第一章作业

1. 解释下列术语:编译程序,源程序,目标程序,编译程序的前端、后端和遍。

答:

(1)编译程序:源语言为高级语言,目标语言为汇编语言或机器语言的翻译程序称为编译程序。
(2)源程序:待翻译的程序,即高级程序设计语言编写的程序。
(3)目标程序:翻译后的程序,即汇编语言或机器语言。
(4)前端:编译过程中主要依赖于源语言与目标机无关的阶段,包括词法分析、语法分析、语义分析和中间代码生成。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:指依赖于目标机而一般不依赖源语言,只与中间代码有关的阶段,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。

2. 编译程序有哪些主要构成成分?各自的主要功能是什么?

答:

编译程序的主要构成成分有: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、代码优化程序、目标代码生成程序、表格管理程序及出错处理程序。
(1)词法分析程序:从左到右扫描源程序,识别出各个单词,确定单词的类型并将其转换成单词串;同时查词法错误,进行标识符登记,即符号表管理。
(2)语法分析程序:识别由词法分析给出的单词符号串是否是给定文法的句子
(3)语义分析程序:审查源程序是否有语义错误,为代码生成阶段收集类型信息,当不符合规范的时候报错。
(4)中间代码生成程序:将源程序转换成一种内部表示形式,如三地址指令或四元式。
(5)中间代码优化程序:对中间代码进行等价变换处理以提高执行效率。
(6)目标代码生成程序:将优化的中间代码转换成目标机上的机器指令代码或汇编代码。
(7)表格管理程序:管理各种符号表(常数、标号、变量、过程、结构……),查、填(登记、查找)源程序中出现的符号和编译程序生成的符号,为编译的各个阶段提供信息。
(8)错误处理程序:进行各种错误的检查、报告、纠正,以及相应的续编译处理。

3. 什么是解释程序?它与编译程序的主要不同是什么?

答:

解释程序直接执行源程序给出运行结果。工作模式:一个个的获取、分析并执行源程序语句,一旦第一个语句分析结束,源程序便开始运行并且生成结果。

不同:编译程序将源高级语言程序翻译成汇编或机器语言程序,而解释程序则是分析处理源高级语言程序直接计算结果,不生成目标语言程序。

4. 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:

*else*没有匹配的*if*	语法分析
数组下标越界	语义分析
使用的函数没有定义	语义分析
在数中出现非数字字符	词法分析

第二章作业

1. 文法$G=(\{A,B,S\},\{a,b,c\},P,S)$，其中$P$为

$S\rightarrow Ac|aB$

$A\rightarrow ab$

$B\rightarrow bc$

写出$L(G[S])$的全部元素。

$S \Rightarrow Ac$

答：

$L(G[S])=\{abc\}$

2. 文法$G[N]$为

$N\rightarrow D|ND$

$D\rightarrow 0|1|2|3|4|5|6|7|8|9$

$G[N]$的语言是什么？

答：

$L(G[N])=V^{+}$，其中$V=\{0,1,2,3,4,5,6,7,8,9\}$，即长度大于等于*1*的数字串

3. 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：

$G[S]:S\rightarrow S+D|S-D|D$

$D\rightarrow 0|1|2|3|4|5|6|7|8|9$

（注：表达同一语言的文法可能不唯一）

5. 已知文法$G[Z]$：

$Z::=aZb$

$Z::=ab$

写出$L(G[Z])$的全部元素。

答：

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

8. 考虑下面的上下文无关文法：

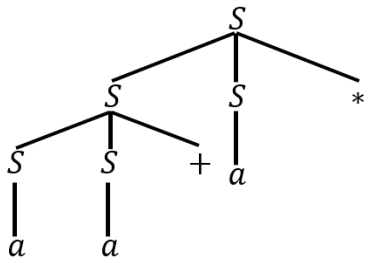
$S\rightarrow SS*|SS+|a$

(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)该文法生成的语言是什么？

答：(1)推导：$S=>SS*=>Sa*=>SS+a*=>Sa+a*=>aa+a*$

语法树如下图所示：



(2)该文法生成的语言是符号a的$*$和$+$运算的后缀表达式。

9. 已知文法$S \to S(S)S|\varepsilon$。

(1)该文法生成的语言是什么?

(2)该文法是二义的吗?说明理由。

答:

(1) 该文法生成的语言是匹配的括号

(2) 是二义的,因为对于()()可以构造两棵不同的语法树

10. 令文法$G[E]$为

$E \to T|E+T|E-T$

$T \to F|T*F|T/F$

$F \to (E)|i$

证明$E+T*F$是它的一个右句型,指出这个句型的所有短语、直接短语和句柄。

答:

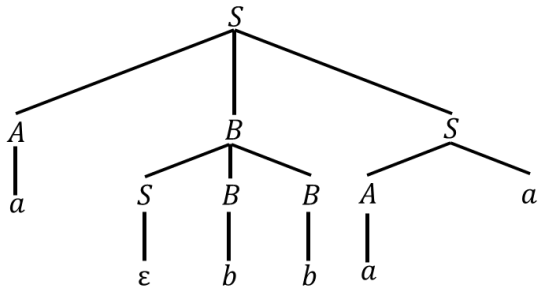
最右推导为:$E => E+T => E+T*F$,所以$E+T*F$是该文法的一个右句型。

短语:$E+T*F, T*F$

直接短语:$T*F$

句柄:$T*F$

11.一个上下文无关文法生成句子$abbaa$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)找出该句子的所有短语、简单短语、句柄。

答:

(1)最左推导:

$S => ABS => aBS => aSBBS => aBBS => abBS => abbS => abbAa => abbaa$

最右推导:

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

(2)产生式有：$S \to ABS|Aa|\varepsilon \quad A \to a \quad B \to SBB|b$

(3)短语：$a\varepsilon bbaa, a, \varepsilon bb, aa, b, \varepsilon$

直接短语：a, ε, b

句柄：a

12.构造产生如下语言的上下文无关文法各一个：

(1) $\{a^n b^n | n \geq 0\}$

(2) $\{a^m b^n | m \geq n \geq 0\}$

(3) $\{uawb \mid u, w \in \{a, b\}^* \wedge |u| = |w|\}$

(4) $\{a^n b^m | n \geq 2m \geq 0\}$

(5) $\{a^n b^m | n \geq 0, m \geq 0$,且$3n \geq m \geq 2n\}$

(6) $\{ww^R | w \in \{a, b\}^*\}$，其中，$w^R$表示$w$的反向串，其含义是将$w$中的字母依次反转，首尾字母交换位置，下同

(7) $\{uvwv^R | u, v, w \in \{a, b\}^+ \wedge |u| = |w| = 1\}$

(8) $\{w | w \in \{a, b\} \wedge w = w^R\}$

答：

$\{a^n b^n \mid n \geq 0\}$	$S \to aSb\|\varepsilon$
$\{a^m b^n \mid m \geq n \geq 0\}$	$S \to aSb\|aS\|\varepsilon$
$\{uawb \mid u, w \in \{a, b\}^* \wedge \|u\| = \|w\|\}$	$S \to Tb$ $T \to aTb\|aTa\|bTa\|bTb$ $T \to a$
$\{a^n b^m \mid n \geq 2m \geq 0\}$	$S \to aaSb\|aS\|\varepsilon$
$\{a^n b^m \mid n \geq 0, m \geq 0$,且$3n \geq m \geq 2n\}$	$S \to aSbb\|aSbbb\|\varepsilon$
$\{ww^R \mid w \in \{a, b\}^*\}$	$S \to aSa\|bSb\|\varepsilon$
$\{uvwv^R \mid u, v, w \in \{a, b\}^+ \wedge \|u\| = \|w\| = 1\}$	$G[S]$: $S \to aA\|bA$ $A \to aBa\|bBb$ $B \to aBa\|bBb\|a\|b$
$\{w \mid w \in \{a, b\} \wedge w = w^R\}$	$S \to a\|b$

第三章作业

1. 构造下列正规式相应的 DFA。

（1）$1(0|1)*101$

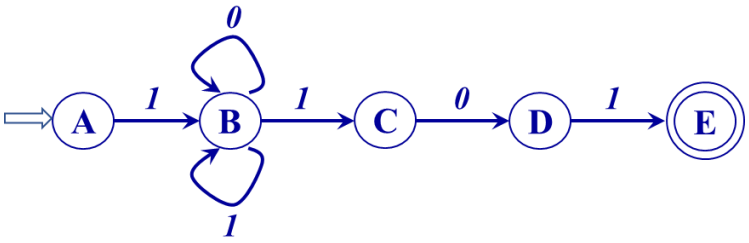
（2）$1(1010^*|1(010)^*1)^*0$

（3）$a((a|b)^*|ab^*a)^*b$

（4）$b((ab)^*|bb)^*ab$

答：

(1)正规式$1(0|1)*101$对应的 NFA 如下：



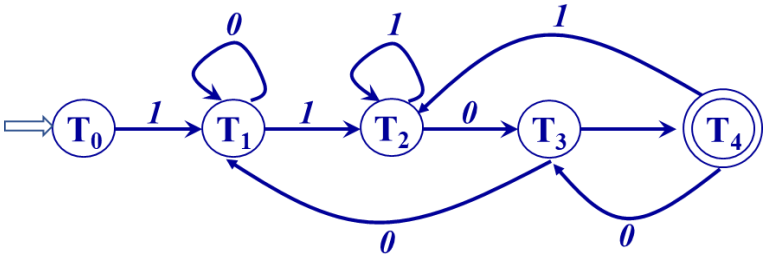
用子集法将此 NFA 确定化的状态转换表如下：

输入符号 / 状态集T	1	0
$S_0=\{A\}$	$\{B\}$	$\emptyset$
$S_1=\{B\}$	$\{B\}$	$\{B,C\}$
$S_2=\{B,C\}$	$\{B,C\}$	$\{B,D\}$
$S_3=\{B,D\}$	$\{B,C,E\}$	$\{B\}$
$S_4=\{B,C,E\}$	$\{B,C\}$	$\{B,D\}$

初态：S_0

终态：S_4

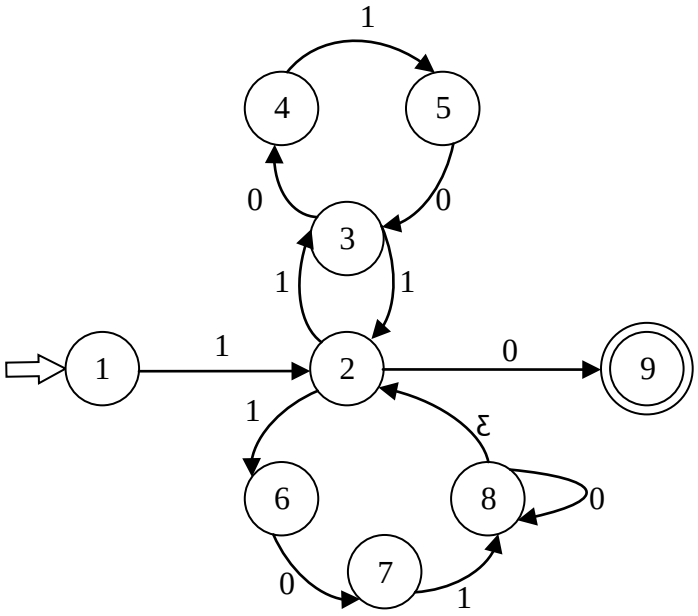
生成的DFA如下为：



分割法最小化

$\Pi=\{T_0\},\{T_1\},\{T_2\},\{T_3\},\{T_4\}$

（2）正规式1(1010*|1(010)*1)*0对应的 NFA 如下：



用子集法将此 NFA 确定化的状态转换表如下：

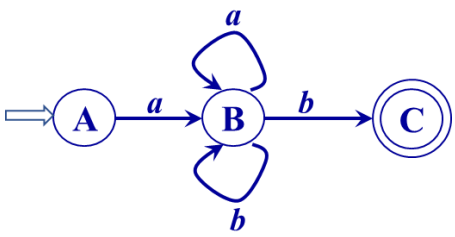
状态集T	符号a	$Move(T,a)$	$\varepsilon\_closure(Move(T,a))$
$T_0=\{1\}$	0 1	∅ {2}	∅ {2}
$T_1=\{2\}$	0 1	{9} {3,6}	{9} {3,6}
$T_2=\{3,6\}$	0 1	{4,7} {2}	{4,7} {2}
$T_3=\{4,7\}$	0 1	∅ {5,8}	∅ {2,5,8}
$T_4=\{2,5,8\}$	0 1	{3,8,9} {3,6}	{2,3,8,9} {3,6}
$T_5=\{2,3,8,9\}$	0 1	{4,8,9} {2,3,6}	{2,4,8,9} {2,3,6}
$T_6=\{2,4,8,9\}$	0 1	{8,9} {3,5,6}	{2,8,9} {3,5,6}
$T_7=\{2,8,9\}$	0 1	{8,9} {3,6}	{2,8,9} {3,6}
$T_8=\{2,3,6\}$	0 1	{4,7,9} {2,3,6}	{4,7,9} {2,3,6}
$T_9=\{3,5,6\}$	0 1	{3,4,7} {2}	{3,4,7} {2}
$T_{10}=\{4,7,9\}$	0 1	∅ {5,8}	∅ {5,8,2}
$T_{11}=\{3,4,7\}$	0 1	{4} {2,5,8}	{4} {2,5,8}
$T_{12}=\{4\}$	0	∅	∅

	1	{5}	{5}
$T_{13}=\{5\}$	0 1	{3} ∅	{3} ∅
$T_{14}=\{3\}$	0 1	{4} {2}	{4} {2}
$T_{15}=\{9\}$	0 1	∅ ∅	∅ ∅

初态： T_0
终态：T_5，T_6，T_7，T_{10}，T_{15}

生成的*DFA*如下为：（图略）

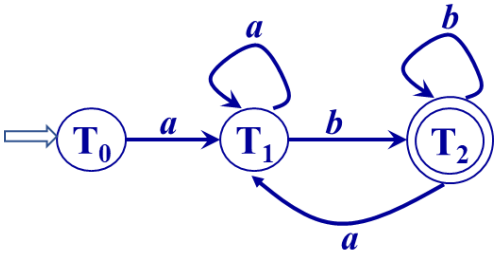
(3)正规式$a((a|b)^*|ab^*a)^*b$对应的 NFA 如下：



用子集法将此 NFA 确定化的状态转换表如下：

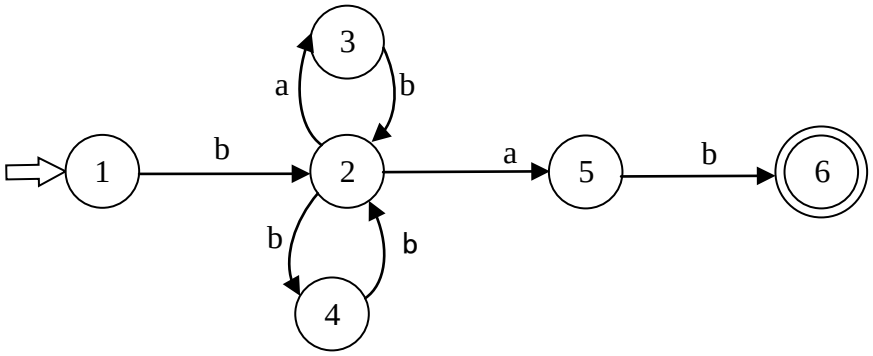
状态集T	符号a	$Move(T,a)$
$T_0=\{A\}$	a b	$\{B\}$ ∅
$T_1=\{B\}$	a b	$\{B\}$ $\{B,C\}$
$T_2=\{B,C\}$	a b	$\{B\}$ $\{B,C\}$

生成的*DFA*如下为：



分割法最小化
$\Pi=\{T_0\},\{T_1\},\{T_2\}$

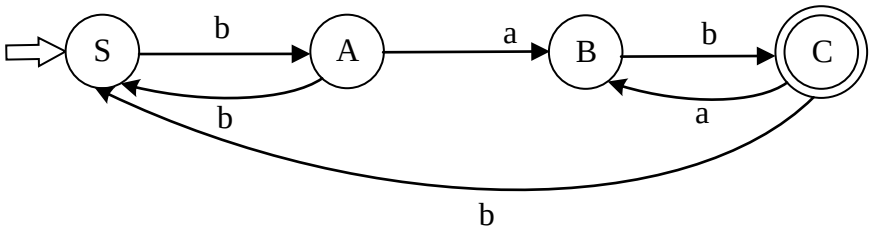
（4）正规式$\boldsymbol{b}((\boldsymbol{ab})^*$|bb)*ab对应的NFA为：



输入符号 / 状态集T	a	b
$S_0=\{1\}$	$\emptyset$	$\{2\}$
$S_1=\{2\}$	$\{3,5\}$	$\{4\}$
$S_2=\{3,5\}$	$\emptyset$	$\{2,6\}$
$S_3=\{4\}$	$\emptyset$	$\{2\}$
$S_4=\{2,6\}$	$\{3,5\}$	$\{4\}$

初态：S_0
终态：S_4
分割法最小化
S_0和S_3等价
$\Pi=\{S_0\},\{S_1\},\{S_2\},\{S_4\}$
令:$S=\{S_0\},A=\{S_1\},B=\{S_2\},C=\{S_4\}$

生成的$\boldsymbol{DFA}$如下为：



4. 把图 3.17(a)和(b)中的NFA分别确定化和最小化。

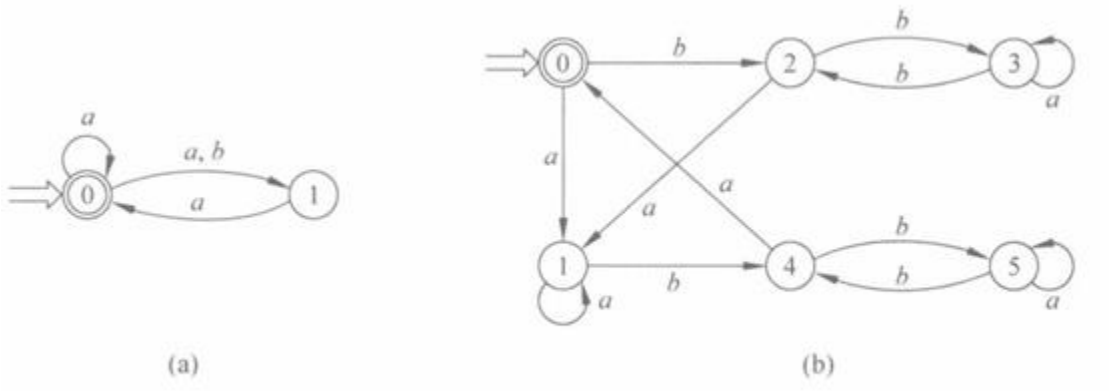


图 3.17 NFA

（a）用子集法将此 NFA 确定化的状态转换表如下：

状态集T	符号a	$Move(T,a)$
$T_0=\{0\}$	a b	$\{0,1\}$ $\{1\}$
$T_1=\{1\}$	a b	$\{0\}$ $\emptyset$
$T_2=\{0,1\}$	a b	$\{0,1\}$ $\{1\}$

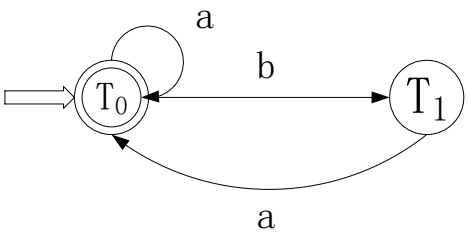
初态： T_0

终态：T_0，T_2

分割法最小化

T_0和T_2等价，删掉T_2用T_0代替出现T_2的位置。

生成的最简*DFA*如下为：



（b）此 FA 是 DFA。

采用分割法进行最小化：

第一次分割π：{1, 2, 3, 4, 5} {0}

状态 4 输入为 a 时结果为 0，将状态 4 分割出去。

第二次分割：{1, 2, 3, 5} {4} {0}

对于划分{1, 2, 3, 5}输入为 b 时将其划分为{1, 5} {2, 3}

第三次分割：{1,5} {2,3} {4} {0}

对于划分{2,3}输入为 a 时可分割为{2} {3}

最终结果为：{1,5} {2} {3} {4} {0}

删除状态 5，用状态 1 代替状态 5 （状态转换图略）

5. 构造一个 DFA，它接受$\sum=\{0,1\}$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。然后构造该语言的正规文法。

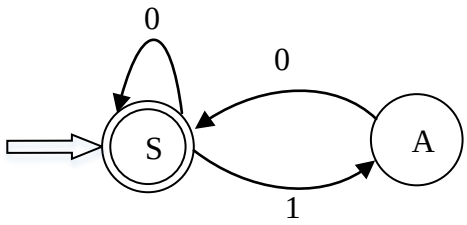
正规式为：（0|10）*

正规文法：

$\boldsymbol{S \rightarrow 0S|1A|0|\varepsilon}$

$\boldsymbol{A \rightarrow 0S}$

构造的 DFA 为：



7. 为正规文法$G[S]$

$\boldsymbol{S \rightarrow aA|bQ}$

$\boldsymbol{A \rightarrow aA|bB|b}$**T**

$\boldsymbol{B \rightarrow bD|aQ}$

$\boldsymbol{Q \rightarrow aQ|bD|b}$**T**

$\boldsymbol{D \rightarrow bB|aA}$

$\boldsymbol{E \rightarrow aB|bF}$

$\boldsymbol{F \rightarrow bD|aE|b}$**T**　　**T 是状态图中的终态**

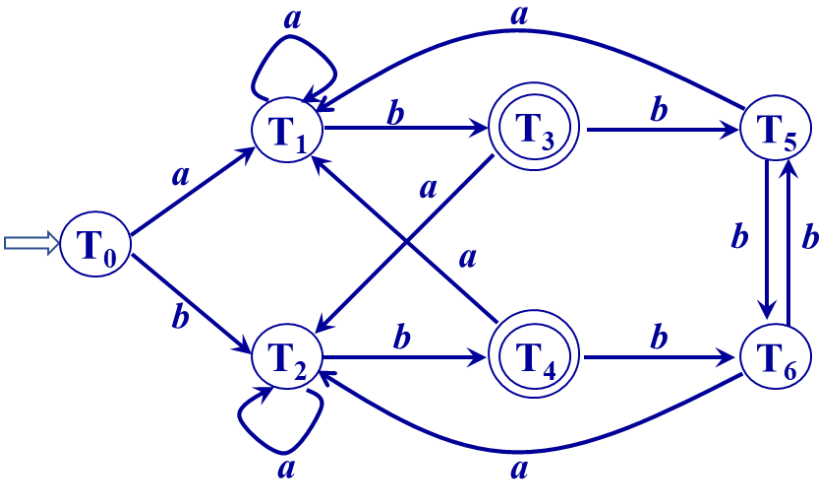
构造相应的最小的DFA

答：用子集法将此 NFA 确定化的状态转换表如下：

状态集T	符号a	$Move(T,a)$
$T_0=\{S\}$	a b	$\{A\}$ $\{Q\}$
$T_1=\{A\}$	a b	$\{A\}$ $\{B,T\}$
$T_2=\{Q\}$	a b	$\{Q\}$ $\{T,D\}$
$T_3=\{B,T\}$	a b	$\{Q\}$ $\{D\}$
$T_4=\{D,T\}$	a b	$\{A\}$ $\{B\}$

$T_5 = \{D\}$	a b	$\{A\}$ $\{B\}$
$T_6 = \{B\}$	a b	$\{Q\}$ $\{D\}$

生成的*DFA*如下为：

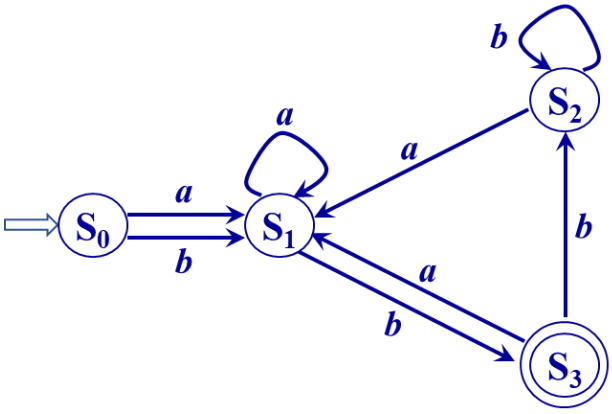


分割法最小化：

$\Pi = \{T_0, T_1, T_2, T_5, T_6\}\{T_3, T_4\}$

$\Pi = \{T_0\}\{T_5, T_6\}\{T_1, T_2\}\{T_3, T_4\}$

令$S_0 = \{T_0\},\ S_1 = \{T_5, T_6\},\ S_2 = \{T_1, T_2\},\ S_3 = \{T_3, T_4\}$，**则最小化的*DFA*：**



8. 给出下述正规文法所对应的正规式：

$S \rightarrow 0A|1B$

$A \rightarrow 1S|1$

$B \rightarrow 0S|0$

答：

将 A 和 B 的产生式代入 S 中：

$S \rightarrow 01S|01|10S|10$

$S \rightarrow (01|10)S|(01|10)$

对 S 进行求解：

$(01|10)^*(01|10)$

9. 构造下述文法$G[S]$的自动机：

$S \to A0$

$A \to A0|S1|0$

该自动机是确定的吗？若不确定，则对它确定化。该自动机相应的语言是什么？

说明：产生式形式为$A \to a$或$A \to Ba, B, A \in V_N, a \in V_T^*$的文法也是正规文法，并称为左线性文法。为左线性文法$G[S]$构造$NFA\ M$的规则如下：

（1） 字母表与 G 的终结符集相同。

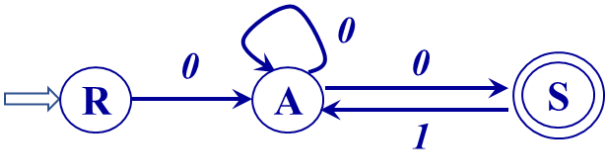
（2） G中的每个非终结符生成M的一个状态。

（3） G 的开始符对应M的终态。

（4） 增加一个新的状态F，作为M的初态。

（5） 对 G 中的形如$A \to Ba$的产生式，构造M的转换函数$f(B, a) = A$；对$A \to a$，构造$f(F, a) = A$。

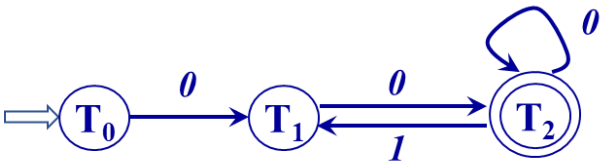
答：NFA 如下：



用子集法将此 NFA 确定化的状态转换表如下：

状态集T	符号a	$Move(T, a)$
$T_0 = \{R\}$	0 1	$\{A\}$ $\emptyset$
$T_1 = \{A\}$	0 1	$\{A, S\}$ $\emptyset$
$T_2 = \{A, S\}$	0 1	$\{A, S\}$ $\{A\}$

DFA 如下：



此 DFA 所识别的语言的正规式为： $00(0|10)^*$

12. 文法 G[<单词>]为

<单词>→<标识符> | <整数>
<标识符>→<标识符><字母> | <标识符><数字> | <字母>
<整数>→<整数><数字> | <数字>
<字母>→A|B|…|Y|Z
<数字>→0|1|2|…|8|9

改写 G 为 G',使 G'为与 G 等价的正规文法。 给出相应的有穷自动机。

答： 令符号 W 表示单词，I 表示标识符，D 表示整数，a 表示字母类的终结符，b 表示数字类的终结符

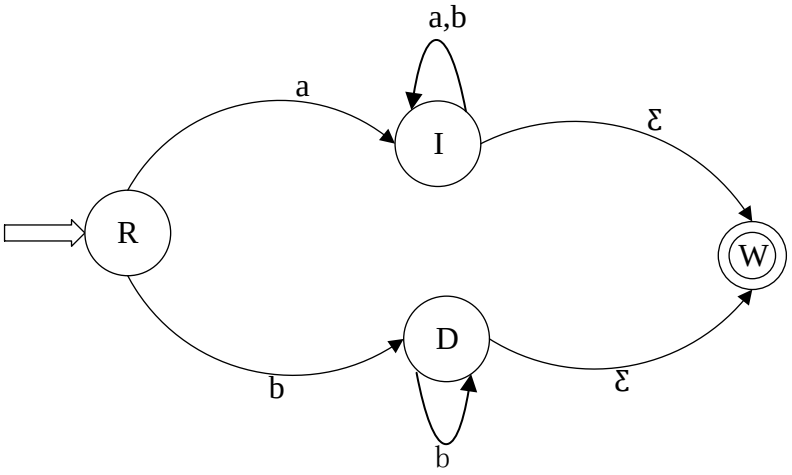
则文法为：

W→I|D

I→Ia|Ib|a

D→Db|b

该文法为一左线性文法。构造自动机如下所示：



用子集法将此 NFA 确定化的状态转换表如下：

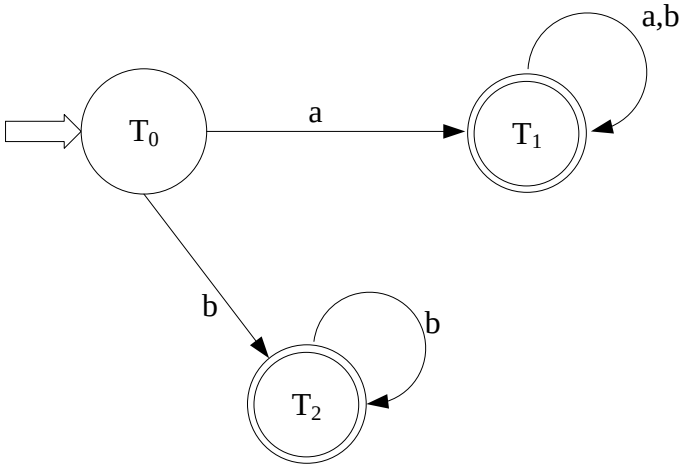
状态集T	符号a	$Move(T,a)$	$\varepsilon\_closure(Move(T,a))$
$T_0=\{R\}$	a b	$\{I\}$ $\{D\}$	$\{I,W\}$ $\{D,W\}$
$T_1=\{I,W\}$	a bc	$\{I\}$ $\{I\}$	$\{I,W\}$ $\{I,W\}$
$T_2=\{D,W\}$	a b	$\emptyset$ $\{D\}$	$\emptyset$ $\{D,W\}$

初态： T_0

终态：T_1,T_2

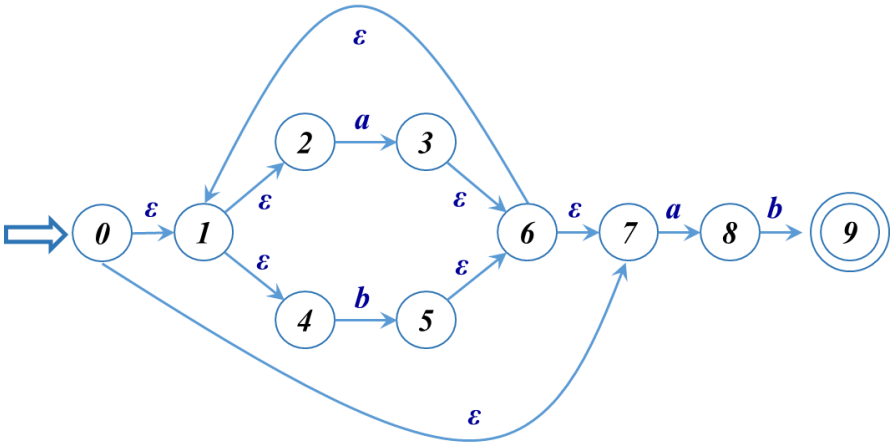
已经是最简化的 DFA

生成的*DFA*如下图为：



补充：将 PPT 中 NFA 的确定化

(1)



答：用子集法将此 NFA 确定化的状态转换表如下：

状态集T	符号a	$Move(T,a)$	$\varepsilon\_closure(Move(T,a))$
$T_0=\{0,1,2,4,7\}$	a b	{3,8} {5}	{1,2,3,4,6,7,8} {1,2,4,5,6,7}
$T_1=\{1,2,3,4,6,7,8\}$	a b	{3,8} {5,9}	{1,2,3,4,6,7,8} {1,2,4,5,6,7,9}
$T_2=\{1,2,4,5,6,7\}$	a b	{3,8} {5}	{1,2,3,4,6,7,8} {1,2,4,5,6,7}
$T_3=\{1,2,4,5,6,7,9\}$	a b	{3,8} {5}	{1,2,3,4,6,7,8} {1,2,4,5,6,7}

初态：T_0

终态：T_3

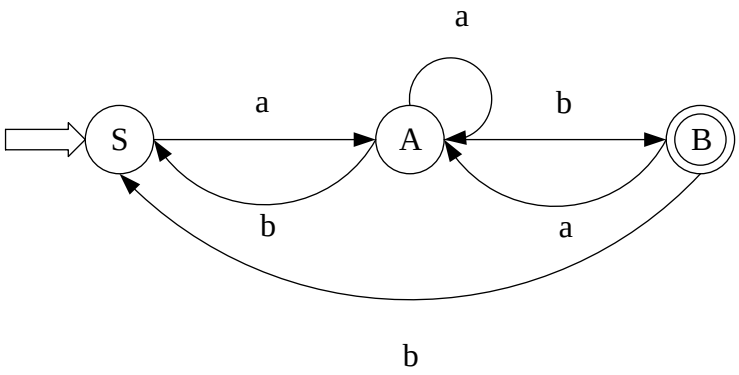
分割法最小化：

$\Pi = \{T_0, T_1, T_2\}\{T_3\}$

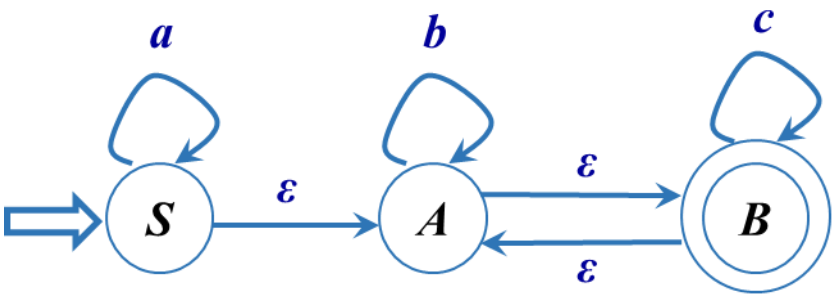
$\Pi = \{T_0, T_2\}\{T_1\}\{T_3\}$

令$S = \{T_0, T_2\}$ $A = \{T_1\}$ $B = \{T_3\}$

生成的*DFA*如下图为：



(2)

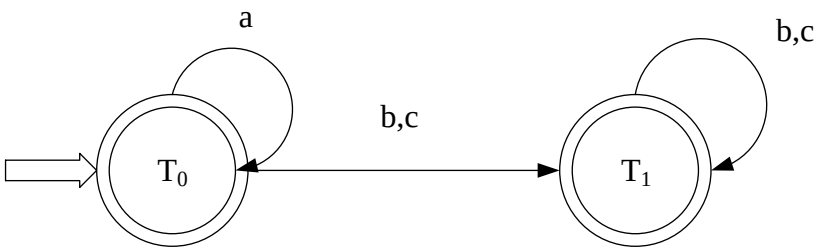


答：用子集法将此 NFA 确定化的状态转换表如下：

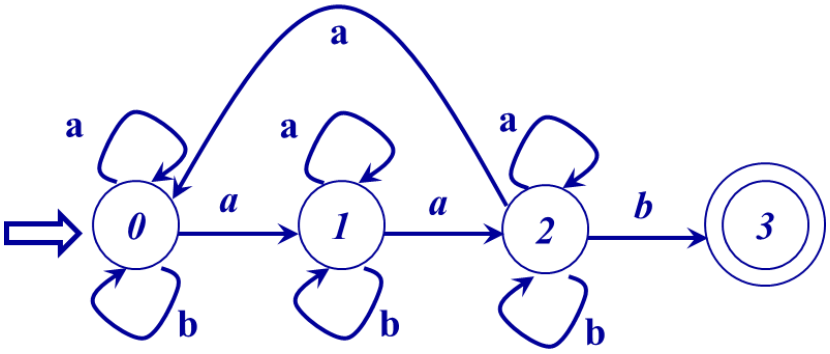
状态集T	符号a	$Move(T,a)$	$\varepsilon\_closure(Move(T,a))$
$T_0 = \{S, A, B\}$	a	$\{S\}$	$\{S, A, B\}$
	b	$\{A\}$	$\{A, B\}$
	c	$\{B\}$	$\{A, B\}$
$T_1 = \{A, B\}$	a	$\emptyset$	$\emptyset$
	b	$\{A\}$	$\{A, B\}$
	c	$\{B\}$	$\{A, B\}$

已经是状态数最小的 DFA

生成的*DFA*如下图为：



(3)



答：用子集法将此 NFA 确定化的状态转换表如下：

状态集T	符号a	$Move(T,a)$
$T_0=\{0\}$	a b	$\{0,1\}$ $\{0\}$
$T_1=\{0,1\}$	a b	$\{0,1,2\}$ $\{0,1\}$
$T_2=\{0,1,2\}$	a b	$\{0,1,2\}$ $\{0,1,2,3\}$
$T_3=\{0,1,2,3\}$	a b	$\{0,1,2\}$ $\{0,1,2,3\}$

初态： T_0

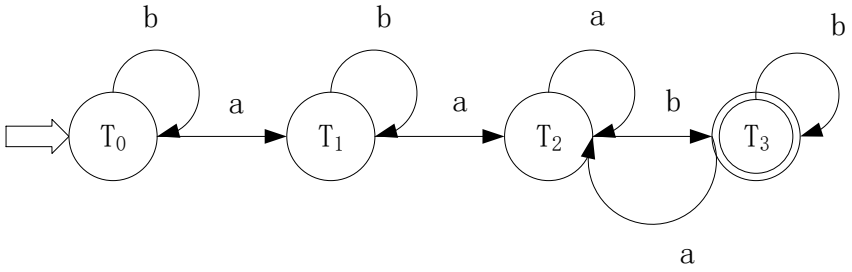
终态：T_3,

分割法最小化：

$\Pi=\{T_0,T_1,T_2\}\{T_3\}$

$\Pi=\{T_0\}\{T_2\}\{T_1\}\{T_3\}$

DFA 如下：


b
b
a
b
a
a
b
T0
T1
T2
T3
a

第四章作业

1.对文法 G[S]

S→a|Λ|(T)

T→T,S|S

(1)给出(a,(a,a))和((a,a),Λ,(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：

(1)(a,(a,a))和((a,a),Λ,(a)),a)的最左推导。

((a,a), Λ,(a)),a)的最左推导：

$S \Rightarrow (T)$

$\Rightarrow (T,S)$

$\Rightarrow (S,S)$

$\Rightarrow ((T),S)$

$\Rightarrow ((T,S),S)$

$\Rightarrow ((T,S,S),S)$

$\Rightarrow ((S,S,S),S)$

$\Rightarrow (((T),S,S),S)$

$\Rightarrow (((T,S),S,S),S)$

$\Rightarrow (((S,S),S,S),S)$

$\Rightarrow (((a,S),S,S),S)$

$\Rightarrow (((a,a),S,S),S)$

$\Rightarrow (((a,a),\Lambda,S),S)$

$\Rightarrow (((a,a),\Lambda,(T)),S)$

$\Rightarrow (((a,a),\Lambda,(S)),S)$

$\Rightarrow (((a,a),\Lambda,(a)),S)$

$\Rightarrow (((a,a),\Lambda,(a)),a)$

(a,(a,a)) 的最左推导：

$S \Rightarrow (T)$

$\Rightarrow (T,S)$

$\Rightarrow (S,S)$

$\Rightarrow (a,S)$

$$\Rightarrow (a,(T))$$
$$\Rightarrow (a,(T,S))$$
$$\Rightarrow (a,(S,S))$$
$$\Rightarrow (a,(a,S))$$
$$\Rightarrow (a,(a,a))$$

（2）消除文法的左递归：

$$S \rightarrow a \mid \wedge \mid (T)$$

$$T \rightarrow ST'$$

$$T' \rightarrow ,ST' \mid \varepsilon$$

递归下降分析程序

```
1.  void MatchToken(char expected)
2.  {
3.     if (lookahead != expected)
4.      {
5.          printf("syntax error \n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
12. }
13. // 解析非终结符 T:T→ST’
14. void ParseT()
15. {
16.     if (lookahead == 'a' || lookahead == '^' || lookahead == '(')
17.       ParseS();
18.       ParseT’();
19.    else {
20.         printf("syntax error \n");
21.         exit(0);
22.       }
23. }
24. // 解析非终结符 T’:T’→,ST’|ε
25. void ParseT’()
26. {
27.    if (lookahead == ',')
28.         {
```

```
29.          MatchToken(',');
30.          ParseS();
31.          ParseT'();
32.        }
33.    else if (lookahead == ')')
34.       {
35.       }
36.    else {
37.        printf("syntax error \n");
38.        exit(0);
39.      }
40. }
41. // 解析非终结符 S→a|Λ|(T)
42. void ParseS()
43. {
44.    switch (lookahead)
45.    {
46.    case 'a':
47.        MatchToken('a');;
48.        break;
49.    case 'Λ':
50.        MatchToken('Λ');;
51.        break;
52.    case '(':
53.        MatchToken('(');
54.        ParseT();
55.        MatchToken(')');
56.       break;
57. default:
58.       printf("syntax error \n");
59.        exit(0);
60.        }
```

(3) 经改写后的文法是否是 LL(1) 的？给出它的预测分析表。

计算文法的每个非终结符的*FIRST*集和*FOLLOW*集

$S \to a \mid \wedge \mid (T)$

$T \to ST'$

$T' \to ,ST' \mid \varepsilon$

非终结符	*FIRST*集	*FOLLOW*集

S	$\{a, \wedge, (\}$	$\{\#, , ,)\}$
T	$\{a, \wedge, (\}$	$\{)\}$
T'	$\{, , \varepsilon\}$	$\{)\}$

证明文法是$LL(1)$的

因为

$SELECT(S \to a) = \{a\} \cap SELECT(S \to \wedge) = \{\wedge\} \cap SELECT(S \to (T)) = \{(\} = \emptyset$

$SELECT(T' \to ,ST') = \{,\} \cap SELECT(T' \to \varepsilon) = \emptyset$

所以该文法是$LL(1)$文法

(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。

构造它的预测分析表

	a	$\wedge$	(	)	,	#
S	$\to a$	$\to \wedge$	$\to (T)$			
T	$\to ST'$	$\to ST'$	$\to ST'$			
T'				$\to \varepsilon$	$\to ,ST'$	

步骤	分析栈	输入缓冲区	选择的产生式或匹配
1	#S	(a,a)#	$S \to (T)$
2	#)T(	(a,a)#	匹配 (
3	#)T	a,a)#	$T \to ST'$
4	#)T'S	a,a)#	$S \to a$
5	#) T'a	a,a)#	匹配 a
6	#) T'	,a)#	$T' \to ,ST'$
7	#) T'S,	,a)#	匹配 ,
8	#) T'S	a)#	$S \to a$
9	#) T'a	a)#	匹配 a
10	#) T’	)#	$T' \to \varepsilon$
11	#)	)#	匹配)
12	#	#	成功

2. 对下面的文法G:

$E \to TE'$

$E' \to +E|\varepsilon$

$T \to FT'$

$T' \to T|\varepsilon$

$F \to PF'$

$F' \to * F'|\varepsilon$

$P \to (E)|a|b|\wedge$

(1) 计算这个文法的每个非终结符的$FIRST$集和$FOLLOW$集

(2) 证明这个文法是$LL(1)$的

(3) 构造它的预测分析表

(4) 构造它的递归下降分析程序

答:

(1) 计算这个文法的每个非终结符的$FIRST$集和$FOLLOW$集

非终结符	$FIRST$集	$FOLLOW$集
E	$\{(, a, b, \wedge\}$	$\{\#,)\}$
E'	$\{+, \varepsilon\}$	$\{\#,)\}$
T	$\{(, a, b, \wedge\}$	$\{+,), \#\}$
T'	$\{(, a, b, \wedge, \varepsilon\}$	$\{+,), \#\}$
F	$\{(, a, b, \wedge\}$	$\{(, a, b, \wedge, +,), \#\}$
F'	$\{*, \varepsilon\}$	$\{(, a, b, \wedge, +,), \#\}$
P	$\{(, a, b, \wedge\}$	$\{*, (, a, b, \wedge, +,), \#\}$

(2)证明这个文法是$LL(1)$的

因为

$SELECT(E' \to +E) = FIRST(+E) = \{+\}$

$SELECT(E' \to \varepsilon) = FOLLOW(E') = \{\#,)\}$

$\underline{SELECT(E' \to +E) \cap SELECT(E' \to \varepsilon) = \emptyset}$

$SELECT(T' \to T) = FIRST(T) = \{(, a, b, \wedge\}$

$SELECT(T' \to \varepsilon) = FOLLOW(T') = \{+,), \#\}$

$\underline{SELECT(T' \to T) \cap SELECT(T' \to \varepsilon) = \emptyset}$

$SELECT(F' \to * F') = FIRST(* F') = \{*\}$

$SELECT(F' \to \varepsilon) = FOLLOW(F') = \{(, a, b, \wedge, +,), \#\}$

$\underline{SELECT(F' \to * F') \cap SELECT(F' \to \varepsilon) = \emptyset}$

$\underline{SELECT(P \to (E)) \cap SELECT(P \to a) \cap SELECT(P \to b) \cap SELECT(P \to \wedge) = \emptyset}$

所以该文法是$LL(1)$文法

(3)构造它的预测分析表

	+	*	(	)	a	b	$\wedge$	#
E			$\to TE'$		$\to TE'$	$\to TE'$	$\to TE'$	
E'	$\to +E$			$\to \varepsilon$				$\to \varepsilon$
T			$\to FT'$		$\to FT'$	$\to FT'$	$\to FT'$	
T'	$\to \varepsilon$		$\to T$	$\to \varepsilon$	$\to T$	$\to T$	$\to T$	$\to \varepsilon$
F			$\to PF'$		$\to PF'$	$\to PF'$	$\to PF'$	
F'	$\to \varepsilon$	$\to * F'$	$\to \varepsilon$	$\to \varepsilon$	$\to \varepsilon$	$\to \varepsilon$	$\to \varepsilon$	$\to \varepsilon$
P			$\to (E)$		$\to a$	$\to b$	$\to \wedge$	

(4)构造它的递归下降分析程序

递归下降分析程序

```
1.  void MatchToken(char expected)
2.  {
3.     if (lookahead != expected)
4.      {
5.          printf("syntax error \n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
```

```
12. }
13. // 解析非终结符 E
14. void ParseE()
15. {
16.     if (lookahead == '(' || lookahead == 'a' || lookahead == 'b' ||
    lookahead == '∧')
17.     {
18.         ParseT();
19.         ParseE’();
20.     }
21.     else
22.     {
23.         printf("syntax error \n");
24.         exit(0);
25.     }
26. }
27. // 解析非终结符 E’
28. void ParseE’()
29. {
30.     if (lookahead == '+')
31.     {
32.         MatchToken(')');
33.         ParseE();
34.     }
35.     else if (lookahead == ')' || lookahead == '#')
36.     {
37.     }
38.     else
39.     {
40.         printf("syntax error \n");
41.         exit(0);
42.     }
43. }
44. // 解析非终结符 T
45. void ParseT()
46. {
47.     switch (lookahead)
48.     {
49.     case '(', 'a', 'b', ' ∧':
50.         ParseF();
51.         ParseT’();
52.         break;
53.     default:
54.         printf("syntax error \n");
```

```
55.         exit(0);
56.     }
57. }
58. // 解析非终结符 T’
59. void ParseT’()
60. {
61.     switch (lookahead)
62.     {
63.     case '(', 'a', 'b', ' ∧':
64.         ParseT();
65.         break;
66.     case '+', ' )', '#':
67.         break;
68.     default:
69.         printf("syntax error \n");
70.         exit(0);
71.     }
72. }
73. // 解析非终结符 F
74. void ParseF()
75. {
76.     switch (lookahead)
77.     {
78.     case '(', 'a', 'b', ' ∧':
79.         ParseP();
80.         break;
81.     default:
82.         printf("syntax error \n");
83.         exit(0);
84.     }
85. }
86. // 解析非终结符 F’
87. void ParseF’()
88. {
89.     switch (lookahead)
90.     {
91.     case '*':
92.         MatchToken('*');
93.         ParseF’();
94.         break;
95.     case '+', '(', ')', 'a', 'b', ' ∧', '#':
96.         break;
97.     default:
98.         printf("syntax error \n");
```

```
99.         exit(0);
100.    }
101. }
102. // 解析非终结符 P
103. void ParseP()
104. {
105.    switch (lookahead)
106.    {
107.    case '(':
108.        MatchToken('(');
109.        ParseE();
110.        MatchToken(')');
111.        break;
112.    case 'a':
113.        MatchToken('a');
114.        break;
115.    case 'b':
116.        MatchToken('b');
117.        break;
118.    case '^':
119.        MatchToken('^');
120.        break;
121.    default:
122.        printf("syntax error \n");
123.        exit(0);
124.    }
125. }
```

3. 已知文法$G[S]$：

$S \rightarrow MH|a$

$H \rightarrow LSo|\varepsilon$

$K \rightarrow dML|\varepsilon$

$L \rightarrow eHf$

$M \rightarrow K|bLM$

判断G是否是$LL(1)$文法，如果是，构造$LL(1)$分析表。

答：非终结符对应的$FIRST$集和$FOLLOW$集

非终结符	$FIRST$集	$FOLLOW$集
S	$\{a,b,d,e,\varepsilon\}$	$\{\#,o\}$
M	$\{b,d,\varepsilon\}$	$\{\#,e,o\}$
H	$\{\varepsilon,e\}$	$\{\#,f,o\}$
L	$\{e\}$	$\{\#,a,b,d,e,o\}$
K	$\{d,\varepsilon\}$	$\{e,\#,o\}$

因为

$SELECT(S \to MH) = (FIRST(MH) - \{\varepsilon\}) \cup FOLLOW(S) = \{d, b, e, \#, o\}$

$SELECT(S \to a) = \{a\}$

$\underline{SELECT(S \to MH) \cap SELECT(S \to a) = \emptyset}$

$SELECT(H \to LSo) = \{e\}$

$SELECT(H \to \varepsilon) = FOLLOW(H) = \{\#, f, o\}$

$\underline{SELECT(H \to LSo) \cap SELECT(H \to \varepsilon) = \emptyset}$

$SELECT(K \to dML) = \{d\}$

$SELECT(K \to \varepsilon) = FOLLOW(K) = \{e, \#, o\}$

$\underline{SELECT(K \to dML) \cap SELECT(K \to \varepsilon) = \emptyset}$

$SELECT(M \to K) = (FIRST(K) - \{\varepsilon\}) \cup FOLLOW(M) = \{\#, d, f, o\}$

$SELECT(M \to bLM) = FIRST(bLM) = \{b\}$

$\underline{SELECT(M \to K) \cap SELECT(M \to bLM) = \emptyset}$

所以文法是$LL(1)$文法

对应预测分析表如下：

	a	o	d	e	f	b	#
S	$\to a$	$\to MH$	$\to MH$	$\to MH$		$\to MH$	$\to MH$
M		$\to K$	$\to K$	$\to K$		$\to bLM$	$\to K$
H		$\to \varepsilon$		$\to LSo$	$\to \varepsilon$		$\to \varepsilon$
L				$\to eHf$			
K		$\to \varepsilon$	$\to dML$	$\to \varepsilon$			$\to \varepsilon$

4. 证明下述文法不是 LL(1)的

$S \to C\$$

$C \to bA|aB$

$A \to a|aC|bAA$

$B \to b|bC|aBB$

能否构造一等价的文法，使其是 LL(1)的？并给出判断过程。

答：

因为$A \to a|aC|bAA$ 与$B \to b|bC|aBB$拥有左公共因子，所以该文法不是$LL(1)$文法。

因此，对这两个产生式 提取左公共因子，构造等价的文法如下：

$S \to C\$$

$C \to bA|aB$

$A \to aD|bAA$

$B \to bD|aBB$

$D \to C|\varepsilon$

证明该文法是 LL(1)文法

首先给出该文法中非终结符的 FIRST 集与 FOLLOW 集

非终结符	FIRST 集	FOLLOW 集
S	$\{a, b\}$	{#}
C	$\{a, b\}$	$\{a, b, \$\}$
A	$\{a, b\}$	$\{a, b, \$\}$
B	$\{b, a\}$	$\{a, b, \$\}$
D	$\{a, b, \varepsilon\}$	$\{a, b, \$\}$

因为

$SELECT(C \to bA) = FIRST(bA) = \{b\}$

$SELECT(C \to aB) = FIRST(aB) = \{a\}$

$\underline{SELECT(C \to bA) \cap SELECT(C \to aB) = \emptyset}$

$SELECT(A \to aD) = FIRST(aD) = \{a\}$

$SELECT(A \to bAA) = FIRST(bAA) = \{b\}$

$\underline{SELECT(A \to aD) \cap SELECT(A \to bAA) = \emptyset}$

$SELECT(B \to bD) = FIRST(bD) = \{b\}$

$SELECT(B \to aBB) = FIRST(aBB) = \{a\}$

$\underline{SELECT(B \to bD) \cap SELECT(B \to aBB) = \emptyset}$

$SELECT(D \to C) = FIRST(C) = \{a, b\}$

$SELECT(D \to \varepsilon) = FOLLOW(D) = \{a, b, \#\}$

$\underline{SELECT(D \to C) \cap SELECT(D \to \varepsilon) \neq \emptyset}$

所以，无法构造一个等价文法使其是$LL(1)$文法。

5. 文法 G 如下：

<程序>→begin<语句表> end

<语句表>→<语句>|<语句表>；<语句>

<语句>→<无条件语句>|<条件语句>

<无条件语句>→*a*

<条件语句>→<如果语句>|<如果语句>else<语句>

<如果语句>→<如果子句><无条件语句>

<如果子句>→ if *b* then

试将 G 改写为 LL(1) 文法，并构造其预测分析表，判断改写后的文法是否为 LL(1) 文法。

答：

将文法符号化：<程序>P，<语句表>L，<语句>S，<无条件语句>U，<条件语句>F，<如果语句>C，<如果子句>I

P→begin L end

L→S|L;S

S→U|F

U→*a*

F→C|C else S

C→I U

I→ if *b* then

对该文法消除左递归、提取左因子：

P→begin L end

L→SL'

L'→;SL'| ε

S→U|F

U→*a*

F→CF'

F'→ ε | else S

C→I U

I→ if *b* then

文法产生式	First	Follow	Select
P→begin L end	begin	#	begin
L→SL'	a, if	end	a, if
L'→;SL'	;	end	;
ε	ε		end
S→U	a	; end	a
F	if		if
U→*a*	a		a
F→CF'	if	; end	if
F'→ ε	ε	; end	; end
else S	else		else
C→I U	if	else ; end	if
I→ if *b* then	if	a	if

Select（L'→;SL'）∩ Select（L'→ ε）=∅

Select（F'→ else S）∩ Select（F'→ ε）=∅

所以该文法是 LL（1）文法。

	begin	end	if	then	else	a	;	#
P	begin L end							
L			SL'			SL'		
L'		ε					;SL'	
S			F			U		
U						a		
F			CF'					
F'		ε			else S		ε	
C			I U					
I			if *b* then					

6. 判断下面哪些文法是 LL(1)的，哪些能改写为 LL(1)文法，并对每个 LL(1)文法设计相应的递归下降识别器。

(2)

$S \rightarrow AB$

$A \rightarrow Ba|\varepsilon$

$B \rightarrow Db|D$

$D \to d|\varepsilon$

答：在该文法中产生式$S \to AB$与产生式$B \to Db|D$存在左公共因子，所以该文法不是$LL(1)$文法。现提取左公共因子构造等价文法：

$S \to BS'$

$S' \to aB|\varepsilon$

$B \to DB'$

$B' \to b|\varepsilon$

$D \to d|\varepsilon$

分析表如下：

	a	b	d	#
S	$\to BS'$	$\to BS'$	$\to BS'$	$\to BS'$
S'	$\to aB$			$\to \varepsilon$
B	$\to DB'$	$\to DB'$	$\to DB'$	$\to DB'$
B'	$\to \varepsilon$	$\to b$		$\to \varepsilon$
D	$\to \varepsilon$	$\to \varepsilon$	$\to d$	$\to \varepsilon$

构造它的递归下降分析程序：

递归下降分析程序

```
1.  void MatchToken(char expected)
2.  {
3.      if (lookahead != expected)
4.      {
5.          printf("syntax error\n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
12. }
13. // 解析非终结符 S
14. void ParseS()
15. {
16.     if (lookahead == 'd' || lookahead == 'b' || lookahead == 'a' || lookah
    ead == '#')
17.     {
18.         ParseB();
19.         ParseS'();
20.     }
21.     else
22.     {
23.         printf("syntax error\n");
24.         exit(0);
```

```
    }
}

// 解析非终结符 S’
void ParseS’()
{
    switch (lookahead)
    {
    case 'a':
        MatchToken('a');
        break;
    case '#':
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
// 解析非终结符 B
void ParseB()
{
    switch (lookahead)
    {
    case 'a', 'd', 'b', '#':
        ParseD();
        ParseB’();
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
// 解析非终结符 B’
void ParseB’()
{
    switch (lookahead)
    {
    case 'b':
        MatchToken('b');
        break;
    case '#', 'a':
        break;
    default:
        printf("syntax error\n");
```

```
        exit(0);
    }
}
// 解析非终结符 D
void ParseD()
{
    switch (lookahead)
    {
    case 'd':
        MatchToken('d');
        break;
    case 'b', 'a', '#':
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
```

(4)

$S \to i|(E)$

$E \to E+S|E-S|S$

答：在该文法中存在直接左递归，所以该文法不是$LL(1)$文法。现消除左递归构造等价文法：

$S \to i|(E)$

$E \to SE'$

$E' \to +SE'|-SE'|\varepsilon$

分析表如下：

	+	−	i	(	)	#
S			$\to i$	$\to (E)$		
E			$\to SE'$	$\to SE'$		$\to \varepsilon$
E'	$+SE'$	$-SE'$			$\to \varepsilon$	

构造它的递归下降分析程序：

递归下降分析程序

```
void MatchToken(char expected)
{
    if (lookahead != expected)
    {
```

```
5.          printf("syntax error\n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
12. }
13. // 解析非终结符 S
14. void ParseS()
15. {
16.     switch (lookahead)
17.     {
18.     case 'i':
19.         MatchToken('i');
20.         break;
21.     case '(':
22.         MatchToken('(');
23.         ParseE();
24.         MatchToken(')');
25.         break;
26.     default:
27.         printf("syntax error\n");
28.         exit(0);
29.     }
30. }
31. // 解析非终结符 E
32. void ParseE()
33. {
34.     switch (lookahead)
35.     {
36.     case 'i', '(':
37.         ParseS();
38.         ParseE’();
39.         break;
40.     default:
41.         printf("syntax error\n");
42.         exit(0);
43.     }
44. }
45. // 解析非终结符 E’
46. void ParseE’()
47. {
48.     switch (lookahead)
```

```
    {
    case '+':
        MatchToken('+');
        ParseS();
        ParseE’();
        break;
    case '-':
        MatchToken('-');
        ParseS();
        ParseE’();
        break;
    case ')':
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
```

(6)

$M \to MaH|H$

$H \to b(M)|(M)|b$

答：在该文法中产生式$M \to MaH$存在直接左递归，产生式$H \to b(M)|(M)|b$存在左公共因子，所以该文法不是$LL(1)$文法。现消除左递归构造等价文法：

$M \to HM'$

$M' \to aHM'|\varepsilon$

$H \to bH'|(M)$

$H' \to (M)|\varepsilon$

分析表如下：

	a	b	(	)	#
M		$\to HM'$	$\to HM'$		
M'	$\to aHM'$			$\to \varepsilon$	$\to \varepsilon$
H		$\to bH'$	$\to (M)$		
H'	$\to \varepsilon$		$\to (M)$	$\to \varepsilon$	$\to \varepsilon$

构造它的递归下降分析程序：

递归下降分析程序

```
void MatchToken(char expected)
{
    if (lookahead != expected)
    {
        printf("syntax error\n");
        exit(0);
    }
    else
    {
        lookahead = getToken();
    }
}
// 解析非终结符 M
void ParseM()
{
    switch (lookahead)
    {
    case 'b', '(':
        ParseH();
        ParseM’();
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
// 解析非终结符 M’
void ParseM’()
{
    switch (lookahead)
    {
    case 'a':
        MatchToken('a');
        ParseH();
        ParseM’();
        break;
    case ')','#':
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
// 解析非终结符 H
```

```
void ParseH()
{
    switch (lookahead)
    {
        case 'b':
             MatchToken('b');
             ParseH’();
             break;
        case '(':
            MatchToken('(');
             ParseM();
             MatchToken(')');
             break;
        default:
             printf("syntax error\n");
             exit(0);
        }
}
// 解析非终结符 H’
void ParseH’()
{
    switch (lookahead)
    {
    case '(':
        MatchToken('(');
        ParseM();
        MatchToken(')');
        break;
    case ')', 'a', '#':
        break;
    default:
        printf("syntax error\n");
        exit(0);
    }
}
```

(1) S→ A | B

A→ aA | a

B→ bB | b

文法中存在左公因子 A→ aA | a，所以不是 LL（1）文法。

对文法进行消除左公因子：

S→ A | B

A→ aA’
A’ → A | ε
B→ bB’
B’ → B | ε

分析表如下：

	a	b	#
S	$\rightarrow A$	$\rightarrow B$	
A	$\rightarrow aA'$		
A'	$\rightarrow A$		$\rightarrow \varepsilon$
B	′	$\rightarrow bB$	
B'		$\rightarrow B$	$\rightarrow \varepsilon$

构造它的递归下降分析程序：

递归下降分析程序

```
1.  void MatchToken(char expected)
2.  {
3.      if (lookahead != expected)
4.      {
5.          printf("syntax error\n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
12. }
13. // 解析非终结符 S
14. void ParseS()
15. {
16.     switch (lookahead)
17.     {
18.     case 'a':  ParseA(); break;
19.     case 'b': ParseB(); break;
20.     default:   printf("syntax error\n");  exit(0);
21.        }
22. }
23. // 解析非终结符 A
24. void ParseA()
25. {
26.       if (lookahead == 'a')
27.         { MatchToken('a');
28.           ParseA’();
```

```
        }
       else
        {   printf("syntax error\n");
            exit(0);
        }
}
// 解析非终结符 B
void ParseB()
{
      if (lookahead == 'b')
        { MatchToken('b');
          ParseB();
        }
       else
        {     printf("syntax error\n");
            exit(0);
        }
}
void ParseA'()  // 解析非终结符 A'
{
    switch (lookahead)
    {
        case 'a':   ParseA();   break;
        case '#':    break;
        default:
            printf("syntax error\n");
            exit(0);
        }
}
// 解析非终结符 B'
void ParseB'()
{
    switch (lookahead)
    {
    case 'b':   ParseB();   break;
    case '#':    break;
      default:
         printf("syntax error\n");
         exit(0);
   }
}
```

（3）S→ aAaB | bAbB

A→ S | db

B→ bB | a

是 LL（1）文法

分析表如下：

	a	b	d	#
S	$\rightarrow aAaB$	$\rightarrow bAbB$		
A	$\rightarrow S$	$\rightarrow S$	$\rightarrow db$	
B	$\rightarrow a$	$\rightarrow bB$		

构造它的递归下降分析程序：

递归下降分析程序

```
1.  void MatchToken(char expected)
2.  {
3.      if (lookahead != expected)
4.      {
5.          printf("syntax error\n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
12. }
13. // 解析非终结符 S
14. void ParseS()
15. {
16.     switch (lookahead)
17.     {
18.     case 'a':
19.             // aAaB
20.         MatchToken('a');
21.          ParseA();
22.          MatchToken('a');
23.          ParseB();
24.         break;
25.       Case 'b': // bAbB
26.          MatchToken('b');
27.          ParseA();
28.          MatchToken('b');
29.          ParseB();
30.         break;
```

```
          default:
             printf("syntax error\n");
              exit(0);
    }
 }
// 解析非终结符 A

void ParseA()
{
       switch (lookahead )
        case 'a':   ParseS();break;
        case 'b':   ParseS(); break;
        case 'd':   MatchToken('d'); MatchToken('b'); break;
       default:
        {   printf("syntax error\n");
            exit(0);
        }
}
// 解析非终结符 B
void ParseB()
{
       switch (lookahead )
        {
          case 'a':   MatchToken('a');break;
          case 'b':   MatchToken('b'); ParseB(); break;
         default:
        {     printf("syntax error\n");
            exit(0);
        }
}
```

（5）S→ SaA | bB

A→ aB | c

B→ Bb | d

文法中存在左递归，所以不是 LL（1）文法。消除文法的左递归：

S→ bBS'

S'→ aAS' | ε

A→ aB | c

B→ dB'

B' →bB' | ε

分析表如下：

	a	b	c	d	#
S		$\rightarrow bBS'$			
S'	$\rightarrow aAS'$				$\rightarrow \varepsilon$
A	$\rightarrow aB$		$\rightarrow c$		
B				$\rightarrow dB$	
B'	$\rightarrow \varepsilon$	$\rightarrow bB'$			$\rightarrow \varepsilon$

构造它的递归下降分析程序：

递归下降分析程序

```
1.  void MatchToken(char expected)
2.  {
3.      if (lookahead != expected)
4.      {
5.          printf("syntax error\n");
6.          exit(0);
7.      }
8.      else
9.      {
10.         lookahead = getToken();
11.     }
12. }
13. // 解析非终结符 S
14. void ParseS()
15. {
16.      if (lookahead == 'b')
17.         { MatchToken('b');
18.           ParseB();
19.           ParseS’();  }
20.       else
21.        {     printf("syntax error\n");
22.             exit(0);
23.        }
24. }
25. // 解析非终结符 S’
26. void ParseS’()
27. {
28.      switch (lookahead)
29.   {
30.          case 'a':   MatchToken('a'); ParseA();ParseS’();break;
31.        case '#':     break;
32.         default:
33.             printf("syntax error\n");
34.             exit(0);
35.         }
```

```
}
 // 解析非终结符 A
void ParseA()
{
     switch (lookahead)
  {
         case 'a':   MatchToken('a'); ParseB();break;
         case 'c':    MatchToken('c'); break;
        default:
            printf("syntax error\n");
            exit(0);
        }
}
void ParseB()  // 解析非终结符 B
{
    if (lookahead=='d')
    {
        MatchToken('d');
        ParseB();
       }
       else
            { printf("syntax error\n");
              exit(0);
         }
}
// 解析非终结符 B'
void ParseB'()
{
    switch (lookahead)
    {
    case 'a', '#':    break;
    case 'b':   MatchToken('b'); ParseB'();  break;
    default:
         printf("syntax error\n");
         exit(0);
   }
}
```

7. 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？试对下面的文法进行改写，并对改写后的文法进行判断。

答：不一定

（1）A→ baB | ε
B→ Abb | a

该文法无左递归，无左公因子

Select（A→ baB）={b}

Select（A→ε）=follow(A)={#,b}

两者有交集，因此不是 LL（1）文法。

（2）A→ aABe | a
B→ Bb | d

对该文法消除左递归，提取左公因子：

A→ aA’
A’→ ABe | ε
B→ dB’
B’→ bB’ | ε

Select（A’→ABe）={a}

Select（A’→ε）=follow(A’)={#}

两者交集为空集

Select（B’→ bB’ ）={b}

Select（B’→ε）=follow(B’)={e}

两者交集为空集

该文法是 LL（1）文法。

（3）S→ Aa | b
A→ SB
B→ ab

对该文法消除左递归，提取左公因子：

S→ bS’
S’→ Ba S’ | ε
B→ ab

Select（S’→ BaS’ ）={a}

Select（S’→ε）=follow(S’)={#}

两者交集为空集

该文法是 LL（1）文法。

第五章作业

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：

（1）计算G[S]的FIRSTVT和LASTVT。

	FirstVT	LastVT
S	a ∧ (	a ∧)
T	, a ∧ (	, a ∧)

（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。

求文法中的各个优先关系：

⋖：（⋖

⋗：T, T) S# （LASTVT）

⋖：(T ,S #S （FIRSTVT）

算符优先关系表：

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

优先关系表中无冲突，所以是算符优先文法。

（3）计算G[S]的优先函数。

	,	(	)	a	∧	#
f	5	3	7	7	7	1
g	6	4	3	7	8	1

（4）给出输入串(a,a)#和(a,(a,a))#的算符优先分析过程。

分析(a,(a,a))#

步骤	分析栈	优先关系	余留输入串	最左素短语	产生式
1	#	<	(a,(a,a))#		
2	#(	<	a,(a,a))#		
3	#(a	>	,(a,a))#	a	S→a
4	#(S	<	,(a,a))#		
5	#(S,	<	(a,a))#		
6	#(S,(	<	a,a))#		
7	#(S,(a	>	,a))#	a	S→a
8	#(S,(S	<	,a))#		
9	#(S,(S,	<	a))#		
10	#(S,(S,a	>	))#	a	S→a
11	#(S,(S,S	>	))#	S,S	T→T,S
12	#(S,(T	=	))#		H→(S)
13	#(S,(T)	>	)#	(T)	S→(T)
14	#(S,S	>	)#	S,S	T→T,S
15	#(T	=	)#		
16	#(T)	>	#	(T)	S→(T)
17	#S		#	分析成功	

分析(a,a)#

步骤	分析栈	优先关系	余留输入串	最左素短语	产生式
1	#	<	(a,a)#		
2	#(	<	a,a)#		
3	#(a	>	,a)#	a	S→a
4	#(S	<	,a)#		
5	#(S,	<	a)#		
6	#(S,a	>	)#	a	S→a
7	#(S,S	>	)#	S,S	T→T,S
8	#(T	=	)#		
9	#(T)	>	#	(T)	S→(T)
10	#S		#	分析成功	

4. 已知文法 G[S]为：

$$S \to S;G|G$$
$$G \to G(T)|H$$
$$H \to a|(S)$$
$$T \to T+S|S$$

（1）构造 G 的算符优先关系表，并判定 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、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，分别说明两个输入串是否为 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规约过程都是最右推导的逆过程吗？

答：

（1）构造 G 的算符优先关系表，并判定 G[S]是否为算符优先文法。

解答：

求文法各个非终结符的 FirstVT 集合和 LastVT 集合：

	FirstVT	LastVT
S	; (a	;) a
G	(a	) a
H	(a	) a
T	; (a +	;) a +

求文法中的各个优先关系：

: (⊖)

⊘: S; G(T) S) T+ S#

⊗: ;G (T (S +S #S

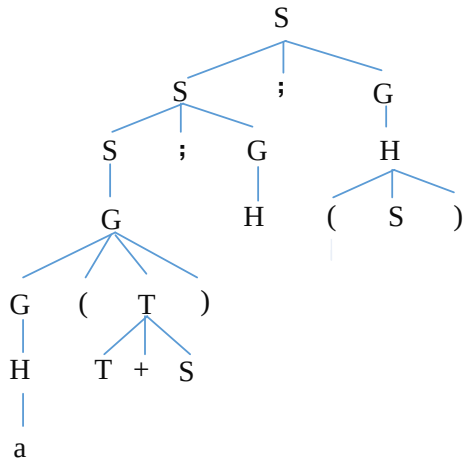
构造优先关系表如下：

	;	(	)	a	+	#
;	⊘	⊗	⊘	⊗	⊘	⊘
(	⊗	⊗	⊖	⊗	⊗	
)	⊘	⊘	⊘		⊘	⊘
a	⊘	⊘	⊘		⊘	⊘
+	⊗	⊗	⊘	⊗	⊘	
#	⊗	⊗		⊗		

（2）给出句型 a(T+S);H;(S)的短语、句柄、素短语、最左素短语

短语：a，T+S，　a(T+S)，　H，　(S)，　a(T+S);H，　a(T+S);H;(S)（7 个）
句柄：a
素短语：a，T+S，　(S)　　（3 个）
最左素短语：a

语法树如下：



（3）分析 a;(a+a)

步骤	分析栈	优先关系	余留输入串	最左素短语	产生式
1	#	<	a;(a+a)#		
2	#a	>	;(a+a)#	a	H→a
3	#H	<	;(a+a)#		
4	#H;	<	(a+a)#		
5	#H;(	<	a+a)#		
6	# H;(a	>	+a)#	a	H→a
7	# H;(H	<	+a)#		
8	# H;(H+	<	a)#		
9	# H;(H+a	>	)#	a	H→a
10	# H;(H+H	>	)#	H+H	T→T+S
11	# H;(T	=	)#		
12	# H;(T)	>	#	(T)	H→(S)
13	# H;H	>	#	H;H	S→S;G
14	#S		#	分析成功	

分析(a+a)

步骤	分析栈	优先关系	余留输入串	最左素短语	产生式
1	#	<	(a+a)#		
2	#(	>	a+a)#	a	H→a

3	#(a	<	+a)#		
4	#(H	<	+a)#		
5	#(H+(	<	a)#		
6	# (H+a	>	)#	a	H→a
7	# (H+H	<	)#	H+H	T→T+S
8	# (T	<	)#		
9	#(T)	>	#	(T)	H→(S)
10	# H	>	#	分析成功	

(4) a;(a+a)的最右推导

S⇒S;G

⇒ S;H

⇒ S;(S)

⇒ S;(G)

⇒ S;(H)　出错

(a+a)的最右推导

S⇒G

⇒ H

⇒ (S)

⇒ (G)

⇒ (H)　出错

（5）算法优先归约的是最左素短语，不是句柄，因此会隐藏语法错误无法发现。

（6）算法优先分析过程不是最右推导逆过程；

规范归约是最右推导逆过程。

第六章作业

1. 已知文法

$A \to aAd|aAb|\varepsilon$

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

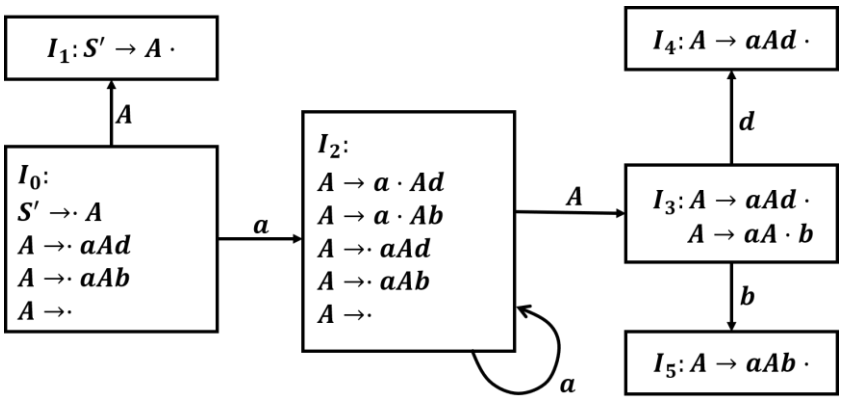
答：拓广文法为G'，增加产生式$S' \to A$，若产生式排序为：

0	$S' \to A$
1	$A \to aAd$
2	$A \to aAb$
3	$A \to \varepsilon$

由产生式知：

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

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：$A \to \cdot aAd$和$A \to \cdot aAb$为移进项目，$A \to \cdot$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：$Follow(A) \cap \{a\} = \{d, b, \#\} \cap \{a\} = \emptyset$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：

状态	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		$r2$	$r2$	$r2$	

对输入串$ab\#$的分析过程

步骤	状态栈	符号栈	输入串	分析动作	下一状态
(1)	0	#	$ab\#$	$S2$	
(2)	02	$\#a$	$b\#$	$r3$	3

(3)	023	#aA	b#	S5	
(4)	0235	#aAb	#	r2	1
(5)	01	#A	#	acc	

分析成功，说明输入串ab是文法的句子。

2.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。

说明：为了便于画图，用'+'代替'.'。

答：拓广文法并对产生式进行编号：

0: S'→S

1: S→L+L

2: S→L

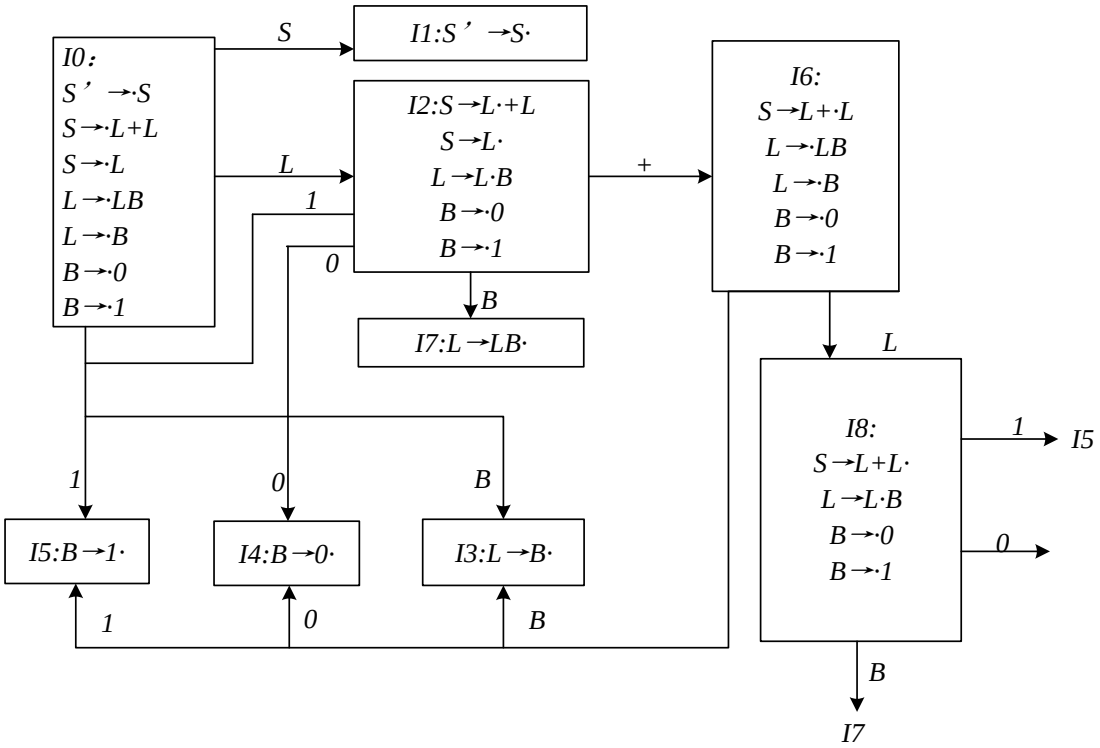
3: L→LB

4: L→B

5: B→0

6: B→1

$G[S']$的$LR(0)$项目集族及识别活前缀的DFA如下图所示：



由产生式知：

$Follow(S) = \{\#\}$

$Follow(L) = \{0, 1, +, \#\}$

$Follow(B) = \{0, 1, +, \#\}$

在I_2中：$S \to L \cdot +L$、$B \to \cdot 0$、$B \to \cdot 1$为移进项目，$S \to L \cdot$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中：$Follow(S) \cap \{+\} \cap \{0\} \cap \{1\} = \emptyset$，所以在$I_2$中的移进-归约冲突可以由$Follow$集解决。

在I_8中：$Follow(S) \cap \{0,1\} = \{\#\} \cap \{0,1\} = \emptyset$所以在$I_8$中的移进-归约冲突可以由$Follow$集解决

综上，该文法是$SLR(1)$文法。

构造的$SLR(1)$分析表如下：

状态	ACTION				GOTO		
	0	1	+	#	S	L	B
0	$S4$	$S5$			1	2	3
1				acc			
2	$S4$	$S5$	$S6$	$r2$			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

对输入串 101.110（101+110）的分析过程

步骤	状态栈	符号栈	输入串	~~ACTION~~ 分析动作	~~GOTO~~ 下一状态
1	0	#	101 + 110#	$S5$	5
2	05	#1	01 + 110#	$r6$	GOTO[0][B]=3
3	03	#B	01 + 110#	$r4$	GOTO[0][L]=2
4	02	#L	01 + 110#	$S4$	4
5	024	#L0	1 + 110#	r5	GOTO[2][B]=7
6	027	#LB	1 + 110#	$r3$	GOTO[0][L]=2
7	02	#L	1 + 110#	$S5$	5
8	025	#L1	+110#	$r6$	GOTO[2][B]=7
9	027	#LB	+110#	$r3$	GOTO[0][L]=2
10	02	#L	+110#	$S6$	6
11	026	#L+	110#	$S5$	5
12	0265	#L+1	10#	$r6$	GOTO[6][B]=3
13	0263	#L + B	10#	$r4$	GOTO[6][L]=8
14	0268	#L+L	10#	$S5$	5
15	02685	#L+L1	0#	$r6$	GOTO[8][B]=7
16	02687	#L+LB	0#	$r3$	GOTO[6][L]=8
17	0268	#L+L	0#	$S4$	4
18	02684	#L+L0	#	r5	GOTO[8][B]=7
19	02687	#L+LB	#	$r3$	GOTO[6][L]=8
20	0268	#L+L	#	$r1$	GOTO[0][S]=1
21	01	#S	#	acc	

分析成功，说明输入串 101.110（101+110）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(l)的吗？若是，构造出它的 SLR 分析表。

(4) 这个文法是 LALR(l)或 LR(1)的吗？

答：

(1) 列出这个文法的所有 LR(0)项目。

0: S' →S

1: S→AS

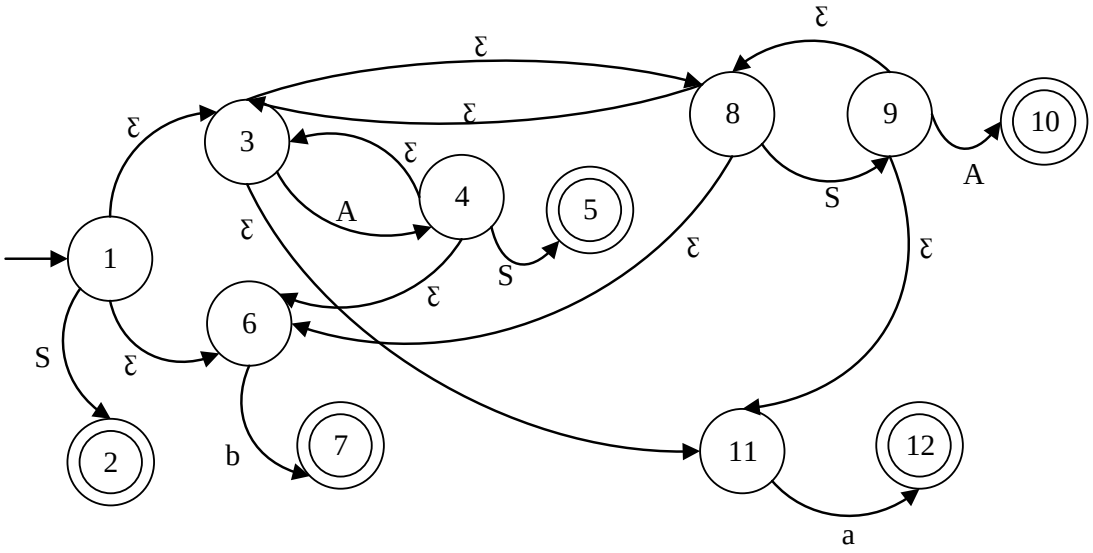
2: S→b

3: A→SA

4: A→a

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

(2)按(1)列出的项目构造识别这个文法活前缀的 NFA，并把这个 NFA 确定化为 DFA：



状态S	$\varepsilon\_closure(S)$
1	{1,3,6,8,11}
2	{2}
3	{3,6,8,11}

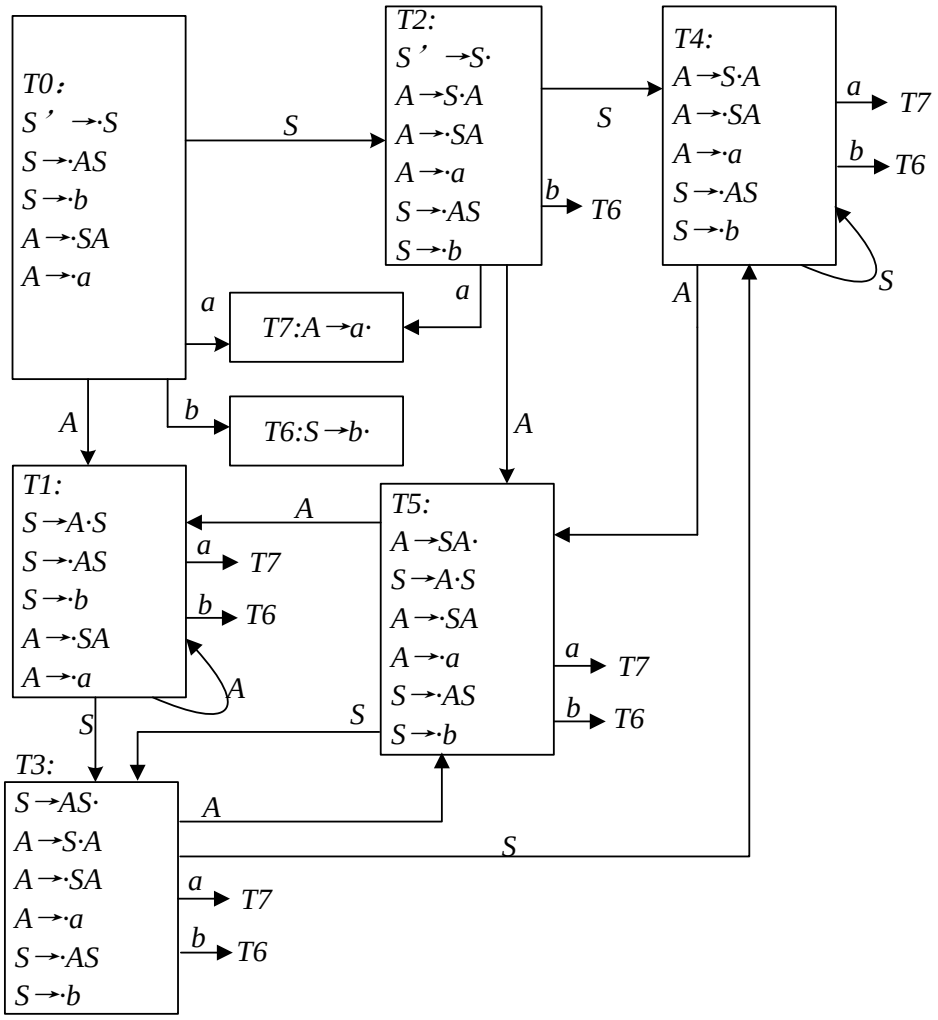
4	{3,4,6,8,11}
5	{5}
6	{6}
7	{7}
8	{3,6,8,11}
9	{3,6,8,9,11}
10	{10}
11	{11}
12	{12}

用有效子集法将此 NFA 确定化的状态转换表如下：

状态集T	符号a	$Move(T,a)$	$\varepsilon\_closure(Move(T,a))$
$T_0=\{1,3,6,8,11\}$	S	{2,9}	{2,3,6,8,9,11} T_2
	A	{4}	{3,4,6,8,11} T_1
	a	{12}	{12} T_7
	b	{7}	{7} T_6
$T_1=\{3,4,6,8,11\}$	S	{5,9}	{3,5,6,8,9,11}T_3
	A	{4}	{3,4,6,8,11} T_1
	a	{12}	{12} T_7
	b	{7}	{7} T_6
$T_2=\{2,3,6,8,9,11\}$	S	{9}	{3,6,8,9,11} T_4
	A	{4,10}	{3,4,6,8,10,11} T_5
	a	{12}	{12} T_7
	b	{7}	{7} T_6
$T_3=\{3,5,6,8,9,11\}$	S	{9}	{3,6,8,9,11} T_4
	A	{4,10}	{3,4,6,8,10,11} T_5
	a	{12}	{12} T_7
	b	{7}	{7} T_6
$T_4=\{3,6,8,9,11\}$	S	{9}	{3,6,8,9,11} T_4
	A	{4,10}	{3,4,6,8,10,11} T_5
	a	{12}	{12} T_7
	b	{7}	{7} T_6
$T_5=\{3,4,6,8,10,11\}$	S	{9}	{3,6,8,9,11} T_4
	A	{4}	{3,4,6,8,11} T_1
	a	{12}	{12} T_7
	b	{7}	{7} T_6
$T_6=\{7\}$	S	∅	∅
	A	∅	∅
	a	∅	∅
	b	∅	∅
$T_7=\{12\}$	S	∅	∅
	A	∅	∅
	a	∅	∅

	b	$\emptyset$	$\emptyset$

把每个子集写出来，得到如下图所示的 DFA



(3) 这个文法不是 SLR(l)的，分析表中有冲突。

由产生式知：

$\boldsymbol{Follow(S)} = \{\#, \boldsymbol{a}\}$

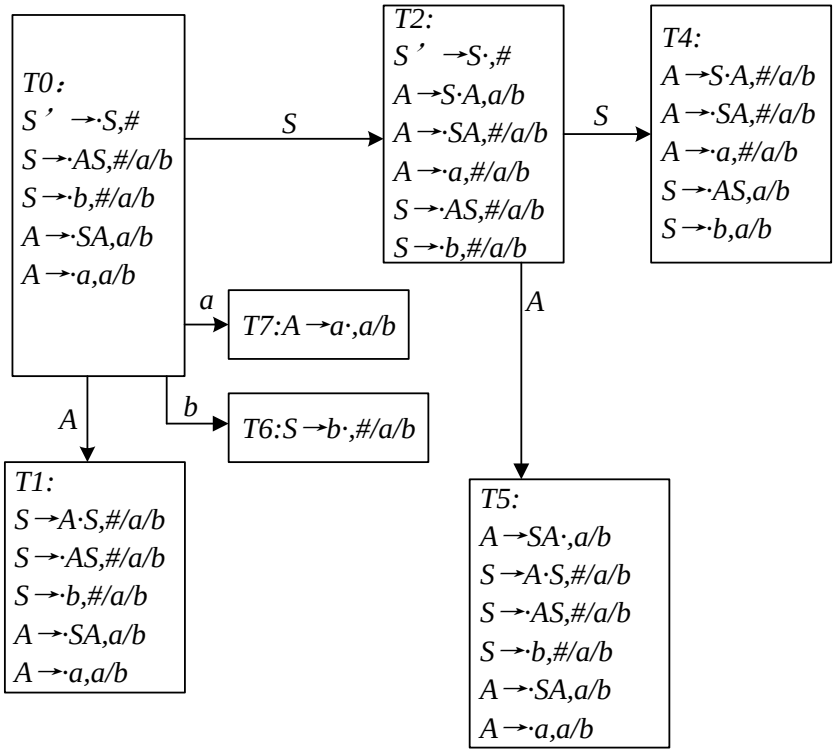
$\boldsymbol{Follow(A)} = \{\boldsymbol{b}\}$

构造的$\boldsymbol{SLR(1)}$分析表如下：

状态	ACTION			GOTO	
	$\boldsymbol{a}$	$\boldsymbol{b}$	#	$\boldsymbol{S}$	$\boldsymbol{A}$
0	$\boldsymbol{S7}$	$\boldsymbol{S6}$		**2**	**1**
1	$\boldsymbol{S7}$	$\boldsymbol{S6}$		**3**	**1**
2	$\boldsymbol{S7}$	$\boldsymbol{S6}$	$\boldsymbol{acc}$	**4**	**5**
3	$\boldsymbol{S7}$ $\boldsymbol{r1}$	$\boldsymbol{S6}$	$\boldsymbol{r1}$	**4**	**5**
4	$\boldsymbol{S7}$	$\boldsymbol{S6}$		**4**	**5**
5	$\boldsymbol{S7}$	$\boldsymbol{S6}$	$\boldsymbol{r3}$	**4**	**1**
6	$\boldsymbol{r2}$		$\boldsymbol{r2}$		**8**
7		$\boldsymbol{r4}$			

这个文法不是 SLR(l)的，分析表中有冲突。

(4) 在构造 LR(1)的项目集规范族时，在项目集 T5 中存在移进-归约冲突，归约符号为{a,b},移进的符号也有 a 和 b，因此冲突无法解决。故文法不是 LR（1）的，也不是 LALR（1）的。



4.下面是一个描述∑={a.b}上的正规式的 LALR(l)文法(实际上也是 SLR(l)文法)，只不过用+代替丨。

E➞E+T | T

T➞TF | F

F➞ F* |(E) | a |b 构造这个文法的 LALR(l)项目集和分析表。

答：文法并对产生式进行编号：

0: E'➞E

1: E➞E+T

2: E➞T

3: T➞TF

4: T➞F

5: F➞F*

6: F➞(E)

7: F➞a

8: F➞b

为便于计算，先计算项目集 I0 如下：

I0:

E'→·E, #

E→·E+T, #/+

E→·T, #/+

T→·TF, #/+/(/a/b

T→·F, #/+/(/a/b

F→·F*,#/+/(/a/b/*

F→·(E), #/+/(/a/b/*

F→·a, #/+/(/a/b/*

F→·b, #/+/(/a/b/*

构造的 LR（1）项目集规范族如下：

	E	T	F	+	*	(	)	a	b
I0:	I1: E'→ E·, # E→E·+T, #/+	I2: E→T·, #/+ T→T·F, #/+/(/a/b F→·F*,#/+/(/a/b/* F→·(E), #/+/(/a/b/* F→·a, #/+/(/a/b/* F→·b, #/+/(/a/b/*	I3: T→F·, #/+/(/a/b F→F·*,#/+/(/a/b/*			I8: F→(·E), #/+/(/a/b/* E→·E+T,)/+ E→·T,)/+ T→·TF,)/+/(/a/b T→·F,)/+/(/a/b F→·F*,)/+/(/a/b/* F→·(E),)/+/(/a/b/* F→·a,)/+/(/a/b/* F→·b,)/+/(/a/b/*		I4: F→a·, #/+/(/a/b/*	I5: F→b·, #/+/(/a/b/*
I1:				I6: E→E+·T, #/+ T→·TF, #/+/(/a/b T→·F, #/+/(/a/b F→·F*,#/+/(/a/b/* F→·(E), #/+/(/a/b/* F→·a, #/+/(/a/b/* F→·b, #/+/(/a/b/*					
I2			I7: T→TF·, #/+/(/a/b F→F·*,#/+/(/a/b/*			I8:		I9: F→a·,)/+/(/a/b/*	I10: F→b·,)/+/(/a/b/*
I3					I11: F→F*·,#/+/(/a/b/*				
I4									
I5									

I6		**I12:** **E→E+T·, #/+** **T→T·F, #/+/(/a/b** **F→·F*,#/+/(/a/b/*** **F→·(E), #/+/(/a/b/*** **F→·a, #/+/(/a/b/*** **F→·b, #/+/(/a/b/***	**I7**			**I8**		**I4:**	**I5:**
I7					**I11:**				
I8	**I13:** **F→(E·),** **#/+/(/a/b/*** **E→E·+T,)/+**	**I14:** **E→T·,)/+** **T→T·F,)/+/(/a/b** **F→·F*,)/+/(/a/b/*** **F→·(E),)/+/(/a/b/*** **F→·a,)/+/(/a/b/*** **F→·b,)/+/(/a/b/***	**I15:** **T→F·,)/+/(/a/b** **F→F·*,)/+/(/a/b/***					**I9:**	**I10:**
I9									
I10									
I11									
I12			**I7**			**I8:**		**I4:**	**I5:**
I13				**I16:** **E→E+·T,)/+** **T→·TF,)/+/(/a/b** **T→·F,)/+/(/a/b** **F→·F*,)/+/(/a/b/*** **F→·(E),)/+/(/a/b/*** **F→·a,)/+/(/a/b/*** **F→·b,)/+/(/a/b/***		**I17;** **F→(E)·, #/+/(/a/b/***		
I14			**I15:**			**I18:** **F→(·E),)/+/(/a/b/*** **E→·E+T,)/+**		**I9:**	**I10:**

						E→·T,)/+/(/a/b **T→·TF,)/+/(/a/b** **T→·F,)/+/(/a/b** **F→·F*,)/+/(/a/b/*** **F→·(E),)/+/(/a/b/*** **F→·a,)/+/(/a/b/*** **F→·b,)/+/(/a/b/***			
I15					**I19:** **F→F*·,)/+/(/a/b/***				
I16		**I20:** **E→E+T·,)/+** **T→T·F,)/+/(/a/b** **F→·F*,)/+/(/a/b/*** **F→·(E),)/+/(/a/b/*** **F→·a,)/+/(/a/b/*** **F→·b,)/+/(/a/b/***	**I15**			**I18**		**I9:**	**I10:**
I17									
I18	**I21:** **F→(E·),)/+/(/a/b/*** **E→E·+T,)/+**	**I14**	**I15**			**I18**		**I9:**	**I10:**
I19									
I20			**I22:** **T→TF·,)/+/(/a/b** **F→F·*,)/+/(/a/b/***			**I18**		**I9:**	**I10:**
I21				**I16:**			**I23:** **F→(E)·,)/+/(/a/b/***		
I22					**I19**				
I23									

构造的 LALR（1）项目集如下：

	E	T	F	+	*	(	)	a	b
I0:	**I1:** **E'→ E·, #** **E→E·+T, #/+**	**I2** **E→T·,)/#/+** **T→T·F,)/#/+/(/a/b** **F→·F*,)/#/+/(/a/b/*** **F→·(E),)/#/+/(/a/b/*** **F→·a,)/#/+/(/a/b/*** **F→·b,)/ #/+/(/a/b/***	**I3** **T→F·,)/#/+/(/a/b** **F→F·*,)/#/+/(/a/b/***			**I8** **F→(·E),)/#/+/(/a/b/*** **E→·E+T,)/+** **E→·T,)/+/(/a/b** **T→·TF,)/+/(/a/b** **T→·F,)/+/(/a/b** **F→·F*,)/+/(/a/b/*** **F→·(E),)/+/(/a/b/*** **F→·a,)/+/(/a/b/*** **F→·b,)/+/(/a/b/***		**I4:** **F→a·,)/#/+/(/a/b/***	**I5:** **F→b·,)/#/+/(/a/b/***
I1:				**I6** **E→E+·T,)/#/+** **T→·TF,)/#/+/(/a/b** **T→·F,)/#/+/(/a/b** **F→·F*,)/#/+/(/a/b/*** **F→·(E),)/#/+/(/a/b/*** **F→·a,)/#/+/(/a/b/*** **F→·b,)/#/+/(/a/b/***				

I2			**I7** **T→TF·,)/#/+/(/a/b** **F→F·*,)/#/+/(/a/b/***			**I8**		**I4:**	**I5:**
I3					**I9** **F→F*·,)/#/+/(/a/b/***				
I4									
I5									
I6		**I10** **E→E+T·,)/#/+** **T→T·F,)/#/+/(/a/b** **F→·F*,)/#/+/(/a/b/*** **F→·(E),)/#/+/(/a/b/*** **F→·a,)/#/+/(/a/b/*** **F→·b,)/#/+/(/a/b/***	**I3**			**I8**		**I4:**	**I5:**
I7					**I9:**				
I8	**I11:** **F→(E·),)/#/+/(/a/b/*** **E→E·+T,)/+**	**I2**	**I3**			**I8**		**I4:**	**I5:**
I9									

I10			**I3**			**I8**		**I4:**	**I5:**
I11							**I12** **F→(E)·,)/#/+/(/a/b/***		
I12									

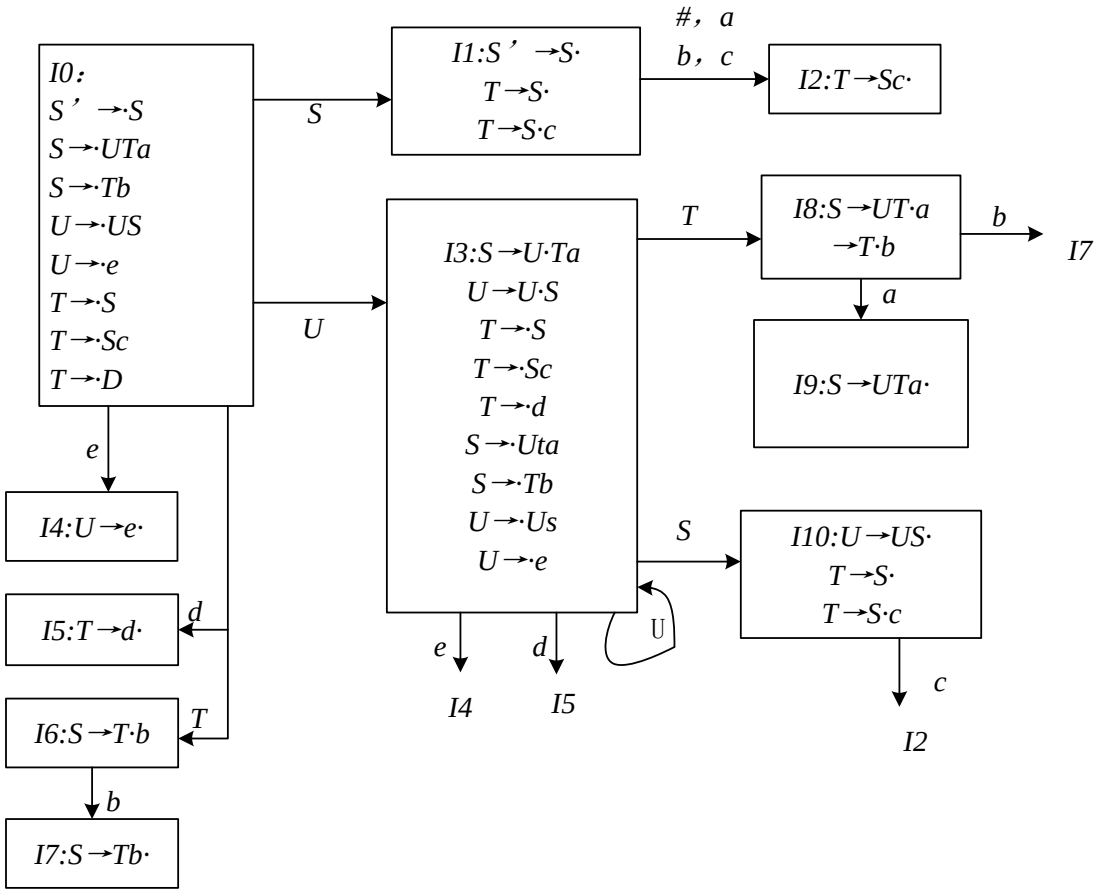
0: E'→E
1: E→E+T
2: E→T
3: T→TF
4: T→F
5: F→F*
6: F→(E)
7: F→a
8: F→b

构造的$LALR(1)$分析表如下：

状态	ACTION							GOTO		
	+	*	(	)	a	b	#	*E*	*T*	F
0			$S8$		$S4$	$S5$		1	2	3
1	$S6$						acc			
2	$r2$		$S8$	$r2$	$S4$	$S5$	$r2$			7
3	$r4$	$S9$	$r4$	$r4$	$r4$	$r4$	$r4$			
4	$r7$	$r7$	$r7$	$r7$	$r7$	$r7$	$r7$			
5	$r8$	$r8$	$r8$	$r8$	$r8$	$r8$	$r8$			
6			$S8$		$S4$	$S5$			10	3
7	$r3$	$S9$	$r3$	$r3$	$r3$	$r3$	$r3$			
8			$S8$		$S4$	$S5$		11	2	3
9	$r5$	$r5$	$r5$	$r5$	$r5$	$r5$	$r5$			
10	$r1$		$S8$	$r1$	$S4$	$S5$	$r1$			3
11				S12						
12	$r6$	$r6$	$r6$	$r6$	$r6$	$r6$	$r6$			

6.文法 G=({U,T,S},{a0,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)的，说明理由。
(2) 构造相应的分析表
答：
拓广文法并对产生式进行编号：
0: S'→S
1: S→UTa
2: S→Tb
3: T→S
4: T→Sc
5: T→d
6: U→US
7: U→e

$G[S']$的$LR(0)$项目集族及识别活前缀的DFA如下图所示：



在I_1中：$S' \to S \cdot$和$T \to S \cdot$为归约项目，$T \to S \cdot c$为移进项目，存在移进-归约冲突，因此所给文法不是$LR(0)$文法。

由产生式知：

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

$Follow(T) = \{a, b\}$

$Follow(U) = \{e, d\}$

在I_1中：$Follow(S') \cap Follow(T) \cap \{c\} = \{\#\} \cap \{a, b\} \cap \{c\} = \emptyset$，所以在$I_1$中的移进-归约冲突可以由$Follow$集解决。

在I_{10}中：$Follow(U) \cap Follow(T) \cap \{c\} = \{e, d\} \cap \{a, b\} \cap \{c\} = \emptyset$,所以在$I_{10}$中的移进-归约冲突可以由$Follow$集解决

综上，该文法是$SLR(1)$文法。

0: S’→S

1: S→UTa

2: S→Tb

3: T→S

4: T→Sc

5: T→d

6: U→US

7: U→e

构造的$SLR(1)$分析表如下：

状态	ACTION						GOTO		
	a	b	c	d	e	#	S	U	T
0				$S5$	$S4$		1	3	6
1	$r3$	$r3$	$S2$			acc			
2	$r4$	$r4$							7
3				$S5$	$S4$	$r4$	10	3	8
4			$r7$	$r7$		$r5$			
5	$r5$	$r5$				$r6$			
6		$S7$						8	3
7	$r2$	$r2$	$r2$	$r2$	$r2$	$r2$			
8	$S9$	$S7$				$r1$			7
9	$r1$	$r1$	$r1$	$r1$	$r1$	$r1$			
10	$r3$	$r3$	$S2$	$r6$	$r6$				

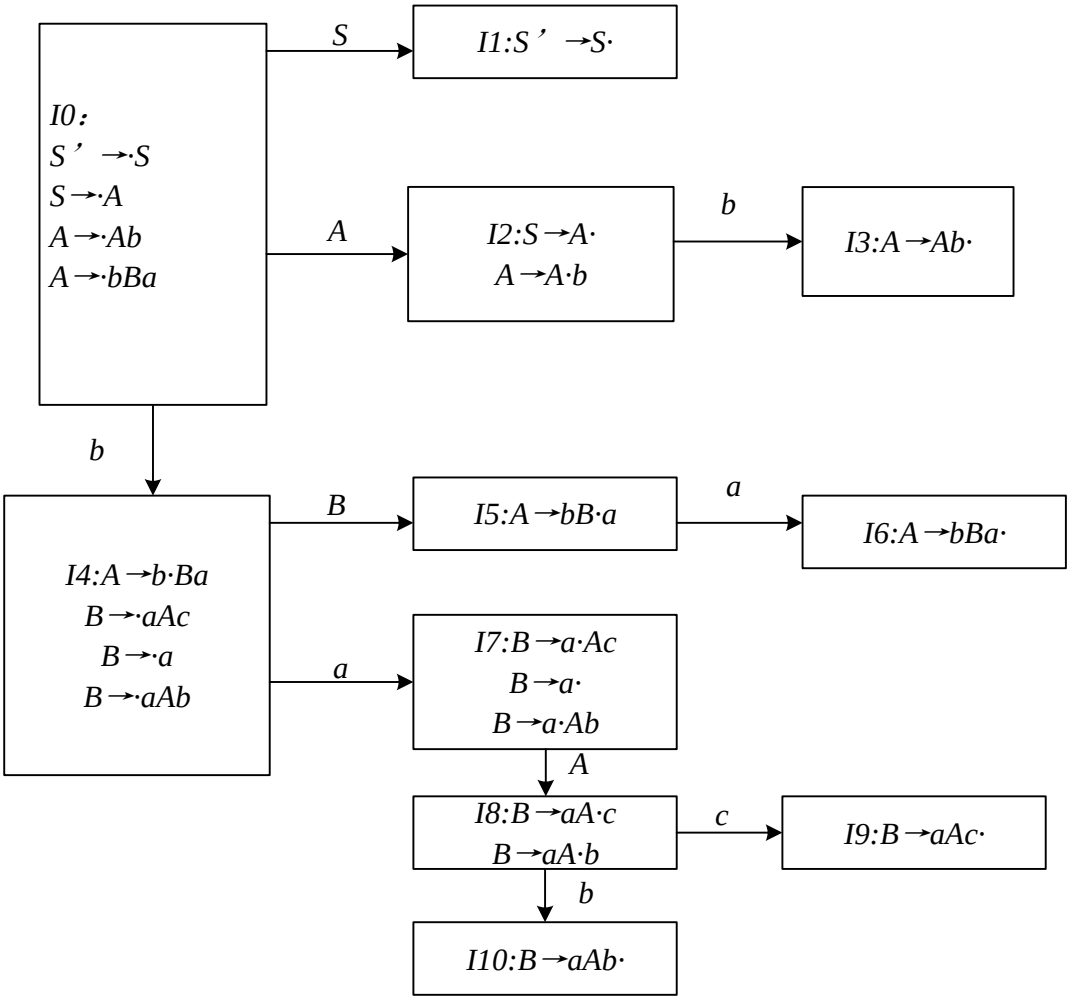
7.证明下面文法不是 LR(0)而是 SLR(1)

S➞A

A➞Ab | bBa

B➞aAc | a | aAb

$G[S']$的$LR(0)$项目集族及识别活前缀的DFA如下图所示：



由产生式知：

$Follow(S) = \{\#\}$

$Follow(A) = \{\#, b, c\}$

$Follow(B) = \{a\}$

在I_2和I_7中存在移进-归约冲突，因此所给文法不是$LR(0)$文法。

在I_2中：$S \to A \cdot$和$A \to A \cdot b$为归约-移进冲突，$Follow(S) \cap \{b\} = \{\#\} \cap \{a, b\} = \emptyset$，所以在$I_2$中的移进-归约冲突可以由$SLR(1)$规则解决。

在I_7中：$B \to a \cdot$为归约项目，在I_{10}中的移进-归约冲突可以由$SLR(1)$规则解决。

综上，该文法是$SLR(1)$文法。

8.证明文法(其中$相当于井)

S➞A$

A➞BaBb | DbDa

B➞ε

D➞ε

是 LR(1)而不是 SLR(1)的。

答：

拓广文法并对产生式进行编号：

0: S'➞S

1:S➞A

2:A➞BaBb

3:A➞DbDa

4:B➞ε

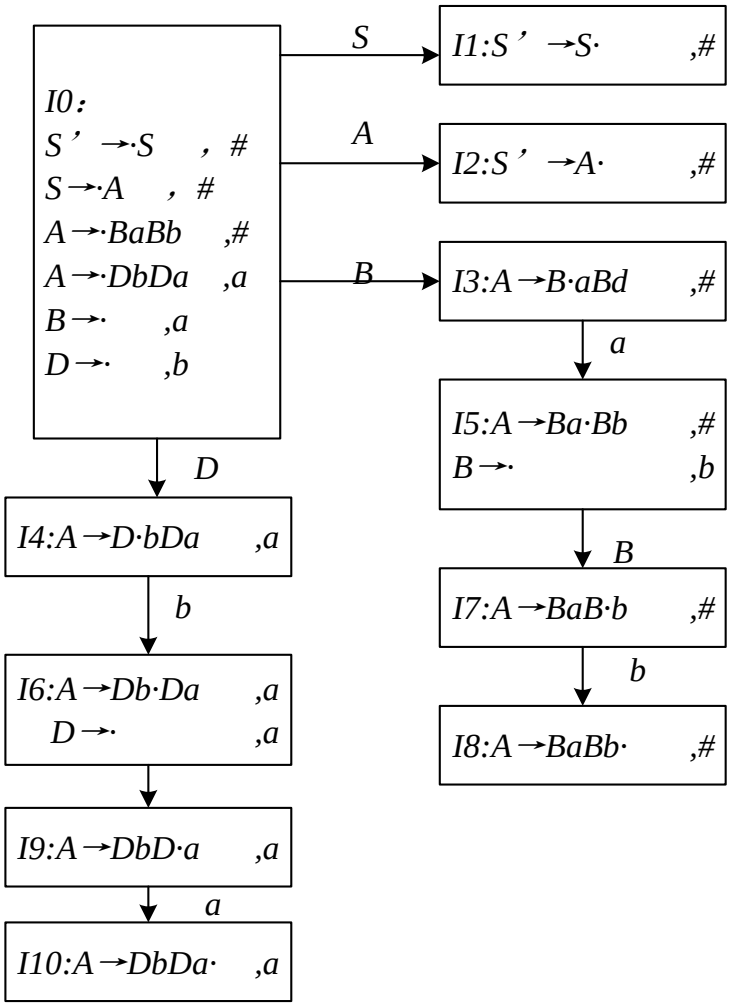
5:D➞ε

$G[S']$的$LR(0)$项目集族及识别活前缀的DFA状态 I0 如下图所示：

I0：
S'→·S
S→·A
A→·BaBb
B→·DbDa
B→·
D→·

在I_0中：$B \to \cdot$和$D \to \cdot$为归约-归约冲突，$Follow(B) \cap Follow(D) = \{a, b\} \cap \{a, b\} \neq \emptyset$，所以在$I_2$中的移进-归约冲突不能由$SLR(1)$规则解决，不是 SLR(1)文法。

$G[S']$的$LR(1)$项目集族及识别活前缀的DFA状态如下图所示：



0: S'→S　　1:S→A　2:A→BaBb　　3:A→DbDa　　4:B→ε　　5:D→ε

构造的$LR(1)$分析表如下：

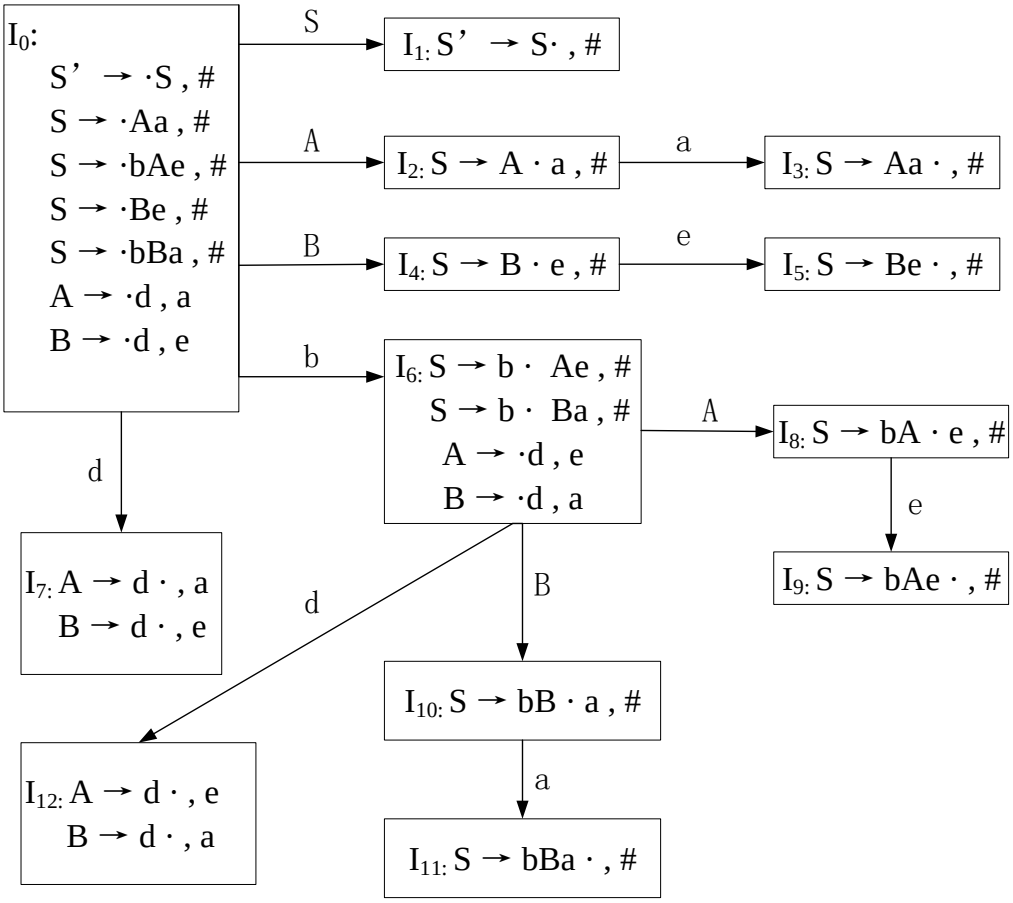
状态	ACTION			GOTO			
	a	b	#	S	A	B	D
0	$r4$	$r5$		1	2	3	4
1			acc				
2			$r1$			7	
3	$S5$						
4		$S6$					
5		$r4$				7	
6	$r5$						9
7		$S8$	$r3$				
8			$r2$			7	
9	$S10$						
10	$r3$						

9.证明下面文法是 LR(1)而不是 LALR(l)的。

S→Aa | bAe |Be | bBa A→d B→d

答：拓广文法并对产生式进行编号：

0: S'→S　1：S→Aa　2：S→bAe　3：S→Be　4：S→bBa　5：A→d　6：B→d



0: S'→S　1：S→Aa　2：S→bAe　3：S→Be　4：S→bBa　5：A→d　6：B→d

构造的*LALR*(1)分析表如下：

状态	ACTION					GOTO		
	a	***b***	**d**	**e**	#	***S***	***A***	***B***
0		***S*6**	***S*7**			**1**	**2**	**4**
1					***acc***			
2	***S*3**							
3					***r*1**			
4				***S*5**				
5					***r*3**			
6			***S*12**				**8**	**10**
12,7	***r*5, *r*6**			***r*5, *r*6**				
8				***S*9**				**7**
9					***r*2**			
10	***S*11**							
11					***r*4**			

在I_7中和在I_{12}中，合并同心集之后，分析表有冲突，所以不是 LALR(l)的。

0: S’→S　1：S→Aa　2：S→bAe　3：S→Be　4：S→bBa　5：A→d　6：B→d

构造的$LR(1)$分析表如下：

状态	ACTION					GOTO			
	a	b	d	e	#	S	A	B	
0		$S6$	$S7$			1	2	4	
1					acc				
2	$S3$								
3					$r1$				
4				$S5$					
5					$r3$				
6			$S12$				8	10	
7	$r5$			$r6$					
8				$S9$				7	
9					$r2$				
10	$S11$								
11					$r4$				
12	$r6$			$r5$					

LR(1)分析表无冲突，所以是 LR(1)的。

10. 判断下列 6 个文法是否为LR类文法，若是，请说明是$LR(0)$、$SLR(1)$、$LALR(1)$或$LR(1)$的哪一种，并构造相应的分析表；若不是，请说明理由。

(1) $S \to AB$

$A \to aBa|\varepsilon$

$B \to bAb|\varepsilon$

答：拓广文法为G'，增加产生式$S' \to S$，若产生式排序为：

0 $S' \to S$

1 $S \to AB$

2 $A \to aBa$

3 $A \to \varepsilon$

4 $B \to bAb$

5 $B \to \varepsilon$

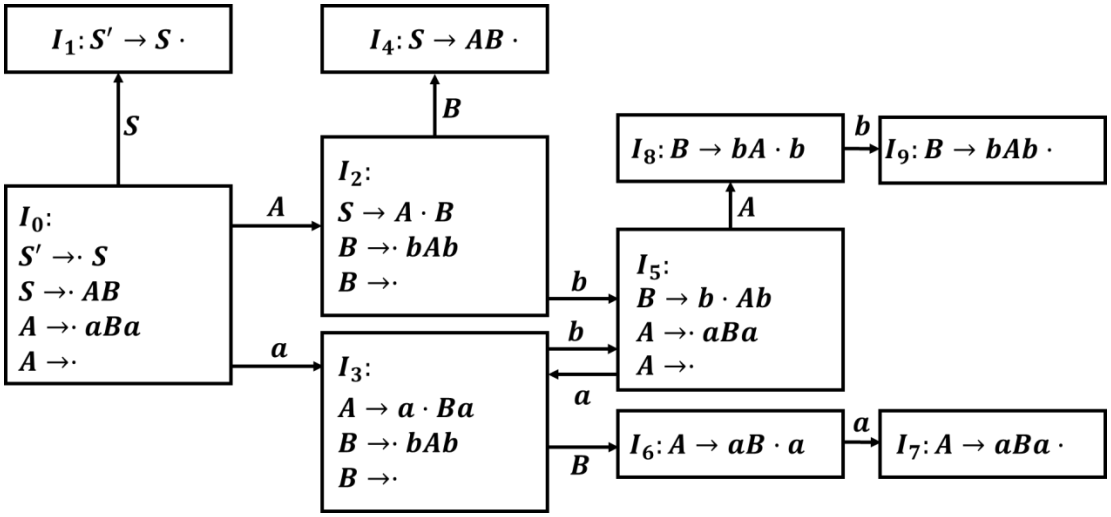
由产生式知：

$Follow(S) = \{\#\}$

$Follow(A) = \{b, \#\}$

$Follow(B) = \{a, \#\}$

G'的$LR(0)$项目集族及识别活前缀的DFA如下图所示：



在$I_0: A \to \cdot aBa$为移约项目，$A \to \cdot$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。用$SLR(1)$文法能解决冲突，所以构造$SLR(1)$。

构造$SLR(1)$的分析表如下：

状态	ACTION			GOTO		
	a	b	#	S	A	B
0	S_3	r_3	r_3	1	2	
1			acc			
2	r_5	S_5	r_5			4
3	r_5	S_5	r_5			6
4			r_1			
5	S_3	r_3	r_3		8	
6	S_7					
7		r_2	r_2			
8		S_9				
9	r_4		r_4			

(2) $S \to D;B|B$

$D \to d|\varepsilon$

$B \to B;a|a|\varepsilon$

答：拓广文法为G'，增加产生式$S' \to S$，若产生式排序为：

0	$S' \to S$
1	$S \to D;B$
2	$S \to B$
3	$D \to d$
4	$D \to \varepsilon$
5	$B \to B;a$
6	$B \to a$
7	$B \to \varepsilon$

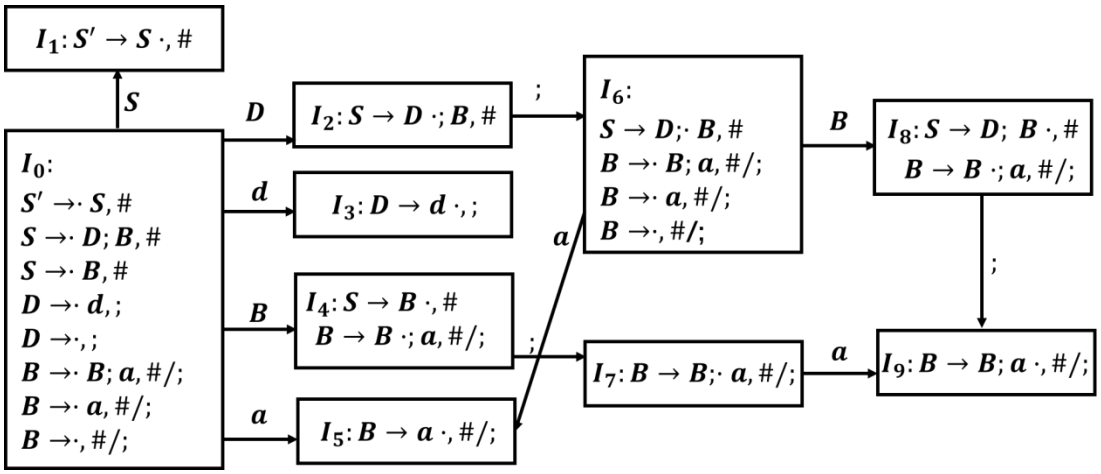
$Follow(S) = \{\#\}$

$Follow(D) = \{;\}$

$Follow(B) = \{\#,;\}$

G'的$LR(0)$项目集族和$SLR(1)$项目集族有归约-归约冲突。

G'的$LR(1)$项目集族及识别活前缀的DFA如下图所示：



在I_0:存在归约-归约冲突，所以不是$LR(1)$。

(3)

$S \to aAd|eBd|aBr|eAr$

$A \to a$

$B \to a$

答：拓广文法为G'，增加产生式$S' \to S$，若产生式排序为：

0	$S' \to S$
1	$S \to aAd$
2	$S \to eBd$
3	$S \to aBr$
4	$S \to eAr$
5	$A \to a$
6	$B \to a$

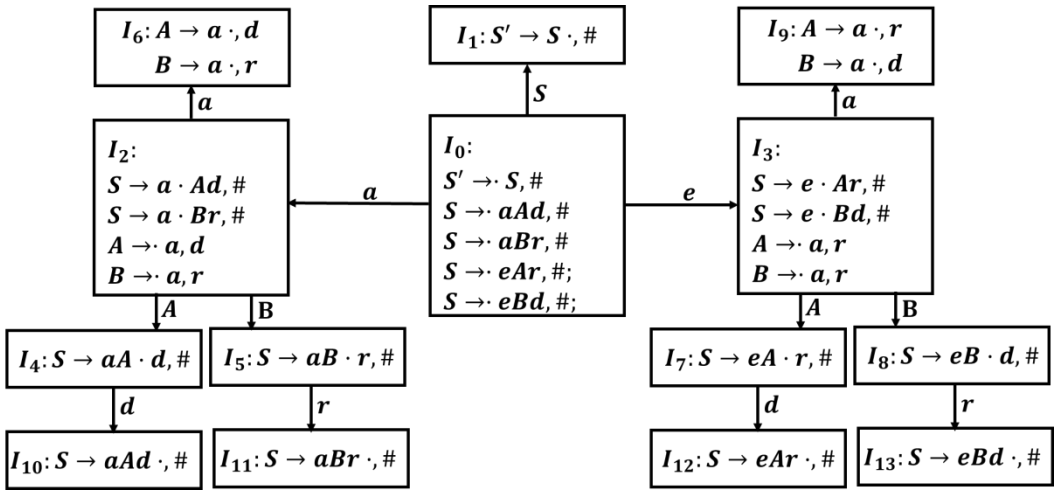
$Follow(S) = \{\#\}$

$Follow(A) = \{d, r\}$

$Follow(B) = \{d, r\}$

G'的$LR(0)$项目集族和$SLR(1)$项目集族有归约-归约冲突。

G'的$LR(1)$项目集族及识别活前缀的DFA如下图所示：



构造$LR(1)$无冲突。而同心集I_6和I_9无法合并，所以是$LR(1)$，而不是是$LALR(1)$

构造$LR(1)$的分析表如下：

状态	ACTION					GOTO		
	a	e	d	r	#	A	B	S
0	S_2	S_3						1
1					acc			
2	S_6					4	5	
3	S_9					7	8	
4			S_{10}					
5				S_{11}				
6			r_5	r_6				
7				S_{12}				
8			S_{13}					
9			r_6	r_5				
10					r_1			
11					r_3			
12					r_4			
13					r_2			

(4)

$A \to AbBa|B$

$B \to a|\varepsilon$

答：拓广文法为G'，增加产生式$A' \to A$，若产生式排序为：

0 $A' \to A$

1 $A \to AbBa$

2 $A \to B$

3 $B \to a$

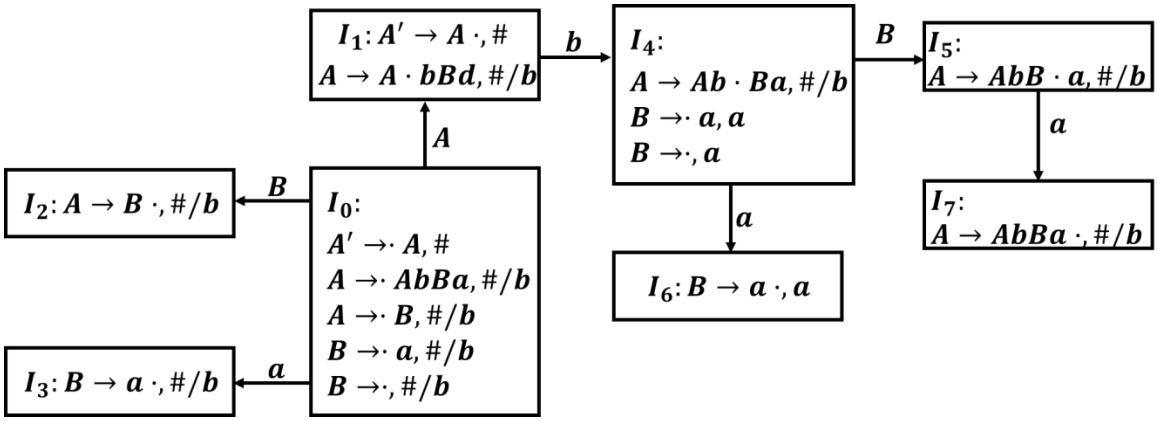
4 $B \to \varepsilon$

$Follow(A) = \{b, \#\}$

$Follow(B) = \{a, b, \#\}$

G'的$LR(0)$项目集族和$SLR(1)$项目集族有移进-归约冲突。

G'的$LR(1)$项目集族及识别活前缀的DFA如下图所示：



在I_4中，$B \to .a, a/\#$和$B \to ., a/\#$有移进-归约冲突，所以也不是$LR(1)$文法。

16.给定文法：

$S \to \underline{do}\ S\ \underline{or}\ S \mid \underline{do}\ S \mid S; S \mid \underline{act}$

(1) 构造识别该文法活前缀的DFA

(2) 该文法是$LR(0)$的吗？是$SLR(1)$的吗？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：

① $\underline{or}$优先性大于$\underline{do}$

② ;服从左结合

③ ;优先性大于$\underline{do}$

④ ;优先性大于$\underline{or}$

请构造该文法的LR分析表。

答：首先化简文法，用d代替$\underline{do}$；用o代替$\underline{or}$；用a代替$\underline{act}$；文法可写成：

$S \to dSoS|dS|S;S|a$

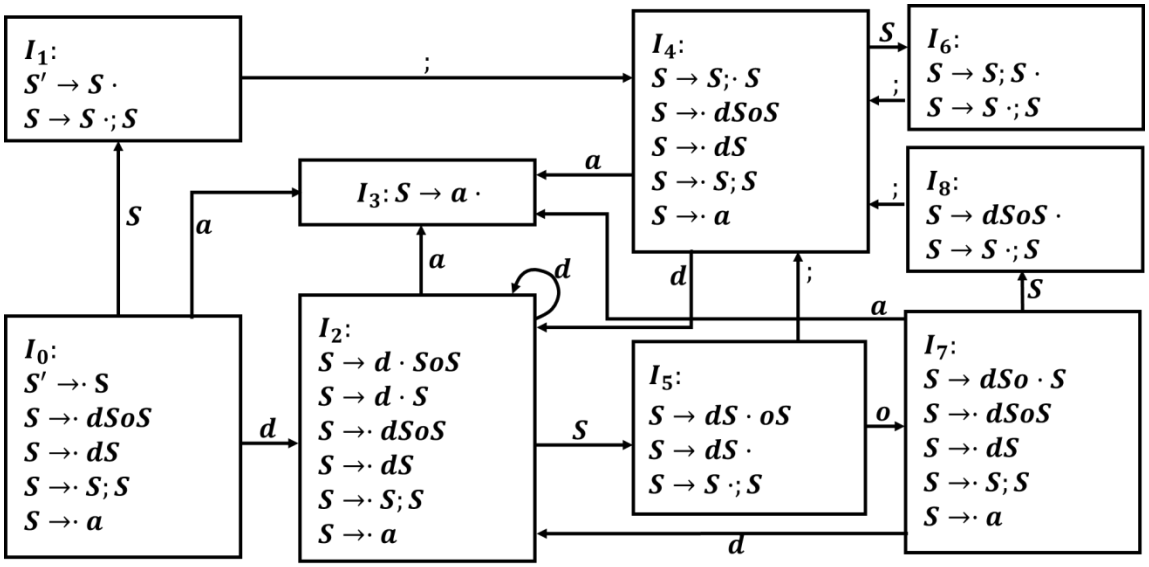
拓广文法为G'，增加产生式$S' \to S$，若产生式排序为：

0	$S' \to S$
1	$S \to dSoS$
2	$S \to dS$
3	$S \to S;S$
4	$S \to a$

由产生式知：

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

(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：

$Follow(S) \cap \{;\} = \{o, ;, \#\} \cap \{;\} = \{;\} \neq \emptyset$，在$I_5$中：

$Follow(S) \cap \{;, o\} = \{o, ;, \#\} \cap \{;\} = \{;, o\} \neq \emptyset$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。

(3) 在I_5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I_6中：;号服从左结合，所以遇输入符$Follow(S)$的都应该归约。

在I_8中：;号优先性大于 do，所以遇输入符;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，因此接受只有遇#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	ACTION					GOTO
	d	o	;	a	#	S
0	S_2			S_3		1
1			S_4		acc	
2	S_2			S_3		5
3		r_4	r_4		r_4	
4	S_2			S_3		6
5		S_7	S_4		r_2	
6		r_3	r_3		r_3	
7	S_2			S_3		8
8		r_1	S_4		r_1	

第七章作业

1. 下面的文法$G[S']$描述由布尔常量$false, true$,联结词∧(合取)、∨(析取)、¬(否定)构成的不含括号的二值布尔表达式的集合：

$S' \to S$

$S \to S \vee T|T$

$T \to T \wedge F|F$

$F \to \neg F|false|true$

试设计一个基于$G[S']$的属性文法，它可以计算出每个二值布尔表达式的取值。如对于句子$\neg true \vee \neg false \wedge true$，输出是 true。

答：

$S' \to S$	$\{print\ S.value\}$
$S \to S_1 \vee T$	$\{ if(S_1.value == true\ or\ T.value == true)$
	$then\ S.value = true$
	$else\ S.value = false\}$
$S \to T$	$\{S.value = T.value\}$
$T \to T_1 \wedge F$	$\{if(T_1.value == true\ and\ F.value == true)$
	$then\ T.value = true$
	$else\ T.value = false$
$T \to F$	$T.value = F.value\}$
$T \to \neg F$	$\{if(F.value == true)$
	$then\ T.value = false$
	$else\ T.value = true\}$
$F \to false$	$\{F.value = false\}$
$F \to true$	$\{F.value = true\}$

2. 给定文法$G[S]$：

$S \to (L)|a$

$L \to L,S|S$

如下是相应于G[S]的一个属性文法(或翻译模式)：

$S \to (L)$	$\{S.num := L.num + 1;\}$
$S \to a$	$\{S.num := 0;\}$
$L \to L_1,S$	$\{L.num := L_1.num + S.num;\}$
$L \to S$	$\{L.num := S.num;\}$

图 7.19 分别是输入串$(a,(a))$的语法分析树和对应的带标注语法树，但后者的属性值没有标出，试将其标出(即填写图 7.19 右图中符号=右边的值)。

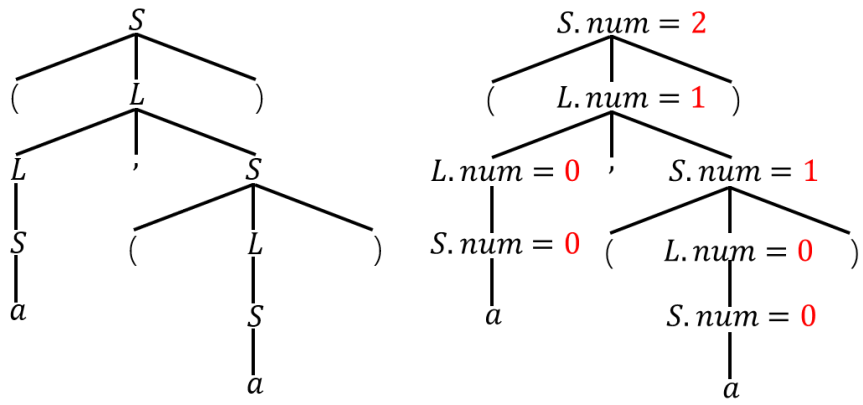


图 7.19 题 2 的语法分析树和带标注语法树

答：如上图所示

4. 以下是简单表达式(只含加、减运算)计算的一个属性文法$G(E)$:

$E \to TR$ $\{R.in := T.val; E.val := R.val\}$

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

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

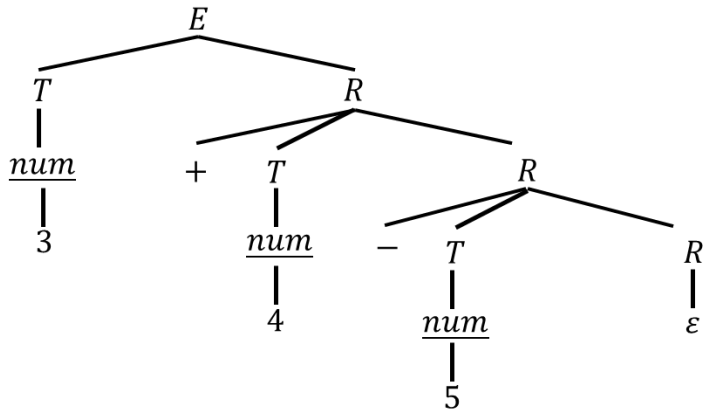
$R \to \varepsilon$ $\{R.val := R.in;\}$

$T \to \underline{num}$ $\{T.val := lexval(num)\}$

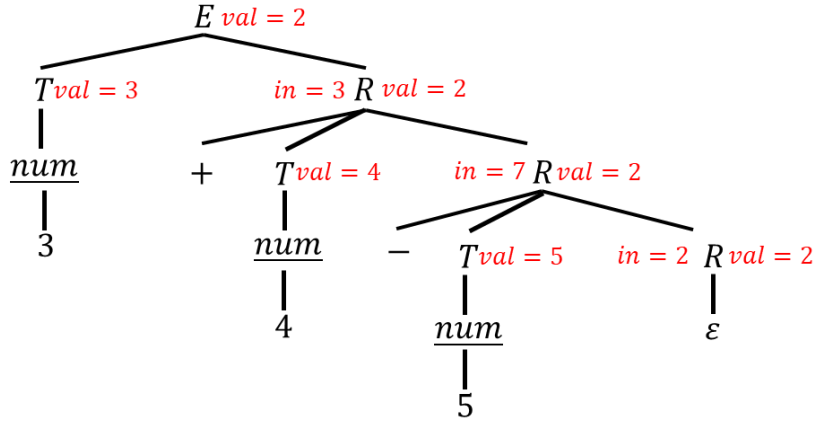
其中，$lexval(num)$表示从词法分析程序得到的常数值。

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：语法分析树如下



带标注的语法树如下：



5. 题 2 中所给的$G[E]$的属性文法是一个S-属性文法，故可以在自底向上分析过程中增加语义栈来计算属性值，图 7.20 是$G[S]$的一个LR分析表，图 7.21 描述了输入串$(a,(a))$的分析和求值过程(语义栈中的值对应$S.num$或$L.num$)，其中，第 14、15 行没有给出，试着补全。

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			

图 7.20 题 5 的LR分析表

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	0246	_ _ 1 _	#(L)	#
15	01	_ 2	#S	#
16	接受			

图 7.21 题 5 的分析和求值过程

答：如上图所示

7.设题 4 中属性文法的基础文法为$G[E]$。

(1) 说明$G[E]$是$LL(1)$文法。

(2) 如下是以$G[E]$作为基础文法设计的翻译模式：

$E \to T \ \{R.in := T.val\} \ R \ \{E.val := R.val\}$

$R \to +T \ \{R_1.in := R.in + T.val\} \ R_1 \ \{R.val := R_1.val\}$

$R \to -T \ \{R_1.in := R.in - T.val\} \ R_1 \ \{R.val := R_1.val\}$

$R \to \varepsilon \quad \{R.val := R.in;\}$

$T \to \underline{num} \quad \{T.val := lexval(num)\}$

试针对该翻译模式构造相应的递归下降(预测)翻译程序(如题 6,可直接使用例 7.9 中的MatchToken函数)。

答:

每个产生式的$SELECT$集合如下:

$SELECT(E \to T) = \{num\}$

$SELECT(R \to +T) = \{+\}$

$SELECT(R \to -T) = \{-\}$

$SELECT(R \to \varepsilon) = \{\#\}$

$SELECT(T \to num) = \{num\}$

相同左部产生式的$SELECT$交集为

$SELECT(R \to +T) \cap SELECT(R \to -T) \cap SELECT(R \to \varepsilon) = \{+\} \cap \{-\} \cap \{\#\} = \emptyset$

所以该文法为$LL(1)$文法。

对应的递归下降翻译程序为

文法$G[E]$对应的递归下降翻译程序

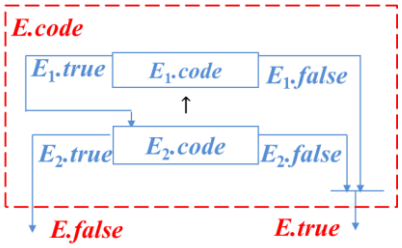
```
1.  int ParseE()
2.  {
3.      Tv:=ParseT();              //变量 Tv 对应属性 T.value
4.      Ri:=Tv;                    //变量 Ri 对应属性 R.in
5.      Rv:=ParseR(Ri);
6.      Ev:=Rv;
7.       return Ev;
8.  }
9.  int ParseR(int f)              //形参 f 对应属性 R.in
10. {
11.     if(lookahead == '+')       //lookahead 是当前扫描的输入符号
12.     {
13.         MatchToken('+');
14.         Tv:=ParseT();
15.         R1i:=f+Tv;             //R1i 对应属性 R1.in
16.         R1v:=ParseR(R1i);
17.         Rv:=R1v;
18.     }
19.     else if(lookahead == '-')
20.     {
21.         MatchToken('-');
22.         Tv:=ParseT();
23.         R1i=f-Tv;
24.         R1v:=ParseR(R1i);
25.         Rv:=R1v;
26.     }
27.     else if(lookahead == '#')
28.     {
```

```
29.         Rv = f;
30.     }
31.     else
32.     {
33.         printf("Syntax error");
34.         exit(0);
35.     }
36.    return Rv;
37. }
38. int ParseT()
39. {
40.     if(lookahead == '(lexvalnum)')
41.     {
42.         MatchToken('(lexvalnum)');
43.         Tv:=lexval(num);
44.     }
45.     else
46.     {
47.         printf("Syntax error.");
48.         exit(0);
49.     }
50.     return Tv;
51. }
52. void MatchToken(int expected)
53. {
54.     if(lookahead!= expected)
55.     {
56.         printf("syntax error\n");
57.         exit(0);
58.     }
59.     else
60.     {
61.         lookahead=getToken();
62.     }
63. }
```

第八章作业

4. 参考 8.3.3.4 节采用短路代码进行布尔表达式翻译的 L-翻译模式片段及用到的语义函数。若在基础文法中增加产生式$E \to E \uparrow E$，试给出与该产生式相应的语义动作集合。其中，↑代表“与非”逻辑算符，其语义可用其他逻辑运算定义为$P \uparrow Q = not(P\ and\ Q)$。

答：



$$E \to \{E_1.true = newlable();\ E_1.false = E.true\}E_1 \uparrow \{lable(E_1.true);\ E_2.true = E.false;\ E_2.false = E.true\}E_2$$

或者

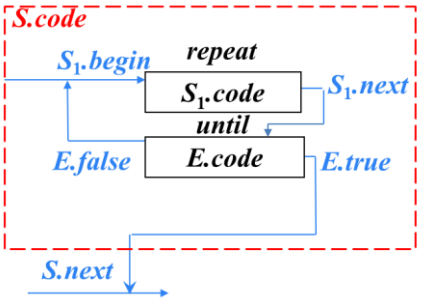
$$E \to \{\ E_1.\text{false} := E.\text{true}\ ;\ E_1.\text{true} := newlabel\ ;\ \}\ E_1 \uparrow \{\ E_2.\text{false} := E.\text{true}\ ;\ E_2.\text{true} := E.\text{false}\ ;\ \}\ E_2$$

$$\{\ E.code := E_1.code \| gen\ (E_1.true\text{':'}) \| E_2.code\ \}$$

5. 参考 8.3.3.5 节进行控制语句（不含 break）翻译的 L-翻译模式片段及所用到的语义函数。若在基础文法中增加产生式$S \to repeat\ S\ until\ E$,试给出与该产生式相应的语义动作集合。

注：控制语句 $repeat <$ 循环体 $>\ until <$ 布尔表达式 $>$的语义为：至少执行<循环体>一次，直到<布尔表达式>成真时结束循环。

答：

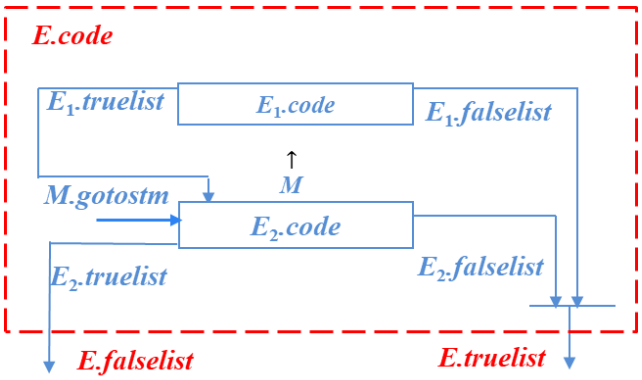


$$S \to repeat\ \{\ S_1.begin = newlable();\ lable(\ S_1.begin);S_1.next = newlable();\}S_1\ until\ \{lable(\ S_1.next);E.true = S.next;\ E.false = S_1.begin\}E$$

或者

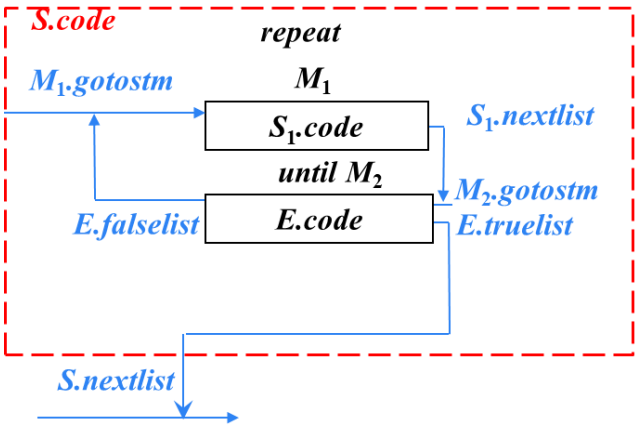
$$S \to repeat\ \{\ S_1.next := newlabel\ ;\ \}\ S_1\ until\ \{\ E.true := S.next\ ;\ E.false := newlabel\ ;\ \}\ E\ \{\ S.code := gen(E.false\text{':'}) \| S_1.code \| gen(S_1.next\text{':'}) \| E.code\ \}$$

6. 参考 8.3.3.6 节采用拉链与代码回填技术进行布尔表达式和控制语句翻译的 S-翻译模式片段及所用到的语义函数，重复题 4 和题 5 的工作。



$E \rightarrow E_1 \uparrow ME_2$ $\{backpatch(E_1.truelist, M.gotostm); E.truelist = Merge(E_1.falselist, E_1.falselist); E.falselist = E_2.truelist\}$

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



$S \rightarrow repeat\ M_1\ S_1\ until\ M_2\ E$
$\{backpatch(E.falselist, M_1.gotostm); backpatch(S_1.nextlist, M_2.gotostm); S.nextlist = E.truelist\}$

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