Charpter 1

3. 什么是解释程序？它与编译程序的主要不同是什么？

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。

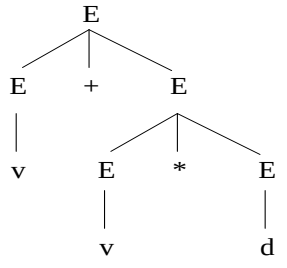
广义上讲，编译程序和解释程序都属于翻译程序，但它们的翻译方式不同，解释程序是边翻译（解释）边执行，不产生目标代码，输出源程序的运行结果。而编译程序只负责把源程序翻译成目标程序，输出与源程序等价的目标程序，而目标程序的执行任务由操作系统来完成，即只翻译不执行。

Charpter 2

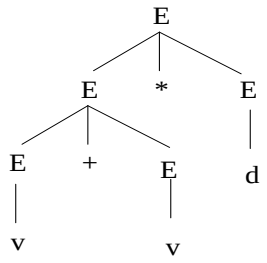
1. 文法 G=（{A,B,S}, {a,b,c}, P,S）
其中 P 为： S → Ac | aB
A→ab
A→bc
写出 L(G[S])的全部元素。
答案：L(G[S])={abc}

2. 文法 G[N]为
N →D | ND
D →0 |1|2|3|4|5|6|7|8|9
G[N]的语言是什么？
答案：
因为：N=>ND=>NDD=>.....=>NDDD....D=>D.....D 所以
G[N]的语言是允许 0 开头的非负十进制整数。
或其语言是 V+， 这里 V={0，1，2，3，4，5，6，7，8，9}

4. 证明文法 G=（{E，O}, {(,),+,*,v,d}, P,E}是二义性的，其中 P 为
E → EOE | (E) | v | d
O → + | *
答案：
该题简单的证明方法就是找出一个句型。画出两种不同形态的语法推导树。
例如句型： v+v*d, 其语法推导树为：

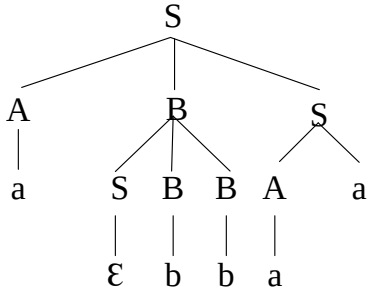


或



所以文法是二义性的。

11. 一个上下文无关文法申城句子 abbaa 的唯一语法树如下：



（1） 给出该句子相应的最左推导和最右推导
（2） 该文法的产生式集合 P 可能有哪些元素？
（3） 找出改句子的所有短语、简短短语、句柄
答案：
（1）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） 产生式集合可能包含： S→ ABS　S→Aa　s　S→ε
A→a　B→SBB
（3） a 是相对于 A 的短语
ε 是相对于 S 的短语
b 是相对于 B 的短语
bb 是相对于 B 的短语
aa 是相对于 S 的短语
abbaa 是相对于 S 的短语
直接短语有 a　b　ε
句柄是 a

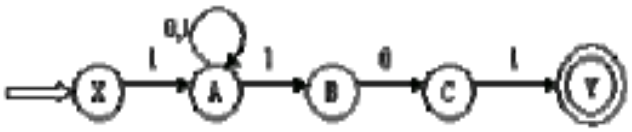
12
（2）、 S → aS | aSb | ε
（5）、 S → aSbb | aSbbb | ε
（6）、 S→ aSa | bSb | ε

18
（2）、 S → aA
A → aA | bB
B → bB | ε
或 S → aS | aB
B → bB | b

Charpter 3

1. 构造下列正规式相应的 DFA
（1）1（0|1）*101
答案：

(1) 先构造 NFA：



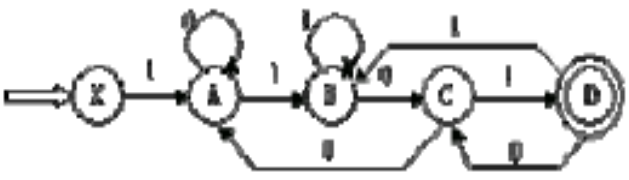
用子集法将 NFA 确定化

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

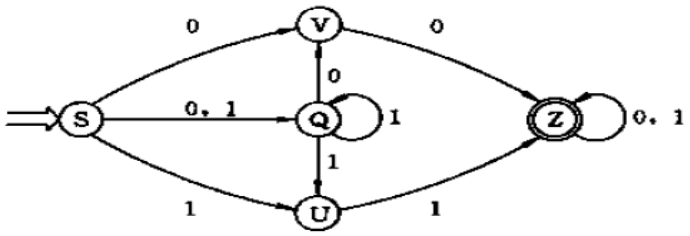
除 X，A 外，重新命名其他状态，令 AB 为 B、AC 为 C、ABY 为 D，因为 D 含有 Y（NFA 的终态），所以 D 为终态。

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

DFA 的状态图：：



3 将下图 NFA 确定化



答案：

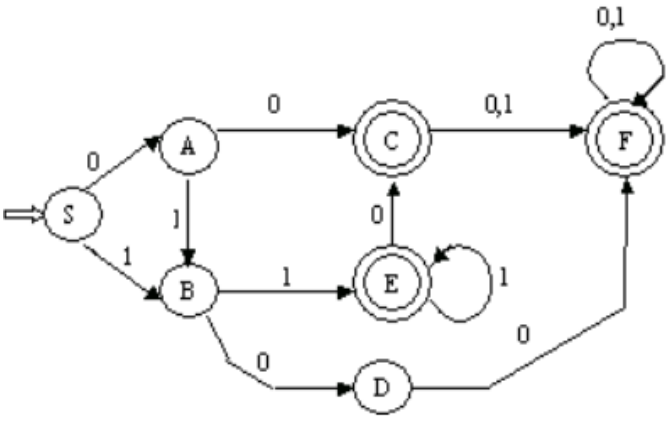
用子集法将 NFA 确定化：

.	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：



7．给文法 G[S]：

S→aA|bQ

A→aA|bB|b

B→bD|aQ

Q→aQ|bD|b

D→bB|aA

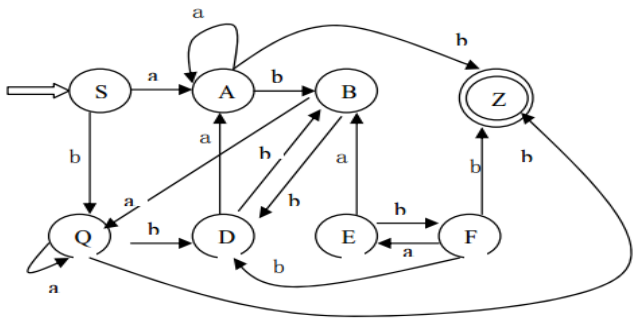
E→aB|bF

F→bD|aE|b

构造相应的最小的 DFA。

答案：

先构造其 NFA：



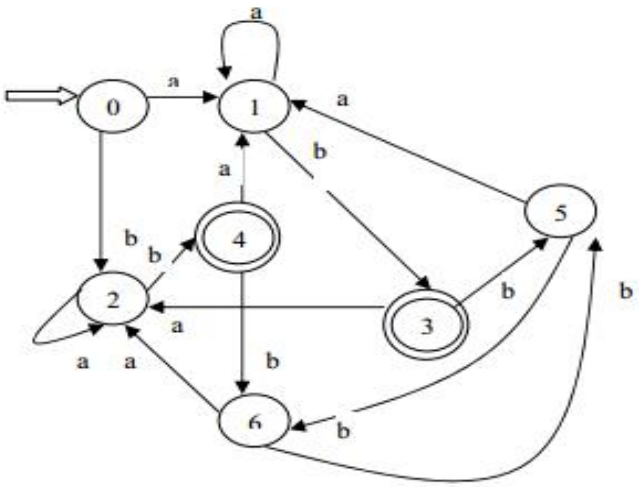
用子集法将 NFA 确定化：

	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：



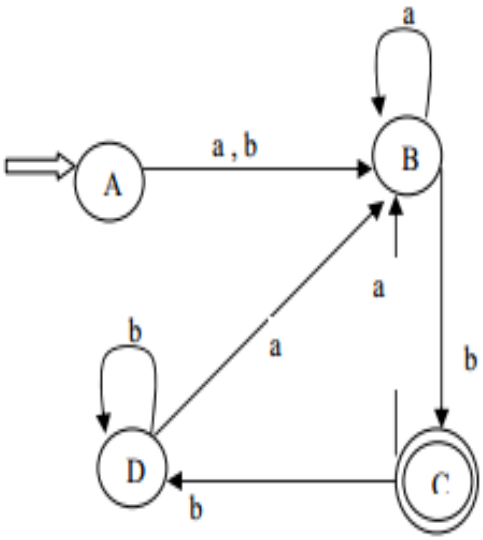
令P_0=（{0,1,2,5,6}，{3,4}）用b进行分割：

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：



11.

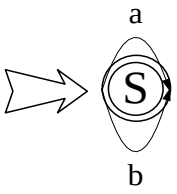
12. 有一种用以证明两个正则表达式等价的方法，那就是构造他们的最小 DFA，表明这两个 DFA 是一样的(除了状态名不同外)。使用此方法。证明下面的正则表达式是等价的。

(1) (a|b)*

(2) (a*|b*)*

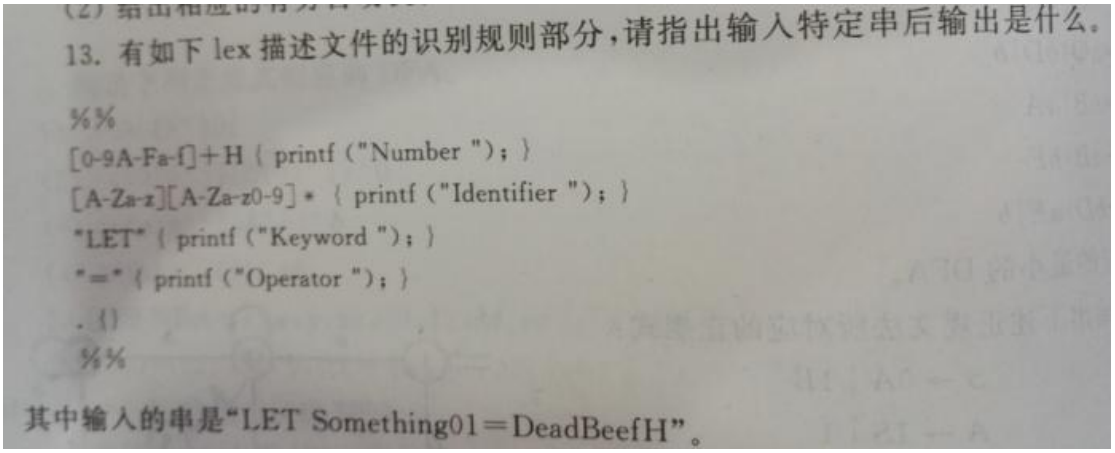
解答：先求出 DFA:

由正规式（1），很容易看出其 DFA 如下，当然也是最小的



由正规式（2），先构造 NFA，再确定化得 DFA 如下，最后最小化得到和上面一样的 DFA，（过程省略），即完成证明。

13.


13. 有如下 lex 描述文件的识别规则部分，请指出输入特定串后输出是什么。

```
%%
[0-9A-Fa-f]+H { printf ("Number "); }
[A-Za-z][A-Za-z0-9]* { printf ("Identifier "); }
"LET" { printf ("Keyword "); }
"=" { printf ("Operator "); }
. {}
%%
```

其中输入的串是“LET Something01＝DeadBeefH”。

输出：Identifier Indentifier Operator Number

规则存在先后顺序，所以 LET 字符串匹配的是[A-Za-a][A-Za-z0-9]*

Charpter 4

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。

(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。

4. 证明下述文法不是 LL（1）的

S→C$

C→bA | aB

A→a | aC | bAA

B→b | bC | aBB

是否能构造一等价文法，使其是 LL（1）的？并给出判断过程。

答案：

由于 A 和 B 的规则中都包含公共左因子，所以：

a 属于 select(A→a)并且 a select(A→aC)，A 的这两条规则的 select 集合相交不为空，故文法不是 LL（1）的

通过提取公共左因子来改写：

S→C$

C→ bA | aB

A→ aC’| bAA

B→ bC’| aBB

C’→C| ε

计算：first(S)={a,b}　first(C)={a,b}　first(A)={a,b }

first(B)={ a,b }　first(C’)={ a,b, ε }

follow(S)={# }　follow (C)={$}　follow (A)={a,b,$}

follow (B)={a,b,$}　follow (C’)={a,b,$}

这样按定义求 select 集合

select(S→C$)={a,b}

select(C→ bA)={b}　select(C→aB)={a}

select(A→ aC’)={a}　select(A→ bAA)={b}

select(B→ bC’)={b}　select(B→aBB)={a}

select(C’→ C)={a,b}　select(C’→ ε)={a,b,$}

非终结符 C’的各规则的 select 集合两两相交为空，不是 LL（1）文法。

7. 对于一个文法若消除了左递归，提取了提取公共左因子后，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。

（4）S→AS | b
A→SA | a

答案：对于一个文法若消除了左递归，提取了提取公共左因子后，不能保证一定为 LL（1）文法，需要验证。

这个文法是间接左递归，所以要消除直接和间接左递归。

首先给一个非终结符的排列： S A

对于第一个非终结符 S 没有直接左递归，不需要处理

对于第二个非终结符 A，将前面的非终结符的规则带入，得到：

A→ASA | bA | a 出现了直接左递归，需要消除，最后得到的文法：

S→AS | b
A→ bAA' | aA'
A'→SAA' | ε

计算得到：

first(S)={a,b} first(A)={a,b} first(A')={a,b, ε}
follow(S)={a,b, #} follow (A)={a,b} follow (A')={a,b, ε}

再依据 first 和 follow 计算 select 集合，首先对 S 的规则，

select（S→AS）={a,b}
select（b）={b} 其交集不为空，所以不是 LL(1)文法。

对其它规则的 select 集合就没必要再计算了。

Charpter 5

1. 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!

2. 对题 1 的 G[S]

(1) 给出(a,(a,a))和(a,a)的最右推导,和规范归约过程。

(2) 将(1)和题 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。

Charpter 6

1. 已知文法

A→aAd　|aAb| ε

判断该文法是否是 SLR(1)文法，如是，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：

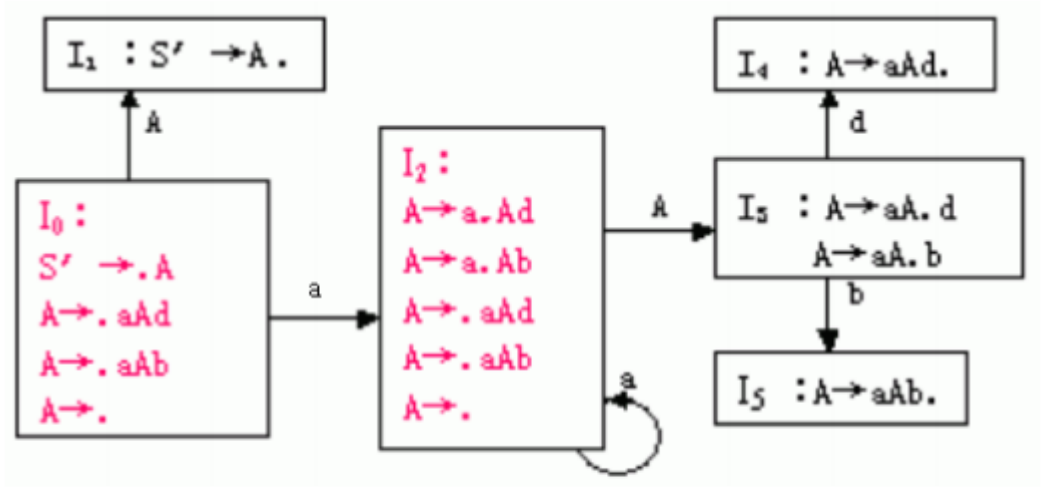
First (S′) = {ε, a}

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、I_2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。

3. 考虑文法

S→AS|b

A→SA|a

(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

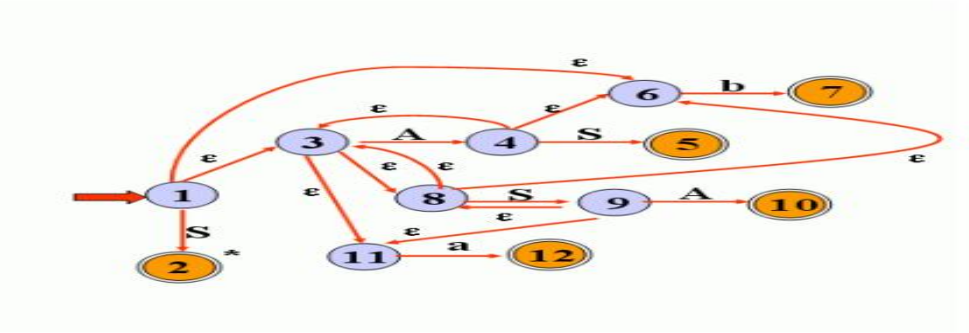
其 LR(0)项目集规范族及识别该文法活前缀的 DFA 如下图所示：

FOLLOW(S)={#, a, b}
FOLLOW(A)={a, b}

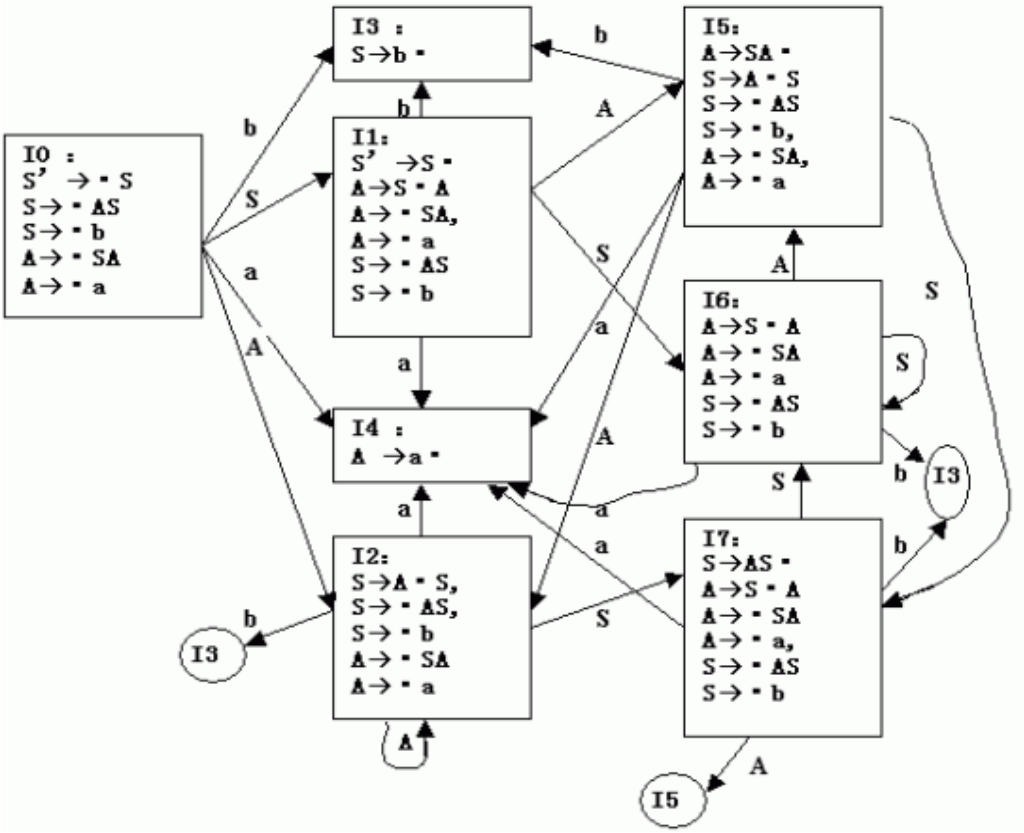
LR(0)项目：

1.S' →.S
2. S' →S.
3. S →.AS
4. S → A.S
5. S → AS.
6. S → .b
7. S → b.
8. A → .SA
9. A → S.A
10. A → SA.
11. A → .a
12. A → a.

（2）NFA 为：

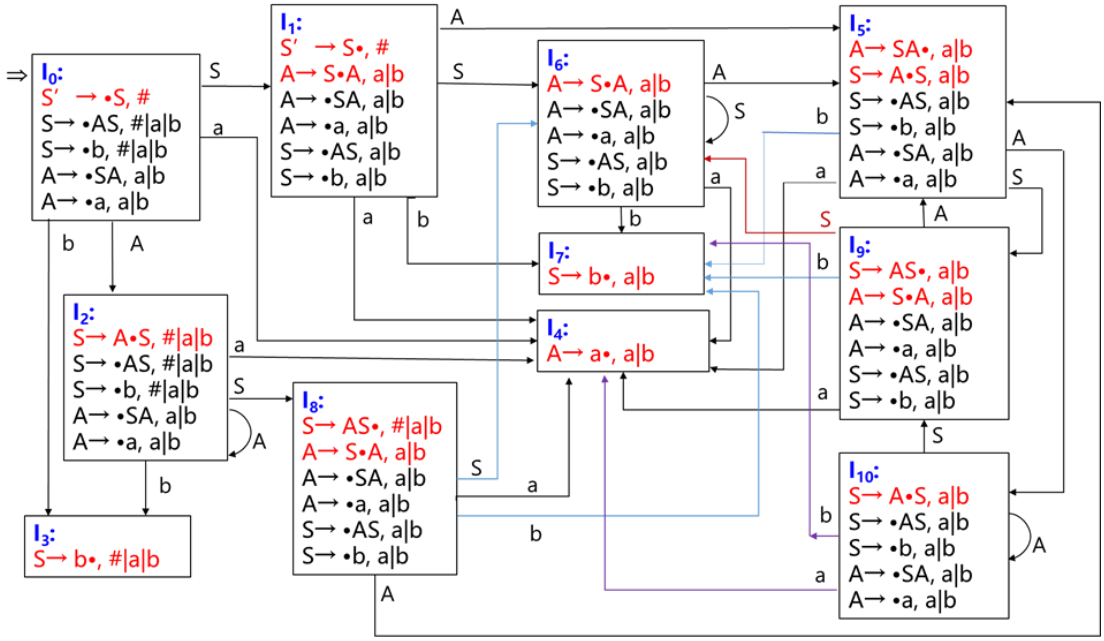


确定化的 DFA 为：



（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）



项目集 I5,I8,I9 存在移进-归约冲突，所以该文法不是 LR(1)文法，当然更不是 LALR(1)文法。

Charpter 7

4. 以下是简单表达式（只含加、减运算）计算的一个属性文法 G（E）：

E→TR　　{R.in:=T.val;　E.val:=R.val}

$R \rightarrow +TR_1$　　$\{R_1.in:=R.in+T.val;\ R.val:=R_1.val\}$

$R \rightarrow -TR_1$　　$\{R_1.in:=R.in-T.val;\ R.val:=R_1.val\}$

R→ ε　　{R.val:=R.in; }

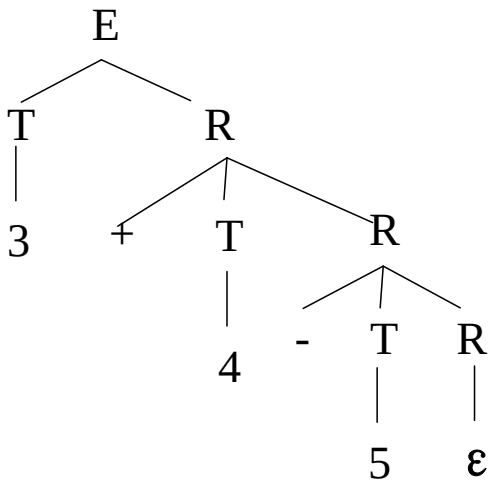
T→num　　{T.val:=lexval(num);}

其中，lexval(num)表示从词法分析程序中得到的常数值。

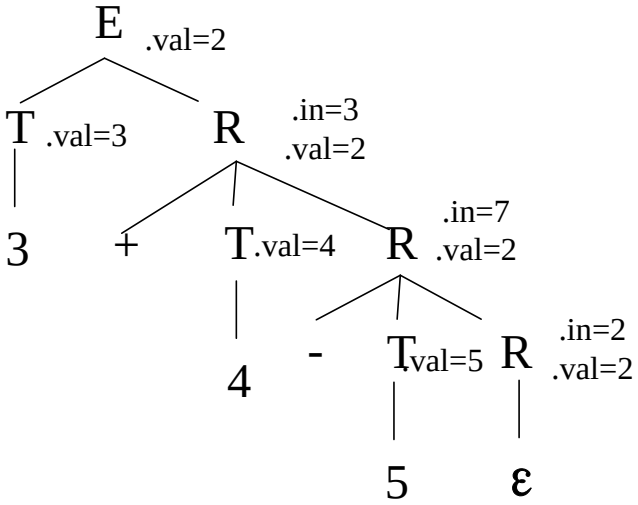
试给出表达式 3+4-5 的语法分析树和相应的带标注语法分析树。

答案：

语法分析树：



带标注语法分析树如下，由于是一个 L 属性文法，所以注意各属性的值是通过语法分析树的遍历时完成的属性计算。



Charpter 8

练习 8 参考解答：拉链与回填技术采用的是 S-翻译模式。

$S \rightarrow for\ (\ S'_1\ ;\ M_1\ E\ ;\ M_2\ S'_2\ N_1\ )\ M_3\ S_1\ N_2$

{

backpatch(E.truelist, M3.gotostm) ;

backpatch(S1.nextlist, M2.gotostm) ;

backpatch(S'1 .nextlist, M1.gotostm) ;

backpatch(S'2.nextlist, M1.gotostm) ;

backpatch(N 1.nextlist, M1.gotostm) ;

backpatch(N 2.nextlist, M2.gotostm) ;

S.nextlist := E.falselist;

}

注意 M,N 的作用，S'_2 后必须有个 N,产生一条 goto 指令，回填时，goto 到 M1.gotostm 重新检查循环终止条件 E 是否满足。

$M \rightarrow \varepsilon$ {M. gotostm := nextstm}

$N \rightarrow \varepsilon$ {N. nextlist := makelist(nextstm); emit('goto __')}

也可以把 N_2 及其相关属性和语义计算规则去掉，在最后加上 emit

$S \rightarrow for\ (\ S'_1\ ;\ M_1\ E\ ;\ M_2\ S'_2\ N_1\ )\ M_3\ S_1$

{

backpatch(E.truelist, M3.gotostm) ;

backpatch(S1.nextlist, M2.gotostm) ;

backpatch(S'1 .nextlist, M1.gotostm) ;

backpatch(S'2.nextlist, M1.gotostm) ;

backpatch(N 1.nextlist, M1.gotostm) ;

S.nextlist := E.falselist;

emit('goto', M2.gotostm);

}

练习 7 参考解答：for 语句的翻译采用的是 L-翻译模式。

$S \rightarrow for\ (\ \{\ S'_1\ .\ next := newlabel\ \}\ S'_1\ ;$

$\{\ E\ .\ true := newlabel\ ;\ E\ .\ false := S\ .next\ \}\ E\ ;$

$\{\ S'_2.\ next := S_1'\ .\ next\ \}\ S'_2\ )$

$\{\ S_1\ .\ next := newlabel\ \}\ S_1$

$\{\ S\ .\ code := S'_1\ .code\ ||\ gen(S'_1\ .next\ ':')\ ||\ E\ .code\ ||\ gen(E\ .\ true\ ':')$

$||\ S_1\ .code\ ||\ gen(S_1\ .next\ ':')\ ||\ S'_2\ .code\ ||\ gen('goto'\ S'_1\ .next)\ \}$

Charpter 9

1. 若按照某种运行时组织方式，如下函数 p 被激活时的过程活动记录如图 9.25 所示。其中 d 是动态数组。

```
static int N;
void p(int a) {
    float b;
    float c[10];
    float d[N];
    float e;
    ……
}
```

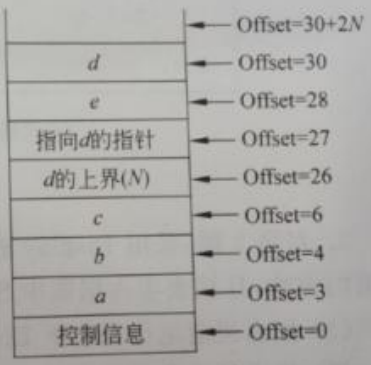


图 9.25

试指出函数 p 中访问 $d[i](0 \leqslant i < N)$ 时相对于活动

记录基址的 Offset 值如何计算？若将数组 c 和 d 的声明次序颠倒，则 $d[i](0 \leqslant i < N)$ 又
何计算？（对于后一问题可选多种不同的运行时组织方式，回答可多样，但需要作相应
解释。）

函数访问 d[i] 的 Offset 的值为： offset = 30 + 2i

若将数组 C 和 d 的声明次序颠倒，由于 d 是一个可变数组，所以 d 的数组元素存放的位置不会因为组 C 和 d 的声明次序颠倒而发生改变，d[i]的起始偏移量仍为 30，offset 不变。

Charpter 10

2. 图 10.26 是图 10.25 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 10.25

(1)	$i:=m-1$	(16)	$t_7:=4*i$
(2)	$j:=n$	(17)	$t_8:=4*j$
(3)	$t_1:=4*n$	(18)	$t_9:=a[t_8]$
(4)	$v:=a[t_1]$	(19)	$a[t_7]:=t_9$
(5)	$i:=i+1$	(20)	$t_{10}:=4*j$
(6)	$t_2:=4*i$	(21)	$a[t_{10}]:=x$
(7)	$t_3:=a[t_2]$	(22)	goto (5)
(8)	if $t_3<v$ goto (5)	(23)	$t_{11}:=4*i$
(9)	$j:=j-1$	(24)	$x:=a[t_{11}]$
(10)	$t_4:=4*j$	(25)	$t_{12}:=4*i$
(11)	$t_5:=a[t_4]$	(26)	$t_{13}:=4*n$
(12)	if $t_5<v$ goto (9)	(27)	$t_{14}:=a[t_{13}]$
(13)	if $i>=j$ goto (23)	(28)	$a[t_{12}]:=t_{14}$
(14)	$t_6:=4*i$	(29)	$t_{15}:=4*n$
(15)	$x:=a[t_6]$	(30)	$a[t_{15}]:=x$

图 10.26

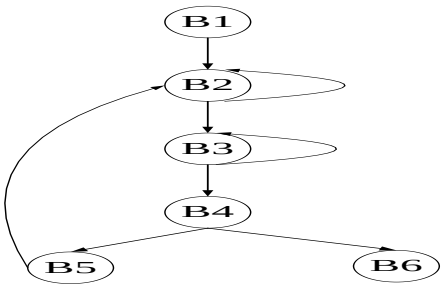
(1) 请将图 10.26 的三地址代码序列划分为基本块并给出其流图。
(2) 将每个基本块的公共子表达式删除。
(3) 找出流图中的循环，将循环不变量计算移出循环外。
(4) 找出每个循环中的归纳变量，并在可能的地方删除它们。

（1） 答案：如下代码，红色标记入口语句

（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）t4:=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）t8:=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

这样就得到 6 个基本快
B1：（1）（2）（3）（4）
B2：（5）（6）（7）（8）
B3：（9）（0）（11）（12）
B4：（13）
B5：（14）……（22）
B6：（23）……（30）



（2）基本块 B1--B4 都没有公共子表达式
B5：　红色标记处为删除公共子表达式的地方
（14）t6:=4*i
（15）x:=a[t6]
（16）t7:= t6
（17）t8:=4*j
（18）t9:=a[t8]
（19）a[t7] :=t9
（20）t10:= t8
（21）a[t10] :=x
（22）goto （5）

B6:

（23）t11:=4*i

（24）x:=a[t11]

（25）t12:= t11

（26）t13:=4*n

（27）t14:=a[t13]

（28）a[t12] :=t14

（29）t15:= t13

（30）a[t15] :=x