第 1 章 引论

第 1 题

解释下列术语：

（1）编译程序

（2）源程序

（3）目标程序

（4）编译程序的前端

（5）后端

（6）遍

答案：

（1） 编译程序：如果源语言为高级语言，目标语言为某台计算机上的汇编语言或机器语言，则此翻译程序称为编译程序。

（2） 源程序：源语言编写的程序称为源程序。

（3） 目标程序：目标语言书写的程序称为目标程序。

（4） 编译程序的前端：它由这样一些阶段组成：这些阶段的工作主要依赖于源语言而与目标机无关。通常前端包括词法分析、语法分析、语义分析和中间代码生成这些阶段，某些优化工作也可在前端做，也包括与前端每个阶段相关的出错处理工作和符号表管理等工作。

（5） 后端：指那些依赖于目标机而一般不依赖源语言，只与中间代码有关的那些阶段，即目标代码生成，以及相关出错处理和符号表操作。

（6） 遍：是对源程序或其等价的中间语言程序从头到尾扫视并完成规定任务的过程。

第 2 题

一个典型的编译程序通常由哪些部分组成？各部分的主要功能是什么？并画出编译程序的总体结构图。

答案：

一个典型的编译程序通常包含 8 个组成部分， 它们是词法分析程序、 语法分析程序、 语义分析程序、中间代码生成程序、中间代码优化程序、目标代码生成程序、表格管理程序和错误处理程序。其各部分的主要功能简述如下。

词法分析程序： 输人源程序， 拼单词、 检查单词和分析单词， 输出单词的机内表达形式。

语法分析程序：检查源程序中存在的形式语法错误，输出错误处理信息。

语义分析程序： 进行语义检查和分析语义信息， 并把分析的结果保存到各类语义信息表中。

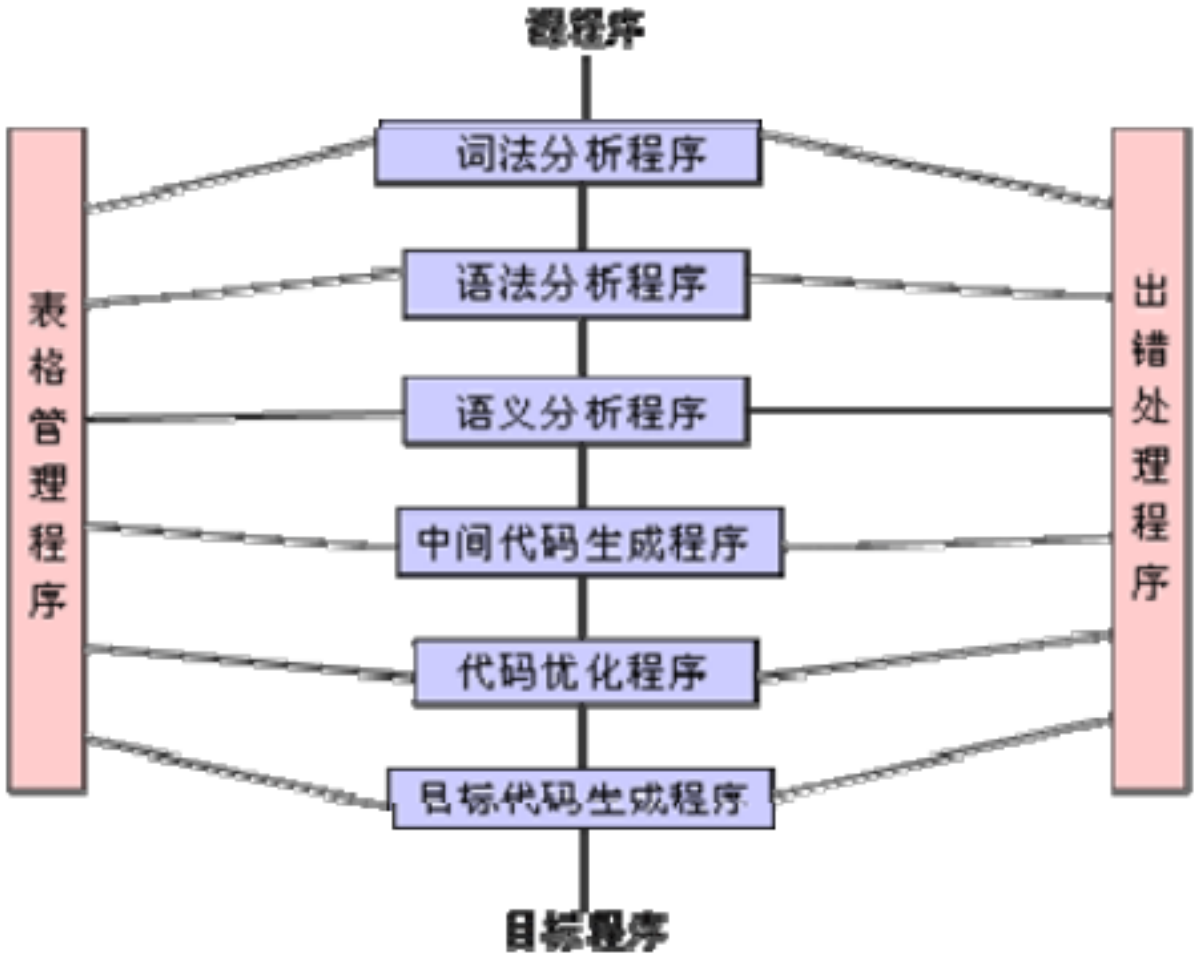
中间代码生成程序： 按照语义规则， 将语法分析程序分析出的语法单位转换成一定形式的中间语言代码，如三元式或四元式。

中间代码优化程序：为了产生高质量的目标代码，对中间代码进行等价变换处理。

目标代码生成程序：将优化后的中间代码程序转换成目标代码程序。

表格管理程序： 负责建立、 填写和查找等一系列表格工作。 表格的作用是记录源程序的各类信息和编译各阶段的进展情况， 编译的每个阶段所需信息多数都从表格中读取， 产生的中间结果都记录在相应的表格中。 可以说整个编译过程就是造表、 查表的工作过程。 需要指出的是，这里的“ 表格管理程序 ”并不意味着它就是一个独立的表格管理模块，而是指编译程序具有的表格管理功能。

错误处理程序： 处理和校正源程序中存在的词法、 语法和语义错误。 当编译程序发现源程序中的错误时， 错误处理程序负责报告出错的位置和错误性质等信息， 同时对发现的错误进行适当的校正（修复） ，目的是使编译程序能够继续向下进行分析和处理。



注意：如果问编译程序有哪些主要构成成分，只要回答六部分就可以。如果搞不清楚，就回答八部分。

第 3 题

何谓翻译程序、编译程序和解释程序？它们三者之间有何种关系？

答案：

翻译程序是指将用某种语言编写的程序转换成另一种语言形式的程序的程序， 如编译程序和汇编程序等。

编译程序是把用高级语言编写的源程序转换 （加工） 成与之等价的另一种用低级语言编写的目标程序的翻译程序。

解释程序是解释、执行高级语言源程序的程序。解释方式一般分为两种：一种方式是，源程序功能的实现完全由解释程序承担和完成，即每读出源程序的一条语句的第一个单词，则依据这个单词把控制转移到实现这条语句功能的程序部分， 该部分负责完成这条语句的功能的实现， 完成后返回到解释程序的总控部分再读人下一条语句继续进行解释、 执行， 如此反复；另一种方式是，一边翻译一边执行，即每读出源程序的一条语句，解释程序就将其翻译成一段机器指令并执行之，然后再读人下一条语句继续进行解释、执行，如此反复。无论

是哪种方式， 其加工结果都是源程序的执行结果。 目前很多解释程序采取上述两种方式的综合实现方案， 即先把源程序翻译成较容易解释执行的某种中间代码程序， 然后集中解释执行中间代码程序，最后得到运行结果。

广义上讲， 编译程序和解释程序都属于翻译程序， 但它们的翻译方式不同， 解释程序是边翻译（解释）边执行，不产生目标代码，输出源程序的运行结果。而编译程序只负责把源程序翻译成目标程序， 输出与源程序等价的目标程序， 而目标程序的执行任务由操作系统来完成，即只翻译不执行。

第 4 题

对下列错误信息，请指出可能是编译的哪个阶段（词法分析、语法分析、语义分析、代码生成）报告的。

（1） else 没有匹配的 if

（2） 数组下标越界

（3） 使用的函数没有定义

（4） 在数中出现非数字字符

答案：

（1） 语法分析

（2） 语义分析

（3） 语法分析

（4） 词法分析

第 5 题

编译程序大致有哪几种开发技术？

答案：

（1）自编译：用某一高级语言书写其本身的编译程序。

（2）交叉编译： A 机器上的编译程序能产生 B 机器上的目标代码。

（3）自展：首先确定一个非常简单的核心语言 L0，用机器语言或汇编语言书写出它的编译程序 T0，再把语言 L0 扩充到 L1，此时 L0？ L1 ,并用 L0 编写 L1 的编译程序 T1，再把语言 L1 扩充为 L2，有 L1 ？ L2 ,并用 L1 编写 L2 的编译程序 T2，……， 如此逐步扩展下去，好似滚雪球一样，直到我们所要求的编译程序。

（4）移植：将 A 机器上的某高级语言的编译程序搬到 B 机器上运行。

第 6 题

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。

第 2 章 PL/0 编译程序的实现

第 1 题

PL/0 语言允许过程嵌套定义和递归调用，试问它的编译程序如何解决运行时的存储管理。

答案：

PL/0 语言允许过程嵌套定义和递归调用，它的编译程序在运行时采用了栈式动态存储管理。（数组 CODE 存放的只读目标程序，它在运行时不改变。）运行时的数据区 S 是由解释程序定义的一维整型数组，解释执行时对数据空间 S 的管理遵循后进先出规则，当每个过程 (包括主程序) 被调用时， 才分配数据空间， 退出过程时， 则所分配的数据空间被释放。应用动态链和静态链的方式分别解决递归调用和非局部变量的引用问题。

第 2 题

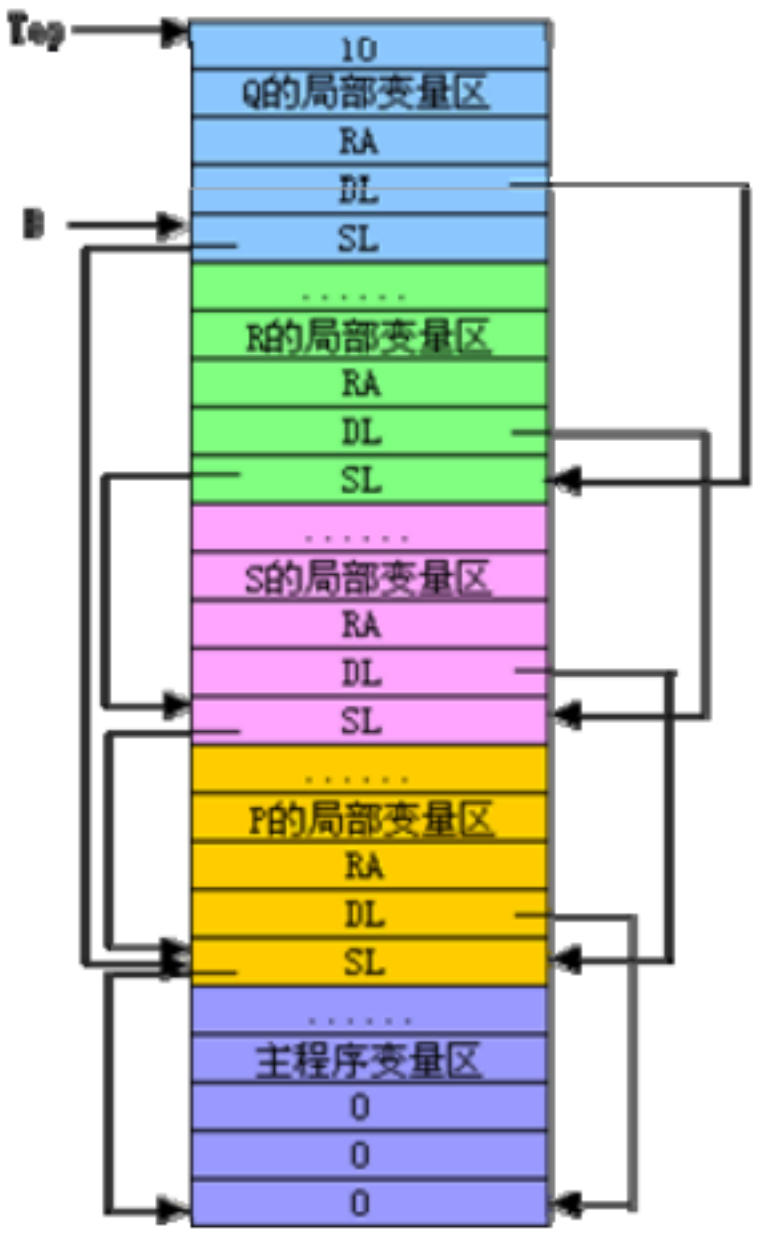
若 PL/0 编译程序运行时的存储分配策略采用栈式动态分配， 并用动态链和静态链的方式分别解决递归调用和非局部变量的引用问题， 试写出下列程序执行到赋值语句 b：=10 时运行栈的布局示意图。

```
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：

程序执行到赋值语句　b：=10 时运行栈的布局示意图为：



第 3 题

写出题 2 中当程序编译到 r 的过程体时的名字表 table 的内容。

name kind		level/val	adr	size

答案：

题 2 中当程序编译到 r 的过程体时的名字表 table 的内容为：

name kind	level/val	adr	size
x variable	0	dx	
y variable	0	dx+1	
p procedure	0	过程 p 的入口（待填） 5	

a variable	1	dx	
q procedure	1	过程 q 的入口 4	
s procedure	1	过程 s 的入口（待填） 5	
c variable	2	dx	
d variable	2	dx	
r procedure	2	过程 r 的入口 5	
e variable	3	dx	
f variable	3	dx+1	

注意： q 和 s 是并列的过程，所以 q 定义的变量 b 被覆盖。

第 4 题

指出栈顶指针 T，最新活动记录基地址指针 B，动态链指针 DL，静态链指针 SL 与返回地址 RA 的用途。

答案：

栈顶指针 T，最新活动记录基地址指针 B，动态链指针 DL，静态链指针 SL 与返回地址 RA 的用途说明如下：

T： 栈顶寄存器 T 指出了当前栈中最新分配的单元 (T 也是数组 S 的下标)。

B：基址寄存器， 指向每个过程被调用时， 在数据区 S 中给它分配的数据段起 始 地址，也称基地址。

SL： 静态链， 指向定义该过程的直接外过程 （或主程序） 运行时最新数据段的基地址，用以引用非局部（包围它的过程）变量时，寻找该变量的地址。

DL： 动态链，指向调用该过程前正在运行过程的数据段基地址，用以过程执行结束释放数据空间时，恢复调用该过程前运行栈的状态。

RA： 返回地址，记录调用该过程时目标程序的断点，即调用过程指令的下一条指令的地址，用以过程执行结束后返回调用过程时的下一条指令继续执行。

在每个过程被调用时在栈顶分配 3 个联系单元，用以存放 SL，DL， RA 。

第 5 题

PL/0 编译程序所产生的目标代码是一种假想栈式计算机的汇编语言，请说明该汇编语言中下列指令各自的功能和所完成的操作。

（1） INT 0 A

（2） OPR 0 0

（3） CAL L A

答案：

PL/0 编译程序所产生的目标代码中有 3 条非常重要的特殊指令，这 3 条指令在 code 中的位置和功能以及所完成的操作说明如下：

INT 0 A

在过程目标程序的入口处，开辟　　A 个单元的数据段。　A 为局部变量的个数　+3。

OPR 0 0

在过程目标程序的出口处，释放数据段（退栈），恢复调用该过程前正在运行的过程的数据段基址寄存器　B 和栈顶寄存器　T 的值，并将返回地址送到指令地址寄存器　　P 中，以使调用前的程序从断点开始继续执行。

CAL L A

调用过程，完成填写静态链、动态链、返回地址，给出被调用过程的基地址值，送入基址寄存器　B 中，目标程序的入口地址　A 的值送指令地址寄存器　　P 中，使指令从　A 开始执行。

第 6 题

给出对　PL/0 语言作如下功能扩充时的语法图和　　EBNF 的语法描述。

(1) 扩充条件语句的功能使其为：

if 〈条件〉 then 〈语句〉 [else〈语句〉]

(2) 扩充 repeat 语句为：

repeat 〈语句〉 {；〈语句〉 }until 〈条件〉

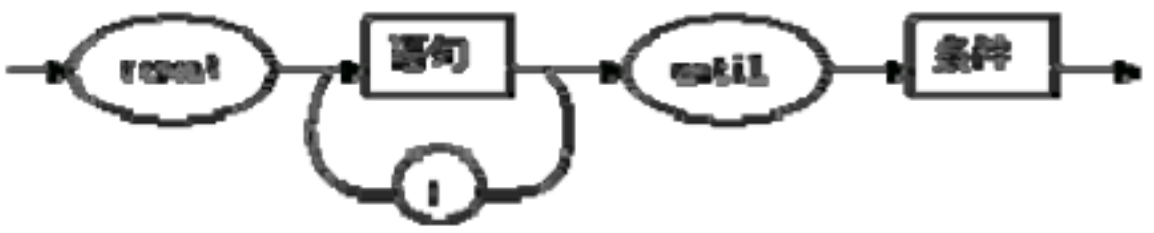
答案：

对 PL/0 语言作如下功能扩充时的语法图和　　EBNF 的语法描述如下：

(1) 扩充条件语句的语法图为：

EBNF 的语法描述为：　　〈条件语句〉 ::= if 〈条件〉 then〈语句〉 [else〈语句〉]

(2) 扩充 repeat 语句的语法图为：



EBNF 的语法描述为：　　〈 重复语句〉 ::= repeat〈语句〉 { ；〈语句〉 }until 〈条件〉

第 3 章　文法和语言

第 1 题

文法 G＝({A,B,S},{a,b,c},P,S) 其中 P 为：

S→Ac|aB

A→ab

B→bc

写出 L(G[S]) 的全部元素。

答案：

L(G[S])={abc}

第 2 题

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

或者：允许 0 开头的非负整数？

第 3 题

为只包含数字、加号和减号的表达式，例如 9-2＋5，3-1，7 等构造一个文法。

答案：

G[S]:

S->S+D|S-D|D

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

第 4 题

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

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

第 5 题

写一文法，使其语言是偶正整数的集合。　　要求：

(1) 允许 0 打头；

(2)不允许 0 打头。

答案：

(1)允许 0 开头的偶正整数集合的文法

E→NT|D

T→NT|D

N→D|1|3|5|7|9

D→0|2|4|6|8

(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|0

G→D|0

第 6 题

已知文法 G：

<表达式 >::=< 项>｜<表达式 >＋<项>

<项>::=< 因子 >｜ <项>*< 因子 >

<因子 >::= （<表达式 >）｜ i

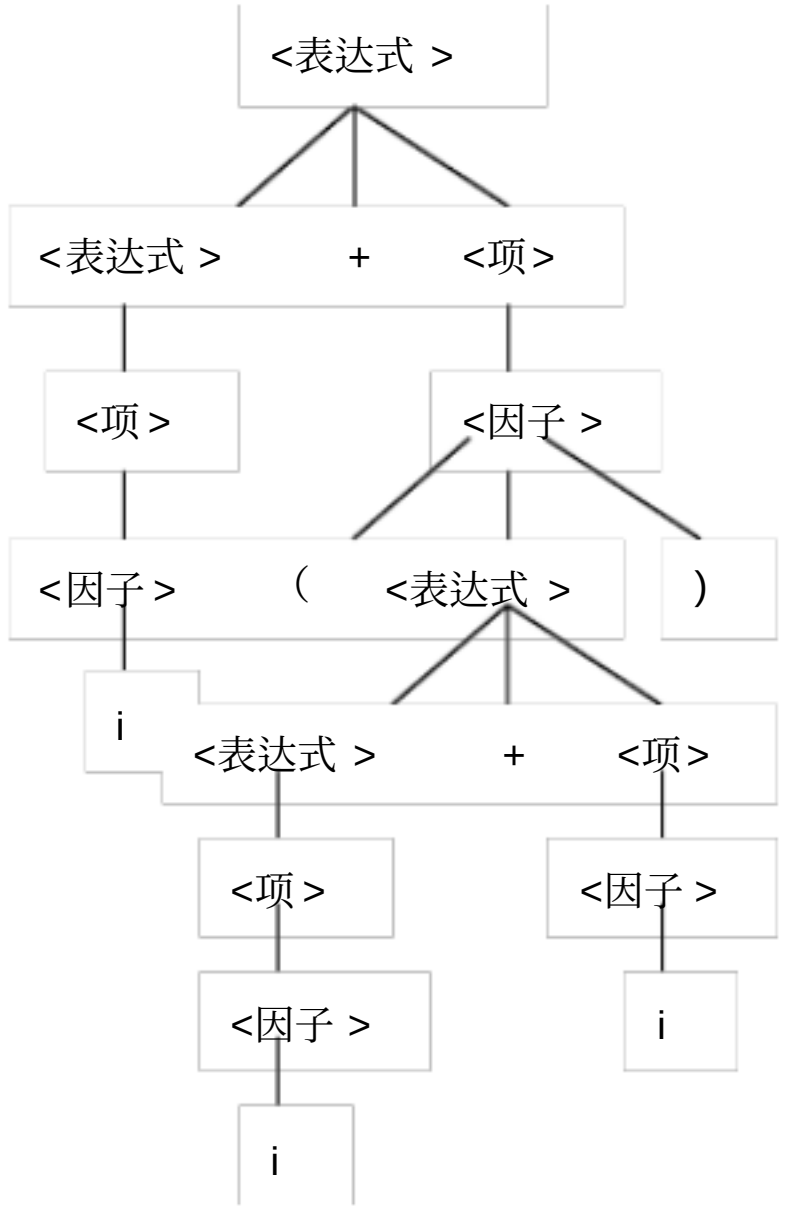
试给出下述表达式的推导及语法树。

（5） i+(i+i)

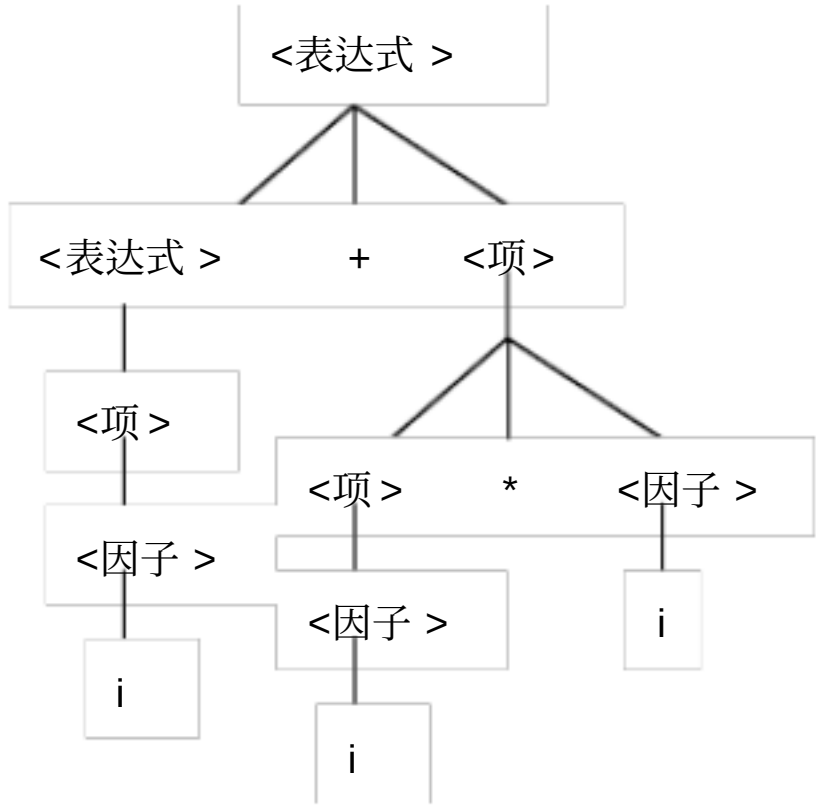
（6） i+i*i

答案：

(5) <表达式 >
=>< 表达式 >＋<项>
=>< 表达式 >＋<因子 >
=>< 表达式 >＋（<表达式 >）
=>< 表达式 >＋（<表达式 >＋<项>）
=>< 表达式 >＋（<表达式 >＋<因子 >）
=>< 表达式 >＋（<表达式 >＋i）
=>< 表达式 >＋（<项>＋i）
=>< 表达式 >＋（<因子 >＋i）
=>< 表达式 >＋（i＋i）
=>< 项>＋（i＋i）
=>< 因子 >＋（i＋i）
=>i＋（i＋i）



(6) <表达式 >
=>< 表达式 >＋<项>
=>< 表达式 >＋<项>*< 因子 >
=>< 表达式 >＋<项>*i
=>< 表达式 >＋<因子 >*i
=>< 表达式 >＋i*i
=>< 项>＋i*i
=>< 因子 >＋ i*i
=>i＋i*i



第 7 题

证明下述文法　G[〈表达式〉]是二义的。

〈表达式〉 ::=a|(〈表达式〉)|〈表达式〉〈运算符〉〈表达式〉

〈运算符〉 ::=+|-|*|/

答案：

可为句子 a+a*a 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〉 a

⇒〈表达式〉 * a

⇒〈表达式〉〈运算符〉〈表达式〉 * a

⇒ 〈表达式〉〈运算符〉 a * a

⇒〈表达式〉 + a * a

⇒ a + a * a

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〉 a

⇒〈表达式〉〈运算符〉〈表达式〉 * a

⇒ 〈表达式〉〈运算符〉 a * a

⇒〈表达式〉 + a * a

⇒ a + a * a

第 8 题

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：

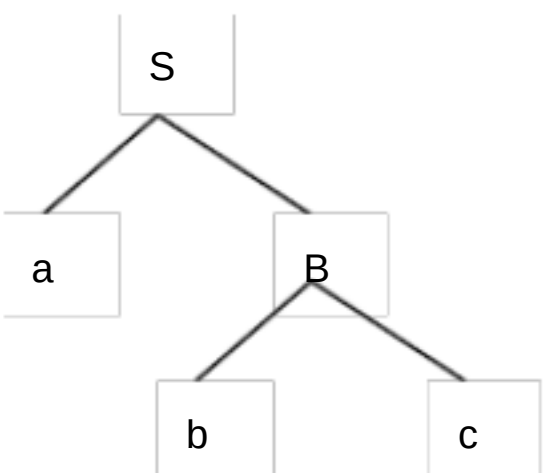
对于串 abc

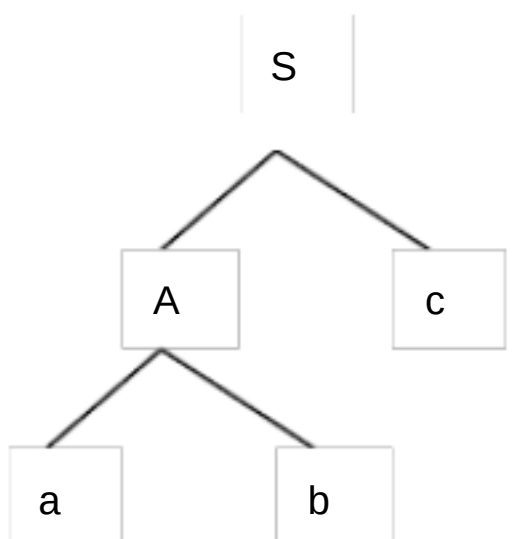
(1)S=>Ac= >abc (2)S= >aB= >abc

即存在两不同的最右推导。所以，该文法是二义的。

或者：

对输入字符串 abc，能构造两棵不同的语法树，所以它是二义的。





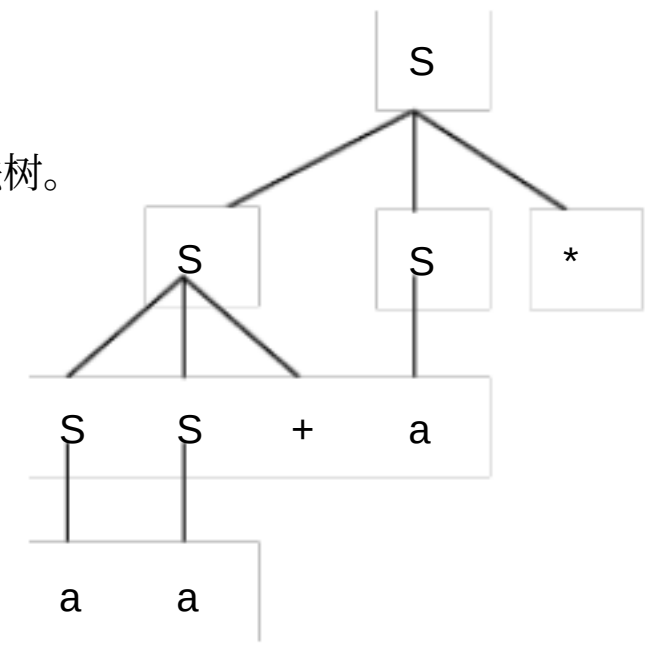
第 9 题

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： * 和 + 的后缀表达式，即逆波兰式。

第 10 题

文法 S→S(S)S|ε

(1) 生成的语言是什么？

(2) 该文法是二义的吗？说明理由。

答案：

（1） 嵌套的括号

（2） 是二义的，因为对于（） （）可以构造两棵不同的语法树。

第 11 题

令文法 G[E] 为：

E→T|E+T|E-T

T→F|T*F|T/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：

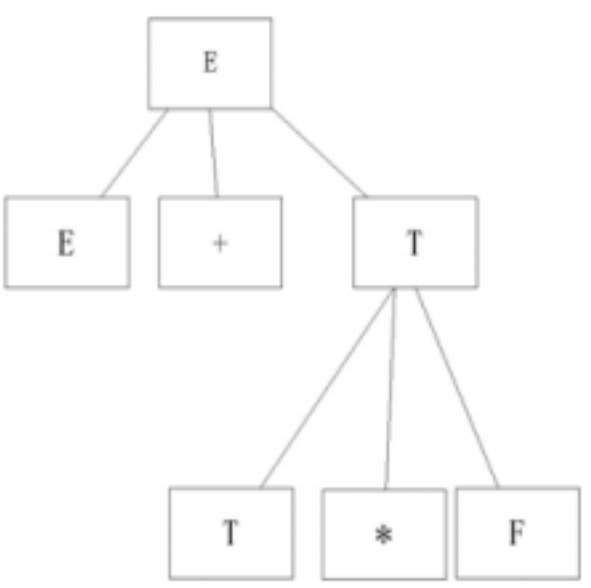
此句型对应语法树如右，故为此文法一个句型。

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



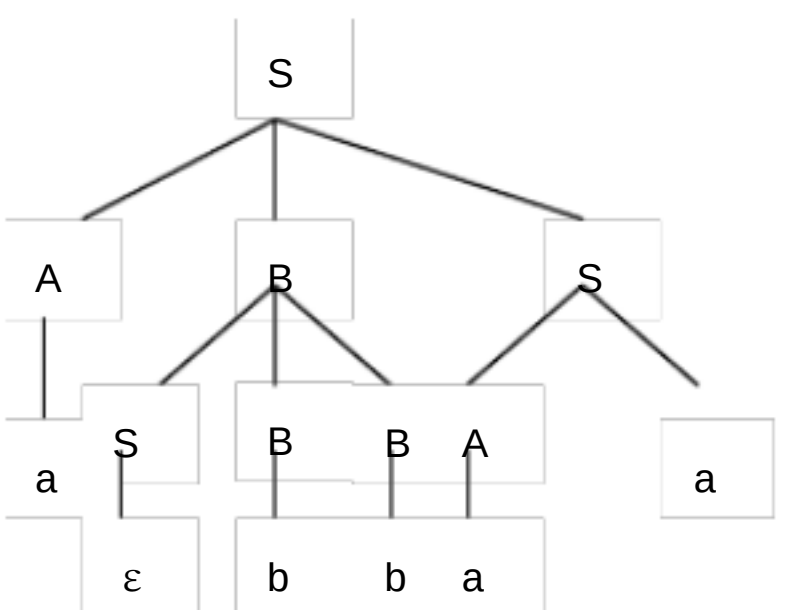
第 13 题

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=>aBBS=>abBS=>abbS=>abbAa=>abbaa

最右推导：

S=>ABS=>ABAa=>ABaa=>ASBBaa=>ASBbaa=>ASbbaa=>Abbaa=>abbaa

(2)产生式有： S→ABS |Aa| ε A→a B →SBB|b

可能元素有： εaa ab abbaa aaabbaa ……

(3)该句子的短语有：

a 是相对 A 的短语

ε是相对 S 的短语

b 是相对 B 的短语

bb 是相对 B 的短语

aa 是相对 S 的短语

abbaa 是相对 S 的短语

直接短语有： a εb

句柄是： a

第 14 题

给出生成下述语言的上下文无关文法：

（1）{ $a^n b^n a^m b^m$ | n, m>= 0}

（2）{ $1^n 0^m 1^m 0^n$ | n, m>=0}

（3）{WaW ^{r} |W 属于 {0|a}* , W^r 表示 W 的逆 }

答案：

（1）

S→AA

A→aAb| ε

（2）

S→1S0|A

A→0A1| ε

（3）

S→0S0|1S1| ε

第 16 题

给出生成下述语言的三型文法：

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

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

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

答案：

(1) $S \rightarrow aS|\varepsilon$

(2)

$S \rightarrow aA$

$A \rightarrow aA|B$

$B \rightarrow bB|b$

(3)

$A \rightarrow aA|B$

$B \rightarrow bB|C$

$C \rightarrow cC|\varepsilon$

第 17 题

习题 7 和习题　11 中的文法等价吗？

答案：

等价。

第 18 题

解释下列术语和概念：

（1）　字母表

（2）　串、字和句子

（3）　语言、语法和语义

答案：

（1）字母表：是一个非空有穷集合。

（2）串：符号的有穷序列。

字：字母表中的元素。

句子：如果　$Z \overset{+}{\Rightarrow} x, x \in V^*_T$　则称 x 是文法 G 的一个句子。

（3）语言：它是由句子组成的集合，是由一组记号所构成的集合。程序设计的语言就是所有该语言的程序的全体。 语言可以看成在一个基本符号集上定义的， 按一定规则构成的一切基本符号串组成的集合。

语法：表示构成语言句子的各个记号之间的组合规律。程序的结构或形式。

语义：表示按照各种表示方法所表示的各个记号的特定含义。语言所代表的含义。

附加题

问题 1：

给出下述文法所对应的正规式：

S→0A|1B

A→1S|1

B→0S|0

答案：

R = (01 | 10) (01 | 10)*

问题 2：

已知文法 G[A]，写出它定义的语言描述

G[A]: A → 0B|1C

B → 1|1A|0BB

C → 0|0A|1CC

答案 :

G[A] 定义的语言由 0、1 符号串组成，串中 0 和 1 的个数相同 .

问题 3：

给出语言描述，构造文法 .

构造一文法 ,其定义的语言是由算符 +, *, (,) 和运算对象 a 构成的算术表达式的集合 .

答案一：

G[E] E →E+T|T

T→ T* F|F

F→ (E)|a

答案二：

G[E] E →E+E|E* E|(E)|a

问题 4：

已知文法 G[S]:

S→dAB

A→aA|a

B→ε |bB

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

问题 5：

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：

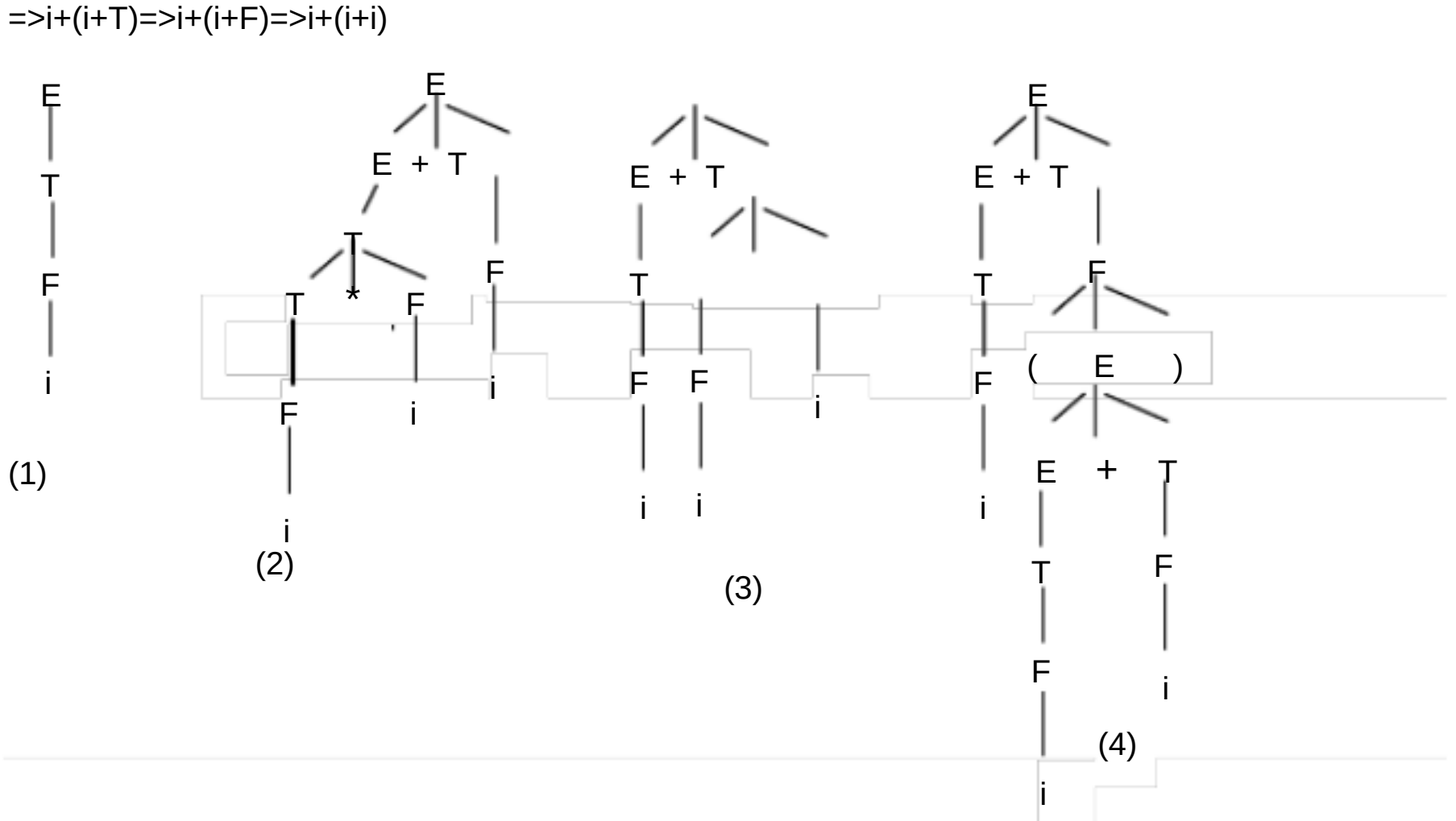
(1)E=>T=>F=>i

(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)



问题 6：

已知文法 G[E]:

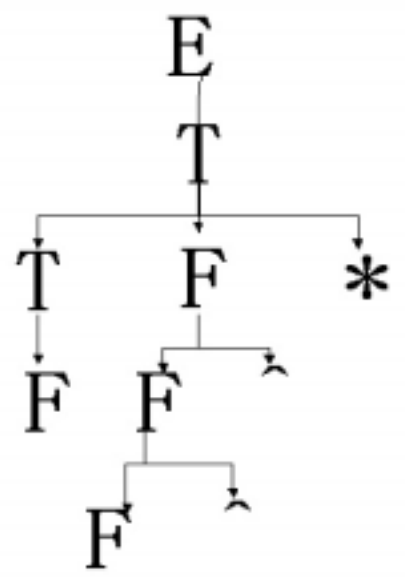
E→ET+|T

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.

问题 7：

适当变换文法，找到下列文法所定义语言的一个无二义的文法：

S → SaS？SbS？ScS？ d

答案：

该文法的形式很典型， 可以先采用优先级联规则变换文法， 然后再规定结合性对文法做进一步变换，即可消除二义性。

设 a、b 和 c 的优先级别依次增高，根据优先级联规则将文法变换为：

S → SaS？A

A → AbA？C

C → CcC？d

规定结合性为左结合，进一步将文法变换为：

S → SaA? A

A → AbC？C

C → CcF？F

F → d

该文法为非二义的。

问题 8：

构造产生如下语言的上下文无关文法：

（1） $\{a^n b^{2n} c^m \mid n, m \geq 0\}$

（2） $\{a^n b^m c^{2m} \mid n, m \geq 0\}$

（3） $\{ a^m b^n ? m \geq n \}$

（4） $\{ a^m b^n c^p d^q ? m+n = p+q \}$

（5） { uawb ? u,w ∈{a, b}* ∧| u | = | w | }

答案：

（1） 根据上下文无关文法的特点，要产生形如 $a^n b^{2n} c^m$的串，可以分别产生形如 $a^n b^{2n}$ 和形如 c^m 的串。设计好的文法是否就是该语言的文法？严格地说，应该给出证明。但若不是特别指明，通常可以忽略这一点。

对于该语言，存在一个由以下产生式定义的上下文无关文法 G[S]：

S → AB

A → ε? aAbb

B → ε? cB

（2）同样，要产生形如 $a^n b^m c^{2m}$的串，可以分别产生形如 a^n 和形如 $b^m c^{2m}$ 的串。对于该语言，存在一个由以下产生式定义的上下文无关文法 G[S]：

S → AB

A → ε? aA

B → ε? bBcc

（3）考虑在先产生同样数目的 a,b，然后再生成多余的 a。以下 G[S] 是一种解法：

S → aSb？aS？ ε

（4）以下 G[S] 是一种解法：

S → aSd？A？D

A → bAd？B

D → aDc ？B

B → bBc？ ε

注：a 不多于 d 时， b 不少于 c；反之， a 不少于 d 时， b 不多于 c。前一种情形通过对应 A，后一种情形对应 D。

（5）以下 G[S] 是一种解法：

S → Ab

A → BAB ？a

B → a ? b

问题 9：

下面的文法 G(S)描述由命题变量 p、q ，联结词 ∧（合取）、∨（析取）、？（否定）构成的命题公式集合：

S → S ∨T ？T

T → T ∧F ？F

F → ？ F？ p？ q

试指出句型 ？ F ∨? q ∧p 的直接短语（全部）以及句柄。

答案：

直接短语： p， q， ？F

句柄： ？F

问题 10：

设字母表 A={a} ，符号串 x=aaa，写出下列符号串及其长度： x^0,xx,x^5 以及 A^+.

答案：

$x^0=(aaa)^0=\varepsilon$ $|x^0|=0$

xx= aaaaaa |xx|= 6

x^5=aaaaaaaaaaaaaaa $|x^5|=15$

$A^+=A^1 \cup A^2 \cup \cdots . \cup A^n \cup \cdots$ ={a,aa,aaa,aaaa,aaaaa ⋯ }

$A^* = A^0 \cup A^1 \cup A^2 \cup \cdots . \cup A^n \cup \cdots$ ={ ε ,a,aa,aaa,aaaa,aaaaa ⋯ }

问题 11：

令∑ ={a， b， c}，又令 x=abc， y=b， z=aab，写出如下符号串及它们的长度： xy， xyz，（xy）3

答案：

xy= abcb |xy|= 4

xyz= abcbaab |xyz|= 7

$(xy)^3=(abcb)^3$ = abcbabcbabcb | $(xy)^3$|=12

问题 12：

已知文法 G[Z] ：Z∷=U0 丨V1 、 U∷=Z1 丨1 、 V∷=Z0 丨0 ,请写出全部由此文法描述的只含有四个符号的句子。

答案：

Z=>U0=>Z10=>U010=>1010
Z=>U0=>Z10=>V110=>0110
Z=>V1=>Z00=>U000=>1000
Z=>V1=>Z00=>V100=>0100

问题 13：

已知文法 G[S]： S∷=AB A ∷=aA | ε B∷=bBc | bc，写出该文法描述的语言。

答案：

A∷=aA | ε 描述的语言 ：$\{a^n|n>=0\}$
B∷=bBc | bc描述的语言 :$\{,b^nc^n|n>=1\}$
L(G[S])=$\{a^nb^mc^m|n>=0,m>=1\}$

问题 14：

已知文法 E∷=T | E+T | E-T 、 T∷=F | T*F | T/F 、 F∷=(E) | i，写出该文法的开始符号、终结符号集合 V_T、非终结符号集合 V_N。

答案：

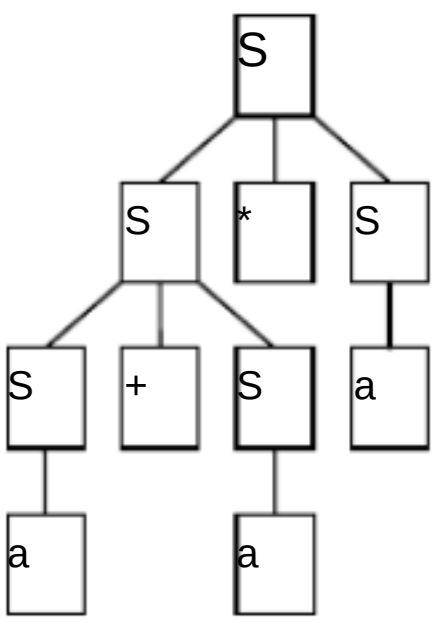
开始符号： E
V_T={+, - , * , / , （ , ） , i}
V_N={E , F , T}

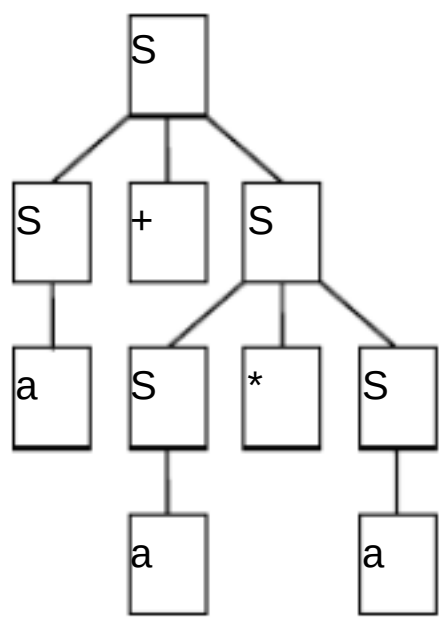
问题 15：

设有文法 G[S] ：S∷=S*S|S+S|(S)|a，该文法是二义性文法吗？

答案：

根据所给文法推导出句子 a+a*a，画出了两棵不同的语法树， 所以该文法是二义性文法。





问题 16：

写一文法，使其语言是奇正整数集合。

答案：

A::=1|3|5|7|9|NA
N::=N0|N1|N2|N3|N4|N5|N6|N7|N8|N9|
N::=0|1|2|3|4|5|6|7|8|9

第 4 章　词法分析

第 1 题

构造下列正规式相应的　DFA.

（1）　1(0|1)　*101

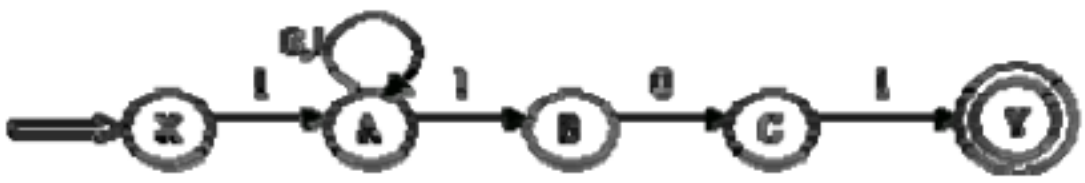
（2）　1（1010*|1(010)*1 ）*0

（3）　a((a|b)*|ab*a)*b

（4）　b((ab)*|bb)*ab

答案：

(1) 先构造　NFA ：



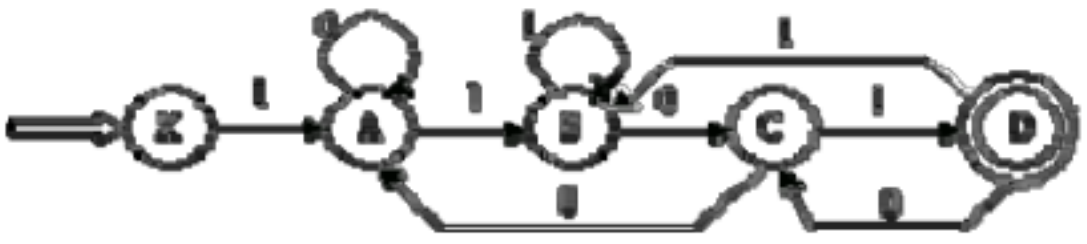
用子集法将　NFA 确定化

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

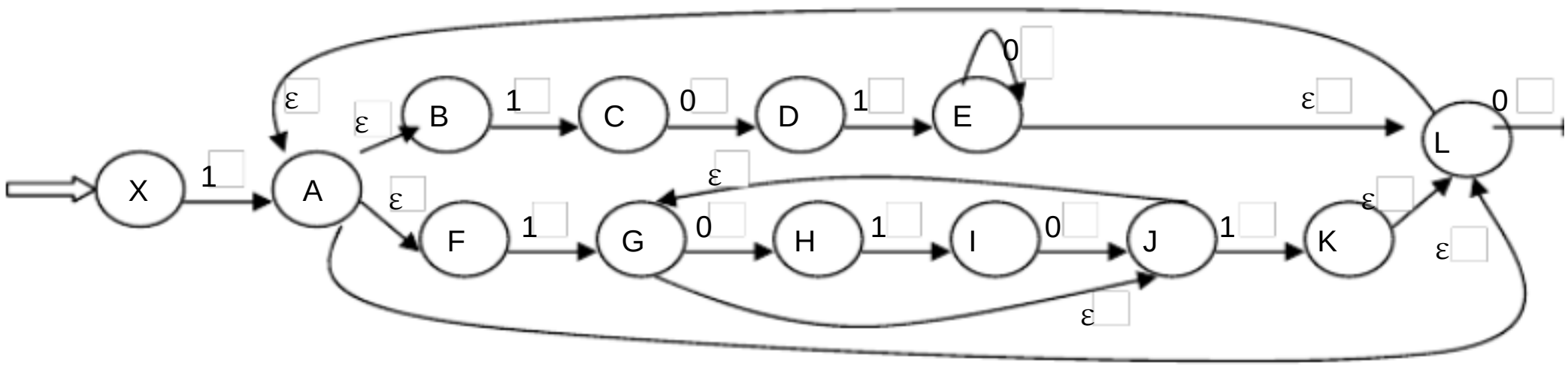
除 X，A 外，重新命名其他状态，令　AB 为 B、AC 为 C、ABY 为 D，因为 D 含有 Y（NFA 的终态），所以　D 为终态。

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

DFA 的状态图：：



(2)先构造 NFA：

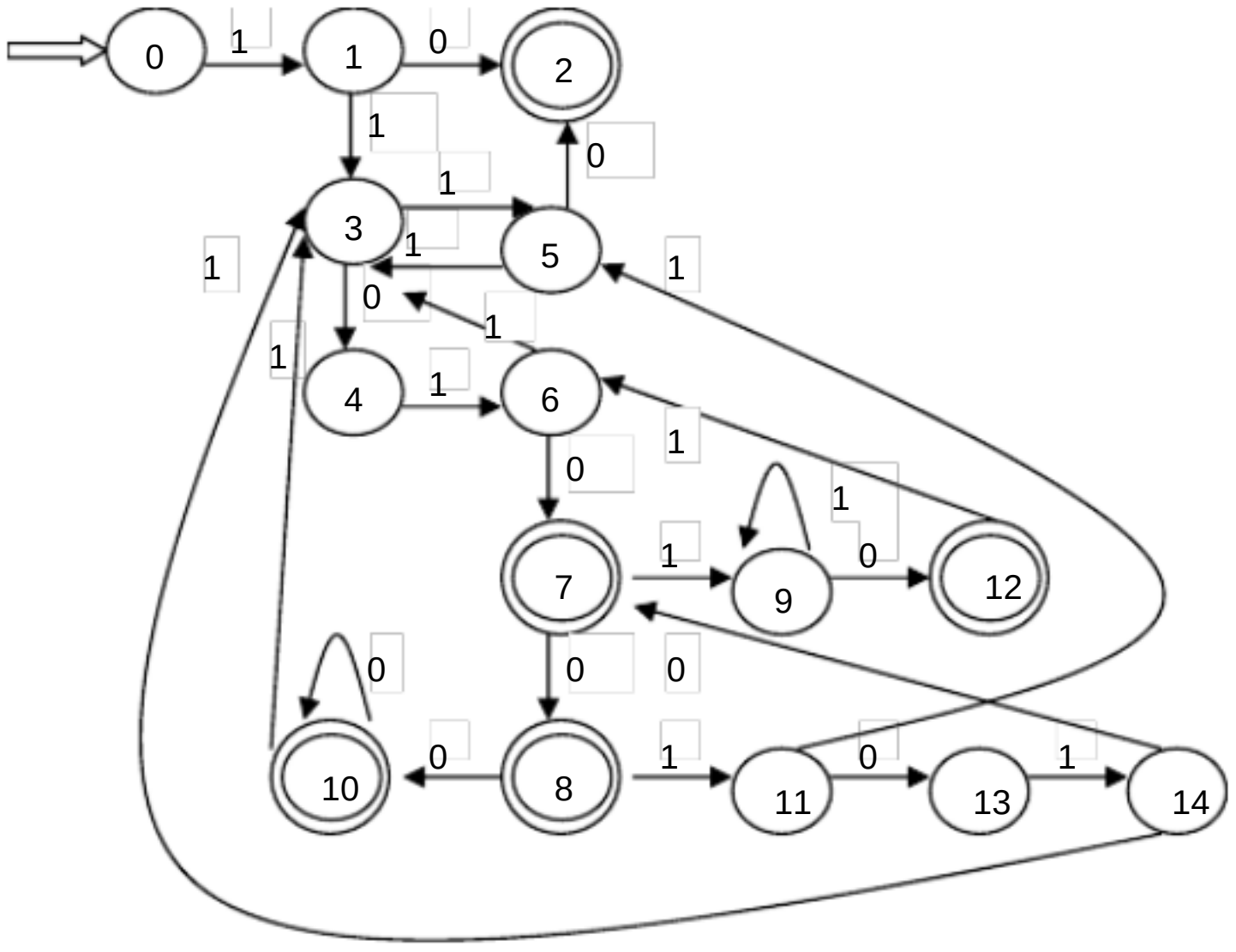


用子集法将 NFA 确定化

	ε 0		1
X X			
T_0 X			A
A ABFL			
T_1= ABFL		Y	CG
Y Y			
CG CGJ			
T_2= Y			
T_3= CGJ		DH	K
DH DH			
K ABFKL			
T_4= DH			EI
EI ABEFIL			
T_5= ABFKL		Y	CG
T_6= ABEFIL		EJY	CG
EJY ABEFGJLY			
T_7= ABEFGJLY		EHY	CGK
EHY ABEFHLY			
CGK ABCFGJKL			
T_8= ABEFHLY		EY	CGI
EY ABEFLY			
CGI CGJI			
T_9= ABCFGJKL		DHY	CGK
DHY DHY			
T_{10}= ABEFLY		EY	CG
T_{11}= CGJI		DHJ	K
DHJ DHJ			
T_{12}= DHY			EI
T_{13}= DHJ			EIK
EIK ABEFIKL			
T_{14}= ABEFIKL		EJY	CG

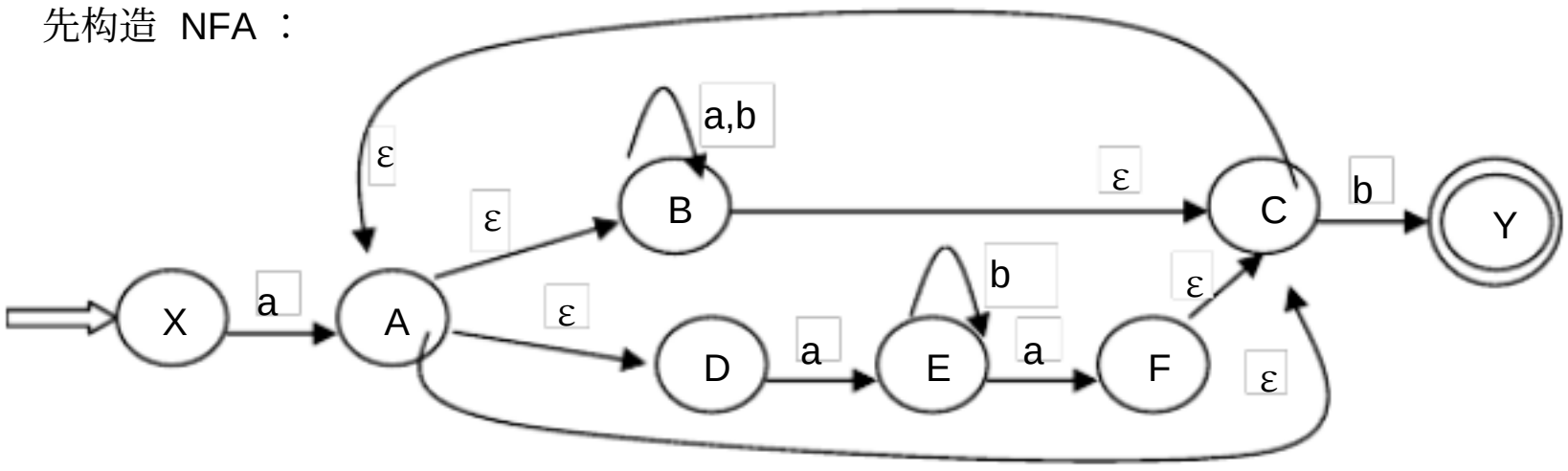
将T_0、T_1、T_2、T_3、T_4、T_5、T_6、T_7、T_8、T_9、T_{10}、T_{11}、T_{12}、T_{13}、T_{14}重新命名，分别用 0、1、2、3、4、5、6、7、8、9、10、11、12、13、14 表示。因为 2、7、8、10、12 中含有 Y，所以它们都为终态。

0		1
0		1
1 2 3		
2		
3 4 5		
4		6
5 2 3		
6 7 3		
7 8 9		
8 10		11
9 12		9
10 10 3		
11 13 5		
12		6
13		14
14 7		3



(3) 先构造 NFA：

先构造 NFA ：

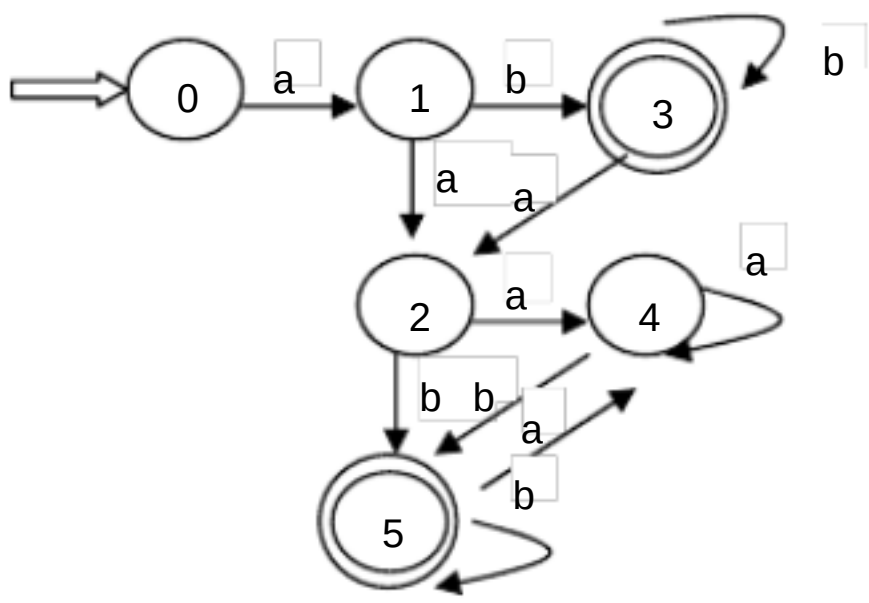


用子集法将 NFA 确定化

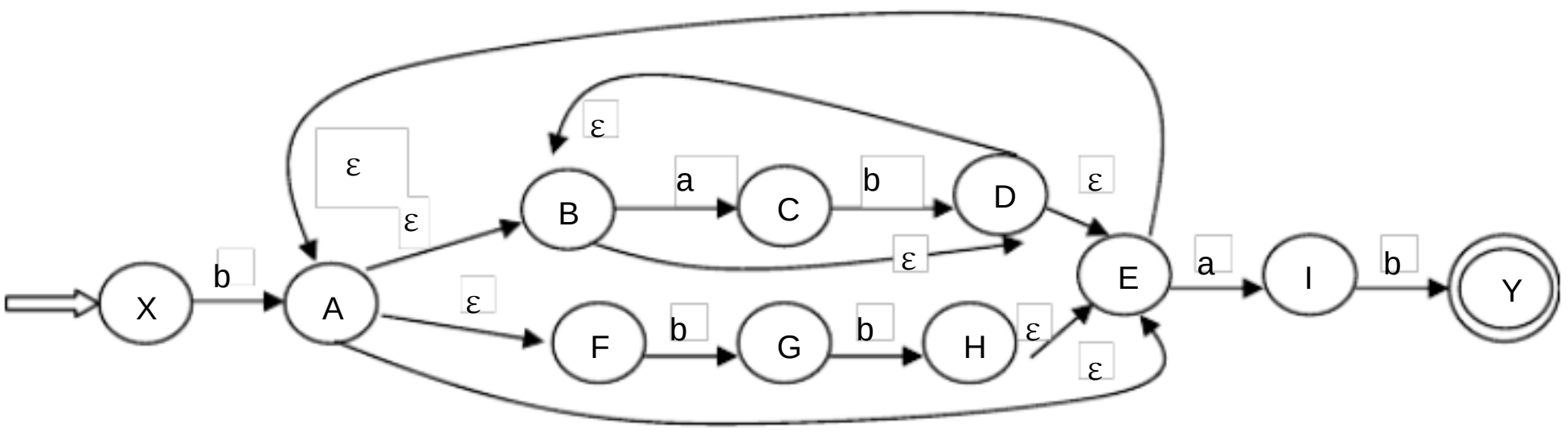
	ε a		b
X X			
T_0 X		A	
A ABCD			
T_1 ABCD		BE	BY
BE ABCDE			
BY ABCDY			
T_2 ABCDE		BEF	BEY
BEF ABCDEF			
BEY ABCDEY			
T_3 ABCDY		BE	BY
T_4 ABCDEF		BEF	BEY
T_5 ABCDEY		BEF	BEY

将T_0、T_1、T_2、T_3、T_4、T_5重新命名，分别用 0、1、2、3、4、5 表示。因为 3、5 中含有 Y，所以它们都为终态。

a		b
0 1		
1 2 3		
2 4 5		
3 2 3		
4 4 5		
5 4 5		



(4) 先构造 NFA：



=

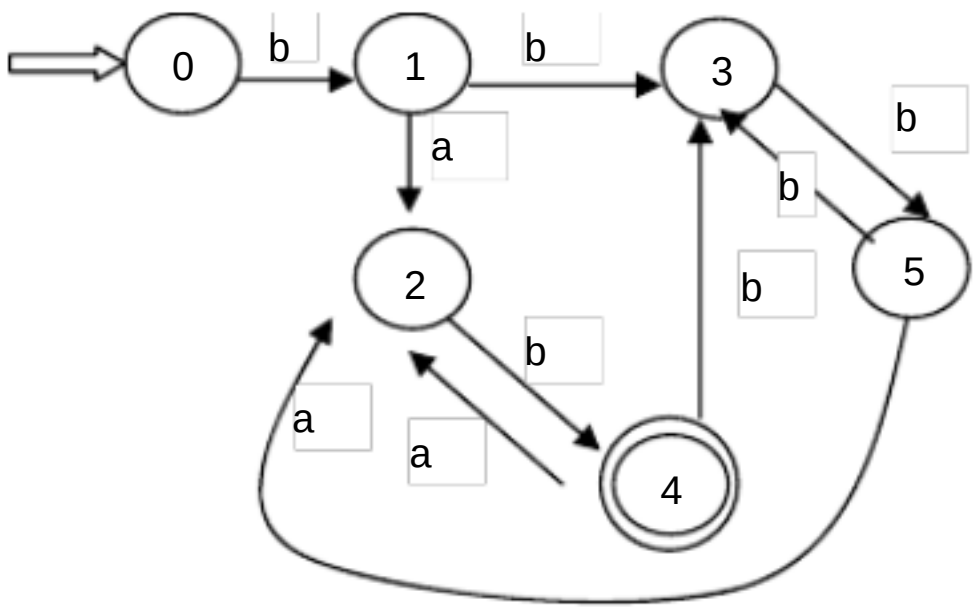
用子集法将 NFA 确定化：

	ε a		b
X X			
T_0 X			A
A ABDEF			
T_1 ABDEF		CI	G
CI CI			
G G			
T_2 CI			DY
DY ABDEFY			
T_3 G			H
H ABEFH			
T_4 ABDEFY		CI	G
T_5 ABEFH		CI	G

将T_0、T_1、T_2、T_3、T_4、T_5重新命名，分别用 0、1、2、3、4、5 表示。因为 4 中含有 Y，所以它为终态。

a		b
0		1
1 2 3		
2		4
3		5
4 2 3		
5 2 3		

DFA 的状态图：


0
1
2
3
4
5
b
b
a
b
b
b
b
a
a

第 2 题

已知 NFA ＝（{x,y,z},{0,1},M,{x},{z} ），其中：M(x,0)={z} ，M(y,0)={x,y} ，,M(z,0)={x,z} ，M(x,1)={x} ， M(y,1)= ϕ， M(z,1)={y} ，构造相应的 DFA。

答案：

先构造其矩阵

0		1
x z x		
y x,y		
z x,z		y

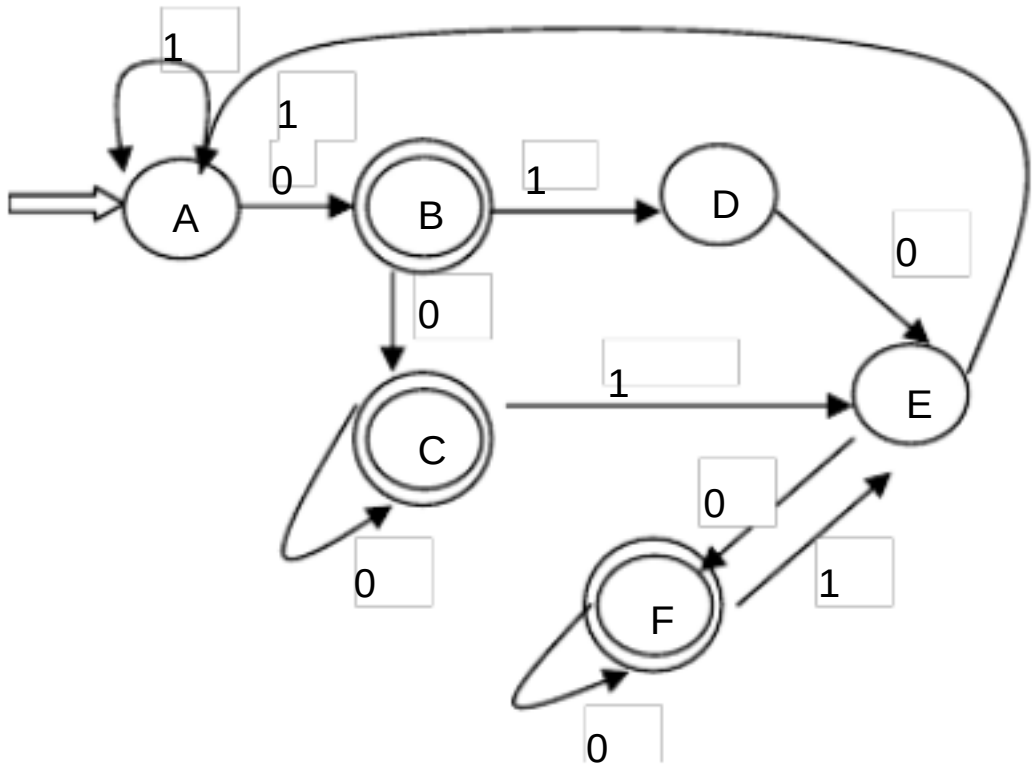
用子集法将 NFA 确定化：

0		1
x	z	x
z xz		y
xz xz xy		
y xy		
xy xyz		x
xyz xyz xy		

将 x、 z、xz、y、xy 、xyz 重新命名，分别用 A、B、C、D、E、 F 表示。因为 B、C、F 中含有 z，所以它为终态。

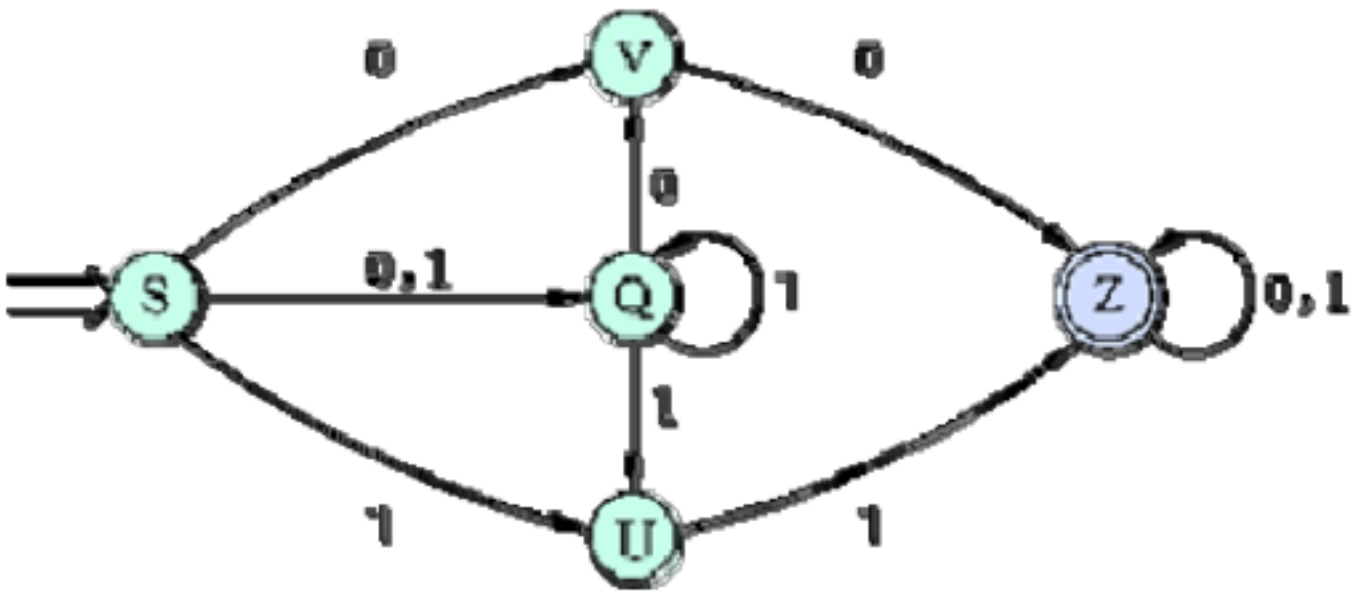
0		1
A B A		
B C D		
C C E		
D E		
E F A		
F F E		

DFA 的状态图：



第 3 题

将下图确定化：



答案：

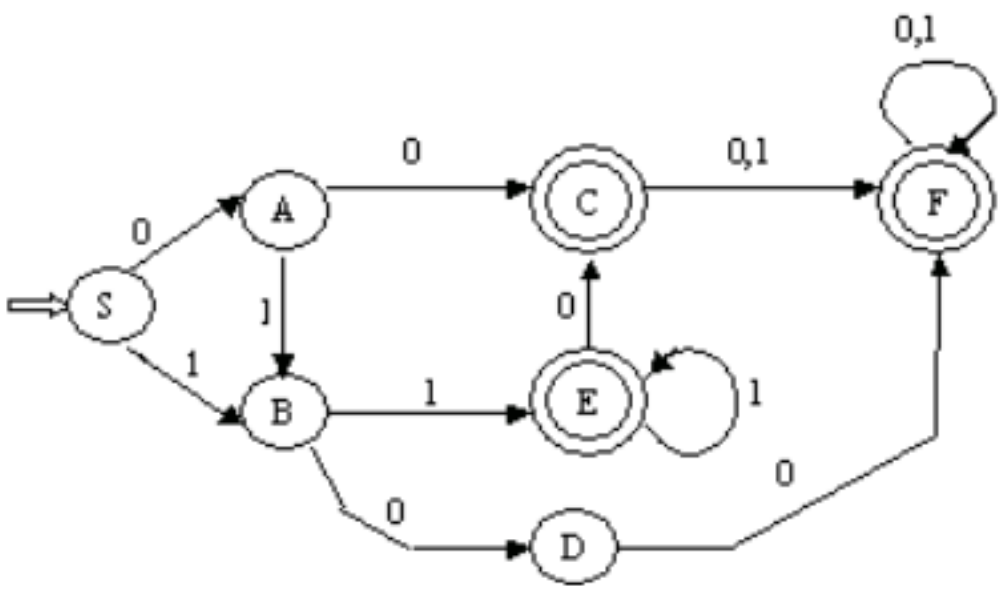
用子集法将　NFA 确定化：

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

重新命名状态子集，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

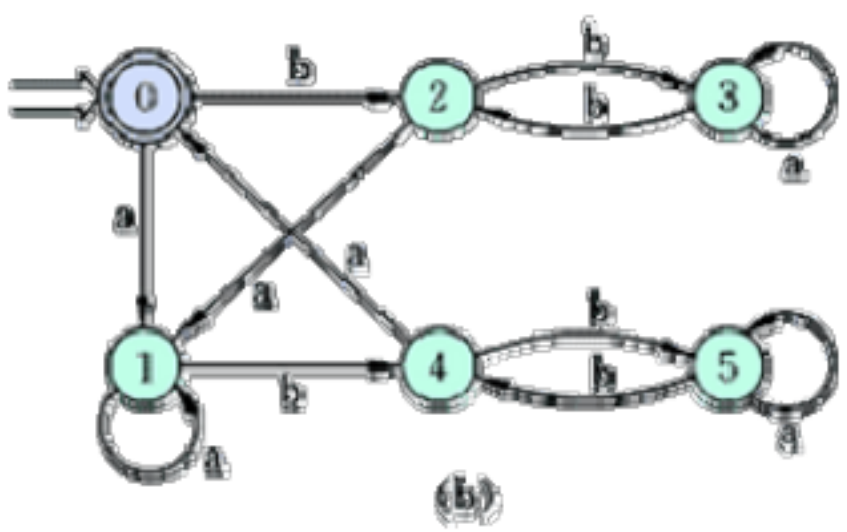
DFA 的状态图：



第 4 题

将下图的（ a）和（ b）分别确定化和最小化：

(a)



(b)

答案：

初始分划得

Π0：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 ⊄{0,1,3,5}

而{0,1,3,5} 既不属于 {0}， 也不属于 {1,2,3,4,5}

∵{4} a ⊂{0} ， 所以得到新分划

Π1：{0} ， {4} ， {1,2,3,5}

对{1,2,3,5} 进行审查：

∵{1,5} b ⊂{4}

{2,3} b ⊄{1,2,3,5} ， 故得到新分划

Π2：{0} ， {4} ， {1, 5} ， {2,3}

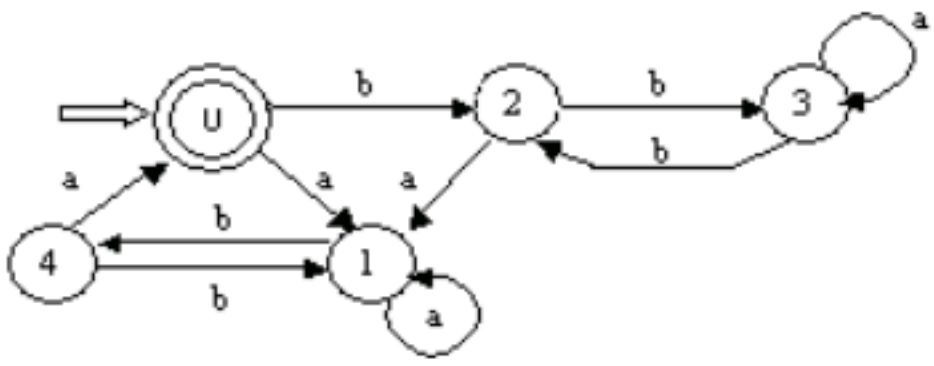
{1, 5} a ⊄{1, 5}

{2,3} a ⊄{1,3} ， 故状态 2 和状态 3 不等价，得到新分划

Π3：{0} ， {2} ， {3} ， {4} ， {1, 5}

这是最后分划了

最小 DFA ：

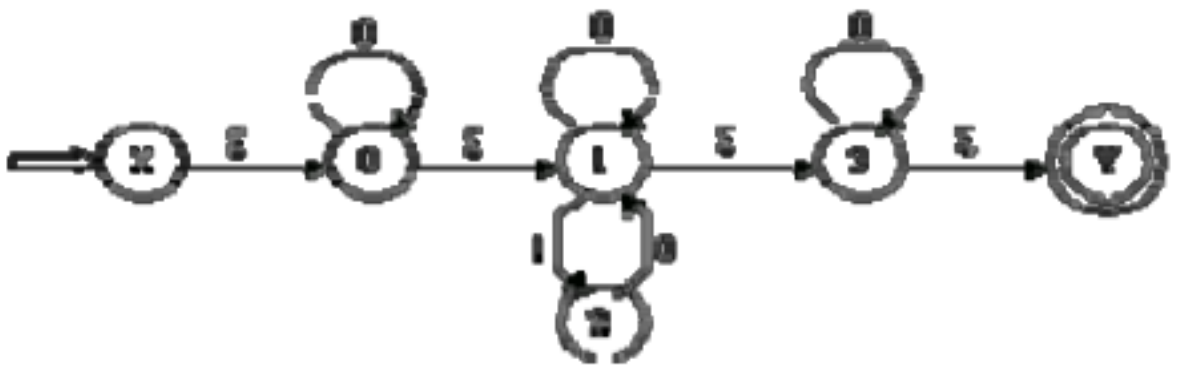


第 5 题

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。

答案：

按题意相应的正规表达式是 (0*10)*0* ，或 0*(0 | 10)*0* 构造相应的 DFA，首先构造 NFA 为



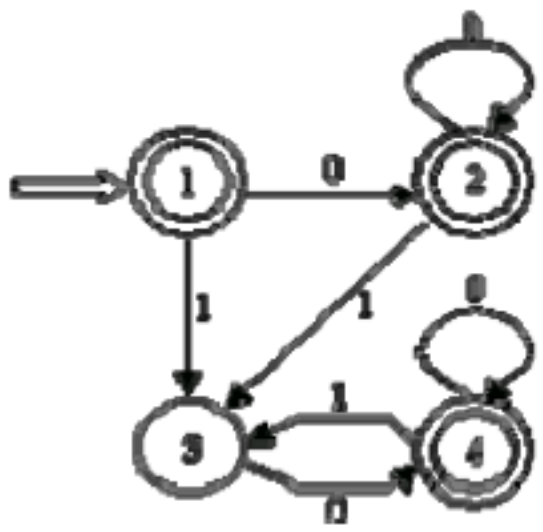
用子集法确定化：

I I0		I1
{X,0,1,3,Y}	{0,1,3,Y}	{2}
{0,1,3,Y}	{0,1,3,Y}	{2}
{2}	{1,3,Y}	
{1,3,Y}	{1,3,Y}	{2}

重新命名状态集：

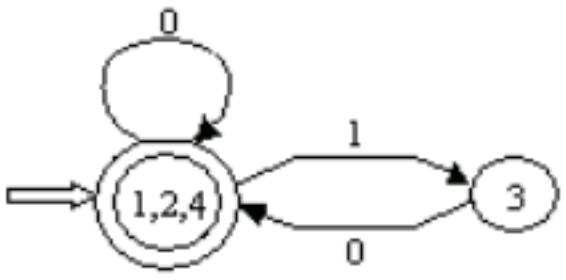
S 0		1
1	2	3
2	2	3
3	4	
4	4	3

DFA 的状态图：



可将该 DFA 最小化：

终态组为 {1,2,4}，非终态组为 {3}，{1,2,4}0 {1,2,4} ，{1,2,4}1 {3} ，所以 1,2,4 为等价状态，可合并。



第 6 题

设无符号数的正规式为 θ ：

θ =dd*|dd*.dd*|.dd*|dd*10(s| ε)dd*

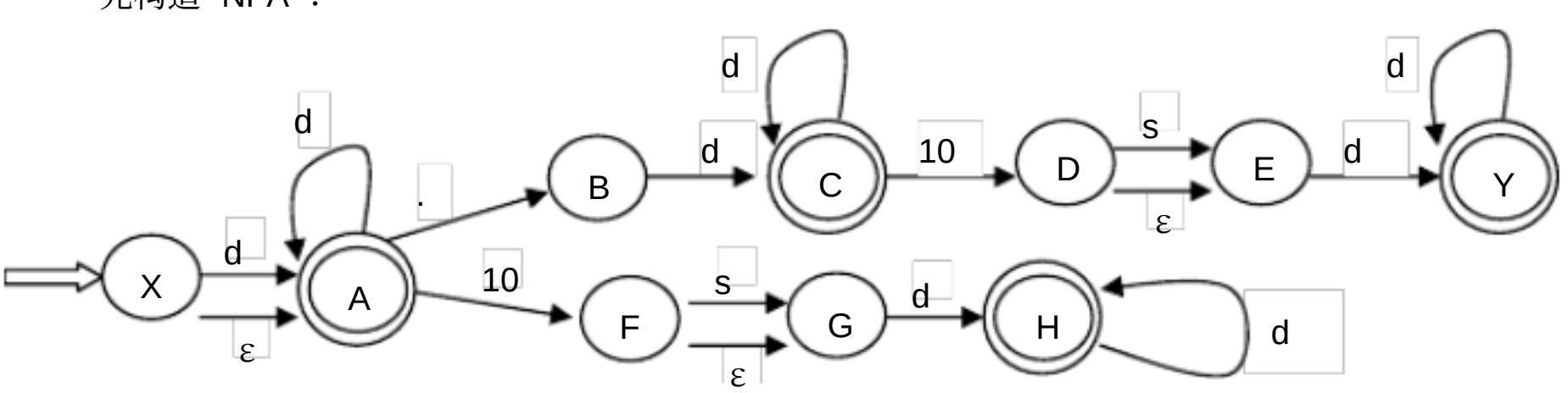
|10(s| ε)dd*| . dd* 10(s| ε)dd*

|dd* . dd* 10(s| ε)dd*

化简 θ，画出 θ 的 DFA，其中 d={0,1,2, …,9} ， s={ ＋， － }

答案：

先构造 NFA ：



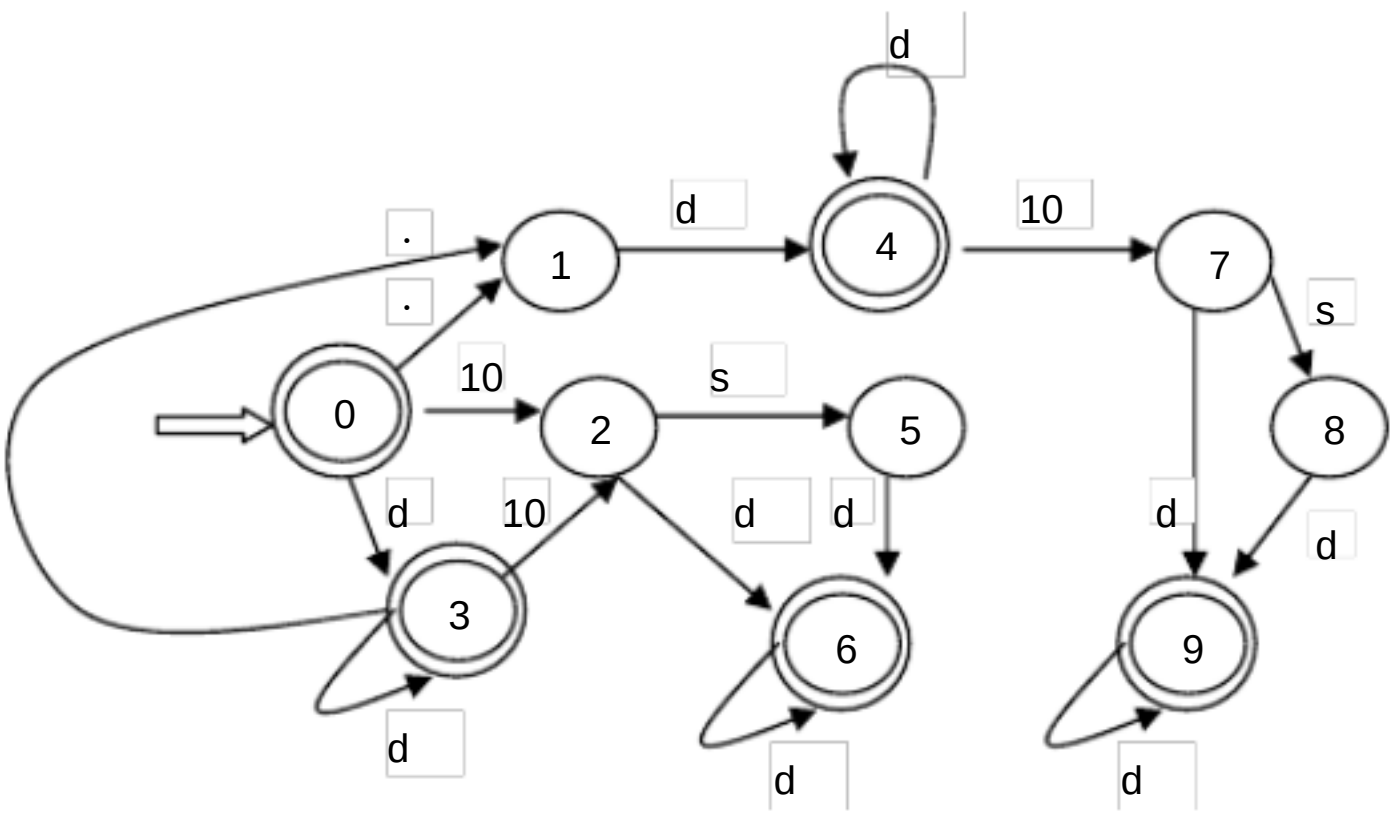
用子集法将 NFA 确定化：

	ε	· s		10	d
X XA					
T0 XA		B		F	A
B B					
F FG					
A A					
T1 B				C	
C C					
T2 FG			G		H
G G					
H H					
T3 A		B		F	A
T4 C				D	C
D DE					
T5 G					H
T6 H					H
T7 DE			E		Y
E E					
Y Y					
T8 E				Y	
T9 Y					Y

将 XA 、B、FG、A、C、G、H、DE、E、Y 重新命名，分别用 0、1、2、3、4、5、6、7、8、9 表示。终态有 0、3、4、6、9。

	· s		10	d
0 1			2	3
1				4
2		5		6
3 1			2	3
4			7	4
5				6
6				6
7		8		9
8				9
9				9

DFA 的状态图：



第 7 题

给文法 G[S] ：

S→aA|bQ

A→aA|bB|b

B→bD|aQ

Q→aQ|bD|b

D→bB|aA

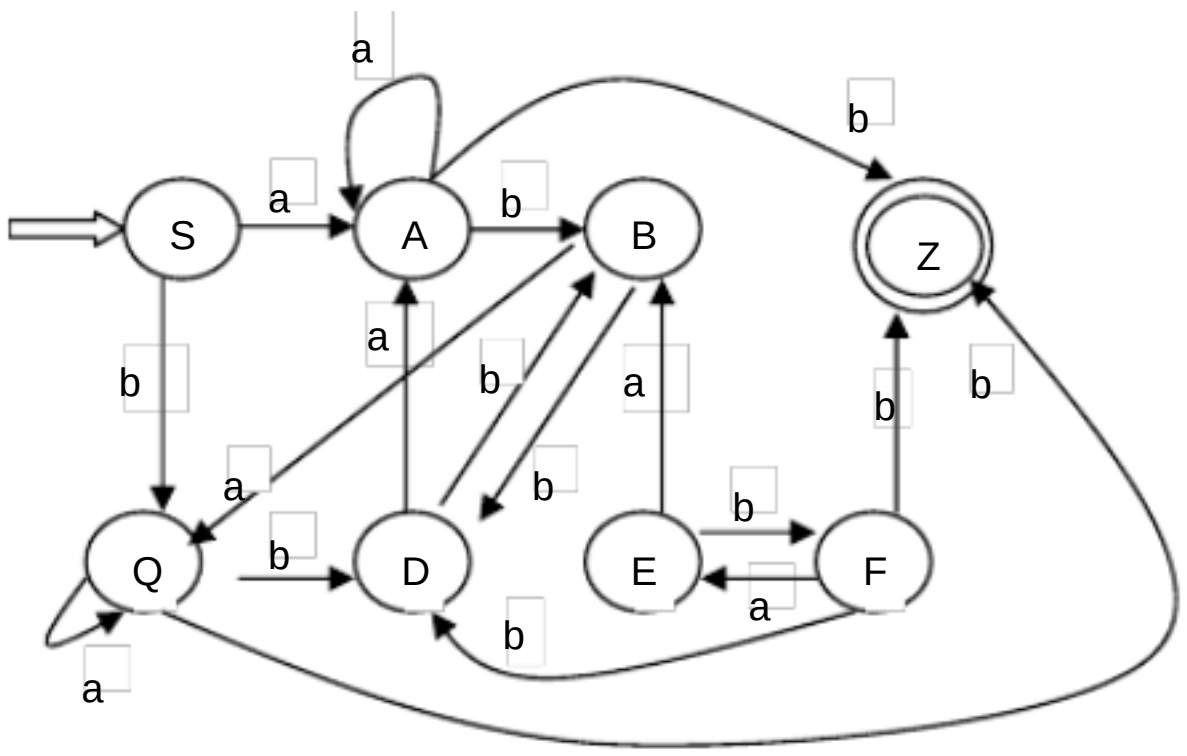
E→aB|bF

F→bD|aE|b

构造相应的最小的 DFA 。

答案：

先构造其 NFA ：



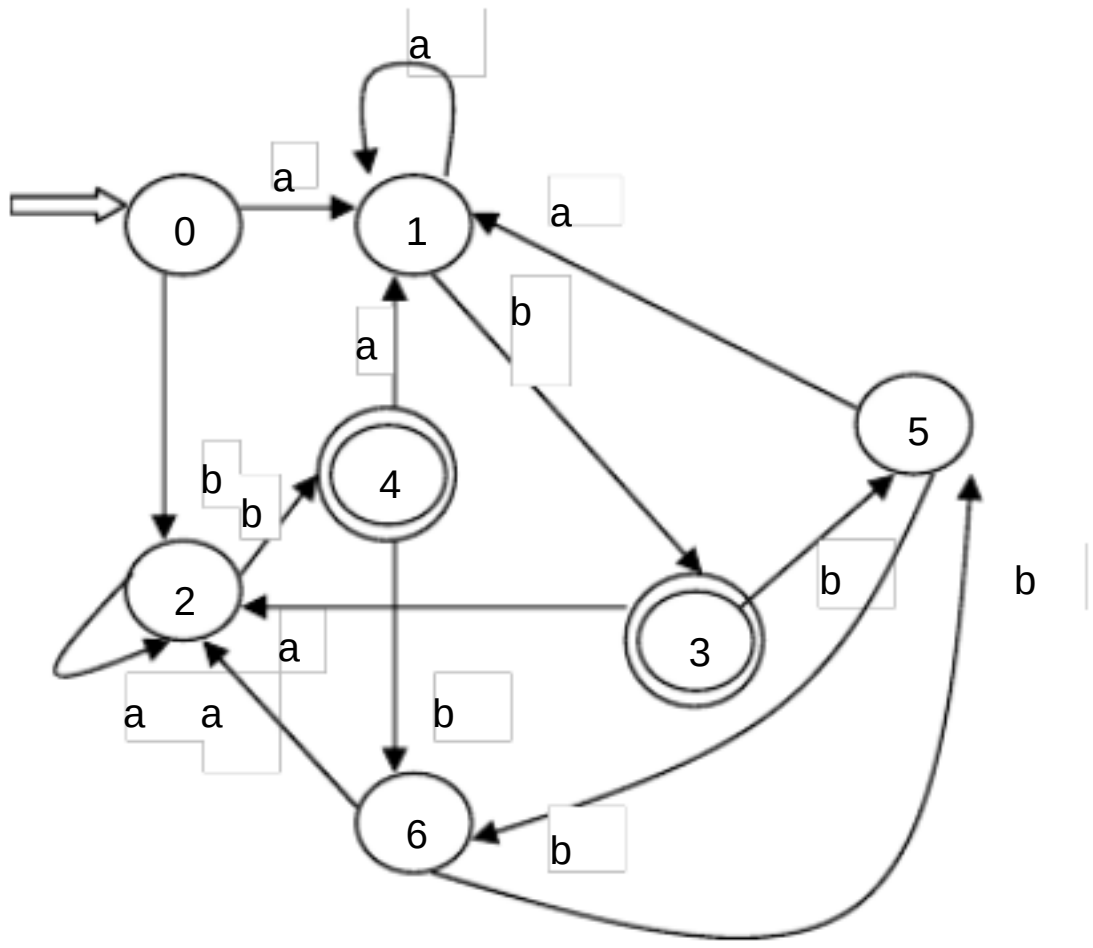
用子集法将 NFA 确定化：

a		b
S A		Q
A A BZ		
Q Q DZ		
BZ Q		D
DZ A		B
D A B		
B Q D		

将 S、A、Q、BZ、DZ、D、B 重新命名，分别用 0、1、2、3、4、5、6 表示。因为 3、4 中含有 z，所以它们为终态。

a		b
0 1 2		
1 1 3		
2 2 4		
3 2 5		
4 1 6		
5 1 6		
6 2 5		

DFA 的状态图：



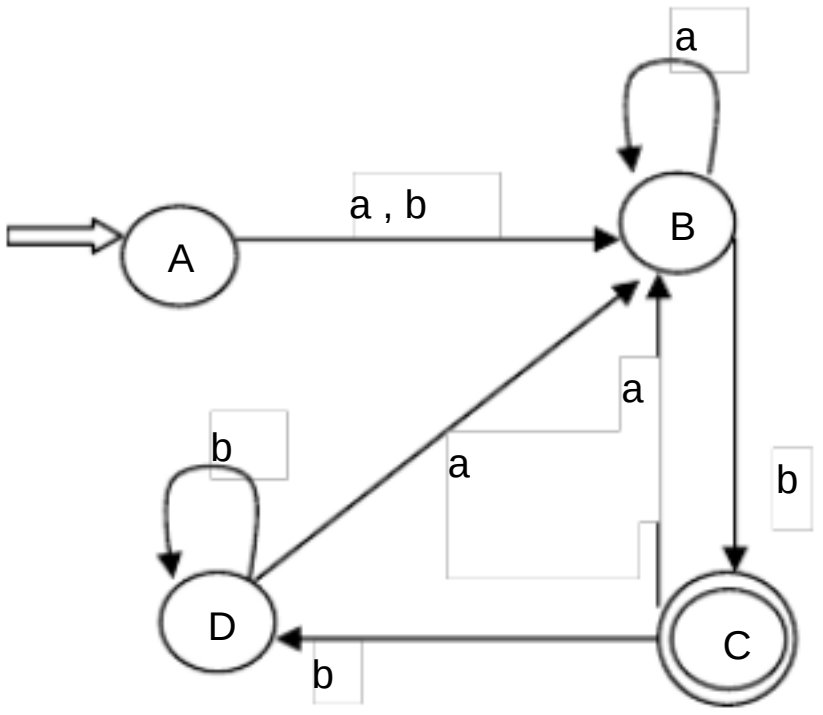
令P_0=（｛0,1,2,5,6｝，｛3,4｝）用 b进行分割：

P_1=（｛0,5, 6｝，｛1,2｝，｛3,4｝）再用 b进行分割：

P_2=（｛0｝，｛5, 6｝，｛1,2｝，｛3,4｝）再用 a、b 进行分割，仍不变。

再令｛0｝为 A，｛1，2｝为 B，｛3，4｝为 C，｛5，6｝为 D。

最小化为：



第 8 题

给出下述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

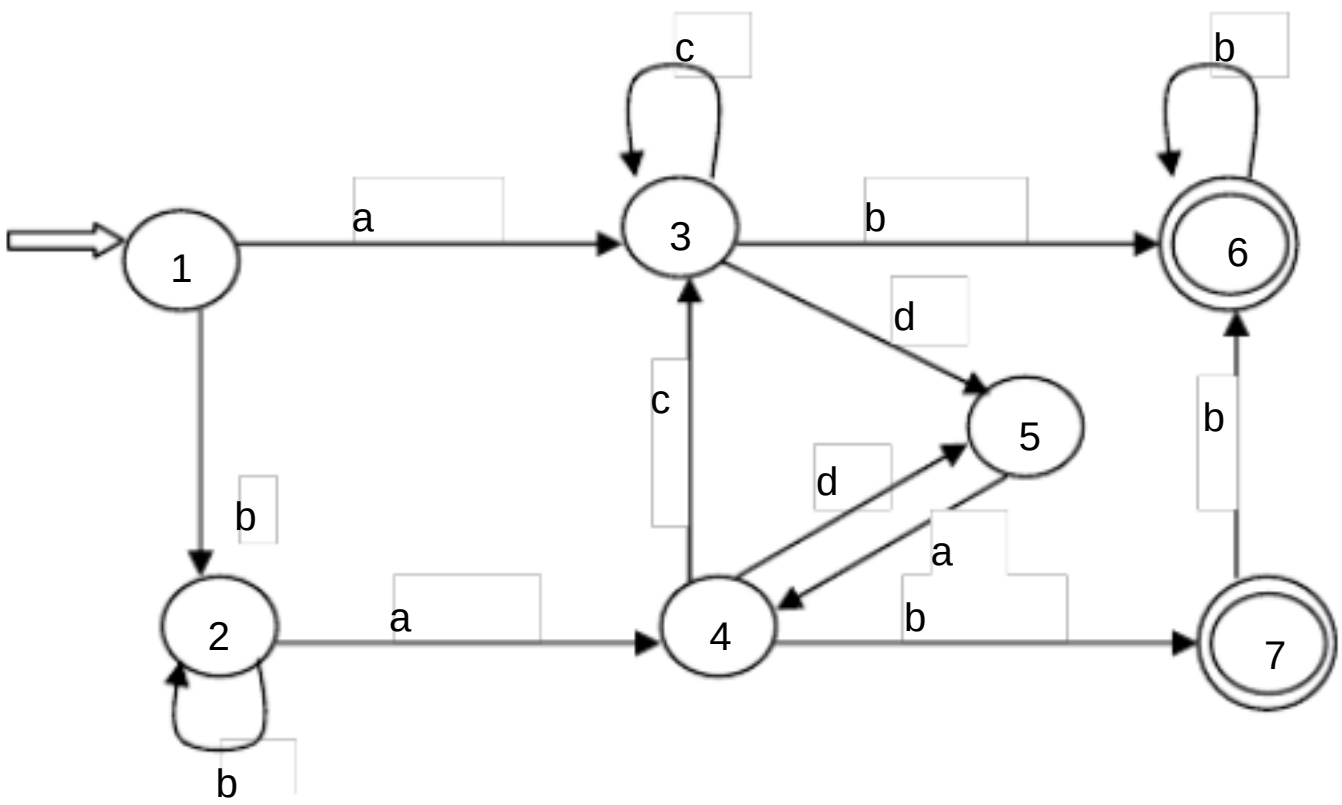
将 A、B 产生式的右部代入 S 中

S=01S|01|10S|10=（01|10）S|（01|10）

所以： S= (01|10)*(01|10)

第 9 题

将下图的 DFA 最小化，并用正规式描述它所识别的语言。



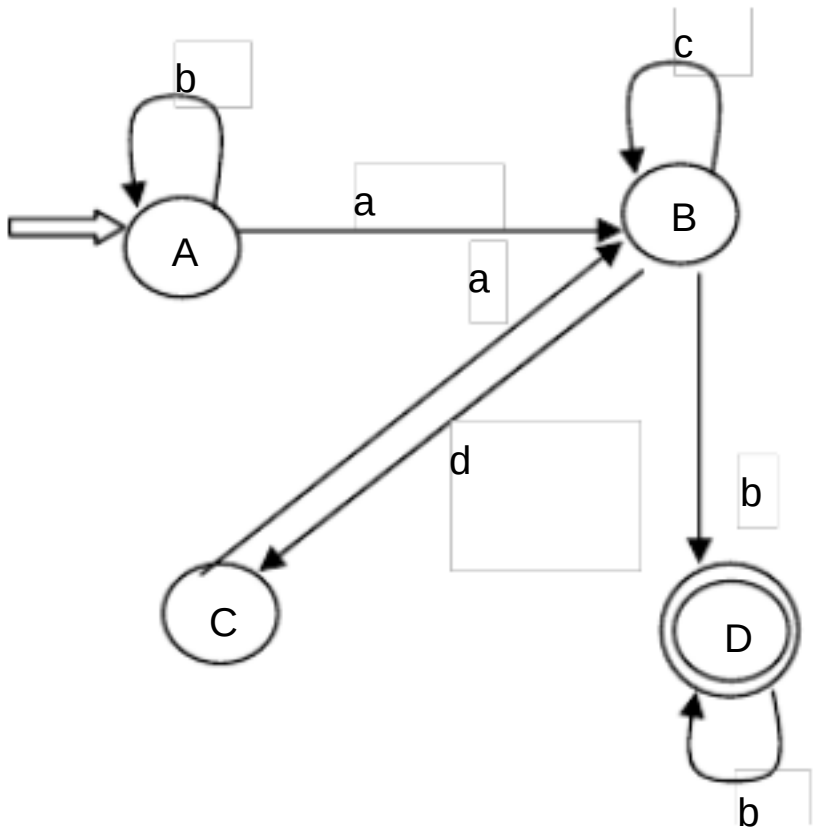
答案：

令P_0=（｛1,2,3,4,5｝，｛6,7｝）用 b进行分割：

P_1=（｛1,2｝，｛3,4｝，｛5｝，｛6,7｝）再用 a、b、c、d进行分割，仍不变。

再令｛1,2｝为 A，｛3，4｝为 B，｛5｝为 C，｛6，7｝为 D。

最小化为：



$r=b^{*}a(c|da)^{*}bb^{*}=b^{*}a(c|da)^{*}b^{+}$

附加题

问题 1：

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*

问题 2：

请描述下面正规式定义的串 . 字母表 {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

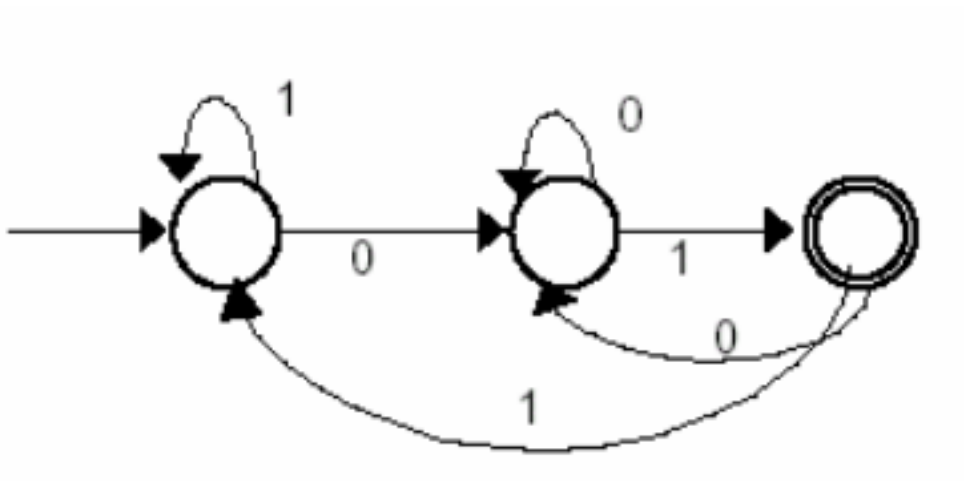
问题 3：

构造有穷自动机 .

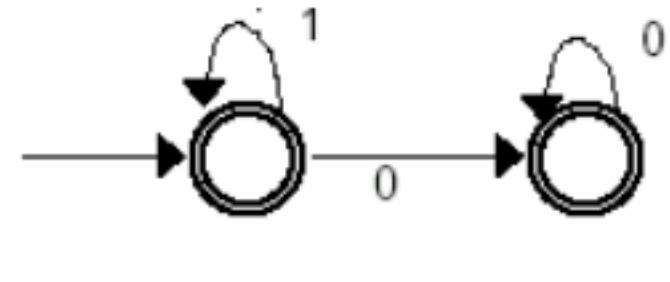
a) 构造一个 DFA，接受字母表 {0, 1} 上的以 01 结尾的所有串
b) 构造一个 DFA，接受字母表 {0, 1} 上的不包含 01 子串的所有串 .
c) 构造一个 NFA，接受字母表 {x,y} 上的正规式 x(x|y)*x 描述的集合
d) 构造一个 NFA，接受字母表 {a, b} 上的正规式 (ab|a)*b+ 描述的集合并将其转换为等价的 DFA. 以及最小状态 DFA

答案：

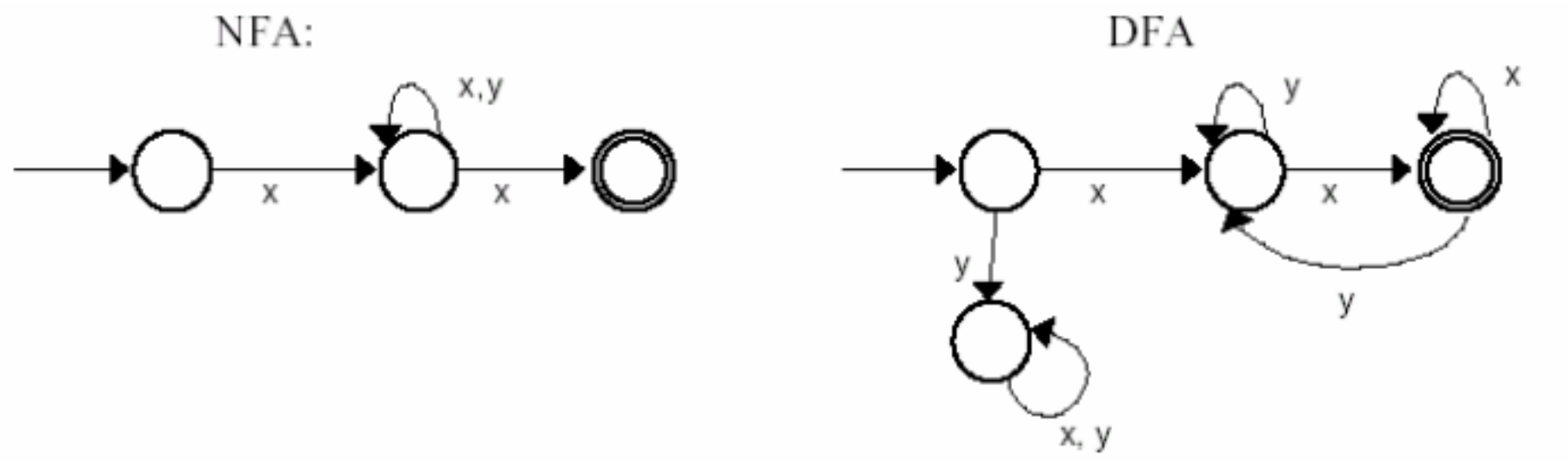
a)



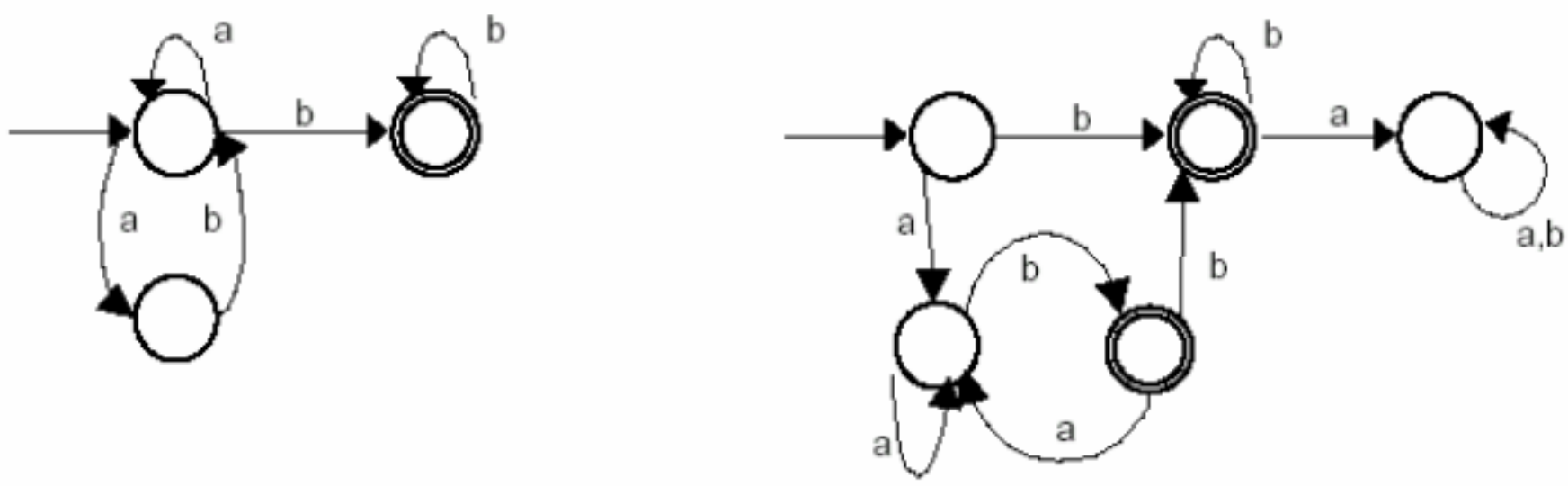
b)



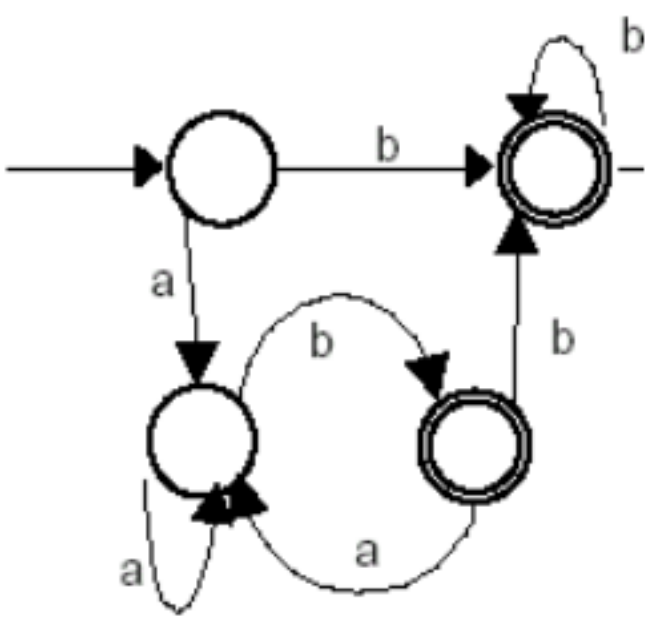
c)



d)

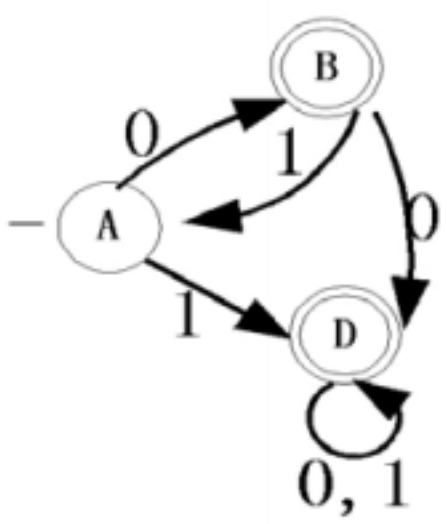


最小化的 DFA



问题 4：

设有如图所示状态转换图，求其对应的正规表达式。



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。

问题 5：

已知正规式：

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

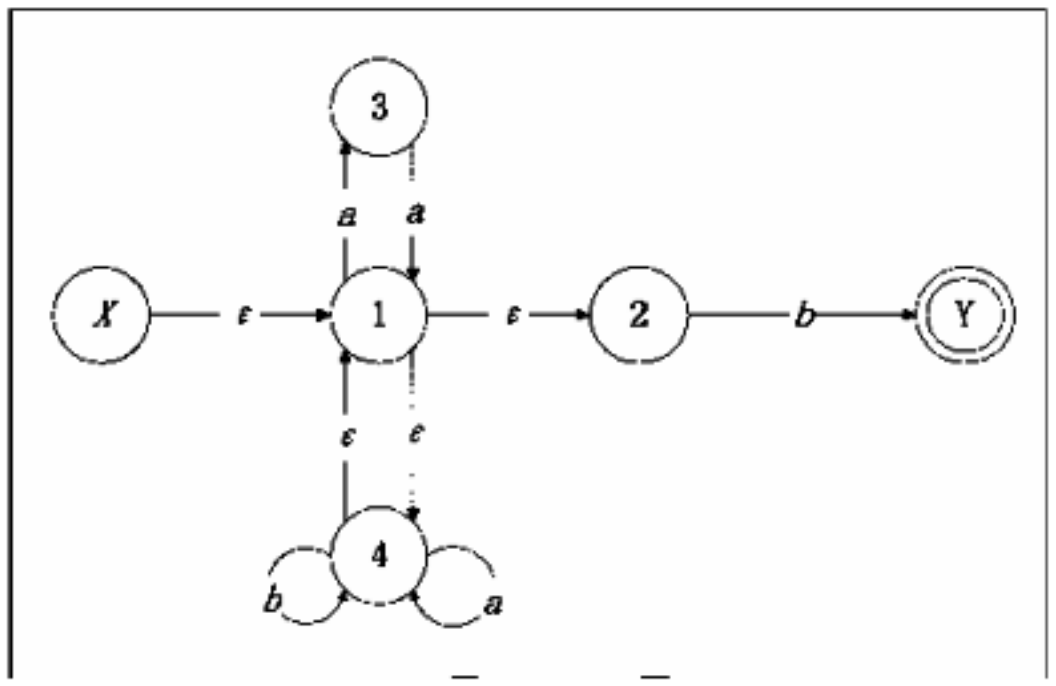
(2)(a|b)*b.

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：



状态转换表 1

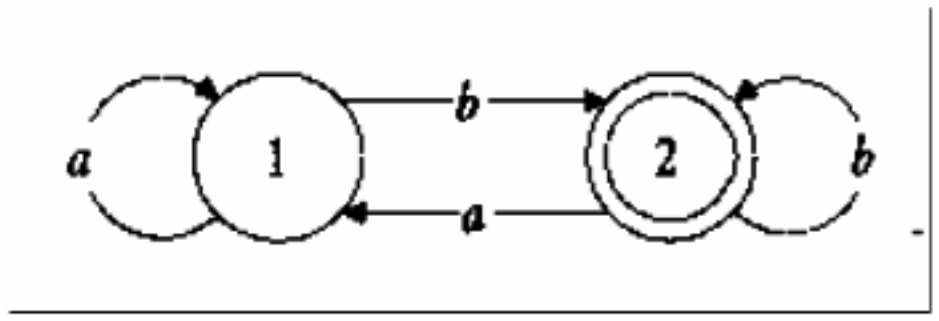
a		b
X124 1234		124Y
1234 1234		124Y
124Y 1234		124Y

状态转换表 2

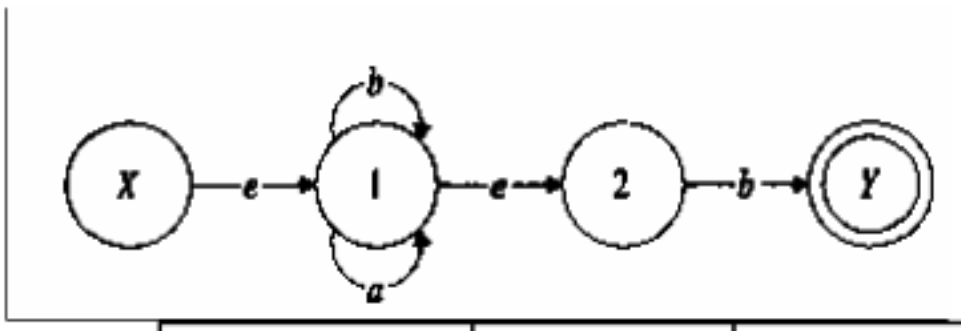
a		B
1 2		3
2 2		3
3 2		3

由于 2 与 3 完全一样，将两者合并，即见下表

a		b
1 2		3
2		3



而对正规式 (2)可画 NFA 图，如图所示。

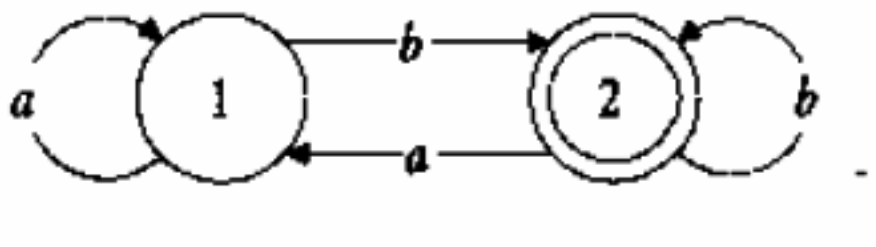


a		b
X12 12		12Y
12 12		12Y
12Y 12		12Y

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。

问题 6：

考虑正规表达式 r = a*b(a | b) ，构造可以生成语言 L(r) 的一个正规文法。

答案：

S → a*b(a | b)

变换为 S → aA, S → b(a | b) , A → aA , A → b(a | b)

变换为 S → aA, S → bB, B → (a | b) , A → aA , A → bC, C → (a | b)

变换为 S → aA, S → bB, B → a, B → b , A → aA , A → bC, C → a, C → b

所以，一个可能的正规文法为 G[S]：

S → aA, S → bB, B → a, B → b , A → aA , A → bC, C → a, C → b

或表示为：

S → aA | bB, B → a | b , A → aA | bC, C → a | b

（适当等价变换也可以，但要作说明，即要有步骤）

问题 7：

考虑下图所示的 NFA N ， 构造可以生成语言 L(N) 的一个正规文法。

答案：

G[P]:

P → 0 P ?1 P ?1 Q

Q → 0 R ?1 R

R → ε

问题 8：

考虑如下文法 G[S] ：

S → 0S?1S?1A

A→ 0B?1B

B → ε

a) 试构造语言为 L(G) 的一个正规表达式。

b) 试构造语言为 L(G) 的一个有限自动机。

答案：

a)

由 A→ 0B ， B → ε 得 A→ 0；

由 A→ 1B ， B → ε 得 A→ 1；

由 A→ 0，A→ 1 得 A→ 0 ?1；

由 S→ 1A，A→ 0 ?1 得 S→ 1(0 ?1)；

由 S→ 1A，A→ 0 ?1 得 S→ 1(0 ?1)；

由 S→ 0S，S→ 1(0 ?1) 得 S→ 0*1(0 ?1)；

由 S→ 1S，S→ 1(0 ?1) 得 S→ 1*1(0 ?1)；

由 S→ 0*1(0 ?1)，S→ 1*1(0 ?1) 得 S→ 0*1(0 ?1)？ 1*1(0 ?1)；

所以，一个可能的正规表达式为：

b)

0*1(0 ?1) ? 1*1(0 ?1)

第 5 章　自顶向下语法分析方法

第 1 题

对文法 G[S]

S→a|∧|(T)

T→T,S|S

(1) 给出 (a,(a,a)) 和 (((a,a), ∧,(a)),a) 的最左推导。

(2) 对文法 G，进行改写，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是 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)),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)

(2) 改写文法为：

0) S→a

1) S→∧

2) S→(T)

3) T→S N

4) N→, S N

5) N→ε

非终结符	FIRST 集	FOLLOW 集
S	{a, ∧,(}	{#,,,)}
T	{a, ∧,(}	{)}
N	{,, ε}	{)}

对左部为 N 的产生式可知：

FIRST （→, S N）={ , }

FIRST （→ε）={ ε}

FOLLOW （N）={)}

由于 SELECT(N →, S N)∩SELECT(N →ε) ={ , } ∩ {)}= ∅

所以文法是 LL(1) 的。

预测分析表（ Predicting Analysis Table ）

	a	∧	(	)	,	#
S	→a	→∧	→(T)			
T	→S N	→S N	→S N			
N				→ε	→, S N	

也可由预测分析表中无多重入口判定文法是 LL(1) 的。

(3) 对输入串（ a,a）#的分析过程为：

栈（STACK）	当前输入符（CUR_CHAR）	剩余输入符（INOUT_STRING）	所用产生式（OPERATION）
#S	(	a,a)#...	S→(T)
#)T(	(	a,a)#...	.
#)T	a	,a)#...	T→SN
#)NS	a	,a)#...	S→a
#)Na	a	,a)#...	.
#)N	,	a)#...	N→,SN
#)NS,	,	a)#...	.
#)NS	a	)#...	S→a
#)Na	a	)#...	.
#)N	)	#...	N→ε
#)	)	#...	
#	#		

可见输入串（ a,a）#是文法的句子。

第 3 题

已知文法 G[S]：

S→MH|a

H→LSo| ε

K→dML| ε

L→eHf

M→K|bLM

判断 G 是否是 LL(1) 文法，如果是，构造 LL(1) 分析表。

答案：

文法展开为：

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

非终结符 FIRST	集 FOLLOW	集
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}	.....

对相同左部的产生式可知：

SELECT(S →M H) ∩SELECT(S →a) ={ d,b ,e , #,o } ∩{ a }= ∅

∅

SELECT(K →d M L) ∩SELECT(K → ε) ={ d } ∩{ e,#,o }= ∅

SELECT(M →K) ∩SELECT(M →b L M) ={ d , e,#,o } ∩{ b }= ∅

所以文法是 LL(1) 的。

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也可判定文法是　　LL(1) 的。

第 7 题

对于一个文法若消除了左递归，提取了左公共因子后是否一定为 LL(1) 文法 ?试对下面文法进行改写，并对改写后的文法进行判断。

(１)A→baB| ε

B→Abb|a

(2) A→aABe|a

B→Bb|d

(3) S→Aa|b

A→SB

B→ab

答案：

(１)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。

（2） 文法：

A→aABe|a

B→Bb|d

提取左公共因子和消除左递归后文法变为：

0) A→a N

1) N→A B e

2) N→ ε

3) B→d N1

4) N1→b N1

5) N1→ ε

非终结符 FIRST	集 FOLLOW	集
A {a}	... {#,d}	
B {d}	... {e}	..
N {a,	ȩ {#,d}	
N1 {b,	ȩ {e}	..

对相同左部的产生式可知：

SELECT(N →A B e) ∩SELECT(N → ε) ={ a } ∩{#,d }= ∅

SELECT(N1 →b N1) ∩SELECT(N1 → ε) ={ b } ∩{ e }= ∅

所以文法是 LL(1) 的。

预测分析表（ Predicting Analysis Table ）

	a	e	b	d	#
A	→a N				
B				→d N1	
N1		→ ε	→b N1		
N	→ ABe			→ ε	→ ε

也可由预测分析表中无多重入口判定文法是 LL(1) 的。

（3）文法：

S→ Aa|b

A→SB

B→ab

第 1 种改写：

用 A 的产生式右部代替 S 的产生式右部的 A 得：

S→ SBa|b

B→ab

消除左递归后文法变为：

0) S→b N

1) N→B a N

2) N→ ε

3) B→a b

非终结符　FIRST	集 FOLLOW	集
S {b}	... {#}	
B {a}	... {a}	
N {	ε,a} {#}	

对相同左部的产生式可知：

SELECT(N →B a N) ∩SELECT(N → ε) ={ a } ∩{# }= ∅

所以文法是　LL(1) 的。

预测分析表（　Predicting Analysis Table ）

	a	b	#
S		→b N	
B	→a b		
N	→B a N		→ ε

也可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：

用 S 的产生式右部代替　A 的产生式右部的　S 得：

S→ Aa|b

A → AaB|bB

B → ab

消除左递归后文法变为：

0) S→ A a

1) S→ b

2) A → b B N

3) N → a B N

4) N → ε

5) B → a b

非终结符　FIRST	集 FOLLOW	集
S {b}	... {#}	
A {b}	... {a}	
B {a}	... {a}	
N {a,	ε} {a}	

对相同左部的产生式可知：

SELECT(S →A a) ∩SELECT(S →b) ={ b } ∩{ b }={ b } ≠∅

SELECT(N →a B N) ∩SELECT(N → ε) ={ a } ∩{ a }={ a } ≠∅

预测分析表：

	a	b	#
S		→ A a..	
		→b....	
A		→b B N	
B	→a b..		
N	→a B N		
	→ ε..		

也可由预测分析表中含有多重入口判定文法不是　　LL(1) 的。

附加题

问题 1：

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();
}
```

问题 2：

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

问题 3：

试验证如下文法 G[E] 是 LL(1) 文法：

E → [F] E ′
E ’→ E? ε
F → aF ’
F ’→ aF ? ε

其中 E,F,E ’,F为非终结符

答案：

各非终结符的 FIRST 集和 FOLLOW 集如下：

FIRST（E）={ [}　　FOLLOW （E）={ # }
FIRST（E ’）={ [, ε}　　FOLLOW （E ’）={ # }
FIRST（F）={ a }　　FOLLOW （F）={] }
FIRST（F ’）={ a , ε}　　FOLLOW （F ’）={] }

对于 E ’→ E? ε, FIRST(E) ∩ FIRST(ε)= φ
FIRST(E) ∩ FOLLOW （E ’）= φ

对于 F ’→ aF ? ε, FIRST(aF ’) ∩ FIRST(ε)= φ
FIRST （aF ’）∩ FOLLOW （F ’）= φ

所以，文法 G[E] 是 LL （1）文法。

问题 4：

文法 G[E] 是 LL(1) 文法：

E → [F] E ′
E ’→ E? ε
F → aF ’
F ’→ aF ? ε

其中 E,F,E ’,F为非终结符。

对文法 G[E] 构造递归下降分析程序。

答案：

/* 用类 C 语言写出 G[E] 的递归子程序， 其中 yylex() 为取下一单词过程， 变量 lookahead 存放当前单词。 */

```
int    lookahead;
```

```
void ParseE( )
{
    MatchToken      ( ['' ');
    ParseF(     );
    MatchToken      ( ]'' ');
    ParseE       ')'; (
}

void ParseE     ’ ( )
{
    switch      (lookahead) {
                case [ :'
                      ParseE( );
                      break;
                case # :'
                      break;
                default:
                      printf("syntax error \n")
                      exit(0);
          }
}

void ParseF( )
{
    MatchToken      ( á '');
    ParseF      ( ');
}

void ParseF     ’ ( )
{
    switch      (lookahead) {
                case á :'
                      MatchToken (  á ');
                      ParseF  ’  ( );
                      break;
                case ] :'
                      break;
                default:
                      printf("syntax error \n")
                      exit(0);
          }
}
```

```
void MatchToken(int expected)
{
    if     (lookahead != expected)  //  判别当前单词是否与期望的终结符匹配
        {
        printf("syntax          error \n");
                exit(0);
        }
      else         //  若匹配 ,消费掉当前单词并读入下一个调用词法分析程序
                lookahead = yylex();
      }
```

问题 5：

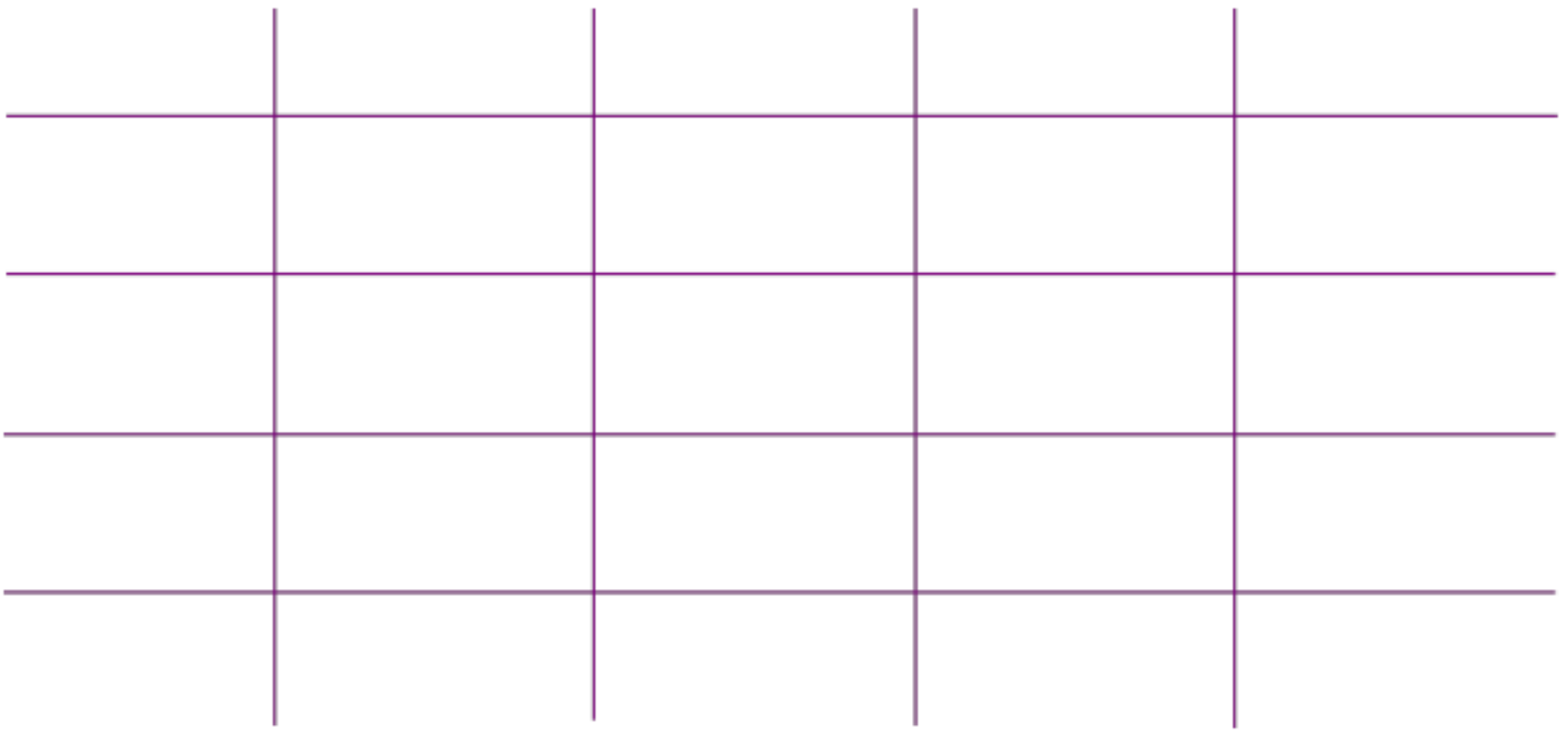
文法 G[E] 是 LL(1) 文法 :

E → [F] E ′
E ’→ E? ε
F → aF ’
F ’→ aF ? ε

其中 E,F,E ’,F为非终结符。

构造文法 G[E] 的 LL(1) 分析表。

答案：



问题 6：

试消除下面文法 G[A] 中的左递归和左公因子, 并判断改写后的文法是否为 LL(1) 文法？

G[A] ： A→aABe ？a

B→Bb ？d

答案：

提取左公共因子和消除左递归后, G［A］变换为等价的 G′［A］如下：

A→a A′

A′→A B e| ε

B→d B′

B′→b B′| ε

计算非终结符的 FIRST 集和 FOLLOW 集结果如下：

FIRST （A）= { a}　　FOLLOW （A）= {#,d }

FIRST （B）= { d}　　FOLLOW （B）= { e }

FIRST （A′）= { a, ε}　　FOLLOW （A′）= {#,d }

FIRST （B′）= { b, ε}　　FOLLOW （B′）= { e }

对相同左部的产生式可知：

FIRST (A B e) ∩FOLLOW (A′)={ a } ∩{#,d }= ？

FIRST (b B′)∩FOLLOW (B′)={ b } ∩{ e }= ？

所以 G′［S］是 LL(1) 文法。

第 6 章 自底向上优先分析

第 1 题

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] 是否为算符优先文法。

(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：

文法展开为：

S→a

S→∧

S→(T)

T→T,S

T→S

(1) FIRSTVT - LASTVT 表：

非终结符	FIRSTVT 集	LASTVT 集
S	{a ∧ (}	{a ∧)}
T	{a ∧ (, }	{a ∧), }

(2) 算符优先关系表：

	a	∧	(	)	,	#
a				⋗	⋗	⋗
∧				⋗	⋗	⋗
(	⋖	⋖	⋖	≐	⋖	
)				⋗	⋗	⋗
,	⋖	⋖	⋖	⋗	⋗	
#	⋖	⋖	⋖			≐

表中无多重人口所以是算符优先（　　OPG）文法。

友情提示：记得增加拓广文法　　S`→#S#，所以 #　FIRSTVT(S)　　，LASTVT(S)　#⋗　　。

(3)对应的算符优先函数为：

a		( ) ,			^	#
f 2 1 2	2 2 1					
g 3 3 1	1 3 1					

（4）对输入串（　a,a）#的算符优先分析过程为

栈（STACK）	当前输入字符（　CHAR）	剩余输入串（　INPUT_STRING　）	动作（ ACTION ）
#	(	a,a)#...	Move in
#(	a	,a)#...	Move in
#(a	,	a)#...	Reduce: S→a
#(N	,	a)#...	Move in
#(N,	a	)#...	Move in
#(N,a	)	#...	Reduce: S→a
#(N,N	)	#...	Reduce: T→T,S
#(N	)	#...	Move in
#(N)	#		Reduce: S→(T)
#N	#		

Success!

对输入串（ a,(a,a)）# 的算符优先分析过程为：

栈（STACK ）	当前字符（CHAR ）	剩余输入串（INPUT_STRING ）	动作（ ACTION ）
#	(	a,(a,a))#...	Move in
#(	a	,(a,a))#...	Move in
#(a	,	(a,a))#...	Reduce: S→a
#(N	,	(a,a))#...	Move in
#(N,	(	a,a))#...	Move in
#(N,(	a	,a))#...	Move in
#(N,(a	,	a))#...	Reduce: S→a
#(N,(N	,	a))#...	Move in
#(N,(N	a	))#...	Move in
#(N,(N,a	)	)#...	Reduce: S→a
#(N,(N,N	)	)#...	Reduce: T→T,S
#(N,(N	)	)#...	Move in
#(N,(N)	)	#...	Reduce: S→(T)
#(N,N	)	#...	Reduce: T→T,S
#(N	)	#...	Move in
#(N)	#		Reduce: S→(T)
#N	#		

Success!

第 2 题

已知文法 G[S] 为：

S→a|∧|(T)

T→T,S|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 1 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)的最右推导过程为：

S⇒(T)

⇒(T,S)

⇒(T,a)

⇒(S,a)

⇒(a,a)

（a,(a,a)）的最右推导过程为：

S⇒(T)

⇒(T,S)

⇒(T,(T))

⇒(T,(T,S))

⇒(T,(T,a))

⇒(T,(S,a))

⇒(T,(a,a))

⇒(S,(a,a))

⇒(a,(a,a))

(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。

第 3 题：

有文法 G[S] ：

S? V

V? T|ViT

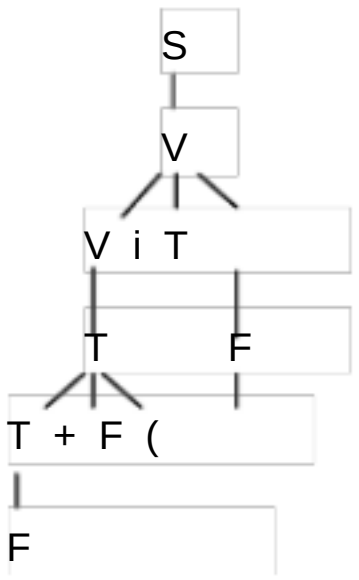
T? F|T+F

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),(,		*,(

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。

(+(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

第 4 题

文法 G［S］为：

$$S \rightarrow S;G \mid G$$
$$G \rightarrow G(T) \mid H$$
$$H \rightarrow a \mid (S)$$
$$T \rightarrow T+S \mid S$$

（1） 构造 G［S］的算符优先关系表，并判断 G［S］是否为算符优先文法。
（2） 给出句型 a(T+S);H;(S) 的短语、句柄、素短语和最左素短语。
（3） 给出 a;(a+a) 和(a+a) 的分析过程，说明它们是否为 G［S］的句子。
（4） 给出（3）中输入串的最右推导，分别说明两输入串是否为 G[S] 的句子。
（5） 由（3）和（4）说明了算符优先分析的哪些缺点。
（6） 算符优先分析过程和规范归约过程都是最右推导的逆过程吗？

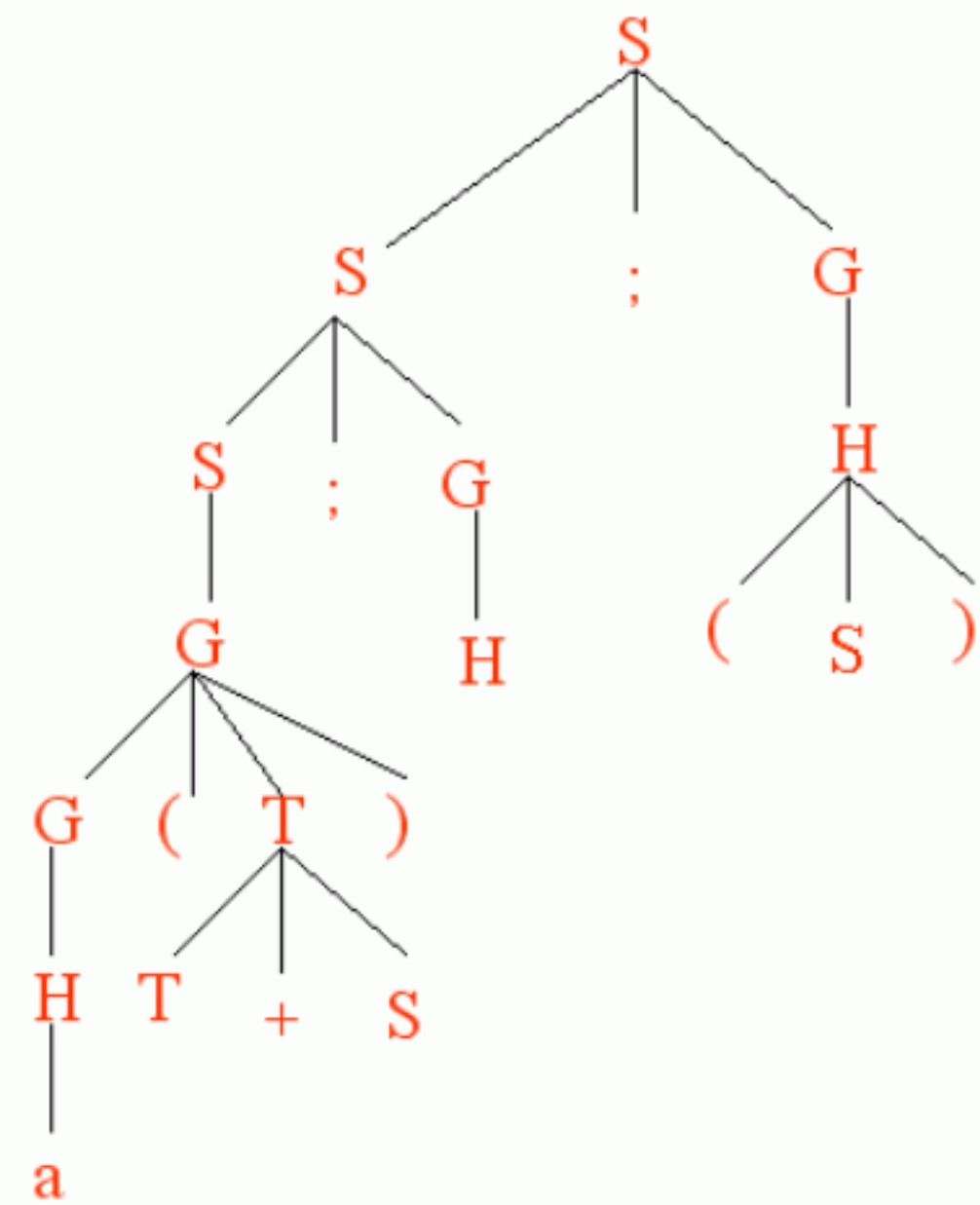
答案：

（1）构造文法 G［S］的算符优先关系矩阵：

	;	(	)	a	+	#
;	·＞	＜·	·＞	＜·	·＞	·＞
(	＜·	＜·	=·	＜·	＜·	
)	·＞	·＞	·＞		·＞	·＞
a	·＞	·＞	·＞		·＞	·＞
+	＜·	＜·	·＞	＜·	·＞	
#	＜·	＜·		＜·		=·

在上表中可看出终结符之间的优先关系是唯一的，或称 G［S］的算符优先关系矩阵不含多重入口，因此，G［S］是一个算符优先文法。

（2）



短语：

a相对H、G

T+S相对T

a(T+S)相对G、S

H相对G

a(T+S);H相对S

(S)相对H、G

a(T+S);H;(S)相对S

句柄：a

素短语：T+S、(S)

最左素短语：T+S

（3）对输入串 (a+a) ＃的分析过程如下：

步骤	栈	当前符号	剩余输入串	移进或归约
（1）	#	(	a+a)#	移进
（2）	#(	a	+a)#	移进
（3）	#(a	+	a)#	归约
（4）	#(N	+	a)#	移进
（5）	#(N+	a	)#	移进
（6）	#(N+a	)	#	归约
（7）	#(N+N	)	#	归约
（8）	#(N	)	#	移进
（9）	#(N)	#		归约
（10）	#N	#		分析成功

说明是它的句子。

（４）试用规范推导：

S? G? H? （S）由此往下 S 不可能推导出 a+a，所以 (a+a) 不是G［S］的句子。

（5）结果说明：由于算符优先分析法去掉了单非终结符之间的归约，尽管在分析过程中，

当决定是否为句柄时采取一些检查措施，但仍难完全避免把错误的句子得到正确的归约。

（6）算符优先分析过程不是最右推导的逆过程。规范归约过程是最右推导的逆过程。

附加题

问题 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

问题 2：

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	⋖	⋖	⋖	⋖	⋗	⋖	⋗	⋗
(	⋖	⋖	⋖	⋖	⋖	⋖	≡	
)				⋗	⋗		⋗	⋗
#	⋖	⋖	⋖	⋖	⋖	⋖		

第 7 章 LR 分析

第 1 题

已知文法

A→aAd|aAb| ε

判断该文法是否是 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：

文法：

A→aAd|aAb| ε

拓广文法为 G′，增加产生式 S′→A

若产生式排序为：

0 S' →A

1 A →aAd

2 A →aAb

3 A → ε

由产生式知：

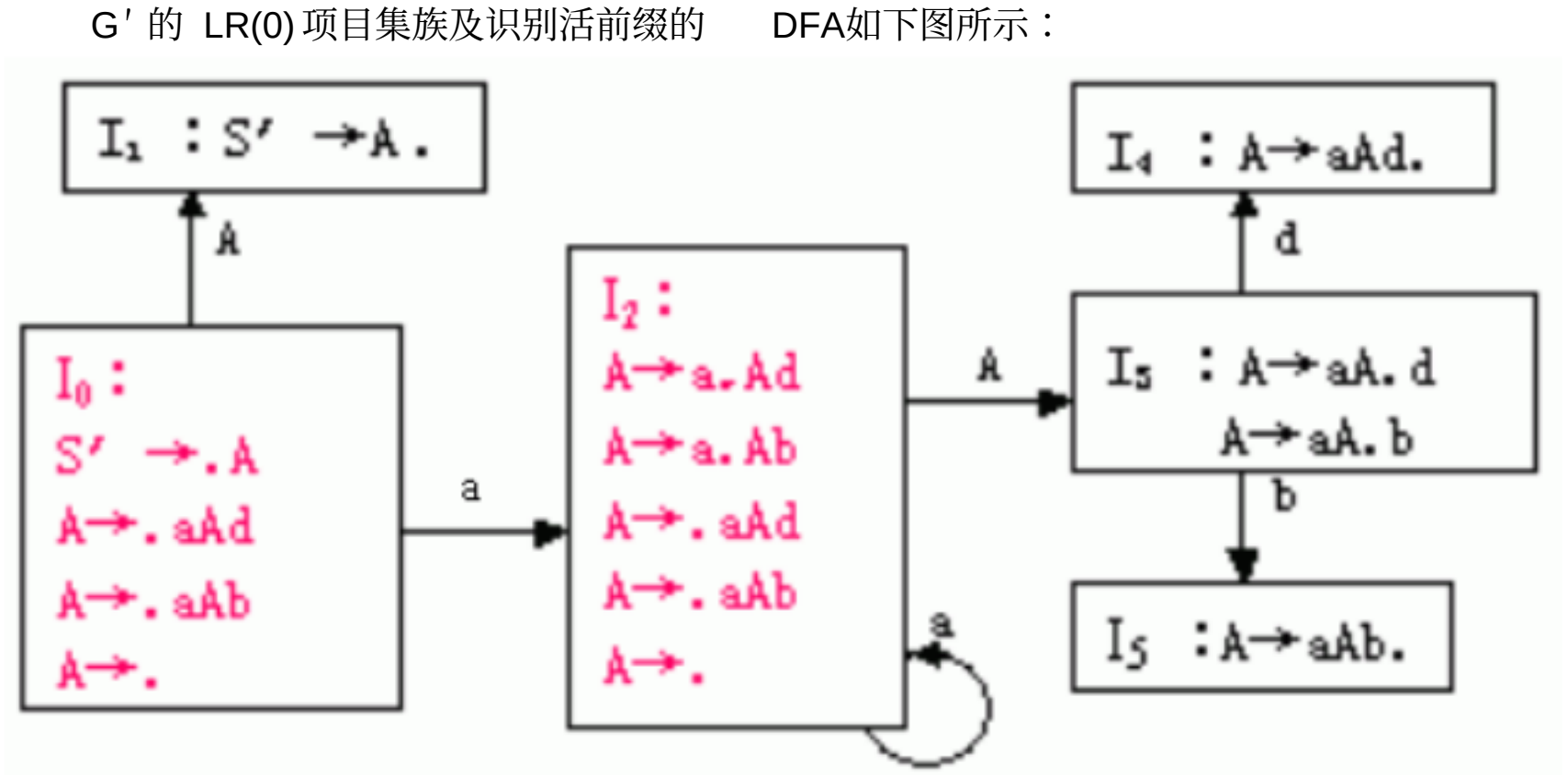
First (S') = { ε ,a}

First (A) = { ε ,a}

Follow(S') = {#}

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

G′的 LR(0) 项目集族及识别活前缀的 DFA如下图所示：



在 I 0中：

A →.aAd 和 A →.aAb 为移进项目， A →. 为归约项目，存在移进 - 归约冲突，因此所给文法不是 LR(0) 文法。

在 I 0、 I2 中：

Follow(A) ∩{a}= {d ， b， #} ∩{a}= ∅

所以在 I_0、I_2中的移进 - 归约冲突可以由 Follow 集解决，所以 G是 SLR(1)文法。
构造的 SLR(1)分析表如下：
题目 1 的 SLR(1)分析表

状态（State）	Action				Goto
	a	d	b	#	A
0	S2	r3	r3	r3	1
1				acc	
2	S2	r3	r3	r3	3
3		S4	S5		
4		r1	r1	r1	
5		r2	r2	r2	

题目 1 对输入串 ab#的分析过程

状态栈（state stack ）	文法符号栈	剩余输入串（input left ）	动作（action ）
0	#	ab#....	Shift
0 2	#a	b#....	Reduce by :A →ε
0 2 3	#aA	b#....	Shift
0 2 3 5	#aAb	#....	Reduce by :A →aAb
0 1	#A	#....	

分析成功，说明输入串 ab 是文法的句子。

第 2 题

若有定义二进制数的文法如下：

S→L · L|L

L→LB|B

B→0|1

(1) 试为该文法构造 LR 分析表，并说明属哪类 LR 分析表。

(2) 给出输入串 101.110 的分析过程。

答案：

文法：

S→L.L|L

L→LB|B

B→0|1

拓广文法为 G′，增加产生式 S′→S

若产生式排序为：

0 S' →S

1 S →L.L

2 S →L

3 L →LB

4 L →B

5 B →0

6 B →1

由产生式知：

First (S') = {0,1}

First (S) = {0,1}

First (L) = {0,1}

First (B) = {0,1}

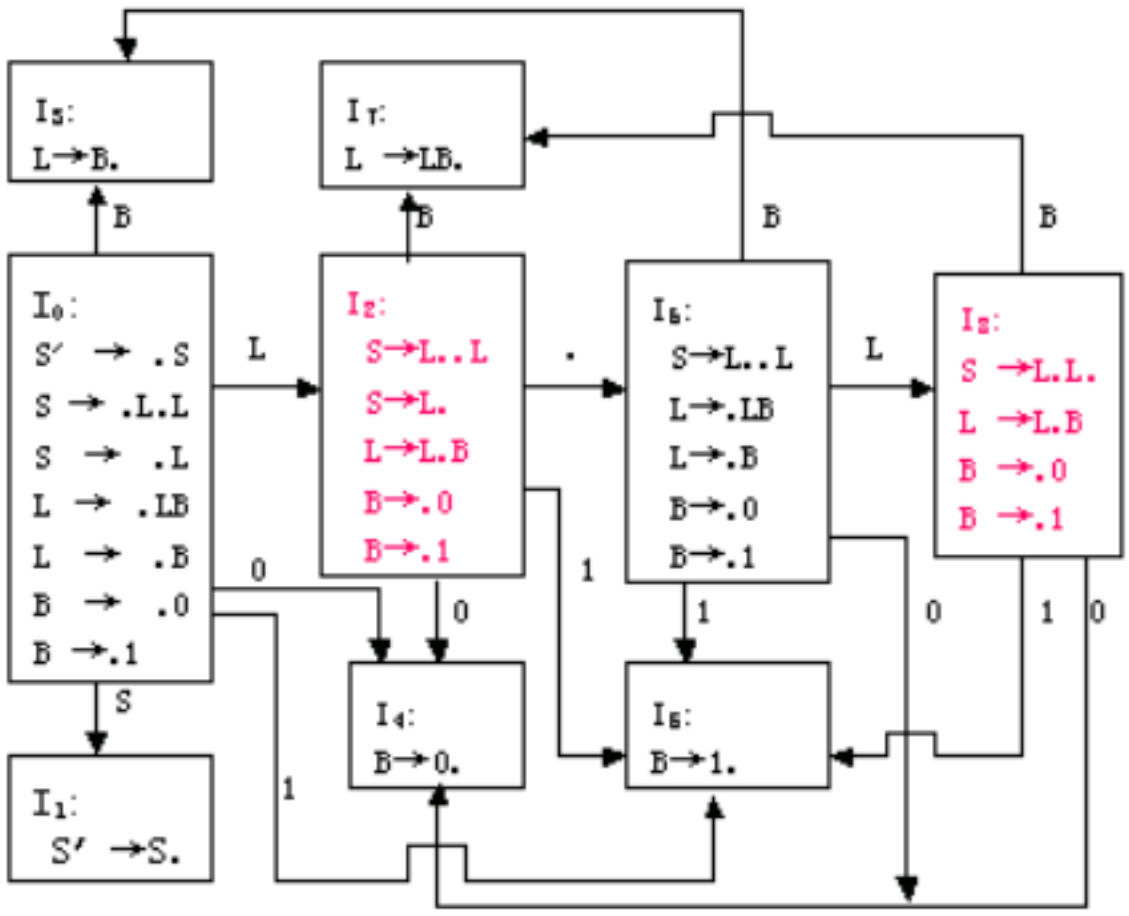
Follow(S') = {#}

Follow(S) = {#}

Follow(L) = {.,0,1,#}

Follow(B) = {.,0,1,#}

G′的 LR(0) 项目集族及识别活前缀的 DFA如下图所示：



在 I 2中：

B →.0 和 B →.1 为移进项目， S →L. 为归约项目，存在移进 - 归约冲突，因此所给文法不是 LR(0) 文法。

在 I 2、 I8 中：

Follow(s) ∩{0， 1}= { #} ∩{0， 1}= ∅

	·	0	1	#	S	L	B
0		S4	S5		1	2	3
1				acc		.	
2	S6	S4	S5	r2			7
3	r4	r4	r4	r4		.	
4	r5	r5	r5	r5		.	
5	r6	r6	r6	r6		.	
6		S4	S5		8	3	
7	r3	r3	r3	r3		.	
8		S4	S5	r1		7	

题目 2 对输入串 101.110#的分析过程

状态栈（state stack ）	文法符号栈	剩余输入串（input left ）	动作（action ）
0	#	101.110#....	Shift
0 5	#1	01.110#....	Reduce by :B →1
0 3	#B	01.110#....	Reduce by :S →LB
0 2	#L	01.110#....	Shift
0 2 4	#L0	1.110#....	Reduce by :B →0
0 2 7	#LB	1.110#....	Reduce by :S →LB
0 2	#L	1.110#....	Shift
0 2 5	#L1	.110#....	Reduce by :B →1
0 2 7	#LB	.110#....	Reduce by :S →LB
0 2	#L	.110#....	Shift
0 2 6	#L.	110#....	Shift
0 2 6 5	#L.1	10#....	Reduce by :B →1
0 2 6 3	#L.B	10#....	Reduce by :S →B
0 2 6 8	#L.L	10#....	Shift
0 2 6 8 5	#L.L1	0#....	Reduce by :B →1
0 2 6 8 7	#L.LB	0#....	Reduce by :S →LB
0 2 6 8	#L.L	0#....	Shift
0 2 6 8 4	#L.L0	#....	Reduce by :B →0
0 2 6 8 7	#L.LB	#....	Reduce by :S →L.L
0 1	#S	#....	

分析成功，说明输入串 101.110 是题目 2 文法的句子。

第 3 题

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 ？ α X · 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

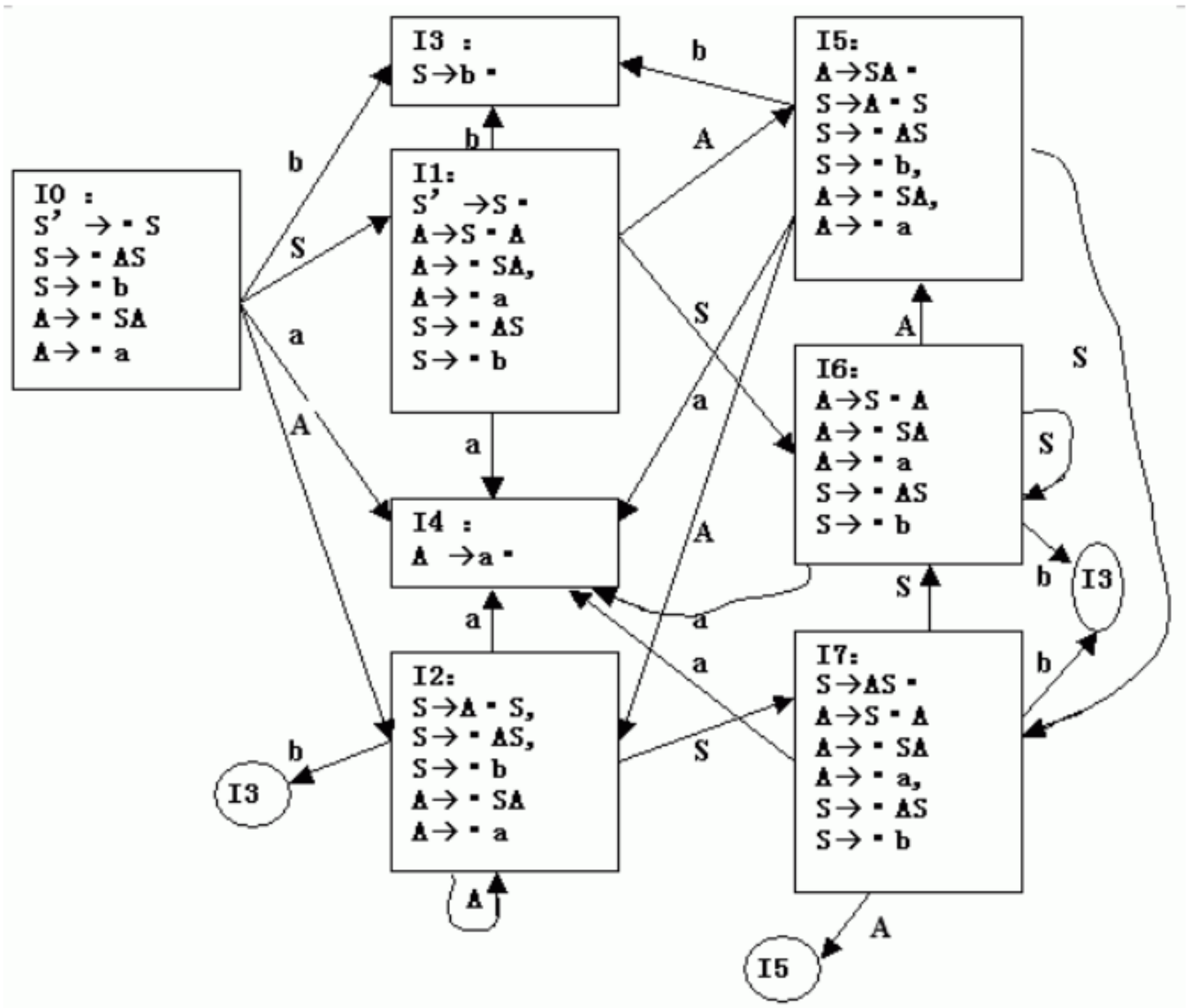
(4)A ？ a

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}

LR(0) 项目：



(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			
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(S) ∩{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，所以不是该文法　　SLR（1）文法。

注意：不是　SLR(1)文法就不能构造　SLR(1)分析表，也不能作分析过程。

(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)	023	#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）

(5)LR(1) 项目集族为：

I0 :

S’ ? ·S, #

S ? ·AS, #

S ? ·b, #

S ? ·SA, a / b

A ? ·a, a / b

I1 : S ’ ? S·,#

A ? S·A, a / b

A ? ·a, a / b

A ? ·SA, a / b

S ? ·AS, a / b

S ? ·b, a / b

I2 : S ? A·S, #

S ? ·b, #

S ? ·AS, #

A ? ·SA, a / b

A ? · a, a / b

I3 : S ? b · , #

I4 : A ? a · , a / b

I5 : A ? SA · , a / b

S ? A · S, a / b

S ? · AS, a / b

S ? · b, a/ b

A ? · SA, a / b

A ? · a, a / b

I6 : A ? S · A,a/b

A ? · SA, a / b

A ? · a, a / b

S ? · AS, a / b

S ? · b, a / b

I7: S ? b · , a / b

I8 : S ? AS · , #

A ? S · A, a / b

A ? · SA, a / b

A ? · a, a / b

S ? · AS, a / b

S ? · b, a / b

I9 :

S ? A · S, #

S ? · AS, #

S ? · b, #

S ? · SA, a / b

A ? · a, a / b

I10 :

S ? AS · , a/b

A ? S · A, a/b

A ? · S A, a/b

A ? · a, a / b

S ? · b, a/b

S ? · AS, a / b

I11 :

S ？ A · S, a/b

S ？ · b, a/b

S ？ · AS, a / b

A ？ · S A, a/b

A ？ · a, a / b

I12 :

S ？ SA· , a/b

S ？ A · S, a/b

S ？ · b, a/b

S ？ · AS, a / b

A ？ · S A, a/b

A ？ · a, a / b

∵I5 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。

注意：其实是可以构造的，这个题目出得不太严格。因为书上的定义是： 根据这种文法构造的 LR（1）分析表不含多重定义时， 称 这样的分析表为 LR（1）分析表， 能用 LR（1）分析表的分析器称为 LR（1）分析器（规范的 LR 分析器），能构造的 LR（1）分析表的文法称为 LR（1）文法。

教材习题：

（1）列出这个文法的所有 LR(0) 项目

（2）按（1）列出的项目构造识别这个文法活前缀的 NFA，把这个 NFA确定化为 DFA，说明这个 DFA的所有状态全体构成这个文法的 LR(0) 规范族

（3）这个文法是 SLR的吗？若是，构造出它的 SLR分析表

（4）这个文法是 LALR或 LR(1) 的吗？

答：

（1）令拓广文法 G’为

0 S ’ ？ S

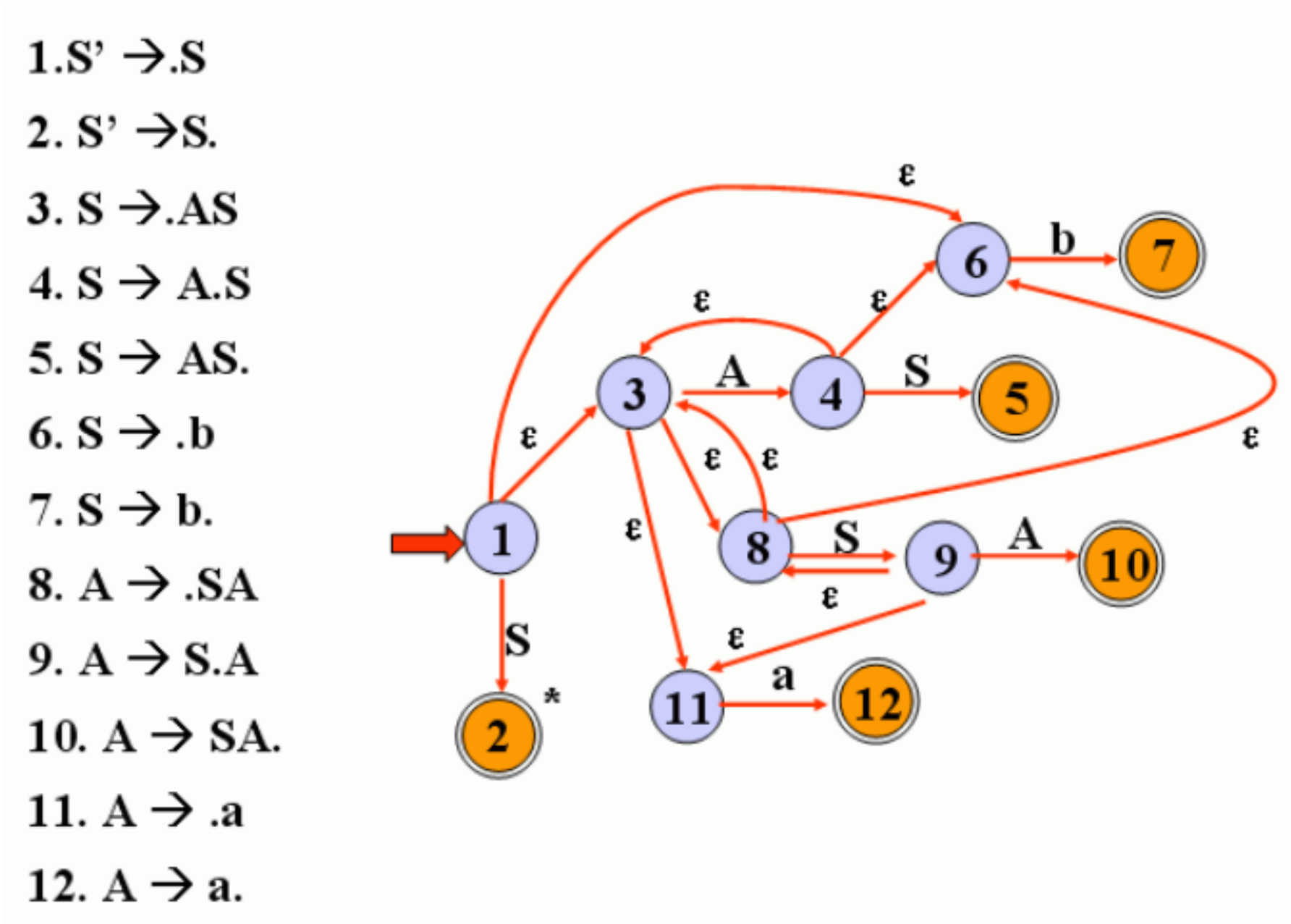
1 S ？ A S

2 S ？ b

3 A ? S A

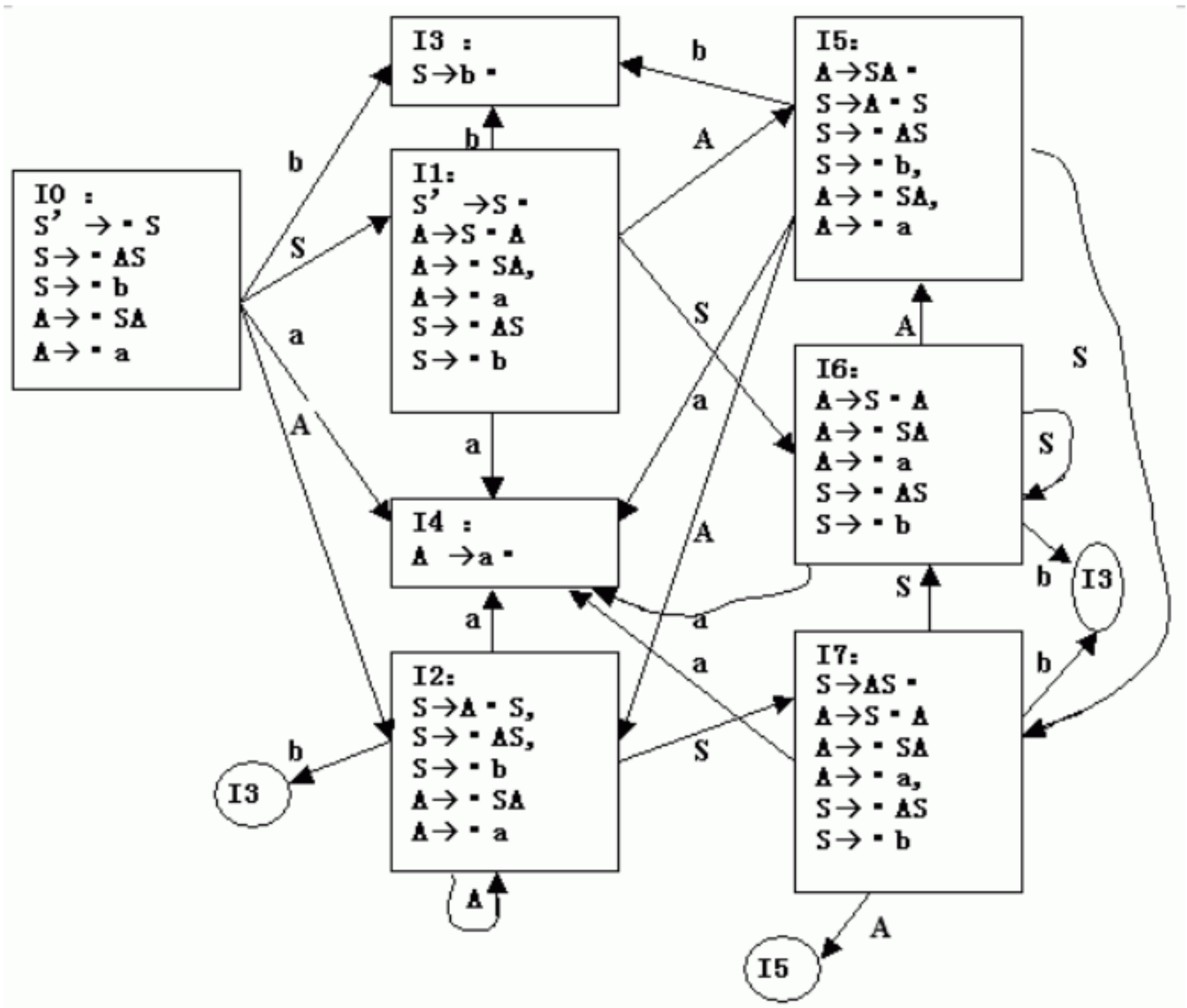
4 A ? a

其 LR(0) 项目：



(2) 识别这个文法活前缀的 NFA如上图所示：

确定化为 DFA如下图所示：



（3）因为 I5 中：FOLLOW(A) ∩{a，b}≠Φ

I7 中：FOLLOW(S) ∩{a，b}≠Φ

所以，该文法不是 SLR（1）文法。

（4）LR(1) 项目集族为：

I0 :

S’ ？ ·S, #

S ？ ·AS, #

S ？ ·b, #

S ？ ·SA, a / b

A ？ ·a, a / b

I1 : S ’ ？ S·,#

A ？ S·A, a / b

A ？ ·a, a / b

A ？ ·SA, a / b

S ？ ·AS, a / b

S ？ ·b, a / b

I2 : S　? A · S, #
　　S ? · b, #
　　S ? · AS, #
　　A ? · SA, a / b
　　A ? · a, a / b

I3 : S　? b · , #

I4 : A　? a · , a / b

I5 : A　? SA · , a / b
　　S ? A · S, a / b
　　S ? · AS, a / b
　　S ? · b, a/ b
　　A ? · SA, a / b
　　A ? · a, a / b

I6 : A　? S · A,a/b
A ? · SA, a / b
A ? · a, a / b
S ? · AS, a / b
S ? · b, a / b

I7: S ? b · , a / b

I8 : S　? AS · , #
　　A ? S · A, a / b
　　A ? · SA, a / b
　　A ? · a, a / b
　　S ? · AS, a / b
　　S ? · b, a / b

I9 :
S ? A · S, #
S ? · AS, #
S ? · b, #
S ? · SA, a / b
A ? · a, a / b

I10 :
S ? AS · , a/b
A ? S · A, a/b
A ? · S A, a/b

A ? · a, a / b

S ? · b, a/b

S ? · AS, a / b

I11 :

S ? A· S, a/b

S ? · b, a/b

S ? · AS, a / b

A ? · S A, a/b

A ? · a, a / b

I12 :

S ? SA· , a/b

S ? A· S, a/b

S ? · b, a/b

S ? · AS, a / b

A ? · S A, a/b

A ? · a, a / b

因为 I5 状态集中存在“归约——移进”冲突，所以不是 LR(1) 文法，也不是 LALR 文法。

第 6 题

文法 G=({U,T,S},{a,b,c,d,e},P,S)

其中 P 为：

S→UTa|Tb

T→S|Sc|d

U→US|e

(1) 判断 G是 LR(0)， SLR(1)， LALR(1)还是 LR(1)，说明理由。

(2) 构造相应的分析表。

答案：

文法：

S→UTa|Tb

T→S|Sc|d

U→US|e

拓广文法为 G'，增加产生式 S'→S

若产生式排序为：

0 S' →S

1 S →UTa

2 S →Tb

3 T →S

4 T →Sc

5 T →d

6 U →US

7 U →e

由产生式知：

First (S') = {d,e}

First (S) = {d,e}

First (U) = {e}

First (T) = {d,e}

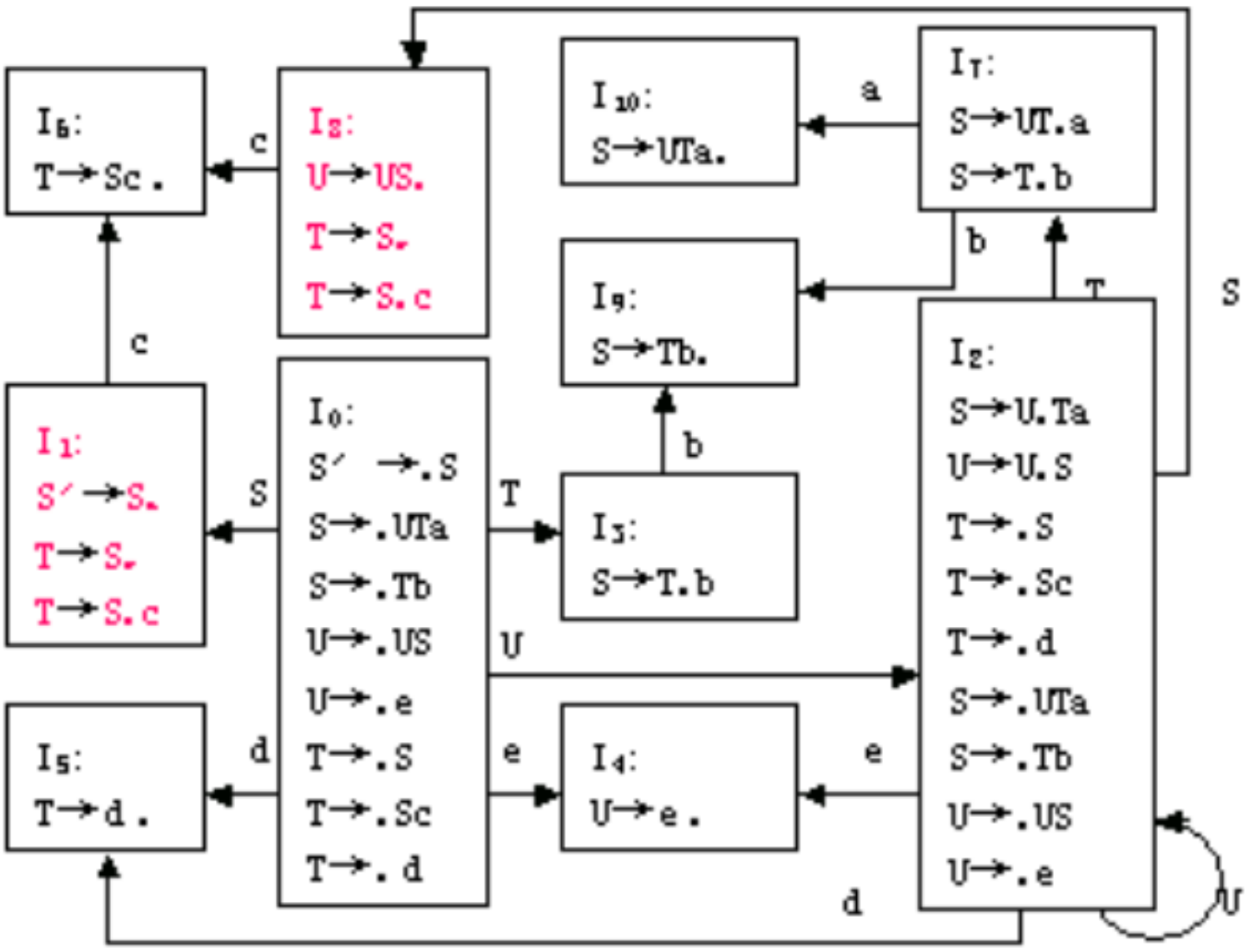
Follow(S') = {#}

Follow(S) = {a,b,c,d,e,#}

Follow(U) = {d,e}

Follow(T) = {a,b}

G′的 LR(0)项目集族及识别活前缀的 DFA如下图所示：



在 I 1中：

S' →S.为接受项目， T →S. 为归约项目， T →S.c 为移进项目，存在接受 - 归约和移进 - 归约冲突，因此所给文法不是 LR(0)文法。

在 I1 中：

Follow(S') ∩ Follow(T)= { #} ∩{a , b}= ∅

Follow(T) ∩{ c}= { a , b} ∩{c}= ∅

在 I 8中：

Follow(U) ∩ Follow(T) ∩{ c}={d,e} ∩{ a , b} ∩{c}= ∅

所以在 I 1中的接受 - 归约和移进 - 归约冲突与 I 8中的移进 - 归约和归约 - 归约冲突可以由 Follow 集解决，所以 G是 SLR(1)文法。

构造的 SLR(1)分析表如下：

状态（State）	Action						Goto		
	a	b	c	d	e	#	S	U	T
0				S5	S4		1	2	3
1	r3	r3	S6			Acc			
2				S5	S4		8	2	7
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			

第 8 题

证明文法：

S→A$

A→BaBb|DbDa

B→ ε

D→ ε

是 LR(1) 但不是 SLR(1)。(其中 '$' 相当于 '#')

答案：

文法：

A→BaBb|DbDa

B→ ε

D→ ε

拓广文法为 G′，增加产生式 S′→A

若产生式排序为：

0 S'→A

1 A →BaBb

2 A →DbDa

3 B → ε

4 D → ε

由产生式知：

First (S') = {a,b}

First (A) = {a,b}

First (B) = { ε }

First (D) = { ε }

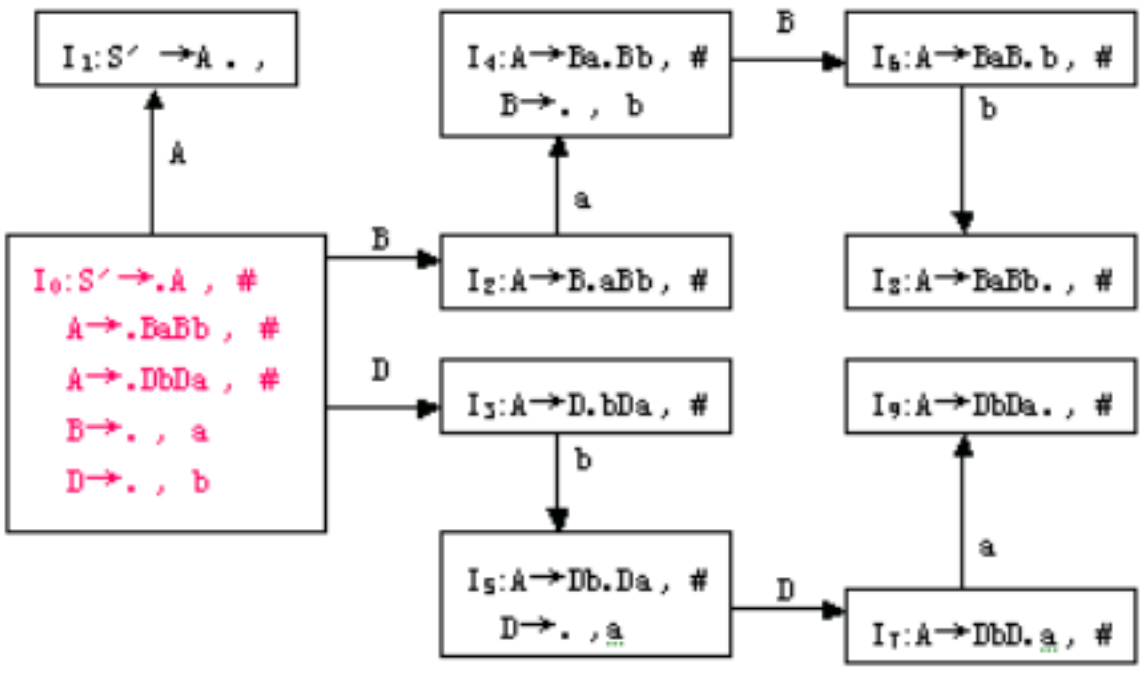
Follow(S') = {#}

Follow(A) = {#}

Follow(B) = {a,b}

Follow(D) = {a,b}

G′的 LR(1) 项目集族及识别活前缀的 DFA如下图所示：



在 I 0中：

B→.,a 和 T →. , b 为归约项目， 但它们的搜索符不同， 若当前输入符为 a 时用产生式 B → ε 归约，为 b 时用产生式 D → ε 归约，所以该文法是 LR(1) 文法。

若不看搜索符，在 I 0中项目 B →. 和 T →. 为归约 - 归约冲突，而 Follow(B) ∩Follow(D) = {a,b} ∩{a,b} ≠ ∅，冲突不能用 Follow 集解决，所以该文法不是 SLR(1)文法。

构造的 LR(1) 分析表如下：

题目 4 的 LR(1) 分析表

State	Action				Goto		
	a	.	b	#	A	B	D
0	r3		r4.....		1	2	3
1				Acc		.	
2	S4.....					.	
3			S5			.	
4			r3			6	
5	r4.....						7
6			S8				
7	S9.....						
8				r1			
9				r2			

第 16 题

给定文法：

S→do S or S|do S|S;S|act

(1) 构造识别该文法活前缀的 DFA。

(2) 该文法是 LR(0) 吗?是 SLR(1)吗?说明理由。

(3) 若对一些终结符的优先级以及算符的结合规则规定如下：

a) or 优先性大于 do;

b) ; 服从左结合；

c) ; 优先性大于 do ；

d) ; 优先性大于 or ；

请构造该文法的 LR 分析表 , 并说明 LR(0) 项目集中是否存在冲突和冲突如何解决的。

答案：

首先化简文法 , 用 d 代替 do; 用 o 代替 or; 用 a 代替 act; 文法可写成：

S→dSoS|dS|S;S|a

拓广文法为 G', 增加产生式 S′→S

若产生式排序为：

0 S' →S

1 S →dSoS

2 S →dS

3 S →S;S

4 S →a

由产生式知：

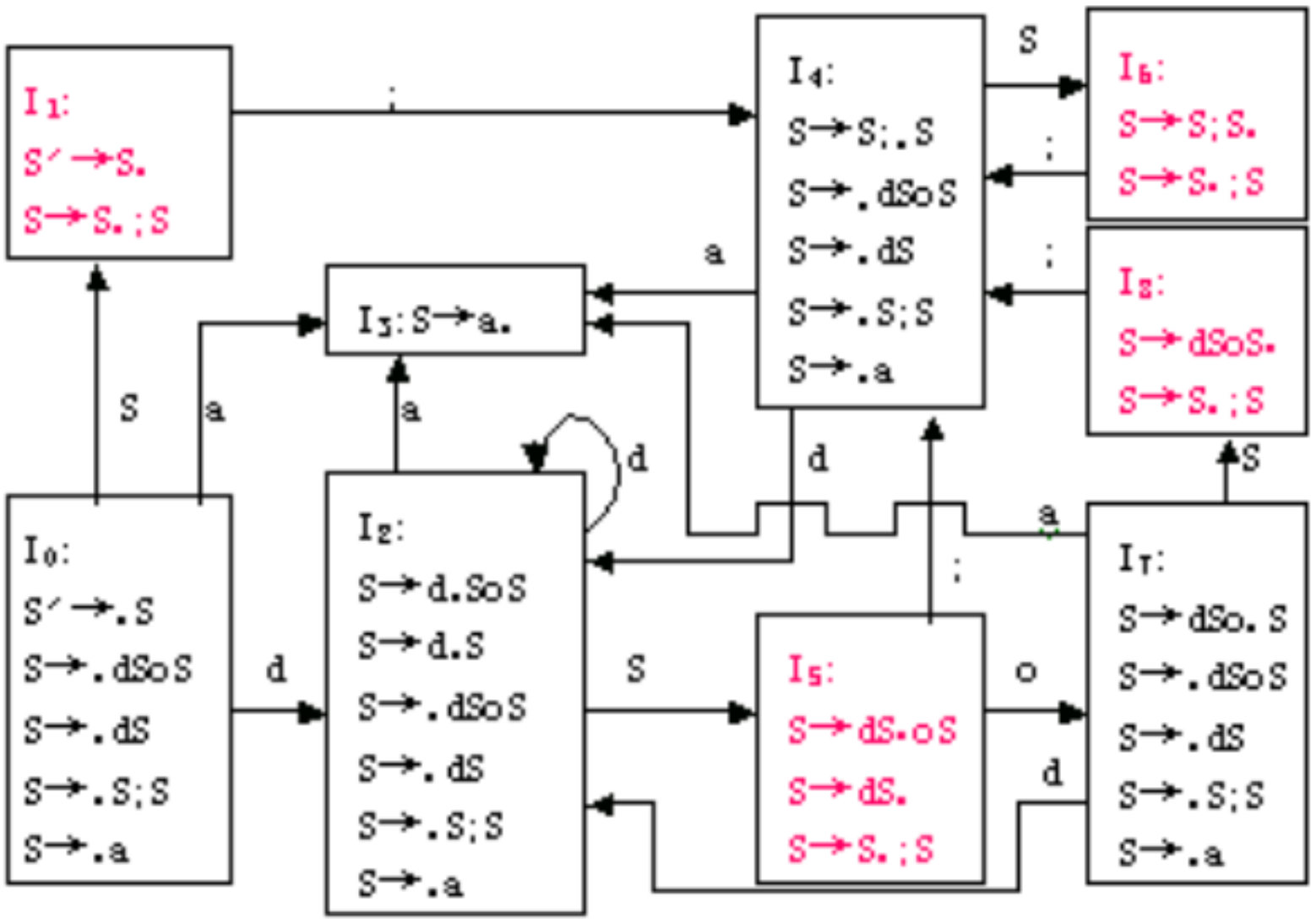
First (S') = {d,a}

First (S) = {d,a}

Follow(S') = {#}

Follow(S) = {o,;,#}

(1) 识别该文法活前缀的 DFA如下图。



(2) 该文法不是 LR(0) 也不是 SLR(1)因为：

在 I_5、I_6、和 I_8 中存在移进 - 归约冲突，因此所给文法不是 LR(0) 文法。

又由于 Follow(S) = {o, ; ,#}

在 I_6、和 I_8 中：

Follow(S) ∩{ ; }={o, ; ,#} ∩{ ; }={;} ≠ ∅

在 I_5 中：

Follow(S) ∩{ ; , o }={o , ; ,#} ∩{ ; , o }={; , o } ≠ ∅

所以该文法也不是 SLR(1) 文法。

此外很容易证明所给文法是二义性的,

S`→S→dSoS→ddSoS

S`→S→dS→ddSoS

因此该文法不可能满足 LR 文法。

(3) 若对一些终结符的优先级以及算符的结合规则规定如下：

a) or 优先性大于 do;

b) ; 服从左结合；

c) ; 优先性大于 do ；

d) ; 优先性大于 or ；

则：

在 I5 中：“ or 和; 优先性都大于 do ”，所以遇输入符 o 和； 移进；遇 #号归约。

在 I6 中：“； 号服从左结合 ”，所以遇输入符属于 Follow(S) 的都应归约。

在 I8 中：“； 号优先性大于 do ”， 所以遇输入符为 ；号 移进；遇 o 和#号归约。

此外，在 I_1中：接受和移进可以不看成冲突，因为接受只有遇 #号。

由以上分析， 所有存在的移进 - 归约冲突可用规定的终结符优先级以及算符的结合规则解决，所构造的 LR 分析表如下：

状态（State ）	Action					Goto
	d	o	;	a	#	S
0	S2			S3		1
1			S4		Acc	
2	S2			S3		5
3		r4	r4		r4	
4	S2			S3		6
5		S7	S4		r2	
6		r3	r3		r3	
7	S2			S3		8
8		r1	r4		r1	

附加题

问题 1：

试判别如下文法是否 LR(0)或 SLR(1)文法：

a) 文法 G[E]: E → E + T ? T

T → (E) ? id ? id [E]

其中 E,T 为非终结符，其余符号为终结符

b) 文法 G[S]: S → Ab ? ABc

A → aA ? a

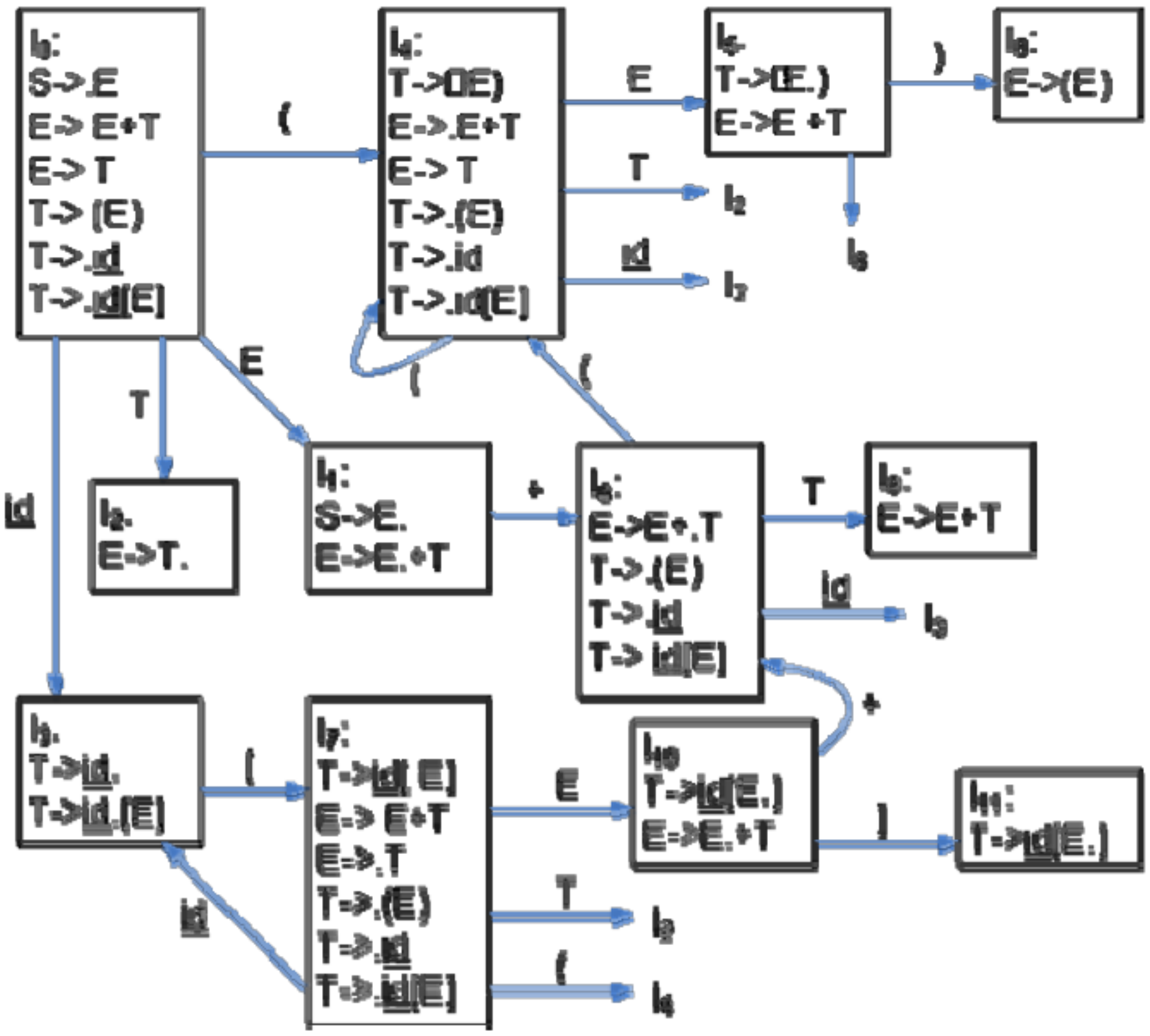
B → b

其中 S,A,B 为非终结符，其余符号为终结符

答案：

a) 增加产生式 S→E, 得增广文法 G’[S]

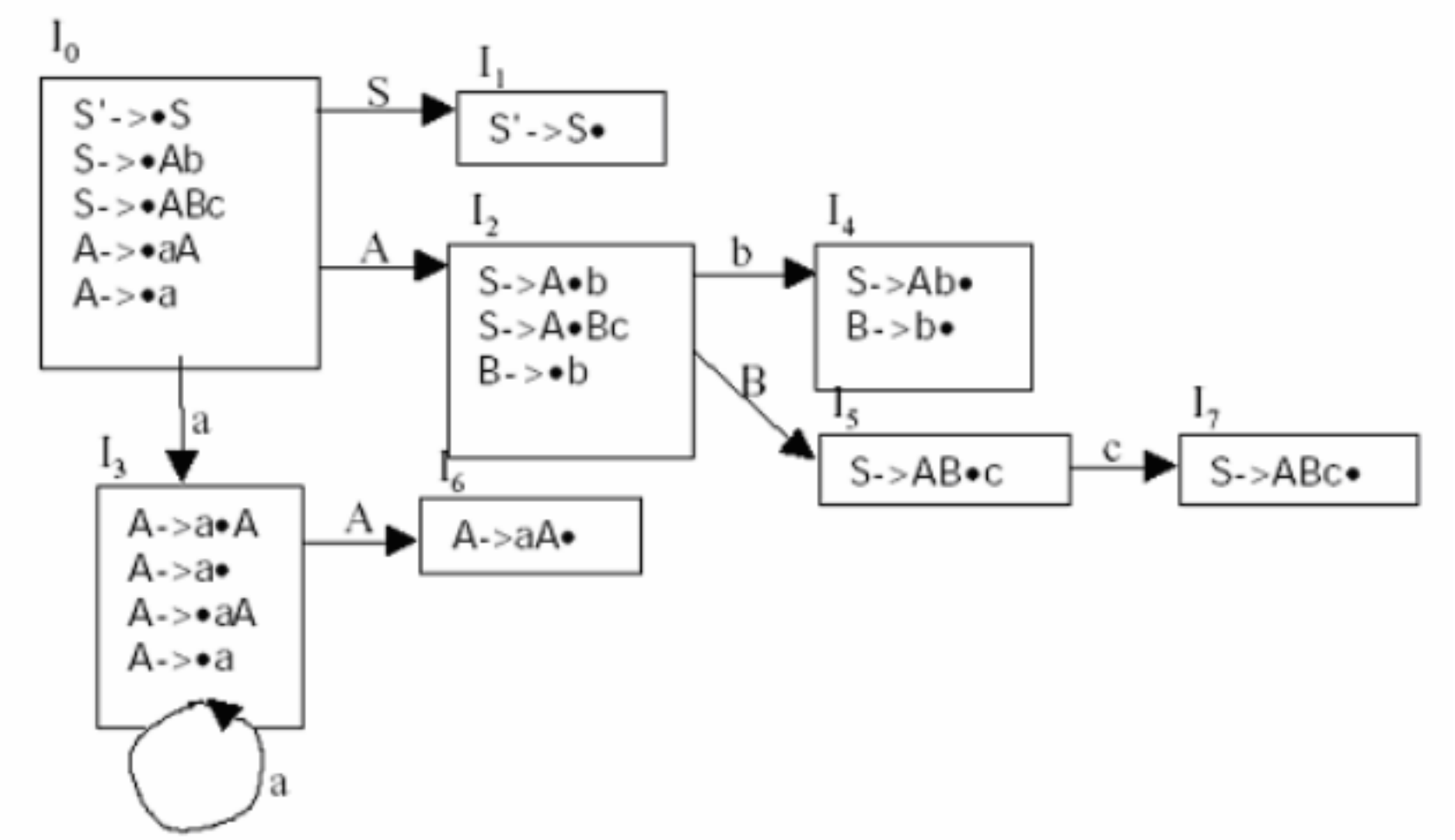
构造识别活前缀的有限状态机如下：



可验证：状态 I_3 有移进 - 归约冲突，所以 G[E] 不是 LR(0) 文法；进一步，因 Follow(T)={+,},},#} ，不含 [，所以 G[E] 是 SLR(1)文法。

b) 增加产生式 S’→S，得增广文法 G’[S’]

构造识别活前缀的有限状态机如下：



I4 存在归约 / 归约冲突， I3 存在归约 / 移进冲突 . 因此不是 LR(0) 文法。

考察能否使用 SLR(1)方法解决冲突： I4 中，因为 Follow(S) = {#} 而 Follow(B) = {c}. 所以可以解决。 I3 中，因 Follow(A) = {b} ，不含 a，因此该移进 / 归约冲突也可解决 . 文法是SLR(1)文法

参考解答：

为下列文法构造 LR（1）分析表，并给出串 baab# 的分析过程：

增广文法文法 G[S’]:

(0) S' → S

(1) S → XX

(2) X → aX

(3) X → b

其中 S’,S,X 为非终结符，其余符号为终结符

参考解答：

参见教材 P146 的例子， 自行补充对输入串 baab#的分析过程

问题 2：

（1）构造下列文法 G（P’）的 LR（1）FSM，验证它是 LR（1）文法：

(0) P' → P

(1) P → P(P)

(2) P → Aa

(3) P → ε

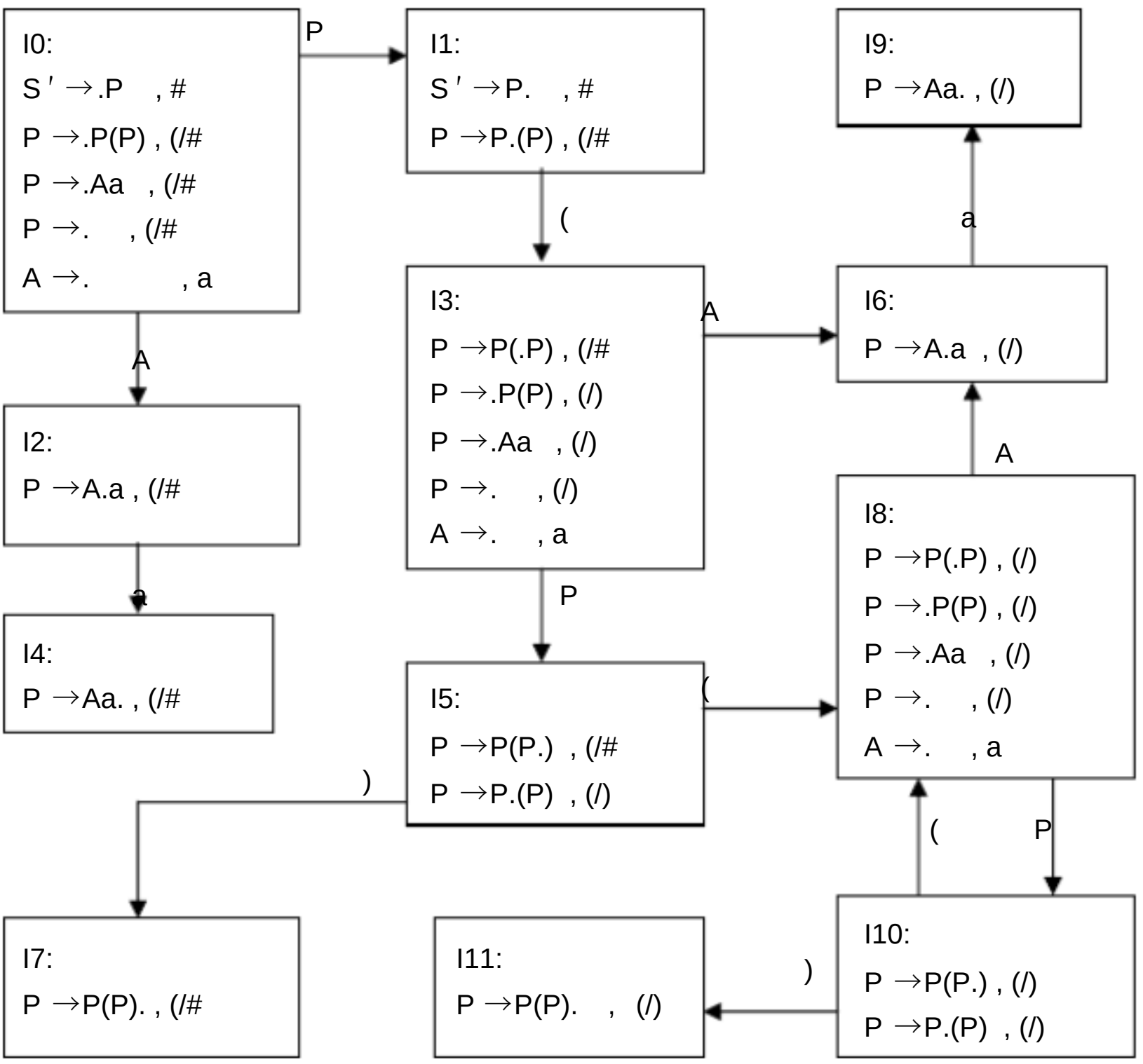
(4) A → ε

其中 P’,P,A 为非终结符

（2）通过合并同芯集（状态）的方法构造相应于上述 LR（1）FSM的 LALR（1）FSM，并判断 G（P’）是否 LALR（1）文法？

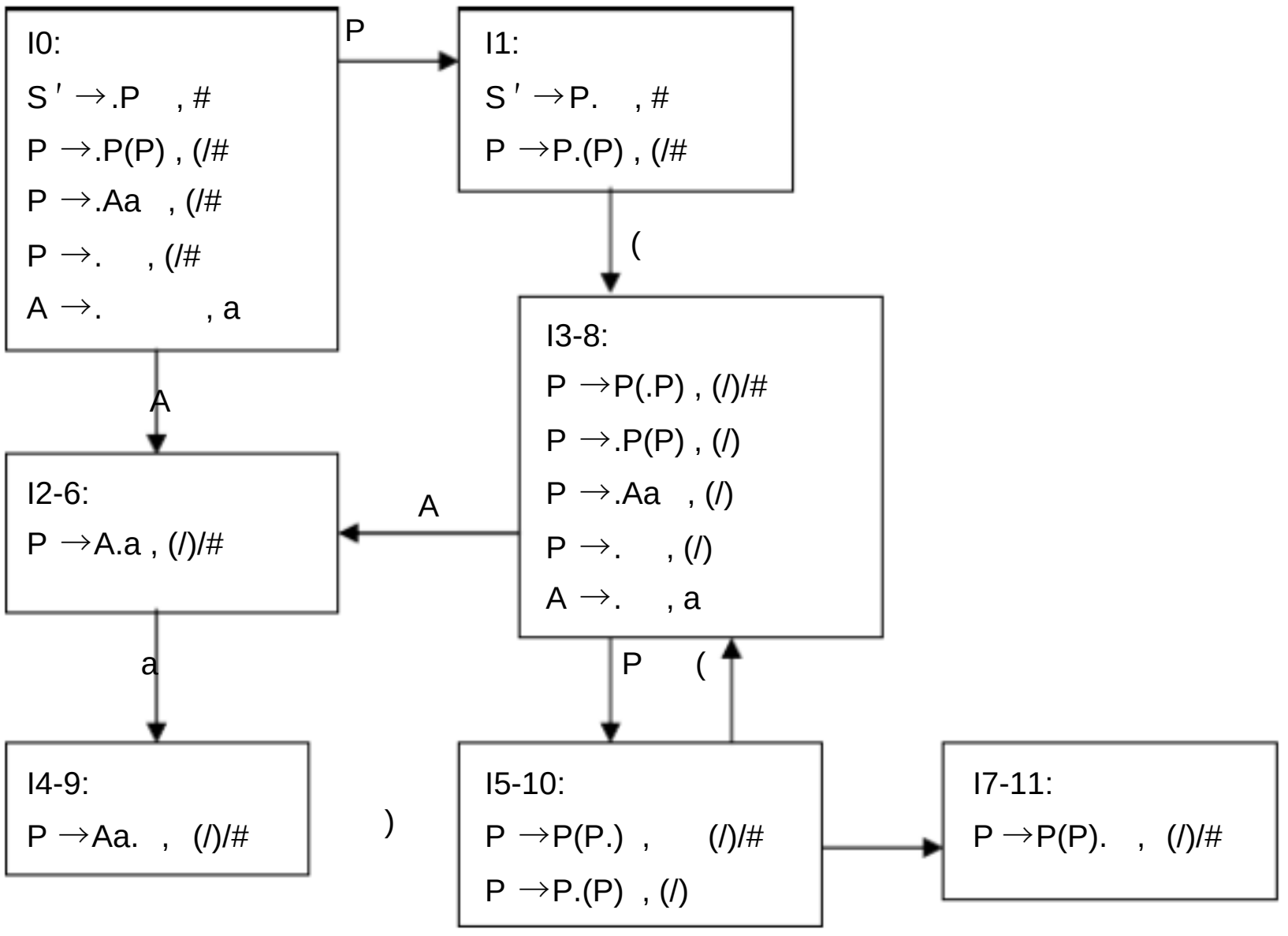
答案：

（1） 构造文法 G（P’）的 LR（1） 项目集族和转换函数如下：



其中的每个状态均无宏冲突，所以 G（P’）是 LR（1） 文法。

（2） 可以看出： I2 和 I6，I3 和 I8，I4 和 I9，I5 和 I10，I7 和 I11 是同芯状态．通过合并同芯状态的方法构造相应于上述 LR（1） 转换图的 LALR（1） 转换图 如下：



可以看出： 所有状态都不存在移进 - 归约和归约 - 归约冲突 . 所以， G（P’）是 LALR（1）文法。

问题 3：

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’:

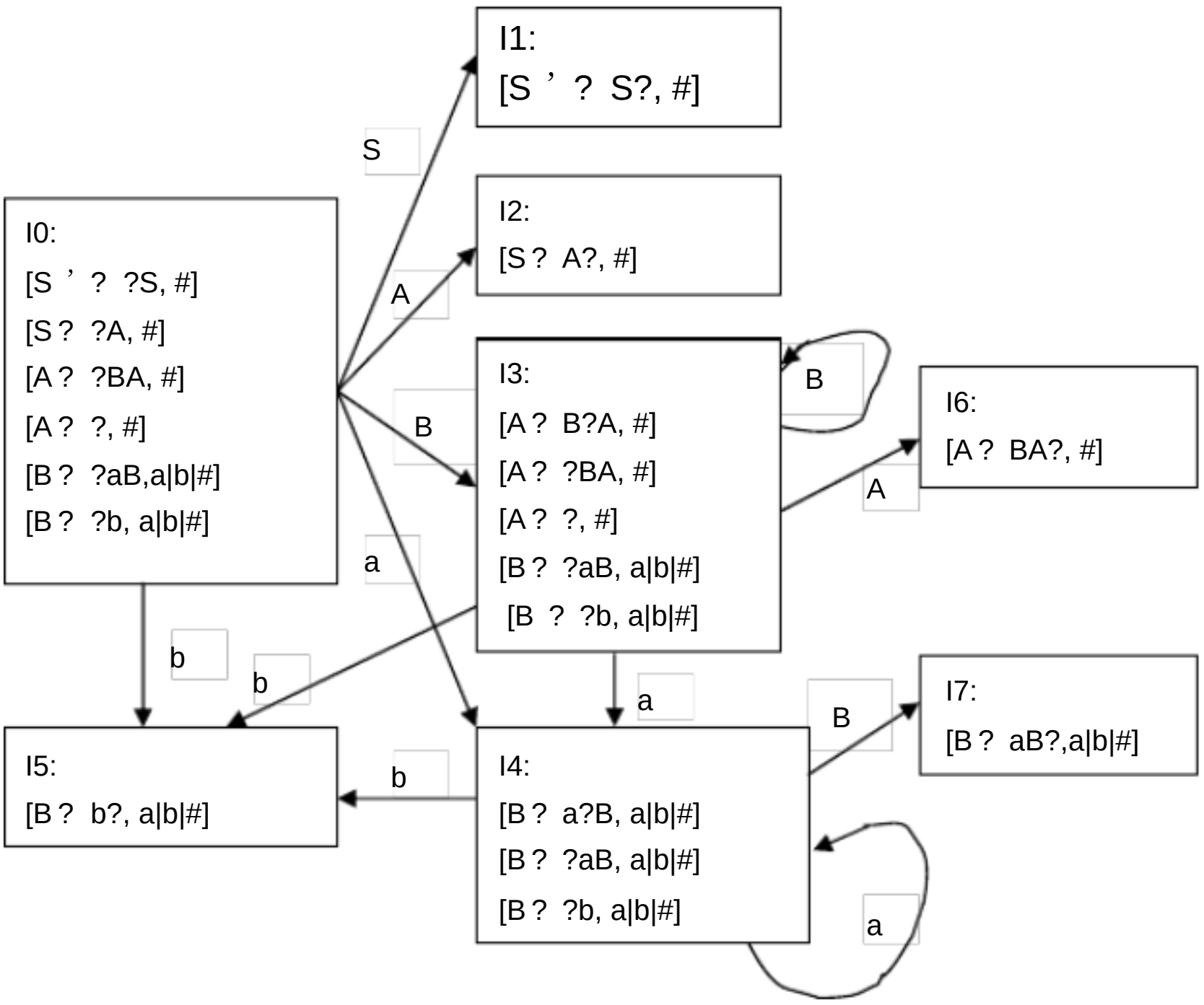
(0) S ’？ S (1) S ？ A (2) A ？ BA

(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) 分析表如下：

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

问题4：

给定文法G ’(S ’):

S ’ → S

S → （L） ？ a

L → L , S ？ S

试为该文法配上属性计算的语义规则（或动作）集合（即设计一个属性文法）， 它输出配对括号的个数。如对于句子 (a,(a)) ， 输出是 2。

答案：

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→ L_1,S {L.num:= L_1.num+S.num}

L→S {L.num:=S.num}

问题5：

给定文法 G[S]:

S → （L） ? a

L → L , S ? S

如下是相应于 G[S]的一个属性文法：

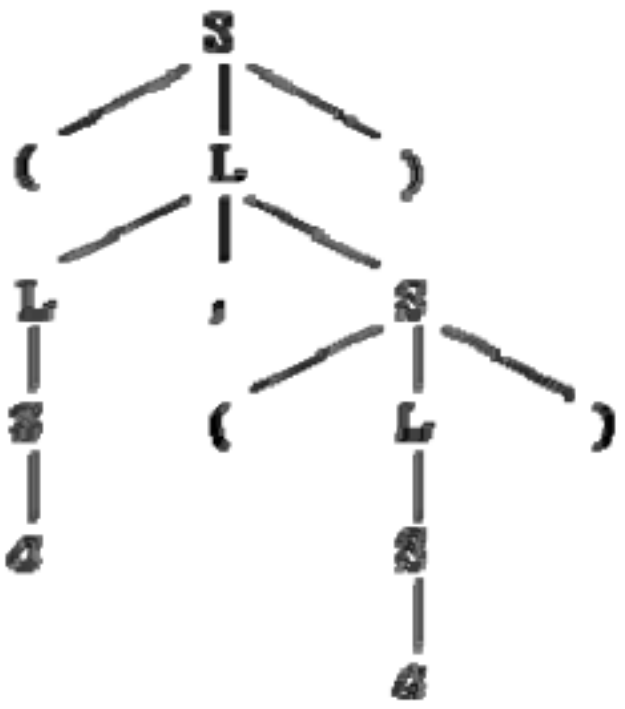
(1) S → （L） { S.num := L.num +1; }

(2) S → a { S.num := 0; }

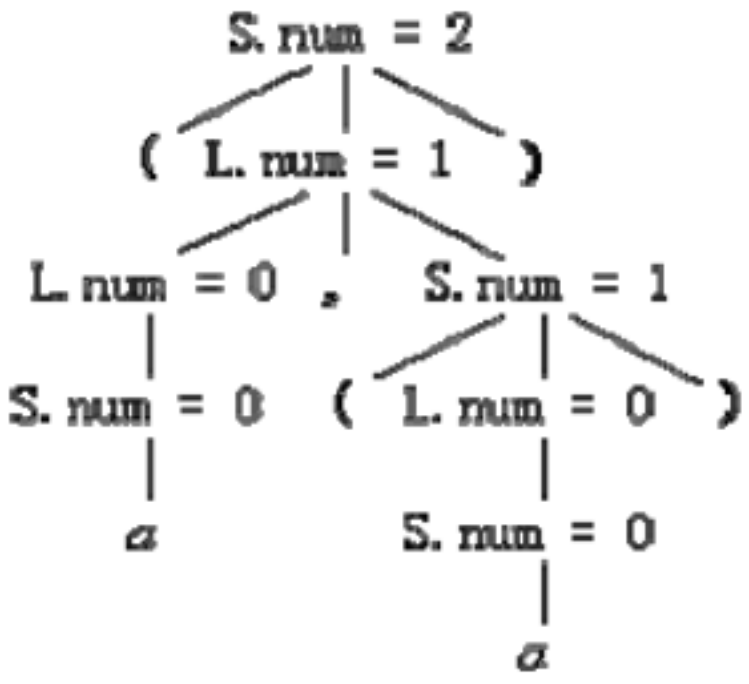
(3) L → L_1 , S { L.num := L_1.num + S.num; }

(4) L → S { L.num := S.num; }

下图分别是输入串 （ a, （ a ））的语法分析树和对应的带标注语法树，但其属性值没有标出，试将其标出（即填写右下图中符号 “ = ” 右边的值）。



答案：



问题 6：

对上题中所给的 G[S]的属性文法是一个 S-属性文法，故可以在自下而上分析的过程中，增加一个语义栈来计算属性值。下图（ a）是 G[S]的一个 LR 分析表，图(b) 描述了输入串（ a,（ a））的分析和计值过程（语义栈中的值对应 S.num 或 L.num ），其中，第 14），15）行没有给出，试补齐之。

状态	ACTION					GOTO	
	a	,	(	)	#	S	L
0	s_3		s_2			1	
1					acc		
2	s_3		s_2			5	4
3		r_2		r_2	r_2		
4		s_7		s_6			
5		r_4		r_4			
6		r_1		r_1	r_1		
7	s_3		s_2			8	
8		r_3		r_3			

题 3 图（a）

步骤	状态栈	语义栈	符号栈	余留符号串
1)	0	-	#	(a,(a))#
2)	02	- -	#(	a,(a))#
3)	023	- - -	#(a	,(a))#
4)	025	- - 0	#(S	,(a))#
5)	024	- - 0	#(L	,(a))#
6)	0247	- - 0 -	#(L,	(a))#
7)	02472	- - 0 - -	#(L,(	a))#
8)	024723	- - 0 - - -	#(L,(a	))#
9)	024725	- - 0 - - 0	#(L,(S	))#
10)	024724	- - 0 - - 0	#(L,(L	))#
11)	0247246	- - 0 - - 0 -	#(L,(L)	)#
12)	02478	- - 0 - 1	#(L,S	)#
13)	024	- - 1	#(L	)#
14)				
15)				
16)	接受			

题 3 图（b）

答案：

14）	0246	- - 1 -	# (L)	#
15）	01	- 2	# S	#

问题 7：

下面的属性文法 G[N]可以将一个二进制小数转换为十进制小数, 令 N.val 为 G[N]生成的二进制数的值, 例如对输入串 101.101, N.val=5.625.

N → S^1 ‘ ? ’ S^2 { N.val := S^1.val + 2 - S^2.len *S^2.val ; }

S → S^1 B { S.val := 2 * S^1.val + B.val; S.len := S^1.len + 1 }

S → B { S.val = B.val; S.len:= 1 }

B → ‘ 0 ’ {B.val :=0}

B → ‘ 1 ’ {B.val:= 1}

（1） 试消除该属性文法（翻译模式）中的左递归，以便可以得到一个可以进行自上而下进行语义处理（翻译）的翻译模式；

（2） 对变换后的翻译模式，构造一个自上而下预测翻译程序。

答案：

（1）消去原文法中的左递归：

N → S ‘ ? ’ S

S → BR

R → BR

R → ε

B → ‘ 0 ’

B → ‘ 1 ’

可验证该文法是 LL（1）文法。

再考虑语义规则，得到如下翻译模式：

N → S^1 ‘ ? ’ S^2 { N.val := S^1.val + 2 - S^2.len *S^2.val ; }

S → B {R.ival := B.val; R.ilen := 1} R { S.val := R.sval; S.len := R.slen }

R → B {R^1.ival:=2*R.ival+B.val; R^1.ilen:=R.ilen+1} R^1 {R.sval:=R^1.sval; R.slen:=R^1.slen}

R → ε{R.sval:=R.ival; R.slen:=R.ilen}

B → ‘ 0 ’ {B.val :=0}

B → ‘ 1 ’ {B.val :=1}

(2) 对变换后的翻译模式，可构造一个如下的自上而下预测翻译程序：

```
real ParseN()
{
(S1val,S1len) := ParseS();   //  变量 S1val,S1len  分别对应属性 S1.val, S1.len
    MatchToken( ‘ ? ’ );   //  匹配 ‘ ? ’ , 函数 MatchToken 参见第 4 讲, 本例省略
(S2val,S2len) := ParseS();   //  变量 S2val,S2len  分别对应属性 S2.val, S2.len
    Nval := S1val + 2 ^ (-S2len) * S2val  ; /  变量 Nval  对应属性 N.val
    return Nval;
}
(int, int) ParseS()   //  （int, int ）代表一个记录 / 结构类型
{
```

```
    Bval := ParseB();
    Rival := Bval;
    Rilen := 1;
    (Rsval, Rslen) := ParseR(Rival, Rilen);
        Sval := Rsval;
        Slen := Rslen ;
        return (Sval,Slen);
}
(int, int) ParseR(int Rival, int Rilen)
{
  switch (lookahead)  {          // lookahead                    为下一个输入符号
      case          ′0′,   ′1′:
                  Bval := ParseB();
                  R1ival := 2*Rival+Bval;
                     R1ilen := Rilen+1      ;
                  (R1sval, R1slen) := ParseR(R1ival, R1ilen);
                     Rsval := R1sval;
                     Rslen := R1slen ;
          break;
              case   ′?′,   ′#′:
                 Rsval:=Rival;
                    Rslen:=Rilen;
          break;
       default:
          printf("syntax error \n")
          exit(0);
   }
       return (Rsval,Rslen);
}
```

```
int ParseB()
{
  switch (lookahead)  {
      case          ′0′:
                    MatchToken(‘0’);
                 Bval := 0;
          break;
              case   ′1′:
                    MatchToken(‘1’);
                 Bval := 1;
          break;
       default:
          printf("syntax error \n")
          exit(0);
```

```
    }
        return Bval;
}
```

问题 8：

对于上题（1）所得到的翻译模式（结果应满足 L-属性的条件），在进行自下而上的语义处理时，语义栈中的值有两个分量，分别对应文法符号的综合属性 val 和 len 。

（1） 若该翻译模式中，嵌在产生式中间的语义规则集中含有除复写规则之外的语义规则，则变换该翻译模式，使嵌在产生式中间的语义规则集中仅含复写规则；

（2） 根据（1）所得到的新翻译模式，文法符号的所有继承属性均可以通过归约前已出现在分析栈中的综合属性进行访问。试写出在按每个产生式归约时语义处理的代码片断（设语义栈由向量 v 表示，归约前栈顶位置为 top，语义值 v[i] 的两个分量分别用 v[i].val 和 v[i].len 表示）。

答案：

(1) 将如下翻译模式

N → S^1 ‘?’ S^2 { N.val := S^1.val + 2 - S^2.len *S^2.val ; }

S → B {R.ival := B.val; R.ilen := 1} R { S.val := R.sval; S.len := R.slen }

R → B {R^1.ival:=2*R.ival+B.val; R^1.ilen:=R.ilen+1} R^1 {R.sval:=R^1.sval; R.slen:=R^1.slen}

R → ε{R.sval:=R.ival; R.slen:=R.ilen}

B → ‘0’ {B.val :=0}

B → ‘1’ {B.val :=1}

变换为：

N → S^1 ‘?’ S^2 { N.val := S^1.val + 2 - S^2.len *S^2.val ; }

S → B { M.bval := B.val } M {R.ival := M.sval; R.ilen := M.val} R { S.val := R.sval; S.len := R.slen }

R → B {P.rival:= R.ival; P.rilen:= R.ilen; P.bval:=B.val; } P {R^1.ival:= P.sval; R^1.ilen:= P.slen } R^1 {R.sval:=R^1.sval; R.slen:=R^1.slen}

R → ε{R.sval:=R.ival; R.slen:=R.ilen}

B → ‘0’ {B.val :=0}

B → ‘1’ {B.val :=1}

M → ε { M.sval := M.bval; M.val := 1 }

P → ε{P.sval = 2*P.rival + P.bval; P.sleP:= P.rilen +1}

(2) 按每个产生式归约时语义处理的代码片断 ：

N → S^1 ‘?’ S^2 {v[top-2].val := v[top-2].val + 2^(-v[top].len) * v[top].val; }

S → B M R {v[top-2].val := v[top].val; v[top-2].len := v[top].len }

R → B P R^1 {v[top-2].val := v[top].val; v[top-2].len := v[top].len }

R → ε {v[top+1].val := v[top].val; v[top+1].len := v[top].len }

B → ‘0’ {v[top].val :=0}

B → ‘1’ {v[top].val :=1}

M → ε { v[top+1].val := v[top].val; v[top+1].len := 1}

P → ε { v[top+1].val := 2*v[top-1].val+v[top].val; v[top+1].len := v[top-1].len+1}

问题 9：

证明 LR 分析过程正确性的一个重要引理：由构造 LR(0) 项目集规范族得到的 DFA，它可以也只能读进所分析文法的活前缀。

需要证明两个方面：

命题 1 所有活前缀一定都可由 DFA 读进，即不会错过合法的归约。

命题 2 DFA 只能读活前缀。

证明思路：

首先要了解活前缀是如何产生的。活前缀的集合 Prefix 可归纳定义如下：

（1） 设 S’是增广文法的开始符号，既有产生式 S’→ S（S 是原文法的开始符号），令 S ∈Prefix。

（2） 若 v ∈Prefix，则 v 的任一前缀 u 都是活前缀，即 u∈Prefix。

（3） 若 v ∈Prefix，且 v 中至少包含一个非终结符，即可以将 v 写成 αBγ，其中 B 为非终结符。若有产生式 B → β，则 αβ的任一前缀 u 都是活前缀，即 u∈Prefix。

（4） Prefix 中的元素只能通过上述步骤产生。

命题 1 可以根据 Prefix 的定义进行归纳证明。

基础：对于由规则（1）产生的活前缀 S∈Prefix，由于在 DFA 的初始项目集（状态） I_0 中含有项目 S’→ ?S，显然 S 是可以被 DFA 读进的。

归纳：若 v ∈Prefix，且 v 可以被 DFA 读进，则 v 的前缀可以被 DFA 读进也是显然的。

若 v ∈Prefix，且可以将 v 写成 αBγ，其中 B 为非终结符并有产生式 B → β。设 DFA 在从初态 I_0 读进 α 后进入状态 I。因为 v= αBγ 可以被 DFA 读进，所以在状态 I 可以读进 B，根据 DFA 的构造过程，一定存在项目 C → α?Bγ ∈I。由闭包的计算过程，可知一定有 B → ?β∈I，因此 β是从 I 开始后续的可读串。所以，从初态 I_0 开始，αβ的任一前缀 u 都是可读进的。

命题 1 证毕。

命题 2 可归纳于 DFA 可读序列的长度 n 来证明。

基础： n=0。显然空序列 ε是活前缀。

归纳：设 αX 是 DFA 可读序列，且？αX ？= n +1, 其中 X 是某个文法符号。在读过 α 后，DFA 一定处于状态 I，在此状态下 X 是可读的。根据 DFA 的构造过程，一定存在项目 C → β?X γ∈I。该项目或者是基本项目，或者是通过闭包计算得到的项目。

下面分这两种情形来讨论。

1） C → β?X γ 是基本项目。

若？β？= 0，则该项目只能是 S'→ ?S，此时 αX = S 显然是活前缀。

若？β？= m ≠0，则 DFA 从状态 I_0 读进 n - m 个符号的序列 μ后进入状态 I'，且必定有 C → ？β?X γ∈I'。根据 DFA 的构造过程，一定存在项目 B → ?C γ ∈I'。这样在状态 I'下，可以读进 B。因为？μB？≤n，由归纳假设，可知 μB 是活前缀。因此，由活前缀集合的归纳定义，得知 μBC = αX 是活前缀。

2） C → β?X γ 是通过闭包计算得到的项目。

此时，β一定为空序列，而项目 C → ?X γ一定是由 I 中的某个基本项目 B → μ?C_0 ν 直接或间接地通过闭包计算序列得到的：C_0 → ?C_1 γ_1，C_1 → ?C_2 γ_2，…，C_k → ?C γ_{k+1}。由 1）的讨论结果，可知 αC_0 是活前缀。从而 αC_1，αC_2，…，αC_k 都是或前缀，αC 也是活前缀。所以，αX 是活前缀。

命题 2 证毕。

第 8 章　语法制导翻译和中间代码生成

第 1 题

给出下面表达式的逆波兰表示　(后缀式)：

(1)a*(-b+c)

(2) if(x+y)*z=0 then s　：=(a+b)*c else s　：=a*b*c

答案：

给出下面表达式的逆波兰表示　(后缀式)：

(1) ab-c+*

(2)xy+z*0= sab+c*:= sab*c*:= ￥（注：￥表示　if-then-else 运算）

如果写成这样：　xy+z*0= sab+c*:= sabc**:= ￥，则是错误的，因为写表达式和赋值语句的中间代码序列，　或是写它们的代码生成过程，　必须注意按照算符优先序进行，　这实际上是按照 LR 分析过程进行的。例如：写出赋值语句　a:=a+b*c*(d+e) 的四元式中间代码　,当前四元式序号为 100。不能写成：

100　(+,d,e,t1)

101　(*,b,c,t2)

102　(*,t2,t1,t3)

103　(+,a,t3,t4)

104　(:= ,t4,-,a)

应该写成：

100　(*,b,c,t1)

101　(+,d,e,t2)

102　(*,t1,t2,t3)

103　(+,a,t3,t4)

104　(:=,t4,-,a)

第 2 题

请将表达式 -(a+b)*(c+d)-(a+b+c) 分别表示成三元式、间接三元式、四元式序列、树形、逆波兰，当前序号为　100。

答案：

三元式：

100 (+,　a, b)

101 (+,　c, d)

102　(* , (1), (2))

103　(-, (3), /)
104　(+, a, b)
105　(+,(5),c)
106　(- (4), (6))

间接三元式：

间接三元式序列			间接码表
100	(+，	a, b)	(100)
101	(+，	c, d)	(101)
102	(*，	(1), (2))	(102)
103	(-，	(3), /)	(103)
104	(+，	(1), c)	(100) (104)
105	(-，	(4), (1))	(105)

或者：

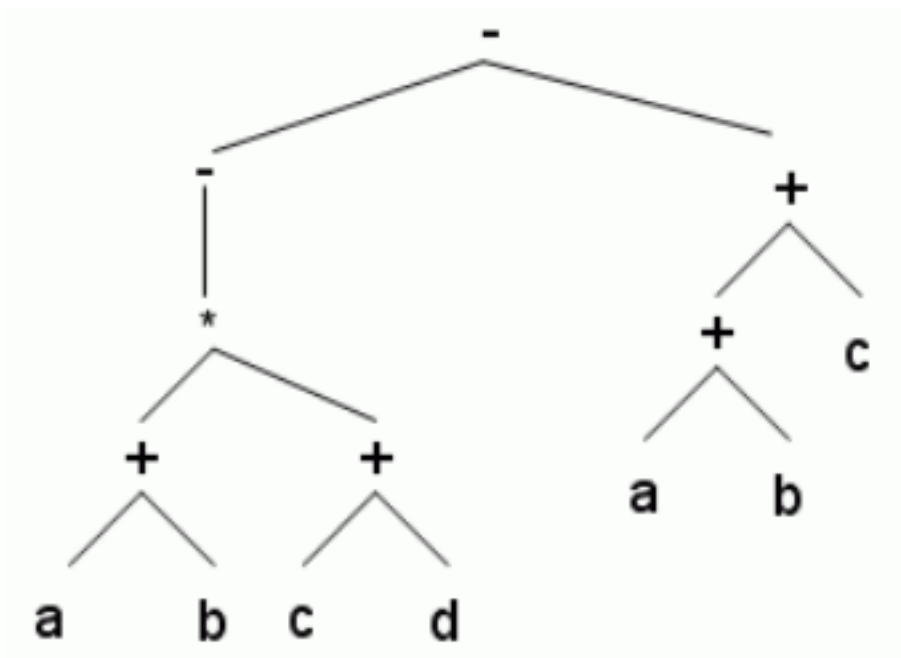
间接三元式：
100　(+ ，　a, b)
101　(+，　c, d)
102　(*，　(1), (2))
103　(-，　(3), /)
104　(+，　(1), c)
105　(-，　(4), (1))

间接码表：　100　101　102　103　100　104　105

四元式：

100　(+, a, b, t1)
101　(+, c, d, t2)
102　(*, t1, t2, t3)
103　(-, t3, /, t4)
104　(+, a, b, t5)
105　(+, t5, c, t6)
106　(-, t4, t6, t7)

树形：



逆波兰： ab+cd+*-ab+c+-

［典型例题］ ：

写出 if A and B and C > D then
if A <B then F:=1
else F:=0
else G:=G+1; 的四元式序列 ， 翻译过程中 ， 采用 then 与 else 的最近匹配原则。

(1) (jnz,A,_,3) /* A and B and C>D 的四元式 */
(2) (j, _, _, 13)
(3) (jnz,B,_,5)
(4) (j, _, _, 13)
(5) (j>,C,D,7)
(6) (j, _, _, 13)
(7) (j<,A,B,9) /* A < B 的四元式 */
(8) (j, _, _, 11)
(9) (:=,1, _, F) /* F:=1 */
(10) (j, _, _,14)
(11) (:=, 0, _,F) /* F:=0 */
(12) (j, _, _,14)
(13) (:=,G ,1,G)
(14)

［典型例题］ ：

写出 WHILE A<C AND B<D DO
IF A=1 THEN C:=C+1 ELSE
WHILE A<=D DO A:=A+2; 的四元式序列。
(100) (j<,A,C,102)
(101) (j,-,-,114)
(102) (j<,B,D,104)

(103)　(j,-,-,114)

(104)　(j=,A,1,106)

(105)　(j,-,-,109)

(106)　(+,C,1,T)

(107)　(:=,T,-,C)

(108)　(j,-,-,100)

(109)　(j<=,A,D,111)

(110)　(j,-,-,100)

(111)　(+,A,2,T)

(112)　(:= ,T,-,A)

(113)　(j,-,-,109)

(114)

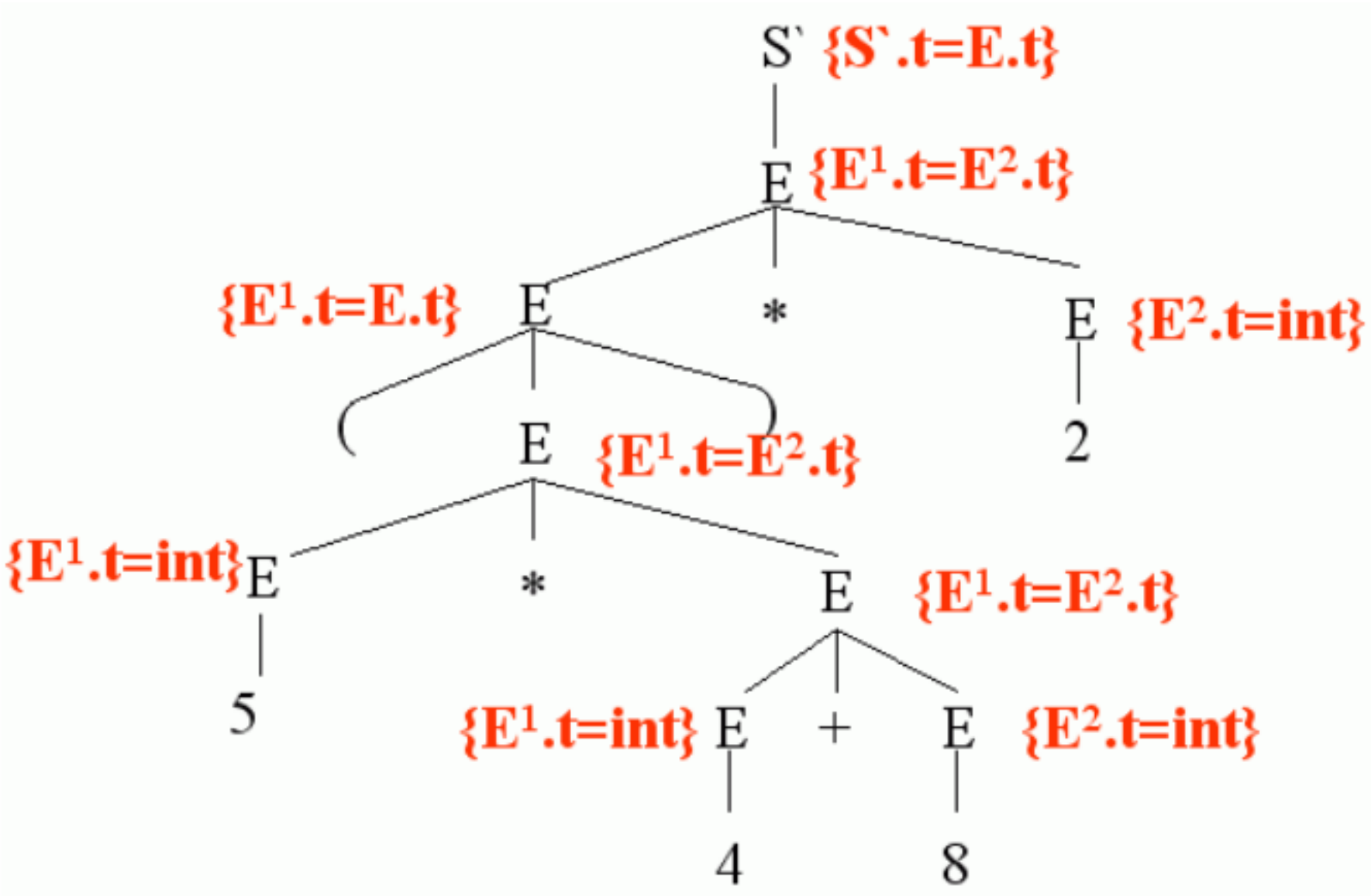
第 3 题

采用语法制导翻译思想，表达式 E 的“值”的描述如下：

产生式	语义动作
(0) S′→E	{print E.VAL}
(1) $E \rightarrow E^1+E^2$	{E.VAL：=E^1.VAL+E^2.VAL}
(2) $E \rightarrow E^1*E^2$	{E.VAL：=E^1.VAL*E^2.VAL}
(3) $E \rightarrow (E^1)$	{E.VAL：=E^1.VAL}
(4) E→n	{E.VAL：=n.LEXVAL}

如采用 LR 分析方法，给出表达式 (5*4+8)*2 的语法树并在各结点注明语义值 VAL。

答案：



采用语法制导翻译思想，表达式 E 的“值”的描述如下：

产生式 语义动作

(0) S′→E {print E.VAL}

(1) $E \rightarrow E^1+E^2$ {E.VAL ：=E^1.VAL+E^2.VAL}

(2) $E \rightarrow E^1*E^2$ {E.VAL ：=E^1.VAL*E^2.VAL}

(3) $E \rightarrow (E^1)$ {E.VAL ：=E^1.VAL}

(4) E→n {E.VAL ：=n.LEXVAL}

假如终结符 n 可以是整数或实数，算符 +和*的运算对象类型一致，语义处理增加“类型匹配检查”，请给出相应的语义描述。

答案：

(0) S′→E { if error ≠1 then print E.VAL}

(1) $E \rightarrow E^1+E^2$ { if E^1.TYPE=int AND E^2.TYPE=int then
begin
E.VAL:=E^1.VAL + E^2.VAL;
E.YTPE:=int;
end
else if E^1.TYPE=real AND E^2.TYPE=real then
begin
E.VAL:=E^1.VAL + E^2.VAL;
E.YTPE:= real;
end
else error=1
}

(2) $E \rightarrow E^1*E^2$ { if E^1.TYPE=int AND E^2.TYPE=int then
begin
E.VAL:=E^1.VAL * E^2.VAL;;
E.YTPE:= int;
end
else if E^1.TYPE=real AND E^2.TYPE=real then
begin
E.VAL:=E^1.VAL * E^2.VAL;;
E.YTPE:=real;
end
else error=1
}

(3) $E \rightarrow (E^1)$ { E.VAL:=E^1.VAL;
E.TYPE:=E^1.TYPE }

(4) E→n { E.VAL:=n.LEXVAL;
E.TYPE:=n.LEXTYPE }

第 5 题

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

按照语法制导翻译的方法，对每个产生式给出相应的语义规则。 （中国科学院计算所 1995 年）

答案： 加入新的开始符号 S`和规则 S`? S，得到增广文法。语法制导定义如下：

产生式	语义规则
S`? S	print(S.val)
S? $L_1.L_2$	$S.val:=L_1.val+L_2.val/2^{L2.length}$
S? L	S.val:=L.val
L? L_1B	L.val:=L_1.val*2+B.val L.length:=L_1.length+1
L? B	L.val:=B.val L.length:=1
B? 0	B.val:= 0
B? 1	B.val:=1

如果题目是 S::=L.L | L L::=LB | B B::=0 | 1 则写成：

S`::=S {print(S.val);}

S::=$L_1.L_2$ { S.val:=L_1.val+L_2.val/$2^{L2.length}$;}

S::= L { S.val:=L.val; }

L::=L_1B { L.val:= L_1.val*2+B.val; L.length:=L_1.length+1; }

L::=B { L.val:=B.val; L.length:=1;}

B::= 0 { B.val:=0; }

B::=1 { B.val:=1;}

第 6 题

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

（1） 给出语法制导定义确定每个子表达式的类型。

（2） 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：

（1）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

（2）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
=
=
End
	(E1.type integer) and (T.type=integer) THEN
	begin
	E.type:= integer
	E.code= '+'||E1.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= '+'||E1.code||T.code;

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)

第 7 题

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

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。

答案：

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

附加题

问题 1：

请将下列语句

while (A<B do if (C>D) then X:=Y+Z

翻译成四元式

答案：

假定翻译的四元式序列从（ 100）开始：

（100） if A<B goto （102）

（101） goto （107）

（102） if C<D got （104）

（103） goto （100）

（104） T ：=Y+Z

（105） X ：=T

（106） goto （100）

（107）

问题 2：

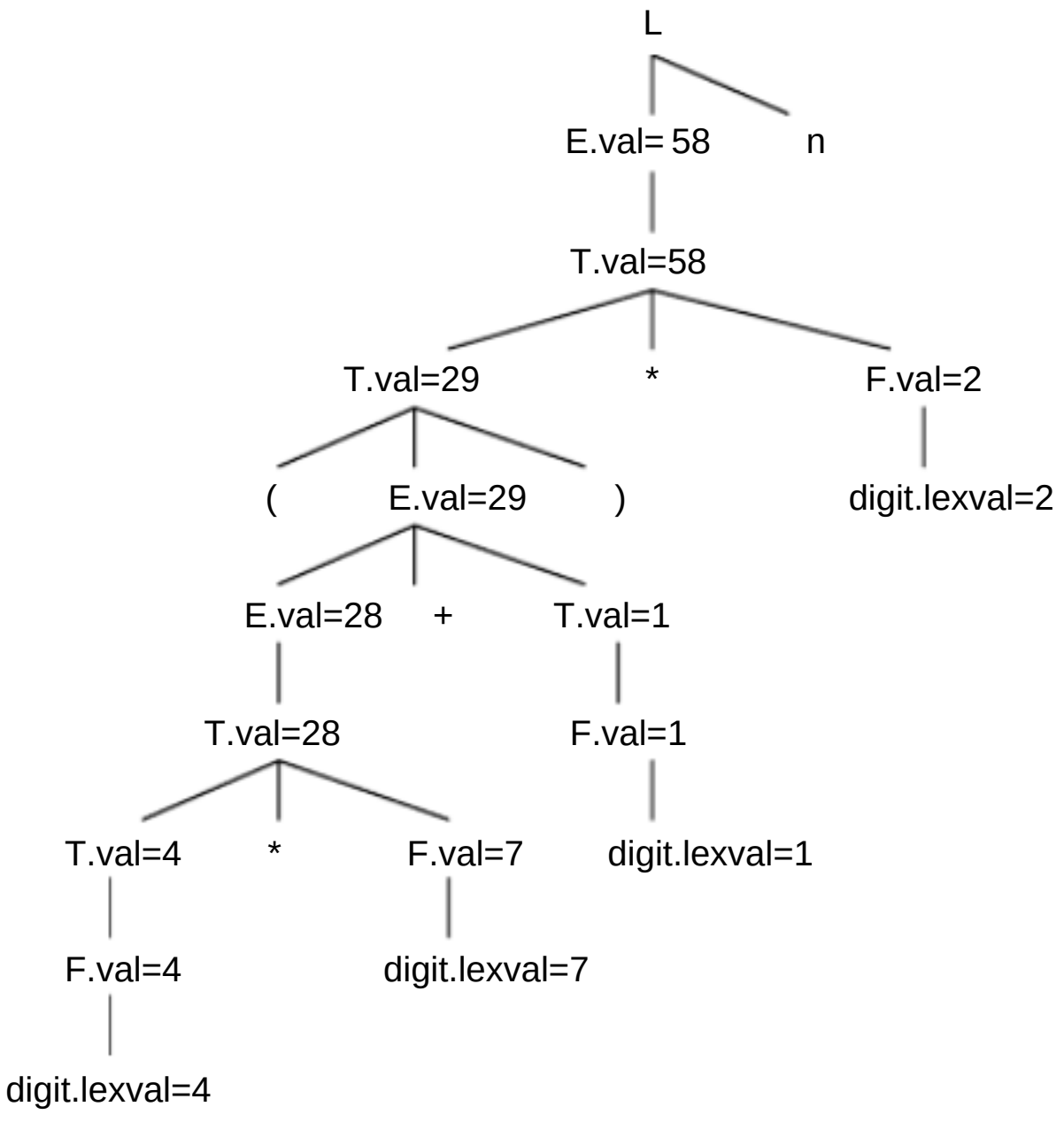
对于输入的表达式 (4*7+1)*2 ， 根据下表的语法制导定义建立一棵带注释的分析树。

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：


L
E.val= 58
n
T.val=58
T.val=29
*
F.val=2
(
E.val=29
)
digit.lexval=2
E.val=28
+
T.val=1
T.val=28
F.val=1
T.val=4
*
F.val=7
digit.lexval=1
F.val=4
digit.lexval=7
digit.lexval=4

问题 3：

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;

问题 4：

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。（中国科学院计算所 1994）

答案：

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，则语法制导定义如下：

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

问题 5：

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。（中国科学院软件所 1999）

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

问题 6：

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

问题 7：

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	{R.i:= T.type} {R.j:= T.val} {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}

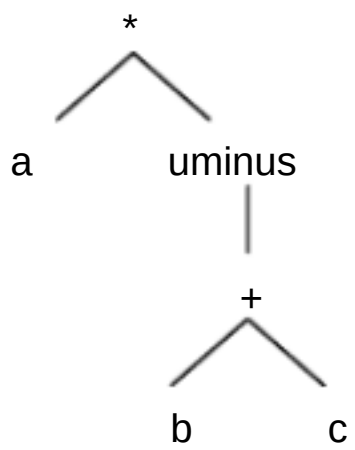
问题 8：

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

问题 9：

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

问题 10：

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
```

问题 11：

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 ||gen(S.curr “ := ” S.first) ||gen(S.begin “ : ”) ||gen(“ if ” S.curr “ > ” S.Last “ goto ” S ||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

问题 12：

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 , L1 =	{L.type := L1.type L.width := L1.width enter(id.name , L1.type , offset) offset := offset + L1.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 T1	{T.type:=array(num.val , T1.Type T.width := num.val * T1.Width)
T？ ^T1	{T.type := pointer(T1.type) T.width := 4}

问题 13：

（浙江大学 1997 年）在一个移入－归约的分析中采用以下的语法制导的翻译模式，在按一产生式归约时，立即执行括号中的动作。

A→aB {print “0”;}

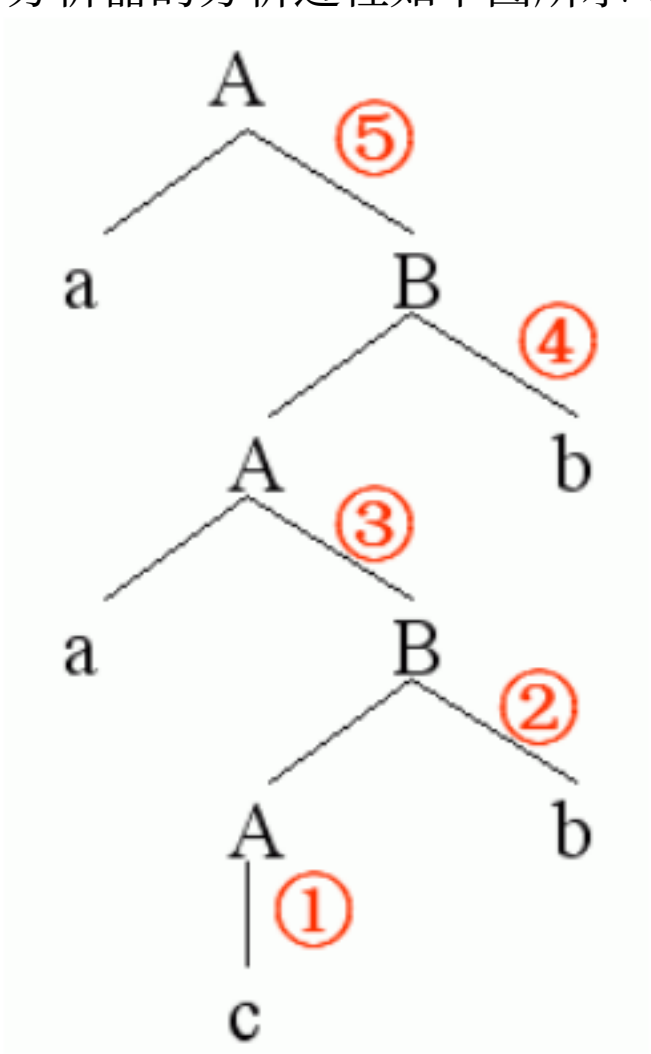
A→c {print “1”;}

B→Ab {print “2”}

当分析器的输入为 aacbb 时，打印的字符串是什么？

答案：

分析器的分析过程如下图所示：



由于分析器采用移入－归约的方式进行分析，符号串 aacbb 的分析过程将如图中所标的归约顺序进行，而在按一产生式归约时，立即执行括号中的动作，所以分析器打印的字符为 12020。

第 9 章　符号表

第 1 题：

根据你所了解的某个　FORTRAN　语言的实现版本，该语言的名字作用域有哪几种　　？

答案：

FORTRAN 中，名字作用域有四种：
<1>在 BLOCK DATA　块中定义的标识符，其作用域是整个程序。
<2>在 COMMON　块中定义的标识符，　其作用域是声明了该　COMMON 块的所有例程（包括函数和过程）。
<3>在例程中定义的标识符（包括哑变量）　　，其作用域是声明该标识符的例程。
<4>在例程中用　SAVE 定义的标识符，其作用域是声明该标识符的例程，且在退出该例程时，该标识符的值仍保留（即内部静态量）　　。

第 2 题：

C 语言中规定变量标识符的定义可分为　　extern,extern static,auto,local static　　和 register 五种存储类：
(1) 对五种存储类所定义的每种变量，分别说明其作用域。
(2) 试给出适合上述存储类变量的内存分配方式。
(3) 符号表中登录的存储类属性，在编译过程中支持什么样的语义检查。

答案：

(1) extern 定义的变量，其作用域是　整个 C 语言程序 。
extern static 定义的变量，其作用域是　该定义所在的　C 程序文件 。
auto 定义的变量，其作用域是　该定义所在的例程　。
local static　定义的变量，其作用域是　该定义所在的例程　。且在退出该例程时，该变量的值仍保留。
register 定义的变量，　其作用域与　auto 定义的变量一样。　这种变量的值，　在寄存器有条件时，可存放在寄存器中，以提高运行效率。

(2) 对 extern 变量，设置一个　全局的静态公共区　进行分配。
对 extern static 变量，为每个　C 程序文件，分别设置一个　局部静态公共区　进行分配。
对 auto 和 register 变量，设定它们在该　例程的动态区中的相对区头的位移量　　。而例程动态区在运行时再做动态分配。
对 local static 变量，为每个具有这类定义的例程，分别设置一个　　内部静态区 进行分配。

(3) 实施标识符变量重复定义合法性检查，及引用变量的作用域范围的合法性检查。

第 3 题：

对分程序结构的语言，为其符号表建立重名动态下推链的目的是什么 ?概述编译过程中重名动态下推链的工作原理。

答案：

重名动态下推链的目的是, 保证在合法重名定义情况下, 提供完整确切的符号表项, 从而保证引用变量的上下文合法性检查和非法重名定义检查。

其工作原理是： 当发生合法重名定义时, 将上层重名表项下推到链中, 而在符号标中原重名表项处登录当前重名定义的符号属性； 在退出本层时, 将最近一次下推的表项, 回推登录到符号表中原重名表项处。

第 4 题：

某 BASIC 语言的变量名字表示为字母开头的字母或数字两个字节的标识符, 该语言的符号表拟采用杂凑法组织，请为其设计实现一个有效散列的杂凑算法，并为解决散列冲突, 设计实现一个再散列算法。

答案：

可采用下列散列杂凑算法（设表长为 N） 散列地址 =MOD （(<第一字节值 >+< 第二字节值 >), N)。

发生冲突时再散列的方法： 在该冲突处开始, 向下寻找第一个空表项, 若找到则该表项地址为再散列地址；若找不到空表项，则循环到表头开始，向下寻找第一个空表项，一直到发生冲突处为止, 若找到空表项则该表项地址为再散列地址, 否则表示符号表已满, 编译系统给出符号表溢出信息，终止编译。

附加题

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

问题 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。

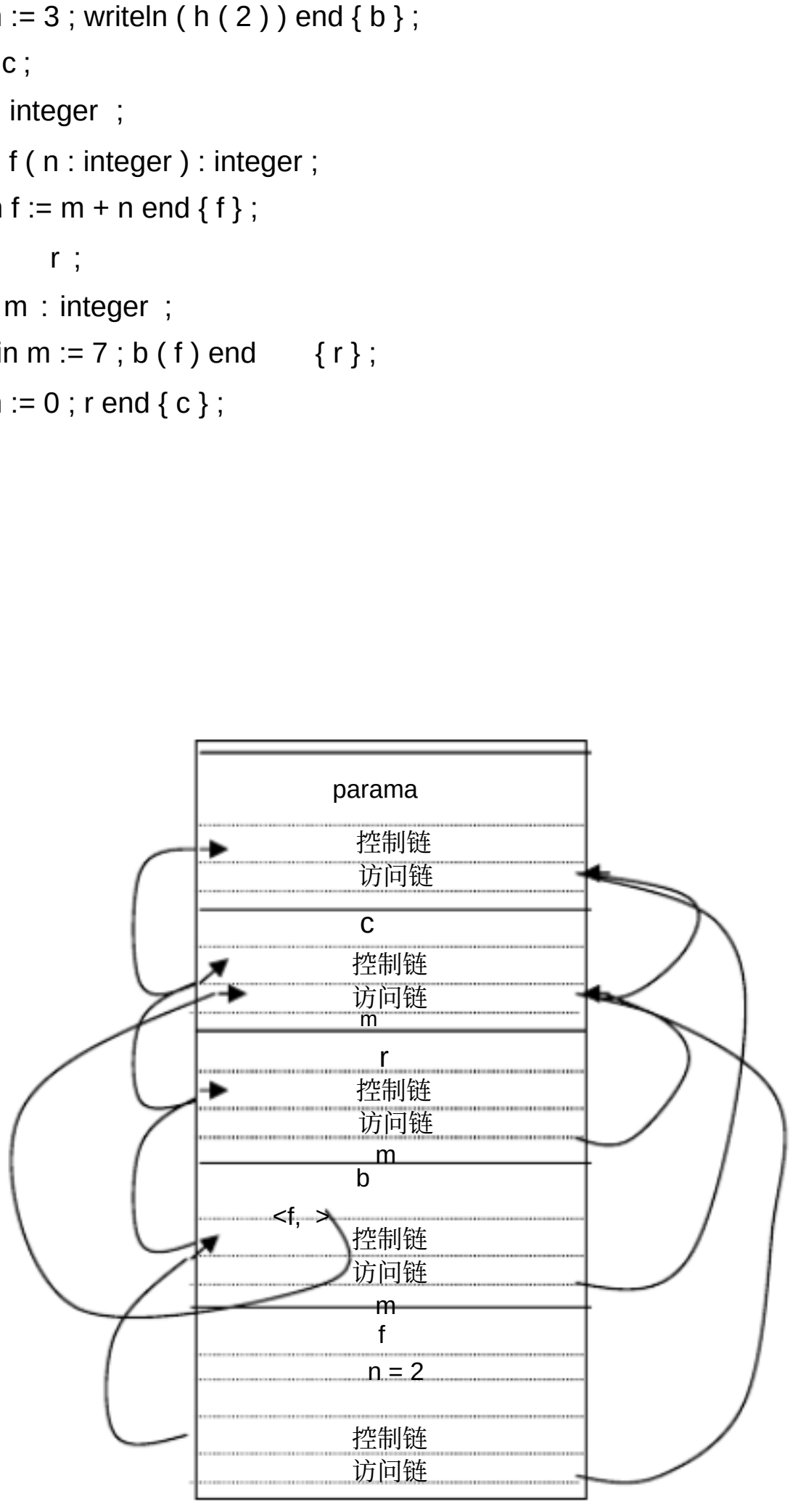
（b）试为此程序画出活动树。

（c）试给出程序的输出。

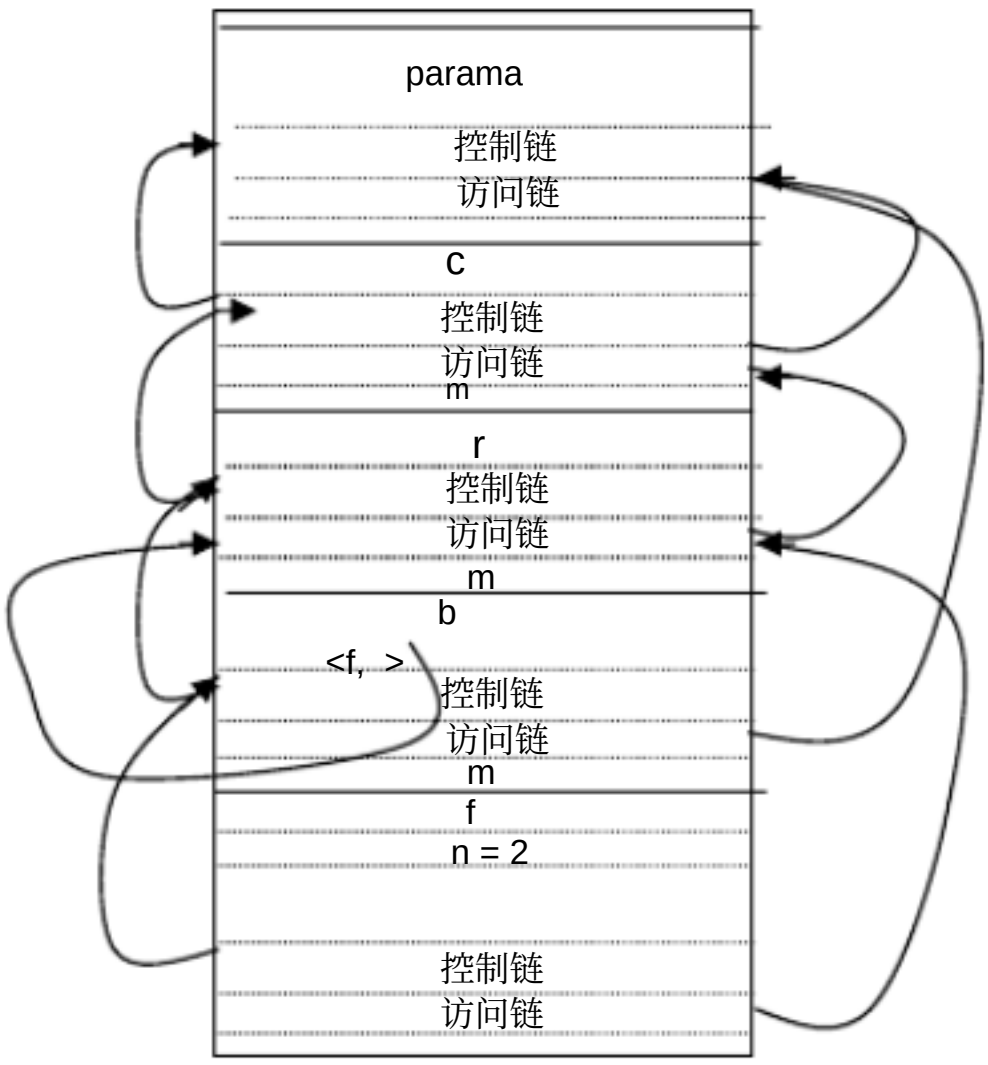
```
(1) program param ( inout, output ) ;
(2)    procedure b ( function h ( n : integer ) : integer ) ;
(3)     var       m : integer  ;
(4)         begin m := 3 ; writeln ( h ( 2 ) ) end { b } ;
(5)  procedure     c ;
(6)     var       m : integer  ;
(7)         function f ( n : integer ) : integer ;
(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：

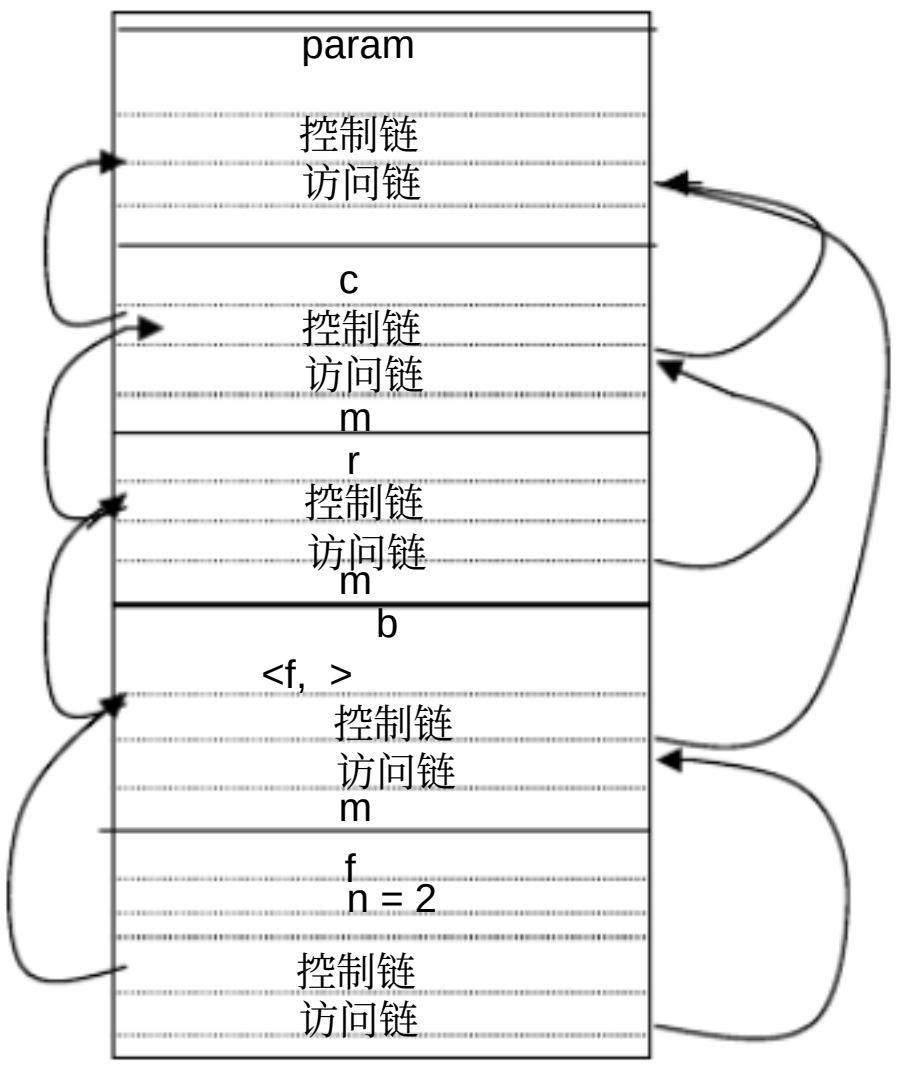
（a）词法环境



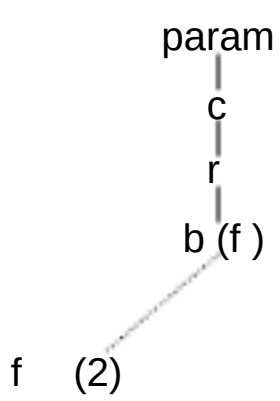
传递环境


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

活动环境


param
控制链
访问链
c
控制链
访问链
m
r
控制链
访问链
m
b
<f, >
控制链
访问链
m
f
n = 2
控制链
访问链

（b）活动树



（c）词法环境： 2；传递环境： 9；活动环境： 5。

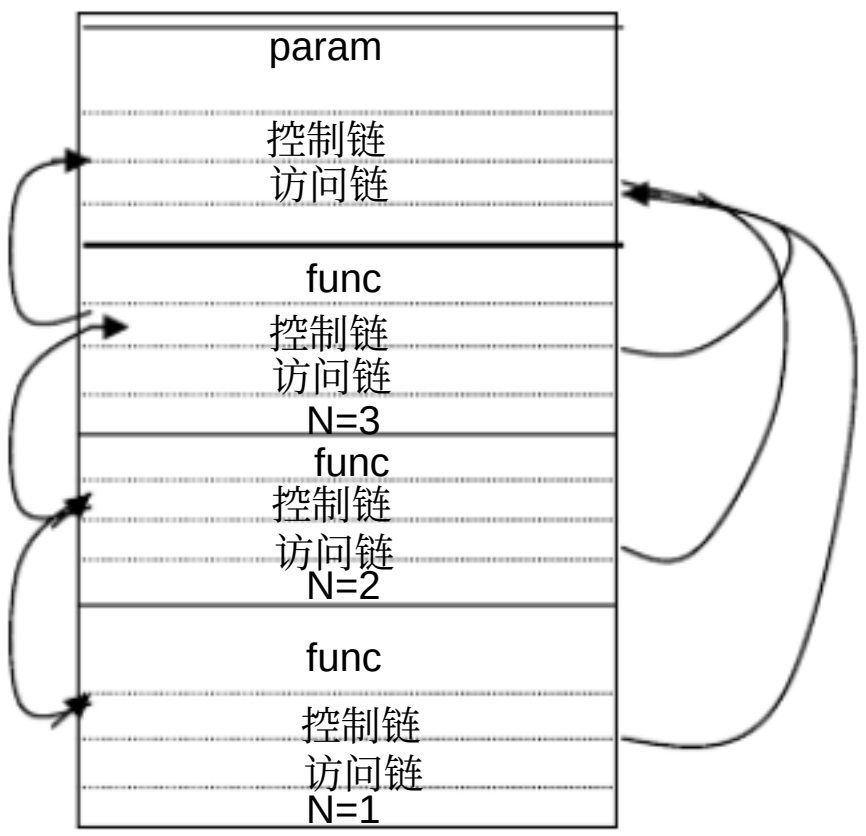
问题 3：

己知程序段：

```
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：

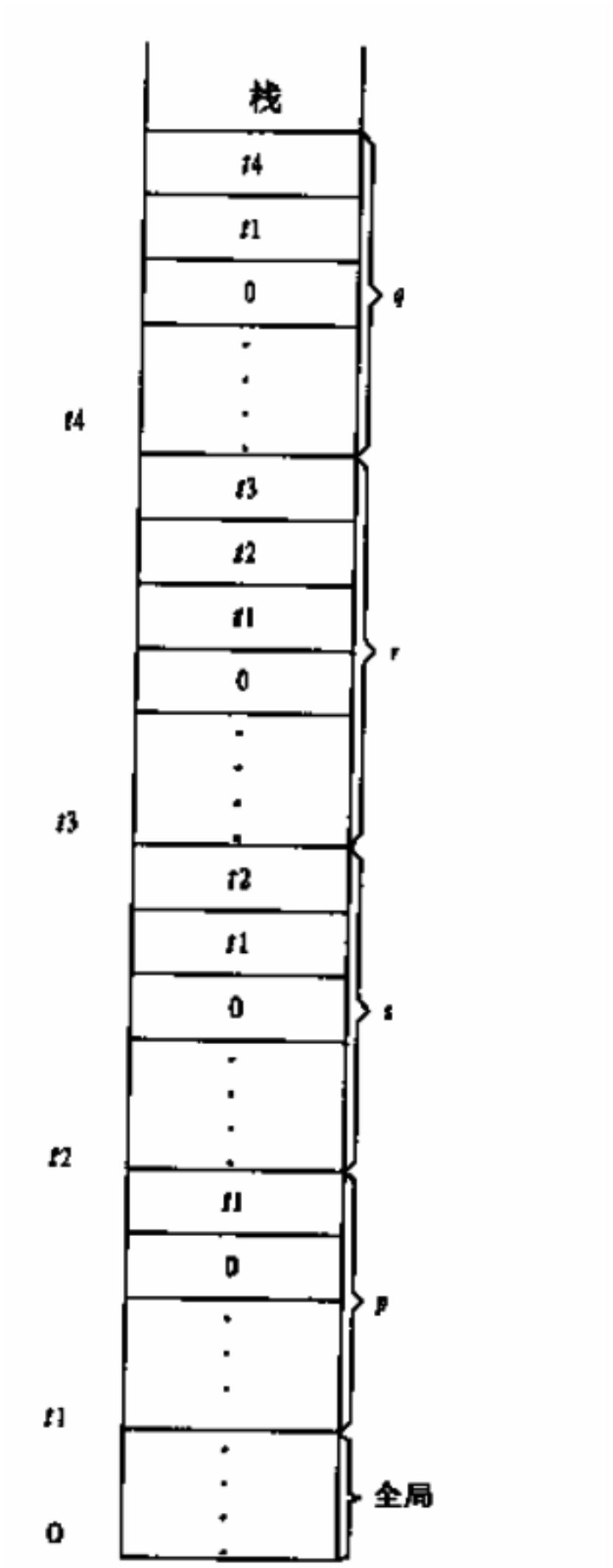


问题 4：

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：



问题 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　。

问题 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。

问题 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。

第 10 章　目标程序运行时的存储组织

第 5 题：

过程参数的传递方式有几种　?简述“传地址”和“传值”的实现原理。

答案：

参数的传递方式有下述几种：
“传值”　-- Call by Value。
“传地址”　-- Call by Address。
“换名”　-- Call by Name。
“得结果”　-- Value-result。

“传值”方式，这是最简单的参数传递方法。即将实参计算出它的值，然后把它传给被调过程。具体来讲是这样的：

1.形式参数当作过程的局部变量处理，　即在被调过程的活动记录中开辟了形参的存储空间，这些存储位置即是我们所说的实参或形式单元。

2.调用过程计算实参的值，并将它们的右值（　r-value）放在为形式单元开辟的空间中。

3.被调用过程执行时，就像使用局部变量一样使用这些形式单元。

“传地址”方式，也称作传地址，或引用调用。调用过程传给被调过程的是指针，指向实参存储位置的指针。

1.如实参是一个名字或是具有左值的表达式，则左值本身传递过去。

2.如实参是一个表达式，比方　a+b 或 2，而没有左值，则表达式先求值，并存入某一位置，然后该位置的地址传递过去。

3.被调过程中对形式参数的任何引用和赋值都通过传递到被调过程的指针被处理成间接访问。

第 6 题：

下面的程序执行时输出的　　a 分别是什么 ?若

(1) 参数的传递办法为“传值”　。

(2) 参数的传递办法为“传地址”　。

```
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) 参数的传递办法为　"传值"时，a 为　2。

(2) 参数的传递办法为　"传地址"，a 为　7。

附加题

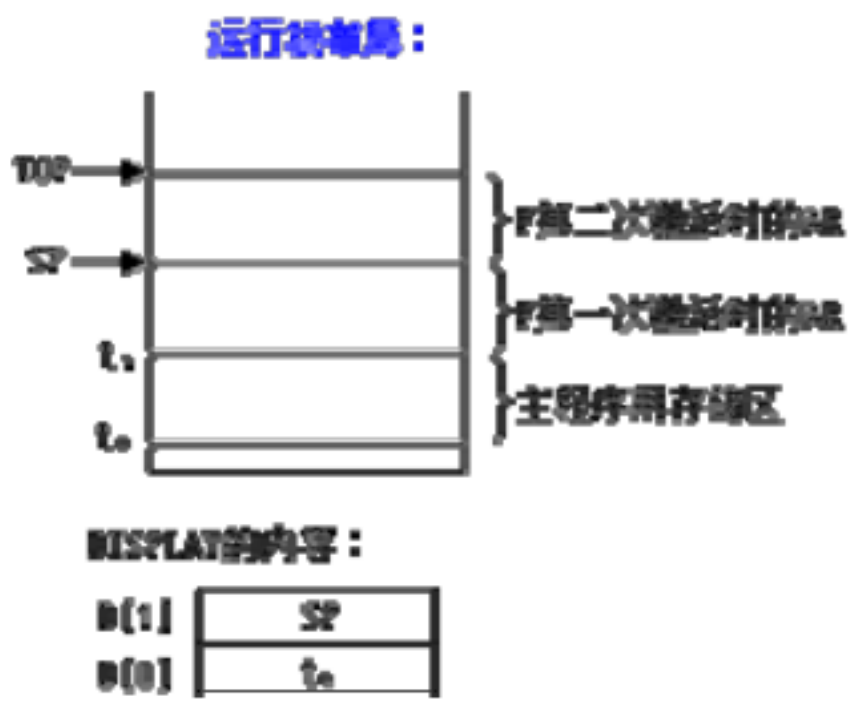
问题 1：

下面是一个 Pascal 程序

```
program PP(input,output)
    var K:integer;
    function F(N:integer):integer
    begin
        if N< =0 then F:=1
        else F:=N * F(N-1);
    end;
begin
    K:=F(10);
    ...
end;
```

当第二次（递归地）进入 F 后， DISPLAY 的内容是什么？当时整个运行栈的内容是什么？

答案：



问题 2：

对如下的 Pascal 程序，画出程序执行到（ 1）和（ 2）点时的运行栈。

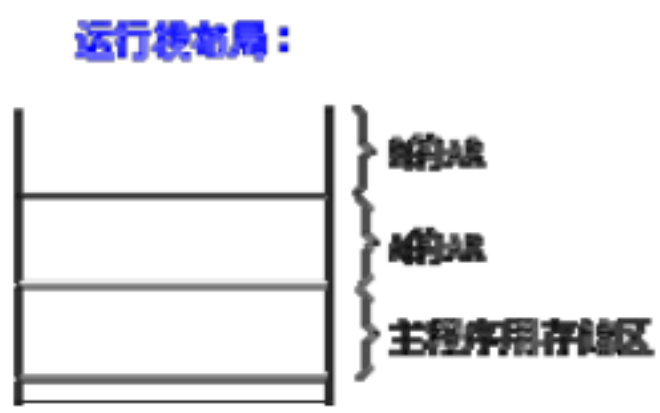
```
program Tr(input,output);
    var i:integer; d:integer;
    procedure A(k:real);
        var p:char;
        procedure B;
            var c:char;
            begin
                ...(1)...
            end; {B}
        procedure C;
            var t:real;
            begin
                ...(2)...
            end;{C}
        begin
            .....
            B;
            C;
            .....
        end;{A}
begin{main}
    ...
    A(d);
    ...
end.
```

答案：

程序执行到（ 1）点时的流程是：

① 主程序激活 A

②A 激活 B



程序执行到（ 2）点时的流程是：

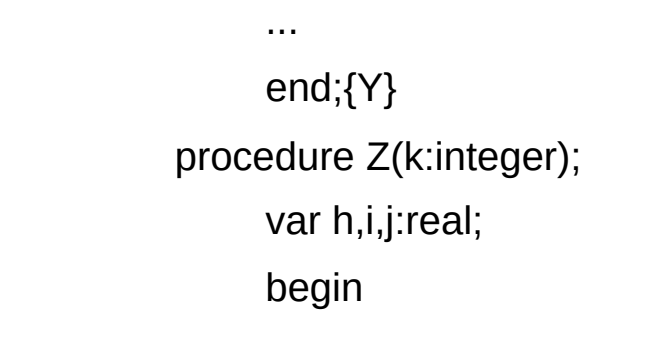
① 主程序激活 A

② A 激活 B

③ B 执行结束返回　A

④ A 激活　C

运行栈布局：



问题 3：

有如下示意的　Pascal 源程序

```
program main;
    var a,b,c:integer;
    procedure X(i,j:integer);
        var d,e:real;
        procedure Y;
                var f,g:real;
                begin
                ...
                end;{Y}
        procedure Z(k:integer);
                var h,i,j:real;
                begin
                ...
                end;{Z}
        begin
                .....
                10:Y;
                .....
                11:Z;
                .....
            end;{X}
        begin
            .....
            X(a,b);
            .....
        end.{main}
```

并已知在运行时刻，以过程为单位对程序中的变量进行动态存储分配。当运行主程序而调用过程语句　X 时，试分别给出以下时刻的运行栈的内容和　　DISPLAY　的内容。

（1）已开始而尚未执行完毕的标号为　　10 的语句。

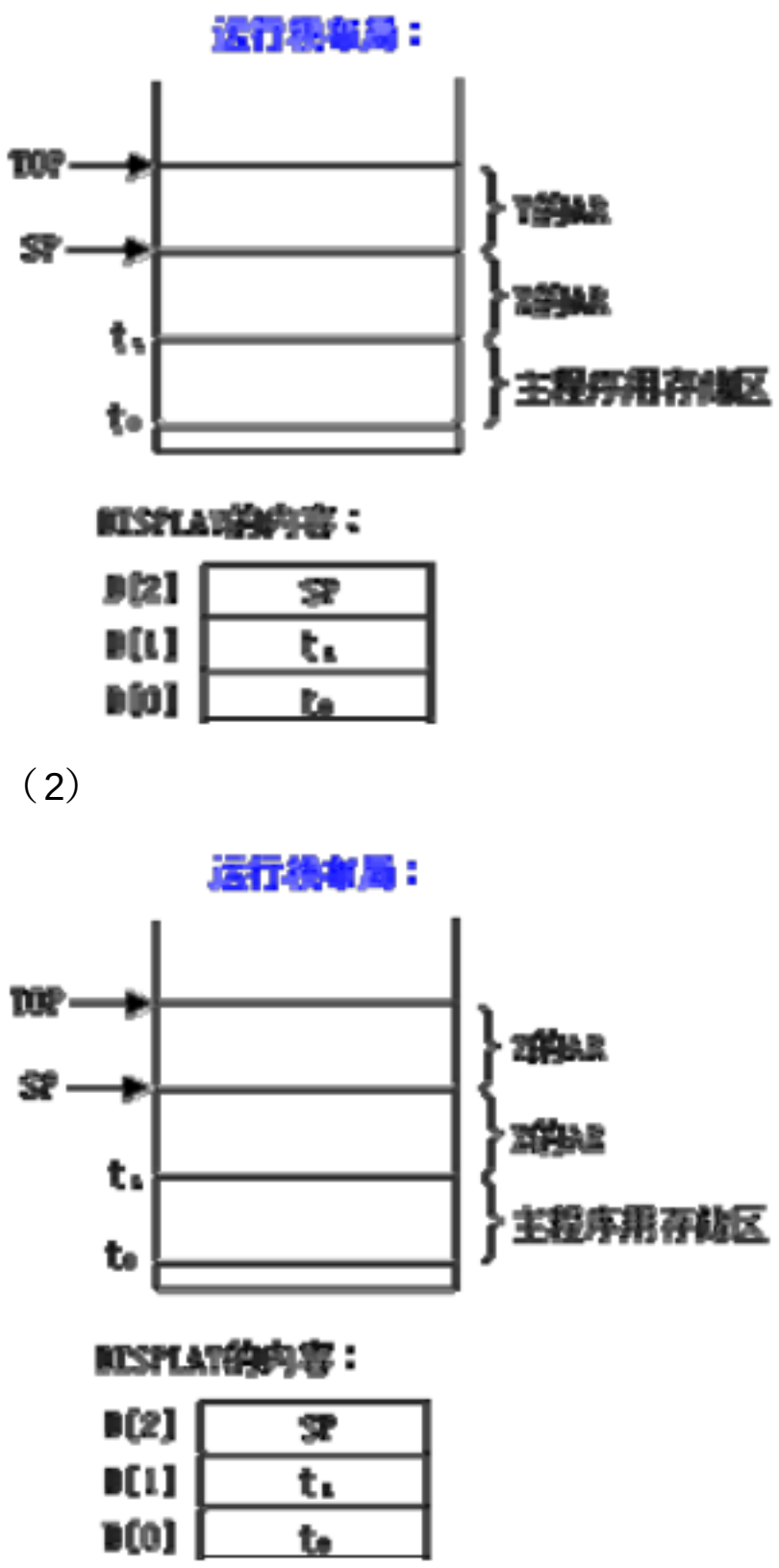
（2）已开始而尚未执行完毕的标号为　　11 的语句。

答案：

程序结构：


主
Y
Z

（1）


运行栈布局：
TOP
SP
t_1
t_0
Y的AR
Z的AR
主程序用存储区
DISPLAY的内容：
D[2]
D[1]
D[0]
SP
t_1
t_0

（2）

运行栈布局：
TOP
SP
t_1
t_0
Z的AR
Z的AR
主程序用存储区
DISPLAY的内容：
D[2]
D[1]
D[0]
SP
t_1
t_0

第 11 章　代码优化

第 1 题

何谓代码优化？进行优化所需要的基础是什么？

答案：

对代码进行等价变换，　使得变换后的代码运行结果与变换前代码运行结果相同，　而运行速度加快或占用存储空间减少，或两者都有。

优化所需要的基础是在中间代码生成之后或目标代码生成之后。

第 2 题

编译过程中可进行的优化如何分类？

答案：

依据优化所涉及的程序范围，可以分为：局部优化、循环优化和全局优化。

第 3 题

最常用的代码优化技术有哪些？

答案：

1. 删除多余运算
2. 代码外提
3. 强度削弱
4. 变换循环控制条件
5. 合并已知量与复写传播
6. 删除无用赋值

第 4 题

图 11.23 是图 11.22 的 C 代码的部分三地址代码序列。

```
void quicksort(m,n)
int m,n;
{ int i,j;
    int v,x; if (n<=m) return;
    /* fragment begins here */
    i = m-1;
    j = n;
    v = a[n];
    while(1) {
        do i = i+1;while (a[i]<v);
        do j = j-1; while (a[j]>v);
        if (i>=j) break;
            x = a[i];
            a[i] = a[j];
            a[j] = x;
    }
    x = a[i];
    a[i] = a[n];
    a[n] = x;
    /* fragment ends here */
    quicksort (m,j);
    quicksort(i+1,n);
}
```

图 11.22

```
(1)     i:=m-1
(2)     j:=n
(3)     t1:=4*n
(4)     v:=a[t1]
(5)     i:=i+1
(6)     t2:=4*i
(7)     t3:=a[t2]
(8)     if t3< v goto (5)
(9)     j:=j-1
(10)    t5:=4*j
(11)    t5:=a[t4]
(12)    if t5> v goto (9)
(13)    if i >= j goto (23)
```

(14)　t6:=4*i

(15)　x:=a[t6] (16) t7:=4*i

(17) t6:=4*j

(18) t9:=a[t8]

(19) a[t7]:=t9

(20) t10:=4*j

(21) a[t10]:=x

(22) goto (5)

(23) t11:=4*i

(24) x:=a[t11]

(25) t12:=4*i

(26) t13:=4*n

(27) t14:=a[t13]

(28) a[t12]:=t14

(29) t15:=4*n

(30) a[t15]:=x

图 11.23

(1) 请将图 11.23 的三地址代码序列划分为基本块并做出其流图。

(2) 将每个基本块的公共子表达式删除。

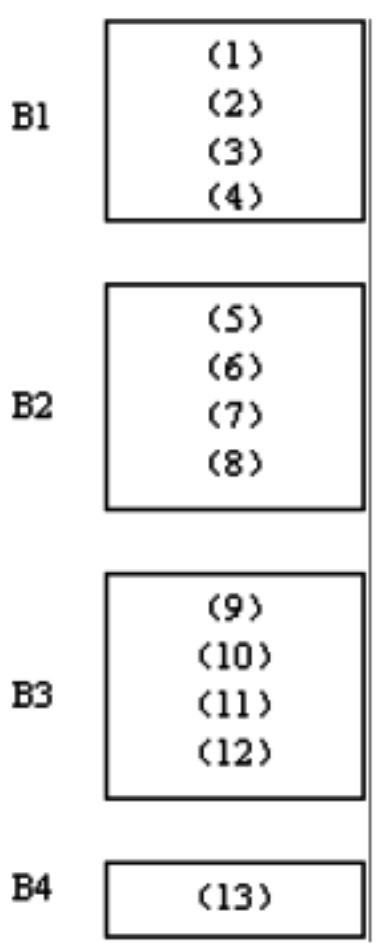
(3) 找出流图中的循环 ,将循环不变量计算移出循环外。

(4) 找出每个循环中的归纳变量，并在可能的地方删除它们。

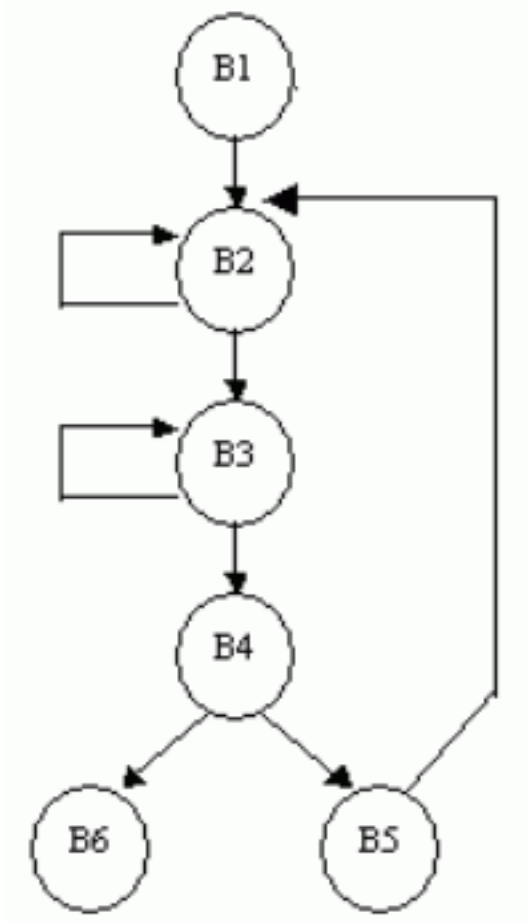
答案：

（1）

基本块



流图



（2）

B5 中（14）和（ 16）是公共子表达式、 （17）和（ 20）是公共子表达式, B5 变为

（14） t_6:=4*I

（15）

（16） t_7:=t_6

（17） t_8:=4*J

…

（20） t_{10}:=t_8

（21）

（22）

B6 中（23）和（ 25）是公共子表达式、 （26）和（ 29）是公共子表达式, B6 变为

（23） t_{11}:=4*I

（24）

（25） t_{12}:=t_{11}

（26） t_{13}:=4*n

…

（29） $t_{15}:=t_{13}$

（3）

循环

① {B2}

② {B3}

③ {B2 ， B3， B4， B5}

(4)

在循环 {B2 ， B3， B4， B5} 中，原来的 (14)(17) 都可以删除。

第 5 题：

如下程序流图 (图 11.24)中， B3 中的 i：=2 是循环不变量，可以将其提到前置结点吗 ？你还能举出一些例子说明循环不变量外移的条件吗 ？

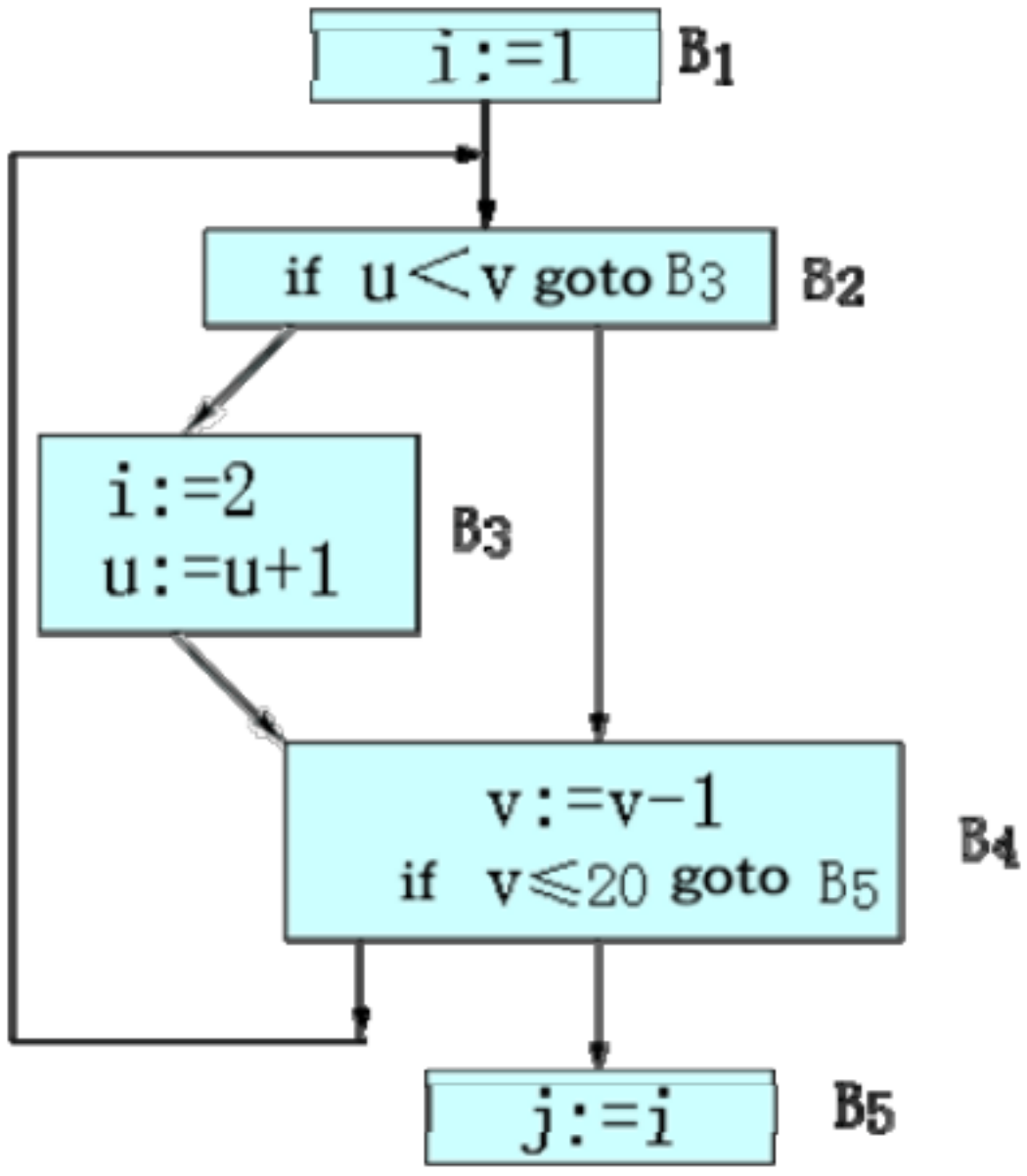


图 11.24

答案：

不能。因为 B3 不是循环出口 B4 的必经结点。

循环不变量外移的条件外有：

（a）(I)s 所在的结点是 L 的所有出口结点的必经结点

(II)A 在 L 中其他地方未再定值

(III)L 中所有 A 的引用点只有 s 中 A 的定值才能到达

（b）A 在离开 L 之后不再是活跃的，并且条件 (a)的(II) 和(III) 成立。所谓 A 在离开 L 后不再是活跃的是指， A 在 L 的任何出口结点的后继结点的入口处不是活跃的（从此点后不被引用）（3）按步骤（ 1）所找出的不变运算的顺序，依次把符合（ 2）的条件 (a)或(b)的不变运算 s 外提到 L 的前置结点中。如果 s 的运算对象（ B 或 C）是在 L 中定值的，则只有当这些定值四元式都已外提到前置结点中时，才可把 s 也外提到前置结点。

第 6 题

试对以下基本块　B1 和 B2：

B1：

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

B2：

B：=3

D：=A+C

E：=A*C

F：=D+E

G：=B*F

H：=A+C

I：=A*C

J：=H+I

K：=B*5

L：=K+J

M：=L

分别应用　DAG 对它们进行优化，并就以下两种情况分别写出优化后的四元式序列：

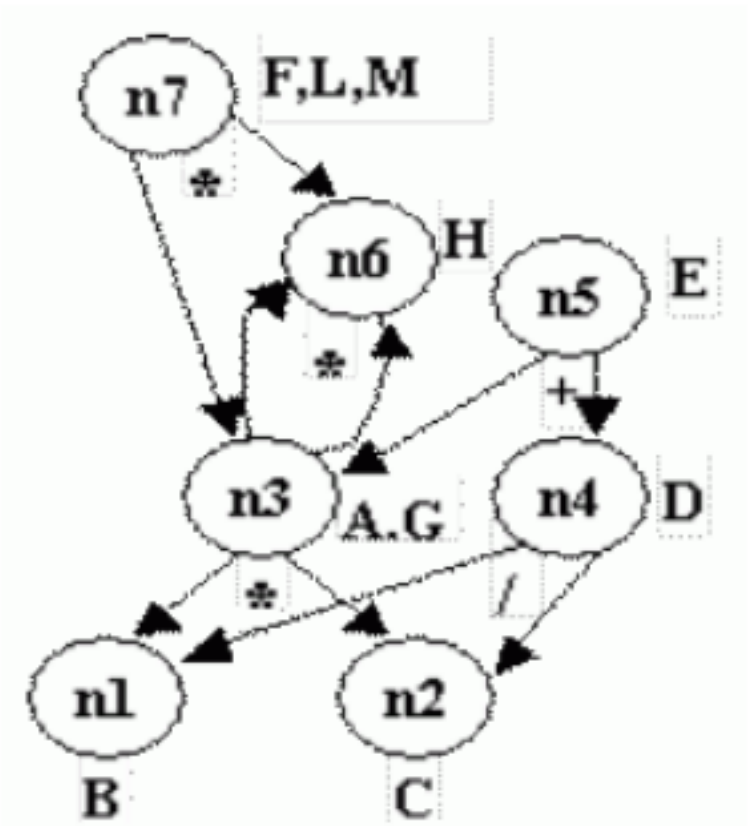
（1）假设只有　G、L、M 在基本块后面还要被引用。

（2）假设只有　L 在基本块后面还要被引用。

答案：

B1：

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

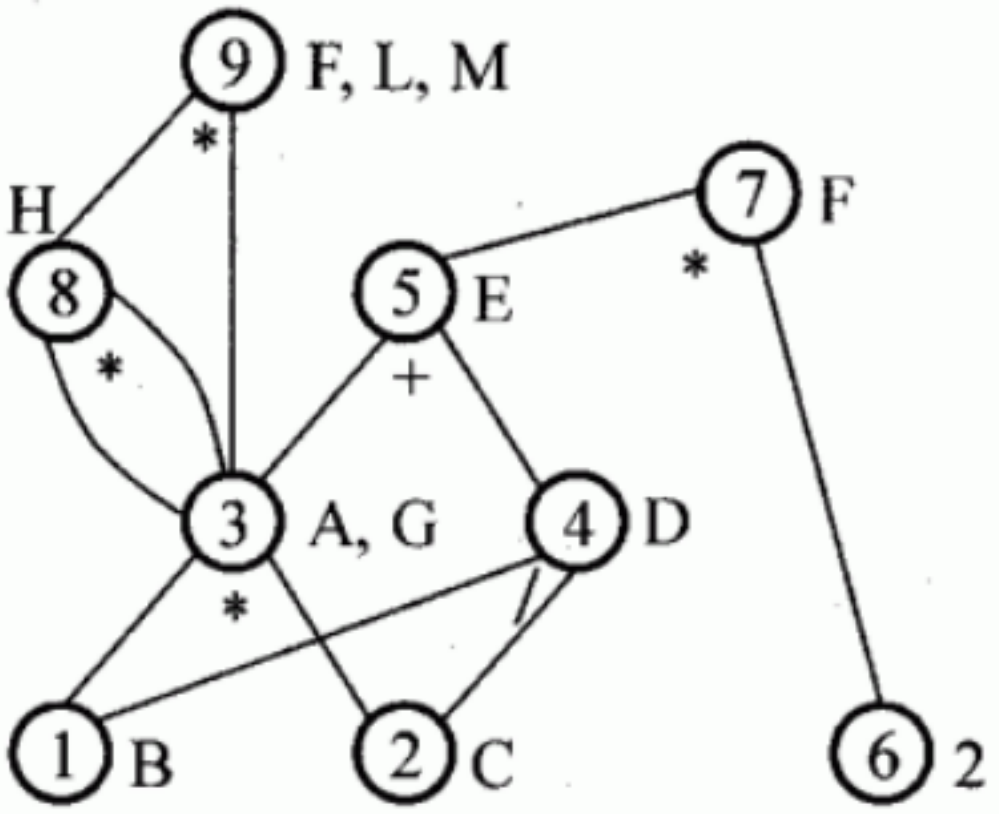
（备注： S1, S2, S3 为新引入的临时变量）

或者：

优化后的四元式序列：

对假设(1)有

G := B * C
H := G * G
L := H * G
M := L

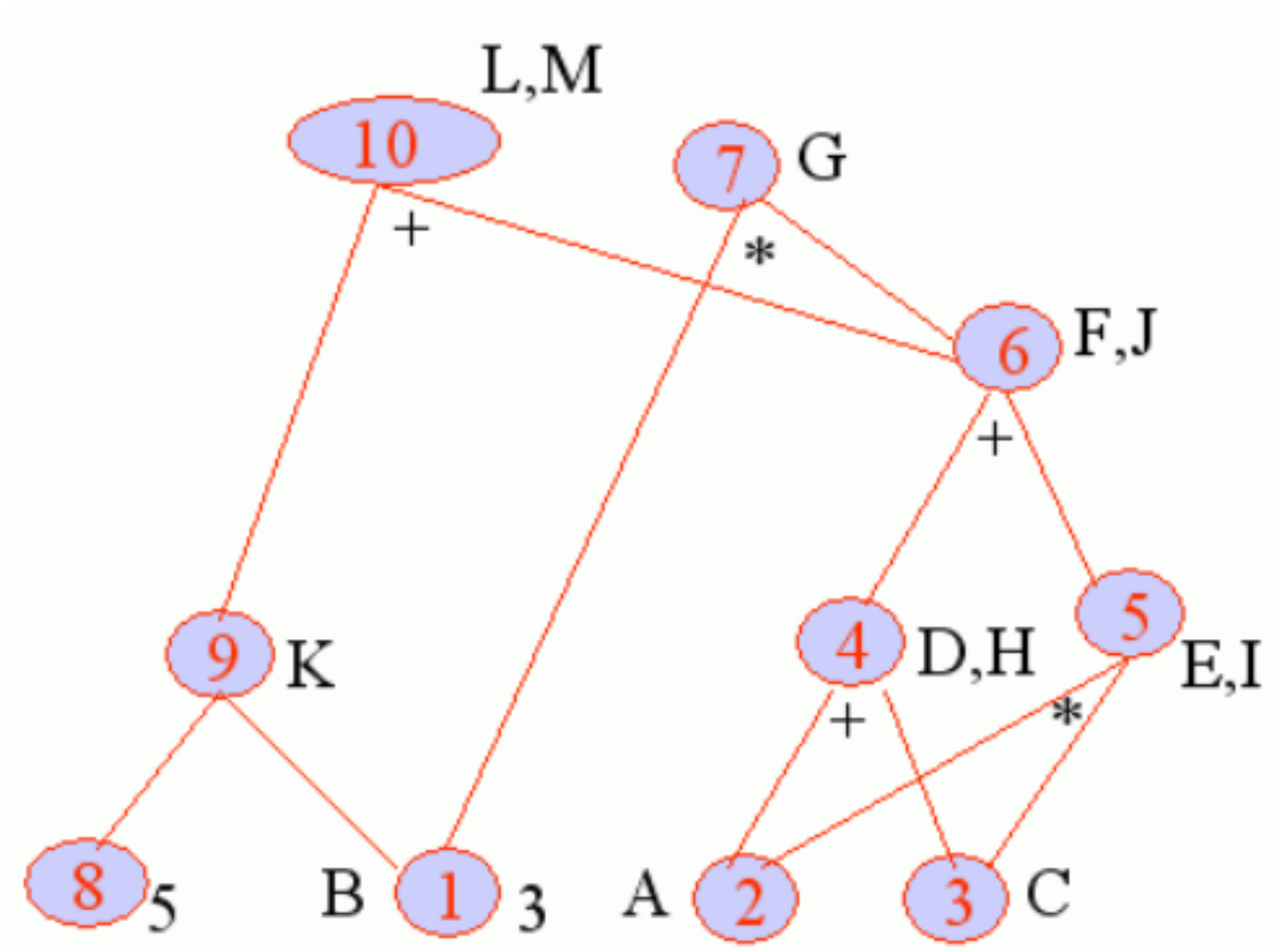


对假设(2)有

G := B * C
H := G * G
L := H * G

B2:

基本块对应的　DAG　如下：



优化后的四元式序列：

对假设（1）

B:=3

D:=A+C

E:=A*C

F:=D+E

G:=B*F

K:=B*5

L:=K+F

M:=L

对假设（ 2）

B:=3

D:=A+C

E:=A*C

F:=D+E

K:=B*5

L:=K+F

第 7 题

分别对图 11.25 和 11.26 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。

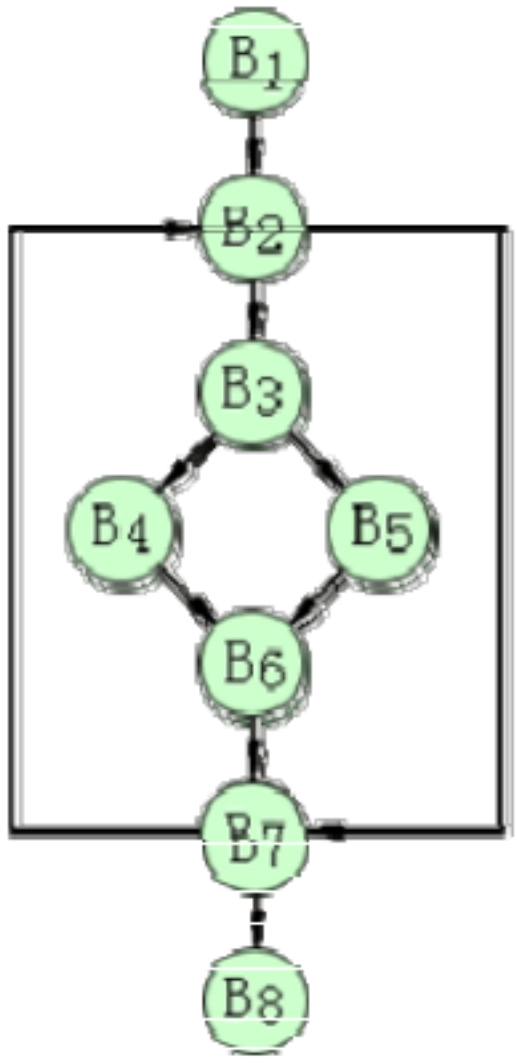


图 11.25

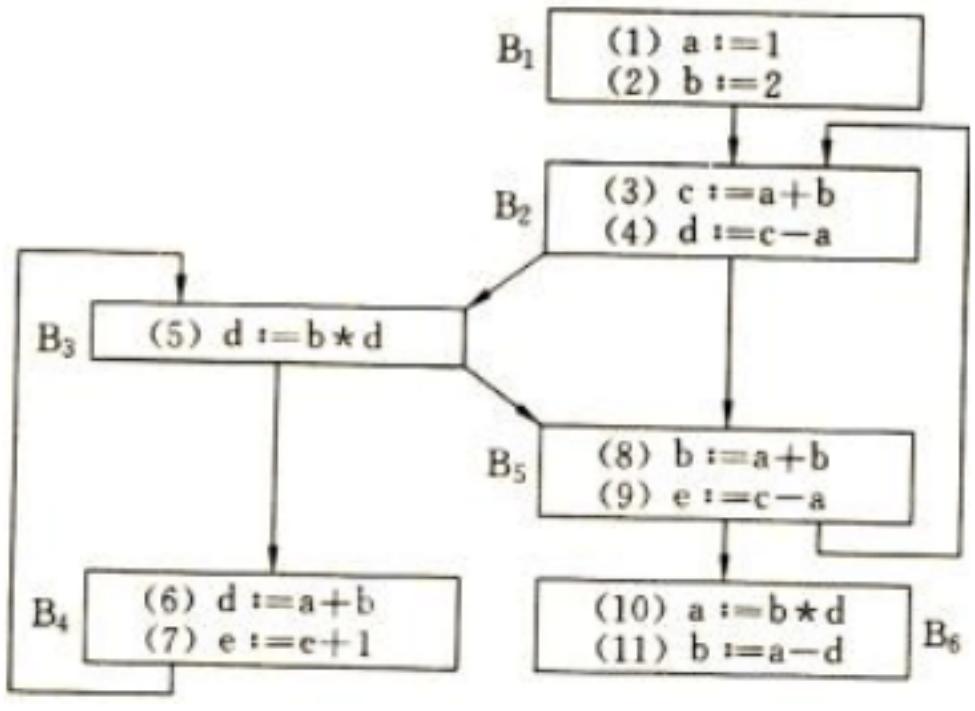


图 11.26

答案：

对图 11.25：

（1）流图中各结点 N 的必经结点集 D(n)：

D(1)={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)= {1,2,7}

D(8)= {1,2,7,8}

（2）回边： 7→2

（3）循环： {2 ， 3， 4， 5， 6， 7}

对图 11.26：

(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}

附加题

问题 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：

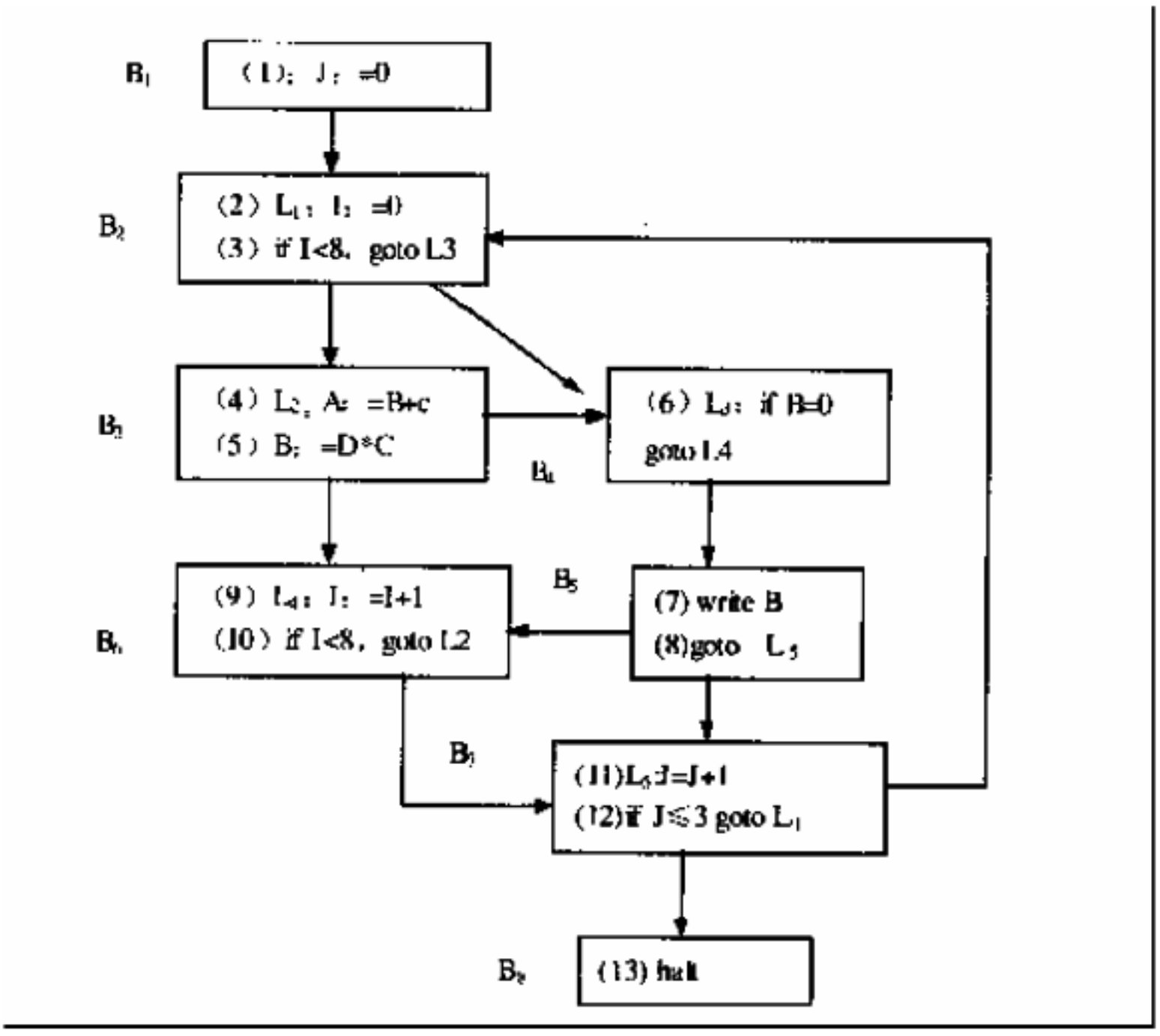
(1)它们是程序的第一个语句；或,

(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。

问题 2：

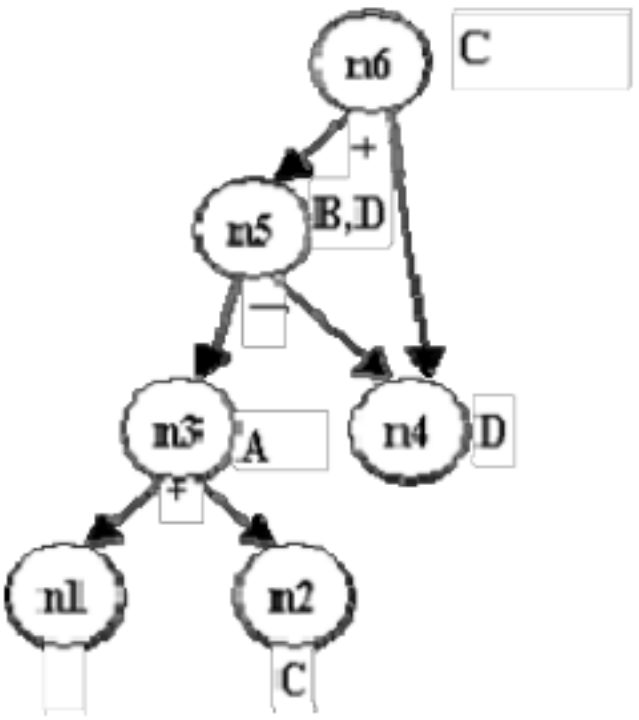
基本块的 DAG 如下图所示，若 (1)B 在该基本块出口处不活跃 ,(2)B 在该基本块出口处活跃的，请分别给出以下代码经过优化后的代码。

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

第 12 章　代码生成

第 1 题

一个编译程序的代码生成要着重考虑哪些问题　　？

答案：

代码生成器的设计要着重考虑目标代码的质量问题，　　而衡量目标代码的质量主要从占用空间和执行效率两个方面综合考虑。

附加题

问题 1：

决定目标代码的因素有哪些？

答案：

决定目标代码的因素主要取决于具体的机器结构、 指令格式、 字长及寄存器的个数和种类，并与指令的语义和所用操作系统、 存储管理等都密切相关。 又由于目标代码的执行效率在很大程度上依赖于寄存器的使用，所以目标代码与寄存器的分配算法也有关。

问题 2：

为什么在代码生成时要考虑充分利用寄存器？

答案：

因为当变量值存在寄存器时， 引用的变量值可直接从寄存器中取， 减少对内存的存取次数, 这样便可提高运行速度。 因此如何充分利用寄存器是提高目标代码运行效率的重要途径。

问题 3：

寄存器分配的原则是什么？

答案：

寄存器分配的原则是：

(1) 当生成某变量的目标代码时，尽量让变量的值或计算结果保留在寄存器中，直到寄存器不够分配时为止。

(2) 当到基本块出口时，将变量的值存放在内存中，因为一个基本块可能有多个后继结点或多个前驱结点，同一个变量名在不同前驱结点的基本块内出口前存放的 R 可能不同, 或没有定值， 所以应在出口前把寄存器的内容放在内存中， 这样从基本块外入口的变量值都在内存中。

(3) 对于在一个基本块内后边不再被引用的变量所占用的寄存器应尽早释放，以提高寄存器的利用效率。 对基本块的划分可按基本块的划分算法 (见 11.2.1)在生成四元式的目标代码时进行，以区分基本块的入口和出口。

第 13 章　编译程序的构造

第 1 题

构造一个编译程序有哪些途径？

答案：

编译程序的实现途径可有：

（1）手工构造：用机器语言、汇编语言或高级程序设计语言书写。

（2）自动构造工具：　Lex,Yacc。 Lex ,Yacc 分别是词法和语法分析器的生成器。

（3）移植方式：目标程序用中间语言。

（4）自展方式：用　T 型图表示。

附加题

问题 1:

如何用 T 型图 表示一个编译程序的实现？

答案：

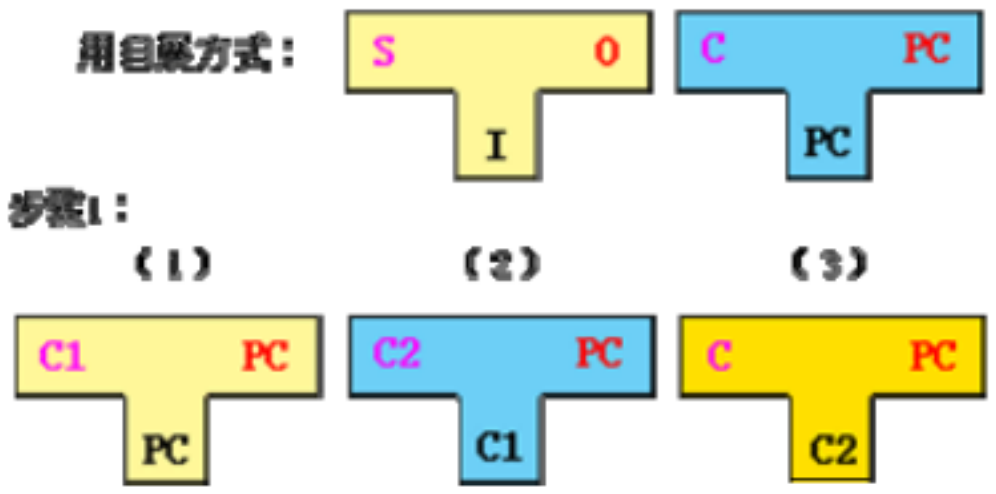
用 T 型图 表示编译程序的实现

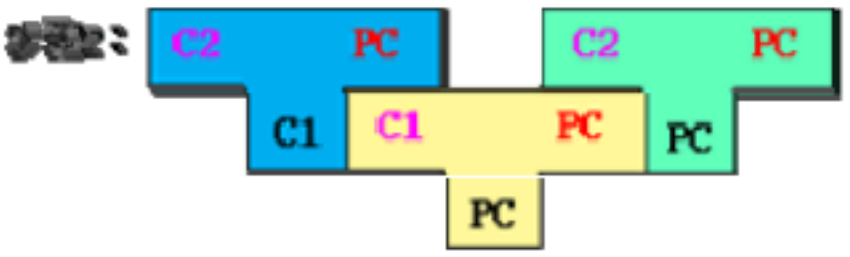
问题 2：

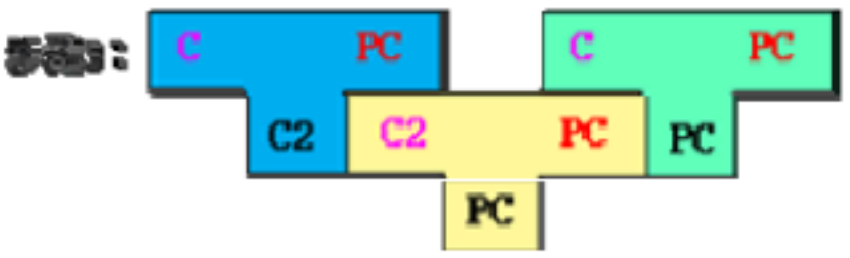
如何用自展方式在 PC 机上实现 C 语言的编译程序？请用 T 型图 表示。

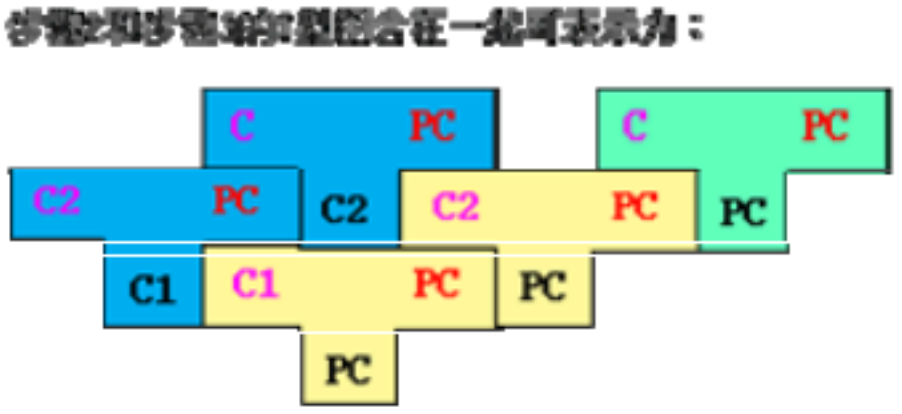
答案：

用自展方式在 PC 机上实现 C 语言的编译程序，首先把 C 划分成真包含的子集 C1 和 C2，然后分 3 步实现。









问题 3：

什么叫做软件移植？

答案：

通常把某个机器 (称为宿主机)上已有的软件移植到另一台机器 (称为目标机)

问题 4：

什么叫做交叉编译？

答案：

交叉编译是指把一个源语言在宿主机上经过编译产生目标机的汇编语言或机器语言。

问题 5：

编译程序的实现应考虑的问题有那些？

答案：

编译程序的实现 应考虑：开发周期、目标程序的效率、可移植性、可调试性、可维护性、可扩充性等。