1. 文法 G[<S>]的产生式为：

<S>→a|b|(<R>)

<T>→<S>c<T>|<S>

<R>→<T>

1）构造句型（bc<S>c<T>）的推导树，并指出该句型的所有短语、简单短语、句柄。

2）若句柄存在，求下述符号串的句柄。

a) ((<S>c(b)))　　b) (<S>)　　c) (a)　　d) <S>c<T>　　e) <S>　　f) b

分析：符号串的某一部分符号要成为句柄，必须满足：

a）该符号串必须是该文法的句型（句子）。

b）是最左简单短语，所以文法的开始符号不是句柄。

因此只要给出句型(句子)对应的推导树就容易求出短语、简单短语和句柄。

解答：

1) 句型（bc<S>c<T>）的推导树如图 1.1 所示：

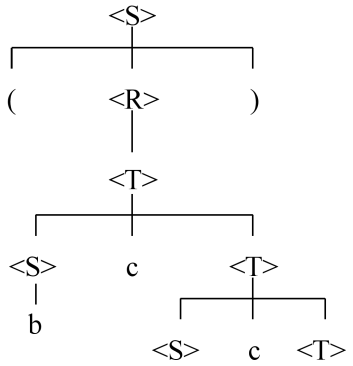

图 1.1 句型（bc<S>c<T>）的推导树

短语：b;bc<S>c<T>;<S>c<T>;(bc<S>c<T>)

简单短语：b; <S>c<T>

句柄：b

2)

a）句型((<S>c(b)))的推导树

如图 1.2 所示：

((<S>c(b)))

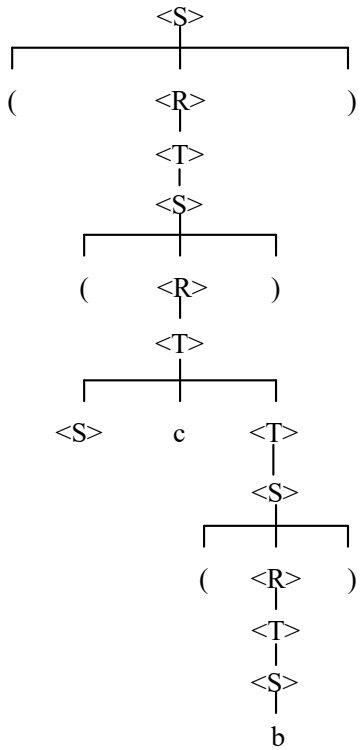

图 1.2　句型((<S>c(b)))的推导树

∴句柄为 b。

b) 句型(<S>)的推导树如图 1.3 所示：

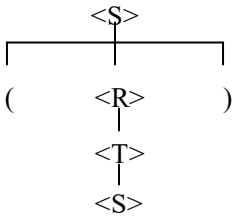

图 1.3 句型(<S>)的推导树

∴句柄为<S>。

c) 句子(a)的推导树如图 1.4 所示：

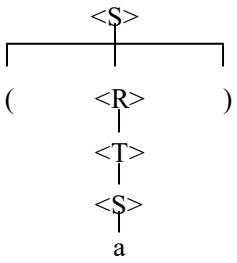

图 1.4 句子(a)的推导树

∴句柄为 a。

d）<S>c<T>不是文法 G[<S>]的句型，∴句柄不存在。

e）<S>是开始符号，∴句柄不存在。

f）句子 b 的推导树如图 1.5 所示：

图 1.5 句子 b 的推导树

∴句柄为 b。

2. 试设计接收以 a 为头符号和尾符号且中间可有任意多个符号 a 或 b 的确定的有限状态自动机 M。

分析：

a) 输入集合 V_T 中只有二个符号 a 和 b，即 V_T={a,b}。

b) a 为头符号和尾符号，因此输入串 a 是满足条件的。

c) 先用状态转换图的方式构造出满足条件的 NFA。

解答：

ⅰ）状态转换图如图 2.4 所示：

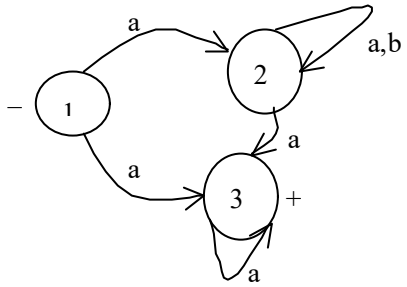

图 2.4　符合题意的不确定有限状态自动机 M

注意：δ(1,a)＝{3}不能少，不然输入串 a 不能被 M 所接收。

ⅱ）确定化

M 的状态转换表（表 2.3）为：

M	a	b	⊥
1	{2, 3}	-	
2	{2, 3}	{2}	
3	{3}	-	F

表 2.3 M 的状态转换表

确定化过程如表 2.4 所示：

M	a	b	⊥
[1]	[23]	-	
[23]	[23]	[2]	F
[2]	[23]	[2]	

表 2.4 M 的确定化

对应的状态转换图如图 2.5 所示：

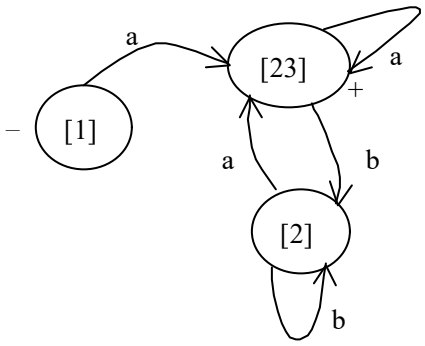

图 2.5　M 的状态转换图

3．①试给出计算上下文无关文法中各非终结符的 FIRST 集和 FOLLOW 集的算法。
②对如下的文法 G[<S>]，利用①给出的算法计算各非终结符的 FIRST 集与 FOLLOW 集。

（1）<S>→<A><B>
（2）<A>→a<A>b
（3）<A>→ε
（4）<B>→b<B>
（5）<B>→c

分析：

a) 计算上下文无关文法中的各非终结符的 FIRST 集和 FOLLOW 集在讨论该文法是否是 LL(1)文法起到重要作用，因为对产生式< A>→ α ，其选择集合为：

Select(< A>→ α)=FIRST(α FOLLOW(<A>))

而文法 G 是 LL(1)文法的充要条件是具有相同左部的产生式的选择集合不相交，同时还可利用各产生式的选择集合机械地构造该文法的句法分析程序。

b) 计算各非终结符的 FIRST 集和 FOLLOW 集可根据定义用观察的方法给出（当产生式个　数较少的时候），也可用布尔矩阵的方法机械地计算。本题给出了计算这两个集合的一般算法。

c) 为了方便理解这两个计算方法，习题②详细地给出了算法的计算过程。

解答：

①计算 FIRST 集的算法描述如下：

for (所有的非终结符号<A>)　FIRST(<A>)={};
do {
　for　(对于每一个产生式< A>→$X_1X_2X_3\ldots X_n$)
　{k=1;continue=true;
　while (continue == true and k<=n)
　　{**add** FIRST(X_k)-{ε} to FIRST(<A>);
　　if (ε is not in FIRST(X_k)) continue=false;
　　k=k+1;
　　}
　if (continue = true) **add** ε to FIRST(<A>);
　}
} **while** (任何一个 FIRST(<A>)集合发生变化)

计算 FOLLOW 集的算法描述如下：

FOLLOW(文法开始符号)={⊥};　　//考虑到增广文法 G[<S'>]所引入的产生式 <S'>→<S>⊥

for (除文法开始符号外的所有的非终结符号<A>)　FOLLOW(<A>)={};

do

{

　　for　(对于每一个产生式< A>→$X_1X_2X_3\ldots X_n$)

　　for　(对于每一个非终结符 X_i)

　　{

　　add FIRST(X_{i+1} $X_{i+2}\ldots X_n$) -{ε} to FOLLOW(X_i);

　　if (ε is in FIRST(X_{i+1} $X_{i+2}\ldots X_n$))

　　add FOLLOW (<A>) to FOLLOW(X_i);

　　}

} **while** (任何一个 FOLLOW(<A>)集合发生变化)

②计算文法 G[<S>]中各非终结符的 FIRST 集：

初始时 FIRST(<S>)=FIRST(<A>)=FIRST(<B>)={}。

对产生式 1 进行计算：

∵FIRST(<A>)={}，

∴continue=false，FIRST(<S>)＝{}。

对产生式 2 进行计算：

∵FIRST(X_1)=FIRST(a)={a}，

∴continue=false，FIRST(<A>)={a}, FIRST(<A>)集合元素发生了变化。

对产生式 3 进行计算：

∵通过 while 循环后，continue 的值仍为 true，

∴FIRST(<A>)={a,ε}, FIRST(<A>)集合元素发生了变化。

对产生式 4 进行计算：

∵FIRST(X_1)=FIRST(b)={b}，

∴continue=false，FIRST(<B>)={b}, FIRST(<B>)集合元素发生了变化。

对产生式 5 进行计算：

∵FIRST(X_1)=FIRST(c)={c}，

∴continue=false，FIRST(<B>)={b,c}, FIRST(<B>)集合元素发生了变化。

由于非终结符的 FIRST 集合发生变化，所以要对每个产生式再做一次计算。下面给出第二次计算的结果：

对产生式 1 进行计算：

∵FIRST(<A>)={a,ε}，

∴continue 的值仍为 true，进一步考察 FIRST(<B>)。

∵FIRST(<B>)＝{b,c}，

∴FIRST(<S>)={a,b,c},FIRST(<S>)集合元素发生了变化。

对产生式 2，3，4，5 的计算结果与第一次计算结果相同，这里不再重复。

由于 FIRST(<S>)集合元素发生了变化，所以要对每个产生式做第三次计算，计算结果同第二次。最终，各非终结符的 FIRST 集合为：

FIRST(<S>)={a,b,c}

FIRST(<A>)={a,ε}

FIRST(<B>)={b,c}

计算文法 G[<S>]中各非终结符的 FOLLOW 集：

初始时 FOLLOW(<S>)={⊥},FOLLOW(<A>)=FOLLOW(<B>)={ }。

对产生式 1 进行计算：

∵FIRST(<B>)={b,c}，

∴FOLLOW(<A>)={b,c},FOLLOW(<A>)集合元素发生了变化。

∵<B>= X_n

∴FOLLOW(<B>)=FOLLOW(<B>)∪FOLLOW(<S>)={⊥}, FOLLOW(<S>)集合 元素发生了变化。

对产生式 2 进行计算：

∵FIRST(b)={b}，

∴FOLLOW(<A>)={b,c}。

对产生式 3 进行计算：

∵在此产生式的右部不存在非终结符。

∴无需进行任何计算。

对产生式 4 进行计算：

∵<B>= X_n

∴FOLLOW(<B>)=FOLLOW(<B>)∪FOLLOW(<B>)={⊥}。

对产生式 5 进行计算：

∵此产生式的右部不存在非终结符，

∴无需进行任何计算。

由于非终结符的 FOLLOW 集合发生变化，所以要对每个产生式再做一次计算，计算结果与第一次计算结果相同，但不存在非终结符的 FOLLOW 集合发生变化。最终，各非终结符的 FOLLOW 集合为：

FOLLOW(<S>)={⊥}

FOLLOW(<A>)={b,c}

FOLLOW(<B>)={⊥}

4.（重点题）设文法 G［<S>］为：

<S>→<S>（<S>）

<S>→a

a）试证 G［<S>］是 LR（0）文法。

b）构造 G［<S>］的 LR（0）句法分析控制表。

c）给出识别句子 a（a（a））的自底向上句法分析的过程。

分析：

a) 可以用图或表的形式构造文法的状态集及转换关系（识别活前缀的 DFA）。二种方法各有所长，可根据自己的喜好，选择一种。

b) 下推自动机进行自底向上句法分析的过程中主要的动作是移入（Shift）和归约（Reduce）。移入时，将所转向的状态一起入栈（亦可单独分开表示），归约时，实际上有二个动作：

i)将归约所采用的产生式的右部符号（连同它们所相关的状态）一起出栈。

ii)将归约所采用的产生式的左部符号视为输入符，再按移入时的方法处理。

例如：在分析句子 a(a(a)) ⊥的过程中（见表 4.16），第一次归约时的“慢动作”是：

	下推栈（底－顶）	输入串	动作
1	T_0	a(a(a)) ⊥	Shift
2	T_0a T_1	(a(a)) ⊥	$R_{\#2}$
3’	T_0	<S>a(a(a)) ⊥	∵goto[T_0,<S>]= T_1
3	T_0<S> T_1	(a(a)) ⊥	Shift
	…	…	…

表 4.16 句子 a(a(a)) ⊥句法分析过程中第一次归约时的情形

解答：

a）

ｉ）识别活前缀 DFA 的表格表示（表 4.17）。

状态 T	项目集	输入符号	下一状态
0	* <S’>→ • <S>⊥	<S>	1
	<S>→ • <S>(<S>)	<S>	1
	<S>→ • a	a	2
1	* <S’>→<S> • ⊥	⊥	Accept
	* <S>→<S> • (<S>)	(	3
2	* <S>→a •		#2
3	* <S>→<S>(• <S>)	<S>	4
	<S>→ • <S>(<S>)	<S>	4
	<S>→ • a	a	2
4	* <S>→<S>(<S> •)	)	5
	* <S>→<S> • (<S>)	(	3
5	* <S>→<S>(<S>) •		#1

表 4.17　识别活前缀 DFA 的表格形式

ii）识别活前缀 DFA 的图形表示如图 4.3 所示：

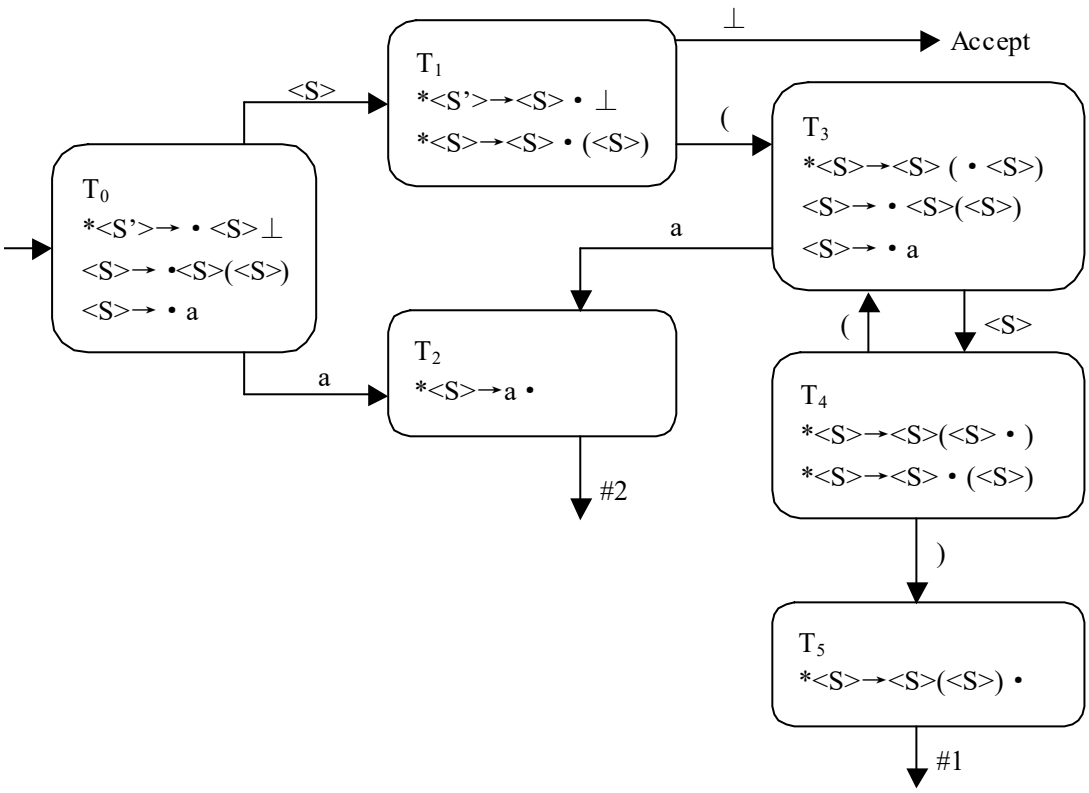

图 4.3　识别活前缀的 DFA 图形形式

∵无不适定状态 ∴G［<S>］是 LR（0）文法。
注：带‘*’的 LR(0)项目是该状态的核心项目。

b）G［<S>］的 LR（0）句法分析控制表（表 4.18）为：

状态	转向表（Goto）				动作表（Action）			
T	a	(	)	<S>	a	(	)	⊥
0	2			1	S			
1		3				S		Accept
2					#2	#2	#2	#2
3	2			4	S			
4		3	5			S	S	
5					#1	#1	#1	#1

表 4.18　G［<S>］的 LR（0）句法分析控制表

c）句子 a（a（a））的分析过程如表 4.19 所示：

	下推栈（底-顶）	输入串	动作
1	T_0	a(a(a))⊥	S
2	T_0aT_2	(a(a))⊥	R#2
3	$T_0<S>T_1$	(a(a))⊥	S

4	$T_0<S>T_1(T_3$	a(a))⊥	S
5	$T_0<S>T_1(T_3aT_2$	(a))⊥	R#2
6	$T_0<S>T_1(T_3<S>T_4$	(a))⊥	S
7	$T_0<S>T_1(T_3<S>T_4(T_3$	a))⊥	S
8	$T_0<S>T_1(T_3<S>T_4(T_3aT_2$	))⊥	R#2
9	$T_0<S>T_1(T_3<S>T_4(T_3<S>T_4$	))⊥	S
10	$T_0<S>T_1(T_3<S>T_4(T_3<S>T_4)T_5$	)⊥	R#1
11	$T_0<S>T_1(T_3<S>T_4$	)⊥	S
12	$T_0<S>T_1(T_3<S>T_4)T_5$	⊥	R#1
13	$T_0<S>T_1$	⊥	Accept

表 4.19　句子 a（a（a））的分析过程

注：表中 R#i 表示用编号为 i 的产生式归约，S 表示移入。