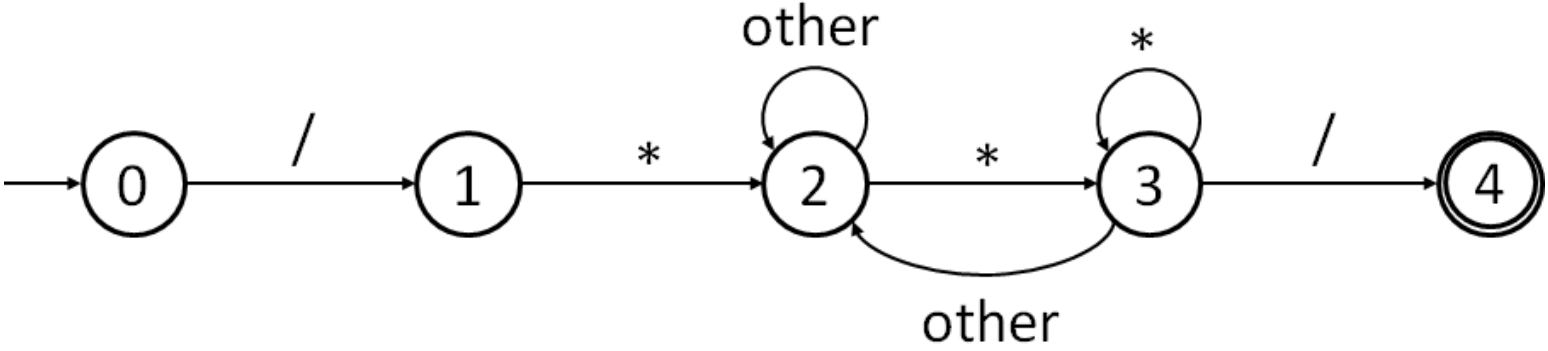
识别注释的DFA

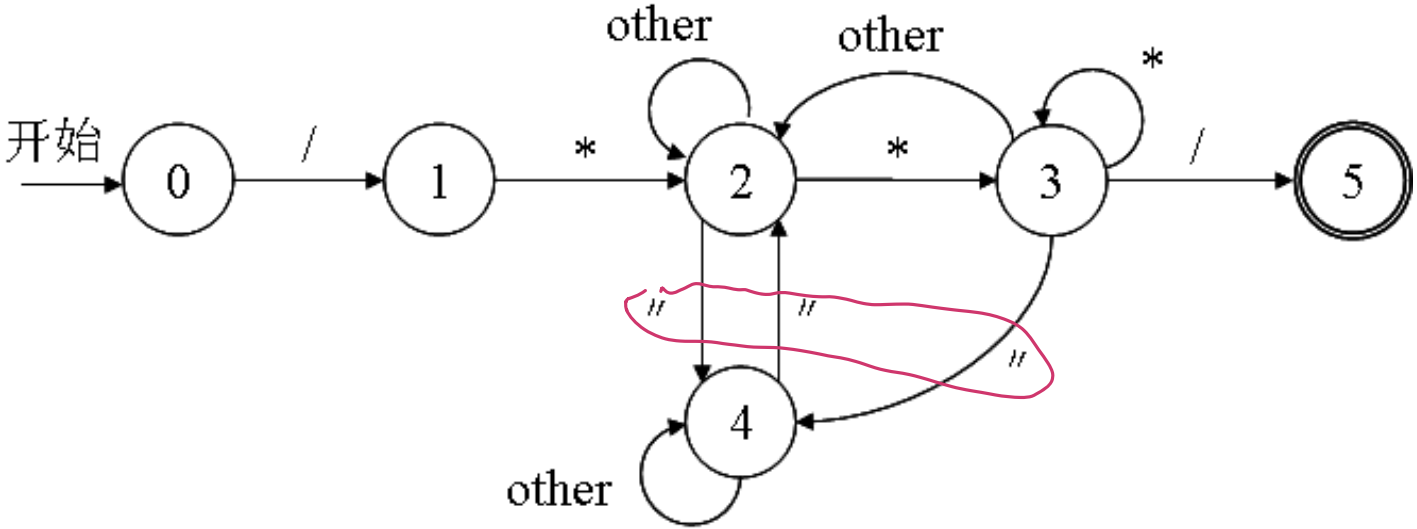


习题3.1：

设某程序设计语言规定，其程序中的注释是由“/*”和“*/”括起来的字符串，注释中不能出现“*/”，除非它们出现在双引号中（假设双引号必须配对使用），请给出识别该语言注释结构的DFA D。

解答:

- 识别形如/*……"…*/…"……*/的注释的DFA

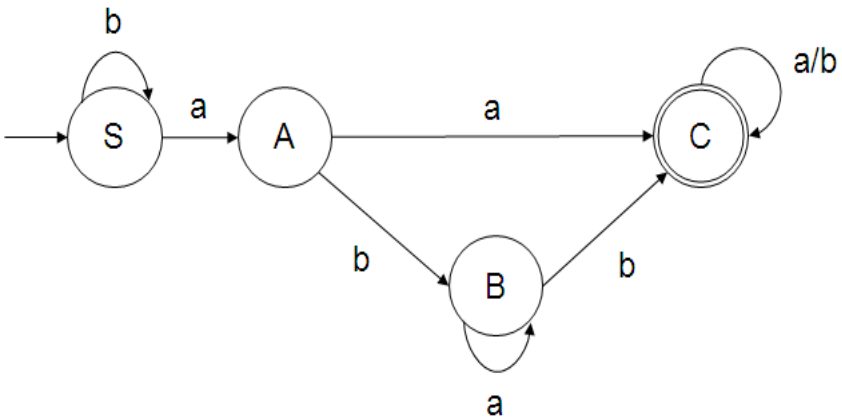


课堂练习1

- 自动机 M 的状态转换矩阵如下所示，其中初态是S，终态是C。
  （1）画出相应的状态转换图；
  （2）写出与之等价的右线性文法。

	a	b
S	A	S
A	C	B
B	B	C
C	C	C

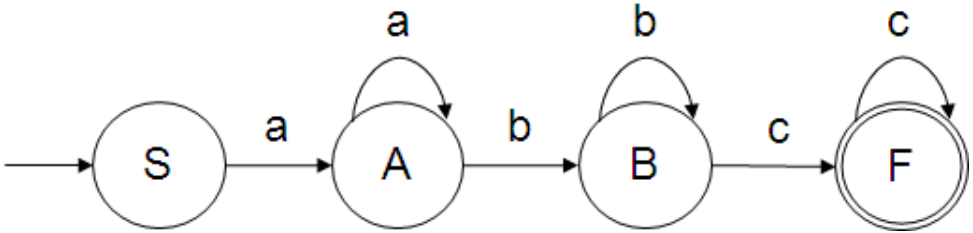
- 解答：



S□aA | bS
A□aC | bB
B□aB | bC
C□aC | bC | □

课堂练习2

- 自动机 M 的状态转换图如下所示。
  （1）该自动机识别的语言是什么?
  （2）给出与之等价的右线性文法。



解答:

（1）根据自动机知其产生的语言是：$L=\{a^m b^n c^i \mid m, n, i \geq 1\}$

（2）与之等价的右线性文法是:

$S \rightarrow aA$
$A \rightarrow aA \mid bB$
$B \rightarrow bB \mid cF$
$F \rightarrow cF \mid \varepsilon$

或者：$S \rightarrow aA$
$A \rightarrow aA \mid bB$
$B \rightarrow bB \mid cF \mid c$
$F \rightarrow cF \mid c$

课堂练习3

- 已知正则表达式：$(a^*|b)^*$ 只有1个C

  $(c|d)$，判断下面哪几个正则表达式与其等价，请简述理由。

  (1) $a^*(c|d)|b(c|d)$

  (2) $a^*(c|d)^*|\ b(c|d)^*$

  (3) $a^*(c|d)|b^*(c|d)$ 可有2个C

  (4) $(a|b)^*c|(a|b)^*d$

  (5) $(a^*|b)^*c|(a^*|b)^*d$

- 解答：

  (1)、(2)、(3)与所给正则表达式不等价；

  (4)和(5)与所给正则表达式等价。

课堂练习4

- 有限自动机M：

  $M=(\{a,b\},\{S_0,S_1,S_2,S_3,S_4,S_5\}, S_0, \{S_1,S_4,S_5\}, \delta)$

  δ由如右的状态转移矩阵给出。

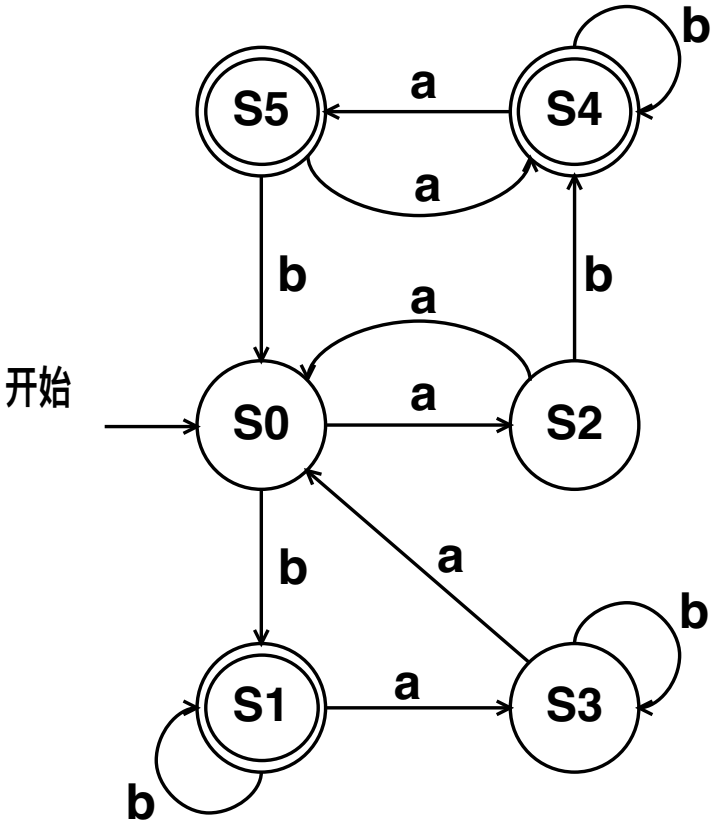
  (1) 试画出该自动机的状态转换图；

  (2) 试找出一个长度最小的输入串，
  使得在识别此输入串的过程中，
  每一状态至少经历一次；

  (3) 试找出一个长度最小的输入串，
  使得每一状态转换至少经历一次。

	a	b
S_0	S_2	S_1
S_1	S_3	S_1
S_2	S_0	S_4
S_3	S_0	S_3
S_4	S_5	S_4
S_5	S_4	S_0

课堂练习4参考答案



baaaba

aaabbaaabbbabab

练习4.1

有如下文法:

bexpr → *bexpr* or *bterm* | *bterm*

bterm → *bterm* and *bfactor* | *bfactor*

bfactor → not *bfactor* | (*bexpr*) | true | false

请构造一个可以用来分析该文法所产生的句子的递归调用分析程序。

Step 1：消除左递归

bexpr → *bterm* *E* →

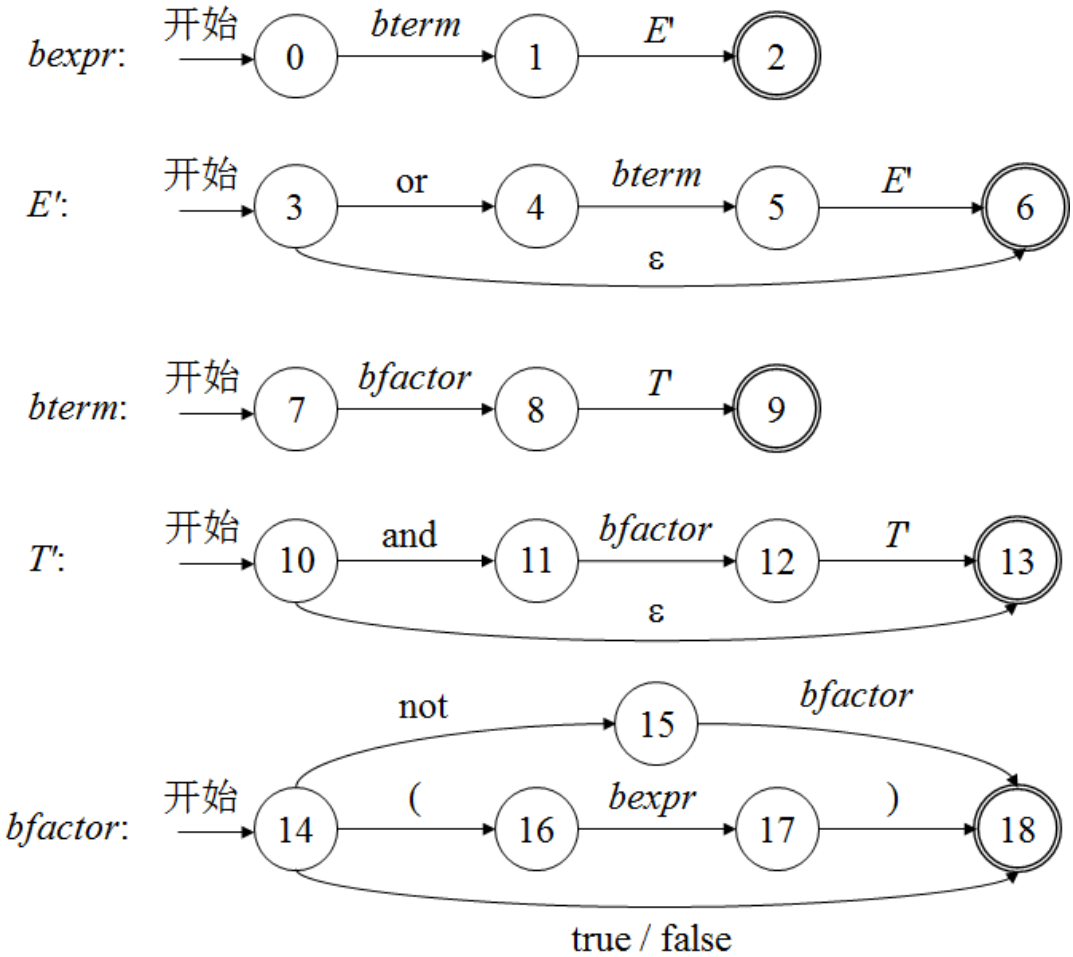
E → or *bterm* *E* → | →

bterm → *bfactor* *T* →

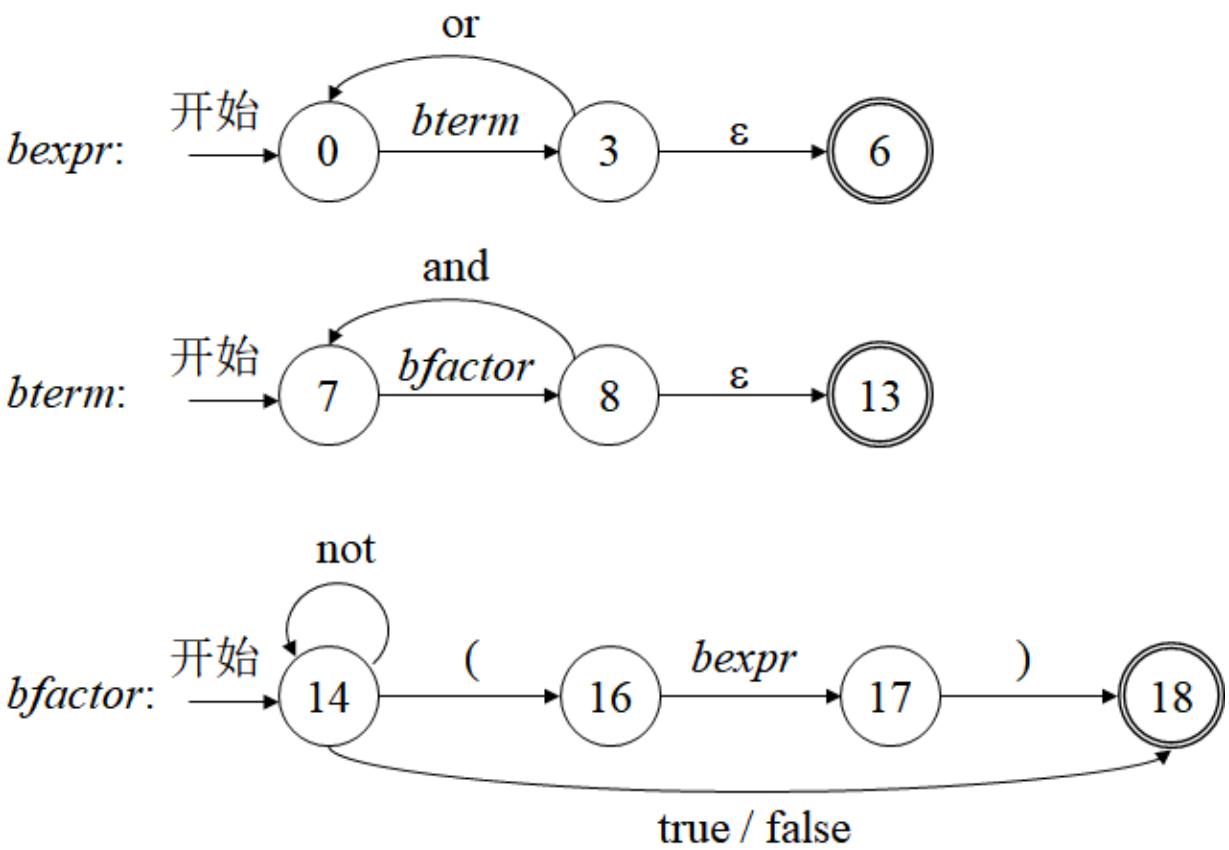
T → and *bfactor* *T* → | →

bfactor → not *bfactor* | (*bexpr*) | true | false

Step 2：文法*G*的预测分析程序状态转换图



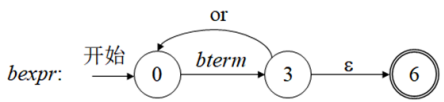
Step 3：化简后的预测分析程序状态转换图



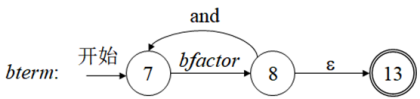
Step4:根据状态转换图进行程序设计

- *bexpr*的函数
- *bterm*的过程
- *bfactor*的过程

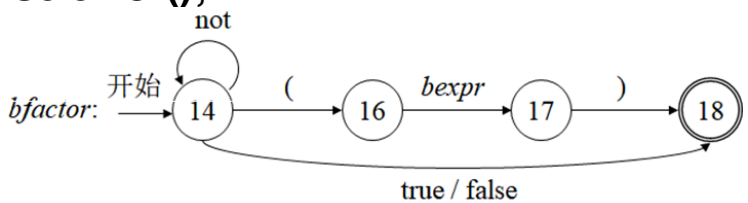
```
void proc_expr(void) {
   proc_term();
   if (char==?or?) {
      forward pointer;
      proc_expr();
   }
}
```




```
void proc_term(void) {
   proc_factor();
   if (char==?and?) {
      forward pointer;
      proc_term();
   }
}
```




```
void proc_factor(void) {
   if (char==?not?){
      forward pointer;
      proc_factor();
   }
   else if (char==?(?) {
      forward pointer;
      proc_expr();
       if (char==?)?)
          forward pointer;
      else  error();
   };
   else if
(char==?true?)||(char==?false?)
          forward pointer;
   else error();
}
```



练习： 判断下面的文法是否为LL(1)文法?
若不是，可否改写为LL(1)文法?
若可以，请构造其LL(1)分析表。

S□(L) | a
L □L , S | S

解答：

- 文法含有左递归，故不是LL(1)文法
- 改写文法： 消除左递归

S□(L) | a
L □SL□
L□ □,SL□ | □

FIRST((L)) ∩FIRST(a) = □
FIRST(,SL□) ∩FOLLOW(L□) = □

	FIRST	FOLLOW
S	(a	$,)
L	(a	)
L□	, □	)

	a	(	)	,	$
S	S□a	S□(L)			
L	L□SL□	L□SL□			
L□			L□□	L□□,	

练习：

1) S□aAcBe
2) A□b
3) A□Ab
4) B□d

构造LL(1)分析表，分析 abbcde。

解答：

S□aAcBe
A□bA'
A' □bA' | □
B□d

	First	follow
S	a	$
A	b	c
A'	b, □	c
B	d	e

	a	b	c	d	e	$
S	S□aAcBe					
A		A□bA'				
A'		A'□bA'	A'□□			
B				B□d		

课堂练习1

有如下文法：

E□E□T | T

T□T□F | F

F□□F | (E) | t | f

(1) 该文法是LL(1)文法吗？说明理由。

若是，做(3)，若不是，做(2)

(2) 请改写该文法为LL(1)文法，继续做(3)。

(3) 构造每个非终结符号的FIRST和FOLLOW函数，继续做(4)。

(4) 构造LL(1)分析表。

参考答案

E□E□T | T
T□T□F | F
F□□F | (E) | t | f

(1) 由于该文法存在左递归，所以不是LL(1)文法。

(2) 改写文法。消除其中的左递归，得到文法G'：

E□TE'
E'□□TE' | □
T□FT'
T'□□FT' | □
F□□F | (E) | t | f

(3) 每个非终结符号的FIRST和FOLLOW集合如右：

	FIRST	FOLLOW
E	¬, (, t, f	$,)
E'	∨, ε	$,)
T	¬, (, t, f	∨, $,)
T'	∧, ε	∨, $,)
F	¬, (, t, f	∧, ∨, $,)

(4) 文法的LL(1)分析表如下：

	¬	∧	∨	t	f	(	)	$
E	E→TE'			E→TE'	E→TE'	E→TE'		
E'			E'→∨TE'				E'→ε	E'→ε
T	T→FT'			T→FT'	T→FT'	T→FT'		
T'		T'→∧FT'	T'→ε				T'→ε	T'→ε
F	F→¬F			F→t	F→f	F→(E)		

课堂练习2

证明下面的文法是LL(1)的，但不是SLR(1)的。

S→(X | E] | F)

X→E) | F]

E→A

F→A

A→ε

解答：证明该文法是LL(1)文法

S?(X | E] | F)
X?E) | F]
E?A
F?A
A??

- 该文法每个非终结符号的FIRST集和FOLLOW集合如下：

	FIRST	FOLLOW
S	(,],)	$
X	],)	$
E	ε	],)
F	ε	],)
A	ε	],)

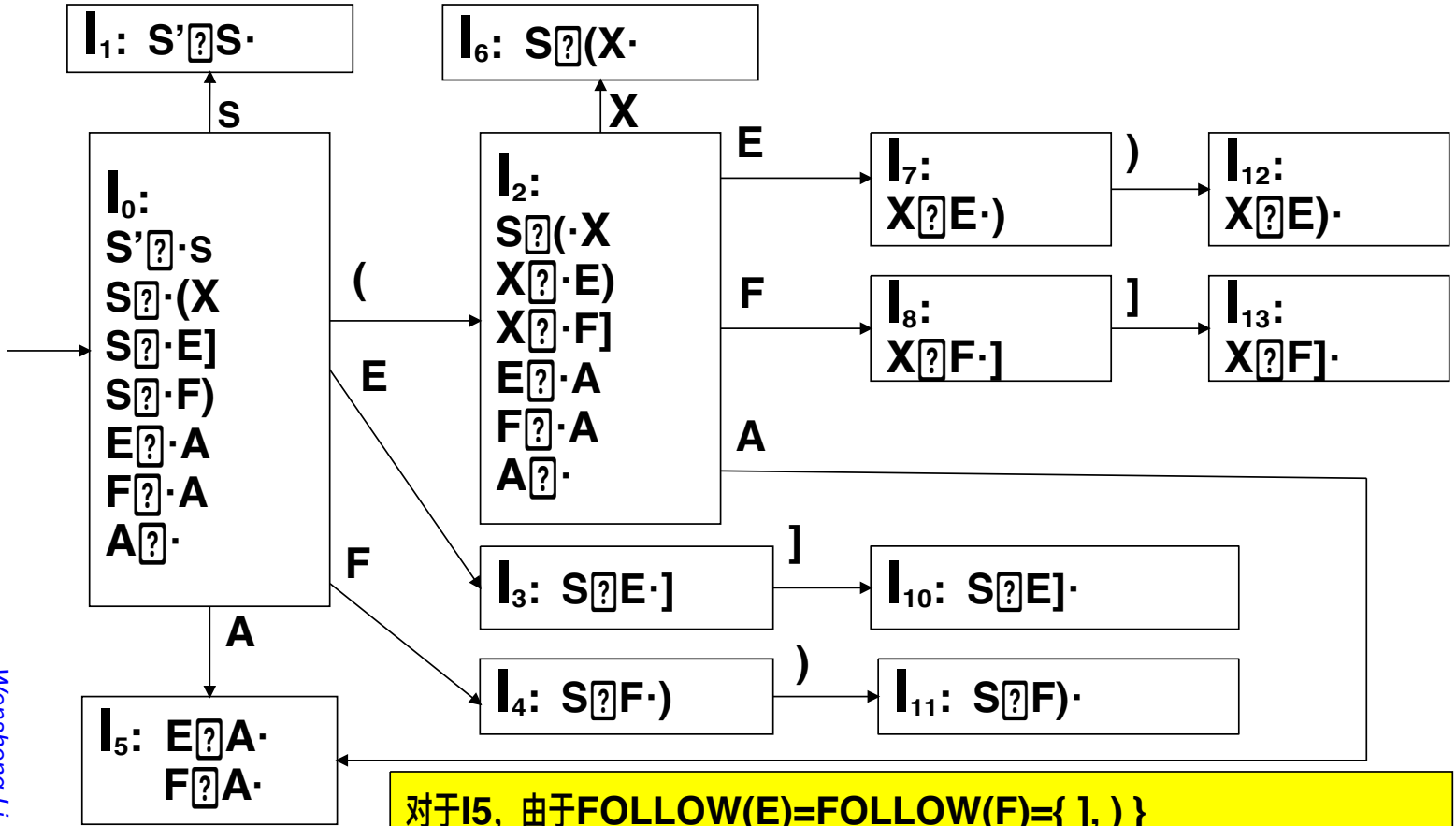
- 该文法的LL(1)分析表如右：

	(	)	]	$
S	S→(X	S→F)	S→E]	
X		X→E)	X→F]	
E		E→A	E→A	
F		F→A	F→A	
A		A→ ε	A→ ε	

- 结论： LL(1)分析表中不含有多重定义的入口，所以该文法是LL(1)文法
- 也可以分析产生式，根据候选式的first集合互不相交来说明。

证明该文法不是SLR(1)文法

- 构造该文法的LR(0)项目集规范族及识别所有活前缀的DFA



对于I5，由于FOLLOW(E)=FOLLOW(F)={],) }
该集合中存在归约-归约冲突，所以该文法不是SLR(1)文法。

课堂练习3

说明下面的文法是LR(1)文法，但不是SLR(1)文法。

X→Ma | bMc | dc | bda

M→d

解答：

首先，拓广文法

(0) S?X

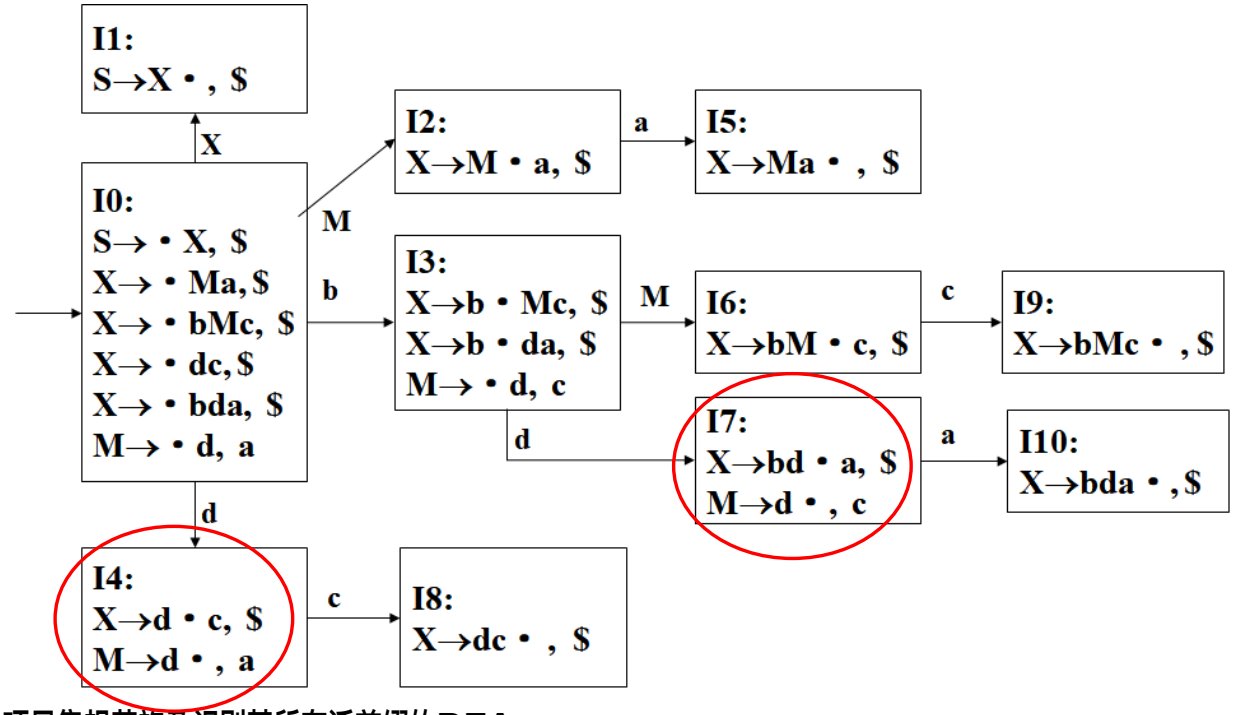
(1) X?Ma

(2) X? bMc

(3) X? dc

(4) X? bda

(5) M?d



其次，构造文法的LR(1)项目集规范族及识别其所有活前缀的DFA。

判断该文法是LR(1)文法：集合I0、I3中没有归约项目，所以，不存在冲突；

集合I1、I2、I5、I6、I8、I9、I10各只有一个归约项目，所以这些集合中没有冲突；

集合I4和I7中既有移进项目又有归约项目，但是归约符号和移进符号不同，所以也没有冲突。结论：是LR(1)

然后，构造文法的LR(0)项目集规范族及识别其所有活前缀的DFA。

FOLLOW(S)={ $ }
FOLLOW(X)={ $ }
FOLLOW(M)={ a, c }

I_4、I_7中存在移进-归约冲突， FOLLOW(M)={ a, c }
这种冲突用SLR(1)方法无法解决，
所以该文法不是SLR(1)文法。

课堂练习4

已知文法G[A]为：

A→aABela

B→Bbld

(1) 试给出与G[A]等价的LL(1)文法G'[A]

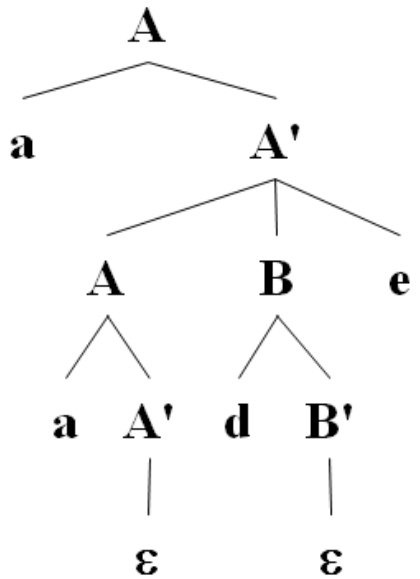
(2) 构造G'[A]的预测分析表

(3) 给出输入串aade的分析过程。

A→aA'
A'→ABe | ?
B→dB'
B'→bB' | ?

	First	Follow
A	a	$, d
A'	a, ?	$, d
B	d	e
B'	b, ?	e

	a	b	d	e	$
A	A→aA'				
A'	A'→ABe		A'→?		A'→?
B			B→dB'		



aade的分析过程

	a	b	d	e	$
A	A→aA'				
A'	A'→ABe		A'→ε		A'→ε
B			B→dB'		
B'		B'→bB'		B'→ε	

步骤	栈	输入	分析动作
(1)	$A	aade$	A□aA'
(2)	$A'a	aade$	
(3)	$A'	ade$	A'□ABe
(4)	$eBA	ade$	A□aA'
(5)	$eBA'a	ade$	
(6)	$eBA'	de$	A'□□
(7)	$eB	de$	B□dB'
(8)	$eB'd	de$	
(9)	$eB'	e$	B'□□
(10)	$e	e$	
(11)	$	$	分析成功

课堂练习5

有如下文法G[A]：

A→BA | a

B→aB | b

(1) 判断该文法是以下哪些类型的文法，要求给出判断过程。

LL(1)、LR(0)、SLR(1)

(2) 构造该文法的LR(1)项目集规范族及识别其所有活前缀的DFA。

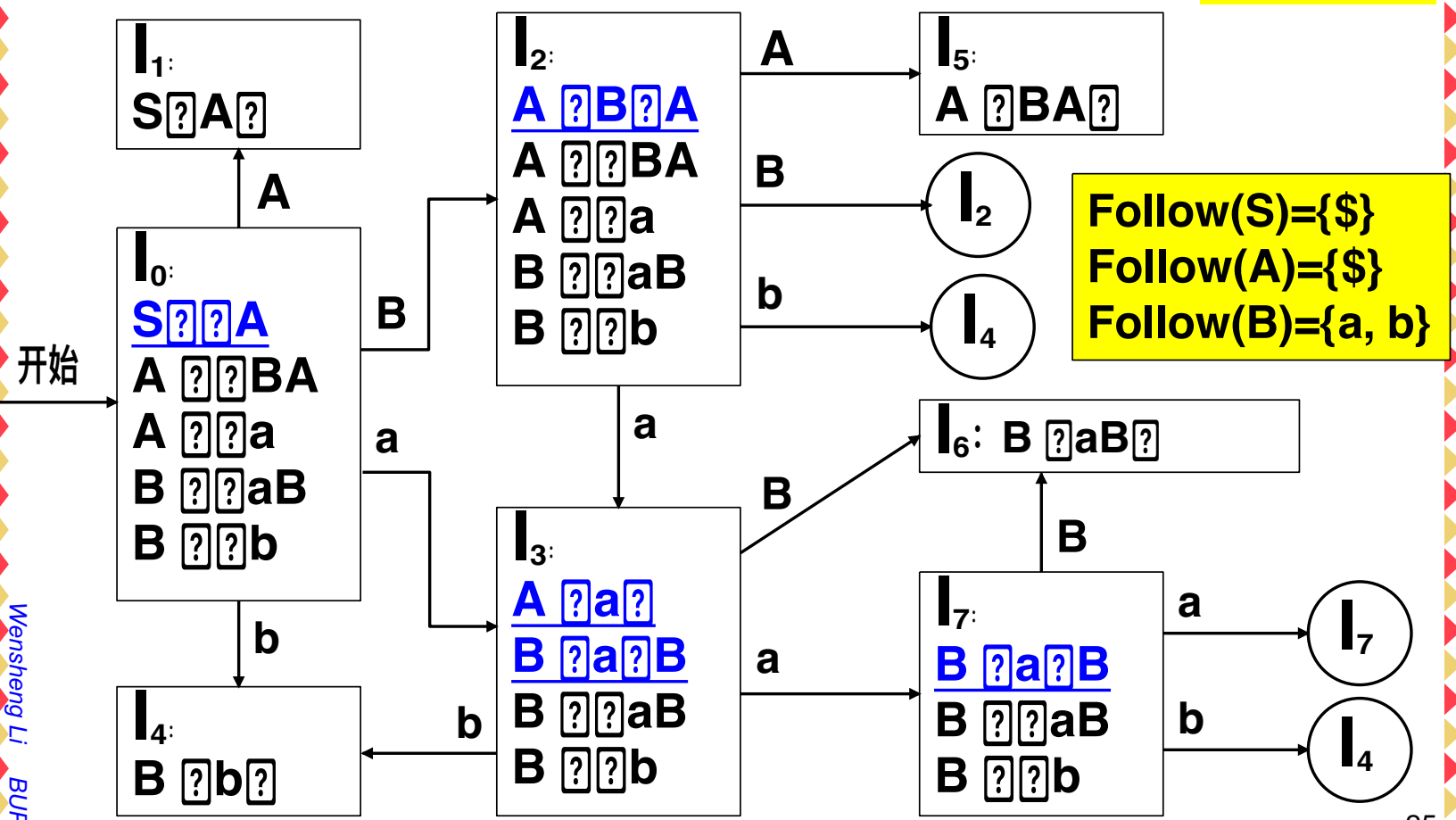
(3) 构造该文法的LR(1)分析表

(4) 给出对输入符号串abb的分析过程。

LR(0)文法，项目集中：
(1) 要么所有元素都是移进-待约项目
(2) 要么只含有唯一的归约项目

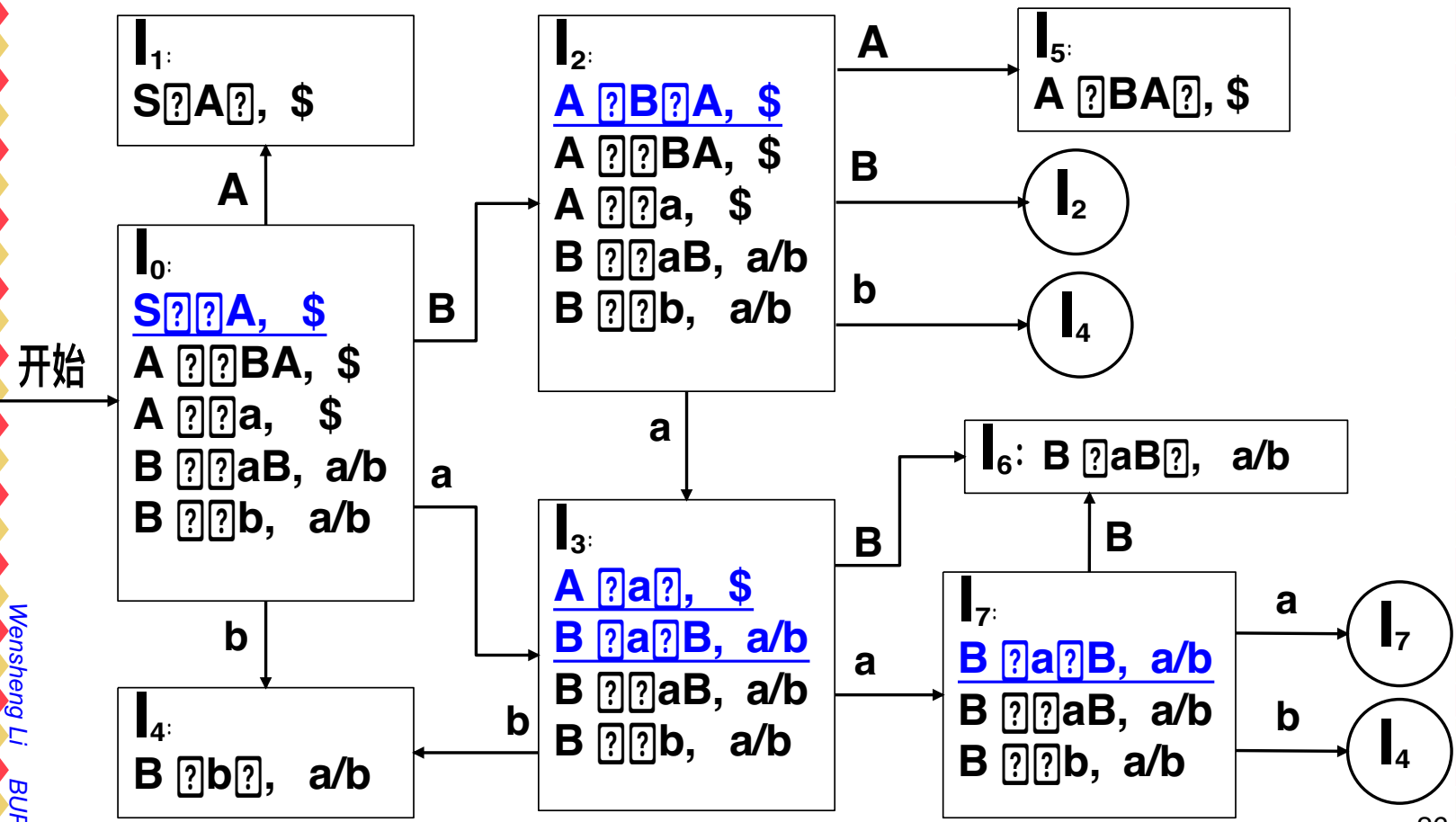
(0) S→A
(1) A→BA
(2) A→a
(3) B→aB
(4) B→b

LR(0)项目集规范族及识别其所有活前缀的DFA：



参考答案（续）

LR(1)项目集规范族及识别其所有活前缀的DFA：



文法的LR(1)分析表

状态	Action			goto	
	a	b	$	A	B
0	S3	S4		1	2
1			ACC		
2	S3	S4		5	2
3	S7	S4			6
4	R4	R4			
5			R1		
6	R3	R3			
7	S7	S4			6

abb的分析过程

步骤	栈	输入	分析动作
(1)	0	abb$	S3
(2)	0 3 - A	bb$	S4
(3)	0 3 4 - a b	b$	R4 B?b
(4)	0 3 6 - a B	b$	R3 B?aB
(5)	0 2 - B	b$	S4
(6)	0 2 4 - B b	$	error 弹出栈顶状态4
(7)	0 2 - B	$	goto(2, A)=5 将状态5压入栈顶
(8)	0 2 5 - B A	$	R1 A?BA
(9)	0 1 - A	$	accept