第三章 词法分析

3.1 有限自动机
3.2 词法分析器的设计与实现
3.3 词法分析器的自动生成

作业：P63-65:2,6-9,12-14,18*,20* （九月二十二交）
上机：(1) 编写识别C的数值型常数的扫描程序
(2) 用LEX自动产生一个指定语言的扫描程序
（上机时间由班级自行安排，定了以后通知辅导老师）

3.1 有限自动机

- 确定有限自动机
- 非确定有限自动机
- 正规文法与确定自动有限自动机的等价性
- 正规式与确定自动有限自动机的等价性
- 确定自动有限自动机的化简

3.1.1 正规式与正规集

正规式和正规集的形式定义

1. ε和Φ都是字母表Σ上的正规式，它们所表示的正规集分别为{ε}和Φ；
2. 任何a∈Σ，a是Σ上的一个正规式，它所表示的正规集为{a}；
3. 假定U和V都是Σ上的正规式,它们所表示的正规集分别记为L(U)和L(V),那么 (U|V)、(U·V) 和(U)* 也都是Σ上的正规式，而且它们所表示的正规集分别为 L(U)∪L(V)、L(U)·L(V)和L(U)*

- 通过有限次使用上述三步骤所得到的表达式是Σ上的正规式。正规式由这些正规式所表示的符号串的集合是Σ上的正规集。

例：Σ={a,b}，那么ba*, a(a|b)*和(a|b)*(aa|bb)(a|b)*都是Σ上的正规式.

a是(2)=>a*(3闭包)是，b是(2)=>ba*是(3连接)
正规集：Σ上所有以b为首后跟0到多个a的符号串.

a是(2)，b是(2)=>a|b是(3或)=>(a|b)*(3闭包)是；a是(2)=> a(a|b)*是(3连接)
正规集：Σ上所有以a为首的符号串.

a是(2)，b是(2)=>a|b是(3或)=>(a|b)*(3闭包)是；a是(2)=> aa是(3连接)；b是(2)=> bb是(3连接)=> (a|b)*(aa|bb)(a|b)*是(3连接)
正规集：Σ上所有含有两个相继a或两个相继b的符号串.

正规式和正规集举例

令字母表Σ={a,b}，则：

正规式a|b表示集合{a, b}

正规式(a|b)(a|b)表示集合{aa, ab, ba, bb}

正规式a*表示由0到多个a构成的符号串的集合

正规式(a|b)*表示由0到多个a或b构成的符号串的集合

正规式a|a*b表示符号串a以及由0到多个a后跟一个b构成的符号串的集合

正规式与正规集

正规式：形式语言中确切定义一组合法的单词符号的字的规则

构造这种规则采用如下三类运算符：

连接	xy	
选择	x|y	x或y
重复	x^*	x重复0到多次
重复	x^+	x重复1到多次

例:正规式(a|b)*表示的单词符号为: a, b, aa, ab, ba, bb, … 即由a和b组成的任意符号串. 这些符号串的全集叫正规集.

<整数>:=<数字>|<整数><数字>
(1|2|3|4|5|6||7|8|9|0)+

<标识符>:=<字母>| <标识符><字母>| <标识符><数字>
(a|b|c|…)+(a|b|c|…|1|2|3|…)*

对于定义在字母表{a,b}上的语言，分别给出下述正规集的正规式:

a) 所有以a为开头和结尾的符号串 a|a(a|b)*a

b) 第奇数位都是a的所有符号串 (a|b)(a(a|b))*|(a|b)(a(a|b))*a

c) 不具有任何两个连续a的所有符号串 a|b*(ab+)*|b*(ab+)*a

若两个正规式U和V所表示的正规集相同，则称为二者等价，记为U=V.

例如， b(ab)*=(ba)*b

(a|b)*=(a*b*)*

正规式的代数性质

公理	描述
U|V = V|U	|是可交换的
U|(V|W) = (U|V)|W	|是可结合的
(UV)W = U(VW)	·是可结合的
U(V|W) = UV|UW (U|V)W = UW|VW	·对|可分配
ε U = U U ε = U	ε 是·的衡等元素
U** = U*	*是幂等的

- 大写字母为正规式
- | 正规式的或运算
- · 正规式的连接运算
- * 正规式的闭包运算

正规定义

字母表 Σ 上的正规定义形如:

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

…

$d_n \rightarrow r_n$

各个d_i的名字不同，每个r_i是字母表 Σ ∪ {d_1, d_2, …,d_{i-1}}上的正规式, 其中 i=1,2,…,n.

<digit> → 0|1|2|3|4|5|6|7|8|9

<digits> → <digit>+

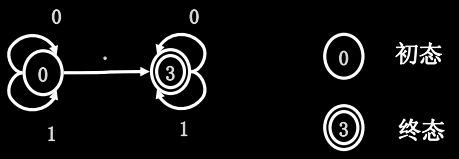
<num> → [+|-]<digits>.<digits>[E[+|-]<digits>]

引入的符号

3.1.2 Finite-State Automata

- Alphabet Σ
- Set of states with initial and accept states
- Transitions between states, labeled with letters

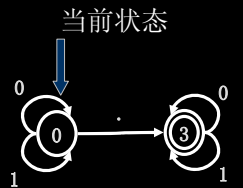
(0 | 1)*.(0|1)*



Automaton Accepting String

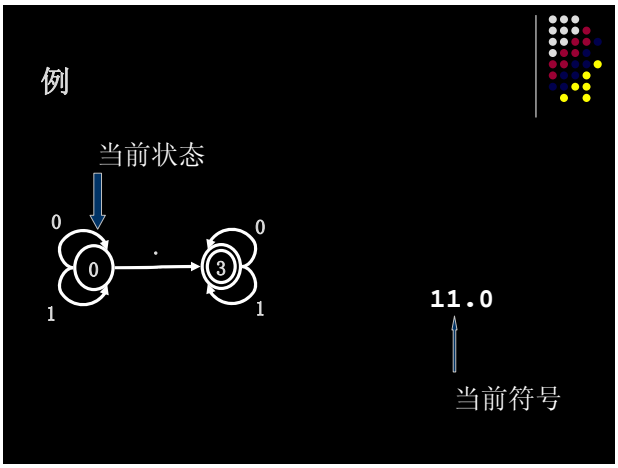
- Conceptually, run string through automaton
- Have current state and current letter in string
- Start with start state and first letter in string
- At each step, match current letter against a transition whose label is same as letter
- Continue until reach end of string or match fails
- If end in accept state, automaton accepts string
- Language of automaton is set of strings it accepts

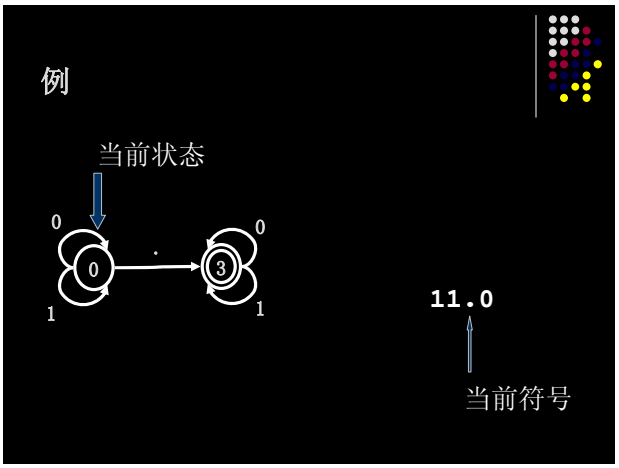
例

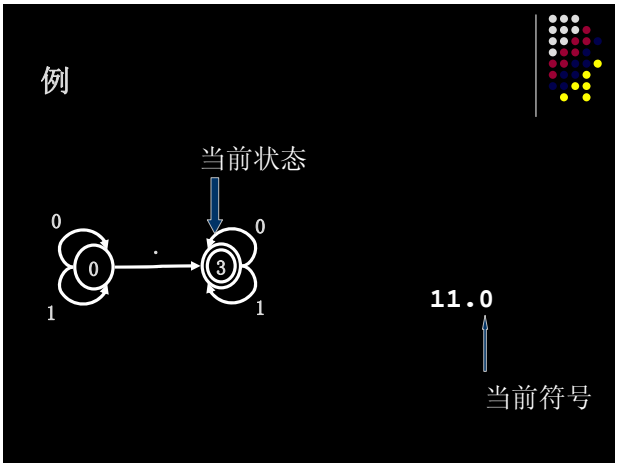


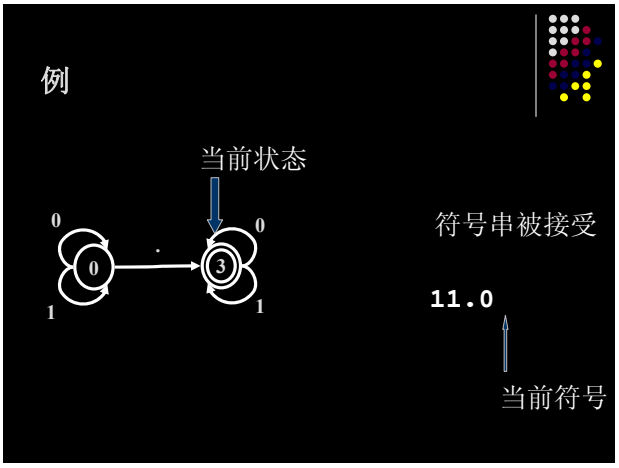
11.0

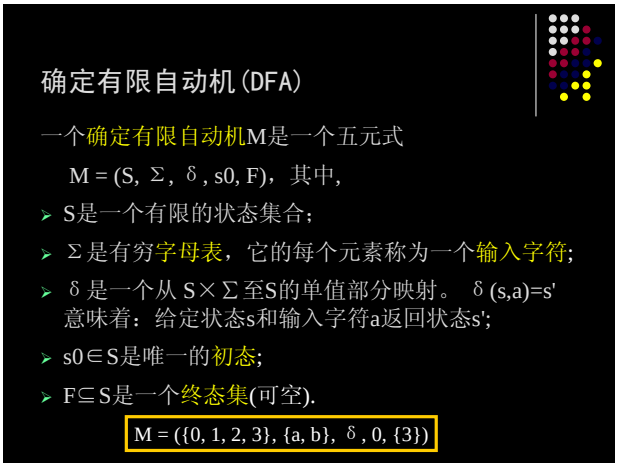
当前符号

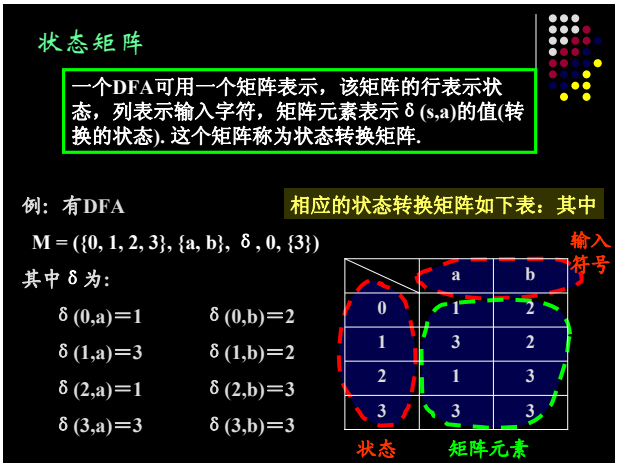

例
当前状态
0
0
0
3
1
1
11.0
当前符号


例
当前状态
0
0
0
3
1
1
11.0
当前符号


例
当前状态
0
0
0
3
1
1
11.0
当前符号


例
当前状态
0
0
0
3
1
1
符号串被接受
11.0
当前符号


确定有限自动机(DFA)
一个确定有限自动机M是一个五元式
M = (S, Σ, δ, s0, F)，其中,
S是一个有限的状态集合;
Σ是有穷字母表，它的每个元素称为一个输入字符;
δ是一个从S×Σ至S的单值部分映射。 δ(s,a)=s'
意味着：给定状态s和输入字符a返回状态s';
s0∈S是唯一的初态;
F⊆S是一个终态集(可空).
M = ({0, 1, 2, 3}, {a, b}, δ, 0, {3})


状态矩阵
一个DFA可用一个矩阵表示，该矩阵的行表示状态，列表示输入字符，矩阵元素表示δ(s,a)的值(转换的状态). 这个矩阵称为状态转换矩阵.
例：有DFA
M = ({0, 1, 2, 3}, {a, b}, δ, 0, {3})
其中δ为:
δ(0,a)=1
δ(0,b)=2
δ(1,a)=3
δ(1,b)=2
δ(2,a)=1
δ(2,b)=3
δ(3,a)=3
δ(3,b)=3
相应的状态转换矩阵如下表：其中
输入符号
a
b
0
1
2
1
3
2
2
1
3
3
3
3
状态
矩阵元素

一个DFA也可用一张（确定的）状态转换图来表示。假定DFA M含有m个状态和n个输入字符，那么，这个状态转换图含有m个状态结点，每个结点顶多有n条箭弧射出和别的结点相连接，整张图含有一个初态结点和若干个（可以为0）终态结点。

M = ({0, 1, 2, 3}, {a, b}, δ, 0, {3})

状态	a	b
0	1	2
1	3	2
2	1	3
3	3	3

$\begin{pmatrix} 1 & 2 \\ 3 & 2 \\ 1 & 3 \\ 3 & 3 \end{pmatrix}$

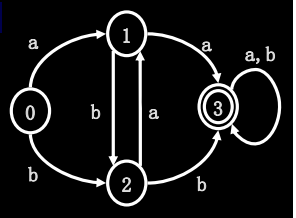
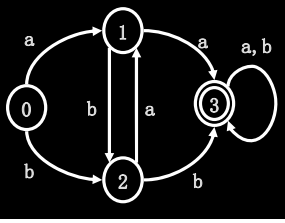


图3.5 状态转换图

给定DFA M = (S, Σ, δ, s_0, F), 对于Σ*中的任意符号串α, 若存在一条从初态结点到终态结点的路径，且这条路径上的所有弧的标记依次连接成一个符号串，同时这个符号串等于α则称α可为DFA M所识别(读出或接受).

给定M = ({0, 1, 2, 3}, {a, b}, δ, 0, {3})有:
S = {0, 1, 2, 3}
Σ = {a, b}
Σ* = { ε, a, b, aa, ab, ba, bb, aaa, aab, …}



例如: 对于Σ*上的符号串:

abaa 识别

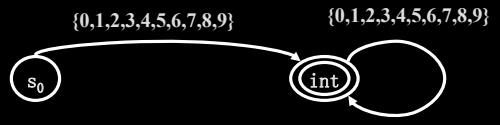
所有为M所识别的Σ*中的符号串组成的集合称为M的语言, 记为L(M).

baba 不识别

正规式和DFA

如果一个DFA M的输入字母表为Σ (M称为Σ上的一个DFA), 则Σ上的一个字集V⊆Σ*是正规的,当且仅当存在Σ上的DFA M使得V=L(M).

(0|1|2|3|4|5|6|7|8|9) (0|1|2|3|4|5|6|7|8|9)*



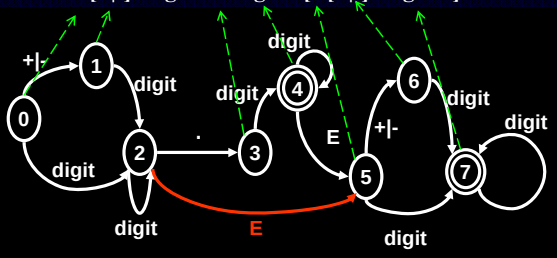
M = ({s_0,int}, {0,1,2,3,4,5,6,7,8,9}, δ, s_0, {int})

δ: $\begin{pmatrix} \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} \\ \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} & \text{int} \end{pmatrix}$

<digit>→0|1|2|3|4|5|6|7|8|9

<digits> →<digit>+

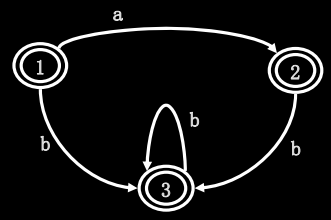
<num> →[+|-]<digits>.<digits>[E[+|-]<digits>]



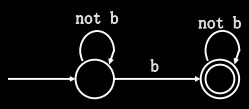
课堂练习: 写出该状态转换图所对应的DFA

123E45 123.0 -123.0 +123.45 -123.45E-67
123.0 -123.0 +123.45 -123.45E-67

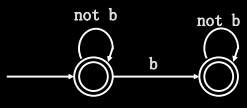
接受正则式 ab+|ab*|b* 的正则集的DFA如下所示:



例3-14：串中只有一个b被如下所示的DFA接受:



例3-15：包含最多一个b的串被如下所示的DFA接受:



3.1.3 非确定有限自动机(NFA)

一个非确定有限自动机M是一个五元式

DFA M = (S, Σ, δ, s_0, F)的确定性表现在δ映射是一个单值函数.即对于任何状态s∈S,和输入符号a∈Σ, δ(s,a)唯一地确定了下一个状态.

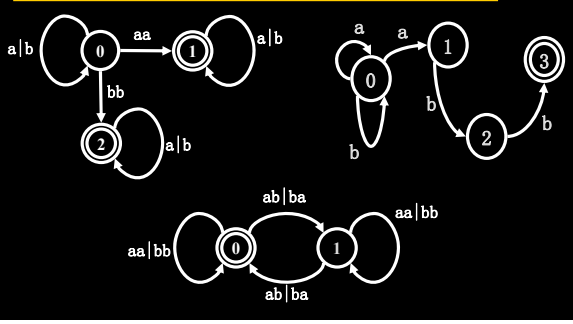
当δ(s,a)的值不唯一时…

给定状态s和输入符号串α,返回状态集合T,其中T⊆S;

- S_0⊆S是非空初态集;
- F⊆S是一个终态集(可空).

每个箭弧上的输入是一个符号串(含ε)

- 状态的迁移不再对于一个输入符号串是唯一的;
- 开始状态可以多个



NFA接受输入串α, 当且仅当转换图中存在从开始状态到终态的一条路径,使得该路径中每条边的输入符号串依次连接成为α

一个NFA也可用一张(确定的)状态转换图来表示。假定NFA M含有m个状态, 每个结点可射出若干个箭弧与别的结点相连,每个弧用Σ*中的一个字作为输入字.整张图至少含有一个初态结点以及0到多个终态结点.

M = ({0, 1, 2, 3}, {a, b}, δ, {0}, {3})

状态	a	b
0	{0,1}	{0}
1	ϕ	{2}
2	ϕ	{3}
3	ϕ	ϕ

$$\begin{pmatrix} \{0,1\} & \{0\} \\ \{\} & \{2\} \\ \{\} & \{3\} \\ \{\} & \{\} \end{pmatrix}$$

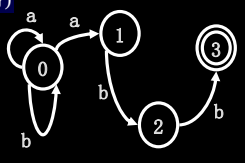


图 状态转换图

对应的正规式为 (a|b)*abb

NFA接受输入串α, 当且仅当转换图中存在从开始状态到终态的一条路径,使得该路径中每条边的输入字依次连接成为α.

对应的正则式为 aa*|bb*

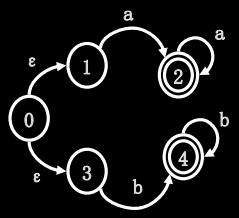
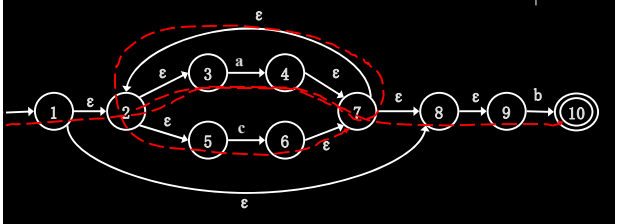


图 状态转换图

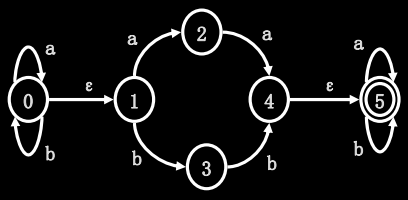
练习: 考虑以下NFA通过怎样的转换接受串acab:



1 →ε 2 →ε 3 →a 4 →ε 7 →ε 2 →ε 5 →c 6

ε →7 ε →2 ε →3 a →4 ε →7 ε →8 ε →9 b →10

NFA接受输入串α, 当且仅当转换图中存在从开始状态到终态的一条路径,使得该路径中每条边的输入字依次连接成为α



可接受的输入串为任意包含连续两个a或连续两个b的符号串

Σ={a,b}

定理：对于每个NFA M存在一个DFA M''使得L(M)=L(M')

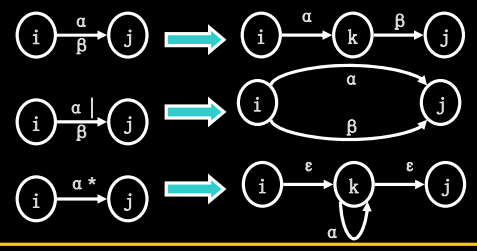
证明：假定NFA M = (S, Σ, δ, S_0, F), 对M的状态转换图进行改变,构造等价的DFA M''

第一步：由NFA M构造等价的NFA M'

第二步：由NFA M'构造等价的DFA M''

①引进新的初态结点x和终态结点y,从x到S_0中每一状态结点连一条输入符号为 ε 的箭弧;从F中每一状态结点连一条输入符号为 ε 的箭弧;

②重复进行如下图所示转换直到每条箭弧上的输入符号串的长度不大于1为止.



将最终得到的NFA记为M', 且有L(M)=L(M').

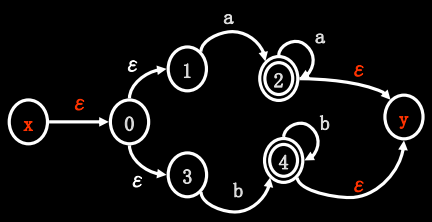
ε _CLOSURE(I) 的定义

DFA是NFA的特例，可以采用子集法将NFA确定化.

假定I是NFA M的状态集的一个子集， 我们定义 ε _CLOSURE(I)为：

- 若s∈I，则s∈ ε _CLOSURE(I)；
- 若s∈I，那么从s出发经过任意条ε弧而能到达的任何状态s'都属于 ε _CLOSURE(I).

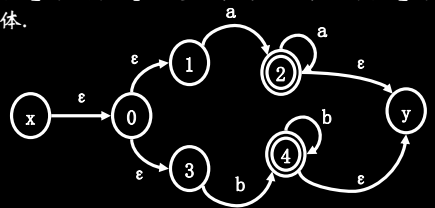
状态集 ε _CLOSURE(I)称为I的 ε _闭包.



ε _CLOSURE({x})={x,0,1,3}
ε _CLOSURE({0})={0,1,3}
ε _CLOSURE({2})={2,y}
ε _CLOSURE({4})={4,y}
ε _CLOSURE({x,0,2})={x,0,1,3,2,y}
ε _CLOSURE({1,3,4})={1,3,4,y}

I_a定义

给定NFA M = (S, Σ, δ, S0, F), 假定I⊆S, a∈Σ, 定义 I_a= ε _CLOSURE(J), 其中, J是那些可从I中的某一状态结点出发经过一条a弧而到达的状态结点的全体.



$\{1\}_a$= ε _CLOSURE({2})={2,y}
$\{4\}_b$= ε _CLOSURE({4})= {4,y}
$\{x,0\}_a$= ε _CLOSURE(φ)= φ
$\{0,3\}_b$= ε _CLOSURE({4})= {4,y}

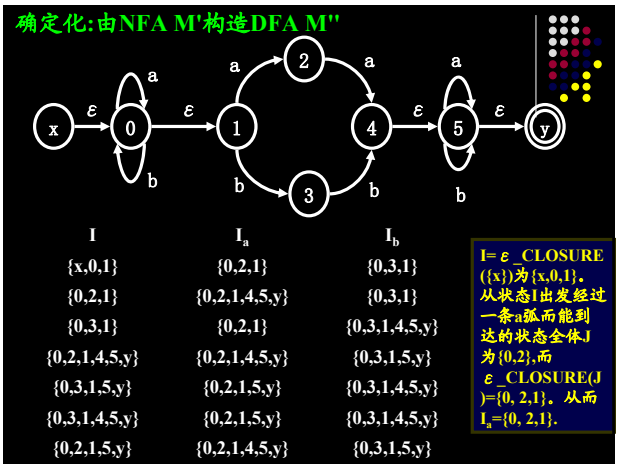
将NFA确定化为DFA的子集法

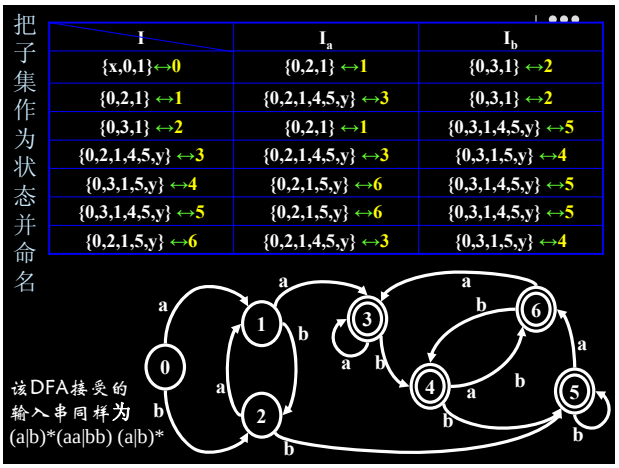
1. 表的初始化构造
2. 处理表的一行
3. 重复处理

设Σ={ a_1,a_2,….a_k }。先构造一张表，该表的行数待定,列数为k+1列. 然后置该表的首行首列为 ε _CLOSURE({x}).

如果某一行的第一列的状态子集已经确定，例如记为I,那么，依次求出I_{a1},I_{a2},…, I_{ak}.同时看它们是否已在表的第一列出现过，将未出现的填入到后面空行的第一列.

重复上述过程，直至所有第2列和第k+1列的子集均已在第一列上出现了为止.

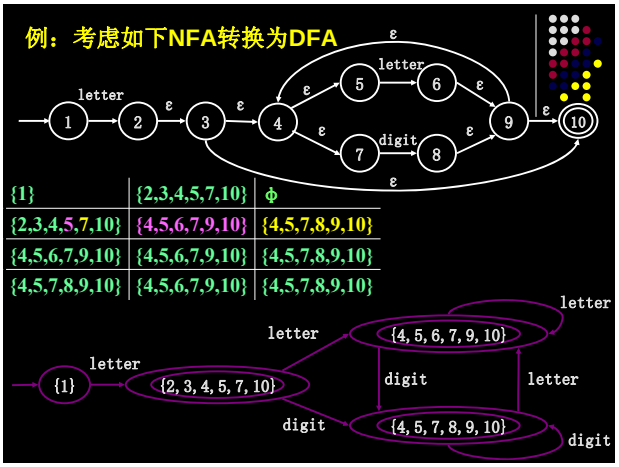




到此我们将这个表作为状态转换矩阵就得到

DFA M''= (S', Σ', δ'', s_0, F')

其中δ''为通过子集算法构造的状态转换矩阵所表示的映射, 而s_0为 ε _CLOSURE({x})命名后的状态名



3.1.4 3型文法与有限自动机的等价性

Chomsky文法 $G=(V_t,V_N,S,P)$

0型文法 $\forall(\alpha,\beta)\in P$ 有：

$\alpha\to\beta,\alpha\in(V_N\cup V_T)^*-(V_T)^*,\beta\in(V_N\cup V_T)^*$

1型文法 G的任何产生式$\alpha\to\beta$均满足$|\alpha|<=|\beta|$, 仅S→ ε 除外，且β中不含S;

2型文法 $A\to\beta,A\in V_N,\beta\in(V_N\cup V_T)^*$

3型文法 $A\to\alpha B$,或,$A\to\alpha$,且$A,B\in V_N,\alpha\in(V_T)^*$

正规文法

3型文法有两种表示形式：左线性文法和右线性文法

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

$A\to B\alpha$,或,$A\to\alpha$,且$A,B\in V_N,\alpha\in V_T^*$

$A\to\alpha B$,或,$A\to\alpha$,且$A,B\in V_N,\alpha\in V_T^*$

对于3型文法 (正规文法) G和有限自动机M, 如果L(G)=L(M), 则称G和M是等价的.

关于3型文法和有限自动机的等价性，有以下结论：

①对于每一个3型文法G, 都存在一个有限自动机M, 使得L(M) = L(G).

②对于每一个有限自动机M, 都存在一个3型文法G, 使得L(M) = L(G).

定理：对于每一个右线性文法G, 都存在一个有限自动机M, 使得L(M) = L(G).

$G = (V_T, V_N, S, P)$ 作如下构造得到的M为一个NFA

令$M = (V_N \cup \{f\}, V_T, \delta, S, \{f\})$,其中$f \notin V_N$,且$\delta$如下：

1) 若对于某个$A \in V_N$及$a \in V_T \cup \{\varepsilon\}$有$(A,a) \in P$

则令$\delta(A,a) = f$;

2) 对任意$A \in V_N$及$a \in V_T \cup \{\varepsilon\}$，设P 中左端为$A$右端第一符号为$a$的产生式形如$A \to aA_1 \mid aA_2 \mid ... \mid aA_k$

则令$\delta(A,a) = \{A_1, ..., A_k\}$.

定理：对于每一个有限自动机M,都存在一个右线性文法G, 使得$L(M) = L(G)$.

$DFA\quad M = (S, \Sigma, \delta, s_0, F)$ 做如下构造得到右线性文法G

1) 若$s_0 \notin F$, 令$G = (\Sigma, S, s_0, P)$, 其中P构造如下：

对于任何$a \in \Sigma$及$A, B \in S$, 若有$\delta(A,a) = B$, 则：

i)当$B \notin F$时， 令$A \to aB$

ii)当$B \in F$时， 令$A \to a \mid aB$

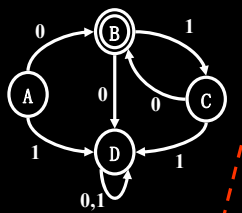
2)$s_0 \in F$, 因为$\delta(s_0, \varepsilon) = s_0$, 所以$\varepsilon \in L(M)$, 从而

$L(G) = L(M) - \{\varepsilon\}$.

因此令$G' = (\Sigma, S \cup \{s_0'\}, s_0', P \cup \{(s_0', s_0), (s_0', \varepsilon)\})$

其中$s_0' \notin S$.最后令G=G'

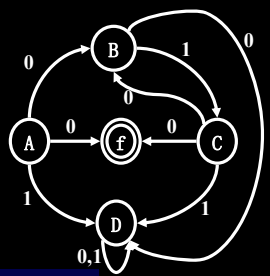
DFA M=({A,B,C,D},{0,1}, δ , A, {B})



G_L=({0,1},{B,C,D,f}, f, P)

f→0|C0　　B→0|C0

C→B1　　D→1|C1|D0|D1|B0



G_R=({0,1},{A,B,C,D}, A, P)

A→0|0B|1D　　B→0D|1C

C→0|0B|1D　　D→0D|1D

NFA M'=({A,B,C,D,f},{0,1}, δ , A, {f})

3.1.5 正规式与有限自动机的等价性

关于正规式和有限自动机的等价性，有以下结论：

1、对于任何有限自动机M，都存在一个正规式R，使得L(R) = L(G)。

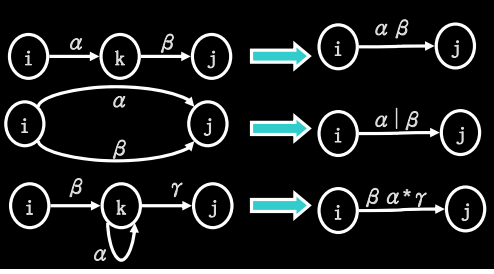
2、对于任何正规式R,都存在一个有限自动机M,使得L(M) = L(R)。

定理：对于任何有限自动机M，都存在一个正规式R，使得L(R) = L(M).

1. 对Σ上的NFA M构造Σ上的正规式R使得L(R)=L(M).

令每条弧以正规式作标记；在图中加进X和Y两个结点，从X用ε弧连接到M的所有初态结点，从M的所有终态结点用ε弧分别连接到Y;将X作为初态，Y作为终态构成新的一个NFA M'且L(M)=L(M').

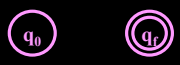
采用图示替换规则消去M'中所有结点，只剩下X和Y，则X到Y的弧的标记就是R



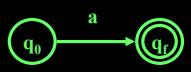
2. 对Σ上的正规式r构造Σ上的NFA M使得L(M)=L(r)

- r= ε

- r= φ

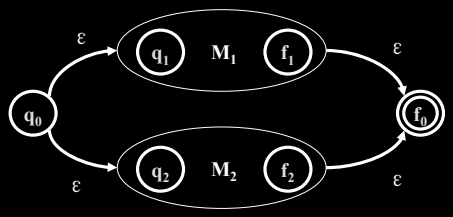


- r=a



2续. 对Σ上的正规式R构造Σ上的NFA M使得L(M)=L(R)

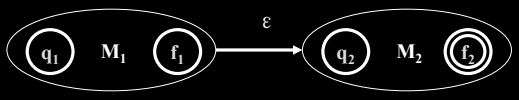
- $r=r_1|r_2$



$L(r)=L(M_1) \cup L(M_2)$

2续. 对Σ上的正规式R构造Σ上的NFA M使得L(M)=L(R)

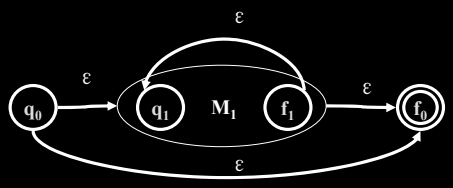
- $r=r_1r_2$



$L(r)=L(M_1)\ L(M_2)$

2续. 对Σ上的正规式R构造Σ上的NFA M使得L(M)=L(R)

- $r=r_1^*$



$L(r)=L(M_1)^*$

结论

- 正规文法、正规式、确定有限自动机、非确定有限自动机在接受语言能力上等价。

3.1.6 确定有限自动机的化简

一个确定有限自动机M的化简是指：

寻找一个状态数比M少的DFA M'，使得L(M)=L(M')。

一个DFA M的状态最少化过程旨在将M的状态集分割成一些不相交的子集，使得任何不同的两个子集中的状态都是可区别的，而同一子集中的任何两个状态都是等价的。最后，在每个子集中选出一个代表，同时消去其它等价状态。

M的两个状态s和t是等价的：当且仅当分别从二者出发至终态均能够读出同一字。

M的两个状态s和t是可区别的：当且仅当二者不等价。

状态集S的划分步骤:

(1)基本划分{终结状态集，其他状态集}是一个划分Π；

(2)对于当前划分Π={$I^{(1)},\ldots,I^{(m)}$}, m>=1; 进行如下处理:

对于每个a∈Σ, 若$I_a^{(i)}$分别与Π的n个不同块相交不为空（其中1<=n<=m），则将$I^{(i)}$划分成n个不相交的子集，使得每个子集J的J_a包含于Π的某个块。然后将每个J加入Π并删除$I^{(i)}$

(3)重复上述过程(2)直至Π不再变化为止。

$$\Pi=\{X_1,\cdots,X_m\},\quad X_i\subseteq X, i=1\cdots m, m\geq 1$$

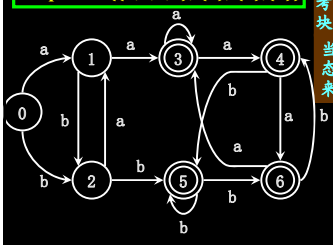
$$X_i\cap X_j=\varphi, \bigcup_{i=1}^{m} X_i = X$$

例：考虑下图所示DFA的化简：

Step1: Π={{3,4,5,6}, {0,1,2}}

Step2: Π={{3,4,5,6}, {1}, {0,2}}

Step3:Π={{3,4,5,6}, {1}, {0},{2}}



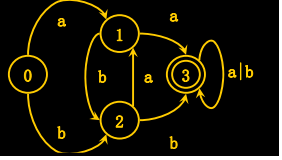
把M的状态分为2组：终态组{3,4,5,6}, 非终态组{0,1,2}

考察{3,4,5,6},由于$\{3,4,5,6\}_a$和$\{3,4,5,6\}_b$都包含于{3,4,5,6},所以它不能再划分

考察{0,1,2},由于$\{0,1,2\}_a=\{1,3\}$，分别与{3,4,5,6和{0,1,2}相交不空，故进行划分。因δ(1,a)=3且δ(0,a)=δ(0,a)=1，故将{0,1,2}分成{1}, {0,2}

考察{0,2},由于$\{0,2\}_b=\{2,5\}$，与上述3块中的2块相交不为空，故划分成{0},{2}

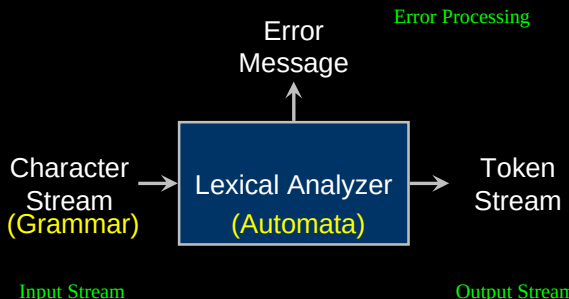
当前Π不能再划分，最后重新命名状态，其中新状态3代表{3,4,5,6}，把原来射入3,4,5,6的弧改为射入新状态3



3.2 词法分析器的设计与实现

- Introduction
- Design Issues
- Examples
- Generators

3.2.1 Introduction to Lexical Analyzer



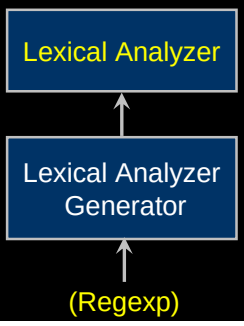
Character Stream

- Source Code
  - Programming Languages
  - Script Languages (HTML,XML)
  - Others
- Grammar
  - Alphabet
  - Production rules

Token Stream

- Category
- Value
- Symbol Table
- Other Tables

Lexical Analyzer Generator



3.2.2 Design Issues on Lexical Analyzer

- Token的表示
  - 全体一种
  - 一字一种
- 词法分析器与语法分析器的关系
  - 单独一遍
  - 作为语法分析器的子程序
- 输入缓冲区的控制
  - 预处理
  - 超前搜索
- DFA的表示
  - Matrix
  - Case语句

①单词符号的分类及表示

- *关键字*：由程序语言定义的具有固定意义的标识符。也可称为保留字或基本字。例如：Pascal中的begin，end，if等。它是确定的。
- *标识符*：用来表示各种名字，如变量名、数组名、过程名等。它是不限的。
- *常数*：常数的类型一般有整型、实型、布尔型、文字型等。它是不限的。
- *运算符*：如+、-、*、/ 等。它是确定的。
- *界符*：如逗号、分号、括号、/*，*/ 等。它是确定的。

Token的表示

- 用二元组表示：<种别，属性值>
- 种别：采用整数表示、常量表示或者特殊标识符表示
- 属性值：对于常数采用内部表示；标识符（变量、常量、数组名、函数名等）采用符号表入口，等等。

种别的确定：主要取决于实现上的方便，例如：标识符一般同归为一种。常数宜按类型（整、实、布尔）分。关键字可以将其全体视为一种，也可一字一种。运算符可采用一符一种，但也可把具有一定共性的视为一种。界符则一般采用一符一种。若是一符一种分种，单词自身值就不需要了。

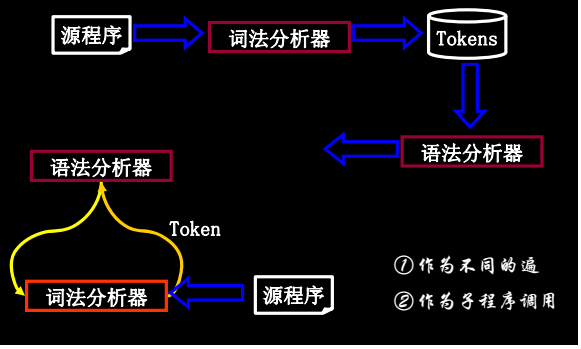
- 例：FORTRAN编译程序的词法分析器在扫描输入串 IF (5·EQ·M) GOTO 100 后，它输出的单词符号流是：

逻辑IF （34，_）
左括号 （2，_）
整常数 （20，‘5’的二进制表示）
等号 （6，_）
标识符 （26，‘M’）
右括号 （16，_）
GOTO （30，_）
标号 （19，‘100’的二进制表示）

- 例：C++代码段：while (i >= j) i --； 经词法分析器处理以后，它将被转换为如下的单词符号流：

(while，_)
((，_)
(id，指向i的符号表指针)
(>=，_)
(id，指向j的符号表指针)
()，_)
(id，指向i的符号表指针)
(--，_)
(；，_)

②词法分析器与语法分析器的关系



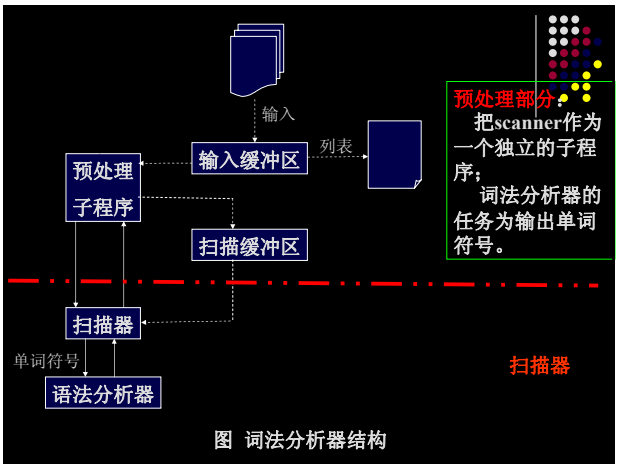
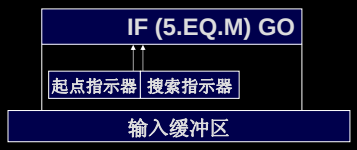


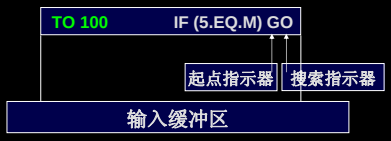
图 词法分析器结构

③超前搜索问题

若识别输入语句 IF (5.EQ.M) GOTO 100，若缓冲区情况如下所示：



若识别输入语句 IF (5.EQ.M) GOTO 100，若缓冲区情况如下所示：



扫描缓冲区的结构：

缓冲区大小：120个字符。

采用两个指示器：起点指示器、搜索指示器。

两个互补区。

例：关键字识别中的超前搜索：

例如：在标准FORTRAN中

1、DO99K = 1,10
2、IF(5.EQ.M)I = 10
3、DO99K = 1.10
4、IF(5) = 55

其中的DO、IF为关键字

其中的DO、IF为标识符的一部分

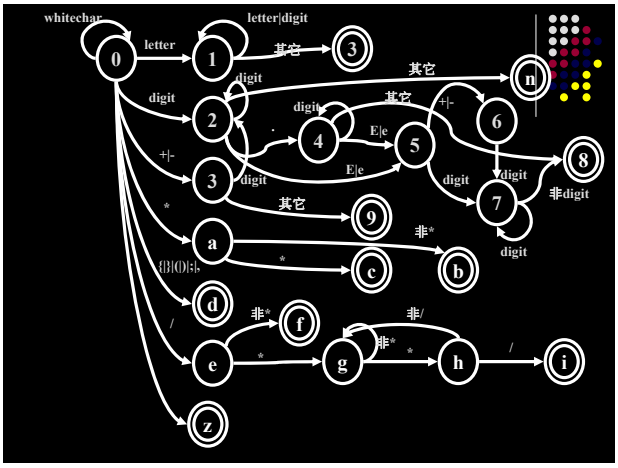
- **标识符的识别**
  - 多数语言的标识符是字母开头的“字母/数字”串，而且在程序中标识符的出现后都跟着算符或界符。因此，不难识别。
- **常数的识别**
  - 对于某些语言的常数的识别也需要使用超前搜索。
- **算符和界符的识别**
  - 对于诸如C++语言中的“++”、“--”，这种复合成的算符，需要超前搜索。

④状态转换图的实现

1. ch 字符变量，存放最新读进的源程序字符。
2. TOKEN 字符数组，存放构成单词的字符串。
3. GETCHAR 过程，将下一输入字符读入ch，搜索指示器前移一个字符。
4. GETBC 过程，检查ch中的字符是否为空白。若是，则调用GETCHAR直至CHAR中进入一个非空白字符。
5. CONCAT 过程，把ch中的字符连接到TOKEN之后。
6. IsLetter 布尔函数过程，它们分别判断ch中的字符是数字或是字母，IsDigit 从而给出真假值TRUE、FALSE。
7. Reserve 整型函数过程，用TOKEN中的字符串查保留字表，若是一个保留字则给予编码，否则回送0值（假定0不是保留字的编码）。
8. Retract 过程，把搜索指示器回调一个字节，把ch中的字符置为空白。
9. InsertID 插入符号表，返回入口（指针）
10. InsertConst 插入常数表，返回入口（指针）

3.2.3 Examples

- C程序（作为子程序）



符号表

```
struct entry
{
        struct entry *next;    /* linked list implementation */
        char *name;  /* the user-defined name of the identifier */
        int type;        /* its type */
        int blockno;   /* scoping information */
        int addr;        /* address assigned by code generator */
};
```

```
/* constants for our scanner */
#define     LPAREN      '('         /* single char tokens */
#define     RPAREN      ')'
#define     COMMA       ','
#define     GREATER     '>'
#define     LESS        '<'
#define     ASSIGN      '='
...
#define     WHILE       257         /* reserved words */
#define     PRINT       258
#define     RETURN      259
#define     IF          260
...
#define     VARIABLE    268         /* other tokens */
#define     INTEGER     269
#define     TEXT 270
#define     EOF         271         /* end of file */
#define     UNKNOWN 272             /* don't recognize */
```

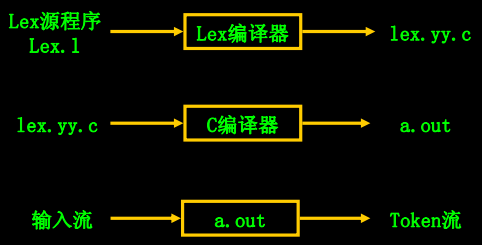
```
/* some globals */

int ch;                     /* next char */
int intval;                 /* integer lexeme */
char textval[255];          /* text string lexeme */

void init()
{
        /* set up a lookup table for reserved words */
        create_reserved_table();
        insert_reserved(WHILE, "WHILE")
        insert_reserved(PRINT, "PRINT")
        insert_reserved(RETURN, "RETURN")
        ....
        /* get the first character */
        ch = getchar();
}
```

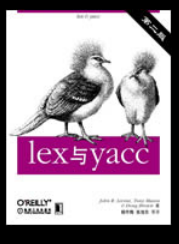
```
int scanner(){
switch(ch){
  case ' ': /* white space */
  case '\n':
  case '\t':
          while (isspace(ch = getchar())); /* get rid of the white */
          return scanner();
  case '/':       /* check for a comment or DIVIDE operator */
          ch = getchar();
          if (ch != '/')
                  return DIVIDE;
          ...     /* code to skip over a comment */
  case '(':
  case ')':
  case ',':....   /* and any other single char tokens */
          intval = ch;
          ch = getchar();
          return intval;
```

```
case 'A': /*assume reserved words are caps, variables lower case*/
case 'B':
case 'C': ...    /* reserved words */
          textval[0] = ch;
          for (i = 0; isupper(ch = getchar()); textval[i++] = ch)
                  ;
          textval[i++] = '\0';
          return lookup_reserved(textval);
case 'a':
case 'b':
case 'c':...     /* variables */
          textval[0] = ch;
          for (i = 0; islower(ch = getchar()); textval[i++] = ch);
          textval[i++] = '\0';
          if (lookup_symtab(textval) == UNKNOWN)
          add_symtab(textval);
          return VARIABLE;
```

```
 case '0':
 case '1':....                  /* integer */
          intval = ch - '0';    /* convert to a number */
          while (isdigit(ch = getchar())
          intval = intval * 10 + ch - '0';
          return INTEGER;
 case EOF:
          return EOF;
 default:
          intval = ch;
          ch = getchar();
          return UNKNOWN;
 }
}
```

3.2.4 词法分析器的自动产生





《lex与yacc（第二版）》
作者: John R. Levine,Tony Mason,Doug Brown 著
杨作梅,张旭东,等 译
出版: 2003年1月
书号: 7-111-10721-7
页数: 392
定价: 45.00元

```
<程序>::=<程序首部>；<分程序>.
        <程序首部>::=program<标识符>
        <分程序>::=<复合语句>
        <复合语句>::=begin<语句序列>end
        <语句序列>::=<语句>{；<语句>}
         <语句>::=<赋值语句>|<复合语句>|<条件语句>
         <赋值语句>::=<标识符>:=<表达式>
         <条件语句>::=if <布尔表达式> then <语句> else <语句>
         <表达式>::=<项>{(+|-)<项>}
         <项>::=<因式>{(*|/)<因式>}
         <因式>::=<标识符>|<无正负号常量>|' (' <表达式>' )'
         <布尔表达式>::=<表达式><关系运算符><表达式>
         <关系运算符>::= =|<|<=|>|>=|<>
         <标识符>::=<字母>{<字母>|<数字>}
         <无正负号常量>::=<数字>{<数字>}[.<数字>{<数字>}]
         <字母>::=a|b|c|d|e|f|g|……|u|v|w|x|y|z
         <数字>::=0|1|2|3|4|5|6|7|8|9
```

输入以 "#" 为结束符

Lex程序结构

声明　　变量、常量和正规定义
%%
翻译规则　　正规式　{动作}
……
%%　　正规式　{动作}
辅助过程

A-Z, 0-9, a-z	构成了部分模式的字符和数字。
.	匹配任意字符，除了 \n。
-	用来指定范围。例如：A-Z 指从 A 到 Z 之间的所有字符。
[ ]	一个字符集合。匹配括号内的*任意*字符。如果第一个字符是 ^ 那么它表示否定模式。例如：[abC] 匹配 a, b, 和 C中的任何一个。
*	匹配*0个*或者多个上述的模式。
+	匹配*1个*或者多个上述模式。
?	匹配*0个或1个*上述模式。
$	作为模式的最后一个字符匹配一行的结尾。
{ }	指出一个模式可能出现的次数。 例如：A{1,3} 表示 A 可能出现1次或3次。
\	用来转义元字符。同样用来覆盖字符在此表中定义的特殊意义，只取字符的本意。
^	否定。
|	表达式间的逻辑或。
"<一些符号>"	字符的字面含义。元字符具有。
/	向前匹配。如果在匹配的模版中的“/”后跟有后续表达式，只匹配模版中“/”前的部分。如：如果输入 A01，那么在模版 A0/1 中的 A0 是匹配的
( )	将一系列常规表达式分组。

```
%{常量定义… %}
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
%%
{ws}    {}
if          {return(IF);}
then        {return(THEN);}
else        {return(ELSE);}
{id}        {yylval=install_id();return(ID);}
{number}    {yylval=install_num();return(NUMBER);}
“<”         {yylval=LT;return(RELOP);}
…
%%
install_id() { 单词装入符号表并返回指针；    }
…
```

上机实验A

- 见ftp上的“编译原理专题实验”目录，根据给定的C_0语言的文法，采用Lex或者编写程序构造词法分析器。
- 上机时间第四周、第五周：计算机41，42，43，44安排在同一时间。由班级负责同学到计算机系实验中心登记，定了以后通知辅导老师。
- 实验部分以当时上机情况和提交的报告进行评分，占本课程成绩10%。

实验题目

实验A：构造词法分析器，采用Lex生成，或者编程实现。

实验B：构造语法分析器，采用Yacc生成，或者编程实现。

实验C：产生中间代码，具体内容另给。

ftp上的“编译原理专题实验”目录内容：
ftp 202.117.15.193 用户名和密码均为parallel

- 实验I：借助Lex和Yacc进行词法语法分析
  - *实验一lex与yacc的使用.doc*
- 实验II：用C或Java编写词法分析器
  - *实验二词法分析器.doc*
- 实验III：用C或Java编写语法分析器
  - *实验三语法分析.doc*

C_0语法

- program→ { var-declaration | fun-declaration }
- var-declaration→ int ID { , ID }
- fun-declaration→ (int | void) ID (params) compound-stmt
- params → int ID { , int ID } | void | empty
- compound-stmt→ { { var-declaration } { statement } }
- statement→ expression-stmt | compound-stmt | if-stmt | while-stmt | return-stmt
- expression-stmt→ [expression] ;

C_0语法（续）

- if-stmt→ if(expression) statement [else statement]
- while-stmt→ while(expression) statement
- return-stmt→ return [expression] ;
- expression→ ID = expression | simple-expression
- simple-expression→ additive-expression [relop additive-expression]

C_0语法（续）

- relop → < | <= | > | >= | == | !=
- additive-expression→ term [(+ | -) term]
- term→ factor [(* | /) factor]
- factor→ (expression)| ID | call | NUM
- call→ ID(args)
- args→ expression { , expression } | empty
- ID →… ;参见C语言标识符定义
- NUM →… ;参见C语言数的定义

报告提交

- 实验报告应该按照格式填写。
- 报告中写明程序设计思想及简要设计方案，代码简要描述，运行测试情况，体会。
- 一共提交3个报告，分别对应实验A，B，C
- 提交方式：作为附件email到指定邮箱，若2天内得到回复确认表示提交成功。
- 指定邮箱为：yinliang_zhao@163.com
- 附件命名格式：姓名学号班级实验X.rar
- 附件(.rar)中含有：源程序、测试数据结运行结果、报告、其他辅助文件（若有）

报告提交

- 例子
  - 赵银亮95434软件91实验A.rar
  - 该文件中包括：实验报告，代码，数据等