课本可以不用看完，但是黄煜廉老师的课件以及这一份上面所有的都是他上课讲的例题都要掌握下来~能把这一份啃下来就几乎可以的了。加油！最后能为你们做的事情啦~(*^__^*) 嘻嘻……

——雪清

正则表达式：

例 2.1 在仅由字母表中的 3 个字符组成的简单字母表∑={a, b, c}中，考虑在这个字母表上的仅包括一个 b 的所有串的集合。

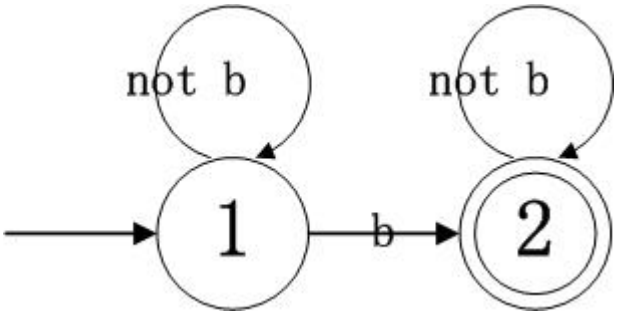
(a | c)* b (a | c)*

例 2.2 在与上面相同的字母表中，如果集合是包括了最多一个 b 的所有串。

(a | c)* b? (a | c)*

DFA：

例 2.6 串中仅有一个 b 的集合的正则表达式对应的 DFA 为？



例 2.8 科学表示法的数字常量的正则表达式为：

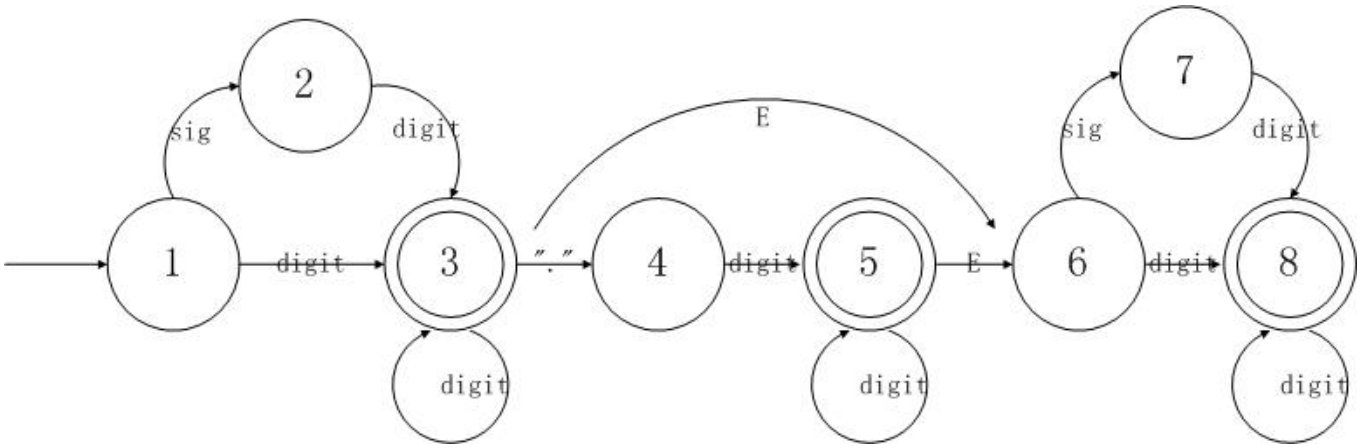
nat = [0-9]+

signedNat = (+|-)? nat

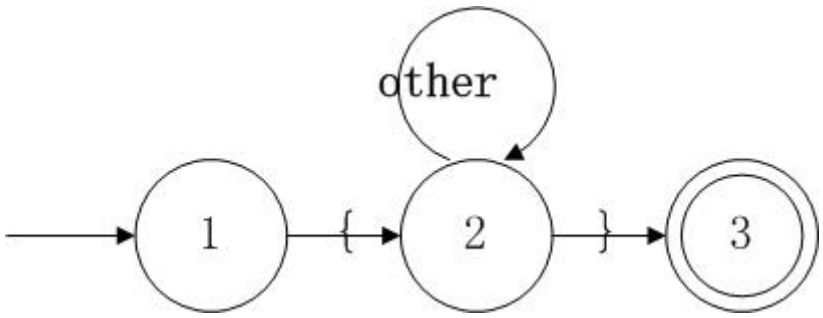
number = signedNat(“.” nat)? (E signedNat)?

如何画对应的 DFA？

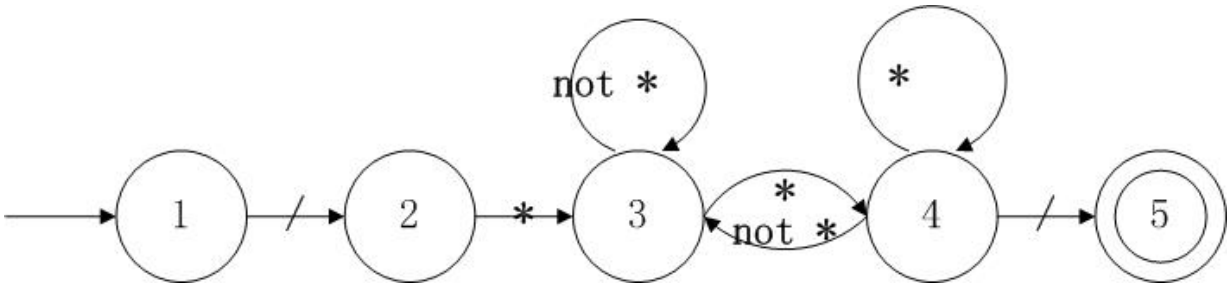
解：先设 digit = [0-9]，sig = (+|-)，得：



例 2.9 非嵌套注释的 DFA 描述。Pascal 注释{ (~})* }对应的 DFA 为：

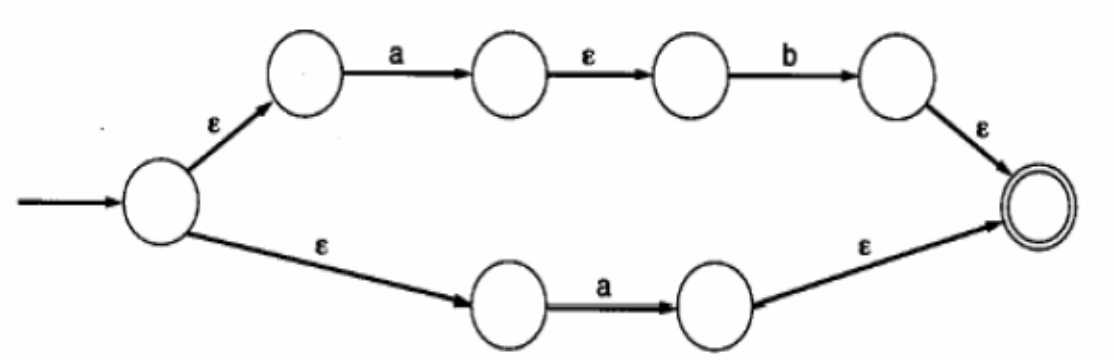


C 注释 /* …(*/不同时出现)… */ 的 DFA 为：

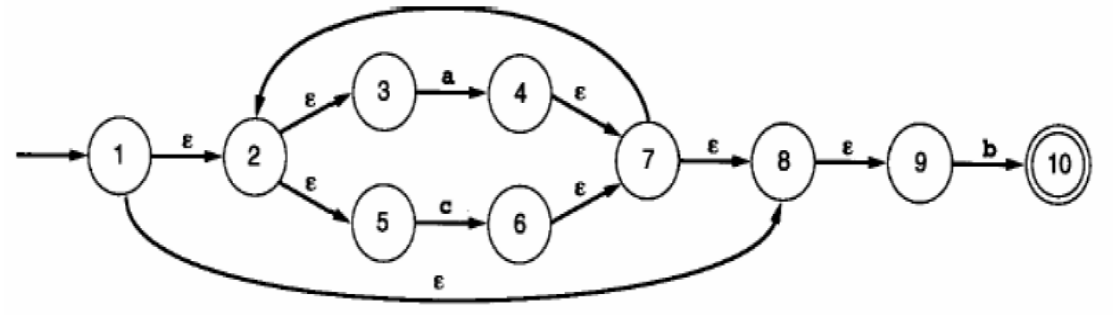


NFA：

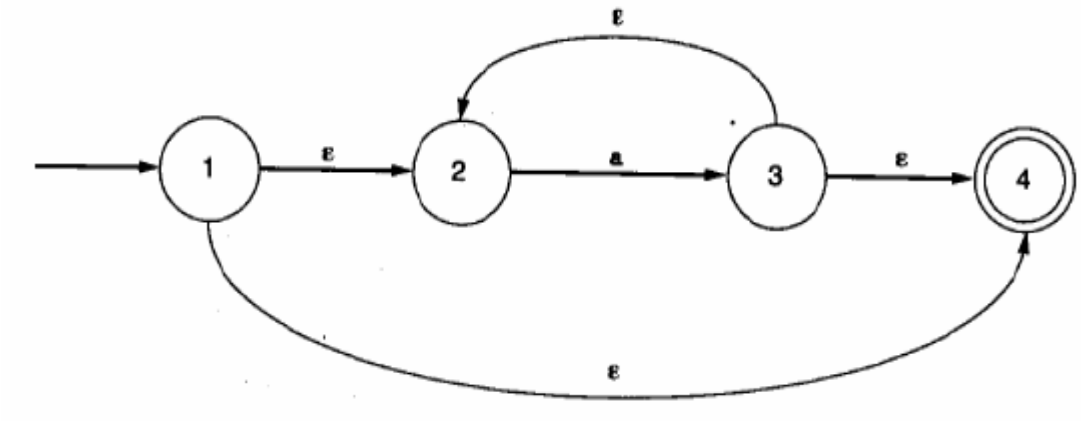
例 2.12 根据 Thompson 方法将正则表达式 ab|a 转换为 NFA。



例 2.13 利用 Thompson 方法画出正则表达式 letter(letter| digit)*对应的 NFA。

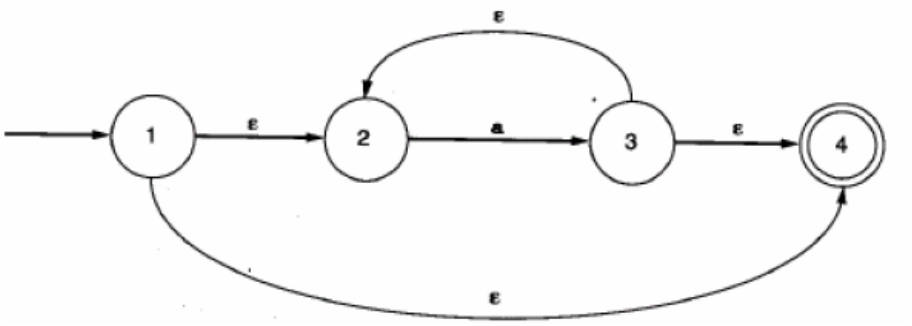


例 2.14 与正则表达式 a*相对应的 NFA 为：

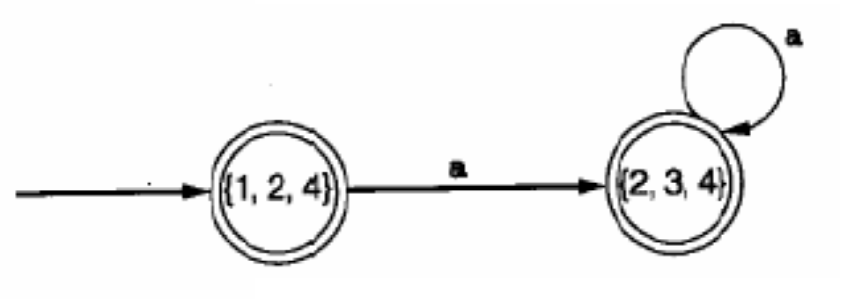


NFA 转 DFA：

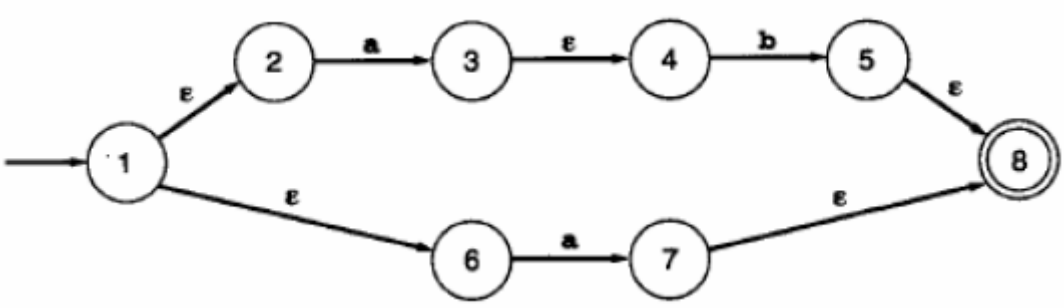
例 2.15 将下面的 NFA 转换为 DFA：



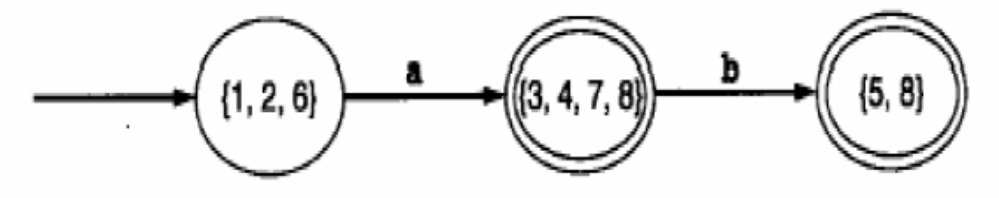
解：



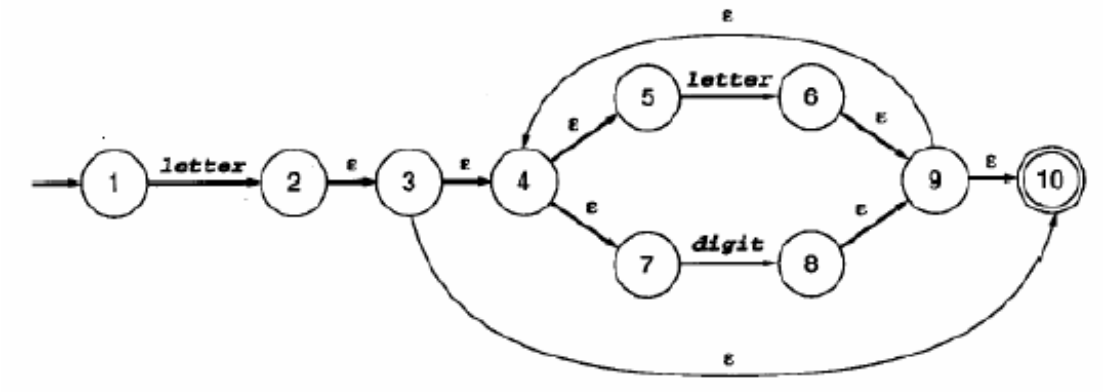
例 2.16 将下面的 NFA 转换为 DFA：



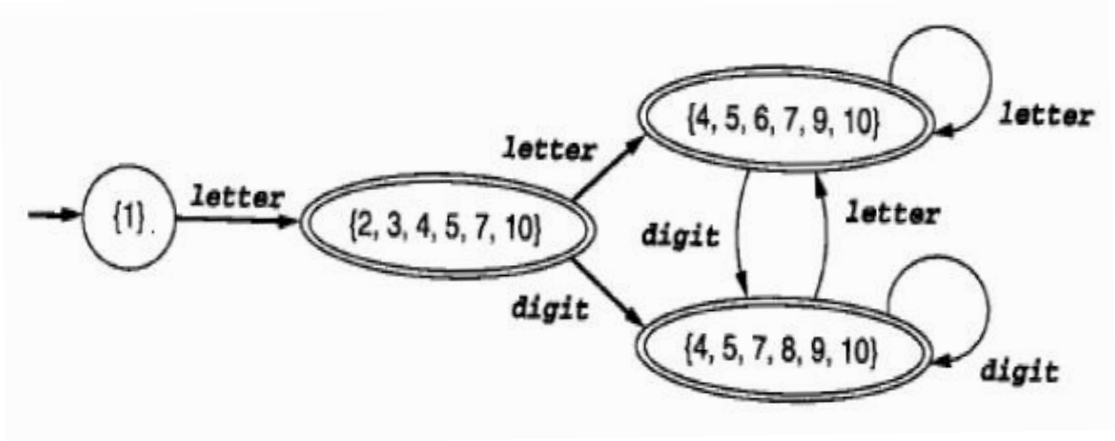
解：



例 2.17 正则表达式 letter(letter| digit)*对应的 NFA 转换成 DFA：

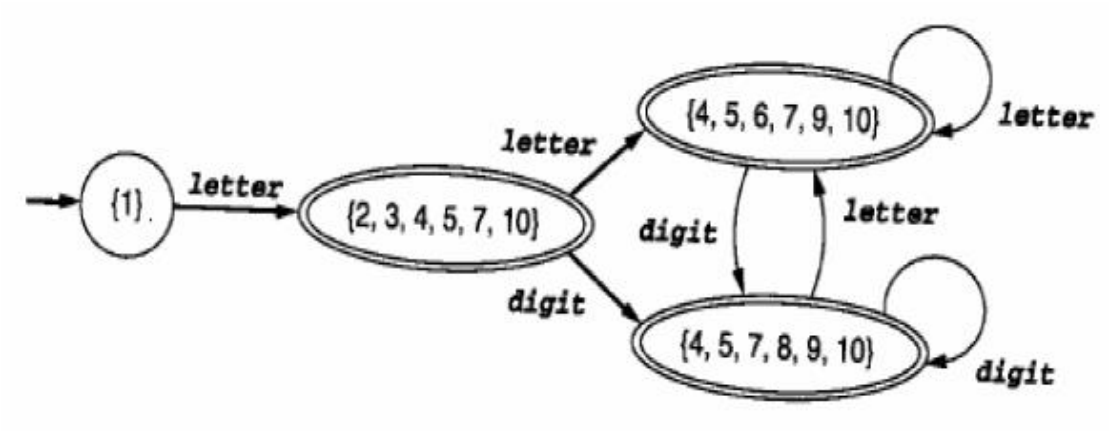


解：



DFA 最小化：

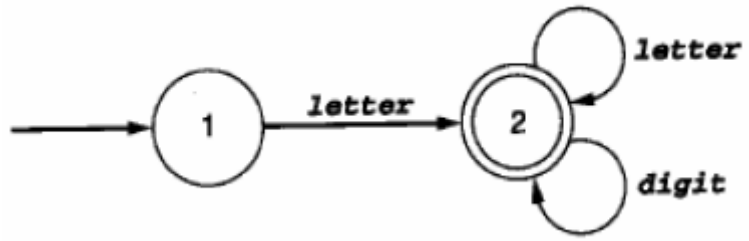
例 2.18 将与正则表达式 letter(letter| digit)*相对应的 DFA 最小化：（08 级的大三第二学期考这道）



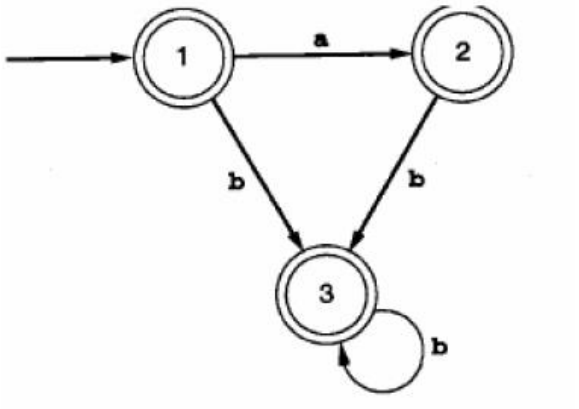
解：状态转换表为

	letter	digit
{1}	{2,3,4,5,7,10}	
{2,3,4,5,7,10}	{4,5,6,7,9,10}	{4,5,7,8,9,10}
{4,5,6,7,9,10}	{4,5,6,7,9,10}	{4,5,7,8,9,10}
{4,5,7,8,9,10}	{4,5,6,7,9,10}	{4,5,7,8,9,10}

最小化 DFA 为：



例 2.19 将下面与正则表达式(a| ε) b*对应的 DFA 进行最小化。

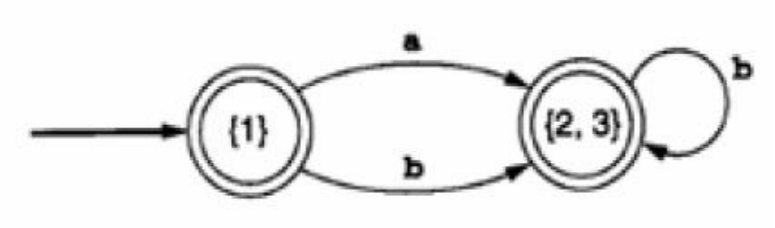


解：状态转换表为

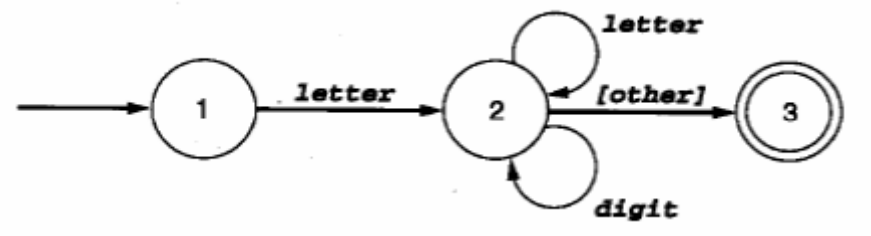
	a	b

{1}	{2}	{3}
{2}		{3}
{3}		{3}

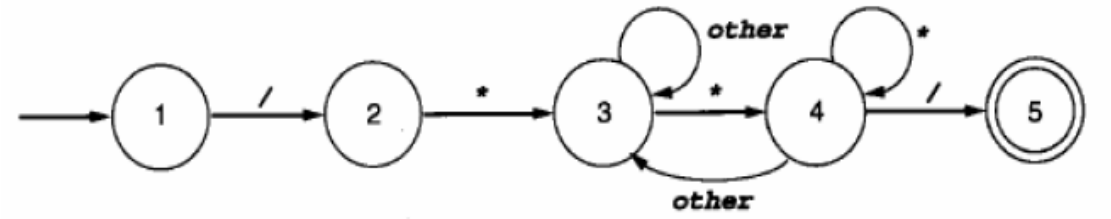
最小化 DFA 为：



词法分析代码：




```
state := 1; {start}
while state = 1 or 2 do
    case state of
    1: case input character of
        letter: advance the input;
                    state := 2;
        else state := ... {error or other};
        end case;
    2: case input character of
        letter, digit:  advance the input;
                            state := 2; {actually unnecessary}
        else state := 3;
        end case;
    end case;
end while;
if state = 3 then accept else error;
```




```
state := 1; {start}
while state = 1, 2, 3 or 4 do
    case state of
    1:  case input character of
        "/":    advance the input;
                state := 2;
        else state := ... ; {error or other}
        end case;
    2:  case input character of
        "*":    advance the input;
                state := 3;
        else state := ... ; {error or other}
        end case;
    3: case input character of
        "*":    advance the input;
                state := 4;
        else advance the input; {and stay in state 3}
        end case;
    4:  case input character of
        "/" :   advance the input;
                state := 5;
        "*":    advance the input; {and stay in state 4}
        else    advance the input;
                state := 3;
        end case;
    end case;
end while;
if state = 5 then accept else error;
```

递归子程序分析法

问题 1：

G[S] =

{

S → aA | bB

A → cdA | d

B → efB | f

}

试编写一个能分析该文法所对应任何串（如串 acdd）的程序。

```
void match( expectedToken ){
    if( token == expectedToken )
        getToken();
    else
        Error();
}

void S(){
    if( token == 'a' ){
        match('a');
        A();
    }else if( token == 'b' ){
        match('b');
        B();
    }else Error();
}//S

void A(){
    if( token == 'c' ){
        match('c');
        match('d');
        A();
    }else if( token == 'd' ){
        match('d');
    }else Error();
}//A

void B(){
    if( token == 'e' ){
        match('e');
        match('f');
        B();
    }else if( token == 'f' ){
        match('f');
    }else Error();
}//B

int main(){
    getToken();
    S();
```

```
    return 0;
}//main
```

问题 2：

exp → exp addop term | term

addop → + | -

term → term mulop factor | factor

mulop → * | /

factor → (exp) | number

请写出递归子程序分析算法。

解：先消除左递归，得到

exp → term { addop term }

addop → + | -

term → factor { mulop factor }

mulop → * | /

factor → (exp) | number

程序如下：

```
void match( expectedToken ){
     if( token == expectedToken )
          getToken();
     else
          Error();
}
void exp(){
     term();
     while( token == '+' || token == '-' ){
          match(token);
          term();
     }
}
void term(){
     factor();
     while( token == '*' || token == '/' ){
          match(token);
          factor();
     }
}
void factor(){
     if( token == '(' ){
          match('(');
          exp();
          match(')');
     }else if( token == number ){
          match(number);
```

```
    }else Error();
}
int main(){
    getToken();
    exp();
    return 0;
}
```

写出递归构建语法树的程序：

```
void match( expectedToken ){
    if( token == expectedToken )       getToken();
    else Error();
}

BtreeNode * exp(){
    BtreeNode * Node, * tempNode;
    Node = term();
    while( token == '+' || token == '-' ){
        tempNode = new BtreeNode;
        tempNode->data = token;
        match(token);
        tempNode->lchild = Node;
        tempNode->rchild = term();
        Node = tempNode;
    }
    return Node;
}

BtreeNode * term(){
    BtreeNode * Node, * tempNode;
    Node = factor();
    while( token == '*' || token == '/' ){
        tempNode = new BtreeNode;
        tempNode->data = token;
        match(token);
        tempNode->lchild = Node;
        tempNode->rchild = factor();
        Node = tempNode;
    }
    return Node;
}

BtreeNode * factor(){
    BtreeNode * Node;
```

```
    if( token == '(' ){
        match('(');
        Node = exp();
        match(')');
    }else if( token == number ){
        Node = new BtreeNode;
        Node->data = token;
        match(number);
        Node->lchild = Node->rchild = NULL;
    }else Error();
    return Node;
}

int main(){
    BtreeNode *root;
    getToken();
    root = exp();
    return 0;
}
```

例 4.3 消除下面文法的左递归

A → Ba | Aa | c

B → Bb | Ab | d

解：

先消除 A 的直接左递归：

A → Ba{a} | c{a}

再将其代入到 B 的文法表达式中：

B → Bb | Ba{a}b | c{a}b | d

再消除 B 的直接左递归：

B → (d|c{a}b){(b|a{a}b)}

例 4.9 考虑简单整型算术表达式的文法：

exp → exp addop term | term

addop → + | -

term → term mulop factor | factor

mulop → * | /

factor → (exp) | number

试求：

First(exp) = ?

First(term) = ?

First(factor) = ?

解：先消除左递归：

exp → term{ addop term }

addop → + | -

term → factor { mulop factor }

mulop → * | /

factor → (exp) | number

由此得 First(exp) = { (, number }

First(term) = { (, number }

First(factor) = { (, number }

例 4.10 考虑 if 语句的文法：

statement → if-stmt | other

if-stmt → if (exp) statement else-part

else-part → else statement | ε

exp → 0 | 1

试求：

Follow(statement) = ?

Follow(if-stmt) = ?

Follow(else-part) = ?

Follow(exp) = ?

解：

Follow(statement) = { else, $ }

Follow(if-stmt) = { else, $ }

Follow(else-part) = { else, $ }

Follow(exp) = {) }

LL(1)分析法：

问题 1：

G[S] = {

S → Ab | Bc

A → aA | dB

B → c | e

}

画出 LL(1)分析表和当匹配串 adcb 时的分析栈。

解：分析表为

	a	b	c	d	e		
S	S → Ab		S → Bc	S → Ab	S → Bc		
A	A → aA			A → dB			

B			B → c		B → e		

分析栈为：

步骤	符号栈	输入串	动作
1	S	adcb	S → Ab
2	bA	adcb	A → aA
3	bAa	adcb	匹配
4	bA	dcb	A → dB
5	bBd	dcb	匹配
6	bB	cb	B → c
7	bc	cb	匹配
8	b	b	匹配
9			成功

问题 2：

G[S] = {

S → AbB | Bc

A → aA | ε

B → d | e

}

画出 LL(1)分析表和当匹配串 abd 时的分析栈。

解：分析表为：

	a	b	c	d	e
S	S → AbB	S → AbB		S → Bc	S → Bc
A	A → aA	A → ε			
B				B → d	B → e

分析栈为：

步骤	符号栈	输入串	动作
1	S	abd	S → AbB
2	BbA	abd	A → aA
3	BbAa	abd	匹配
4	BbA	bd	A → ε
5	Bb	bd	匹配
6	B	d	B → d
7	d	d	匹配
8			成功

LR(0)分析法：

问题 1：

G[A]：

A → (A) | a

画出 DFA，并写出串((a))的分析过程。

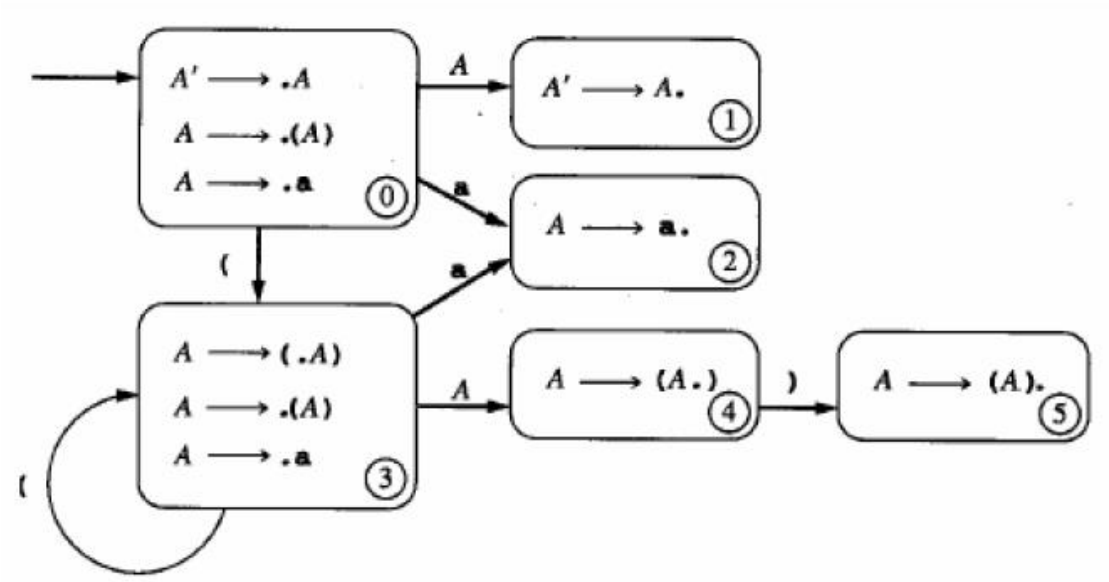
解：先扩充文法，得到：

A' → .A

A → .(A)

A → .a

DFA 为：



分析表为：

状态	动作	规则	输入			Goto
			(	a	)	A
0	移进		3	2		1
1	规约	A' → A				
2	规约	A → a				
3	移进		3	2		4
4	移进				5	
5	规约	A → (A)				

分析串((a))时的分析栈为：

步骤	分析栈	输入	动作
1	$0	((a))$	移进
2	$0(3	(a))$	移进
3	$0(3(3	a))$	移进
4	$0(3(3a2	))$	用A → a规约
5	$0(3(3A4	))$	移进
6	$0(3(3A4)5	)$	用A → (A) 规约
7	$0(3A4	)$	移
8	$0(3A4)5	$	用A → (A) 规约
9	$0A1	$	接受

SLR(1)分析：

问题1：

文法： E → E + n | n

画出该文法的SLR(1)的DFA图、分析表和分析串n+n+n时的分析栈。

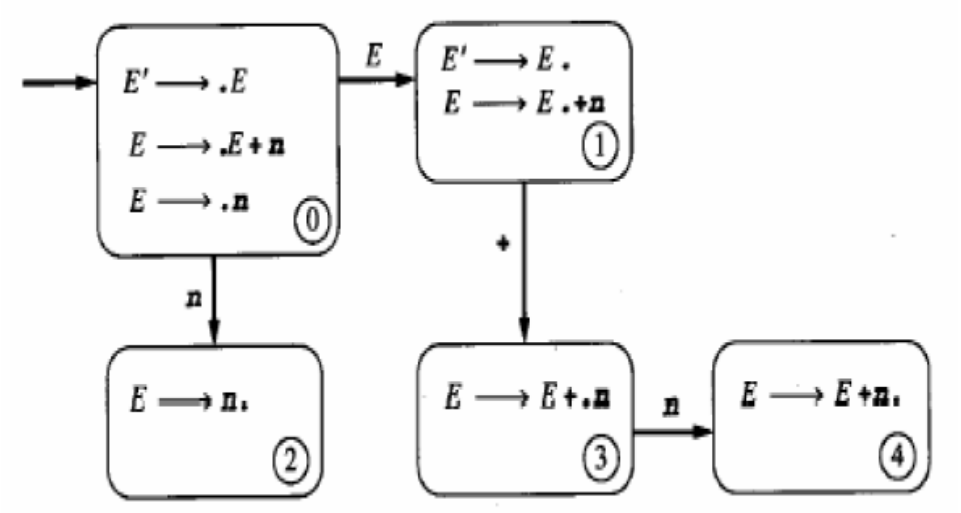
解：先扩充文法，得到

E' → E

E → E + n

E → n

画出其 DFA 为：



其分析表如下：

状态	输入			Goto
	n	+	$	E
0	s2			1
1		s3	接受	
2		r(E → n)	r(E → n)	
3	s4			
4		r(E → E + n)	r(E → E + n)	

分析串 n+n+n 时的分析栈为：

步骤	分析栈	输入	动作
1	$0	n+n+n$	移进 2
2	$0n2	+n+n$	用 E → n 规约
3	$0E1	+n+n$	移进 3
4	$0E1+3	n+n$	移进 4
5	$0E1+3n$	+n$	用 E → E + n 规约
6	$0E1	+n$	移进 3

7	$0E1+3	n$	移进 4
8	$0E1+3n4	$	用 E → E + n 规约
9	$0E1	$	接受

LR(1)分析：

问题 1：

文法： A → (A) | a

画出该文法的 LR(1)的 DFA 图和分析表。

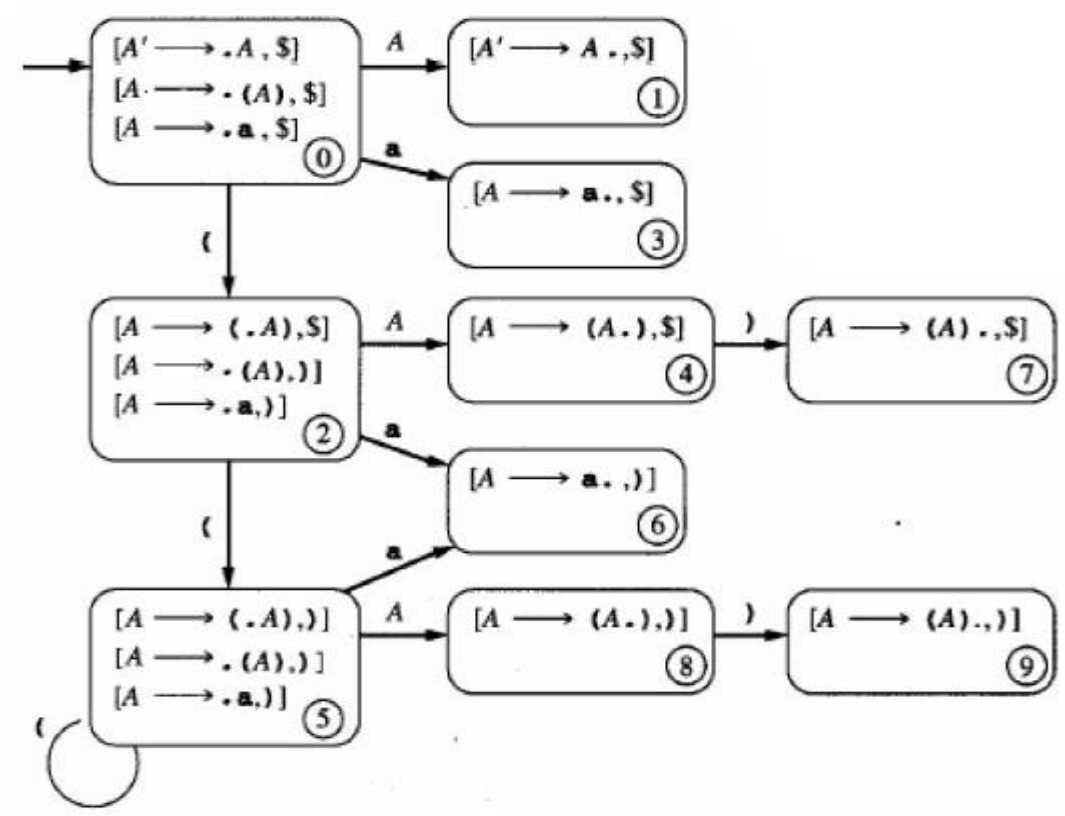
解：先扩充文法，得到：

A' → A

A → (A)

A → a

其 DFA 为：



分析表为：

状态	输入				Goto
	(	a	)	$	A
0	s2	s3			1

1				接受	
2	s5	s6			4
3				r(A → a)	
4			s7		
5	s5	s6			8
6			r(A → a)		
7				r(A → (A))	
8			s9		
9			r(A → (a))		

问题 2：

文法：

S → id | V := E

V → id

E → V | n

画出该文法的 LR(1)的 DFA 图和分析表。

解：首先将文法扩展，得到：

S' → S

S → id

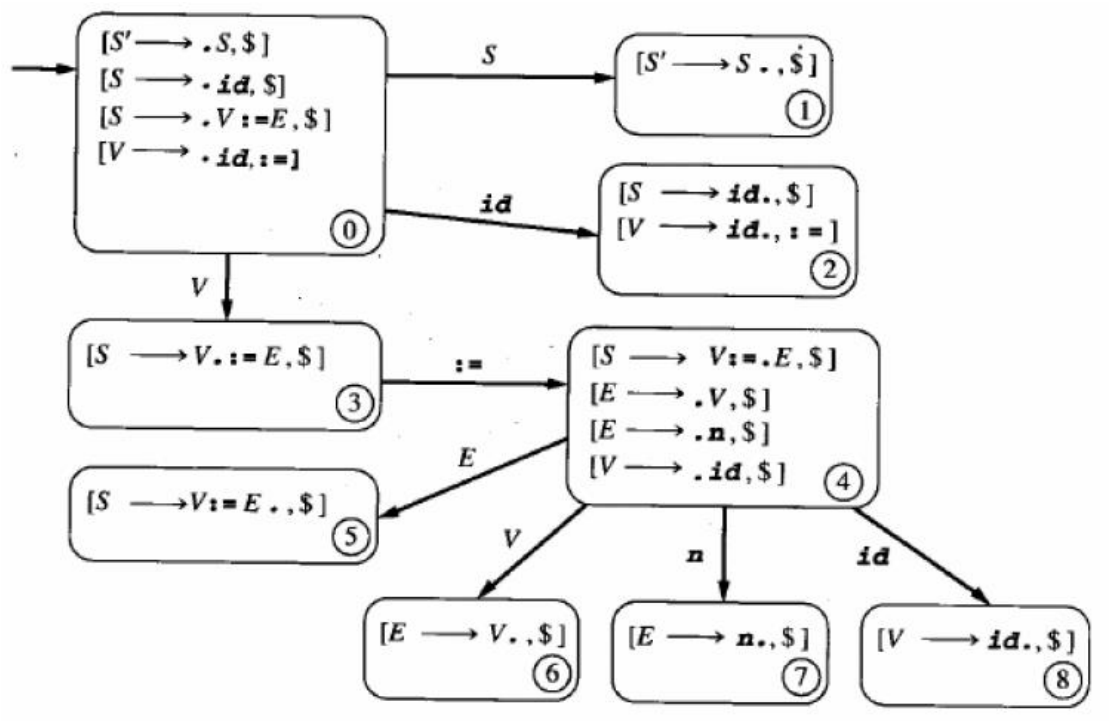
S → V := E

V → id

E → V

E → n

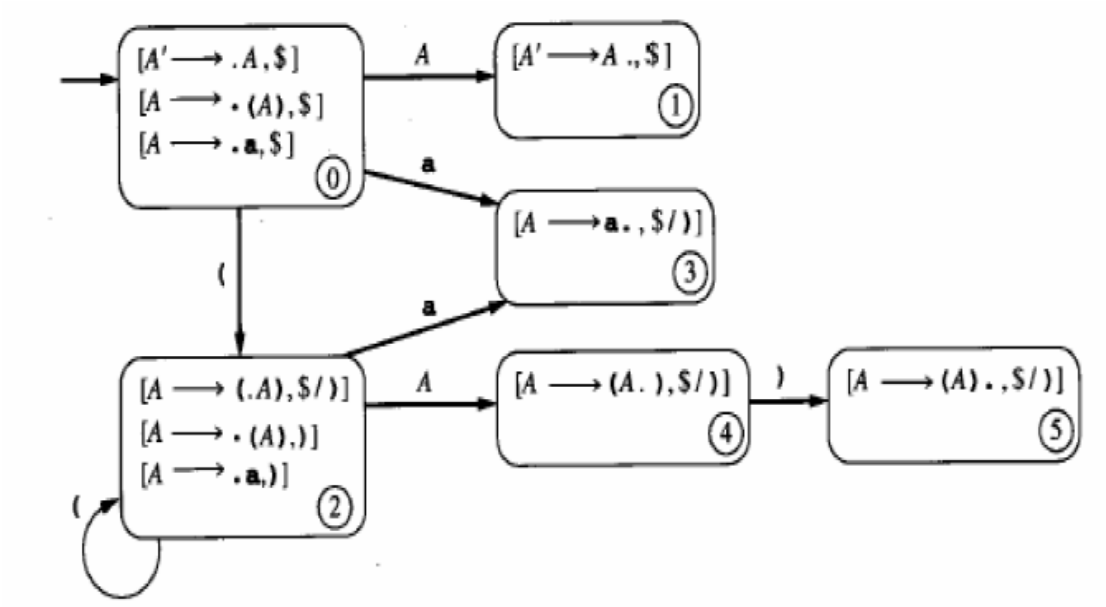
DFA 图如下：



分析表如下：

状态	输入				Goto		
	id	:=	n	$	V	E	S
0	s2				3		1
1				接受			
2		r(V → id)		r(S → id)			
3		s4					
4	s8		s7		6	5	
5				r(S→V:=E)			
6				r(E → V)			
7			r(E → n)				
8				r(V → id)			

将上题的 DFA 图改写成符合 **LALR(1)** 的 DFA：



后缀表示：

写出 `if( m>n ) k=1; else m=0;` 的后缀表示。

解：

```
m n > 10 BZ k 1 = 20 BR
10: m 0 =
20:
```

三元组表示：

写出表达式 `a*( b+c )`对应的三元组。

解：

```
(1)  ( +, b, c )
(2)  ( *, a, (1) )
```

四元组表示：

例子 1

```
if( m>n ) { k=0; S=1; } else m=0;
```

将该语句转换为四元组表示。

解：

```
(1)  (J>,m,n,3)
(2)  (J, , ,6)
(3)  (=,0, ,k)
(4)  (=,1, ,S)
(5)  (J, , ,7)
(6)  (=,0, ,m)
(7)
```

例题 2

```
if( A<B && C>D ) x=1; else x=0;
```

将该语句转换为四元组表示。

解：

```
(1)  (J<,A,B,3)
(2)  (J, , ,7)
(3)  (J>,C,D,5)
(4)  (J, , ,7)
(5)  (=,1, ,x)
(6)  (J, , ,8)
(7)  (=,0, ,x)
(8)
```

例题 3

```
while( m>n ) { k=1; S=0; }
```

将该语句转换为四元组表示。

解：

```
（1）(J>,m,n,3)
（2）(J, , ,6)
（3）(=,1, ,k)
（4）(=,0, ,S)
（5）(J, , ,1)
（6）
```

例题 4（08 级的大三第二学期考这道）

```
if( A && B && C > D )
    if( X<Y ) F=1; else F=0;
else G=1;
```

将该语句转换为四元组表示。

解：

```
(1)    (JNZ,A, ,3)
(2)    (J, , ,13)
(3)    (JNZ,B, 5)
(4)    (J, , ,13)
(5)    (J>,C,D,7)
(6)    (J, , ,13)
(7)    (J<,X,Y,9)
(8)    (J, , ,11)
(9)    (=,1, ,F)
(10) (J, , ,14)
```

(11) (=,0, ,F)

(12) (J, , ,14)

(13) (=,G, ,1)

(14)

例题 5

```
while ( A && B && C>D ) x=1;
```

将该语句转换为四元组表示。

解：

(1) (JNZ,A, ,3)

(2) (J, , ,9)

(3) (JNZ,B, ,5)

(4) (J, , ,9)

(5) (J>,C,D,7)

(6) (J, , ,9)

(7) (=,1, ,x)

(8) (J, , ,1)

(9)