习题一

1、画出编译程序的总体结构图，简述其部分的主要功能。

［答案］

编译程序的总框图见下图。

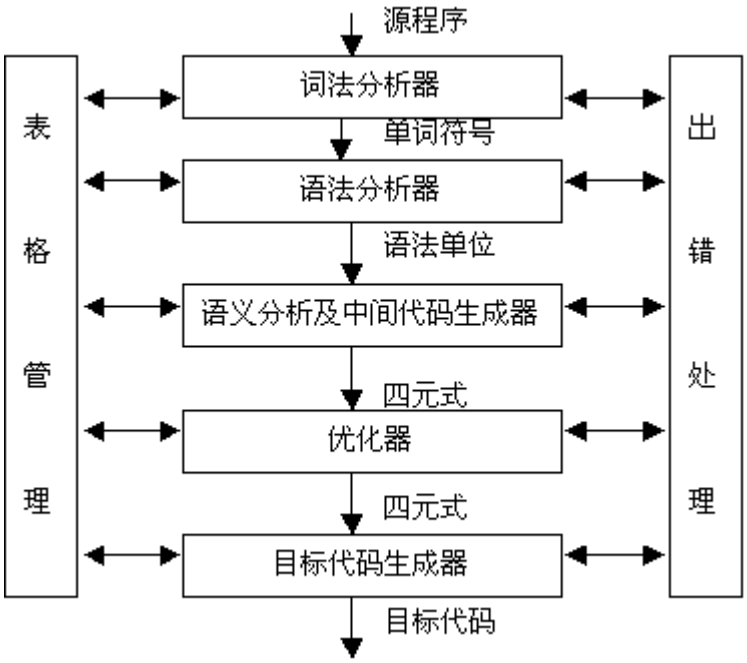


图 编译程序的总体结构图

其中词法分析器，又称扫描器，它接受输入的源程序，对源程序进行词法分析，识别出一个个的单词符号，其输出结果上单词符号。

语法分析器对单词符号串进行语法分析（根据语法规则进行推导或归纳），识别出程序中的各类语法单位，最终判断输入串是否构成语法上正确的“程序”。

语义分析及中间代码产生器，按照语义规则对语法分析器归纳出（或推导出）的语法单位进行语义分析并把它们翻译成一定形式的中间代码。编译程序可以根据不同的需要选择不同的中间代码形式，有的编译程序甚至没有中间代码形式，而直接生成目标代码。

优化器对中间代码进行优化处理。一般最初生成的中间代码执行效率都比较低，因此要做中间代码的优化，其过程实际上是对中间代码进行等价替换，使程序在执行时能更快，并占用更小的空间。

目标代码生成器把中间代码翻译成目标程序。中间代码一般是一种与机器无关的表示形式，只有把它再翻译成与机器硬件相关的机器能识别的语言，即目标程序，才能在机器上运行。

表格管理模块保持一系列的表格，登记源程序的各类信息和编译各阶段的进展状况。编译程序各个阶段所产生的中间结果都记录在表格中，所需要的信息也大多从表格中，所需要的信息也大多从表格中获取，整个编译过程都在不断地和表格打交道。

出错处理程序对出现在源程序中的错误进行处理。如果源程序有错误，编译程序应设法发现错误，把有关错误信息报告给用户。编译程序的各个阶段都有可能发现错误，出错处理程序要对发现的错误进行处理、记录，并反映给用户。

2、计算机执行用高级语言编写的程序有哪些途径?它们之间的主要区别是什么?

[答案]

计算机执行用高级语言编写的程序主要途径有两种，即解释与编译。

像 Basic 之类的语言，属于解释型的高级语言。它们的特点是计算机并不事先对高级语言进行全盘翻译，将其变为机器代码，而是每读入一条高级语句，就用解释器将其翻译为一条机器代码，予以执行，然后再读入下一条高级语句，翻译为机器代码，再执行，如此反复。

总而言之，是边翻译边执行。

像 C，Pascal 之类的语言，属于编译型的高级语言。它们的特点是计算机事先对高级语言进行全盘翻译，将其全部变为机器代码，再统一执行，即先翻译，后执行。从速度上看，编译型的高级语言比解释型的高级语言更快。

习题二

1.文法 G[S]为：

S->Ac|aB

A->ab

B->bc

写出 L(G[S])的全部元素。

[答案]

S=>Ac=>abc

或 S=>aB=>abc

所以 L(G[S])={abc}

2. 文法 G[N]为：

N->D|ND

D->0|1|2|3|4|5|6|7|8|9

G[N]的语言是什么？

[答案]

G[N]的语言是 V^+。V={0,1,2,3,4,5,6,7,8,9}

N=>ND=>NDD.... =>NDDDD...D=>D.....D

3.已知文法 G[S]：

S→dAB　A→aA|a　B→ ε |bB

问：相应的正规式是什么？G[S]能否改写成为等价的正规文法？

[答案]

正规式是 daa*b*;

相应的正规文法为(由自动机化简来)：

G[S]:S→dA　A→a|aB　B→aB|a|b|bC　C→bC|b

也可为(观察得来):G[S]:S→dA　A→a|aA|aB　B→bB| ε

4.已知文法 G[Z]：

Z->aZb|ab

写出 L(G[Z])的全部元素。

[答案]

Z=>aZb=>aaZbb=>aaa..Z...bbb=> aaa..ab...bbb

L(G[Z])={a^nb^n|n>=1}

5.给出语言{$a^nb^nc^m$|n>=1,m>=0}的上下文无关文法。

[分析]

本题难度不大，主要是考上下文无关文法的基本概念。上下文无关文法的基本定义是：A->β,A∈Vn，β∈（Vn∪Vt）*，注意关键问题是保证 a^nb^n 的成立，即“a 与 b 的个数要相等”，为此，可以用一条形如 A->aAb|ab 的产生式即可解决。

[答案]

构造上下文无关文法如下：

S->AB|A

A->aAb|ab

B->Bc|c

[扩展]

凡是诸如此类的题都应按此思路进行，本题可做为一个基本代表。基本思路是这样的：

要求符合 $a^nb^nc^m$，因为 a 与 b 要求个数相等，所以把它们应看作一个整体单元进行，而 c^m 做为另一个单位，初步产生式就应写为 S->AB，其中 A 推出 a^nb^n，B 推出 c^m。因为 m 可为 0，故上式进一步改写为 S->AB|A。接下来考虑 A,凡是要求两个终结符个数相等的问题，都应写为 A->aAb|ab 形式，对于 B 就很容易写成 B->Bc|c 了。

6 .写一文法，使其语言是偶正整数集合。

要求：

(1)允许 0 开头；

(2)不允许 0 开头。

[答案]

(1)允许 0 开头的偶正整数集合的文法

E->NT|G|SFM

T->NT|G

N->D|1|3|5|7|9

D->0|G

G->2|4|6|8

S->NS| ε

F->1|3|5|7|9|G

M->MO|O

(2)不允许0开头的偶正整数集合的文法

E->NT|D

T->FT|G

N->D|1|3|5|7|9

D->2|4|6|8

F->N|O

G->D|O

7.已知文法G：

E->E+T|E-T|T

T->T*F|T/F|F

F->(E)|i

试给出下述表达式的推导及语法树

(1)i;　(2)i*i+i　　(3)i+i*i　(4)i+(i+i)

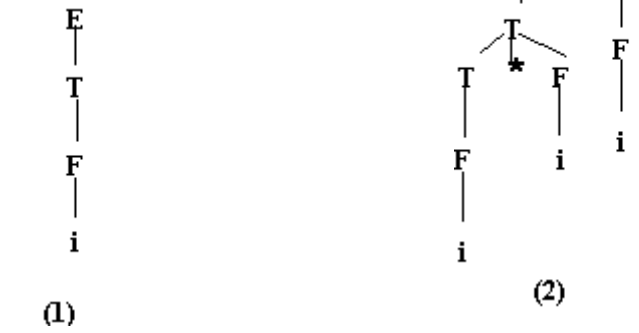
[答案]

(1)E=>T=>F=>i

(2)E=>E+T=>T+T=>T*F+T=>F*F+T=>i*F+T=>i*i+T=>i*i+F=>i*i+i

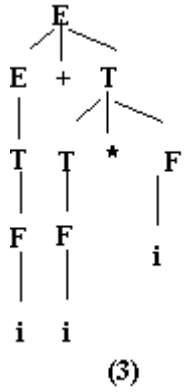
(3)E=>E+T=>T+T=>F+T=>i+T=>i+T*F=>i+F*F=>i+i*F=>i+i*i

(4)E=>E+T=>T+T=>F+T=>i+T=>i+F=>i+(E)=>i+(E+T)=>i+(T+T)=>i+(F+T)
=>i+(i+T)=>i+(i+F)=>i+(i+i)

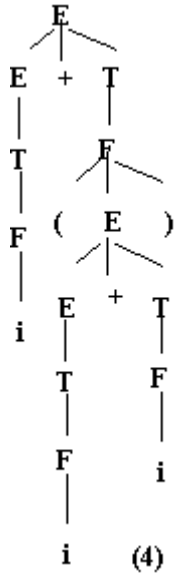


(1)

(2)



(3)



(4)

8 .为句子 i+i*i 构造两棵语法树，从而证明下述文法 G[<表达式>]是二义的。

〈表达式〉->〈表达式〉〈运算符〉〈表达式〉|(〈表达式〉)|i

〈运算符〉->+|-|*|/

[答案]

可为句子 i+i*i 构造两个不同的最右推导:

最右推导 1

〈表达式〉=>〈表达式〉〈运算符〉〈表达式〉

=>〈表达式〉〈运算符〉i

=>〈表达式〉* i

=>〈表达式〉〈运算符〉〈表达式〉＊ i

=>〈表达式〉〈运算符〉i ＊ i

=>〈表达式〉+ i ＊ i

=> i + i ＊ i

最右推导 2

〈表达式〉=>〈表达式〉〈运算符〉〈表达式〉

=>〈表达式〉〈运算符〉〈表达式〉〈运算符〉〈表达式>

=>〈表达式〉〈运算符〉〈表达式〉〈运算符〉 i

=>〈表达式〉〈运算符〉〈表达式〉 ＊ i

=> 〈表达式〉〈运算符〉i ＊ i

=>〈表达式〉+ i ＊ i

=> i + i ＊ i

所以，该文法是二义的。

9. 文法 G[S]为：

S->Ac|aB

A->ab

B->bc

该文法是否为二义的？为什么？

[答案]

对于串 abc

(1)S=>Ac=>abc

(2)S=>aB=>abc

即存在两不同的最右推导

所以，该文法是二义的。

10.考虑下面上下文无关文法：

S->SS*|SS+|a

(1)表明通过此文法如何生成串 aa+a*，并为该串构造语法树。

(2) G[S]的语言是什么？

[答案]

(1)此文法生成串 aa+a*的最右推导如下

S=>SS*=>SS*=>Sa*=>SS+a*=>Sa+a*=>aa+a*

(2)该文法生成的语言是即加法和乘法的逆波兰式,

11. 令文法 G[E]为:

E->E+T|E-T

T->T*F|T/F|F

F->(E)|I

证明 E+T*F 是它的一个句型，指出这个句型的所有短语、直接短语和句柄。

[答案]

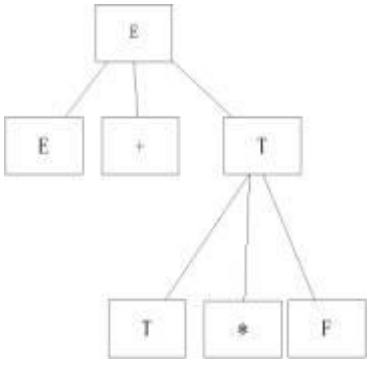
此句型对应语法树如右，故为此文法一个句型。

或者：因为存在推导序列: E=>E+T=>E+T*F，所以 E+T*F 句型

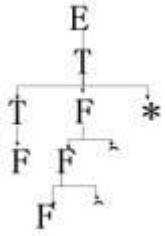
此句型相对于 E 的短语有:E+T*F;相对于 T 的短语有 T*F,

直接短语为：T*F；。

句柄为：T*F



12.已知文法 G[E]:



E→ET+|T　T→TF*｜F　F→F^｜a

试证：FF^^*是文法的句型，指出该句型的短语、简单短语和句柄.

[答案]

该句型对应的语法树如下：

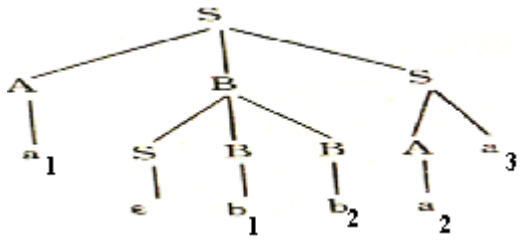
该句型相对于 E 的短语有 FF^^*；相对于 T 的短语有 FF^^*,F;相对于 F 的短语有 F^;F^^;简单短语有 F;F^;句柄为 F.

13.一个上下文无关文法生成句子 abbaa 的推导树如下：

(1)给出串 abbaa 最左推导、最右推导。

(2)该文法的产生式集合 P 可能有哪些元素？

(3)找出该句子的所有短语、直接短语、句柄。



[答案]

(1)串 abbaa 最左推导:

S=>ABS=>aBS=>aSBBS=>a ε BBS=>a ε bBS=>a ε bbS=>a ε bbAa=>a ε bbaa

最右推导：

S=>ABS=>ABAa=>ABaa=>ASBBaa=>ASBbaa=>ASbbaa=>A ε bbaa=>a ε bbaa

(2)产生式有：S→ABS |Aa| ε

A→a

B→SBB|b

(3)该句子的短语有 $a_1b_1b_2a_2a_3$、a_1、b_1、b_2、b_1b_2、a_2a_3、a_2;

直接短语有 a_1、b_1、b_2、a_2;

句柄是 a_1。

14.给出生成下列语言的上下文无关文法。

（1）{ $a^nb^na^mb^m$ |n，m>=0}　　(2) { 1^n0^m 1^m0^n| n，m>=0}

（3）{WaW^r|W 属于{0|a}*，W^r 表示 W 的逆}

[答案]

（1）{$a^nb^na^mb^m$| n，m>=0}

S->AA

A->aAb| ε

(2) { 1^n0^m 1^m0^n| n，m>=0}

S->1S0|A

A->0A1| ε

（3）{WaW^r|W 属于{0|a}*，W^r 表示 W 的逆}

S->0S0|1S1| ε

15 .给出生成下列语言的三型文法。

(1){ a^n|n >=0 }

(2){ a^nb^m|n,m>=1 }

(3) $\{a^n b^m c^k | n,m,k>=0\}$

[答案]

(1) $\{a^n | n>=0\}$的三型文法为：

S->aS| ε

(2)$\{a^n b^m | n,m>=1\}$的三型文法为：

S->aA

A->aA|bB

B->bB| ε

(3)$\{a^n b^m c^k | n,m,k>=0\}$的三型文法为:

A->aA|bB|cC| ε

B->bB|cC| ε

C->cC| ε

16.构造一文法产生任意长的 a,b 串，使得

|a|<=|b|<=2|a|

其中，“|a|”表示 a 字符的个数;“|b|”表示 b 字符的个数。

[分析]

b 的个数在 a 与 2a 之间，所以应想到形如 aSBS 和 BSaS 的形式，B 为 1 到 2 个 b，即可满足条件。

[答案]

如分析中所述，可得文法如下：

S-aSBS|BSaS| ε

B->bb|b

第 1 个产生式为递归定义，由于在第 2 个产生式中 B 被定义为 1 或 2 个 b，所以第 1 个产生式可以保证 b 的个数在|a|与 2|a|之间，而 a 与 b 的位置可以任意排布，所以此文法即为所求，注意第 1 个产生式中要包括 s。

17.下面的文法产生 a 的个数和 b 的个数相等的非空 a,b 串

S->aB|bA

B->bS|aBB|b

A->aS|bAA|a

其中非终结符 B 推出 b 比 a 的个数多 1 个的串，A 则反之。

说明该文法是二义的。

对上述文法略作修改，使之非二义，并产生同样的语言。（略做修改的含义是：不增加非终结符。）

[答案]

句子 aabbab 有两种不同的推导。

S=>aB=>aaBB=>aabB=>aabbS=>aabbaB=>aabbab

S=>aB=>aaBB=>aabSB=>aabbAB=>aabbaB=>aabbab

即它可以产生两棵不同的语法树，故它是二义的。

修改后的无二义文法如下：

S->aBS|bAS|aB|bA

B->aBB|b

A->bAA|a

18.给出 0，1，2，3 型文法的定义。

[答案]

乔姆斯基(chomsky)把文法分成类型，即 0 型，1 型，2 型和 3 型，0 型强于 1 型，1 型强于 2 型，2 型强于 3 型。

如果它的每个产生式 α ->β 的结构是 α ∈（VnUVt）*且至少含有一个非终结符,而 β ∈（VnUVt）*，我们说 G=（Vt，V_N，S，δ）是一个 0 型文法。

0 型文法也称短语文法。一个非常重要的理论结果是，0 型文法的能力相当于图灵(Tunring)机。或者说，任何 0 型语言都是递归可枚举的；反之，递归可枚举集必定是一个 0 型语言。

如果把 0 型文法分别加上以下的第 i 条限制，则我们就得 i 型文法为：

1．G 的任何产生式 α ->β 均满足|α|<=|β|；仅仅 S->ε 例外，但 S 不得出现在任何产生式的右部。

2．G 的任何产生式为 A->β，A∈Vn，β∈（VnUVt）*

3．G 的任何产生式为 A->aB 或 A->a，其中 A，B∈Vn

1 型文法也称上下文有关文法。这种文法意味着，对非终结符进行替换时务必考虑上下文，而且，一般不允许替换成空串。

2 型文法对非终结符进行替换时无须考虑上下文，

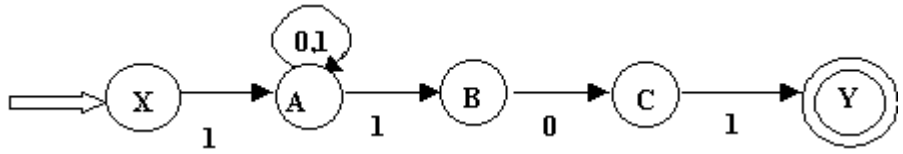
3 型文法也称线性文法

习题三

1.构造正规式 1(0|1)*101 相应的 DFA.

［答案］

先构造 NFA



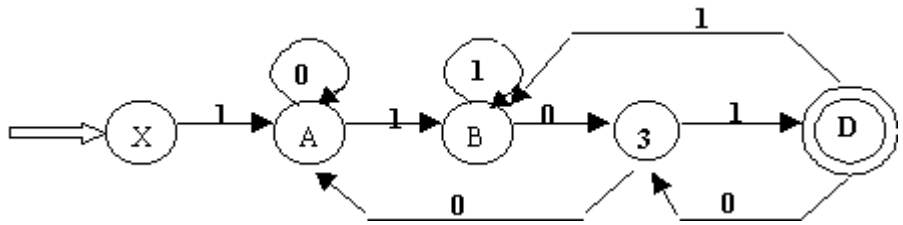
确定化

	0	1
X		A
A	A	AB
AB	AC	AB
AC	A	ABY
ABY	AC	AB

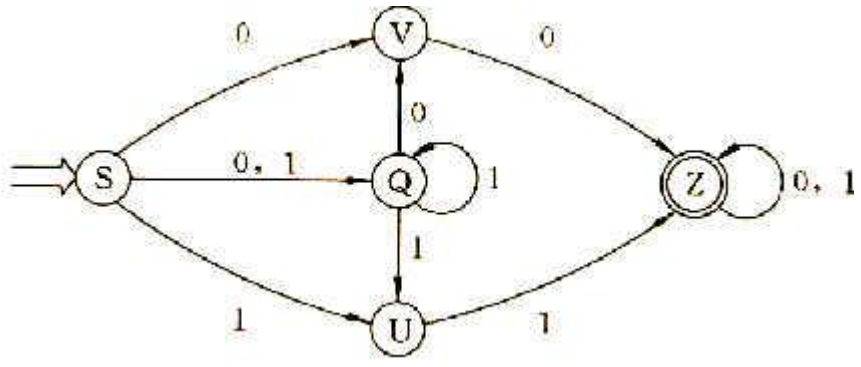
重新命名，令 AB 为 B、AC 为 C、ABY 为 D

	0	1
X		A
A	A	B
B	C	B
C	A	D
D	C	B

DFA：



2.将下图确定化：



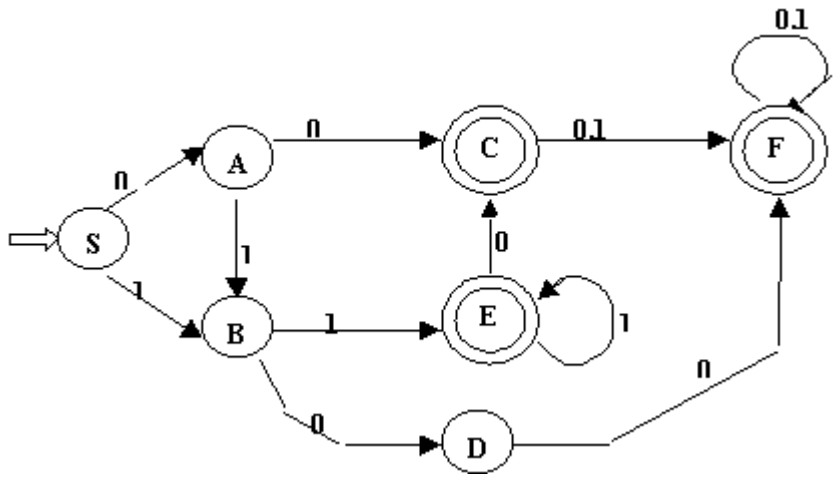
［答案］

	0	1
S	VQ	QU
VQ	VZ	QU
QU	V	QUZ
VZ	Z	Z
V	Z	
QUZ	VZ	QUZ
Z	Z	Z

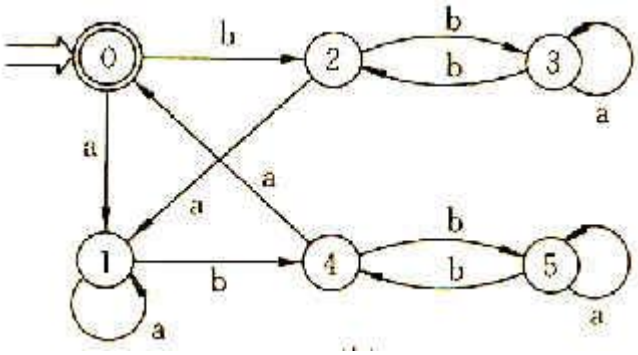
重新命名，令 VQ 为 A、QU 为 B、VZ 为 C、V 为 D、QUZ 为 E、Z 为 F。

	0	1
S	A	B
A	C	B
B	D	E
C	F	F
D	F	
E	C	E
F	F	F

DFA：



3. 把下图最小化：



[答案]

初始分划得Π_0：终态组{0}，非终态组{1,2,3,4,5}

对非终态组进行审查：

$\{1,2,3,4,5\}_a \subset \{0,1,3,5\}$

而{0,1,3,5}既不属于{0}，也不属于{1,2,3,4,5}

$\because \{4\}_a \subset \{0\}$，所以得新分划

Π_1：{0}，{4}，{1,2,3,5}

对{1,2,3,5}进行审查：

$\because \{1,5\}_b \subset \{4\}$

$\{2,3\}_b \subset \{1,2,3,5\}$，故得新分划

Π_2：{0}，{4}，{1，5}，{2,3}

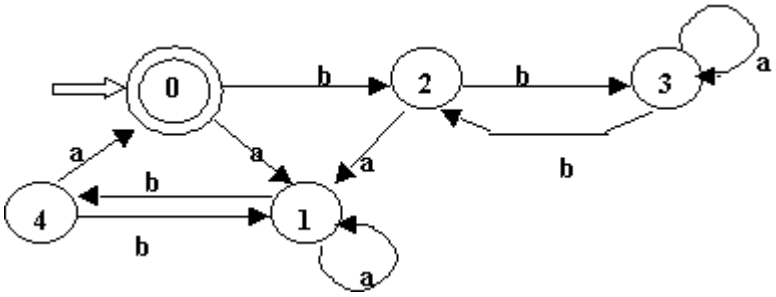
$\{1, 5\}_a \subset \{1, 5\}$

$\{2,3\}_a \subset \{1,3\}$，故状态 2 和状态 3 不等价，得新分划

Π_3：{0}，{2}，{3}，{4}，{1，5}

这是最后分划了

最小 DFA：

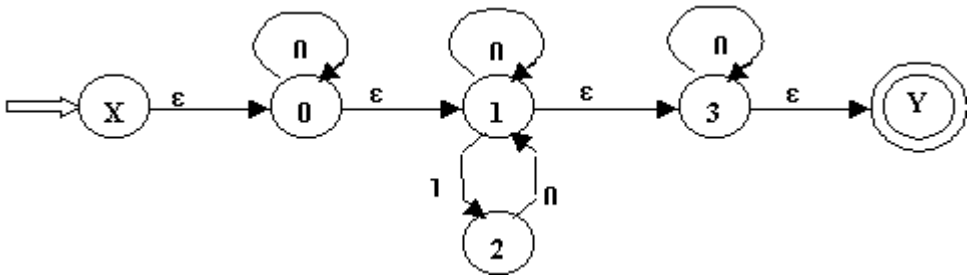


4.构造一个 DFA，它接收 Σ={0,1}上所有满足如下条件的字符串：每个 1 都有 0 直接跟在右边。并给出该语言的正规式和正规文法。

[答案]

按题意相应的正规表达式是 0*(0 | 10)*0*或 0*(100*)*0*

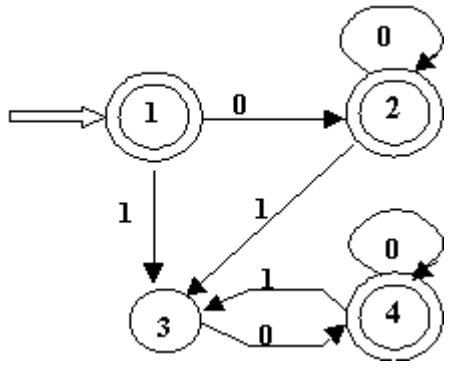
构造相应的 DFA，首先构造 NFA 为



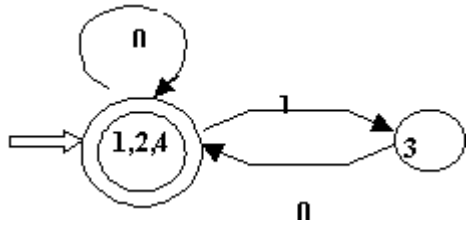
用子集法确定化

I	I_0	I_1	S	0	1
{X,0,1,3,Y}	{0,1,3,Y}	{2}	1	2	3
{0,1,3,Y}	{0,1,3,Y}	{2}	2	2	3
{2}	{1,3,Y}	/	3	4	
{1,3,Y}	{1,3,Y}	{2}	4	4	3

DFA：



可最小化，终态组为{1,2,4}，非终态组为{3}，$\{1,2,4\}_0 \subset \{1,2,4\}$，$\{1,2,4\}_1 \subset \{3\}$，所以1,2,4 为等价状态，可合并。



5．给出下列文法所对应的正规式：

S->0A|1B

A->1S|1

B->0S|0

[答案]

解方程组 S 的解：

S=0A|1B

A=1S|1

B=0S|0

将 A、B 产生式的右部代入 S 中

S=01S|01|10S|10=（01|10）S|（01|10）

所以：S= (01|10)*(01|10)

6.为下边所描述的串写正规式，字母表是 {a,b}.

a) 以 ab 结尾的所有串

b) 包含偶数个 b 但不含 a 的所有串

c) 包含偶数个 b 且含任意数目 a 的所有串

d) 只包含一个 a 的所有串

e) 包含 ab 子串的所有串

f) 不包含 ab 子串的所有串

[答案]

注意 正规式不唯一

a) (a|b)*ab

b) (bb)*

c) (a*ba*ba*)*

d) b*ab*

e) (a|b)*ab(a|b)*

f) b*a*

7.请描述下面正规式定义的串. 字母表Σ = {0，1}.

a) $0*(10^{+})*0*$

b) (0|1)*(00|11) (0|1)*

c) 1(0|1)*0

[答案]

a) 每个 1 至少有一个 0 跟在后边的串

b）所有含两个相继的 0 或两个相继的 1 的串

c）必须以 1 开头和 0 结尾的串

8. 构造有穷自动机.

a）构造一个 DFA，接受字母表Σ = {0, 1}上的以 01 结尾的所有串

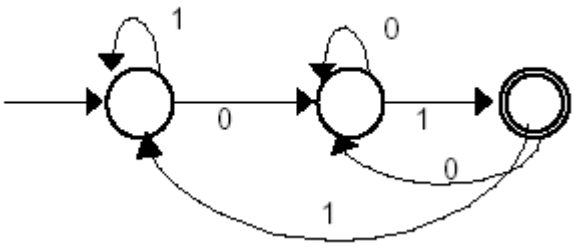
b）构造一个 DFA，接受字母表Σ = {0, 1}上的不包含 01 子串的所有串.

c）构造一个 NFA，接受字母表Σ = {x,y}上的正规式 x(x|y)*x 描述的集合

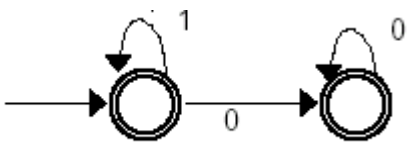
d）构造一个 NFA，接受字母表Σ = {a, b}上的正规式(ab|a)*b+描述的集合并将其转换为等价的 DFA.以及最小状态 DFA

[答案]

a)

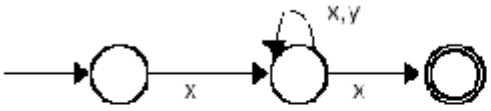


b)

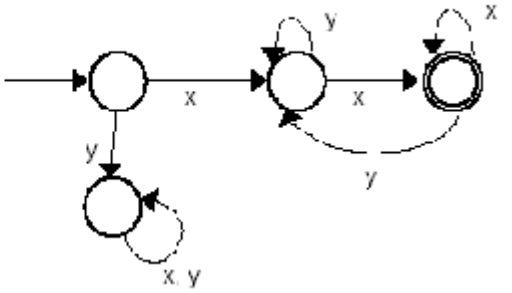


c)

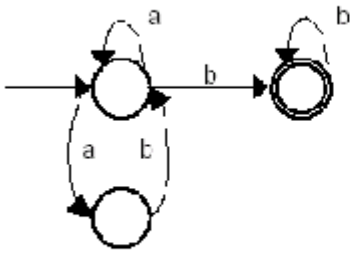
NFA:



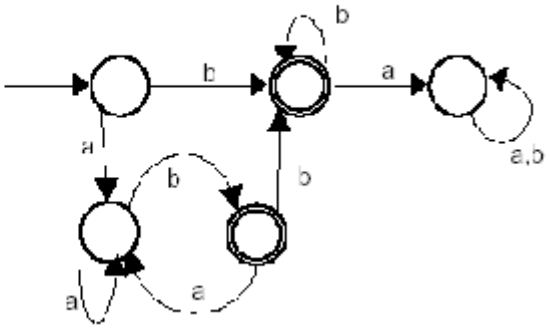
DFA



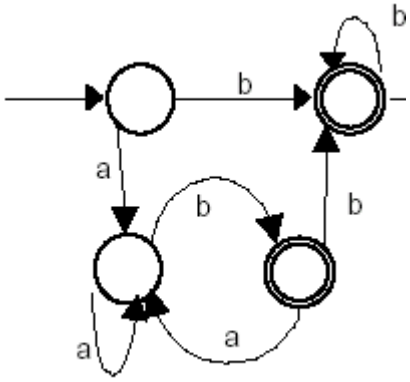
d) NFA:



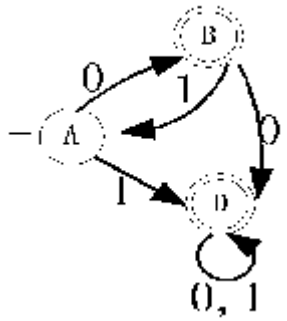
DFA



最小化的 DFA



9. 设有如图所示状态转换图，求其对应的正规表达式。



[答案]

可通过消结法得出正规式

R=(01)*((00|1)(0|1)*|0)

也可通过转换为正则文法，解方程得到正规式。

10. 已知正规式：

(1)((a|b)*|aa)*b;

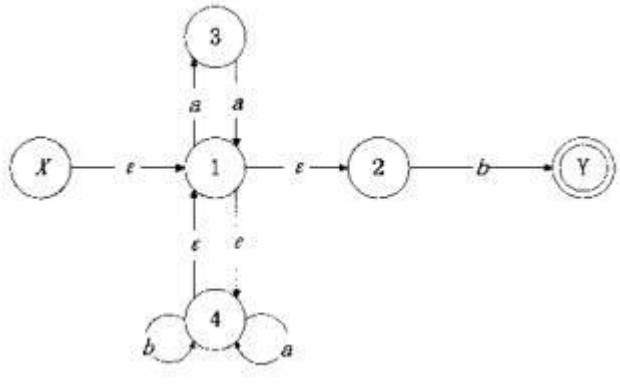
(2)(a|b)*b.

试用有限自动机的等价性证明正规式(1)和(2)是等价的，并给出相应的正规文法。

[分析]

基本思路是对两个正规式，分别经过确定化、最小化、化简为两个最小 DFA，如这两个最小 DFA 一样，也就证明了这两个正规式是等价的。

[答案]



[答案

状态转换表 1

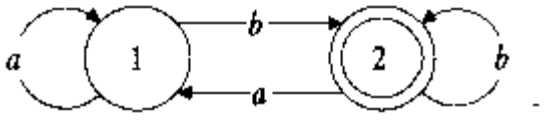
	a	b
X124	1234	124Y
1234	1234	124Y
124Y	1234	124Y

状态转换表 2

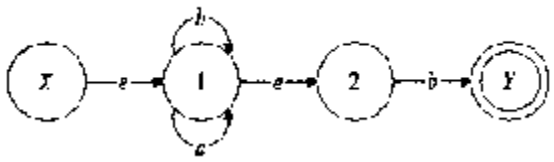
	a	B
1	2	3
2	2	3
3	2	3

由于 2 与 3 完全一样，将两者合并，即见下表

	a	b
1	2	3
2	2	3



而对正规式(2)可画 NFA 图，如图所示。

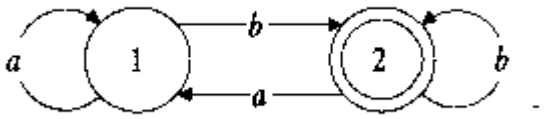


	a	b
X12	12	12Y
12	12	12Y
12Y	12	12Y

可化简得下表

	a	b
1	2	3
2	2	3

得 DFA 图



两图完全一样，故两个自动机完全一样，所以两个正规文法等价。

对相应正规文法,令 A 对应 1，B 对应 2

故为：

A->aA|bB|b

B->aA|bB|b

即为 S->aS|bS|B,此即为所求正规文法

习题四

1.文法

S->a|^|(T)

T->T,S|S

(1) 对(a,(a,a)和(((a),^,(a)),a)的最左推导。

(2)对文法 G 改写，然后对每个非终结符写出不带回溯的递归子程序。

(3)经改写后的文法是否为 LL（1）的？给出它的预测分析表。

(4)给出输入串(a,a)#的分析过程，并说明该串是否为 G 的句子。

[答案]

(1) 对(a,(a,a)的最左推导为：

S=>(T)

=>(T,S)

=>(S,S)

=>(a,S)

=>(a,(T))

=>(a,(T,S))

=>(a,(S,S))

=>(a,(a,S))

=>(a,(a,a))

对(((a),^,(a)),a) 的最左推导为：

S=>(T)

=>(T,S)

=>(S,S)

=>((T),S)

=>((T,S),S)

=>((T,S,S),S)

=>((S,S,S),S)

=>(((T),S,S),S)

=>(((T,S),S,S),S)

=>(((S,S),S,S),S)

=>(((a,S),S,S),S)

=>(((a,a),S,S),S)

=>(((a,a),^,S),S)

=>(((a,a),^,(T)),S)

=>(((a,a),^,(S)),S)

=>(((a,a),^,(a)),S)

=>(((a,a),^,(a)),a)

(3)改写文法为：

0) S->a

1) S->^

2) S->(T)

3) T->S N2

4) N2->, S N2

5) N2->ε

	FIRST	FOLLOW
S	a ^ (	# ,)
T	a ^ (	)
N2	, ε	)

对左部为 N2 的产生式可知：

FIRST （->, S N2）={，}

FIRST （->ε）={ε}

FOLLOW （N2）={)}

{，}∩ {)}=Ø

所以文法是 LL(1)的。

预测分析表

	a	^	(	)	,	#
S	->a	->^	->(T)			
T	->S N2	->S N2	->S N2			
N2				->ε	->, S N2	

也可由预测分析表中无多重入口判定文法是 LL(1)的。

(4)对输入串（a,a）#的分析过程为：

步骤	状态栈	当前字符	剩余输入串	操作
1	#S	(	a,a)#	S->(T)
2	#)T(	(	a,a)#	匹配
3	#)T	A	,a)#	T->SN2
4	#)N2S	A	,a)#	S->a
5	#)N2a	A	,a)#	匹配
6	#)N2	,	a)#	N2->,SN2
7	#)N2S,	,	a)#	匹配
8	#)N2S	a	)#	S->a
9	#)N2a	a	)#	匹配
10	#)N2	)	#	N2->ε
11	#)	)	#	匹配
12	#	#		

可见输入串（a,a）#是文法的句子。

2．已知文法 G[A]如下，试用类 C 或类 PASCAL 语言写出其递归下降子程序.(主程序不需写)

G[A]: A→[B

B→X]{A}

X→(a|b){a|b}

[答案]

不妨约定:在进入一个非终结符号相应的子程序前，已读到一个单词.word：存放当前读到的单词，Getsym()为一子程序，每调用一次，完成读取一单词的工作。error()为出错处理程序.FIRST(A)为终结符 A 的 FIRST 集.

类 C 程序如下：

```
void A()
{
   if word=='['
   {
      Getsym();
      B();
      }
   else error();
}
```

```
void B()
{ X();
    if word==']'
  {
       Getsym();
       while(word in FIRST(A))
       A();
    }
    else error();
}
```

```
void X()
{
  if (word= ='a'||word=='b')
  {
    Getsym();
    while(word= ='a'||word=='b')
       Getsym();
    }
  else error();
}
```

3．文法：

S->MH|a

H->LSo|ε

K->dML|ε

L->eHf

M->K|bLM

判断 G 是否为 LL(1)文法，如果是，构造 LL(1)分析表。

[答案]

源文法 G 展开为：

0） S->M H

1） S->a

2） H->L S o

3） H->ε

4） K->d M L

5） K->ε

6） L->e H f

7） M->K

8） M->b L M

	FIRST	FOLLOW
S	{a,d,b,ε,e}	{#,o}
M	{d, ε,b}	{e,#,o}
H	{ε,e}	{#,f,o}
L	{e}	{a,d,b,e,o,#}
K	{d,ε}	{e,#,o}

预测分析表

	a	o	d	e	f	b	#
S	->a	->MH	->MH	->MH		->MH	->MH
M		->K	->K	->K		->bLM	->K
H		->ε		->LSo	->ε		->ε
L				->eHf			
K		->ε	->dML	->ε			->ε

由预测分析表中无多重入口判定文法是 LL(1)的。

4.改写下列文法，并判断是否为 LL(1)文法。

（1） 文法：

A->aABe|a

B->Bb|d

改写文法为：

0) A->a N3

1) N3->A B e

2) N3->ε

3) B->d N2

4) N2->b N2

5) N2->ε

	FIRST	FOLLOW
A	{a}	{#,d}
B	{d}	{e}
N2	{b,ε}	{e}
N3	{ε,a}	{#,d}

预测分析表

	a	e	b	d	#
A	->a N3				
B				->d N2	
N2		->ε	->b N2		
N3	->A B e			->ε	->ε

由预测分析表中无多重入口判定文法是 LL(1)的。

（2）文法：

S->Aa|b

A->SB

B->ab

第 1 种改写：

S->SBa|b

B->ab

0) S->b N2

1) N2->B a N2

2) N2->ε

3) B->a b

	FIRST	FOLLOW
S	{b}	{#}
B	{a}	{a}
N2	{ε,a}	{#}

预测分析表

	a	b	#
S		->bN2	
B	->ab		
N2	->BaN2		->ε

由预测分析表中无多重入口判定文法是 LL(1)的。

第 2 种改写：

S->Aa|b

A->AaB|bB

B->ab

0) S->A a

1) S->b

2) A->b B N3

3) N3->a B N3

4) N3->ε

5) B->a b

	FIRST	FOLLOW
S	{b}	{#}
A	{b}	{a}
B	{a}	{a}
N3	{a,ε}	{a}

预测分析表

	a	b	#
S		->A a	
		->b	
A		->b B N3	
B	->a b		
N3	->a B N3		
	->ε		

由预测分析表中含有多重入口判定文法不是 LL(1)的。

（3） 文法：

A->baB|ε

B->Abb|a

先改写文法为：

0) A->baB

1) A->ε

2) B->baBbb

3) B->bb

4) B->a

再改写文法为：

0) A->baB

1) A->ε

2) B->bN

3) B->a

4) N->aBbb

5) N->b

	FIRST	FOLLOW
A	{b}	{#}
B	{b,a}	{#,b}
N	{b,a}	{#,b }

预测分析表

	a	b	#
A		->baB	->ε
B	->a	->bN	
N	->aBbb	->b	

由预测分析表中无多重入口判定文法是 LL(1)的。

5．设有文法 G[A]的产生式集为:

A→BaC|CbB

B→Ac|c

C→Bb|b

试消除 G[A]的左递归。

[答案]

提示：不妨以 A、B、C 排序.先将 A 代入 B 中，然后消除 B 中左递归；再将 A、B 代入 C 中。再消除 C 中左递归。

最后结果为:G[A]:

A→BaC|CbB

B→CbBcB'|cB'

B'→aCcB'|ε

C→cB'bC'|bC'

C'→bBcB'bC'|ε

习题五

1、对于文法 S → (L) | a

L → L, S | S

(1)给出句子(a, ((a, a), (a, a)))的一个最右推导，并指出右句型的句柄；

(2)按照(1)的最右推导，说明移进－归约分析器的工作步骤。

[答案]

(1)S => (L) => (L, S) => (L, (L)) => (L, (L, S)) => (L, (L, (L)))

=> (L, (L, (L, S)))=> (L, (L, (L, a))) => (L, (L, (S, a)))

=> (L, (L, (a, a))) => (L, (S, (a, a)))=> (L, ((L), (a, a)))

=> (L, ((L, S), (a, a))) => (L, ((L, a), (a, a)))

=> (L, ((S, a), (a, a))) => (L, ((a, a), (a, a)))

=> (S, ((a, a), (a, a))) => (a, ((a, a), (a, a)))

(注：句柄下面有下划线)

(2)

步骤	栈	输 入	动 作
1	#	(a, ((a, a), (a, a)))#	移进
2	#(	a, ((a, a), (a, a)))#	移进
3	#(a	, ((a, a), (a, a)))#	归约，S→a
4	#(S	, ((a, a), (a, a)))#	归约，L→S
5	#(L	, ((a, a), (a, a)))#	移进
6	#(L,	((a, a), (a, a)))#	移进
7	#(L, (	(a, a), (a, a)))#	移进
8	#(L, ((	a, a), (a, a)))#	移进
9	#(L, ((a	, a), (a, a)))#	归约，S→a
10	#(L, ((S	, a), (a, a)))#	归约，L→S
11	#(L, ((L	, a), (a, a)))#	移进
12	#(L, ((L,	a), (a, a)))#	移进
13	#(L, ((L, a	), (a, a)))#	归约，S→a
14	#(L, ((L, S	), (a, a)))#	归约，L→L, S
15	#(L, ((L	), (a, a)))#	移进
16	#(L, ((L)	, (a, a)))#	归约，S→(L)
17	#(L, (S	, (a, a)))#	归约，L→S
18	#(L, (L	, (a, a)))#	移进
19	#(L, (L,	(a, a)))#	移进
20	#(L, (L, (	a, a)))#	移进
21	#(L, (L, (a	, a)))#	归约，S→a
22	#(L, (L, (S	, a)))#	归约，L→S
23	#(L, (L, (L	, a)))#	移进
24	#(L, (L, (L,	a)))#	移进
25	#(L, (L, (L, a	)))#	归约，S→a
26	#(L, (L, (L, S	)))#	归约，L→L, S
27	#(L, (L, (L	)))#	移进
28	#(L, (L, (L)	))#	归约，S→(L)

29	#(L, (L, S	))#	归约，L→L, S
30	#(L, (L	))#	移进
31	#(L, (L)	)#	归约，S→(L)
32	#(L, S	)#	归约，L→L, S
33	#(L	)#	移进
34	#(L)	#	归约，S→(L)
35	#S	#	接受

2、已知文法 G[S]为：

S→a|^|(T)

T→T,S|S

(1)计算 G[S]的 FIRSTVT 和 LASTVT。

(2)构造 G[S]的算符优先关系表并说明 G[S]是否未算符优先文法。

(3)计算 G[S]的优先函数。

(4)给出输入串(a,a)#和(a,(a,a))#的算符优先分析过程。

[答案]

(1)

FIRSTVT 和 LASTVT

	FIRSTVT	LASTVT
S	a、^、(	a、^、)
T	，、a、^、(	，、a、^、)

(2)

算符优先关系

	a	(	)	,	^	#
a			≻	≻		≻
(	≺	≺	≡		≺	
)			≻	≻		≻
,	≺	≺	≻	≻	≺	

^			≻	≻		≻
#	≺	≺			≺	

(3)

对应的算符优先函数为：

	a	(	)	,	^	#
S	2	1	2	2	2	1
T	3	3	1	1	3	1

(4)

句子(a,a)#分析过程如下：

步骤	栈	优先关系	当前符号	剩余输入串	移进或归约
1	#	#≺(	(	a,a)#	移进
2	#(	(≺a	a	,a)#	移进
3	#(a	a≻,	,	a)#	归约
4	#(F	(≺,	,	a)#	移进
5	#(F,	,≺a	A	)#	移进
6	#(F,a	A≻)	)	#	归约
7	#(F,F	,≻)	)	#	归约
8	#(F	(≡)	)	#	移进
9	#(F)	)≻#	#		归约
10	#F	#≡#	#		接受

句子(a, (a, a))分析过程如下：

步骤	栈	优先关系	当前符号	剩余输入串	移进或归约
1	#	#≺(	(	a,(a,a))#	移进
2	#(	(≺a	a	, (a,a))#	移进
3	#(a	a≻,	,	(a,a))#	归约
4	#(F	(≺,	,	(a,a))#	移进
5	#(F,	,≺(	(	a,a))#	移进
6	#(F,(	(≺a	a	,a))#	移进
7	#(F,(a	a≻,	,	a))#	归约

8	#(F,(F	(⋖,	,	a))#	移进
9	#(F,(F,	,⋖a	a	))#	移进
10	#(F,(F,a	a⋗)	)	)#	归约
11	#(F,(F,F	,⋗)	)	)#	归约
12	#(F,(F	(≡)	)	)#	移进
13	#(F,(F)	)⋗)	)	#	归约
14	#(F,F	,⋗)	)	#	归约
15	#(F	(≡)	)	#	移进
16	#(F)	)⋗#	#		归约
17	#F	#≡#	#		接受

3、对题 2 的 G[S]

(1)给出(a,(a,a))和(a,a)的最右推导和规范归约过程。

(2)将(1)和题 2 中的(4)进行比较给出算符优先归约和规范归约的区别。

[答案]

(1)

(a,(a,a))的最右推导过程：

S => (T) => (T, S) => (T, (T)) => (T, (T, S))

=> (T, (T,a)) => (T, (S, a))=> (T, (a, a))

=> (S, (a, a)) => (a, (a, a))

(注：句柄下面有下划线)

(a,a))的最右推导过程：

S => (T) => (T, S) => (T, a) => (S, a)=> (a, (a, a))

(注：句柄下面有下划线)

(2)

(a,(a,a))的规范归约过程。

步骤	栈	输 入	动 作
1	#	(a, (a, a))#	移进
2	#(	a, (a, a))#	移进
3	#(a	, (a, a))#	归约，S→a
4	#(S	, (a, a))#	归约，L→S
5	#(T	, (a, a))#	移进
6	#(T,	(a, a))#	移进
7	#(T, (	a, a))#	移进
8	#(T, (a	, a))#	归约，S→a
9	#(T, (S	, a))#	归约，T→S
10	#(T, (T	, a))#	移进
11	#(T, (T,	a))#	移进
12	#(T, (T,a	))#	归约，S→a
13	#(T, (T, S	))#	归约，T→T, S
14	#(T, (T	))#	移进
15	#(T, (T)	)#	归约，S→(T)
16	#(T, S	)#	移进
17	#(T, S)	#	归约，T→T, S
18	#(T)	#	归约，S→(T)
19	#S	#	接受

(a,a)的规范归约过程。

步骤	栈	输 入	动 作
1	#	(a, a)#	移进
2	# (	a, a)#	移进
3	# (a	, a)#	归约，S→a
4	#(S	, a)#	归约，T→S
5	#(T	, a)#	移进
6	#(T,	a)#	移进
7	#(T, a	)#	归约，S→a
8	#(T, S	)#	归约，T→T, S
9	#(T	)#	移进
10	#(T)	#	归约，S→(T)
11	# S	#	接受

(2)算符优先文法在归约过程中只考虑终结符之间的优先关系从而确定可归约串，而与非终结符无关，只需知道把当前可归约串归约为某一个非终结符，不必知道该非终结符的名字是什么，因此去掉了单非终结符的归约。

规范归约的可归约串是句柄，并且必须准确写出可归约串归约为哪个非终结符。

4、有文法 G[S]：

S→V

V→T|ViT

T→F|T+F

F→)V*|(

(1) 给出(+(i(的规范推导。

(2) 指出句型 F+Fi(的短语，句柄，素短语。

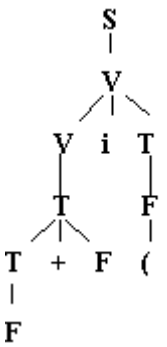
(3) G[S]是否为 OPG？若是，给出(1)中句子的分析过程。

[答案]

(1)

S=>V=>ViT=>ViF=>Vi(=>T i(=>T+F i(=>T+(i(=>F+(i(=>(+(i(

(2)句型 F+Fi(的语法树：



短语：F，F+F，(，F+Fi(

句柄：F

素短语：(

(3)

FIRSTVT 和 LASTVT

	FIRSTVT	LASTVT
S	i,+,),(	i,+,*,(
V	i,+,),(	i,+,*,(
T	+,),(	+,(,*
F	),(,	*,(

(2)算符优先关系

	i	+	*	(	)	#
i	⋗	⋖	⋗	⋖	⋖	⋗
+	⋗	⋗	⋗	⋖	⋖	⋗
*	⋗	⋗	⋗			⋗
(	⋗	⋗	⋗			⋗
)	⋖	⋖		⋖	⋖	
#	⋖	⋖		⋖	⋖	≡

因为该文法是 OP，同时任意两个终结符的优先关系唯一，所以该文法为 OPG。

(3)(+(i(的分析过程

步骤	栈	优先关系	当前符号	剩余输入串	移进或归约
1	#	#⋖(	(	(+(i(#	移进
2	#(	(⋗+	+	(i(#	归约
3	#F	#⋖+	+	(i(#	移进
4	#F+	+⋖(	(	i(#	移进
5	#F+(	(⋗i	i	(#	归约
6	#F+F	+⋗i	i	(#	归约
7	#F	#⋖i	i	(#	移进
8	#Fi	i⋖(	(	#	移进
9	#Fi(	(⋗#	#		归约
10	#FiF	i⋗#	#		归约

11	#F	#=#	#		接受

5、试为下列各文法建立算符优先关系表。

E->E and T|T

T->T or F|F

F->not F|N

N->(E)|true|false

[答案]

算符优先关系表如下：

	true	false	not	and	or	(	)	#
true				⋗	⋗		⋗	⋗
false				⋗	⋗		⋗	⋗
not	⋖	⋖	⋖	⋗	⋗	⋖	⋗	⋗
and	⋖	⋖	⋖	⋗	⋗	⋖	⋗	⋗
or	⋖	⋖	⋖	⋖	⋗	⋖	⋗	⋗
(	⋖	⋖	⋖	⋖	⋖	⋖	=	
)				⋗	⋗		⋗	⋗
#	⋖	⋖	⋖	⋖	⋖	⋖		

习题六

1、已知文法

A->aAd| aAb| ε

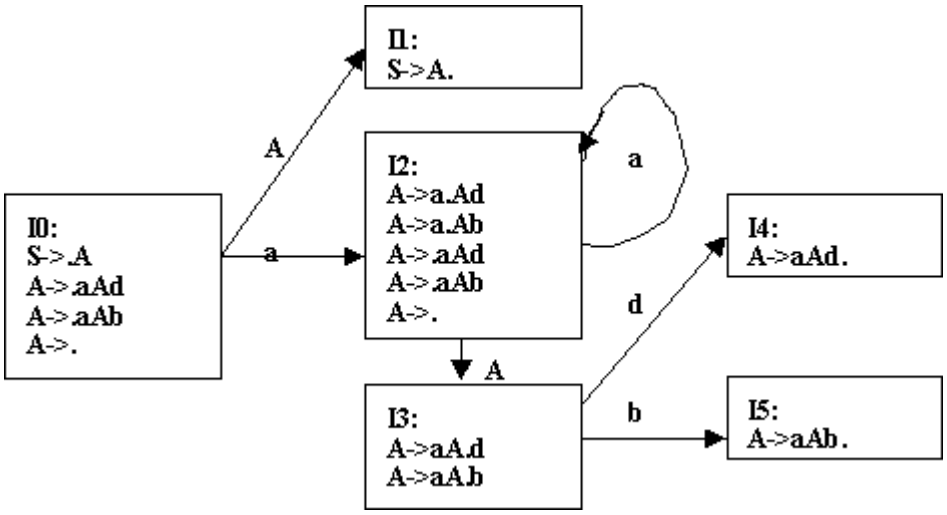
判断该文法是否 SLR（1）文法，若是构造相应分析表，并对输入串 ab#给出分析过程。

[答案]

(1)拓广文法

(0)S->A　(1) A->aAd　(2)A-> aAb　(3)A->ε

(1)构造识别活前缀的 DFA



FOLLOW(A)={d,b,#}

对于状态 I0：FOLLOW(A)∩{a}=Φ

对于状态 I1：FOLLOW(A)∩{a}=Φ

因为，在 DFA 中无冲突的现象，所以该文法是 SLR(1)文法。

(3)SLR(1)分析表

状态	ACTION				GOTO
	a	B	d	#	A
0	S2	r3	r3	r3	1
1				acc	
2	S2	r3	r3	r3	3
3		S5	S4		
4		r1	r1	r1	
5		r2	r2	r2	

(4)串 ab#的分析过程

步骤	状态栈	符号栈	当前字符	剩余字符串	动作
1	0	#	a	b#	移进
2	02	#a	b	#	归约 A->ε
3	023	#aA	b	#	移进

4	0235	#aAb	#		归约 A-> aAb
5	01	#A	#		接受

2、设文法 G 为 S → A

A → BA | ε

B → aB | b

(1)证明它是 LR(1)文法；

(2)构造它的 LR(1)分析表；

(3)给出输入符号串 abab 的分析过程。

[答案]

(1)拓广文法 G'：

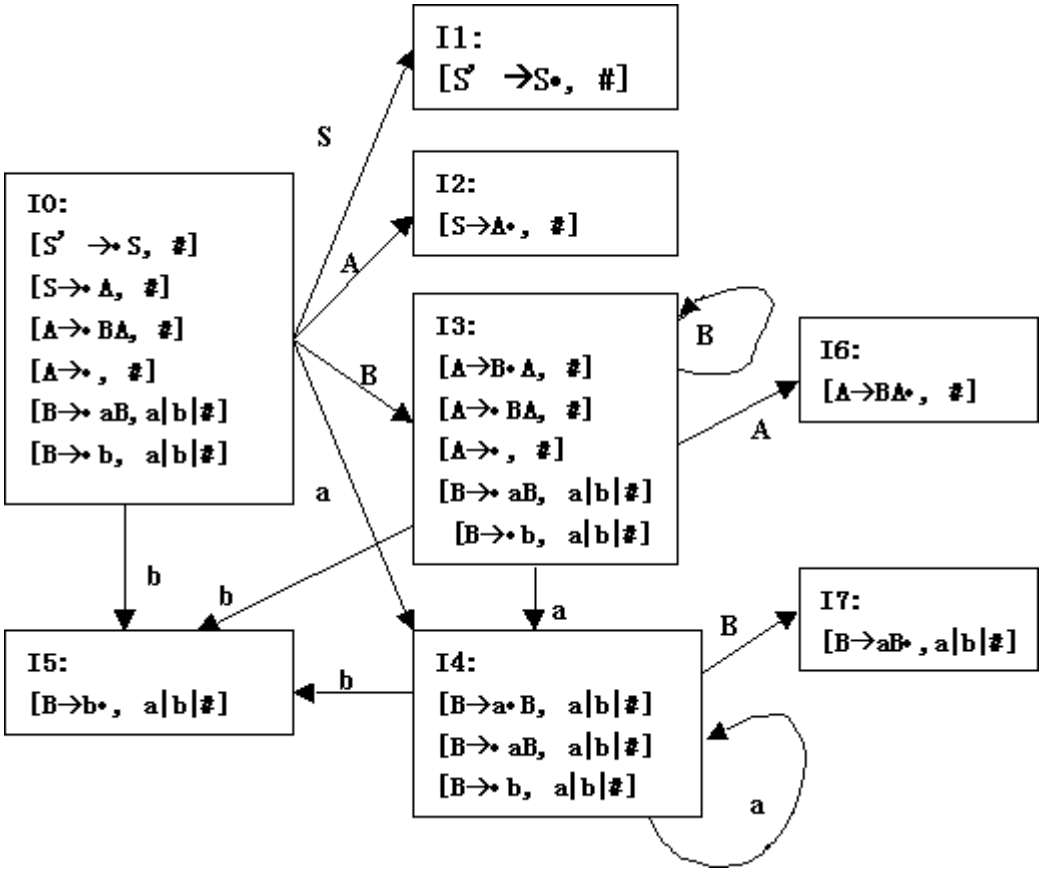
(0) S' → S　　(1) S → A　　(2) A → BA

(3) A → ε　　(4) B → aB　　(5) B → b

FIRST(A) = { ε , a, b}

FIRST(B) = {a, b}

构造的 DFA 如下:



由项目集规范族看出，不存在冲突动作。

∴该文法是 LR(1)文法。

(2)

LR(1)分析表

状态	Action			Goto		
	a	B	#	S	A	B
0	S4	S5	r3	1	2	3
1			acc			
2			r1			
3	S4	S5	r3		6	3
4	S4	S5				7
5	r5	r5	r5			
6			r2			
7	r4	r4	r4			

(3)输入串 abab 的分析过程为：

步骤	状态栈	符号栈	当前字符	剩余字符串	动作

(1)	0	#	a	bab#	移进
(2)	04	#a	b	ab#	移进
(3)	045	#ab	a	b#	归约 B→b
(4)	047	#aB	a	b#	归约 B→aB
(5)	03	#B	a	b#	移进
(6)	034	#Ba	b	#	移进
(7)	0345	#Bab	#		归约 B→b
(8)	0347	#BaB	#		归约 B→aB
(9)	033	#BB	#		归约 A→ ε
(10)	0336	#BBA	#		归约 A→BA
(11)	036	#BA	#		归约 A→BA
(12)	02	# A	#		归约 S→A
(13)	01	#S	#		acc

3、考虑文法 S→A S ｜ b

A → S A ｜ a

(1)构造文法的 LR(0)项目集规范族及相应的 DFA。

(2)如果把每一个 LR(0)项目看成一个状态，并从每一个形如 B → α ·Xβ 的状态出发画一条标记为 X 的箭弧刀状态 B → αX· β，而且从每一个形如 B → α ·Aβ 的状态出发画标记为 ε 的箭弧到所有形如 A → ·γ 的状态。这样就得到了一个 NFA。说明这个 NFA 与(a)中的 DFA 是等价的。

(3)构造文法的 SLR 分析表。

(4)对于输入串 bab，给出 SLR 分析器所作出的动作。

(5)构造文法的 LR(1)分析表和 LALR 分析表。

[答案]

(1)令拓广文法 G’为

(0)S’ → S

(1)S → A S

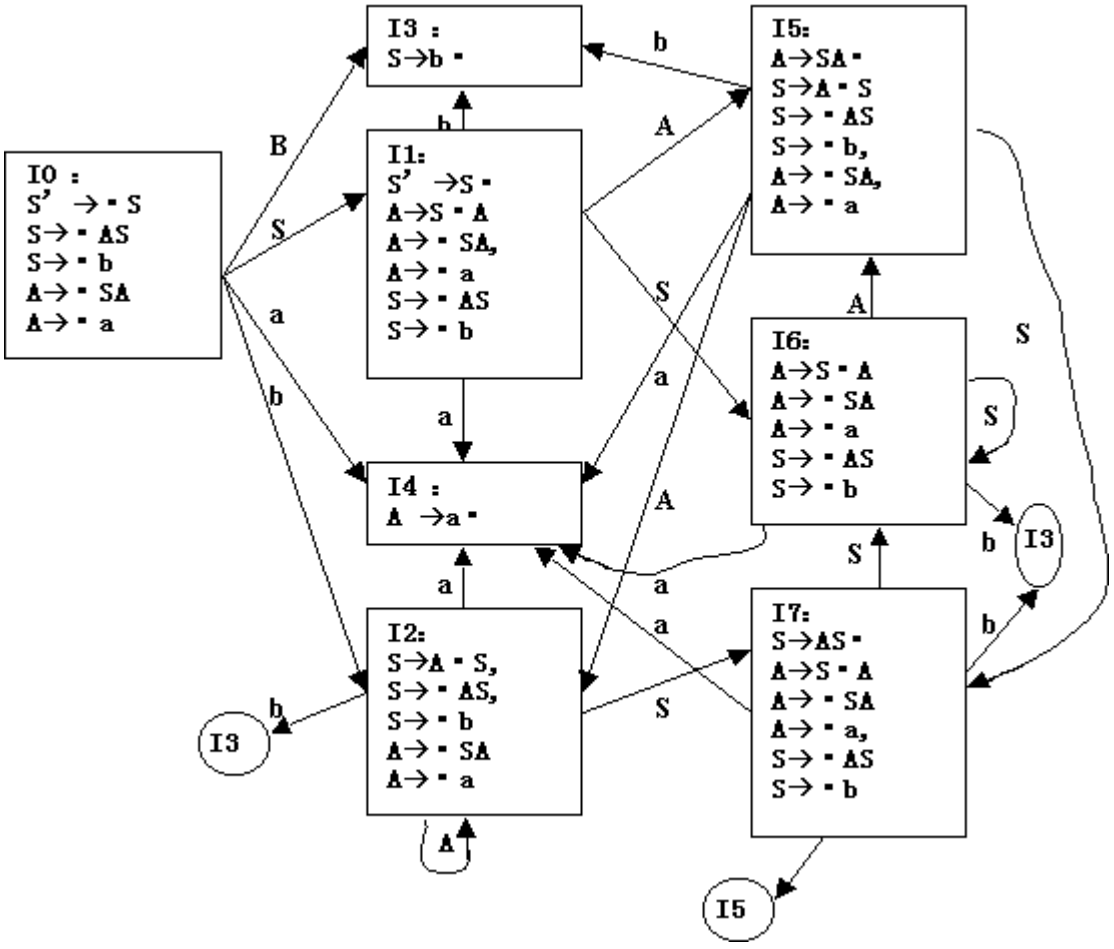
(2)S → b

(3)A → S A

(4)A → a

其 LR(0)项目集规范族及识别该文法活前缀的 DFA 如下图所示：

FOLLOW(S)={#,a,b}　　FOLLOW(A)={a,b}



(2)显然，对所得的 NFA 求 ε 闭包，即得上面的 LR(0)项目集，即 DFA 中的状态。故此 NFA 与(a)中 DFA 是等价的。

(3)文法的 SLR 分析表如下：

状态	action			goto	
	A	b	#	S	A
0	S4	S3		1	2
1	S4	S3	acc	6	5
2	S4	S3		7	2
3	r2	r2	r2		
4	r4	r4	r4		

5	S4/ r3	S3/r3		7	2
6	S4	S3		6	5
7	S4 / r1	S3 / r1	R1	6	5

因为 I5 中：FOLLOW（A）∩{a，b}≠Φ

I7 中：FOLLOW（B）∩{a，b}≠Φ

所以，该文法不是 SLR（1）文法。

或者：

从分析表中可看出存在歧义，所以该文法不是 SLR（1）文法。

(4)对于输入串 bab，SLR 分析器所作出的动作如下：

步骤	状态栈	符号栈	当前字符	剩余字符串	动作
(1)	0	#	b	ab#	移进
(2)	03	#b	a	b#	归约 S→b
(3)	01	#S	a	b#	移进
(4)	014	#Sa	b	#	归约 A→a
(5)	015	#SA	b	#	归约 A→SA
(6)	02	#A	b	#	移进
(7)	0A2b3	#Ab	#		归约 S→b
(8)	027	#AS	#		归约 S→AS
(9)	01	#S	#		接受

（在第 5 个动作产生歧义）

(b)LR(1)项目集族为：

I0 ：S’ →·S, #　I1 ：S’ → S·

S →·AS, # | a | b　A → S·A, a | b

S →·b, # | a | b　A →·a, a | b

S →·SA, a | b　S →·AS, a | b

A →·a, a | b　S →·b, a | b

I2 : S →A·S, # | a | bI3 : S → b·, # | a | b

S →·b, # | a | b

S →·AS, # | a | b I4 : A → a·, a | b

A →·SA, a | b

A →·a, a | b

I5 : A → SA·, a | b I6 : A → S·

S → A·S, a | b A →·SA, a | b

S → ·AS, a | b
A →·a, a | b

S →·b, a | b S →·AS, a | b

A →·SA, a | b S →·b, a | b

A →·a, a | b

I7 : S → AS·, a | b

A → S·A, a | b

A →·SA, a | b

A →·a, a | b

S →·AS, a | b

S →·b, a | b

∵I6 状态集中存在“归约－－移进”冲突，故无法构造 LR(1)分析表，因而也就无法构造 LALR 分析表。

4、对下面的文法

S->UTa|Tb

T->S|Sc|d

U->US|e

判断是否为LR(0),SLR(1),LALR(1),LR(1)，说明理由，并构造相应的分析表。

（1）拓广文法：

(0)S’->S

(1)S->UTa

(2)S->Tb

(3)T->S

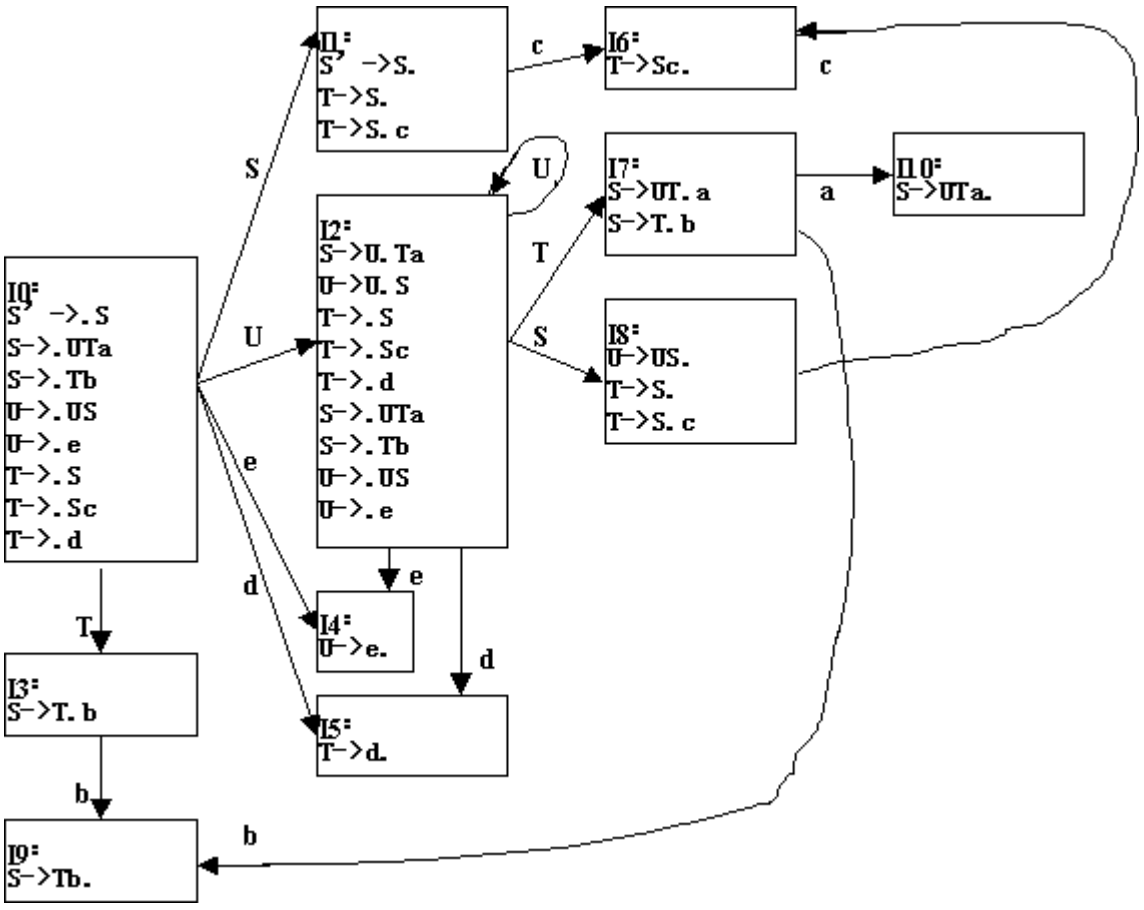
(4)T->Sc

(5)T->d

(6)U->US

(7)U->e

(2)构造识别 **LR(0)**项目的 **DFA** 如下：



因为 I1、I8 中存在冲突，所以该文法不是 LR（0）文法。

因为 FOLLOW(S)={#,c,a,b,d,e},FOLLOW(T)={a,b},FOLLOW(U)={d,e}

I1 中 FOLLOW(T)∩{c}=Φ

I8 中 FOLLOW(U)∩{c}=Φ,FOLLOW(T)∩{c}=Φ,

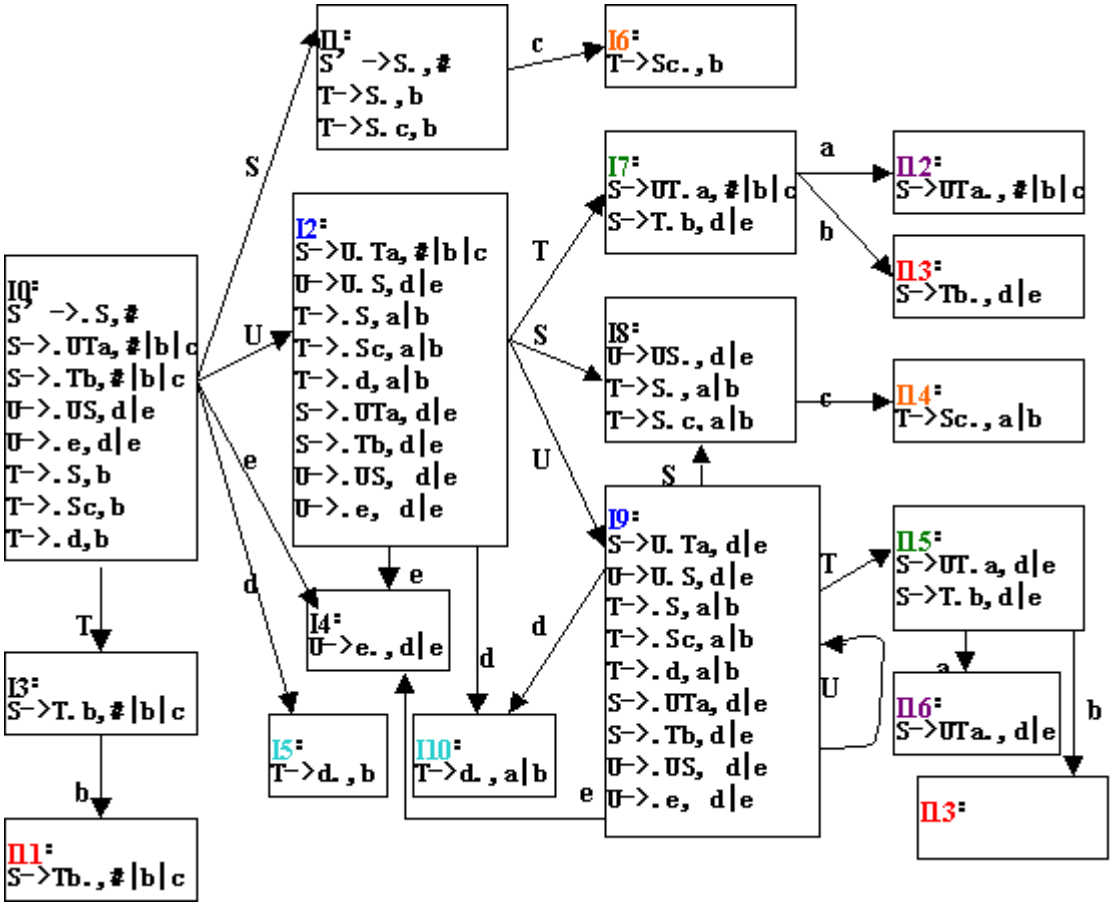
FOLLOW(T)∩FOLLOW(U)=Φ

I1、I8 中冲突解决，所以该文法是 SLR（1）文法。

	ACTION						GOTO		
状态	a	b	c	d	e	#	S	T	U
0				S5	S4		1	3	2
1	r3	r3	S6			acc			
2				S5	S4		8	7	2
3		S9							
4				r7	r7				
5	r5	r5							
6	r4	r4							
7	S10	S9							
8	r3	r3	S6	r6	r6				
9	r2	r2	r2	r2	r2	r2			
10	r1	r1	r1	r1	r1	r1			

	FIRST	FOLLOW
S	e,d	a,b ,d,e,#,c
T	e,d	a,b
U	e	d,e

构造识别 LR(1)项目的 DFA 如下：



因为不存在冲突，所以该文法是 LR（1）文法。

其中 I2、I9 可合并，I11、I13 可合并，I5、I10 可合并，I6、I14 可合并，

I12、I16 可合并，I7、I15 可合并，合并后无冲突，所以为 LALR(1)文法。

LR(1)分析表

	ACTION						GOTO		
状态	a	b	C	d	e	#	S	T	U
0				S5	S4		1	3	2
1		r3	S6			acc			
2				S10	S4		8	7	9
3		S11							
4				r7	r7				
5		r5							
6		r4							
7	S12	S13							
8	r3	r3	S14	r6	r6				

9				S10	S4		8	15	9
10	r5	r5							
11		r2	r2			r2			
12		r1	r1			r1			
13				r2	r2				
14	r4	r4							
15	S16	S13							
16				r1	r1				

LALR(1)分析表

	ACTION						GOTO		
状态	a	b	c	d	e	#	S	T	U
0				S5,10	S4		1	3	2,9
1		r3	S6			acc			
2,9				S5,10	S4		8	7,15	2,9
3		S11,13							
4				r7	r7				
5,10	r5	r5							
6,14	r4	r4							
7,15	S12,16	S11,13							
8	r3	r3	S6,14	r6	r6				
11,13		r2	r2	r2	r2	r2			
12,16		r1	r1	r1	r1	r1			

习题七

1、对于输入的表达式(4*7+1)*2，根据下表的语法制导定义建立一棵带注释的分析树。

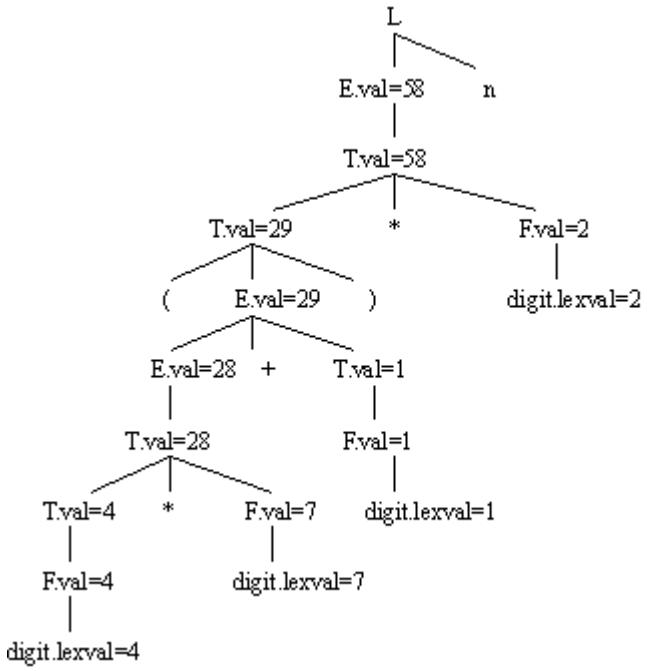
val:表示非终结符的整数值,综合属性,lexval 是单词 digit 的属性

语法制导定义

产生式	语义规则
L →E	print(E.val)
E →E^1+T	E.val:=E^1.val+T.val
E →T	E.val:=T.val
T →T^1*F	T.val:=T^1.val*F.val

T →F	T.val:=F.val
F →(E)	F.val:=E.val
F →digit	F.val:=digit.lexval

[答案]



2、令S.val为下面的文法由S生成的二进制数的值(如,对于输入101.101,S.val=5.625)；

S→L.L | L

L→LB | B

B→0 | 1

[答案]

val为综合属性，代表值属性

语法制导定义

产生式	语义规则
S→L1.L2	$S.val:=L1.val+L2.val/2^{L2.length}$
S→L	S.val:=L.val

L→L1B	L.val:=L1.val*2+B.val L.length:=L1.length+1
L→B	L.val:=B.val L.length:=1
B→0	B.val:=0
B→1	B.val:=1

3、下面文法产生的表达式是对整型和实型常数应用算符+形成的。当两个整数相加时,结果为整数，否则为实数。

E→E+T | T

T→num.num | num

给出语法制导定义确定每个子表达式的类型。

[答案]

解：a)设 type 是综合属性，代表各非终结符的“类型”属性

语法制导定义

产生式	语义规则

E→E1+T	IF (E1.type=integer) and (T.type=integer) THEN
	
	E.type:=integer
	
	ELSE
	
	E.type:=real

E→T	E.type:=T.type
T→num.num	T.type:=real
T→num	T.type:=integer

4、利用下列文法

E→E+T | T

T→num.num | num

把表达式翻译成前缀形式，并且决定类型。试用一元运算符 inttoreal 把整型值转换为相等的实型值，以使得前缀表达式中两个运算对象是同类型的。

[答案]

设 code　为综合属性，代表各非终结符的代码属性

type 为综合属性，代表各非终结符的类型属性

inttoreal 把整型值转换为相等的实型值

vtochar 将数值转换为字符串

语法制导定义

产生式	语义规则
S->E	print E.code

E→E1+T	IF (E1.type=integer) and (T.type=integer) THEN
	begin
	E.type:=integer
	E.code='+'||E_1.code ||T.code;
	end
	ELSE begin
	E.type:=real
	IF E1.type=integer THEN
	begin
	E1.type:=real
	E1.val:=inttoreal(E1.val)
	E1.code=vtochar(E1.val)

	end
	IF T.type:=integer THEN
	begin
	T.type:=real
	T.val:=inttoreal(T.val)
	T.code=vtochar(T.val)
	end
	E.code='+'||E_1.code||T.code;
	End

E→T	E.type:=T.type E.val:=T.val E.code=vtochar(E.val)
T→num.num	T.type:=real T.val:=num.num.lexval T.code=vtochar(T.val)
T→num	T.type:=integer T.val:=num.lexva T.code=vtochar(T.val)

5、请按语法制导的定义，将后缀表达式翻译成中缀表达式。注意，不允许出现冗余括号，后续表达式的文法如下：

E->EE+

E->EE*

E->id

[答案]

语法制导定义

产生式	语义规则
S->E	print E.code
E->E_1E_2+	E.code=E_1.code||'+'||E_2.code; E.op='+'
E->E_1E_2*	IF E_1.op='+' AND E_2.op='+' THEN E.code="("||E_1.code||')'||'*'||('||E_2.code||')'; ELSE IF E_1.op='+'THEN E.code="("||E_1.code||')'||'*'||E_2.code; ELSE IF E_2.op='+'THEN E.code=E_1.code||'*'||('||E_2.code||')'; ELSE E.code=E_1.code||'*'||E_2.code||;
E 一>id	E.code:=id.lexeme;

6、有文法：

S->(L)|a

L->L,S|S

给此文法配上语义动作子程序(或者说为此文法写一个语法制导定义)，它输出配对括号的个数。如对于句子(a,(a,a))，输出是 2。

[答案]

拓展文法：加入新开始符号 S'和产生式 S'->S

设 num 为综合属性，代表值属性

语法制导定义

产生式	语义规则
S'->S	print(S.num)
S->(L)	S.num:=L.num+1
S->a	S.num:=0
L->L1,S	L.num:=L1.num+S.num
L->S	L.num:=S.num

7、假设变量的说明是由下列文法生成的：

D→i L

L→,i L | :T

T→integer | real

1）建立一个语法制导定义，把每一个标志符的类型加在符号表中

2）为 1）构造一个预翻译程序

[答案]

1)type 为综合属性，代表类型属性，

函数 addtype 实现向符号表中 i 对应项填类型信息。

语法制导定义

产生式	语义动作
D→i L	D.Type:=L.Type addtype(i.entry, D.type)
L→,i L1	L.Type:=L1.Type addtype(i.entry, L.type)
L→:T	L.type:=T.type
T→integer	T.type:=integer
T→real	T.type:=real

b）采用递归下降分析法编写预翻译程序：

```
    Procedure D;
    begin
      if lookahead=id then
         begin
           match(id);
           D.type=L;
           addtype(id.entry,D.type)
         end
        else
           error
    end
  Function L: DataType;
    begin
      if lookahead=’,’  then
begin
             match( ‘,’ );
             if lookahead=id then
                begin
                    match(id);
                   L.Type=L;
                    addtype(id.entry,L.type);
                    return(L.type)
                end
              else
```

```
            error
        end
    else if lookahead=' :'  then
            begin
                match( ':' );
                L.Type=T;
                return(L.Type)
            end
        else
            error
end


 Function T: DataType;
begin
  if lookahead=integer then
    begin
      match(integer);
      return(integer)
    end
  else if lookahead=real then
        begin
          match(real);
          return(real)
        end
```

else

error

end

8、文法 G 的产生式如下：

S->(L)|a

L->L,S|S

①试写出一个语法制导定义，它输出配对括号个数；

②写一个翻译方案，打印每个 a 的嵌套深度。如((a),a),打印 2,1

[答案]

①为 S,L 引入综合属性 num，代表配对括号个数；

语法制导定义

产生式	语义动作
S'->S	print(S.num)
S->(L)	S.num:=L.num+1
S->a	S.num:=0
L->L1,S	L.num:=L1.num+S.num
L->S	L.num:=S.num

②引入继承属性 f,代表嵌套深度

S'-> {S.f:=0} S

S->'(' {L.f:=S.f+1;}

L

')'

S->a {print(S.f);}

L->{L1.f:=L.f;}

L1, {S.f:=L.f}

S

L->{S.f:＝L. f;}

S

9、对下面的文法，只利用综合属性获得类型信息。

D→L,id | L

L→T id

T→int | real

[答案]

语法制导定义

产生式	语义规则
D→L,id	D.type:=L.type addtype(id.entry,L.type)
D→L	D.type:=L.type
L→T id	L.type:=T.type addtype(id.entry,T.type)
T→int	T.type:=integer
T→real	T.type:=real

10、下面文法产生的表达式是对整型和实型常数应用算符+形成的。当两个整数相加时,结果为整数，否则为实数。

E→ TR

R →+ TR| ε

T→num.num | num

a)给出语法制导定义确定每个子表达式的类型。

b) 把表达式翻译成前缀形式，并且决定类型。试用一元运算符 inttoreal 把整型值转换为相等的实型值，以使得前缀表达式中两个运算对象是同类型的。

[答案]

a)设 type 是综合属性，代表各非终结符的“类型”属性

设 in 是继承属性，

翻译方案

产生式	语义规则
E→T R	{R.i:=T.type} {E.Type:=R.s}
R→+ T R1	 {IF (R.i=integer) and (T.type=integer) THEN R1.i:=integer ELSE R1.i :=real} {R.s:=R1.s}
R→ ε	{R.s:=R.i}
T→num.num	T.type:=real
T→num	T.type:=integer

b) 设属性 s 和 i 用于传递属性 type，属性 t 和 j 用于传递属性 val。

翻译方案

产生式	语义规则
E→T	{R.i:=T.type} {R.j:=T.val}

R	{E.Type:=R.s} {E.val:=R.t}

R→+T
 	
R1
	{IF (R.i=integer) and (T.type=integer)
	THEN
	BEGIN
	R1.i:=integer
	Print('+' ,R.j,T.val)
	R1.j:=R.j+T.val
	END
	ELSE BEGIN
	R1.i :=real
	IF R.i=integer THEN
	Begin
	R.i:=real
	R.j:=inttoreal(R.j)
	End
	IF T.type=integer THEN
	Begin
	T.type:=real
	T.val:=inttoreal(T.val)
	End
	Print('+' ,R..j,T.val)
	R1.j :=R.j+T.val
	END}
	
	{R.s:=R1.s} {R.t:=R1.t}

R→ ε	{R.s:=R.i} {R.t:=R.j}

T→num.num	{T.type:=real} {T.val:=num.num.lexval}
T→num	{T.type:=integer} {T.val:=num.lexval}

习题八

1、利用 Pascal 的作用域规则，试确定在下面的 Pascal 程序中的名字 a 和 b 的每一次出现所应用的说明。

```
program m ( input, output ) ;
  procedure n ( u, v, x, y : integer ) ;
    var m : record m, n : interger end ;
       n : record n, m : interger end ;
    begin
      with m do begin m := u ; n:= v end ;
      with n do begin m := x ; n := y end ;
      writeln ( m.m, m.n, n.m, n.n )
    end ;
  begin
    m ( 1, 2, 3, 4 )
  end.
```

[答案]

图中用蓝色数字下标相应标明。

```
program m1 ( input, output ) ;
  procedure n1( u, v, x, y : integer ) ;
    var m2 : record m3, n2 : interger end ;
```

```
      n3 : record n4, m4 : interger end ;
    begin
      with m2 do begin m3 := u ; n2 := v end ;
      with n3 do begin m4 := x ; n4 := y end ;
      writeln ( m2.m3, m2.n2, n3.m4, n3.n4 )
    end ;
  begin
    m1 ( 1, 2, 3, 4 )
  end.
```

2、 当一个过程作为参数被传递时，我们假定有以下三种与此相联系的环境可以考虑，下面的 Pascal 程序是用来说明这一问题的。

一种是词法环境（lexical environment），如此这样的一个过程的环境是由这一过程定义之处的各标识符的联编所构成；

一种是传递环境（passing environment），是由这一过程作为参数被传递之处的各标识符的联编所构成；

另一种是活动环境（activation environment），是由这一过程活动之处的各标识符的联编所构成。

试考虑在第（11）行上的作为一个参数被传递的函数 f。利用对于 f 的词法环境、传递环境和活动环境，在第（8）行上的非局部量 m 将分别处在第（6）行、（10）行和（3）行上的 m 的说明的作用域中。

（a）图示出每个过程的活动记录。

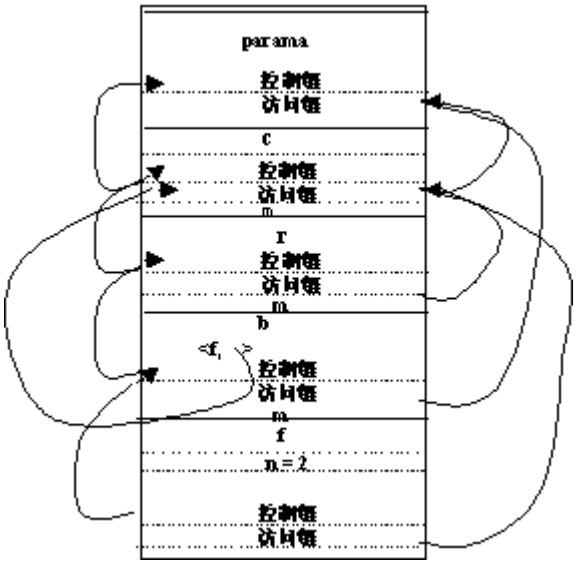
（b）试为此程序画出活动树。

（c）试给出程序的输出。

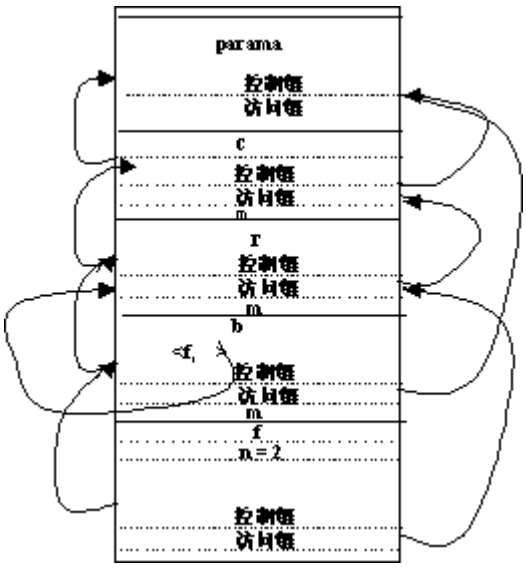
```
(1) program param ( inout, output ) ;
(2)  procedure b ( function h ( n : integer ) : integer ) ;
(3)     var m : integer ;
(4)     begin m := 3 ; writeln ( h ( 2 ) ) end { b } ;
(5)  procedure c ;
(6)     var m : integer ;
(7)     function f ( n : integer ) : integer ;
(8)       begin f := m + n end { f } ;
(9)     procedure r ;
(10)       var m : integer ;
(11)       begin m := 7 ; b ( f ) end  { r } ;
(12)    begin m := 0 ; r end { c } ;
(13)  begin
(14)    c
(15)  end .
```

[答案]

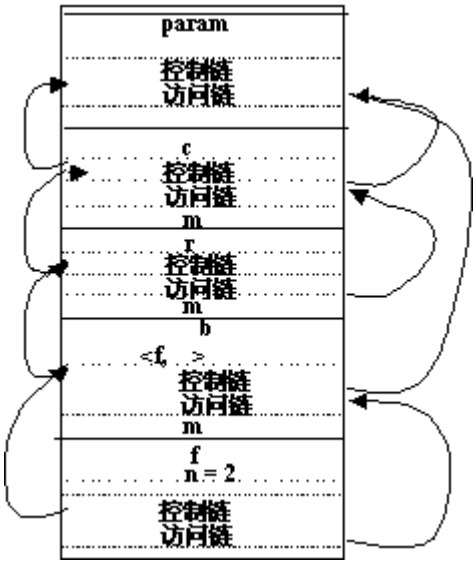
（a）词法环境


parama
控制链
访问链
c
控制链
访问链
m
r
控制链
访问链
m
b
控制链
访问链
m
f
n = 2
控制链
访问链

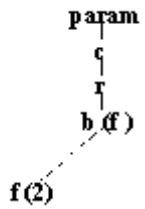
传递环境


parama
控制链
访问链
c
控制链
访问链
m
r
控制链
访问链
m
b
控制链
访问链
m
f
n = 2
控制链
访问链

活动环境



（**b**）活动树



（c）词法环境：2；传递环境：9；活动环境：5。

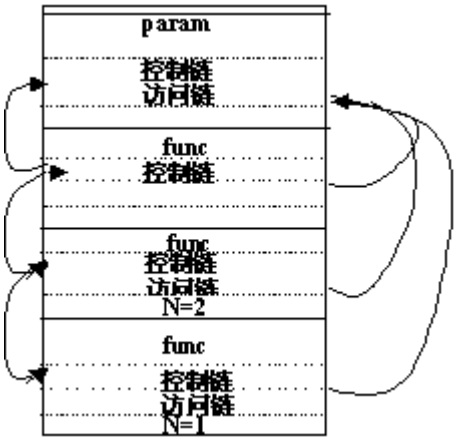
3、已知程序段：

```
BEGIN
  integer i;
  read(i);
  write("value=",func(i));
  ...
END
integer procedure func(N)
```

```
integer N;
BEGIN
 IF N==0 THEN func=1;
 ELSE IF N==1,THEN func=1;
 ELSE func=N*func(N-1)
END;
```

求当输入 i=3 时，本程序执行期间对运行栈的存储分配图。

[答案]

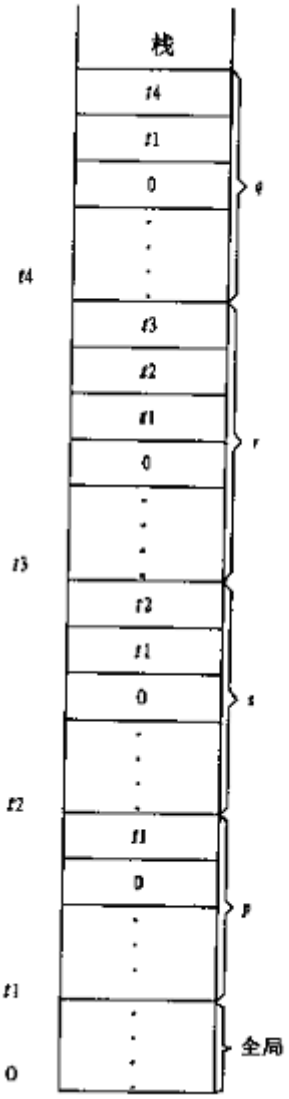


4、某语言允许过程嵌套定义和递归调用(如 Pascal 语言)，若在栈式动态存储分配中采用嵌套层次显示表 display 解决对非局部变量的引用问题，试给出下列程序执行到语句“b:＝10;”时运行栈及 display 表的示意图。

```
var x, y;
PROCEDURE P;
var a;
PROCEDURE q;
    var b:
    BEGIN{q}
     b: ＝10;
```

```
    END{q};
    PROCEDURE s;
        var c, d;
       PROCEDURE r;
        var e,f;
        BEGIN{r}
          call q;
        END {r};
       BEGIN{s}
        call r;
       END{s};
    BEGIN {p}
        call s;
    END{p};
    BEGIN{main}
      call p;
END{main}.
```

[答案]



5、试问下面的程序将有怎样的输出？分别假定：

（a）传值调用（call-by-value）；

（b）引用调用（call-by-reference）；

（c）复制恢复（copy-restore）；

（d）传名调用（call-by-name）。

```
program main ( input, output ) ;
  procedure p ( x, y, z ) ;
    begin
```

```
    y := y + 1 ;
    z := z + x ;
  end ;
begin
  a := 2 ;
  b := 3 ;
  p ( a + b , a, a ) ;
  print a
end.
```

[答案]

1)．传地址：所谓传地址是指把实在参数的地址传递给相应的形式参数。在过程段中每个形式参数都有一对应的单元，称为形式单元。形式单元将用来存放相应的实在参数的地址。当调用一个过程时，调用段必须领先把实在参数的地址传递到一个为被调用段可以拿得到的地方。当程序控制转入被调用段之后，被调用段首先把实参地址捎进自己相应的形式单元中，过程体对形式参数的任何引用 1 或赋值都被处理成对形式单元的间接访问。当调用段工作完毕返回时，形式单元(它们都是指示器)所指的实在参数单元就持有所指望的值。

2)．传结果：和“传地址”相似(但不等价)的另一种参数传递力法是所谓“传结果”。这种方法的实质是，每个形式参数对应有两个申元，第 1 个单元存放实参的地址，第 2 个单元存放实参的值。在过程体中对形参的任何引用或赋值都看成是对它的第 2 个单元的直接访问。但在过程工作完毕返回前必须把第 2 个单元的内容行放到第 1 个单元所指的那个实参单元之中。

3)．传值：所谓传值，是一种简单的参数传递方法。调用段把实在参数的值计算出来并存放在一个被调用段可以拿得到的地方。被调用段开始丁作时，首先把这些值抄入形式中元中，然后就好像使用局部名一样使用这些形式单元。如果实在参数不为指示器，那末，在被调用段中无法改变实参的值。

4)．传名：所谓传名，是一种特殊的形——实参数结合方式。解释“传名”参数的意义：过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把其中任一出现的形式参数都替换成相应的实在参数(文字替换)。它与采用“传地址”或“传值”的方式所产生的结果均不相同。

(a)2;

(b)8；

(c)7；

(d)9。

6、 下面程序的结果是 120，但是如果把第 11 行的 abs(1)改成 1 的话，则程序结果是 1，分析为什么会有这样不同的结果。

```
int fact()
{
 static int i=5;
 if(i==0)
  {
    return(i);
  }
 else
  {
    i=i-1;
    return((i+abs(1))*fact());/*第 11 行*/
  }
}
main()
{
 printf("factor of 5=%d\n",fact());
}
```

[答案]

(1)i 是静态变量，所有地方对 i 的操作都是对同一地址空间的操作，所以每次递归进入 fact 函数后，上一层对 i 的赋值仍然有效。值得注意的是，每次递归时，(i+abs(1)*fact()中(i+abs(1))的值都先于 fact 算出。所以，第 1 次递归时，所求值为 5*fact,第 2 次递归时，所求值为 4*fact，第 3 次递归时，所求值为 3*fact，如此类推，第 5 次递归时所求值为 1*fact,而此时 fact 值为 1。这样一来，实质上是求一个阶乘函数 5*4*3*2*1，所以结果为 120。

(2)之所以改动 abs(1)为 1 后会产生变化，这主要和编译时的代码生成策略有关。对于表达式(i+abs(1))*fact()，因两个子表达式都有函数调用，因此编译器先产生左子表达式代码，后产生右子表达式代码。当 abs(1)改成 1 后，那么左子表达式就没有函数调用了，于是编译器会先产生右子表达式代码，这样一来，每次递归时，(i+abs(1))*fact()如 c4)中(i+abs(1))的值都后于 fact 算出。第 1 次递归时，所求值为(i+1)*fact，第 2 次递归时，所求值为(i+1)*fact，第 3 次递归时，所求值为(i+1)*fact，如此类推，第 5 次递归时所求值为(i+1)*fact，而此时 fact 为 1，i 为 0。这样每次递归时所求的实际都是 1*fact，最后输出还是 1。因此有不问的结果。

7、下面是一个 c 语言程序及其运行结果。从运行结果看，函数 FUNC 中 4 个局部变量 i1,j1,f1，e1 的地址间隔和它们类型的大小是一致的。而 4 个形式参数 i，j，f，e 的地址间隔和它们类型的大小不一致，试分析不一致的原因。

```
#include＜stdio.h>

FUNC(i,j,f,e)

short i,j;

float f,e;

{

short i1,j1;

float f1,e1;

i1=i;j1=j;f1=f;e1=e;

printf("Addrsses of i,j,f,e＝%ld %ld %ld %ld\n",&i,&j,&f,&e);

printf("Addrsses of i1,j1,f1,e1＝%ld %ld %ld %ld\n",

    &i1,&j1,&f1,&e1);

printf("size of short,int,long,float,double＝%ld %ld %ld %ld\n",
```

```
sizeof(short),sizeof(int),sizeof(long),sizeof(float),sizeof(double));
    }
    main()
    {
        short i,j;
        float f,e;
        i=j=1;f=e=1.0;
        FUNC(i,j,f,e);
    }
```

运行结果为：

```
Addrsses of i,j,f,e＝-268438178,-268438174,-268438172,-2684364
Addrsses of i1,j1,f1,e1＝-268438250,-268438252,-268438256,-268438260
size of short,int,long,float,double＝2,4,4,4,8
```

[答案]

C 编译是不作调用时形参和实参一致性检查的。注意在 C 语言中，整数有 short，int 和 long 共 3 种类型，实数有 float 和 double 两种类型。C 语言的参数调用是按传值方式进行的，一个整型表达式的值究竟是按 short，int 还是 long 方式传递呢?

显然，这儿约定为应该按取参数的最大方式来读取参数，即对于整数用 **long** 方式，实数用 **double** 方式。虽然参数传递是按最大方式进行，但是被调用函数中形式参数是按自己类型定义来取值的。这样一来，参数传递和取值是不一致的，就要考虑边界对齐问题。因 **float** 类型需要 **4** 字节，而 **double** 类型为 **8** 字节，故当从 **double** 类型变量取 **float** 类型的值时，应从第 **l~4** 个字节分配 **float** 类型数。**short** 型的形参是取 **long** 型值 **4** 字节中的后两字节内容，**float** 型的形参是取 **double** 值 **8** 字节的前 **4** 字节。这样才出现这 **4** 个形参地址的结果。

习题九

1.翻译算术表达式 a*- (b+c)为

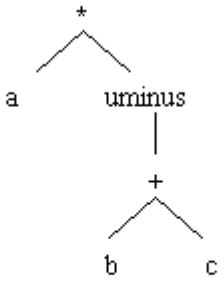
a)：一棵语法树

b)：后缀式

c)：三地址代码

[答案]

a) 语法树：



b) 后缀式:

a b c + uminus *

c)三地址代码:

t1 := b + c

t2 := - t1

t3 := a * t2

2. 翻译算术表达式 –(a+b)*(c+d) +(a+b+c) 为

a)：四元式

b)：三元式

c)：间接三元式

[答案]

先写出三地址代码为：

t1 := a + b

t2 := - t1

t3 := c + d

t4 := t2 * t3

t5 := a + b

t6 := t5 + c

t7: = t4 + t6

a)：对应的四元式为：

	op	arg1	arg2	result
(0)	**+**	**a**	**b**	**t1**
(1)	**uminus**	**t1**		**t2**
(2)	**+**	**c**	**d**	**t3**
(3)	*****	**t2**	**t3**	**t4**
(4)	**+**	**a**	**b**	**t5**
(5)	**+**	**t5**	**c**	**t6**
(6)	**+**	**t4**	**t6**	**t7**

b)：对应的三元式为：

	op	arg1	arg2
(0)	**+**	**a**	**b**
(1)	**Uminus**	**(0)**	
(2)	**+**	**c**	**d**
(3)	*****	**(1)**	**(2)**
(4)	**+**	**a**	**b**
(5)	**+**	**(4)**	**c**
(6)	**+**	**(3)**	**(5)**

c)：对应的间接三元式为：

	statement
(0)	15
(1)	16
(2)	17
(3)	18
(4)	15
(5)	19
(6)	20

	op	arg1	arg2
15	+	a	b
16	uminus	15	
17	+	c	d
18	*	16	17
19	+	15	c
20	+	18	19

3.写出语句

WHILE(A<B) DO

IF(C<D)THEN X:=Y+Z;

的四元式表示

[答案]

100 IF A<B GOTO 102

101 GOTO 107

102 IF C<D GOTO 104

103 GOTO 100

104 T:=Y+Z

105 X:=T

106 GOTO 100

107 ……

4.将下列赋值语句译成三地址代码。

A[i,j] :=B[i,j] + C[A[k,l]] + D[i+j]

[答案]

t11 := i * 20

t12 := t11+j

t13 := A-84;

t14 := 4*t12

t15 := t13[t14] //A[i,j]

t21 := i*20

t22 := t21+j

t23 := B-84;

t24 := 4*t22

t25 := t23[t24] //B[i,j]

t31 := k*20

t32 := t31+l

t33 := A-84

t34 := 4*t32

t35 := t33[t34] //A[k,l]

t36 := 4*t35

t37 := C-4

t38 := t37[t36] //C[A[k,l]]

t41 := i+j

t42 := 4*t41

t43 := D-4

t44 := t43[t42] //D[i+j]

t1 := t25 +t38

t2 := t1 + t44

t23[t24] := t2

5.写出 for 语句的翻译方案

[答案]

产生式	动作
S→for E do S1	**S.begin := newlabel** **S.first := newtemp** **S.last := newtemp** **S.curr:= newtemp** **S.code:=gen(S.first ":=" E.init)** **||gen(S.last ":=" E.final)** **||gen("if" S.first ">" S.last "goto" S.next)** **||gen(S.curr ":=" S.first)** **||gen(S.begin ":")** **||gen("if " S.curr ">" S.Last "goto" S.next)** **||S1.code** **||gen(S.curr := succ(S.curr))** **||gen("goto" S.begin)**
E→v:=initial to final	**E.init := initial.place** **E.final := final.place**

6.写出说明语句中的名字和类型及相对地址的翻译模式，以允许在形如 D→id : T 的说明中可用一串名字表来代替单个名字。

[答案]

产生式	动作
P→ **D**	**{offset := 0}**
D→D;D	
D→id L	**{enter(id.name , L.type , offset)** **offset := offset + L.width}**
L→id , L_1	**{L.type := L_1.type** **L.width := L_1.width** **enter(id.name , L_1.type , offset)** **offset := offset + L_1.width }**
L→:T	**{L.type := T.type** **L.width := T.width}**
T→integer	**{T.type := integer** **T.width := 4}**
T→real	**{T.type := real** **T.width := 8}**
T→array [num] of T_1	**{T.type:=array(num.val , T_1.Type** **T.width := num.val * T_1.Width)**
T→^T_1	**{T.type := pointer(T_1.type)** **T.width := 4}**

习题十

1、给出如下 4 元式序列：

(1) J:=0;

(2)L1:I:=0;

(3) IF I<8,goto L3;

(4)L2:A:=B+C;

(5)　B:=D*C;

(6)L3:IF B=0,goto L4;

(7)　Write B;

(8)　goto L5;

(9)L4:I:=I+1;

(10)　IF I<8,goto L2;

(11)L5:J:=J+1;

(12)　IF J<=3,goto L1;

(13)　STOP

①画出上述 4 元式序列的程序流程图 G,

②求出 G 中各结点 N 的必经结点集 D(n),

③求出 G 中的回边与循环。

[答案]

①四元式程序基本块入口语句的条件是：

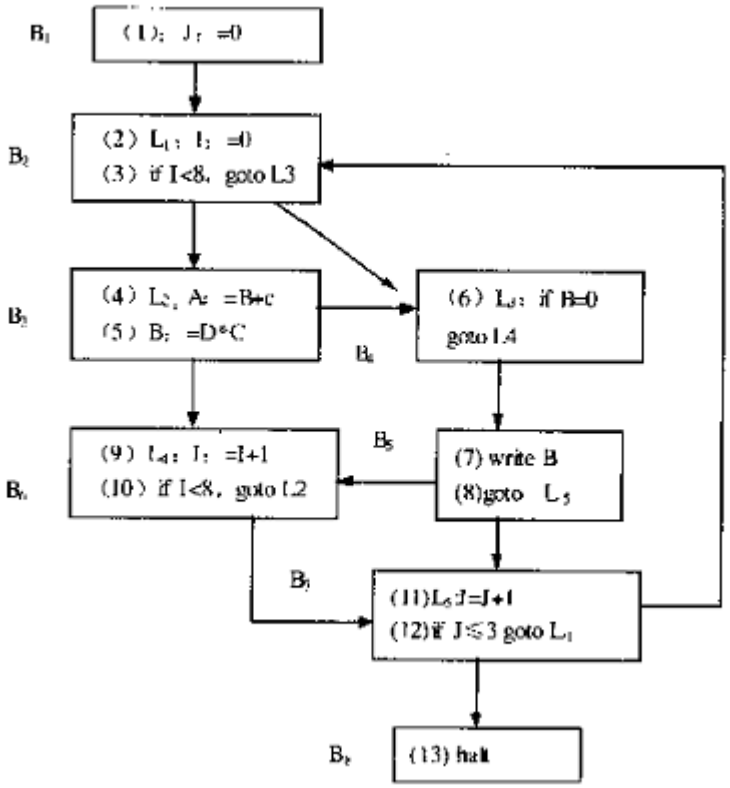
(1)它们是程序的第一个语句；或，

(2)能由条件转移语句或无条件转移语句转移到的语句；或，

(3)紧跟在条件转移语句后的语句。

(4)根据这 3 个条件，可以判断，设 1，2，3，4，6，7，9，11，13 为入口语句，故基本块为 l，2/3，4/5，6，7/8，9/1O，11/12，13，

故可画出程序流图如下图所示



②D(l)＝{1}，D(2)＝{1，2}，D(3)＝{1，2，3}，D(4)={1，2，4}，D(5)＝{1，2，4，5}，D(6)＝{1，2，4，6}，D(7)＝{1，2，4，7}，D(8)＝{1，2，4，7，8}，即为所求必经结点集。

③回边的定义为：假设 a->b 为流图中一条有向边，若 b DOM a，则 a->b 为流图中一条回边。

故当已知必经结点集时，可立即求出所有回边。

易知本题回边只有 7->2。(按递增顺序考察所有回边。）

称满足如下两个条件的结点序列为一个循环。

(1)它们强连通，即任意两个结点，必有一通路，且该通路上各结点都属于该结点序列，如序列只包含一个结点，则必有一有向边从该结点引到自身。

(2)它们中间有一个而且只有一个是入口结点。所谓入口结点，是指序列中具有下列性质的结点，从序列外某结点有一有向边引到它，或它就是程序流图的首结点。

求出回边 7->2，可知循环为 234567，即为所求。

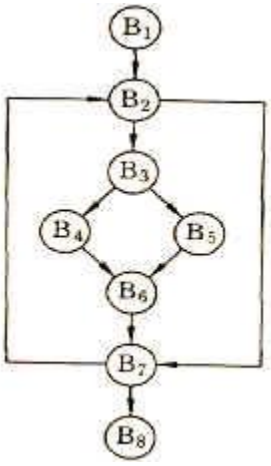
返回

2、对下面的流图，

(1)求出流图中各结点 N 的必经结点集 D(n)，

(2)求出流图中的回边，

(3)求出流图中的循环。



[答案]

(1)流图中各结点 N 的必经结点集 D(n)，

D(l)＝{1}，D(2)＝{1，2}，D(3)＝{1，2，3}，D(4)={1，2，3，4}，D(5)＝{1，2，3，5}，D(6)＝{1，2，3，6}，D(7)＝{1，2}，D(8)＝{1，2，7，8}

(2)求出流图中的回边，

7->2

(3)求出流图中的循环:2、7、3、4、5、6

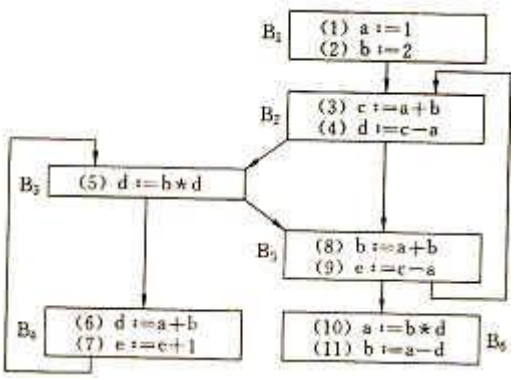
返回

3、对下面的流图，

(1)求出流图中各结点 N 的必经结点集 D(n)，

(2)求出流图中的回边，

(3)求出流图中的循环。



[答案]

(1)流图中各结点 N 的必经结点集 D(n)，

D(l)＝{1}，D(2)＝{1,2}，D(3)＝{1,2,3}，D(4)={1,2,3,4}，D(5)＝{1,2,5}，

D(6)＝{1,2,5,6}

(2)求出流图中的回边，

5->2，4->3

(3)求出流图中的循环:

回边 5->2 对应的循环：2、5、3、4；

回边 4->3 对应的循环：3、4

返回

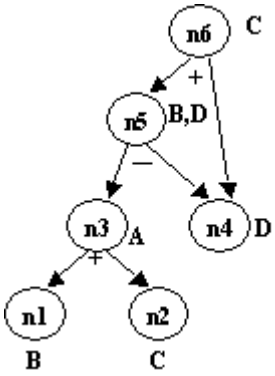
4、基本块的 DAG 如下图所示，若(1)B 在该基本块出口处不活跃,(2)B 在该基本块出口处活跃的，请分别给出以下代码经过优化后的代码。

A:＝B+C

B:＝A-D

C:＝B+C

D:＝A-D



[答案]

①当 B 在出口不活跃时，则 B 在外面就无用了，故 B：=A-D 这条赋值语句可删去，另外，由于代码生成方面的关系，可把 D 的赋值语句提前到 C 的赋值语句以前。

故得到：

A:=B+C

D:=A-D

C:=D+C

②当 B 在出口活跃时，则 B 在出口处要引用，B 的赋值语句就不可删去了，然而 D 与 B 充全一样，故 D 的赋值语句可简化，得：

A:＝B+C

B:＝A-D

D:=B

C:=B+C

返回

5、试对以下基本块：

A：=B*C

D：=B/C

E：=A+D

F：=2*E

G：=B*C

H：=G*G

F：=H*G

L：=F

M：=L

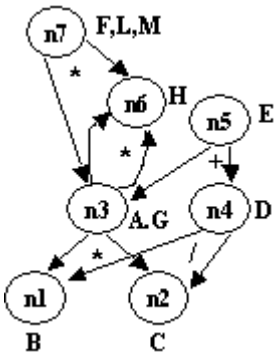
应用 DAG 对其进行优化，并就以下两种情况分别写出优化后的四元式序列；

（1）假设只有 G、L、M 在基本块后面还要被引用；

（2）假设只有 L 在基本块后面还要被引用。

[答案]

该基本块对应的 DAG 如下：



根据 DAG 图,优化后的语句序列为

A：=B*C

G：=A

D：=B/C

E：=A+D

H：=A*A

F：=A*H

L：=F

M：=F

（1）假设只有 G、L、M 在基本块后面还要被引用；

S1：=B*C

G：=S1

S2：=S1*S1

S3：=S1*S2

L：=S3

M：=S3

（2）假设只有 L 在基本块后面还要被引用；

S1：=B*C

S2：=S1*S1

S3：=S1*S2

L：=S3

（备注：S1，S2，S3 为新引入的临时变量）