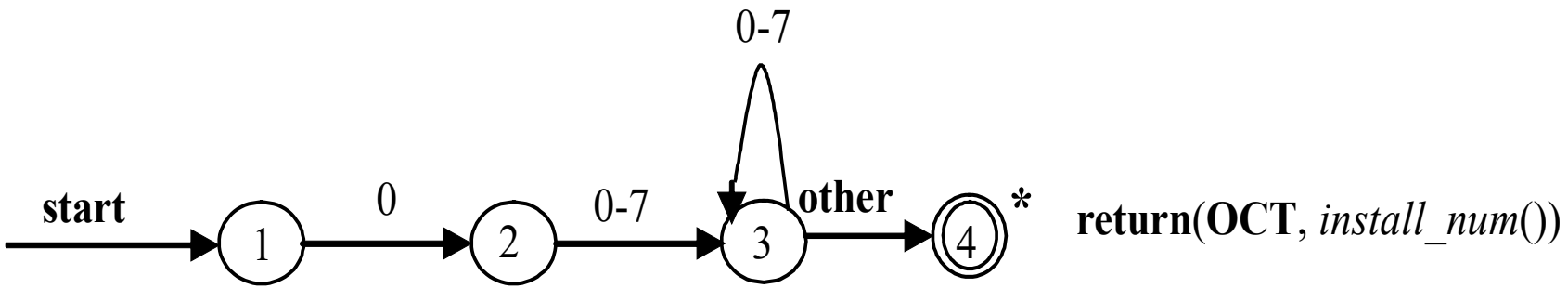


图3.11 识别八进制无符号整数的状态转换图

•*

*

识别不同进制数的状态图

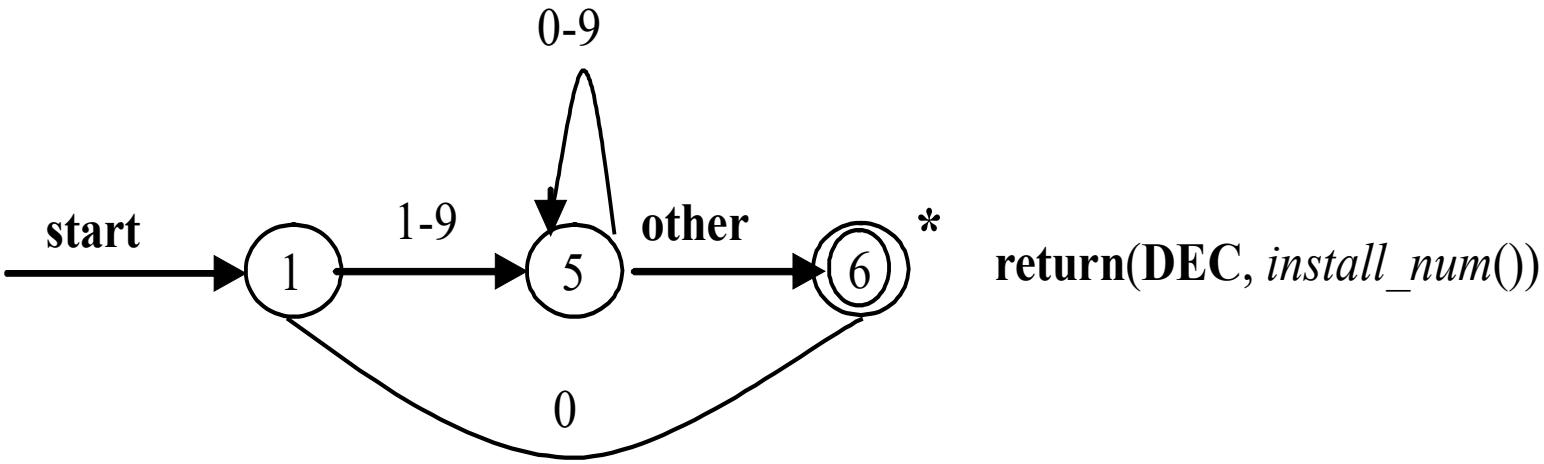


图3.12 识别十进制无符号整数的状态转换图

•*

*

识别不同进制数的状态图

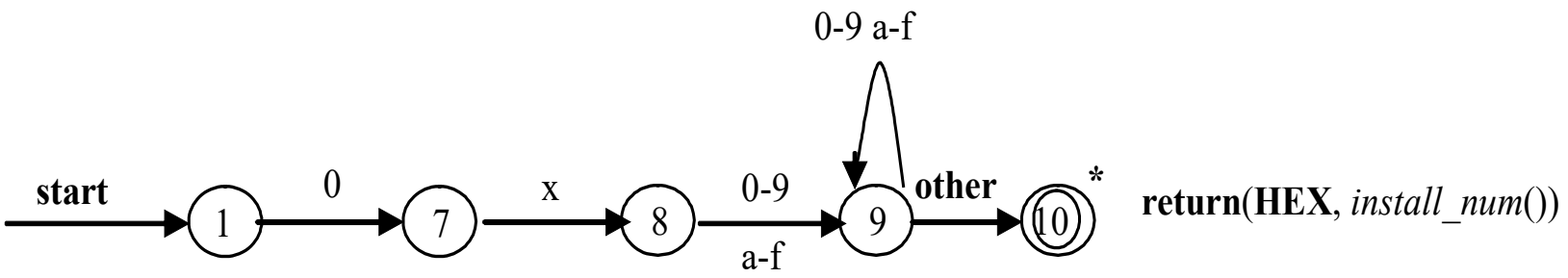


图3.13 识别十六进制无符号整数的状态转换图

•*

*

识别不同进制数的状态图

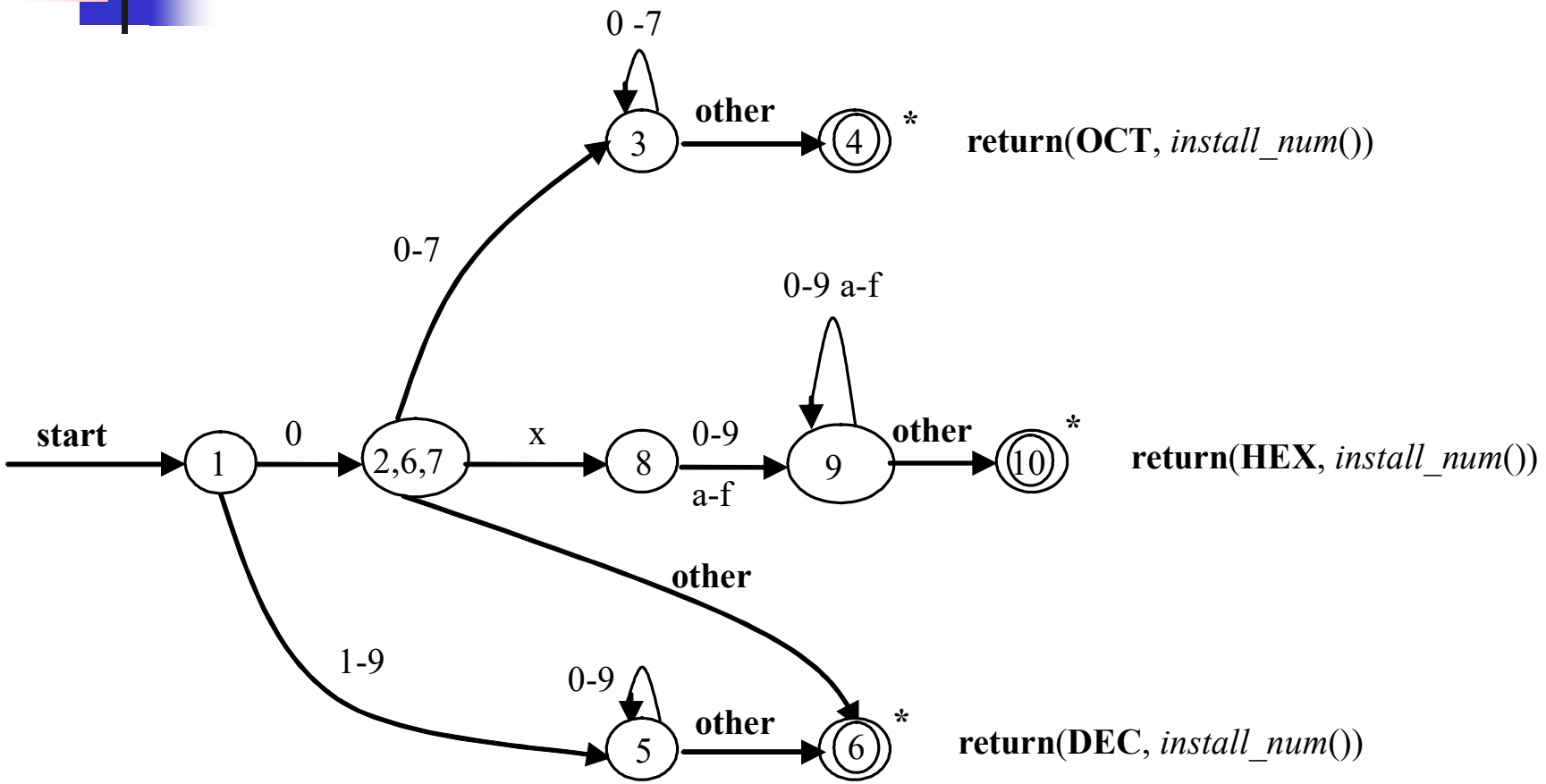


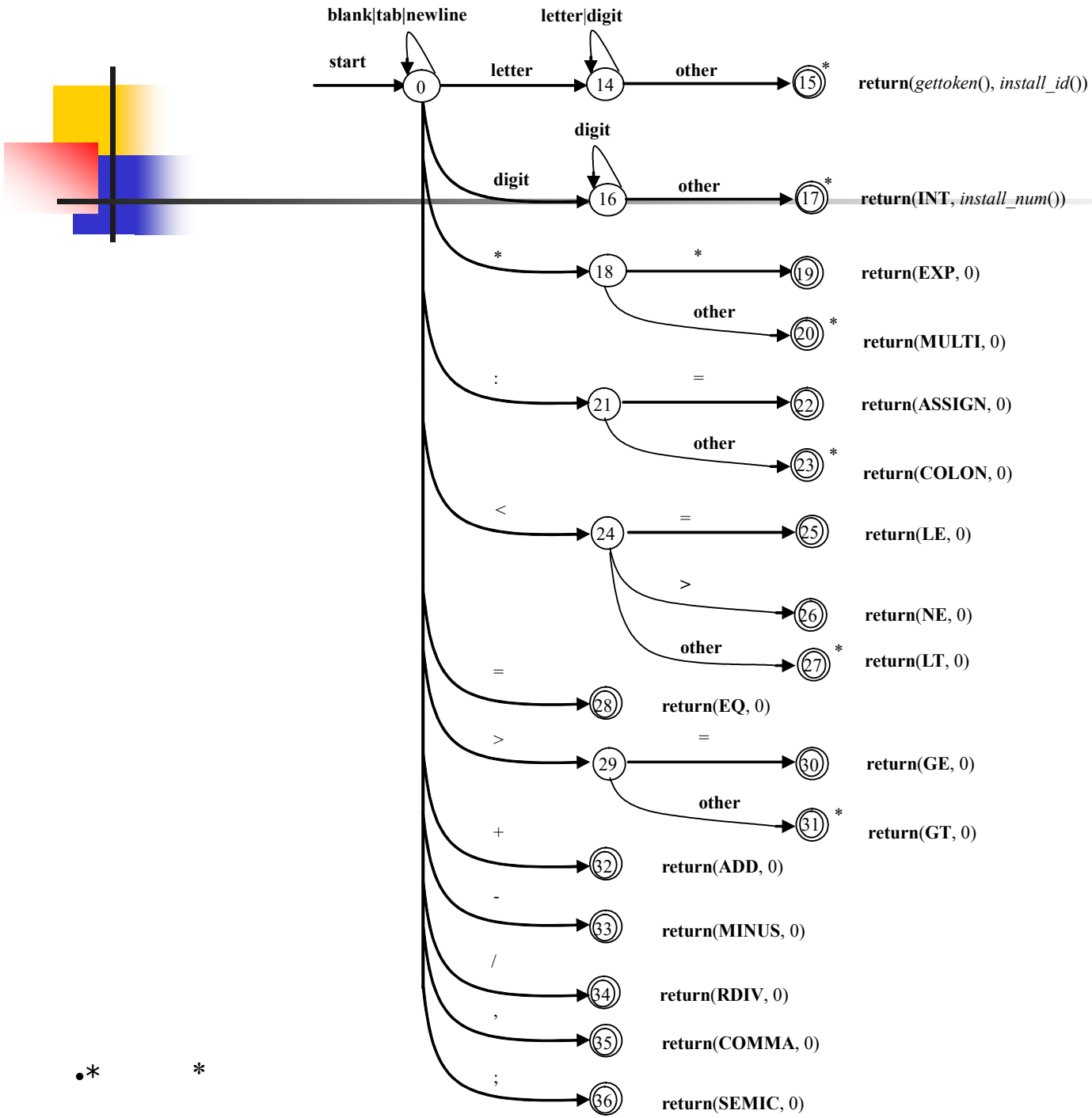
图3.14 识别C语言不同进制无符号整数的状态转换图

•*

*

3.3.2 单词识别的状态转换图表示

- <id> → letter <id_left>
- <id_left> →ε| letter <id_left> | digit <id_left>
- <int> → digit <int_left>
- <int_left> →ε| digit <int_left>
- <assignment> → :=
- <relop> → < | <= | = | < > | > | >=
- <op> → + | - | * | / | **
- <delimiter> → : | , | ;


blank|tab|newline
letter|digit
start
0
letter
14
other
15
*
return(gettoken(), install_id())
digit
digit
16
other
17
*
return(INT, install_num())
*
18
*
19
return(EXP, 0)
other
20
*
return(MULTI, 0)
:
=
21
22
return(ASSIGN, 0)
other
23
*
return(COLON, 0)
<
24
=
25
return(LE, 0)
>
26
return(NE, 0)
other
27
*
return(LT, 0)
=
28
return(EQ, 0)
>
=
29
30
return(GE, 0)
other
31
*
return(GT, 0)
+
32
return(ADD, 0)
-
33
return(MINUS, 0)
/
34
return(RDIV, 0)
,
35
return(COMMA, 0)
;
36
return(SEMIC, 0)

利用状态转换图识别单词

(1) 从初始状态出发；

(2) 读入一个字符；

(3) 按当前字符转入下一状态；

(4) 重复 (2)-(3) 直到无法继续转移为止。

在遇到读入的字符是单词的分界符时，若当前状态是终止状态，说明读入的字符组成了一个单词；否则，说明输入字符串w不符合词法规则。

利用状态转换图识别单词

(1) 从初始状态出发；

(2) 读入一个字符；

(3) 按当前字符转入下一状态；

(4) 重复 (2)-(3) 直到无法继续转移为止。

- 在遇到读入的字符是单词的分界符时，若当前状态是终止状态，说明读入的字符组成了一个单词；否则，说明输入字符串w不符合词法规则。
- 如果从状态转换图的初始状态出发，分别沿着所有可能的路径到达终止状态，并将每条路径上的标记依次连接成字符串，则可以得到状态转换图能够识别的所有单词，这些单词组成的集合也就是状态转换图识别的语言。
- 读入字符a时从状态A转换到状态B正好对应着一步推导过程，即$A \Rightarrow aB$，边正好与产生式$(A \rightarrow aB)$相对应

由正则文法构造状态转换图

(1) 以每个语法变量(或其编号)为状态结点，开始符号对应初始状态S；

(2) 增设一个终止状态 T；

(3) 对G中每个形如$A \rightarrow aB$的产生式，从状态A到状态B画一条有向弧，并标记为a；

(4) 对G中每个形如$A \rightarrow a$的产生式，从状态A到终止状态T画一条标记为a的有向弧；

(5) 对G中每个形如$A \rightarrow \varepsilon$的产生式，从状态$A$到终止状态$T$画一条标记为any的有向弧，any表示$T$中的任何符号。

3.3.3 几种典型的单词识别问题

- 标识符的识别
- 关键字的识别
- 常数的识别
- 算符和分界符的识别
- 回退

3.3.4 状态转换图的实现

- 如果将状态转换图看成是单词的识别规则库的话，则单词识别程序从当前状态(最初为初始状态)出发，读入一个输入字符后，将首先查询该规则库。
- 重复以下过程，直至到达某个终止状态。
  - 如果从当前状态出发有一条边上标记了刚刚读入的输入字符，则单词识别程序将转入这条边所指向的那个状态，并再读入一个输入字符；
  - 否则调用出错处理程序；
  - 将从初始状态到该终止状态所经历的路径上的字符所组成的字符串作为一个单词输出；
  - 并将当前状态重新置为开始状态，以便进行下一个单词的识别；
  - 如果读完输入字符流后仍未进入某个终止状态则调用出错处理程序。

3.3.5 词法分析程序的编写

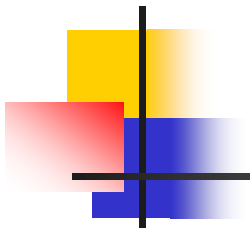
- 状态转移图——教材P93图3.15
- 状态转移图的实现——教材P105图3.22
- 词法分析程序*token_scan*( )
  - 输入：字符流
  - 输出：
    - *symbol*:单词种别
    - *attr*:属性（全局变量）

数据结构与子例程

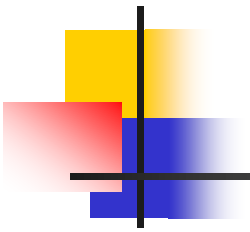
- 数据结构
  - *ch*　　字符变量，存放当前读入的输入字符
  - *token*　字符串变量，存放构成单词的字符串
  - *symbol*　单词种别（词法分析子程序的返回值）
  - *attr*　　属性（全局变量）
- 子例程
  - *install_id*(*token*):将*token*存入符号表，返回入口指针
  - *getchar*():从输入缓冲区中读入一个字符放入*ch*
  - *retract*():将向前指针回退一个字符，将*ch*置为空白符
  - *copytoken*():返回输入缓冲区中从开始指针*lexeme_beginning*到向前指针*forward*之间的字符串
  - *isLetter*()　*isalpha*()　*isalnum*()

图3.15的状态转换图的实现算法

```
token token_scan()
{ char ch;
char* token;
ch = getchar();
while (ch = =blank || ch = = tab || ch = = newline) {
ch = getchar();
    lexeme_beginning++;
}
if (isalpha(ch)) {ch = getchar();
    while (isalnum(ch))
ch = getchar();
retract(1);
token = copytoken();
return(gettoken(token), install_id(token));}
```



```
else
if (isdigit(ch)) {
    ch = getchar();
    while (isdigit(ch))
       ch = getchar();
retract(1);
token = copytoken();
return(INT, install_num(token));
}
else
switch(ch)
{
    case '*': ch = getchar();
              if(ch = = '*') return(EXP, 0);
              else {
                  retract(1);
                  return(MULTI, 0);
              } break;
```



```
case ':': ch = getchar();
          if(ch = = '=') return(ASSIGN, 0);
          else { retract(1); return(COLON, 0);
          } break;
case '<': ch = getchar();
          if(ch = = '=') return(LE, 0);
          else if(ch = = '>') return(NE, 0);
          else { retract(1);   return(LT, 0);
          } break;
case '=': return(EQ, 0); break;
case '>': ch = getchar();
          if(ch = = '=') return(GE, 0);
          else { retract(1); return(GT, 0);
          } break;
case '+': return(PLUS, 0); break;
case '-': return(MINUS, 0); break;
case '/': return(RDIV, 0); break;
case ',': return(COMMA, 0); break;
case ';': return(SEMIC, 0); break;
default: error_handle(); break;}return;}
```

需要说明的问题

- 缓冲区预处理，超前搜索
- 关键字的处理，符号表的实现
- Install_id()：查找效率，算法的优化实现
- 词法错误处理
- 由于高级语言的词组成的集合为3型语言，所以，这里讨论的词法分析技术可以用于处理所有的3型语言，也就是所有的可以用3型文法描述的语言。如：信息检索系统的查询语言、命令语言等

3.4 词法分析程序的自动生成

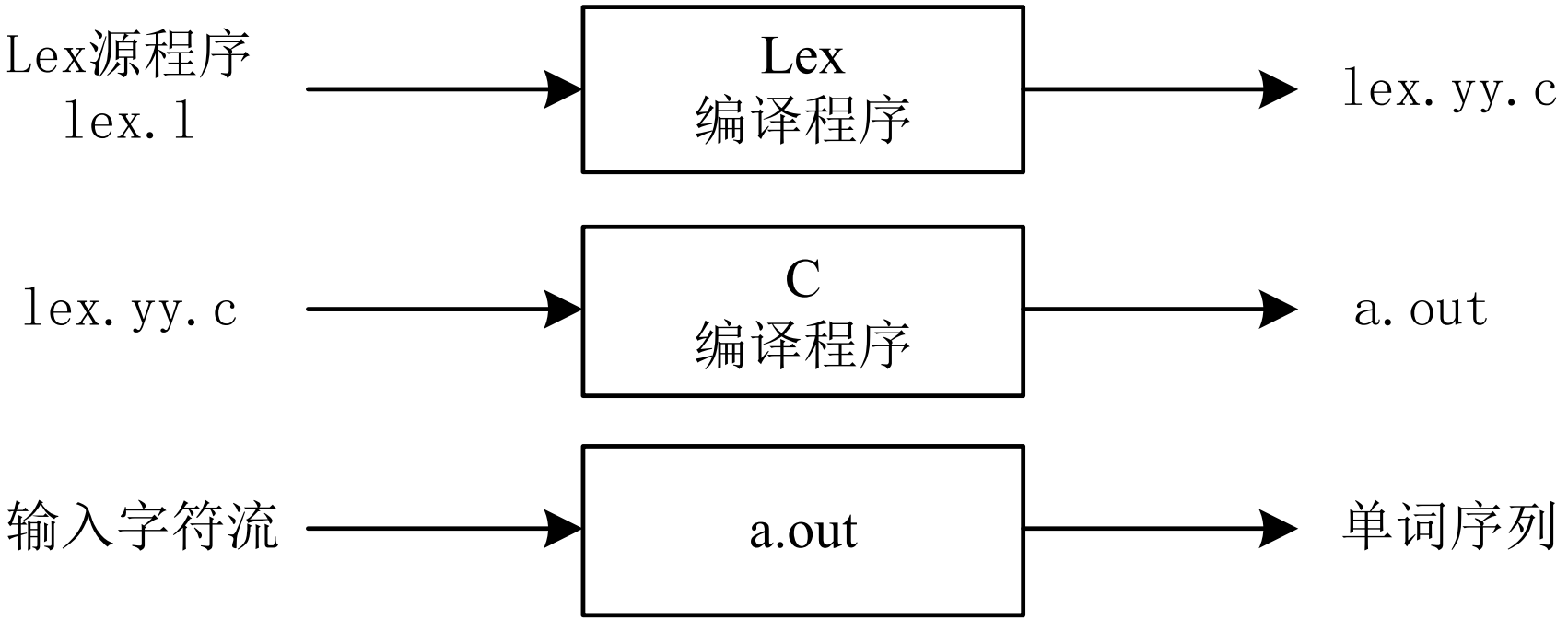
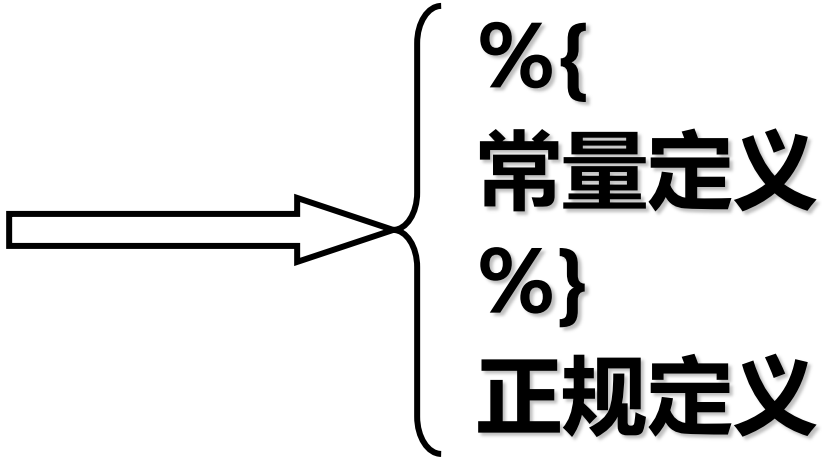


图3.23　利用Lex建立词法分析程序的过程

3.4.1 Lex源程序



3.4.1 Lex源程序

1、正规定义式

letter→A|B|C|…|Z|a|b|c|…|z

digit→0|1|2|…|9

identifier→letter(letter|digit)*

integer→digit(digit)*

2、识别规则

正规式	动作描述
$token_1$	$\{action_1\}$
$token_2$	$\{action_2\}$
……	
$token_n$	$\{action_n\}$

```
%{
#include <stdio.h>
#include "y.tab.h"
#define ID          1
#define INT         2
#define EXP         3
#define MULTI       4
#define COLON       5
#define EQ          6
#define NE          7
#define LE          8
#define GE          9
#define LT          10
#define GT          11
#define PLUS        12
#define MINUS       13
#define RDIV        14
#define COMMA       15
#define SEMIC       16
#define RELOP       17
#define ASSGIN      18
int line_no = 1;    %}
```

```
delim       [\t\n]
ws          [delim]+
letter      [a-zA-Z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+
%%
{ws}        ;
begin       return(BEGIN);
end         return(END);
if          return(IF);
then        return(THEN);
else        return(ELSE);
do          return(DO);
program     return(PROGRAM);
{id}        {yyval = install_id(); return(ID);}
{number}    {yyval = install_num();
             return(INT);}
```

- "<" {yyval =LT; return(RELOP);}
- "<=“ {yyval =LE; return(RELOP);}
- "=" {yyval =EQ; return(RELOP);}
- ">" {yyval =GT; return(RELOP);}
- ">=“ {yyval =GE; return(RELOP);}
- "<>“ {yyval =NE; return(RELOP);}
- "+" return(PLUS);
- "-" return(MINUS);
- "*" return(MULTI);
- "/" return(RDIV);
- "**" return(EXP);
- ":" return(COLON);
- ":=" return(ASSGIN);
- "," return(COMMA);
- ";" return(SEMIC);
- \n line_no++;
- . { fprintf (stderr,"'%c' (0%o): illegal charcter at line

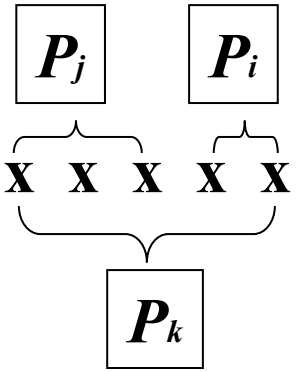
  %d\n", yytext[0], yytext[0], line_no); }
- %%
- install_id()
- {……}
- install_num()
- {……}

如:begin:=

LEX二义性问题的两条原则

1.最长匹配原则

在识别单词过程中，有一字符串
根据最长匹配原则，应识别为这是
一个符合P_k规则的单词，
而不是P_j和P_i规则的单词。



2.优先匹配原则

如果有一字符串有两条规则可以同时匹配时，那么用规则序列中位于前面的规则相匹配，所以排列在最前面的规则优先权最高。

3.4.2 Lex的实现原理

Lex的功能是根据Lex源程序构造一个词法分析程序，该词法分析器实质上是一个有穷自动机。

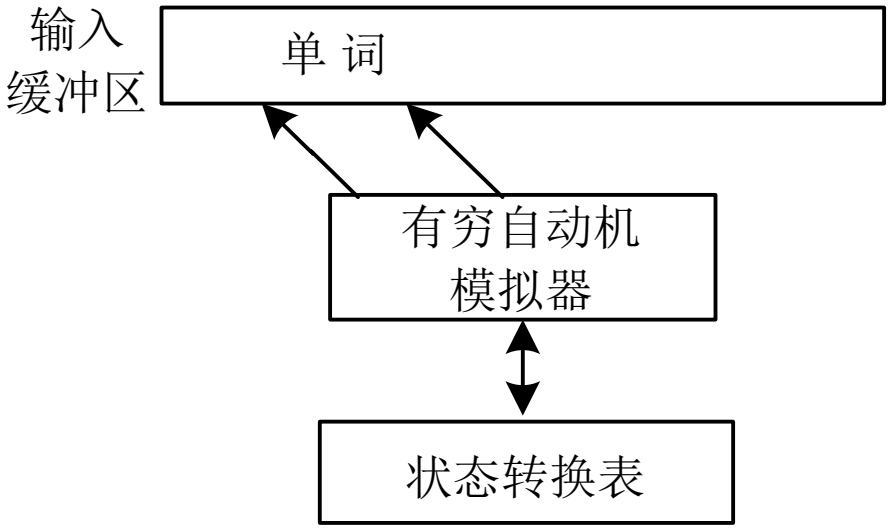


图 3.24　Lex生成的词法分析器结构

Lex的功能是根据Lex源程序生成状态转换矩阵和控制程序

•*　　*　　•*

三点说明

1) 以上是Lex的构造原理，虽然是原理性的，
但据此就不难将Lex构造出来。

2) 所构造出来的Lex是一个通用的工具，
用它可以生成各种语言的词法分析程序，
只需要根据不同的语言书写不同的LEX源文件
就可以了。

3) Lex不但能自动生成词法分析器，
而且也可以产生多种模式识别器及文本编辑程序等

本章小结

- 词法分析器接收表示源程序的“平滑字符流”，输出与之等价的单词序列；
- 单词被分成多个种类，并被表示成(种别，属性值)的二元组形式；
- 为了提高效率，词法分析器使用缓冲技术，而且在将字符流读入缓冲区时，是经过剔除注解、无用空白符等预处理后的结果；

本章小结

- 单词的识别相当于正则语言的识别；
- 词法的等价描述形式有正则文法、有穷状态自动机、正则表达式，其中有穷状态自动机可以用状态转换图表示；
- 实现词法分析器时状态转换图是一个很好的设计工具，根据该图，容易构造出相应的分析程序；
- 使用恰当的形式化描述，可以实现词法分析器的自动生成，Lex就是一种自动生成工具。