



中山大學
SUN YAT-SEN UNIVERSITY



国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS²⁹⁰

Compilation Principle 编译原理

第四章 语法分析 (7)

郑馥丹

zhengfd5@mail.sysu.edu.cn

CONTENTS

目录

01

自顶向下分析
Top-Down Parsing

02

LL(1)分析
LL(1) Parsing

03

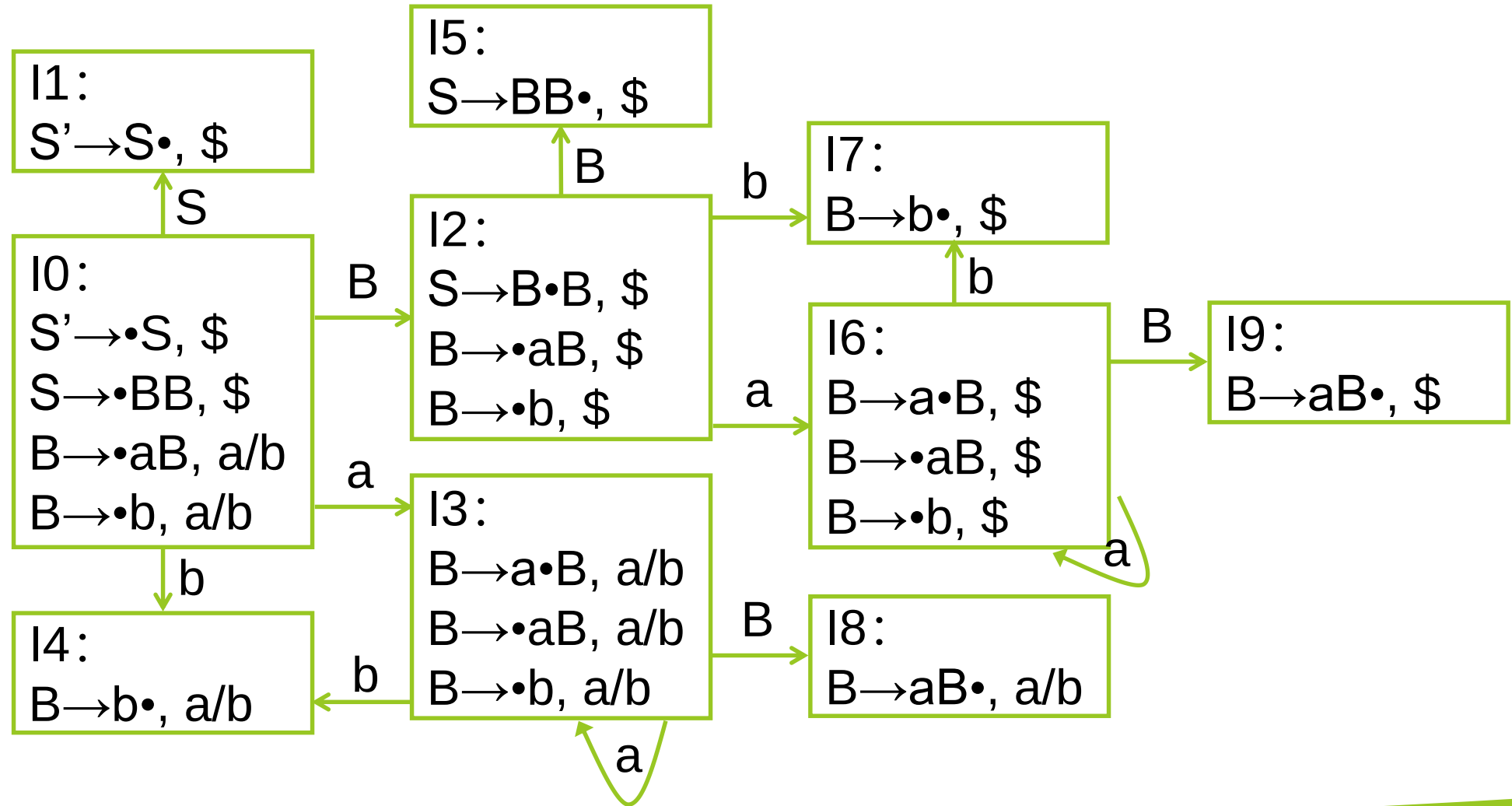
自底向上分析
Bottom-Up Parsing

04

LR分析
LR Parsing

随堂练习 (9)

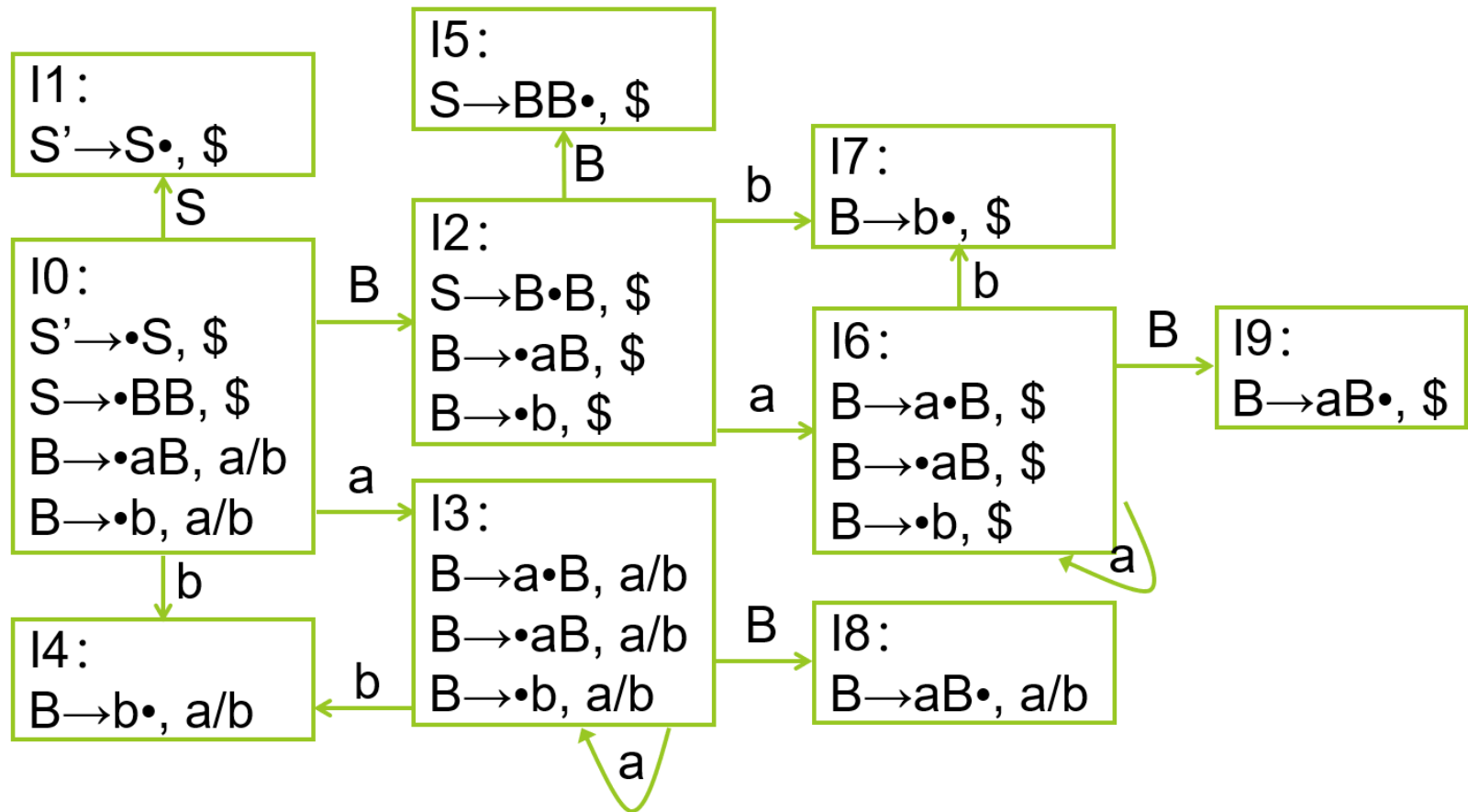
- 若文法 G' 为：(0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$ ，请画出其对应的LR(1)DFA。



4. LR(1)分析

• 构造LR(1)分析表

– G' : (0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$

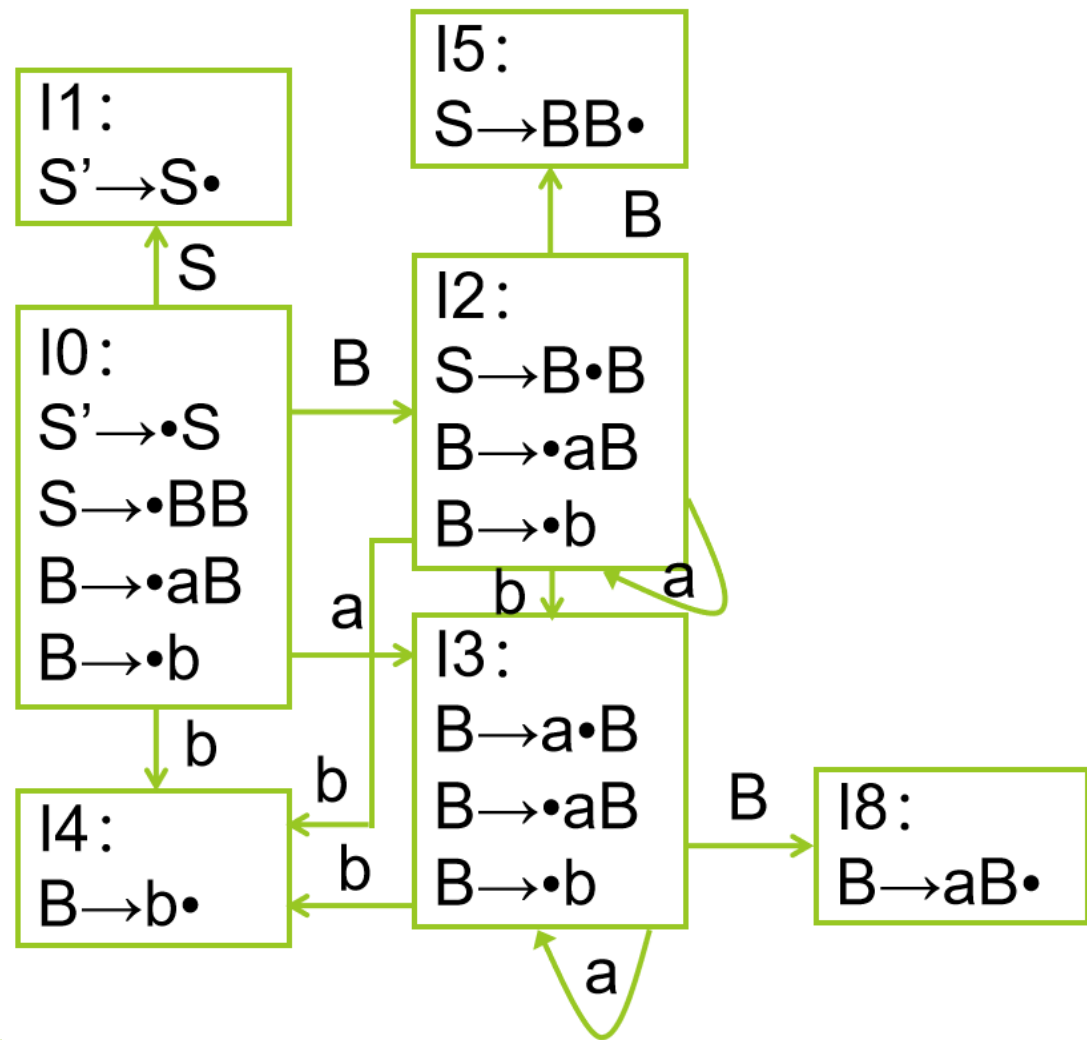


状态	ACTION			GOTO	
	a	b	\$	S	B
0	S ₃	S ₄		1	2
1			acc		
2	S ₆	S ₇			5
3	S ₃	S ₄			8
4	r ₃	r ₃			
5			r ₁		
6	S ₆	S ₇			9
7			r ₃		
8	r ₂	r ₂			
9			r ₂		

发现在该例子中，即使不考查搜索符，也不存在冲突
——实际上是一个LR(0)文法

4. LR(1)分析

- 一个文法是LR(0)文法，就一定也是SLR(1)文法，也是LR(1)文法，反之则不一定成立
- 上述例子的LR(0)项目集，仅有7个状态
- LR(1)的优缺点：
 - 优点：LR(1)分析对搜索符的计算方法比较确切，能消除更多的无效归约，适应的文法更广
 - 缺点：LR(1)分析表的状态数目庞大，可能存在项目集的冗余——**合并, LALR(1)**；LR(1)分析方法也不能解决所有冲突——**LR(k), $k > 2$** 。

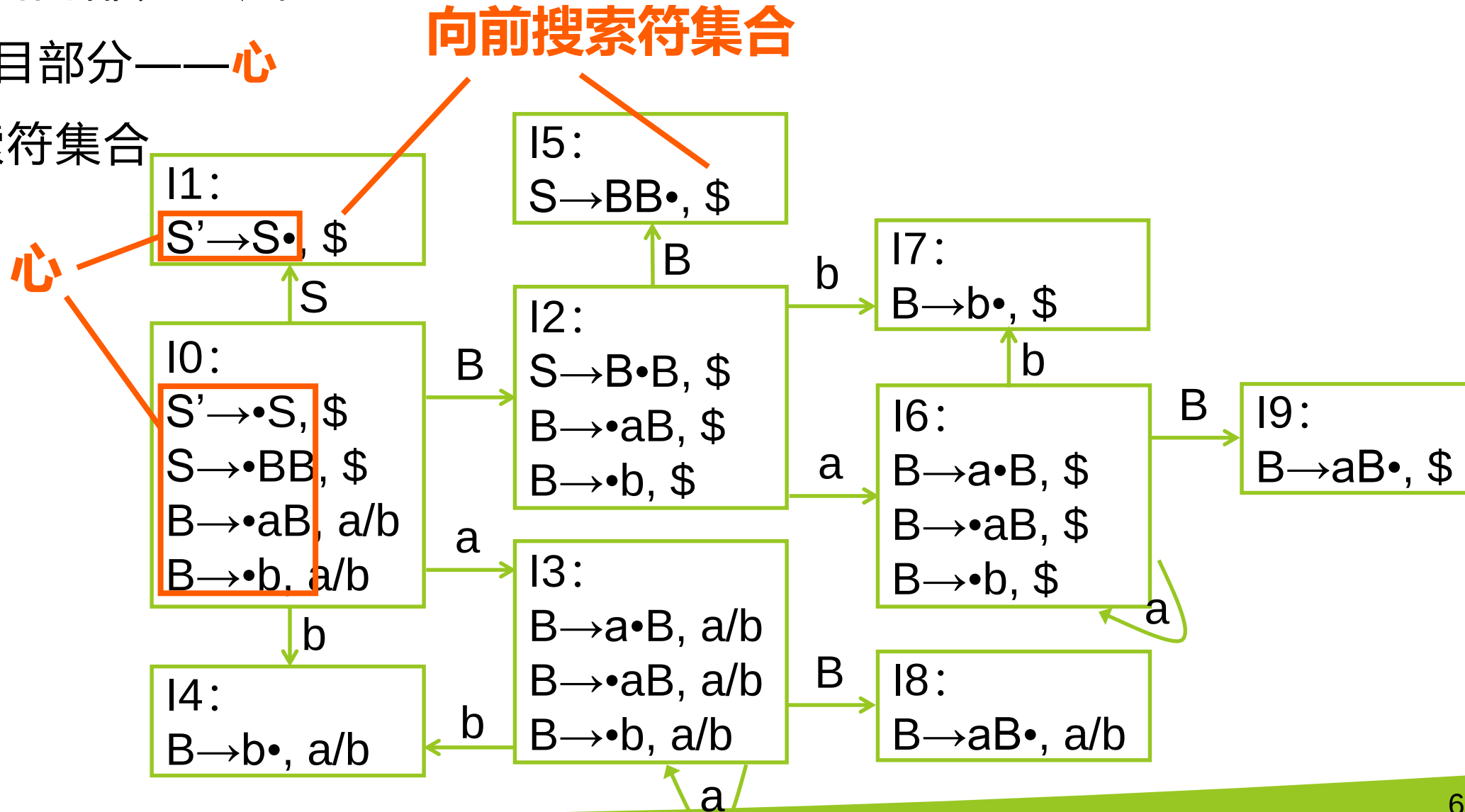


5. LALR(1)分析

- LR(1)项目由两部分组成:

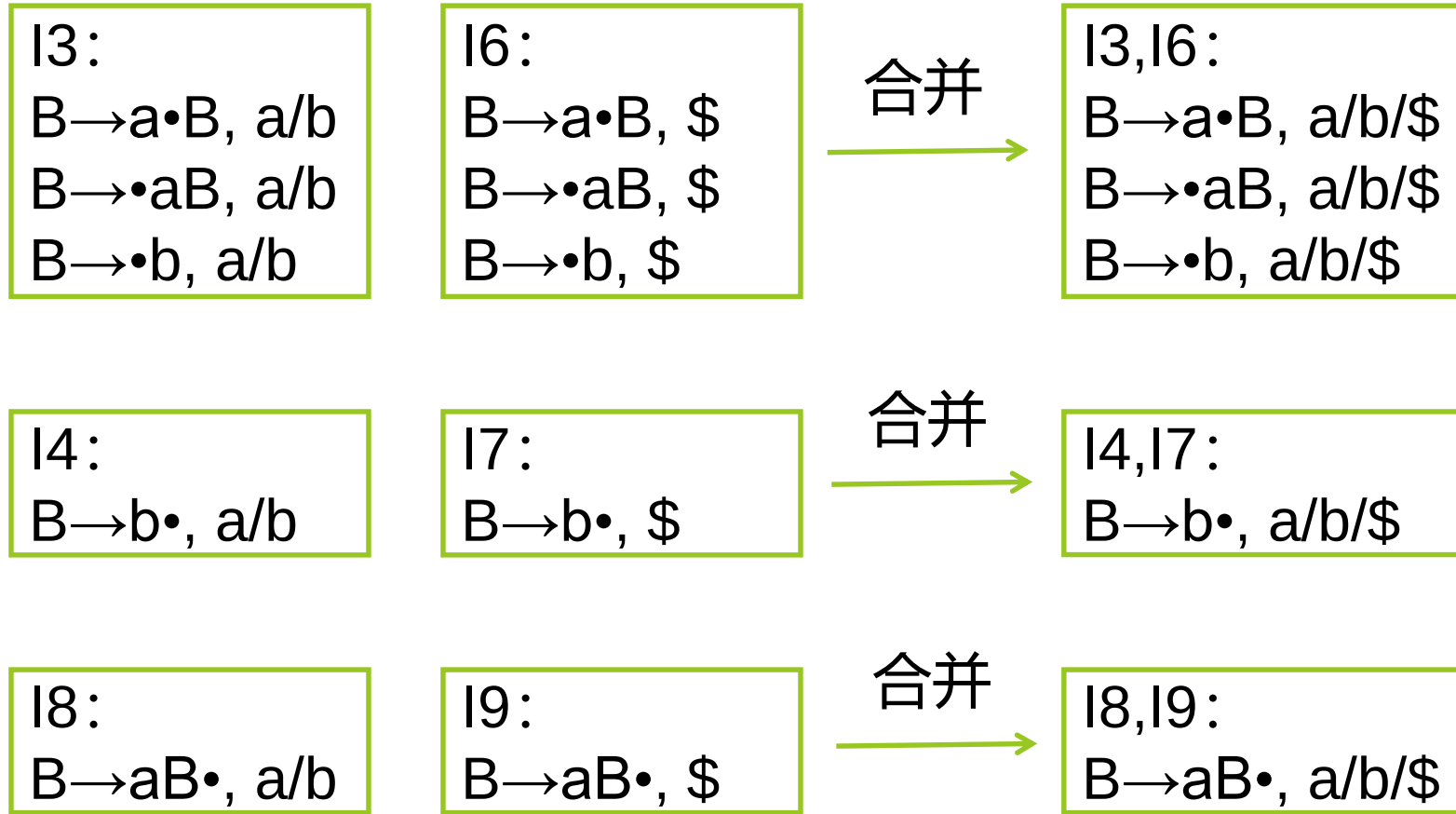
- LR(0)项目部分——心

- 向前搜索符集合



5. LALR(1)分析

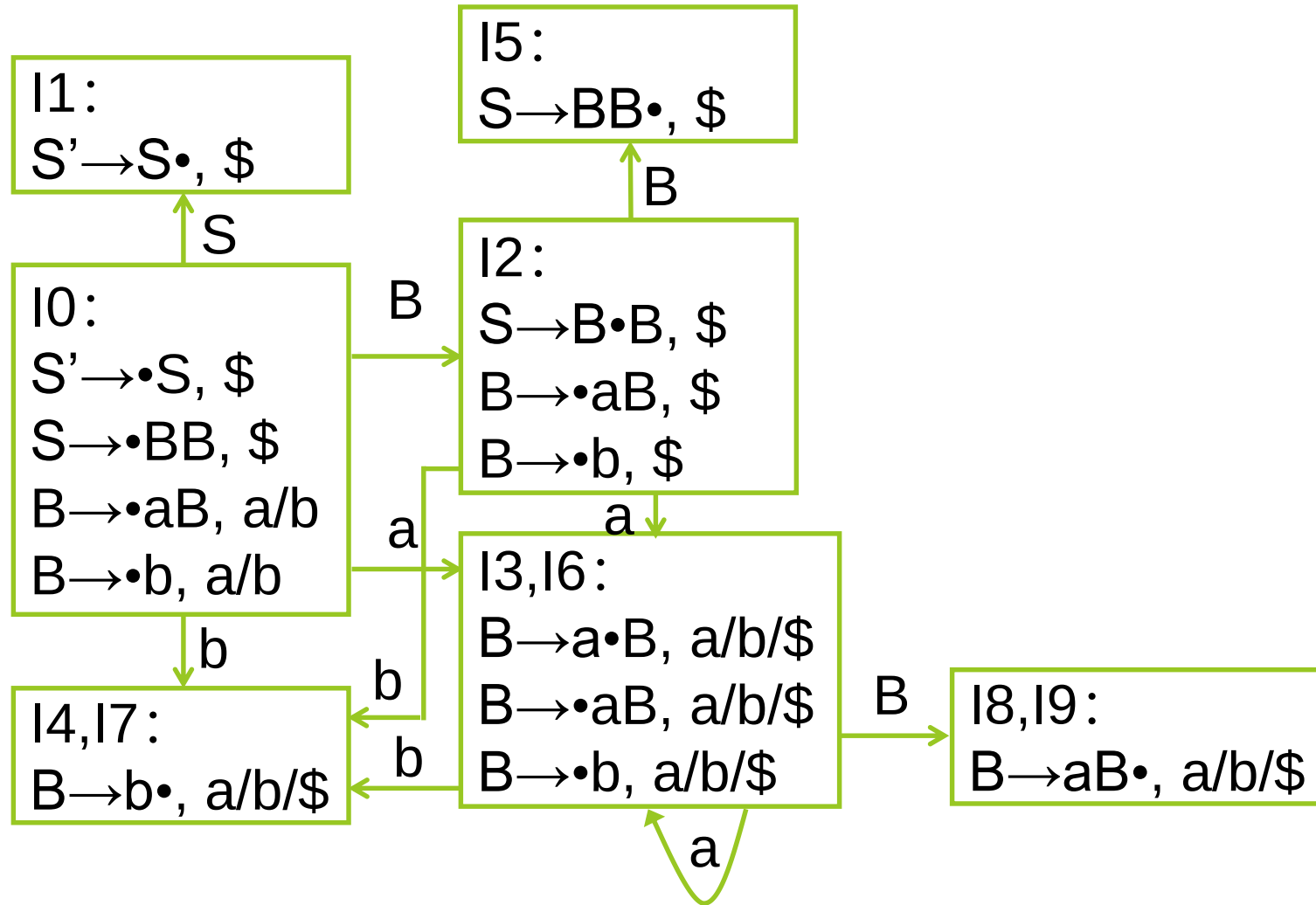
- 对同心的项目集，合并——**合并同心集**



**同心集合并后不包含冲突，是LALR(1)项目集，文法是LALR(1)文法，
可以用LALR(1)方法分析**

5. LALR(1)分析

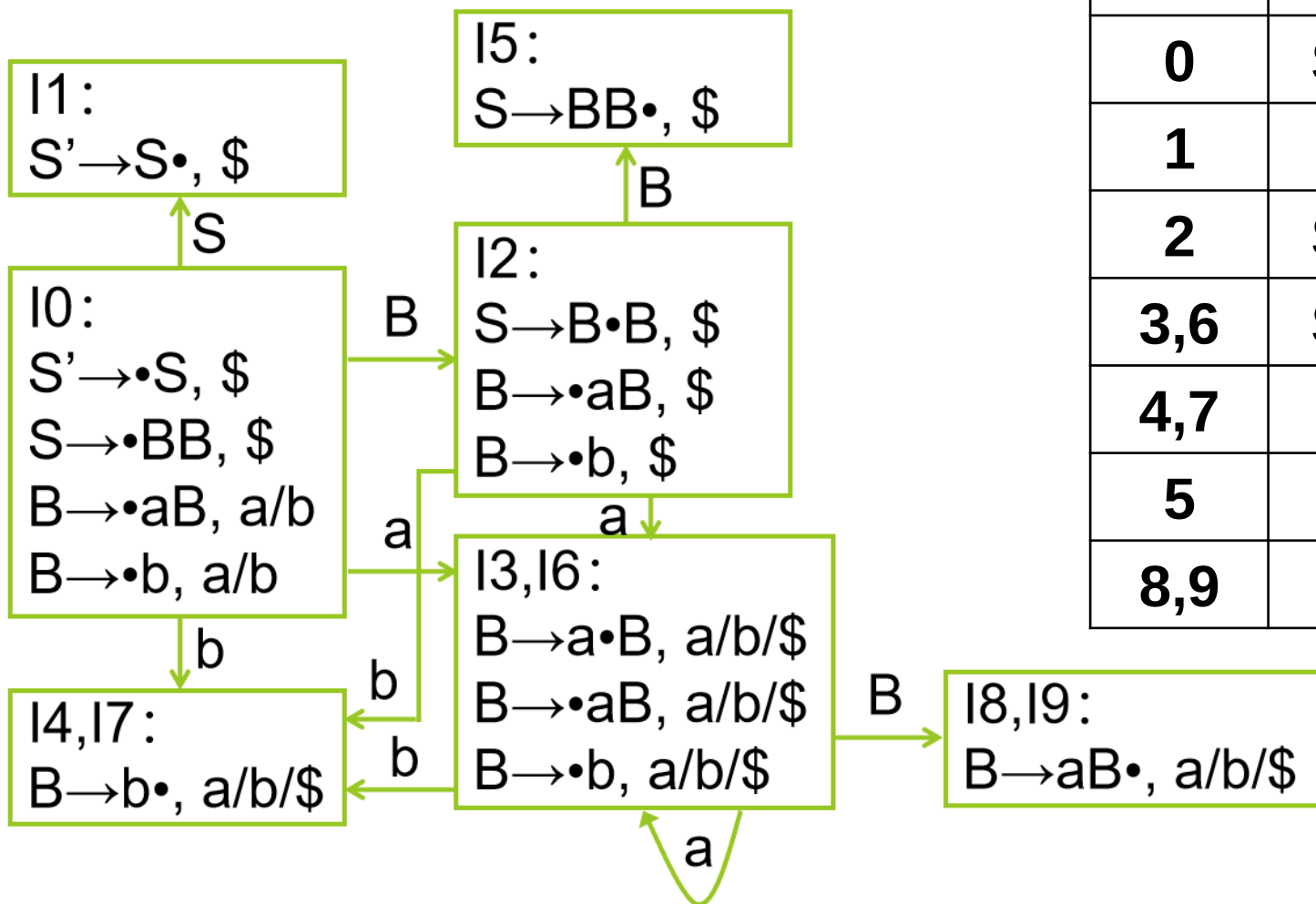
- 构造LALR(1)分析的DFA (合并同心集后)



5. LALR(1)分析

• 构造LALR(1)预测分析表

(0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$



状态	ACTION			GOTO	
	a	b	\$	S	B
0	$S_{3,6}$	$S_{4,7}$		1	2
1			acc		
2	$S_{3,6}$	$S_{4,7}$			5
3,6	$S_{3,6}$	$S_{4,7}$			8,9
4,7	r_3	r_3	r_3		
5			r_1		
8,9	r_2	r_2	r_2		

5. LALR(1)分析

- 利用LALR(1)分析对某些错误发现的时间可能会产生推迟，但错误的出现位置仍是准确的

(0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$

对输入串ab\$

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	\$	ab\$	$S_{3,6}$	
2	0(3,6)	\$a	b\$	$S_{4,7}$	
3	0(3,6)(4,7)	\$ab	\$	r_3	8,9
4	0(3,6)(8,9)	\$aB	\$	r_2	2
5	02	\$B	\$	ERROR	

状态	ACTION			GOTO	
	a	b	\$	S	B
0	$S_{3,6}$	$S_{4,7}$		1	2
1			acc		
2	$S_{3,6}$	$S_{4,7}$			5
3,6	$S_{3,6}$	$S_{4,7}$			8,9
4,7	r_3	r_3	r_3		
5			r_1		
8,9	r_2	r_2	r_2		

第5步才发现错误!

5. LALR(1)分析

- 若利用LR(1)分析表进行分析：

(0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$

对输入串ab\$

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	\$	ab\$	S_3	
2	03	\$a	b\$	S_4	
3	034	\$ab	\$	ERROR	

第3步即发现错误！

原因：LALR(1)分析合并同心集后，向前搜索符集合扩大了，因此推迟发现错误

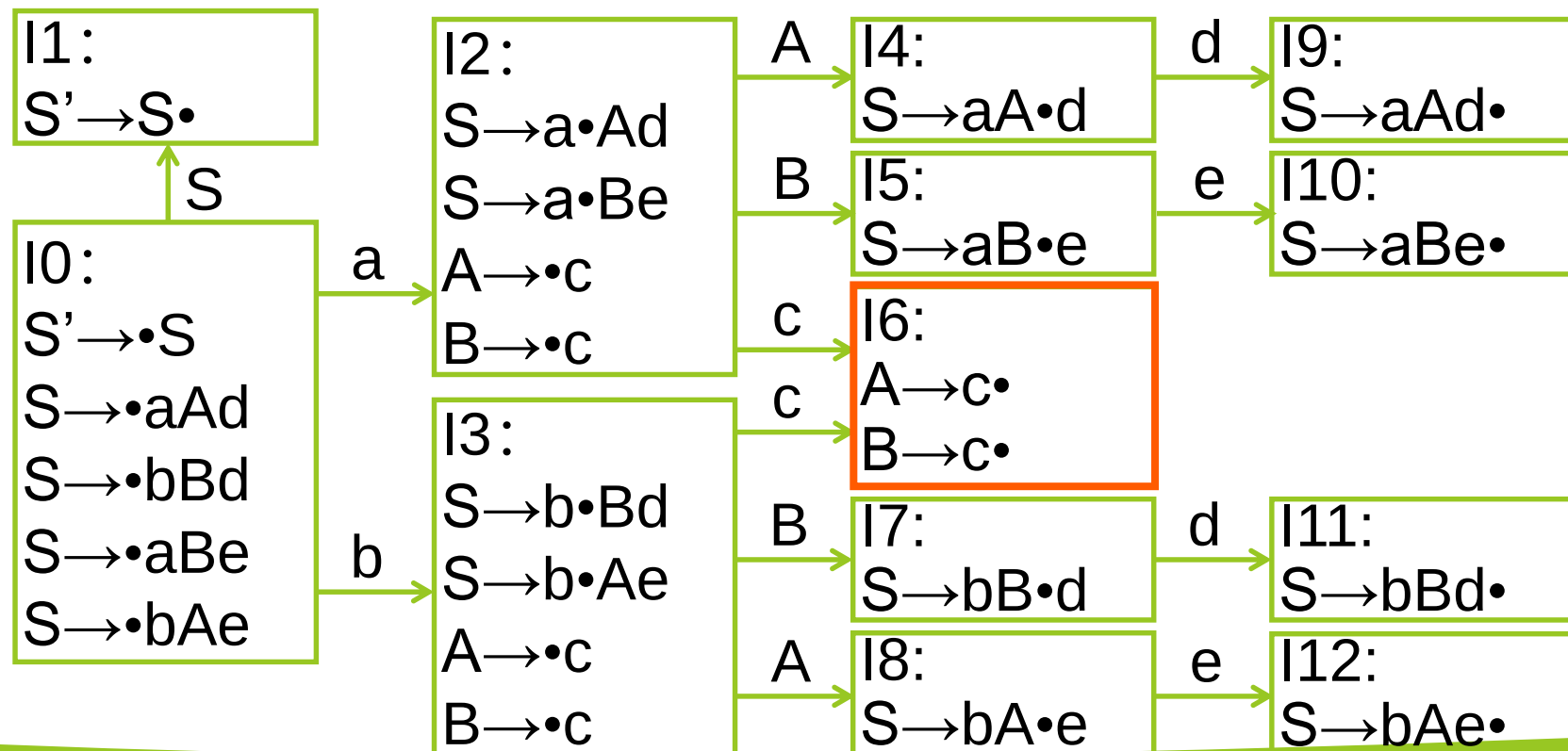
状态	ACTION			GOTO	
	a	b	\$	S	B
0	S_3	S_4		1	2
1			acc		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

随堂练习 (10)

- 请验证以下文法是LR(0)文法或SLR(1)文法或LR(1)文法或LALR(1)文法
文法 $G'[S']$:

(0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bBd$ (3) $S \rightarrow aBe$ (4) $S \rightarrow bAe$ (5) $A \rightarrow c$ (6) $B \rightarrow c$

1. 验证是否为LR(0)文法:



存在归约-归约冲突
不是LR(0)文法

随堂练习 (10)

- 请验证以下文法是LR(0)文法或SLR(1)文法或LR(1)文法或LALR(1)文法
文法 $G'[S']$:

(0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bBd$ (3) $S \rightarrow aBe$ (4) $S \rightarrow bAe$ (5) $A \rightarrow c$ (6) $B \rightarrow c$

2. 验证是否为SLR(1)文法:

I6:
 $A \rightarrow c \bullet$
 $B \rightarrow c \bullet$

FOLLOW(A) = {d, e}

FOLLOW(B) = {d, e}

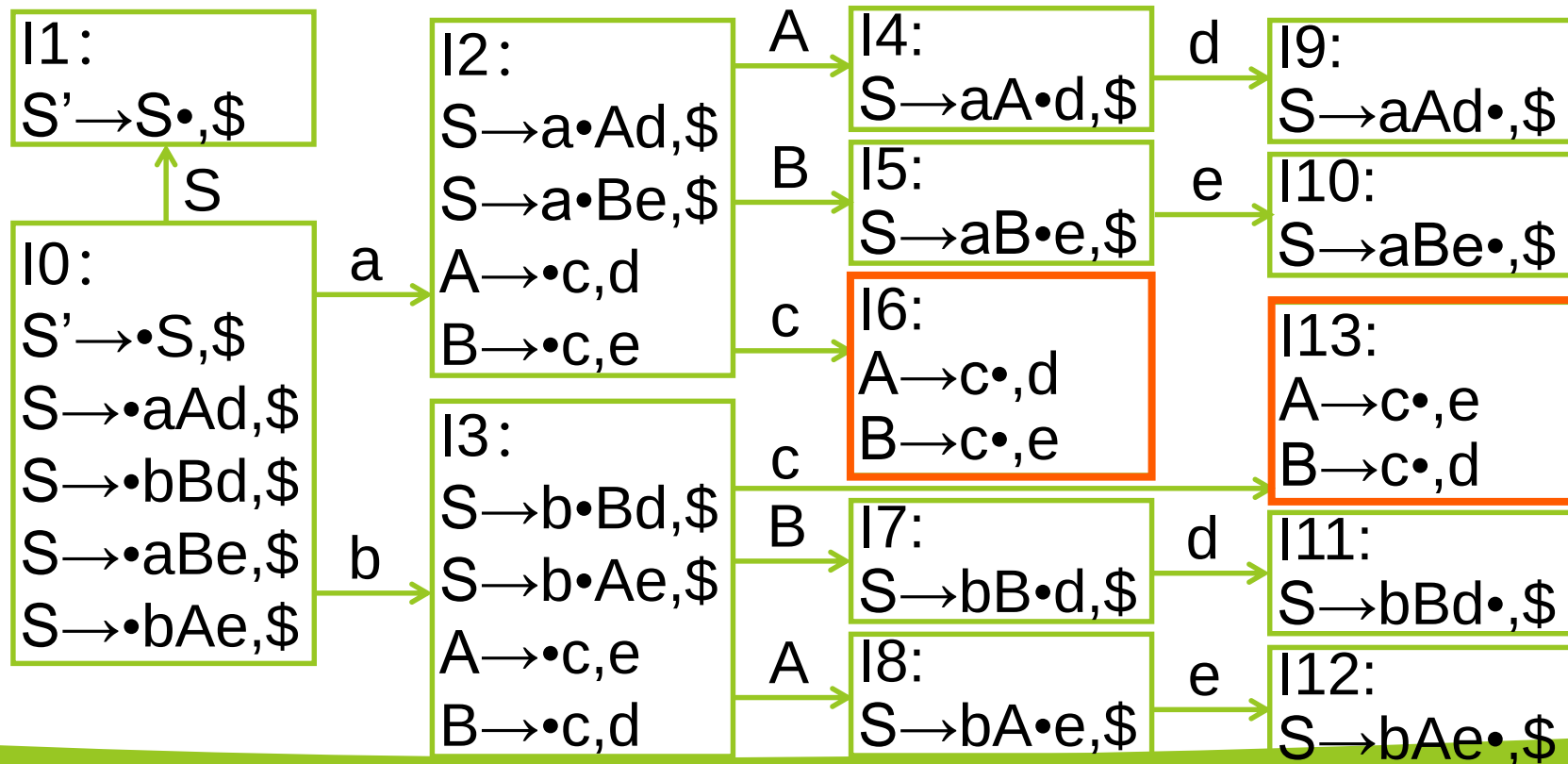
**FOLLOW(A) = {c, d} 与 FOLLOW(B) = {d, e} 交集不为空
， SLR(1) 仍无法解决冲突， 不是SLR(1)文法。**

随堂练习 (10)

- 请验证以下文法是LR(0)文法或SLR(1)文法或LR(1)文法或LALR(1)文法
文法 $G'[S']$:

(0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bBd$ (3) $S \rightarrow aBe$ (4) $S \rightarrow bAe$ (5) $A \rightarrow c$ (6) $B \rightarrow c$

3. 验证是否为LR(1)文法:



归约-归约冲突得以解决, 是LR(1)文法

随堂练习 (10)

- 请验证以下文法是LR(0)文法或SLR(1)文法或LR(1)文法或LALR(1)文法
文法 $G'[S']$:

(0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bBd$ (3) $S \rightarrow aBe$ (4) $S \rightarrow bAe$ (5) $A \rightarrow c$ (6) $B \rightarrow c$

4. 可再进一步验证是否为LALR(1)文法:



合并同心集后出现归约-归约冲突，不是LALR(1)文法

结论：该文法是LR(1)文法，非LR(0)、SLR(1)和LALR(1)文法。

5. LALR(1)分析

- LALR(1)分析

- LALR(1)是LR(1)的优化版本，对LR(1)项目集规范族**合并同心集**，若合并同心集后不产生新的冲突，则为LALR(1)项目集
- LALR(1)分析可以大大减少项目集（状态）的数目。
- LALR(1)方法是介于SLR(1)和LR(1)之间的一种方法，即其功能比LR(0)和SLR(1)强，比LR(1)弱。
- 它具有LR(0)和SLR(1)的状态数少的优点和LR(1)的适用范围广的优点。
- 能处理大多数编程语言语法
- Yacc/Bison等工具默认使用它

LR(0)/SLR(1)/LALR(1)/LR(1)小结

• LR(0):

- 无需向前查看任何输入字符，仅依赖当前状态即可归约；
- 分析表中一整行均为 r_j ；
- 如果存在冲突，即，同一个项目集中既有归约项目，又有移进项目，或存在多条归约项目，则可尝试采用SLR(1)方法解决；

状态	ACTION					GOTO		
	a	b	c	d	\$	E	A	B
0	S_2	S_3				1		
1					acc			
2			S_5	S_6			4	
3			S_8	S_9				7
4	r_1	r_1	r_1	r_1	r_1			
5			S_5	S_6			10	

I_3 $S \rightarrow rD \bullet$
 $D \rightarrow D \bullet, i$

移进-归约冲突

LR(0)/SLR(1)/LALR(1)/LR(1)小结

• SLR(1):

- 在LR(0)的基础上, 其他不变, 仅当**当前输入字符** $\in \text{FOLLOW}(A)$ 时才归约;
- 分析表中只有 $\text{FOLLOW}(A)$ 中字符所在的列填写 r_j ; 比LR(0)更精确;
- **当移进符号集和归约符号集无交集时, 可解决LR(0)冲突;**
- 否则, 仍存在冲突, 可尝试采用LR(1)方法解决。

I3:
 $S \rightarrow rD \bullet$
 $D \rightarrow D \bullet, i$

$\text{FOLLOW}(S) = \{\$ \}$

当前输入符为\$时, 进行归约;
 当前输入符为*i*时, 进行移进操作;
 冲突得以解决。

I5:
 $S \rightarrow ae \bullet c$
 $A \rightarrow e \bullet$

$\text{FOLLOW}(A) = \{c, d\}$

$\text{FOLLOW}(A) = \{c, d\}$ 与移进符号集 $\{c\}$ 交集不为空, SLR(1)仍无法解决冲突。

LR(0)/SLR(1)/LALR(1)/LR(1)小结

• LR(1):

- 在LR(0)的基础上, 每个项目**显式**增加一个**向前搜索符号集** $[A \rightarrow \alpha \bullet \beta, a]$;
- 该符号集由**FIRST集**计算而得, 或由信息传递而得;
 - ✓ 若有项目 $[A \rightarrow \alpha \bullet B \beta, a]$ 属于 $CLOSURE(I)$, $B \rightarrow \gamma$ 是文法的产生式, $\beta \in V^*$, $b \in FIRST(\beta a)$, 则 $[B \rightarrow \bullet \gamma, b]$ 也属于 $CLOSURE(I)$
- 分析表中只有向前搜索符号集所在列填写 r_j , 比SLR(1)更精确;
- 可解决绝大部分冲突;
- 但也无法保证解决所有冲突, 若仍有冲突, 则应考虑LR(k), $k \geq 2$ 。

I5:

 $S \rightarrow ae \bullet c, \$$ $A \rightarrow e \bullet, d$

遇到输入字符c时, 移进;

遇到输入字符d时, 用产生式 $A \rightarrow e$ 归约;

冲突得以解决。

LR(0)/SLR(1)/LALR(1)/LR(1)小结

- LALR(1):
 - 如果LR(1)的项目集中有同心集，则合并同心集；
 - 若合并后不存在归约-归约冲突，则可进行LALR(1)分析；
 - LALR(1)分析基于合并后的项目集进行；
 - 若合并后有冲突，则该文法只能用LR(1)方法进行分析；
 - 比LR(1)弱，比SLR(1)强，是两者的trade-off。



**同心集合并后不包含冲突，是LALR(1)项目集，文法是LALR(1)文法，
可以用LALR(1)方法分析**

LR(0)/SLR(1)/LALR(1)/LR(1)小结

• 主要区别：归约的判定方法

- LR(0)不看下一个字符即可判定归约；
- SLR(1)看下一个字符，但把所有 $B \rightarrow \gamma \bullet$ 的向前搜索字符集均定义为Follow(B)，即把符号栈顶上的句柄 γ 归约为B的条件是：向前搜索字符为Follow(B)中的元素；
- LR(1)也是看下一个字符，但归约不同位置上的B时，采用不同的向前搜索字符集，每个项由心与向前搜索符组成，搜索符长度为1；由于LR(1)方法对于归约条件的判定比SLR(1)更精确，可大大减少移入/归约冲突；
- LALR(1): 对LR(1)项目集规范族合并同心集，减少状态个数。

LR(0)/SLR(1)/LALR(1)/LR(1)小结

- 评价
 - LR(0)分析表局限性大，但其构造方法是其他构造方法的基础；
 - SLR(1)分析表虽然不是对所有文法都存在，但这种分析表较易实现又极有使用价值；
 - LR(1)分析表的分析能力最强，能适用于一大类文法，但是，实现代价过高（表过大）；
 - LALR(1)分析表的能力介于SLR(1)和LR(1)之间，实现效率较高。

LR(0)/SLR(1)/LALR(1)/LR(1)小结

方法	核心思想	向前看符号	分析能力	冲突情况	适用场景
LR(0)	基于LR(0)项，无向前看符号，仅依赖当前状态决定动作	无	最弱，只能处理极简单的无冲突文法	极易出现移进-归约冲突	仅用于教学，实际工程极少使用（因冲突太多）
SLR(1)	在LR(0)基础上，用FOLLOW集解决冲突，仅归约时检查FOLLOW符号	FOLLOW集	比LR(0)强，但仍有限	比LR(0)少，但仍有部分冲突	早期简单解析器，现较少使用（因LALR(1)更优）
LALR(1)	合并LR(1)的同心项集	局部向前看符号	比SLR(1)强，接近LR(1)	可能因合并状态引入延迟冲突	最常用（在分析能力和状态数间取得平衡） ，是Yacc/Bison默认的方法
LR(1)	每个项精确记录不同的向前看符号，不合并任何状态	精确向前看符号	最强 ，能处理所有确定性上下文无关文法	几乎没有冲突，除非文法本身有二义性	需要最强分析能力时使用（但状态数大，解析表可能爆炸，一般仅用于理论研究）

LR(0)/SLR(1)/LALR(1)/LR(1)小结

- 思考：句柄哪去了？LR分析的过程中，好像感受不到句柄的存在？
 - LR分析中，句柄仍是核心概念
 - 但它是**隐式存在**的：句柄的识别是通过状态栈和符号栈的动态组合完成的，而非显式标记
 - LR分析器通过DFA跟踪可能的句柄，当你看到“移进-归约”动作时，本质是DFA在识别句柄的边界
 - 当LR分析器按产生式 $A \rightarrow \beta$ 执行“归约”动作时，**栈顶的符号串 β 就是句柄**
 - LR分析表（ACTION/GOTO）预计算了所有可能的句柄识别路径
 - 用户无需手动识别句柄