

编译原理

第三章 词法分析

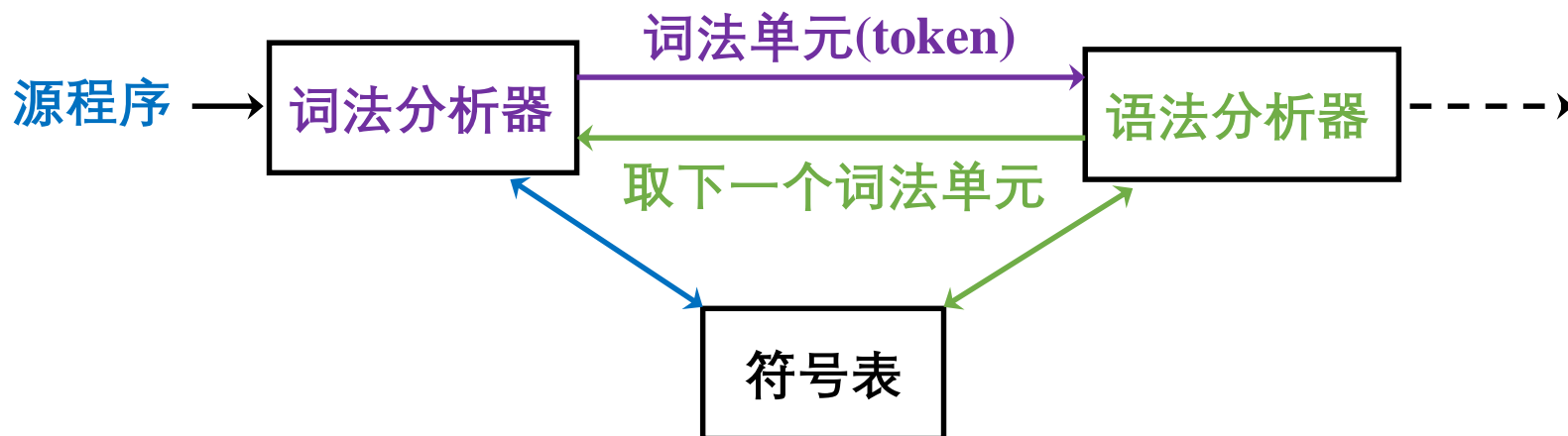
方徽星

扬州大学 信息工程学院(505)

fanghuixing@yzu.edu.cn

2018年春季学期

词法分析器与语法分析器的关系



本章主要内容

一. 正规文法与有限自动机

- 正规文法、正规式、正规集
- 确定的有限自动机 (DFA)
- 非确定的有限自动机 (NFA)
- NFA的确定化
- DFA的简化
- 正规式与有限自动机之间的关系
- 正规文法与有限自动机之间的关系

二. 词法分析程序的设计

- 预处理与超前搜索
- 扫描器的输出格式
- 扫描器的设计

词法分析基本术语

- 词法单元

- 词法单元名：抽象符号用于区别不同种类的词法单元，用于当作语法分析器的输入符号
- 属性值（可选）：指针或常量值（由具体实现决定）

- 例：position = initial + rate * 60的词法单元名和属性值：

⟨id, 指向符号表中position条目的指针⟩

⟨assign_op⟩

⟨id, 指向符号表中initial条目的指针⟩

⟨add_op⟩

⟨id, 指向符号表中rate条目的指针⟩

⟨mul_op⟩

⟨number, 整数值60⟩

词法分析基本术语

- 模式：描述一个词法单元的词素可能具有的形式
- 词素：源代码中的实际字符序列，可与某词法单元模式相匹配，可被词法分析器识别为一个词法单元实例
- 例：词素示例

词素	词法单元名
if, else	Keyword
<=, !=	CompOp
index, score	ID
3.14159	FloatNumber

第一节 正规文法与有限自动机

1.1 正规文法、正规式、正规集

- 由正规文法产生的语言称为正规集,可以使用正规式来形式地表示正规集, 设非空有限字母表 $A = \{a_i \mid i = 1, 2, \dots, n\}$, 则
 - 空串符号 ε 是正规式, $L(\varepsilon) = \{\varepsilon\}$
 - 空集符号 \emptyset 是正规式, $L(\varepsilon) = \{\}$
 - $a \in A$ 是正规式, $L(a) = \{a\}$
 - 如果 α, β 是正规式, 则下面都是正规式:
 - $\alpha \mid \beta$, $L(\alpha \mid \beta) = L(\alpha) \cup L(\beta)$ (或)
 - $\alpha \cdot \beta$, $L(\alpha \cdot \beta) = L(\alpha)L(\beta)$ (连接)
 - α^* 和 β^* , $L(\alpha^*) = (L(\alpha))^*$ (闭包)
- 正规式扩展: α^+ , $L(\alpha^+) = (L(\alpha))^+$; $\alpha?$, $L(\alpha?) = L(\alpha) \cup \{\varepsilon\}$; $[a_1 a_2 \cdots a_m] = a_1 \mid a_2 \mid \cdots \mid a_m$, $[a-z] = a \mid b \mid \cdots \mid z$

1.1 正规文法、正规式、正规集

- 正规式代数定律：如果 α , β , γ 是正规式，则下述等式成立

定律	描述
$\alpha \mid \beta = \beta \mid \alpha$	交换律
$\alpha \mid (\beta \mid \gamma) = (\alpha \mid \beta) \mid \gamma$ $\alpha(\beta\gamma) = (\alpha\beta)\gamma$	结合律
$\alpha(\beta \mid \gamma) = \alpha\beta \mid \alpha\gamma$ $(\alpha \mid \beta)\gamma = \alpha\gamma \mid \beta\gamma$	分配律
$\varepsilon\alpha = \alpha\varepsilon = \alpha$	ε 是连接的单位元
$(\alpha^*)^* = \alpha^*$	$*$ 具有幂等性
$\alpha^* = \alpha^+ \mid \varepsilon$	闭包中一定包含 ε

1.1 正规文法、正规式、正规集

- 定理1: 设 α, β, γ 是字母表A上的正规式, 则
 - $\alpha = \beta \mid \alpha\gamma$ 当且仅当 $\alpha = \beta\gamma^*$
 - $\alpha = \beta \mid \gamma\alpha$ 当且仅当 $\alpha = \gamma^*\beta$
- 由正规文法求解对应语言的正规式:
 - 首先由各个产生式写出对应的正规方程式, 获得联立方程组
 - 这些方程组中的变元是非终结符, 求解方程组得到一个关于开始符号的解

1.1 正规文法、正规式、正规集

- 例：已知正规文法的产生式如下：

$$S \rightarrow aS \mid aB, \quad B \rightarrow bB \mid bA, \quad A \rightarrow cA \mid c$$

求出它所定义的正规式

解：由产生式写出联立方程组：

$$S = aS \mid aB \quad (1)$$

$$B = bB \mid bA \quad (2)$$

$$A = cA \mid c \quad (3)$$

由定理1，(1)的解为 $S = a^*aB = a^+B$ ；

同理(2)的解为 $B = b^*bA = b^+A$ ；

由(3)可得 $A = c^*c = c^+$ ；

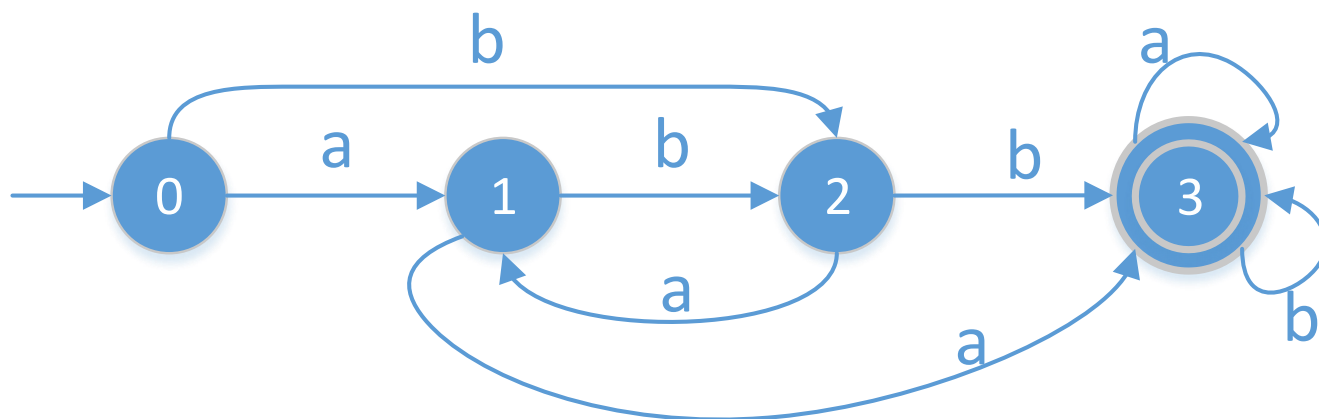
最后解得正规式 $S = a^+b^+c^+$

1.2 确定的有限自动机

- 确定的有限自动机 (DFA: Deterministic Finite Automata) $M=(S, \Sigma, f, s_0, Z)$, 其中:
 - S : 有限状态集, S 的每一个元素 s 称为状态
 - Σ : 有穷字母表, Σ 的每一个元素 σ 称为输入字符
 - $f: S \times \Sigma \rightarrow S$, 转换函数, 对于状态 $s \in S$ 和输入符号 $\sigma \in \Sigma$ **最多只有一条**标号为 σ 的边离开状态 s
 - $s_0 \in S$ 是初始状态
 - $Z \subseteq S$ 是终止状态集, Z 为空集时, 该DFA不接受任何东西

1.2 确定的有限自动机

- 例：DFA $M = (\{0,1,2,3\}, \{a, b\}, f, 0, \{3\})$, 其中
 $S = \{0,1,2,3\}, \Sigma = \{a, b\}, s_0 = 0, Z = \{3\}$
转换图如下所示：



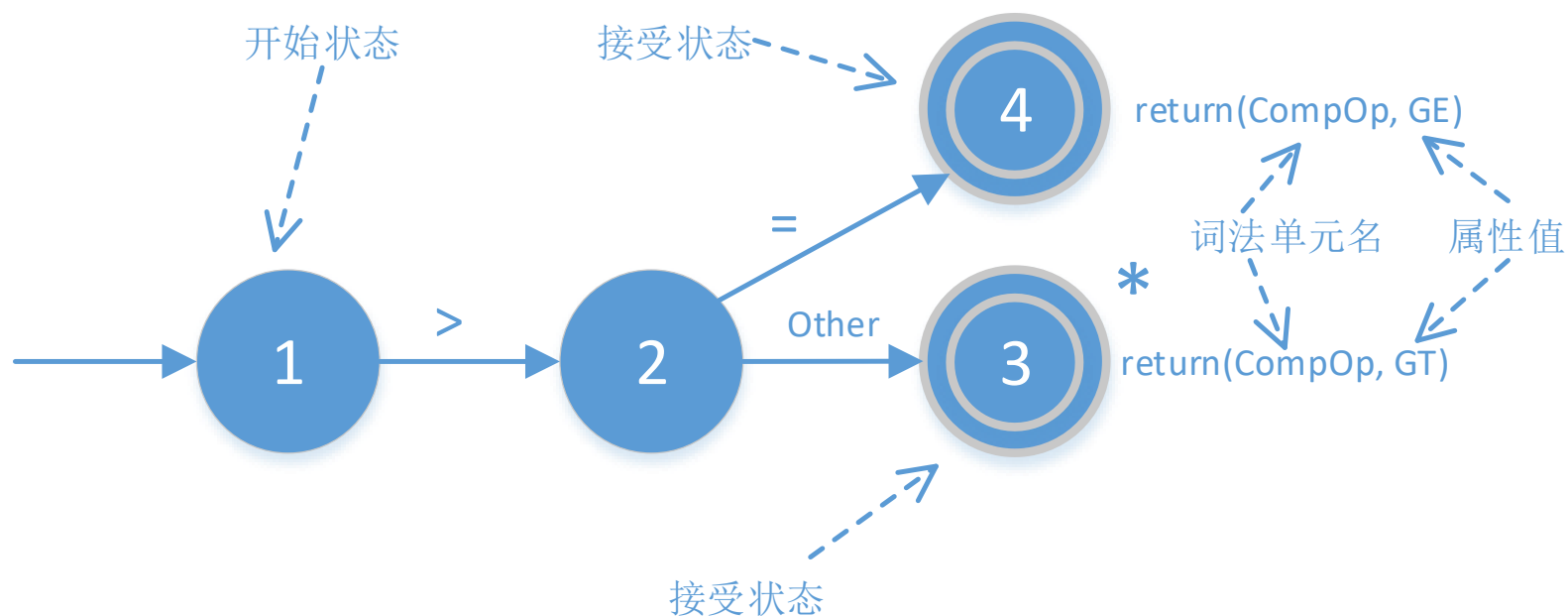
此例中转换函数 f 是如何定义的？

1.2 确定的有限自动机

- **状态转换图**用于表示词法分析器被语法分析器调用时，词法分析器为返回下一个词法单元所做的动作
 - 开始识别词法单元时，控制**进入开始状态**
 - 如果**离开状态**的某条边上的**标记与当前输入字符匹配**，则**进入目标状态**，否则识别过程失败
 - 由**双圈**表示的状态规定为**接受状态**，控制进入该类状态则表示**成功识别**了一个词法单元
 - 接受状态**可以有动作**，到达接受状态时执行该动作
 - 符号*：表示**需要回退**一个字符

1.2 确定的有限自动机

状态转换图示例



1.2 确定的有限自动机

- DFA接受输入串 x ，当且仅当转换图中存在从开始状态到某个终止状态的路径，使得该路径上各边上的标记组成 x

DFA模拟算法

输入：一个以文件文件结束符EOF结尾的字符串 x . DFA M 的开始状态为 s_0 ，终止状态集为 Z ，转换函数为 f

输出：如果 M 接受 x ，则回答“yes”，否则回答“no”

```
 $s = s_0;$   
 $c = nextChar();$   
While( $c \neq EOF$ ){  
     $s = f(s, c);$   
     $c = nextChar();$   
}  
  
If ( $s \in Z$ )  
    return “yes”;  
Else  
    return “no”;
```

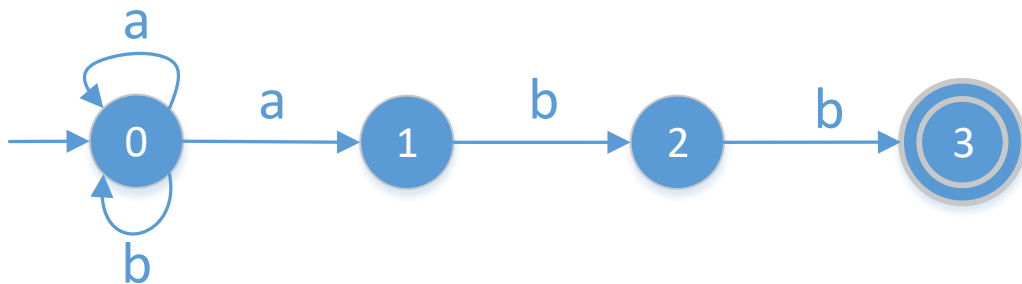
1.3 非确定的有限自动机

- 非确定的有限自动机 (NFA: Nondeterministic Finite Automata) $M=(S, \Sigma, f, s_0, Z)$, 其中:
 - S : 有限状态集, S 的每一个元素 s 称为状态
 - Σ : 有穷字母表, Σ 的每一个元素 σ 称为输入字符
 - $f: S \times \Sigma \cup \{\varepsilon\} \rightarrow 2^S$, 转换函数, 对于每个状态 $s \in S$ 和每个输入符号 $\sigma \in \Sigma$ 可能有多条标号为 σ 的边离开状态 s
 - $s_0 \in S$ 是初始状态
 - $Z \subseteq S$ 是终止状态集, Z 可为空集
- 确定与非确定的有限自动机能识别的语言的集合是相同的, 且是能够用正规式描述的语言集合

1.3 非确定的有限自动机

- 例：NFA $M = (\{0,1,2,3\}, \{a, b\}, f, 0, \{3\})$, 其中
 $S = \{0,1,2,3\}, \Sigma = \{a, b\}, s_0 = \{0\}, Z = \{3\}$

转换图如下所示：



此例中转换函数 f 是如何定义的？

1.4 NFA的确定化

- 将NFA转换为DFA
- 子集构造算法(Subset Construction):
 - 输入：一个NFA $N = (S_N, \Sigma_N, f_N, s_0^N, Z_N)$
 - 输出：一个DFA $D = (S_D, \Sigma_D, f_D, s_0^D, Z_D)$
 - 基本方法：构造 D 的状态和转换函数， D 的每个状态对应 N 的一个状态集合， N 中每个状态集合的转换对应 D 中的一个状态转换

1.4 NFA的确定化

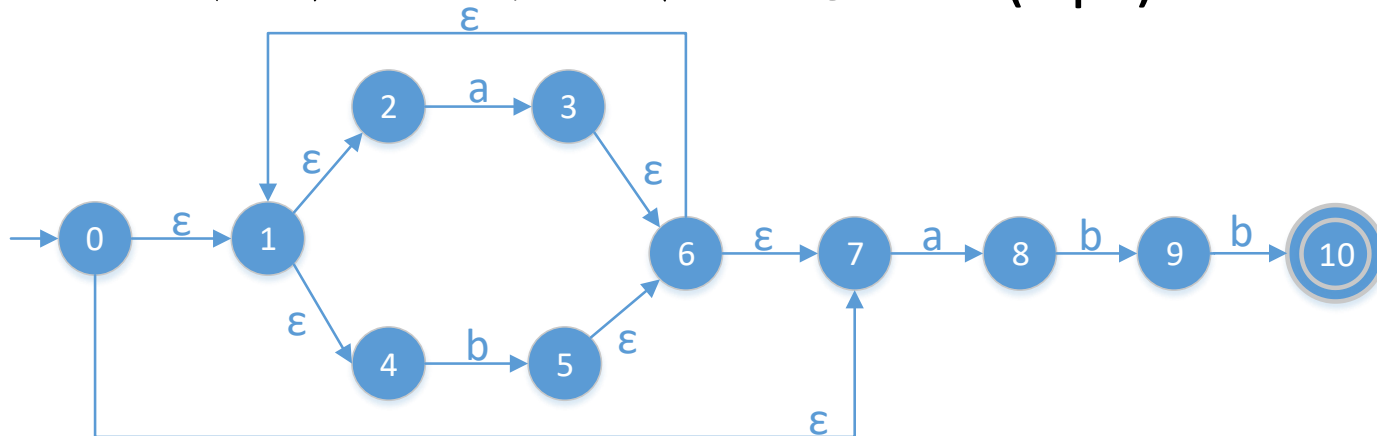
子集构造算法主体:

```
一开始,  $\varepsilon\text{-closure}(s_0^N)$  是  $S_D$  中的唯一状态, 且未加标记;  
While (在  $S_D$  中有一个未标记的状态  $T$ ) {  
    给  $T$  加上标记;  
    For (每个输入符号  $a$ ) {  
         $U = \varepsilon\text{-closure}(\text{move}(T, a));$   
        If ( $U$  不在  $S_D$  中)  
            Then 将  $U$  加入到  $S_D$  中, 且不加标记;  
         $f_D(T, a) = U;$   
    }  
}
```

操作	描述
$\varepsilon\text{-closure}(s)$	能够从NFA的状态 s 出发只通过 ε 转换到达的NFA状态集合
$\varepsilon\text{-closure}(T)$	能够从 T 中某个NFA的状态 s 出发只通过 ε 转换到达的NFA状态集合
$\text{move}(T, a)$	能够从 T 中某个NFA的状态 s 出发只通过 a 转换到达的NFA状态集合

1.4 NFA的确定化

- 例：下图给出了一个接受语言 $(a|b)^*abb$ 的NFA



构造DFA：DFA的开始状态A为 ϵ -closure(0)={0, 1, 2, 4, 7}

NFA状态集	DFA状态	符号a	符号b
{0,1,2,4,7}	A	move(A,a)={3,8} B= ϵ -closure({3,8}) ={1,2,3,4,6,7,8}	move(A,b)={5} C= ϵ -closure({5}) ={1,2,4,5,6,7}
{1,2,3,4,6,7,8}	B	?	?

1.5 DFA的简化

- 任何一个正则语言都有一个唯一的状态数目最少的DFA
- 从任意一个接受相同语言的DFA出发，通过分组合并等价状态，总可以构建得到该状态数最少的DFA
- 如果分别从状态 s 和 t 出发，沿着串 x 的路径到达的两个状态中只有一个是接受状态，则串 x 区分状态 s 和 t
- 如果存在某个能够区分状态 s 和 t 的串，则 s 和 t 是可区分的

1.5 DFA的简化

- 最小化DFA状态数量算法

输入：DFA $D = (S, \Sigma, f, s_0, Z)$

输出：最小化DFA D'

1. 首先构造包含两个组 Z 和 $S \setminus Z$ 的初始分划 P ，分别是 D 的接受状态组和非接受状态组；

2. 构造新分划 P_n ；

3. 若 $P_n = P$ ，令 $P_f = P$ 跳到步骤4；否则，用 P_n 替换 P 并重复步骤2；

4. 在分划 P_f 的每个组中选取一个代表，则所有代表构成 D' 的状态：

- D' 的开始状态是 D 的开始状态所在组的代表， D' 的接受状态是那些包含了 D 的接受状态所的组的代表；
- 令 s 为某组代表，且 $s \xrightarrow{a} t$ ，若 r 为 t 组的代表，则 D' 中有 $s \xrightarrow{a} r$

1.5 DFA的简化

- 最小化DFA状态数量算法

令 $P_n = P$;

For (P中的每个组G) {

 将G划分为更小的组，使得两个状态s和t在同一个组中 当且仅当对于所有的输入符号a，状态s和t在a上转换到P中的同一组里；

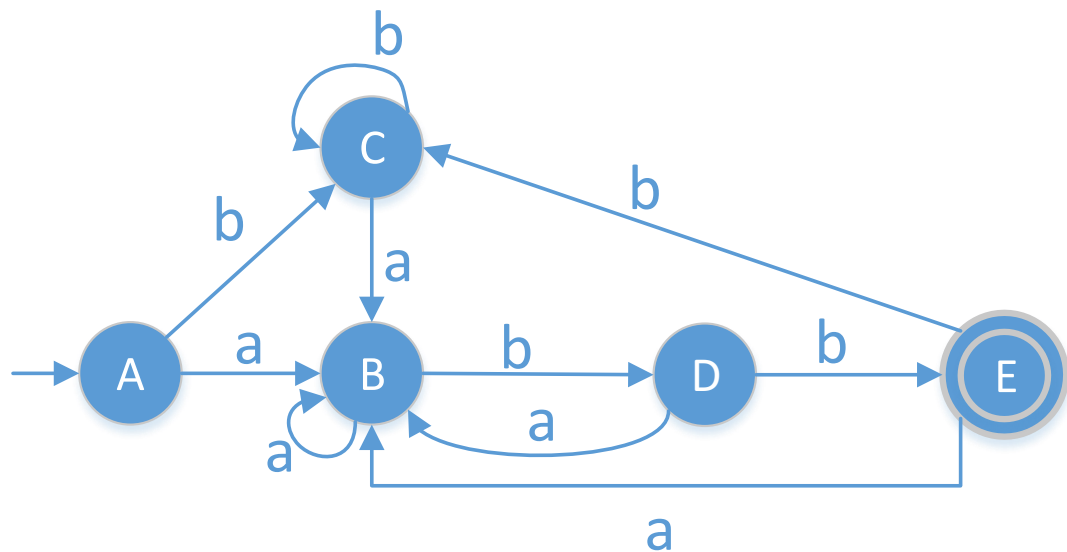
 在 P_n 中将G替换为对G进行划分得到的那些小组；

}

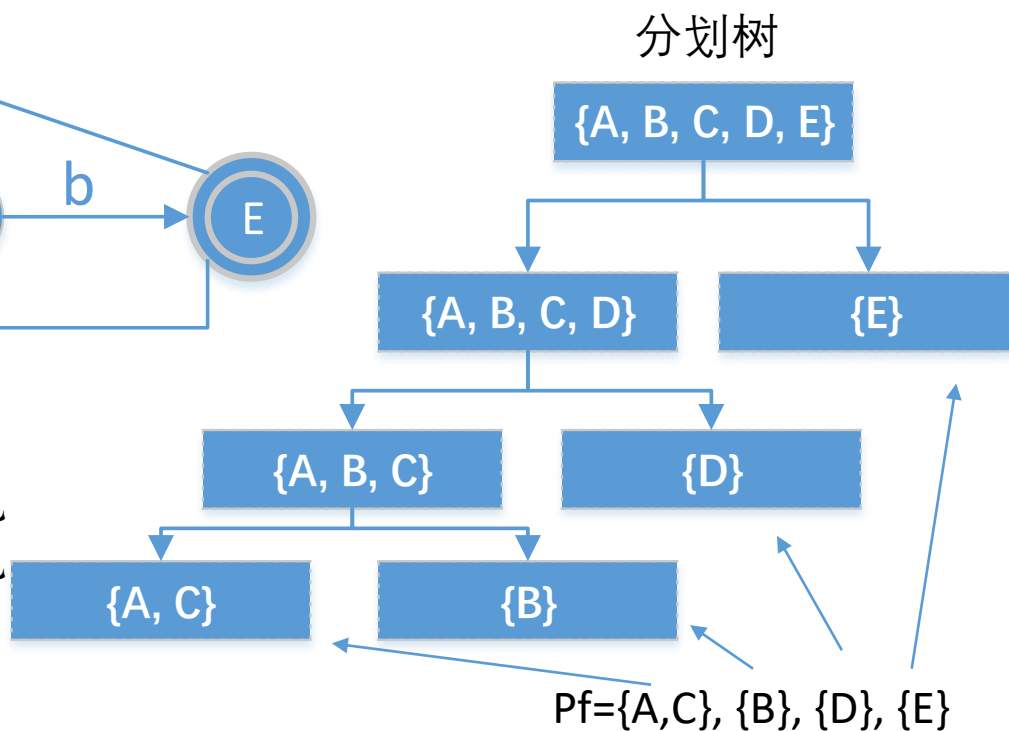
构造新分划 P_n

1.5 DFA的简化

- 例：将下图所示DFA最小化

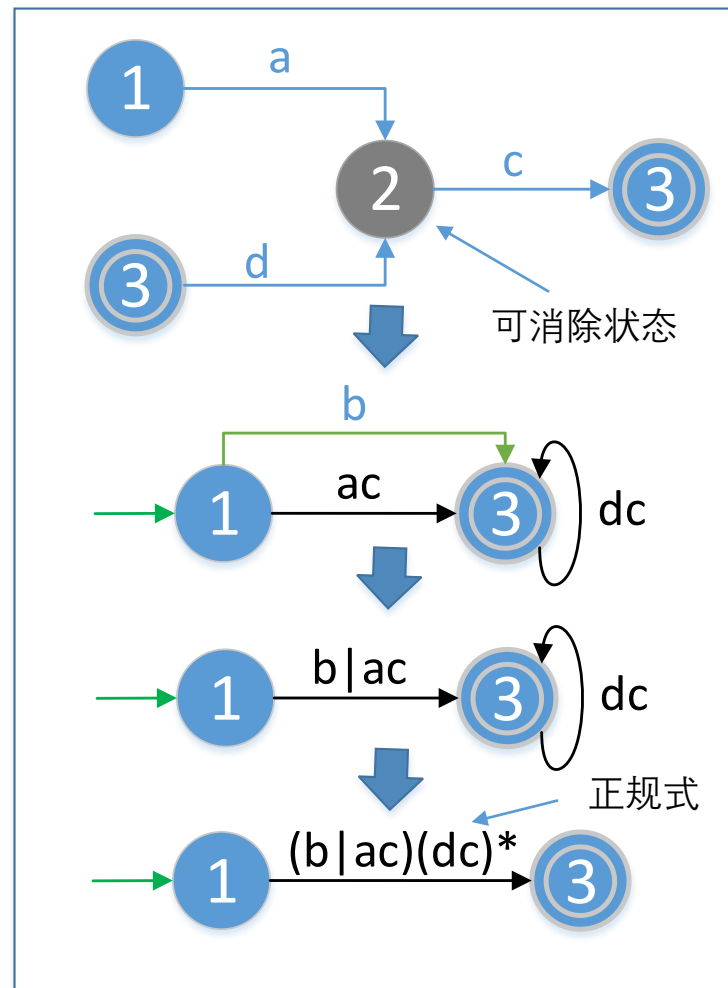
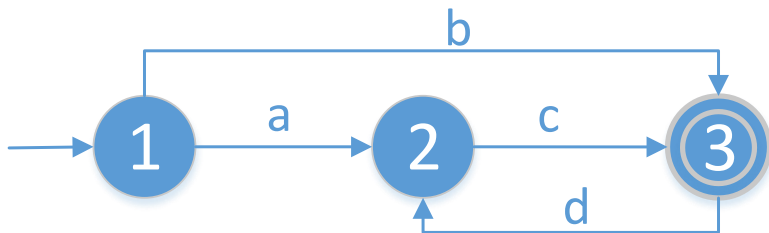


- 第一次分划：非接受和接受
- 第二次分划：D在b上不同于其他
- 第三次分划：B在b上不同于其他



1.6 正规式与有限自动机之间的关系

- 定理2: 字母表 Σ 上的NFA M 所能识别的语言 $L(M)$ 可以用 Σ 上的正规式表示
- 定理3: 对于 Σ 上的任何正规式 α , 存在一个DFA M 使得 $L(M)=L(\alpha)$
- 例: 将下图所示的有限自动机使用正规式表示



1.6 正规式与有限自动机之间的关系

- 由正规式构造NFA的McMaughton-Yamada-Thompson
算法：沿着正规式的语法分析树自底向上递归地处理，对于每个子表达式，构造一个只有一个接受状态地NFA
 - **输入：**字母表 Σ 上的一个正规式 γ
 - **输出：**一个接受 $L(\gamma)$ 的NFA N
 - **方法：**首先对 γ 进行语法分析，分解出组成它的子表达式，然后使用构造规则

构造
规则

基本规则：处理不包含运算符的子表达式

归纳规则：根据一个给定表达式的直接子表达式的NFA构造出这个表达式的NFA

1.6 正规式与有限自动机之间的关系

McMaughton-Yamada-Thompson 算法

- 基本规则

- 对于表达式 ε ，构造下面的NFA



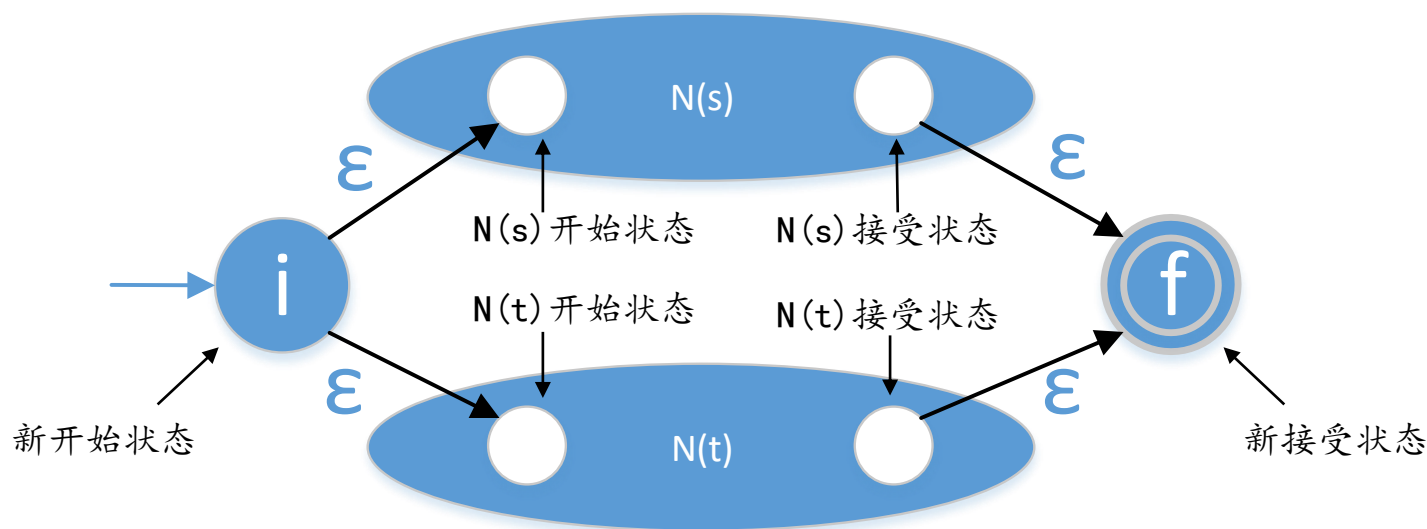
- 对于字母表中的每个符号 a ，构造下面的NFA



1.6 正规式与有限自动机之间的关系

McMaughton-Yamada-Thompson算法

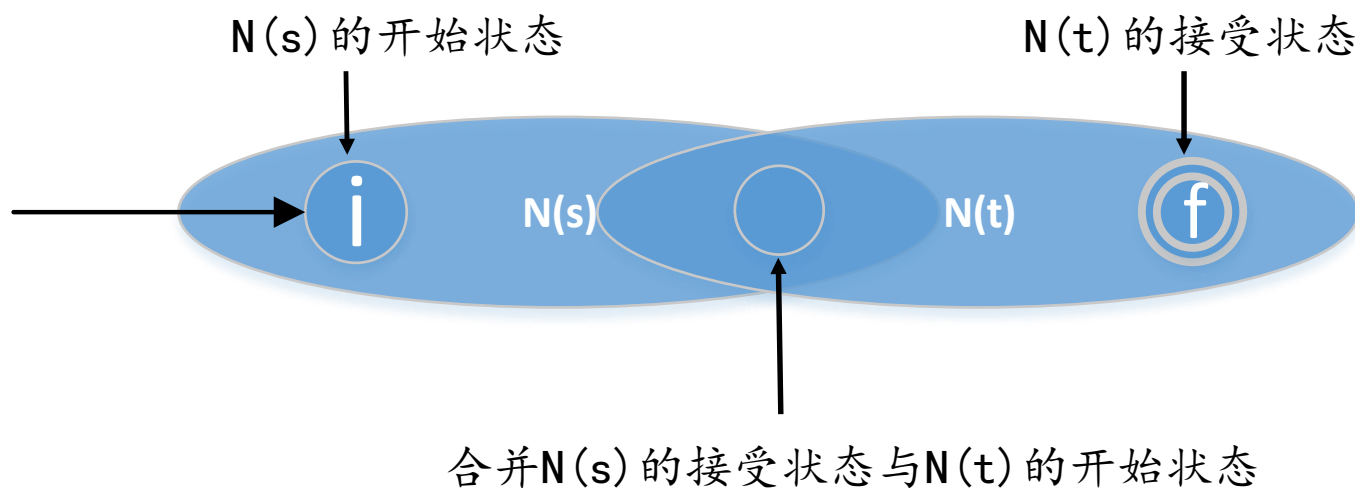
- 归纳规则：假设正规式 s 和 t 的NFA分别是 $N(s)$ 和 $N(t)$ ，则
 - 如果 $\gamma = s \mid t$ ，则 γ 的NFA，可以按照下图方式构造



1.6 正规式与有限自动机之间的关系

McMaughton-Yamada-Thompson算法

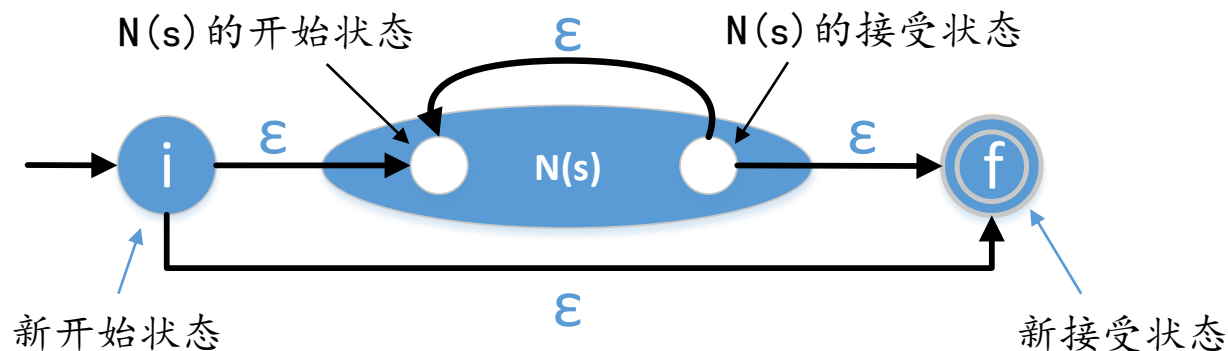
- 归纳规则：假设正规式 s 和 t 的NFA分别是 $N(s)$ 和 $N(t)$ ，则
 - 如果 $\gamma = st$ ，则 γ 的NFA，可以按照下图方式构造



1.6 正规式与有限自动机之间的关系

McMaughton-Yamada-Thompson算法

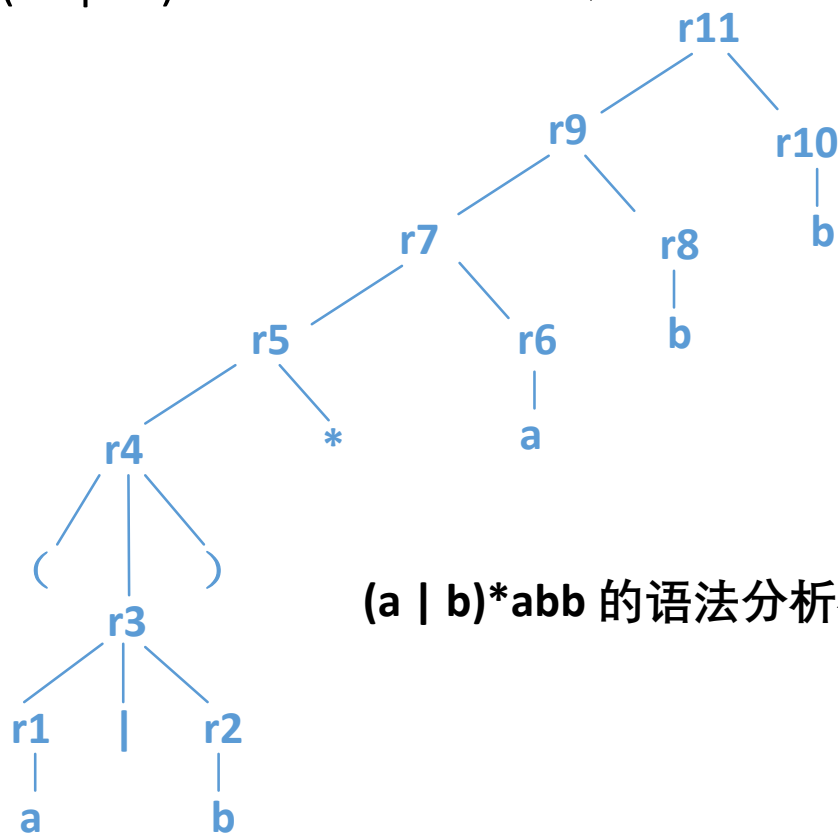
- 归纳规则：假设正规式 s 和 t 的NFA分别是 $N(s)$ 和 $N(t)$ ，则
 - 如果 $\gamma = s^*$ ，则 γ 的NFA，可以按照下图方式构造



- 如果 $\gamma = (s)$ ，则 $L(\gamma) = L(s)$ ，直接把 $N(s)$ 当作 $N(\gamma)$

1.6 正规式与有限自动机之间的关系

- 例：使用McMaughton-Yamada-Thompson算法为正规式 $r=(a \mid b)^*abb$ 构造一个NFA



$(a \mid b)^*abb$ 的语法分析树

1.7 正规文法与有限自动机之间的关系

- 设 $G = (V_N, V_T, P, S)$ 是正规文法，则存在一个有限自动机 $M = (Q, \Sigma, f, q_0, Z)$ 使得 $L(G) = L(M)$

右线性文法的自动机构造规则：

- 令 M 中的 $Q = V_N \cup \{T\}$ ，其中 T 是新增的接受状态；
- 令 $\Sigma = V_T$ ， $q_0 = S$ ；
- 若 P 中含有产生式 $S \rightarrow \varepsilon$ ，则 $Z = \{S, T\}$ ，否则 $Z = \{T\}$
- 转换函数 f 构造如下
 - 对于 P 中每一条形如 $A_1 \rightarrow aA_2$ 的产生式，有 $f(A_1, a) = A_2$
 - 对于 P 中每一条形如 $A_1 \rightarrow a$ 的产生式，有 $f(A_1, a) = T$
 - 对于 Σ 上所有的 a 有 $f(T, a) = \emptyset$ ，在接受状态下自动机没有动作

1.7 正规文法与有限自动机之间的关系

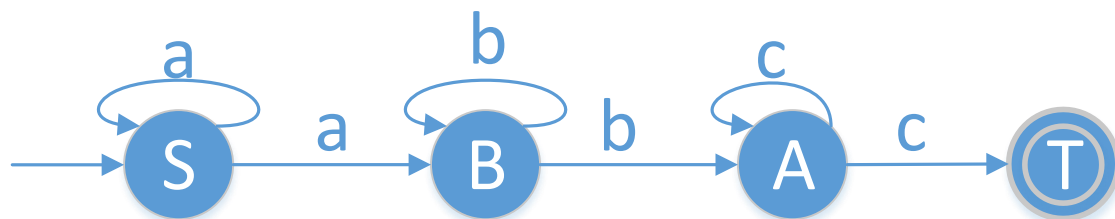
- 例：已知文法 $G = (\{S, A, B\}, \{a, b, c\}, P, S)$ ，其中 P 定义如下

1. $S \rightarrow aS$,
2. $S \rightarrow aB$,
3. $B \rightarrow bB$,
4. $B \rightarrow bA$,
5. $A \rightarrow cA$,
6. $A \rightarrow c$

试构造等价的自动机

解：按照构造规则，构造等价自动机为：

$$M = (\{S, A, B, T\}, \{a, b, c\}, f, S, \{T\})$$



其中 f 定义如下：

1. $f(S, a) = S$
2. $f(S, a) = B$
3. $f(B, b) = B$
4. $f(B, b) = A$
5. $f(A, c) = A$
6. $f(A, c) = T$

1.7 正规文法与有限自动机之间的关系

- $G = (V_N, V_T, P, S)$, $M = (Q, \Sigma, f, q_0, Z)$
- 左线性文法的自动机构造规则:
 - 令 M 中的 $Q = V_N \cup \{q_0\}$, 其中 q_0 是新增的初始状态;
 - 令 $\Sigma = V_T$;
 - 若 P 中含有产生式 $S \rightarrow \varepsilon$, 则 $Z = \{S, q_0\}$, 否则 $Z = \{S\}$
 - 转换函数 f 构造如下
 - 对于 P 中每一条形如 $A_1 \rightarrow A_2 a$ 的产生式, $f(A_2, a) = A_1$
 - 对于 P 中每一条形如 $A_1 \rightarrow a$ 的产生式, $f(q_0, a) = A_1$

1.7 正规文法与有限自动机之间的关系

- 例：构造左线性文法 $G = (\{S, A, B\}, \{a, b\}, P, S)$ 的相应的自动机，其中 P 定义如下：

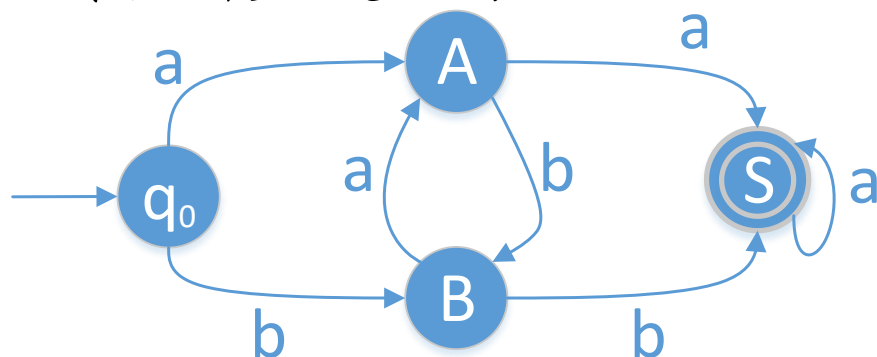
1. $S \rightarrow Sa \mid Aa \mid Bb$

2. $A \rightarrow Ba \mid a$

3. $B \rightarrow Ab \mid b$

解：按照构造规则， $M = (\{S, A, B, q_0\}, \{a, b\}, f, q_0, \{S\})$

转换函数 f 定义为：



$f(S, a) = S$

$f(A, a) = S$

$f(B, b) = S$

$f(B, a) = A$

$f(q_0, a) = A$

$f(A, b) = B$

$f(q_0, b) = B$

第二节 词法分析程序的设计

2.1 预处理与超前搜索

- **预处理**：包括对空白符、跳格符、回车符和换行符等编辑性字符的处理，及删除注解，识别标号区、找出续行符连接成完整语句等
- **超前搜索**：当读到一个单词后，不能确定该单词的作用，要向前多读几个字符后才能确定；在读到一个单词之后，在缓冲区或扫描缓冲区上做标记，然后继续先前读，直到明确之后再退回标记处重新分析
 - 如算符的识别 C/C++，java的++，--，>=，!=，==等

2.2 扫描器的输出格式

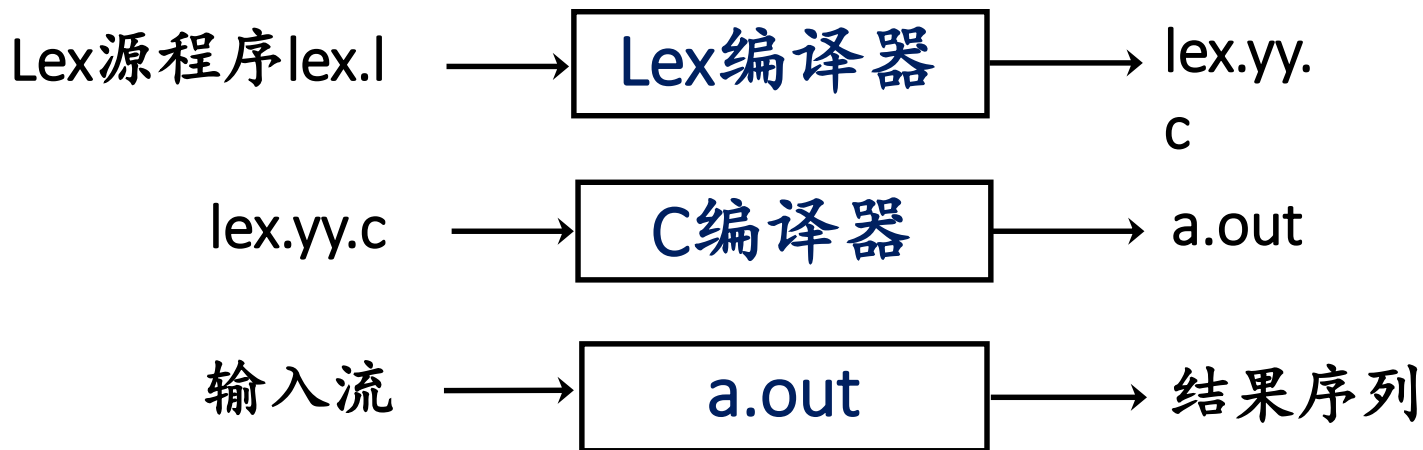
- 词法分析程序通常又叫作扫描器
- 单词分类（Pascal语言为例）
 - **关键字**：具有特殊含义的标识符，起分隔语法成分的作用，如：program、var、procedure、begin、end
 - **标识符**：用于表示各种名字，如：变量名、数组名、函数名、过程名
 - **运算符**
 - 算术运算符+、-、*、/
 - 逻辑运算符：and、or、not
 - 关系运算符：>、>=、<、<=、=
 - **分界符**：‘,’、‘;’、‘(’、‘)’、‘:’

2.2 扫描器的输出格式

- 扫描器的输出格式为二元式序列，每个单词对应一个二元式（类号，内码）
 - 类号用整数表示，区分单词的种类，方便程序处理
 - 每个关键字有一个类号，编译程序内部通常有关键字表，表中的编号可以用作类号，而内码可以省略
 - 对于标识符，内码用于区分标识符名，把符号表中地址作为内码
 - 常量，按不同类型分成相应类号，将常量表中的地址作为内码
 - 分界符，整型类号，内码可省略

2.3 扫描器的设计

- 用Lex建立词法分析器的步骤



- 词法分析器仅返回记号名（代表词法单元的抽象名字）给语法分析器
- 记号的属性值通过共享整型变量yylval传递

2.3 扫描器的设计

- Lex程序包括三个部分

声明

%%

翻译规则

%%

辅助过程

- Lex程序的翻译规则

模式	{动作}
----	------

p1	{动作1}
----	-------

...	...
-----	-----

pn	{动作n}
----	-------

- 每个模式是一个正规式
- 每个动作描述该模式匹配词法单元时，词法分析器应执行的程序段

2.3 扫描器的设计

- 例—声明部分

```
%{
```

```
/* 常量LT, LE, EQ, NE, GT, GE, WHILE, DO, ID, NUMBER,  
   RELOP的定义,用C的#define方式来写*/
```

```
%}
```

```
/* 正规定义 */
```

```
delim      [ \t \n ]
```

```
ws          {delim}+
```

```
letter      [A-Za-z]
```

```
digit       [0-9]
```

```
id           {letter}({letter}|{digit})*
```

```
number       {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
```

2.3 扫描器的设计

- 例—翻译规则部分

{ws}	{/* 没有动作，也不返回 */}
while	{return (WHILE);}
do	{return (DO);}
{id}	{yylval = install_id (); return (ID);}
{number}	{yylval = install_num(); return (NUMBER);}
" < "	{yylval = LT; return (RELOP);}
" <= "	{yylval = LE; return (RELOP);}
" = "	{yylval = EQ; return (RELOP);}
" <> "	{yylval = NE; return (RELOP);}
" > "	{yylval = GT; return (RELOP);}
" >= "	{yylval = GE; return (RELOP);}

2.3 扫描器的设计

- 例—辅助过程部分

```
int installId ( ) {  
    /* 把词法单元装入符号表并返回指针 */  
}
```

```
int installNum ( ) {  
    /* 类似上面的过程，但词法单元不是  
       标识符而是数，并放入数表 */  
}
```

- Lex最初的作者：Mike Lesk, Eric Schmidt (前谷歌CEO)

2.3 扫描器的设计

```
/* recognize tokens for the calculator and
print them out */
%{
    enum yytokentype {
        NUMBER = 258,
        ADD = 259,
        SUB = 260,
        MUL = 261,
        DIV = 262,
        ABS = 263,
        EOL = 264
    };

    int yylval;
}%

%%
```

```
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"|" { return ABS; }
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
\n    { return EOL; }
[\t] { /* ignore whitespace */ }
. { printf("Mystery character %c\n", *yytext); }
%%

int main(int argc, char **argv)
{
    int tok;
    while(tok = yylex()) {
        printf("%d", tok);
        if(tok == NUMBER) printf(" = %d\n", yylval);
        else printf("\n");
    }
}
```

小结

- 正规文法与有限自动机
 - 主要是DFA、NFA及正规式、正规文法
- 词法分析程序的设计
 - 了解实现词法分析程序的技术