

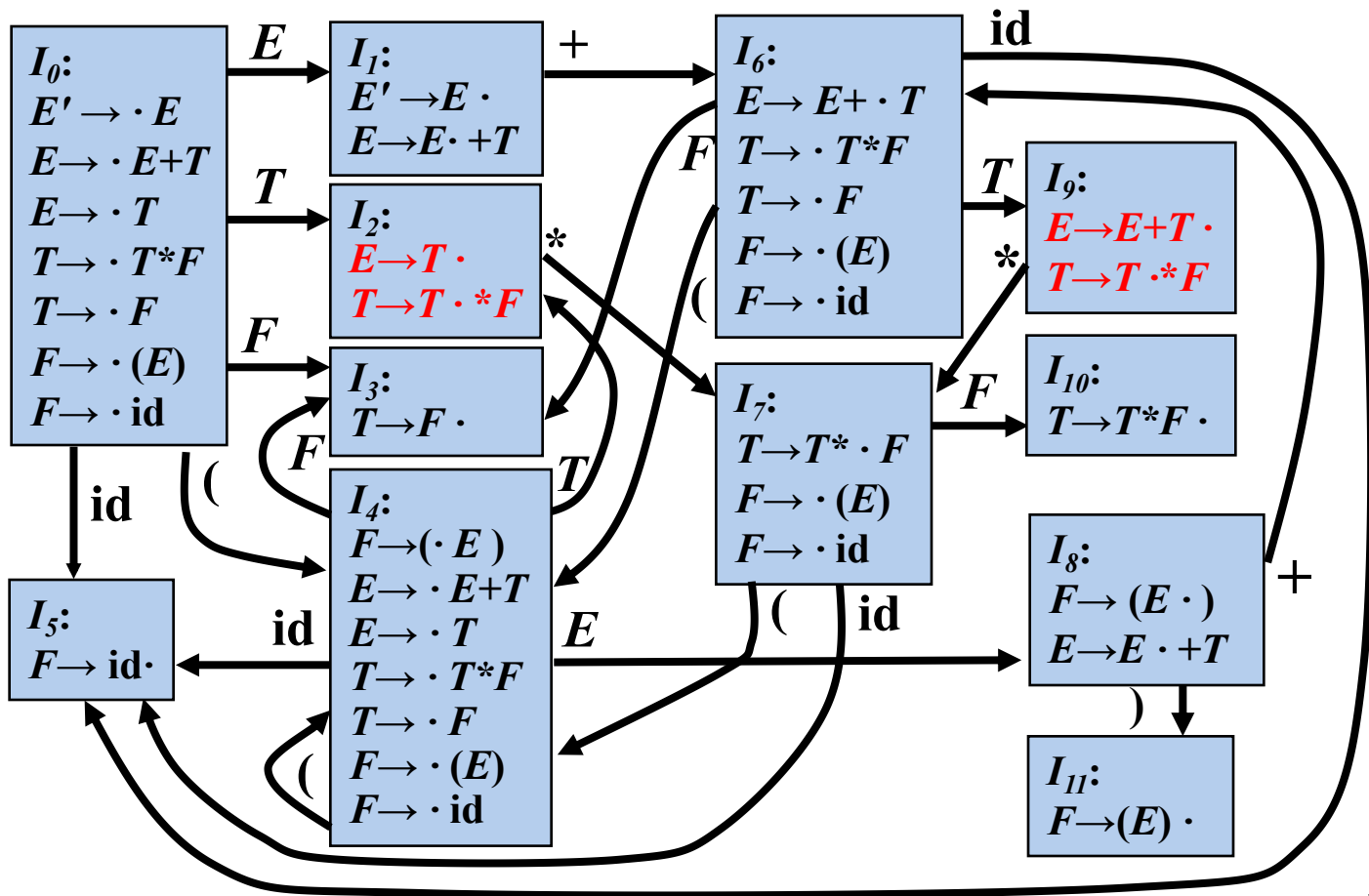
4.4.2 SLR 分析

例：LR(0) 分析过程中的冲突

文法：

- (0) $E' \rightarrow E$
- (1) $E \rightarrow E+T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T*F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

X	$FOLLOW(X)$
E	$), +, \$$
T	$), +, \$, *$
F	$), +, \$, *$



SLR 分析

➤ SLR分析法的基本思想

已知项目集 I :

$A_1 \rightarrow \alpha_1 \cdot a_1 \beta_1$
 $A_2 \rightarrow \alpha_2 \cdot a_2 \beta_2$
...
 $A_m \rightarrow \alpha_m \cdot a_m \beta_m$

} m 个移进项目

$B_1 \rightarrow \gamma_1 \cdot$
 $B_2 \rightarrow \gamma_2 \cdot$
...
 $B_n \rightarrow \gamma_n \cdot$

} n 个归约项目

如果集合 $\{a_1, a_2, \dots, a_m\}$ 和 $FOLLOW(B_1), FOLLOW(B_2), \dots, FOLLOW(B_n)$ 两两不相交, 则项目集 I 中的冲突可以按以下原则解决:

设 a 是下一个输入符号

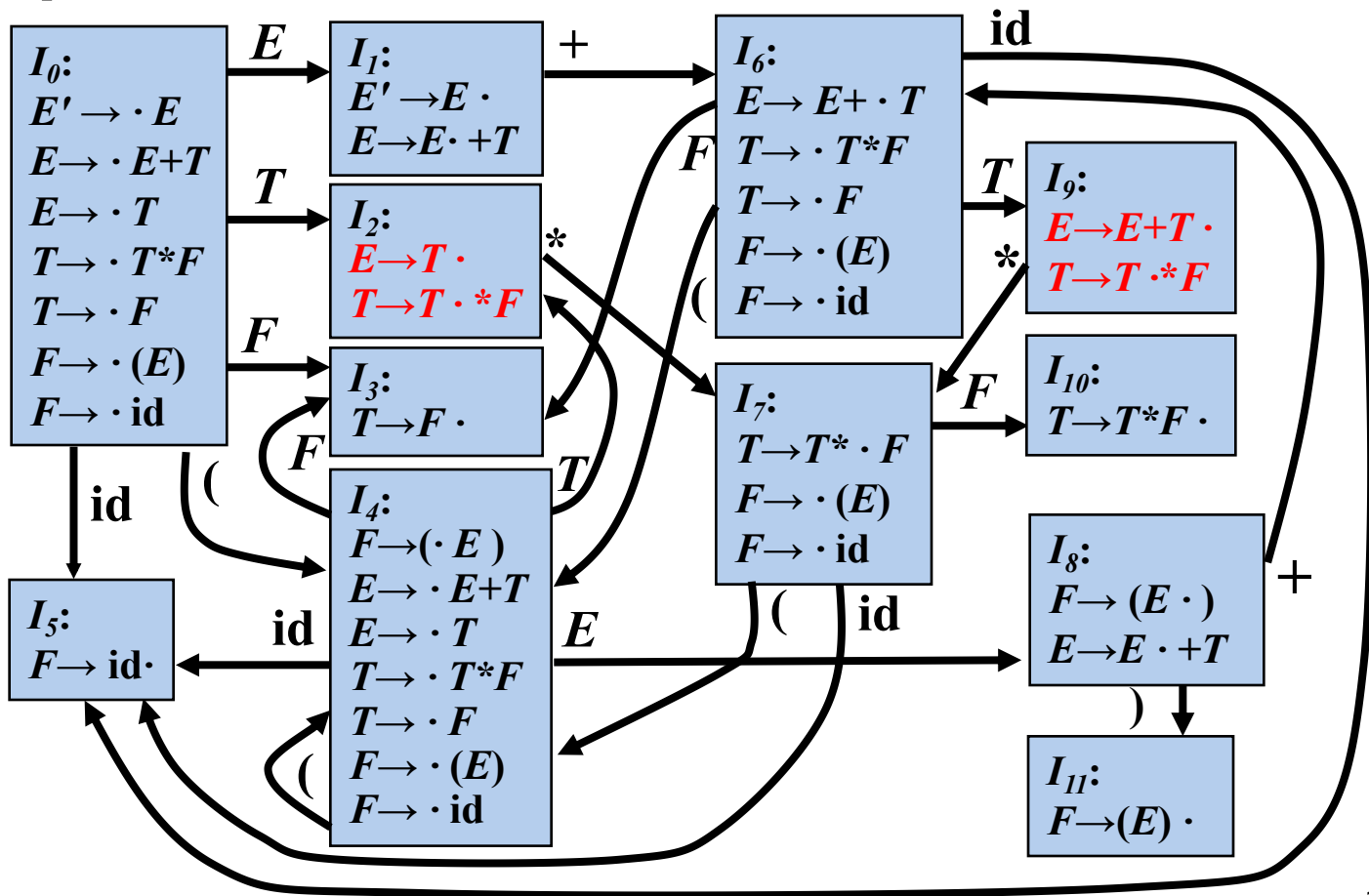
- 若 $a \in \{a_1, a_2, \dots, a_m\}$, 则移进 a
- 若 $a \in FOLLOW(B_i)$, 则用产生式 $B_i \rightarrow \gamma_i$ 归约
- 此外, 报错

例：LR(0) 分析过程中的冲突

文法：

- (0) $E' \rightarrow E$
- (1) $E \rightarrow E+T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T*F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

X	$FOLLOW(X)$
E	$), +, \$$
T	$), +, \$, *$
F	$), +, \$, *$



表达式文法的SLR分析表

文法:

- (0) $E' \rightarrow E$
- (1) $E \rightarrow E+T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T*F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow \text{id}$

状态	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

X	FOLLOW(X)
E), +, \$
T), +, \$, *
F), +, \$, *

例

文法

(0) $S' \rightarrow T$

(1) $T \rightarrow aBd$

(2) $T \rightarrow \varepsilon$

(3) $B \rightarrow Tb$

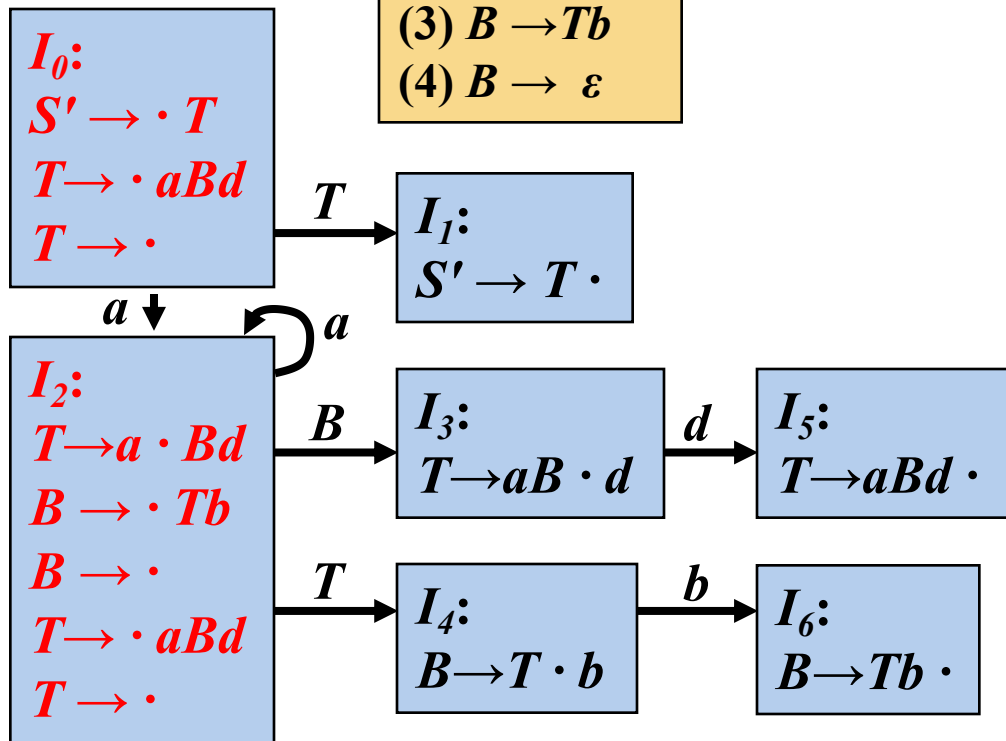
(4) $B \rightarrow \varepsilon$

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

$FOLLOW(T) = \{ \$, b \}$

$FOLLOW(B) = \{ d \}$

SLR(1)分析表



状态	ACTION				GOTO	
	a	b	d	\$	T	B
0	s2	r2		r2	1	
1				acc		
2	s2	r2	r4	r2	4	3
3			s5			
4		s6				
5		r1		r1		
6			r3			

SLR 分析表构造算法

- 构造 G' 的规范LR(0)项集族 $C = \{ I_0, I_1, \dots, I_n \}$ 。
- 根据 I_i 构造得到状态 i 。状态 i 的语法分析动作按照下面的方法决定：
 - **if** $A \rightarrow \alpha \cdot a \beta \in I_i$ and $GOTO(I_i, a) = I_j$ **then** $ACTION[i, a] = sj$
 - **if** $A \rightarrow \alpha \cdot B \beta \in I_i$ and $GOTO(I_i, B) = I_j$ **then** $GOTO[i, B] = j$
 - **if** $A \rightarrow \alpha \cdot \in I_i$ 且 $A \neq S'$ **then** for $\forall a \in FOLLOW(A)$ **do**
 $ACTION[i, a] = rj$ (j 是产生式 $A \rightarrow \alpha$ 的编号)
 - **if** $S' \rightarrow S \cdot \in I_i$ **then** $ACTION[i, \$] = acc$;
- 没有定义的所有条目都设置为“error”。

与LR(0)分析表不同

如果给定文法的SLR分析表中不存在有冲突的动作，那么该文法称为**SLR文法**

课后练习

- 画出文法的LR(0)自动机，并画出SLR分析表，说明是不是SLR文法，为什么？

$$S \rightarrow a \mid S + S \mid S S \mid S^* \mid (S)$$

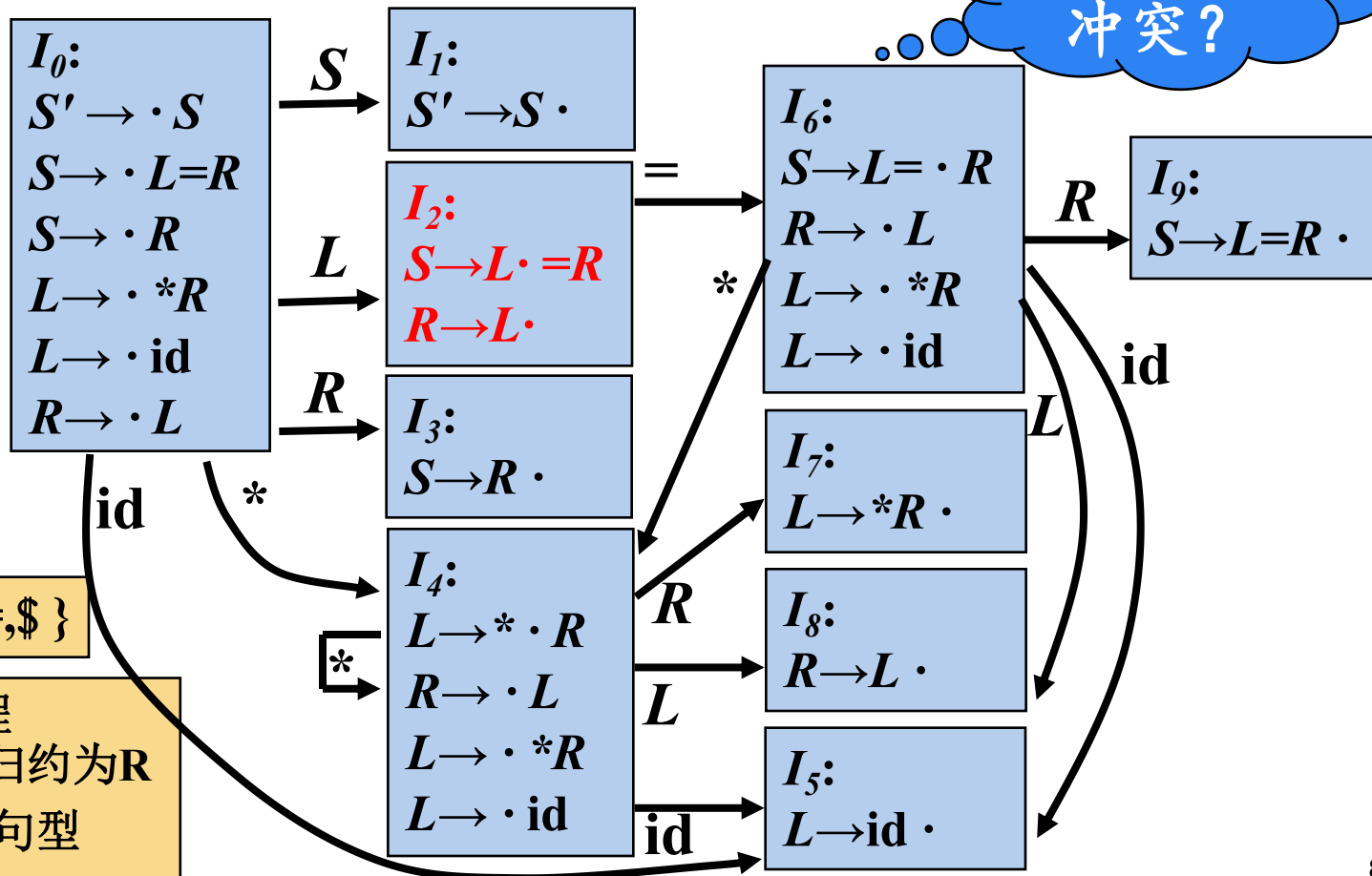
SLR 分析中的冲突

例

- 0) $S' \rightarrow S$
- 1) $S \rightarrow L=R$
- 2) $S \rightarrow R$
- 3) $L \rightarrow *R$
- 4) $L \rightarrow id$
- 5) $R \rightarrow L$

$FOLLOW(R) = \{ =, \$ \}$

例: $id = id$ 归约过程
 $\Leftarrow L=id$ L 不能再归约为 R
 rm
因为 $R=id$ 不是规范句型



4.4.3 LR(1)分析

➤SLR分析存在的问题

- SLR只是简单地考察下一个输入符号 b 是否属于与归约项目 $A \rightarrow \alpha$ 相关联的 $FOLLOW(A)$ ，但并不是只要 $b \in FOLLOW(A)$ 就可以归约的，还要考虑 α 出现的上下文。
- 在特定的上下文中， A 的后继符号的集合是 $FOLLOW(A)$ 的子集，因此，并不是在任何情况下，都能归约为 A 的。

LR(1)分析法的提出

- 对于产生式 $A \rightarrow \alpha$ 的归约， α 出现在不同的上下文中， α 是否可以归约为 A 需要参考不同的后继符号

0) $S' \rightarrow S$

1) $S \rightarrow L = R$

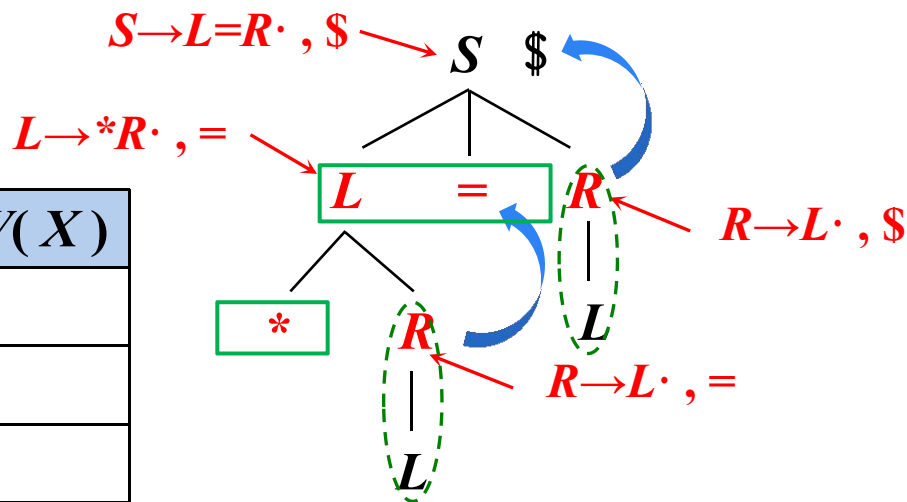
2) $S \rightarrow R$

3) $L \rightarrow *R$

4) $L \rightarrow \text{id}$

5) $R \rightarrow L$

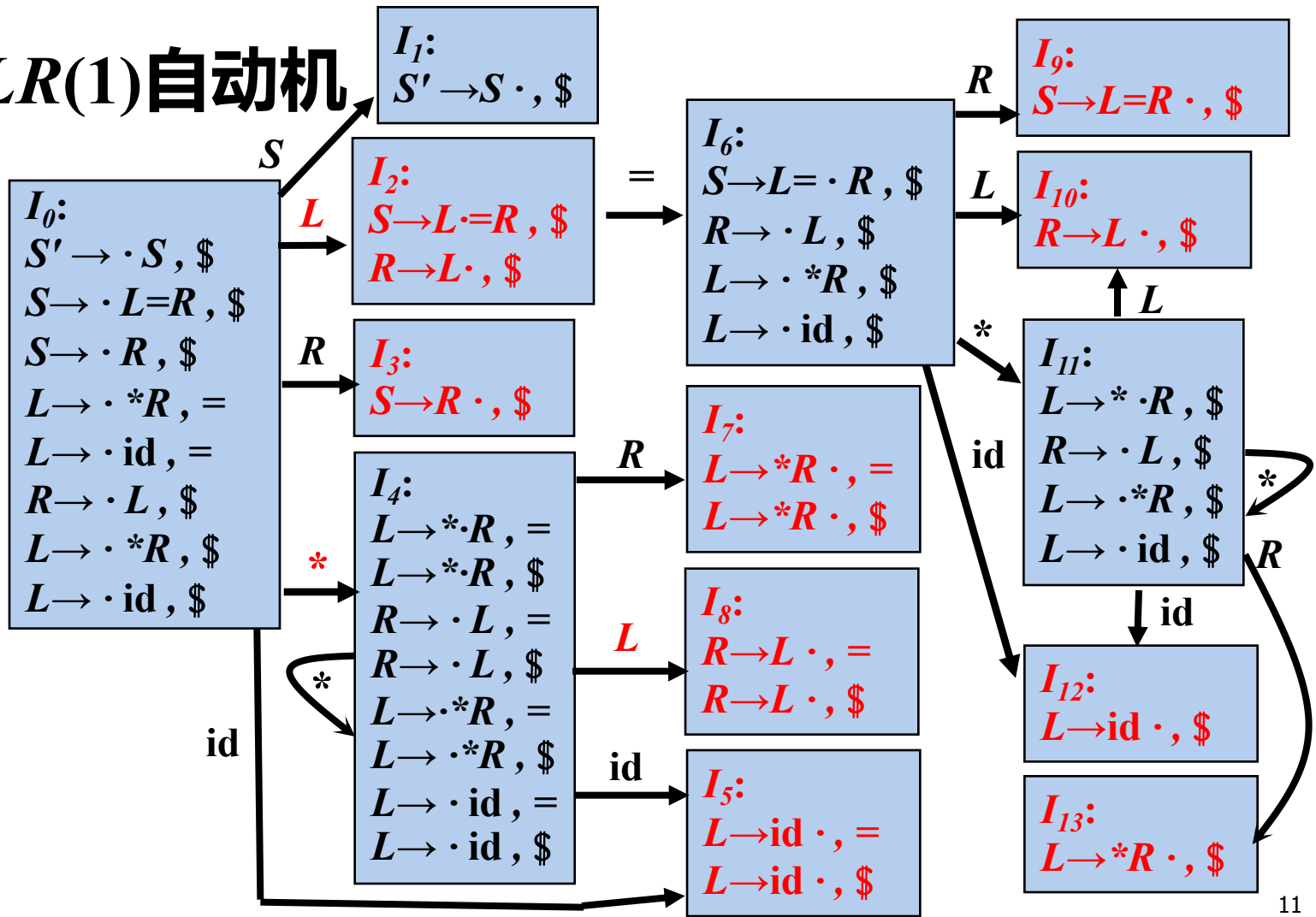
X	$FOLLOW(X)$
S	$\$$
L	$=, \$$
R	$=, \$$



- 在特定上下文中， A 的后继符集合是 $FOLLOW(A)$ 的子集

例: $LR(1)$ 自动机

- 0) $S' \rightarrow S$
- 1) $S \rightarrow L=R$
- 2) $S \rightarrow R$
- 3) $L \rightarrow *R$
- 4) $L \rightarrow id$
- 5) $R \rightarrow L$



规范LR(1)项目

形如 $[A \rightarrow \alpha \cdot \beta, a]$ 的项称为 **LR(1) 项**

其中 $A \rightarrow \alpha \beta \in P$, $a \in V_T \cup \{\$ \}$, a 称为该项的**展望符**(*lookahead*, 向前看符号), 也称为**搜索符**。

- 对于形如 $[A \rightarrow \alpha \cdot, a]$ 的归约项, 当前输入符号为 a 时才能归约
- 对于形如 $[A \rightarrow \alpha \cdot \beta, a]$ 且 $\beta \neq \varepsilon$ 的非归约项, 展望符 a 不起作用
- 具有相同第一分量的所有项的展望符 a 的集合总是 $FOLLOW(A)$ 的子集, 而且通常是真子集, A 为对应产生式的左部。
- LR(1) 中的 1 指的是项的第二个分量的长度, 也是归约时向前看符号的个数。

LR(1)项目展望符的确定

1、初始项目展望符是:\$
[$S' \rightarrow \cdot S$, \$]

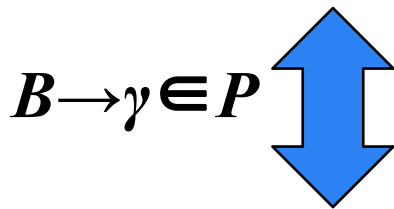
2. 闭包运算中新加入项的展望符
 $FIRST(\beta a)$

3.后继项目继承原项目的展望符

若存在[$A \rightarrow \alpha \cdot X \beta$, **a**] $\in I_i$,
则其后继项目为:

[$A \rightarrow \alpha X \cdot \beta$, **a**]

[$A \rightarrow \alpha \cdot B \beta$, **a**]



$B \rightarrow \gamma \in P$

[$B \rightarrow \cdot \gamma$, **b**]

$b \in FIRST(\beta a)$

赋值语句文法的 $LR(1)$ 分析表

文法

0) $S' \rightarrow S$

1) $S \rightarrow L=R$

2) $S \rightarrow R$

3) $L \rightarrow *R$

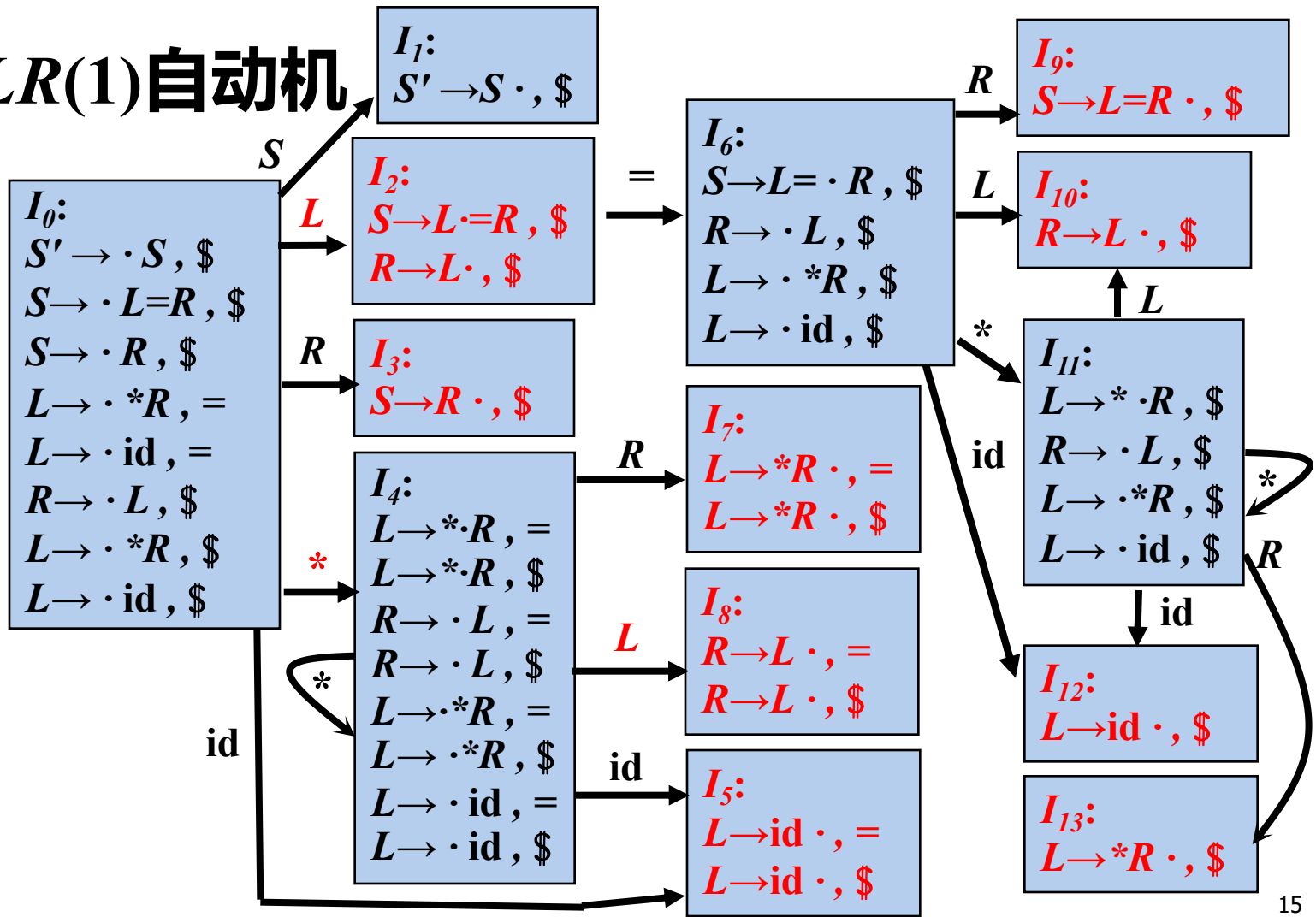
4) $L \rightarrow id$

5) $R \rightarrow L$

状态	ACTION				GOTO		
	*	id	=	\$	S	L	R
0	s4	s5			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s4	s5				8	7
5			r4	r4			
6	s11	s12				10	9
7			r3	r3			
8			r5	r5			
9				r1			
10				r5			
11	s11	s12				10	13
12				r4			
13				r3			

例: $LR(1)$ 自动机

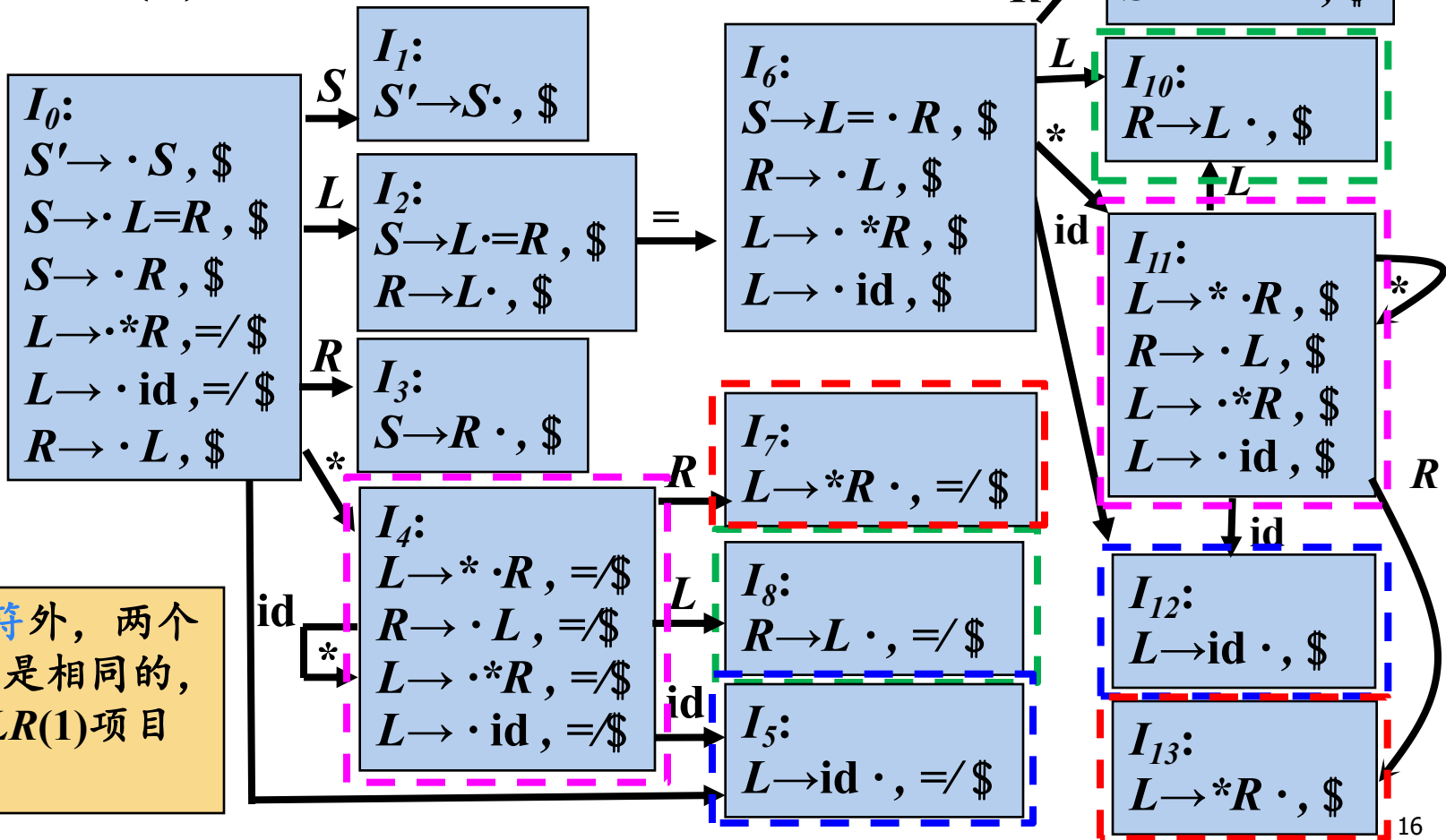
- 0) $S' \rightarrow S$
- 1) $S \rightarrow L=R$
- 2) $S \rightarrow R$
- 3) $L \rightarrow *R$
- 4) $L \rightarrow id$
- 5) $R \rightarrow L$



例: LR(1)自动机

- 0) $S' \rightarrow S$
- 1) $S \rightarrow L=R$
- 2) $S \rightarrow R$
- 3) $L \rightarrow *R$
- 4) $L \rightarrow \text{id}$
- 5) $R \rightarrow L$

如果除展望符外，两个LR(1)项目集是相同的，则称这两个LR(1)项目集是**同心的**



LR分析法与LL(1)分析法

设有规范推导 $S \xRightarrow{rm*} \gamma A b w \xRightarrow{rm} \gamma l \beta b w$

$A \rightarrow l \beta \in P, l \in V_T \cup V_N, \gamma, \beta \in (V_T \cup V_N)^*, b \in V_T, w \in (V_T)^*$

- LL(1)分析：在看见产生式体的第1个符号 l 时就必须决定使用哪个产生式推导。
- LR(1)分析：在看见了产生式体的所有符号 $l \beta$ 后，还要再向后看1个后继符号 b 才决定是否归约，用哪个产生式归约，因此分析能力更强。

LR分析表构造算法——LR(1)项目集闭包

$$CLOSURE(I) = I \cup \{ [B \rightarrow \cdot \gamma, b] \mid \exists [A \rightarrow \alpha \cdot B \beta, a] \in CLOSURE(I), \\ B \rightarrow \gamma \in P, b \in FIRST(\beta a) \}$$

```
SetOfItems CLOSURE ( I ) {  
    repeat  
        for ( I中的每个项[ A → α·Bβ, a ] )  
            for ( G'的每个产生式 B → γ )  
                for ( FIRST(βa)中的每个符号 b )  
                    将[ B → ·γ, b ]加入到集合 I 中;  
    until 不能向 I 中加入更多的项;  
    until I ;  
}
```

LR分析表构造算法——GOTO 函数

$$GOTO(I, X) = CLOSURE(\{[A \rightarrow \alpha X \cdot \beta, a] \mid \forall [A \rightarrow \alpha \cdot X \beta, a] \in I\})$$

```
SetOfItems GOTO ( I, X ) {  
    将J初始化为空集;  
    for ( I 中的每个项  $[A \rightarrow \alpha \cdot X \beta, a]$  )  
        将项  $[A \rightarrow \alpha X \cdot \beta, a]$  加入到集合J 中;  
    return CLOSURE ( J );  
}
```

后继项目继承(传播)展望符

为文法 G' 构造 $LR(1)$ 项集族

```
void items ( $G'$ ) {  
    将 $C$ 初始化为  $\{ \text{CLOSURE} (\{ [ S' \rightarrow \cdot S, \$ ] \} ) \}$  ;  
    repeat  
        for (  $C$  中的每个项集  $I$  )  
            for ( 每个文法符号  $X$  )  
                if (  $\text{GOTO}(I, X)$  非空且不在  $C$  中 )  
                    将  $\text{GOTO}(I, X)$  加入  $C$  中 ;  
    until 不再有新的项集加入到  $C$  中 ;  
}
```

LR(1)自动机的形式化定义

➤ 文法

$$G = (V_N, V_T, P, S)$$

➤ LR(1) 自动机

$$M = (C, V_N \cup V_T, GOTO, I_0, F)$$

$$\text{➤ } C = \{ I_0 \} \cup \{ I \mid \exists J \in C, X \in V_N \cup V_T, I = GOTO(J, X) \}$$

$$\text{➤ } I_0 = CLOSURE(\{ S' \rightarrow \cdot S, \$ \})$$

$$\text{➤ } F = C \text{ (活前缀的识别状态)}$$

语法分析句子的识别状态： $F = \{ \text{包含 } S' \rightarrow S \cdot, \$ \text{ 的状态} \}$

LR(1)分析表构造算法

- 构造 G' 的规范LR(1)项集族 $C = \{ I_0, I_1, \dots, I_n \}$
- 根据 I_i 构造得到状态 i 。状态 i 的语法分析动作按照下面的方法决定：
 - if $[A \rightarrow \alpha \cdot a \beta, b] \in I_i$ and $GOTO(I_i, a) = I_j$ then $ACTION[i, a] = sj$
 - if $[A \rightarrow \alpha \cdot B \beta, b] \in I_i$ and $GOTO(I_i, B) = I_j$ then $GOTO[i, B] = j$
 - if $[A \rightarrow \alpha \cdot, a] \in I_i$ 且 $A \neq S'$ then $ACTION[i, a] = rj$
 - if $[S' \rightarrow S \cdot, \$] \in I_i$ then $ACTION[i, \$] = acc$;
- 没有定义的所有条目都设置为“error”

与LR(0)和SLR
分析表不同

如果LR(1)分析表中没有语法分析动作冲突,
那么给定的文法就称为LR(1)文法

LR(1)自动机某项集中的项目 $[A \rightarrow \alpha \cdot X \beta, b]$ 若有后继项目 $A \rightarrow \alpha X \cdot \beta$, 其后继项目的展望符是:

- ☒ A 继承的(传播的) b
- ☐ B FOLLOW(X)
- ☐ C FOLLOW(A)
- ☐ D FIRST(βb)

练习

➤ 画出文法的LR(1)自动机，并构建LR(1)分析表

文法G'

$$0) S' \rightarrow S$$

$$1) S \rightarrow BB$$

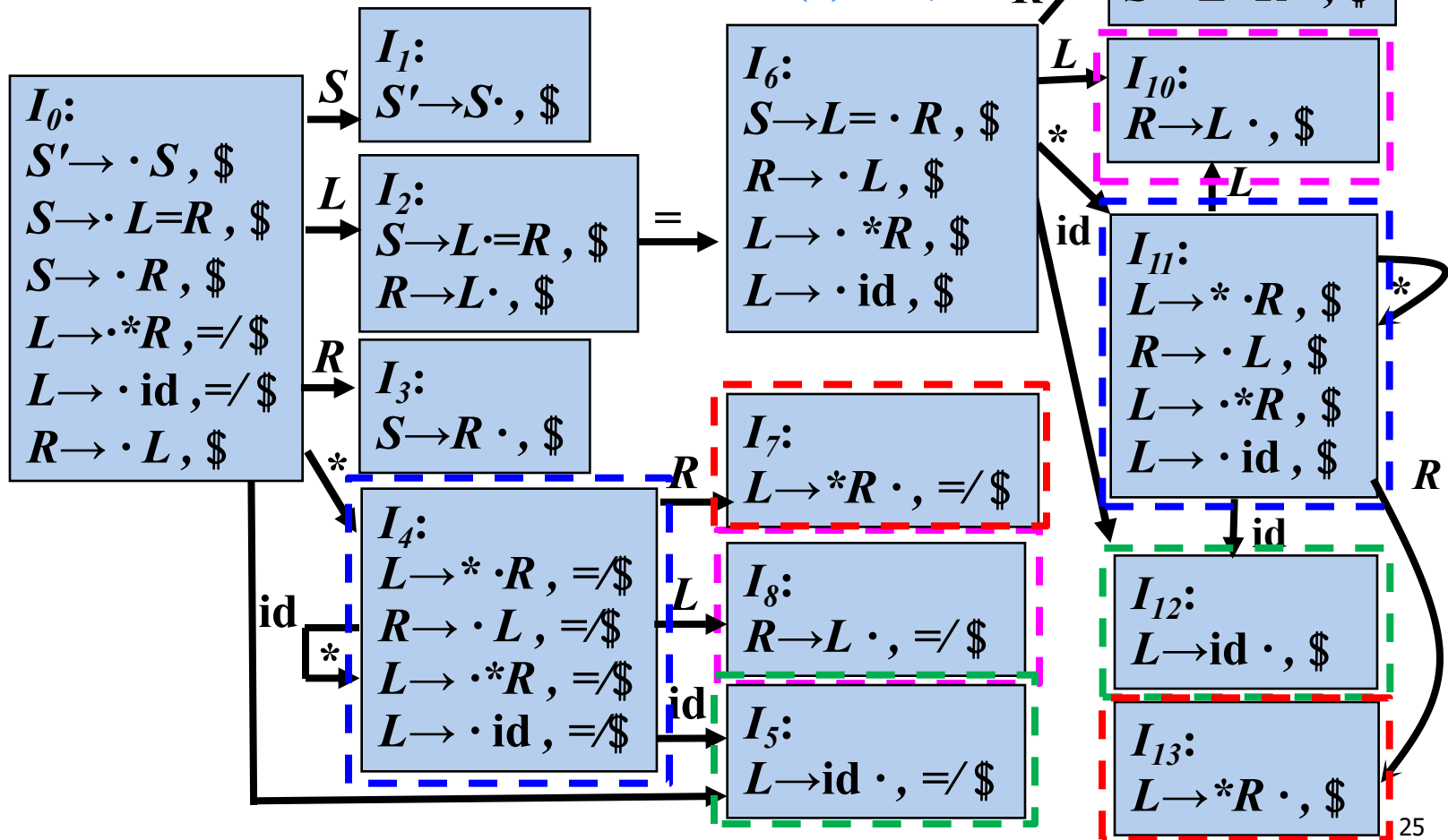
$$2) B \rightarrow aB$$

$$3) B \rightarrow b$$

4.4.4 LALR分析

➤ 例

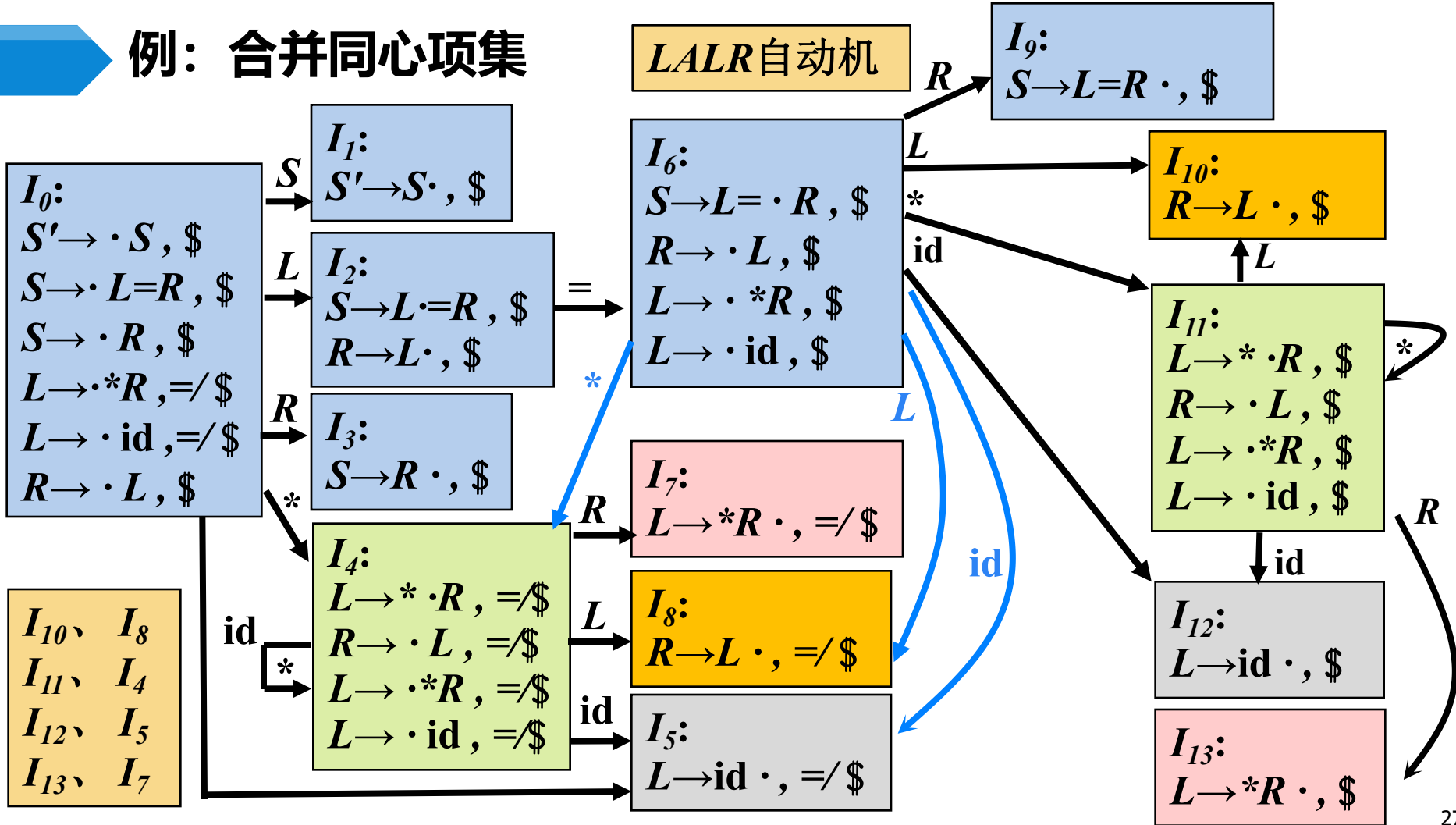
- 0) $S' \rightarrow S$
- 1) $S \rightarrow L=R$
- 2) $S \rightarrow R$
- 3) $L \rightarrow *R$
- 4) $L \rightarrow \text{id}$
- 5) $R \rightarrow L$



*LALR (lookahead-LR)*分析的基本思想

- 寻找具有**相同核心**的 LR (1) 项集，并将这些项集合并为一个项集。所谓项集的核心就是其第一分量的集合
- 然后根据合并后得到的项集族构造语法分析表
- 如果分析表中**没有**语法分析动作**冲突**，给定的文法就称为 **$LALR$ (1) 文法**，就可以根据该分析表进行语法分析

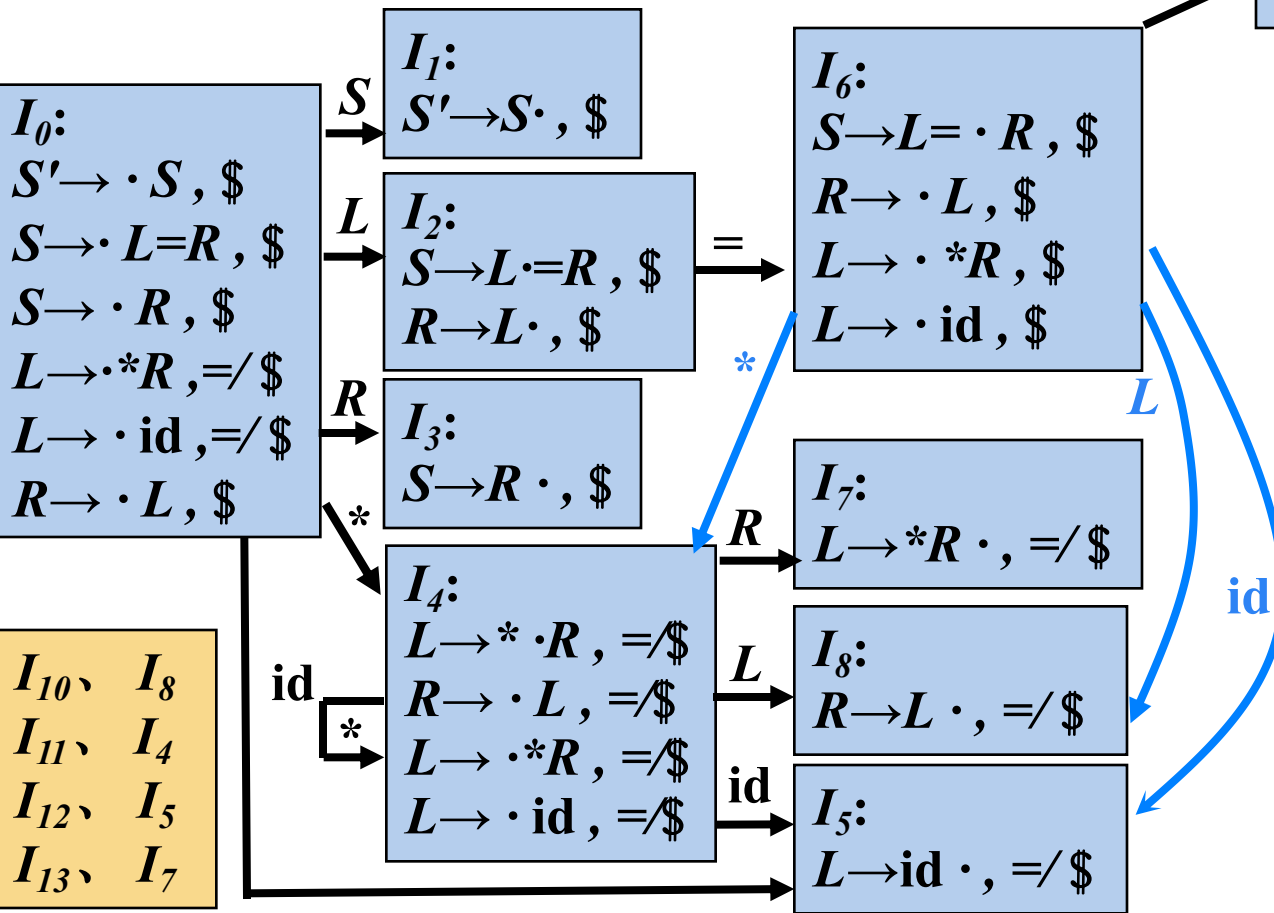
例：合并同心项集



例：合并同心项集

LALR 自动机

I_9 :
 $S \rightarrow L = R \cdot, \$$



LALR 分析表

状态	ACTION				GOTO		
	*	id	=	\$	S	L	R
0	s4	s5			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s4	s5				8	7
5			r4	r4			
6	s4	s5				8	9
7			r3	r3			
8			r5	r5			
9				r1			

合并同心项集时产生归约-归约冲突的例子

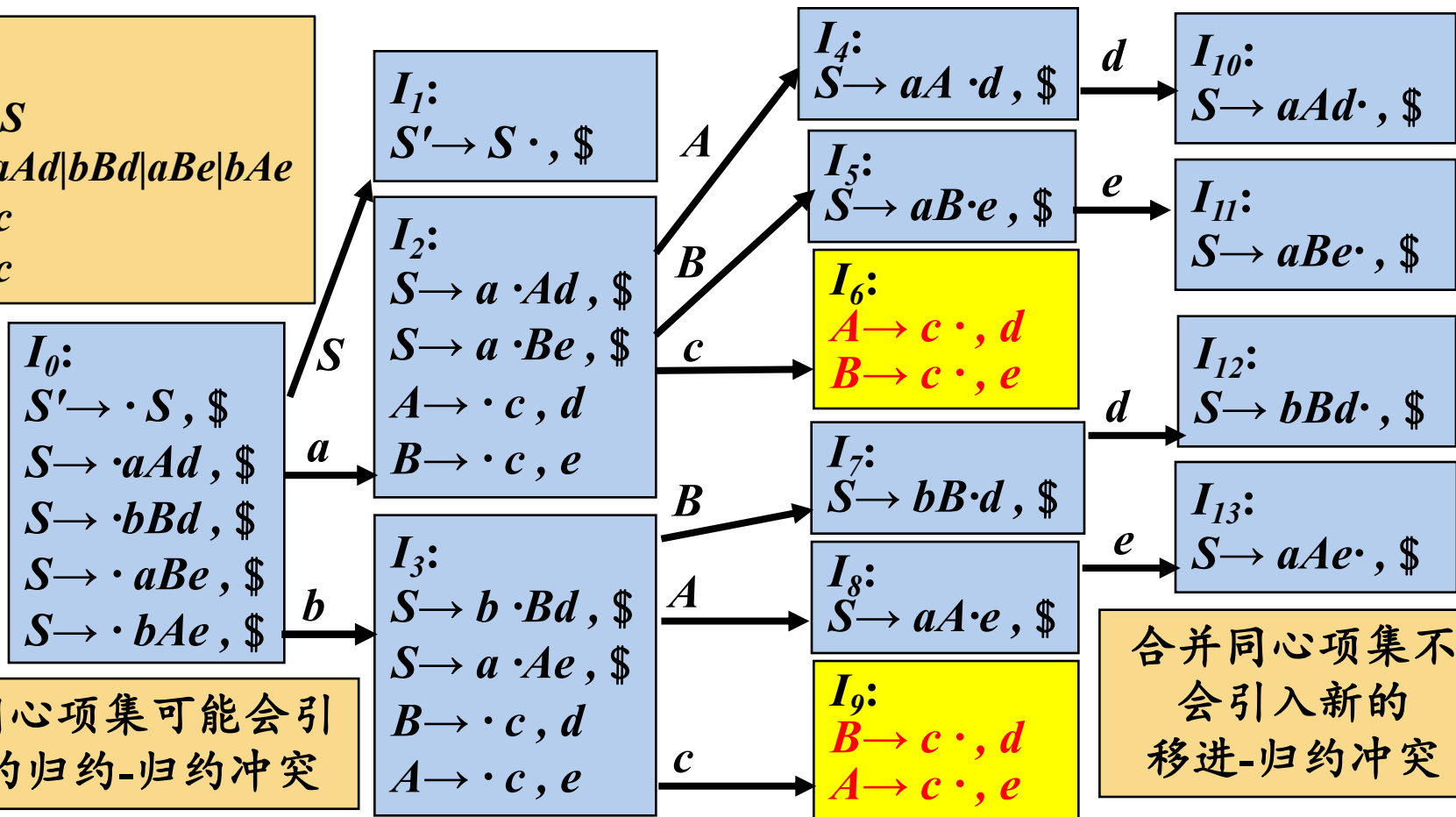
文法

0) $S' \rightarrow S$

1) $S \rightarrow aAd | bBd | aBe | bAe$

2) $A \rightarrow c$

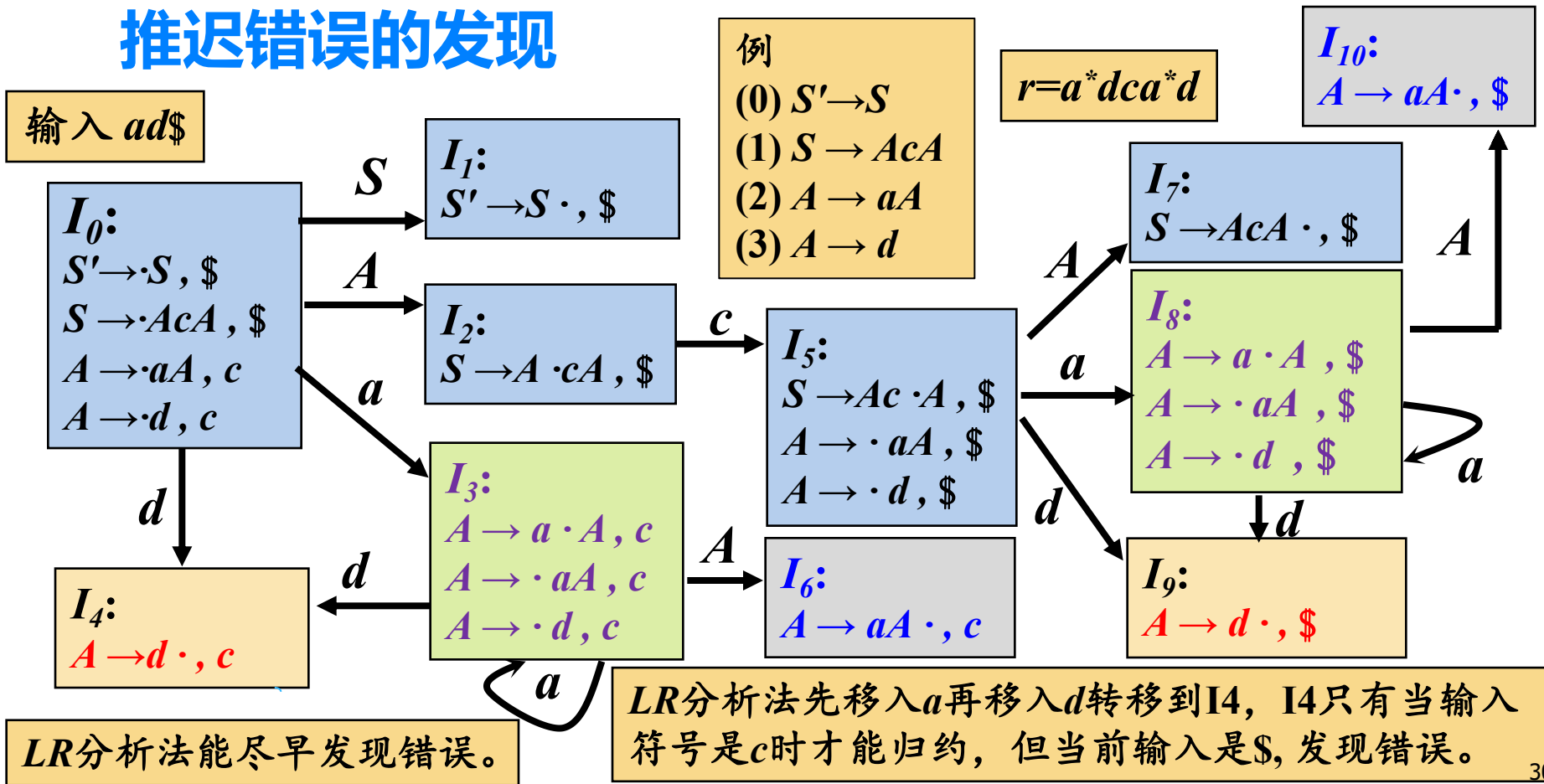
3) $B \rightarrow c$



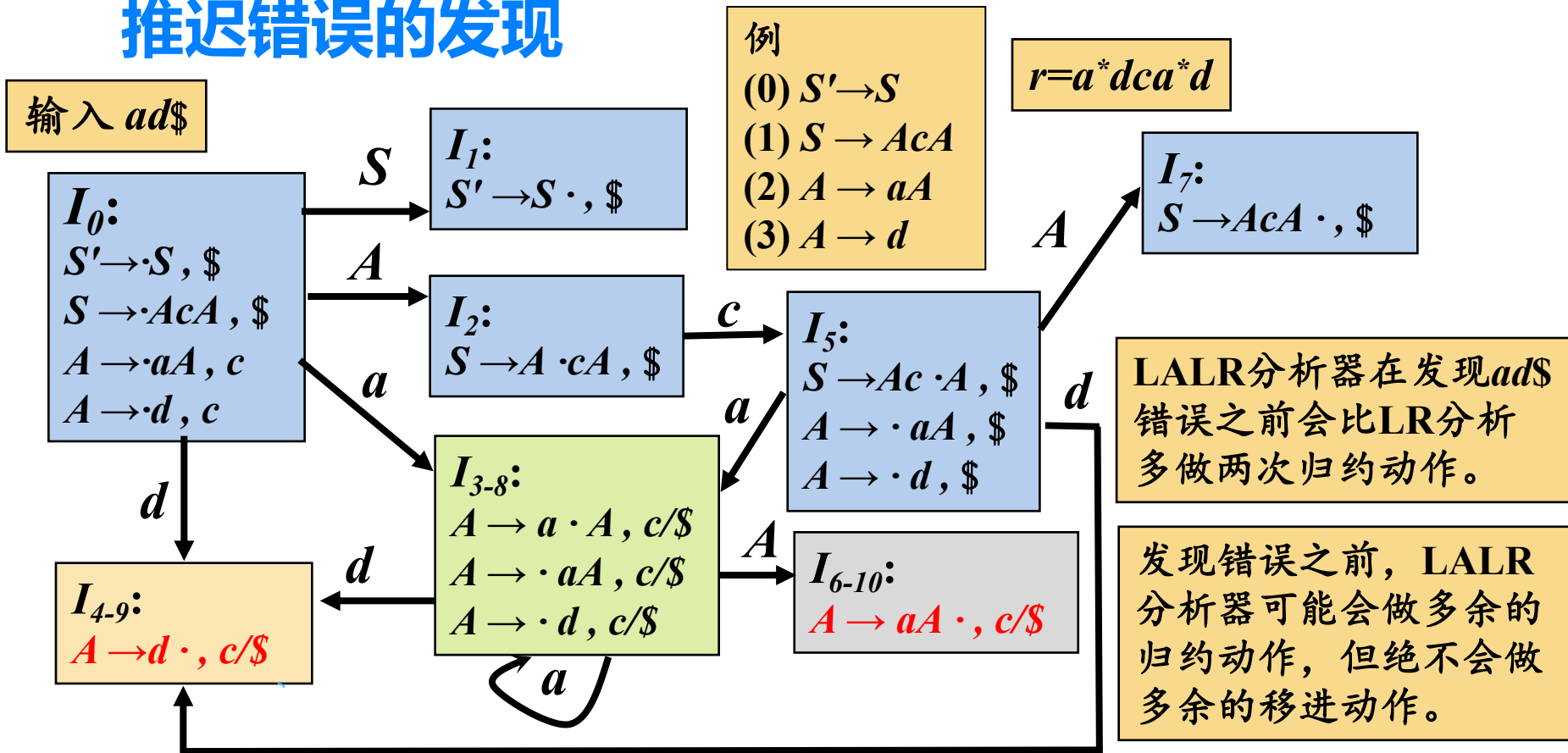
合并同心项集可能会引入新的归约-归约冲突

合并同心项集不会引入新的移进-归约冲突

合并同心项集后，虽然不产生冲突，但可能会推迟错误的发现

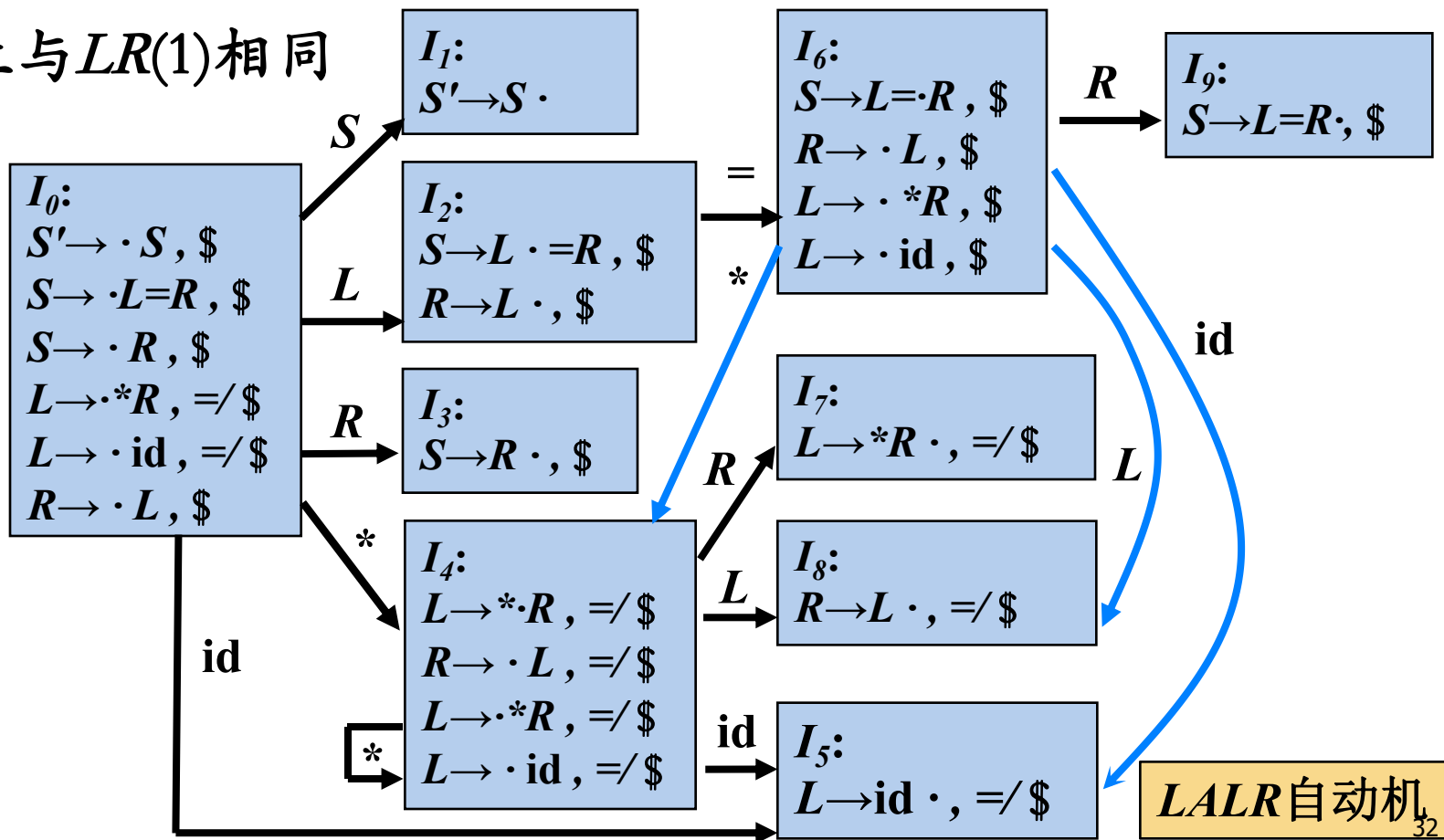


合并同心项集后，虽然不产生冲突，但可能会推迟错误的发现



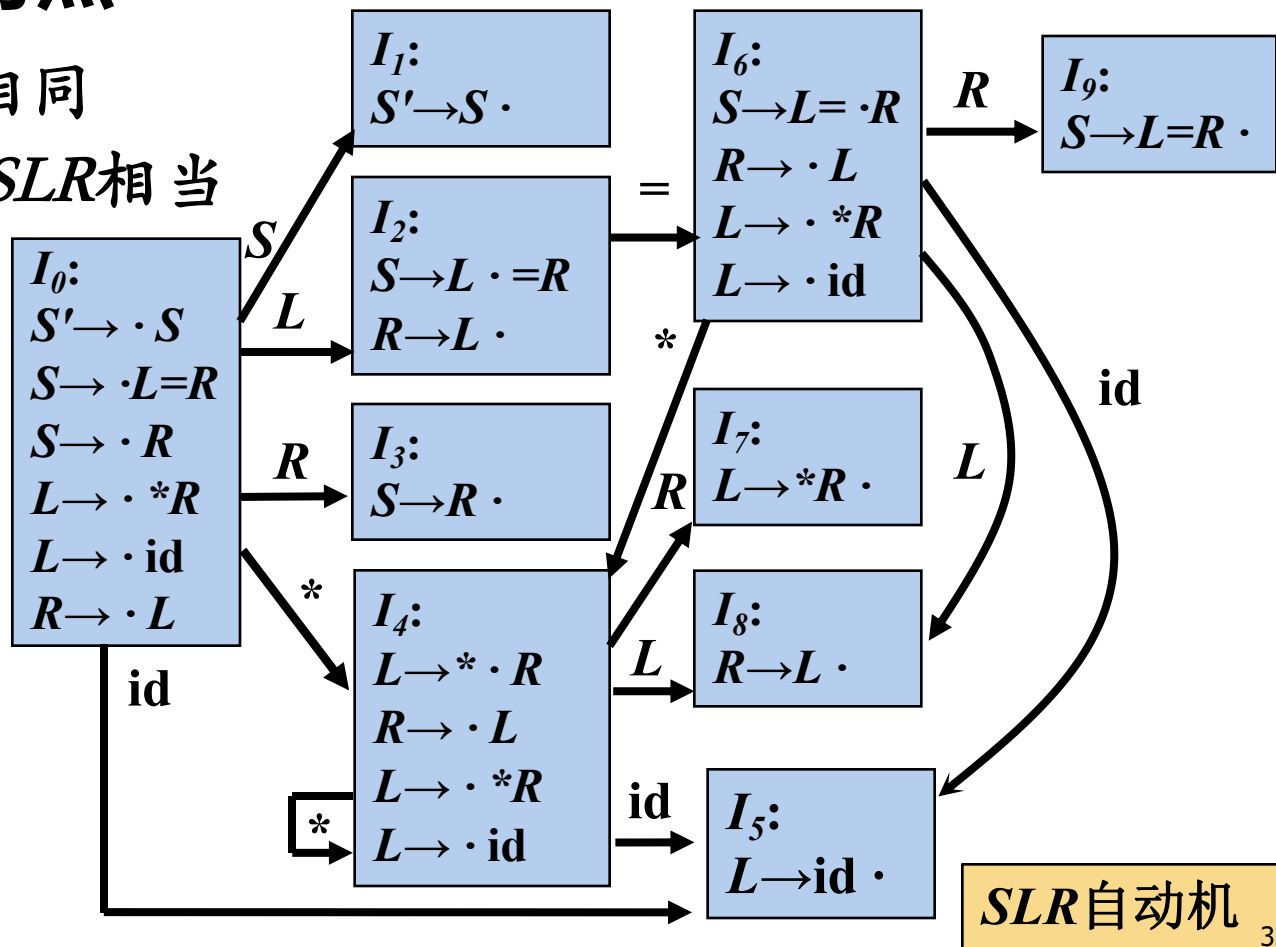
LALR(1)的特点

➤ 形式上与LR(1)相同



LALR(1)的特点

- 形式上与 $LR(1)$ 相同
- 大小上与 $LR(0)/SLR$ 相当



$LALR(1)$ 的特点

- 形式上与 $LR(1)$ 相同
- 大小上与 $LR(0)/SLR$ 相当
- 分析能力介于 SLR 和 $LR(1)$ 二者之间

$$SLR < LALR(1) < LR(1)$$

- 合并后的展望符集合仍为 $FOLLOW$ 集的子集

LALR自动机的状态数与（ ）相同。

- ☒ A LR(0)自动机
- ☒ B SLR自动机
- ☐ C LR自动机
- ☐ D LR项集族中项集数

4.4.5 二义性文法的LR分析

- 每个二义性文法都不是 LR 的
- 某些类型的二义性文法在语言的描述和实现中很有用
 - 更简短、更自然
 - 例

二义性文法

- ① $E \rightarrow E + E$
- ② $E \rightarrow E * E$
- ③ $E \rightarrow (E)$
- ④ $E \rightarrow \text{id}$

非二义性文法

- ① $E \rightarrow E + T$
- ② $E \rightarrow T$
- ③ $T \rightarrow T * F$
- ④ $T \rightarrow F$
- ⑤ $F \rightarrow (E)$
- ⑥ $F \rightarrow \text{id}$

二义性算术表达式文法的SLR分析器

➤ 例

文法

$E \rightarrow E + E$

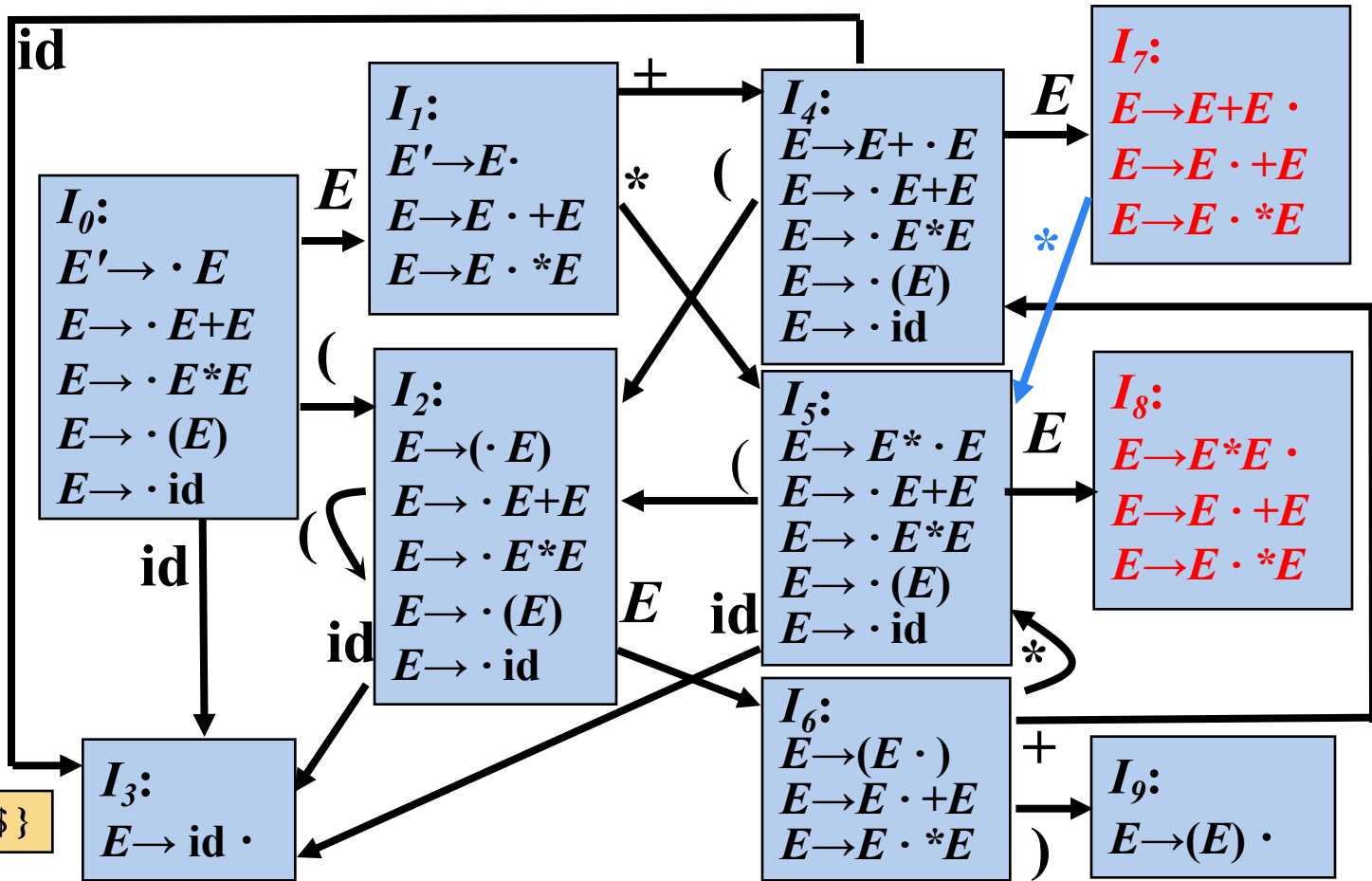
$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow id$

用优先级和结合性解决冲突

$FOLLOW(E) = \{ +, *,), \$ \}$



二义性算术表达式文法的SLR分析表

文法

(1) $E \rightarrow E + E$

(2) $E \rightarrow E * E$

(3) $E \rightarrow (E)$

(4) $E \rightarrow \text{id}$

I_7 :

$E \rightarrow E + E \cdot$

$E \rightarrow E \cdot + E$

$E \rightarrow E \cdot * E$

I_8 :

$E \rightarrow E * E \cdot$

$E \rightarrow E \cdot + E$

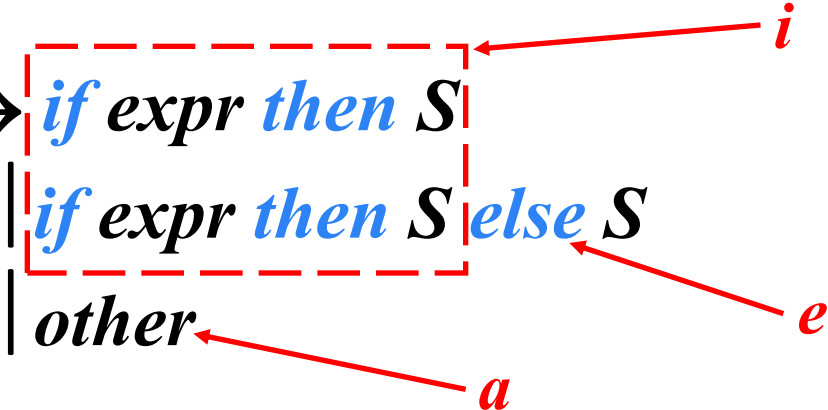
$E \rightarrow E \cdot * E$

$FOLLOW(E) = \{ +, *,), \$ \}$

状态	ACTION						GOTO
	id	+	*	()	\$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

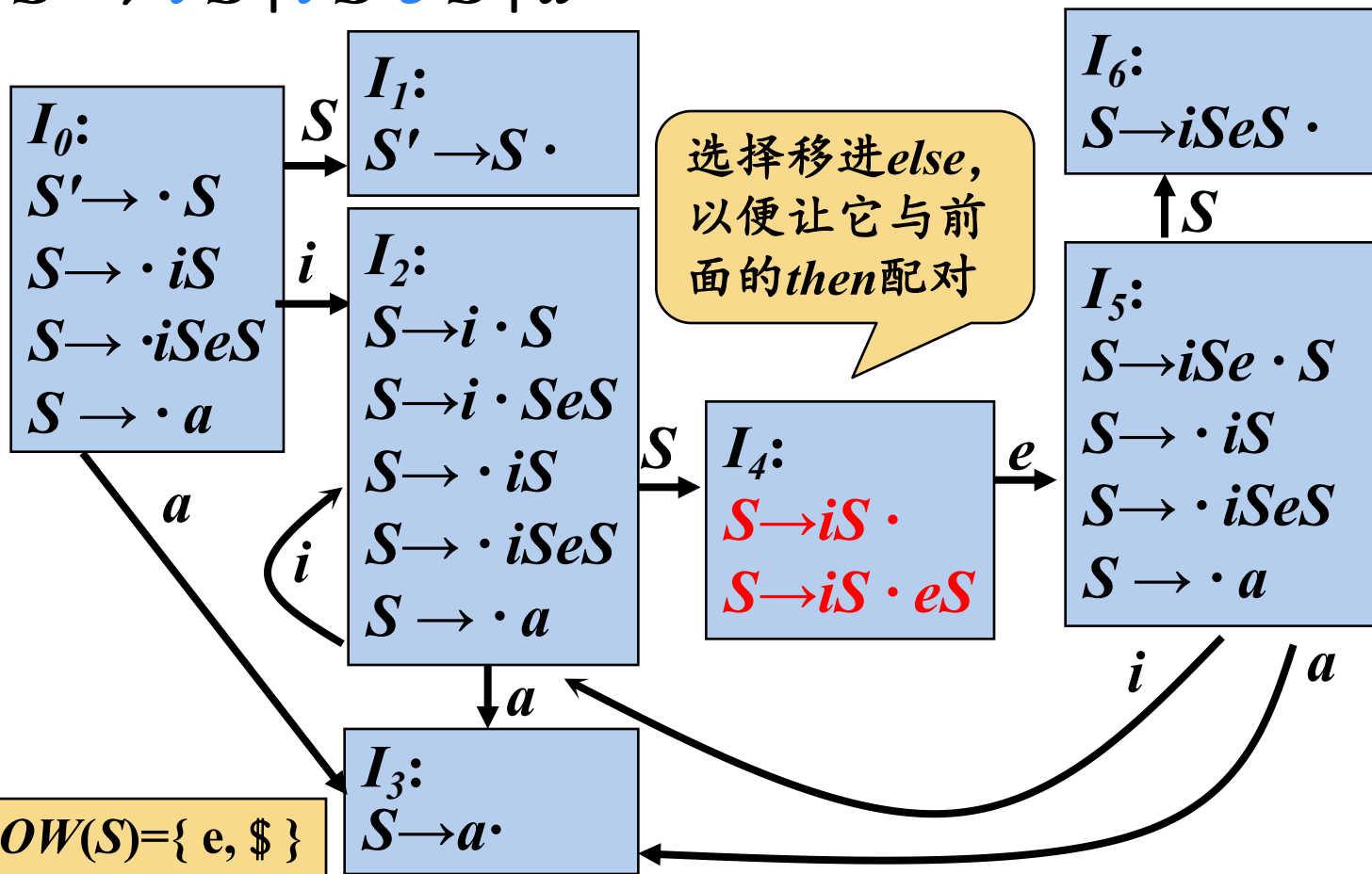
例：二义性 if 语句文法的 LR 分析

➤ $S \rightarrow$ $\begin{cases} if\ expr\ then\ S \\ if\ expr\ then\ S\ else\ S \\ other \end{cases}$



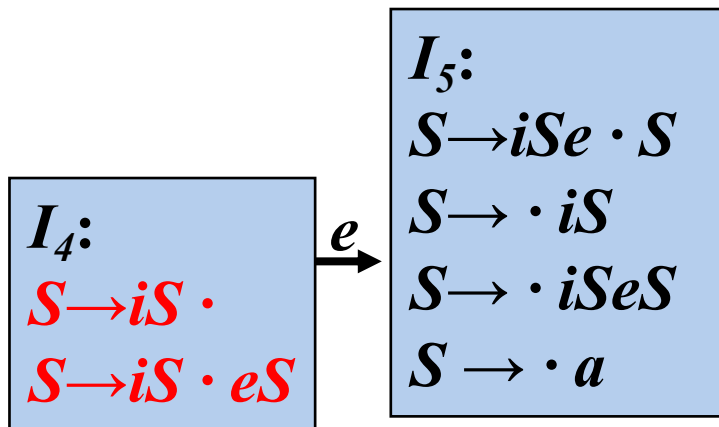
➤ $S \rightarrow i\ S \mid i\ S\ e\ S \mid a$

$$S \rightarrow iS \mid iSeS \mid a$$



二义性 if 语句文法的 SLR 分析表

➤ $S \rightarrow iS \mid iSeS \mid a$



$FOLLOW(S) = \{ e, \$ \}$

状态	ACTION				GOTO
	i	e	a	\$	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5		r1	
5	s2		s3		6
6		r2		r2	

二义性文法的使用

- 应该保守地使用二义性文法，并且必须在严格控制之下使用，因为稍有不慎就会导致语法分析器所识别的语言出现偏差
- 实验中Bison处理二义性文法的方法
 - C--文法是有二义性的
 - 2.2.12 Bison：二义性与冲突处理 P38

4.4.6 LR分析中的错误处理

➤ 语法错误的检测

- 当LR分析器在查询语法分析**动作表**并发现一个**报错条目**时，就检测到了一个语法错误

➤ 错误恢复策略

- 恐慌模式错误恢复
- 短语层次错误恢复
- 错误产生式

恐慌模式错误恢复

$s_0 s_1 \dots s_i s_{i+1} \dots s_m$

$\$X_1 \dots X_i X_{i+1} \dots X_m$

$a_j a_{j+1} \dots a_{j+k} a_{j+k+1} \dots a_n \$$

A

- 从栈顶向下扫描，直到发现某个状态 s_i ，它有一个对应于某个非终结符 A 的 $GOTO$ 目标，可以认为从这个 A 推导出的串中包含错误
- 然后丢弃0个或多个输入符号，直到发现一个可能合法地跟在 A 之后的符号 a 为止。
- 之后将 $s_{i+1} = GOTO(s_i, A)$ 压入栈中，继续进行正常的语法分析

短语层次错误恢复

- 检查 LR 分析表中的每一个报错条目，并根据语言的使用方法来决定程序员所犯的何种错误最有可能引起这个语法错误
- 然后构造出适当的恢复过程

例：算数表达式文法的LR分析器

文法

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow id$$

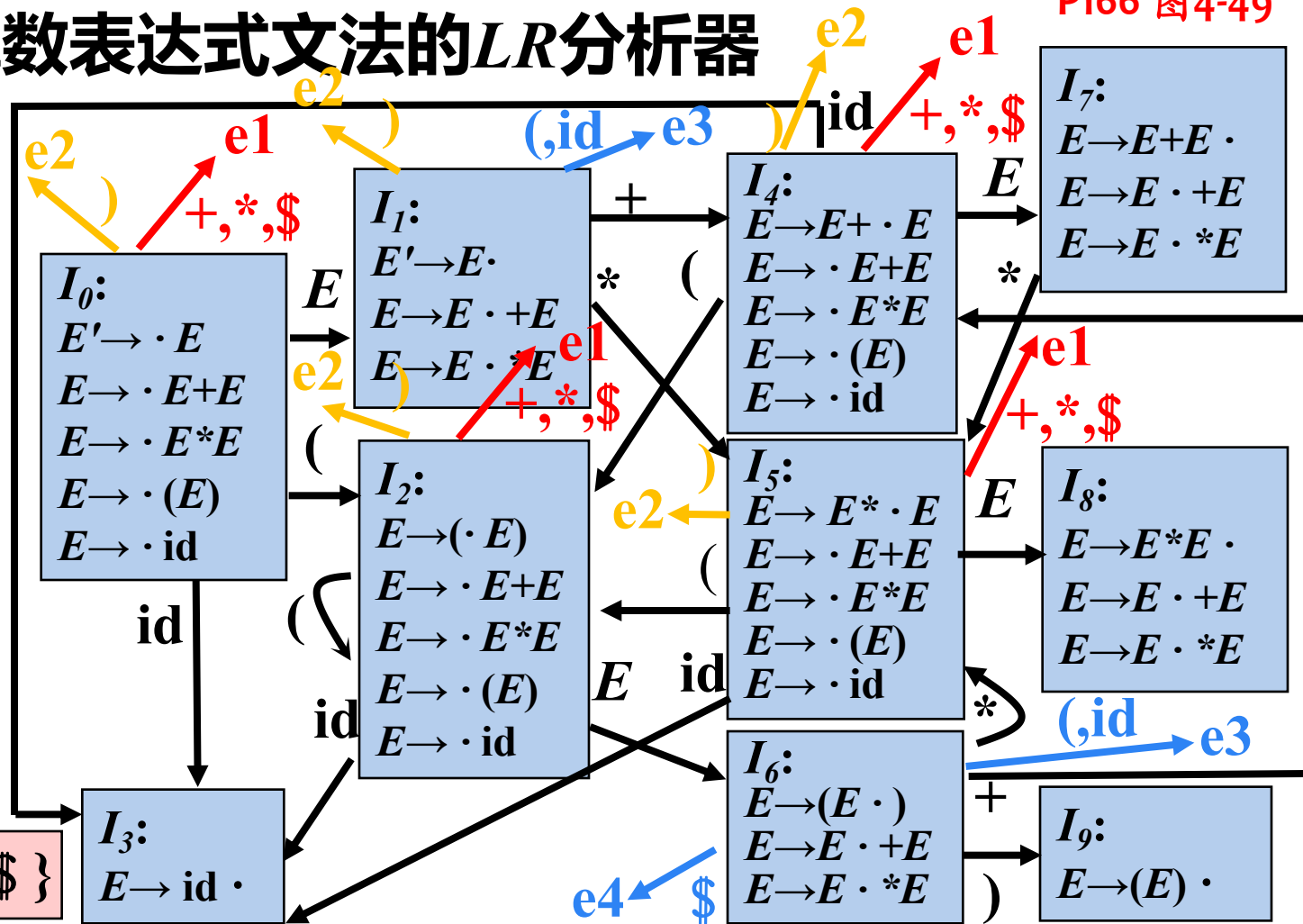
e1: 缺少运算分量
状态3入栈

e2: 不匹配的右括号
忽略右括号

e3: 缺少运算符
状态4入栈

e4: 缺少右括号
状态9入栈

{ +, *, (,), id, \$ }



带有错误处理子程序的算术表达式文法LR分析表

状态	ACTION						GOTO
	id	+	*	()	\$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

➤ 错误处理例程

- $e1$: 将状态3压入栈中，发出诊断信息“缺少运算分量”
- $e2$: 从输入中删除“ $)$ ”，发出诊断信息“不匹配的右括号”
- $e3$: 将状态4压入栈中，发出诊断信息“缺少运算符”
- $e4$: 将状态9压入栈中，发出诊断信息“缺少右括号”

➤ 3、7、8、9行，id 和 (列，被替换为相应状态中的归约动作

带有错误处理子程序的算术表达式文法LR分析表

状态	ACTION						GOTO
	id	+	*	()	\$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

【例】处理错误的输入 id+) \$

栈	输入	动作
0	id+) \$	
03	+) \$	
01	+) \$	
014) \$	e2不匹配的右括号
014	\$	e1 缺少运算分量
0143	\$	状态3入栈
0147	\$	
01	\$	acc

➤ 3、7、8、9行，id 和 (列，被替换为相应状态中的归约动作

错误产生式

➤ 错误产生式 (Yacc, Bison, P175)

➤ 通过预测可能遇到的常见错误，在文法中加入特殊的错误产生式。

➤ `error RP //`表示右括号不匹配

➤ `error SEMI//`表示语句错误。

➤ 为包含错误产生式的文法构造分析表和分析器

➤ 分析中如使用了错误产生式归约，即发现了此错误产生式代表的错误类型

➤ 据此生成准确的错误诊断信息

LR分析的基本步骤

- 1、编写增广文法
- 2、求识别所有规范句型活前缀的DFA
- 3、构造LR分析表

课后练习

➤ 文法 $S \rightarrow SS + \mid SS * \mid a$ 构造

① 规范LR(1)项集族 (DFA)

② LALR项集族 (DFA)



提纲

4.1 自顶向下的分析

4.2 预测分析法

4.3 自底向上的分析

4.4 LR分析法

4.5 语法分析器自动生成工具

4.6 语法分析程序的自动生成工具Yacc

%{ /*变量定义：头文件和全局变量*/ **%}**

%union{ /*定义类型*/

int type_int;

float type_float;

double type_double;

}

%token <type_int> INT /*终结符号声明*/

.....

%type <type_double> EXP FACTOR TERM /* 语法变量属性值类型*/

%%规则部分 给出文法规则的描述

Exp : Factor

| Exp PLUS Factor { **\$\$ = \$1 + \$3;** }

| Exp MINUS Factor { **\$\$ = \$1 - \$3;** }

%%程序部分 扫描器和语义动作程序

1.定义部分

语义动作

2.规则部分

3.用户函数部分

4.6 语法分析程序的自动生成工具Yacc

➤ Yacc源程序的规则部分

- 用于放置翻译规则，每条规则由文法产生式和语义动作组成

例： $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$

A : α_1 { 语义动作 1 }

| α_2 { 语义动作 2 }

...

| α_n { 语义动作 n }

归约时调用

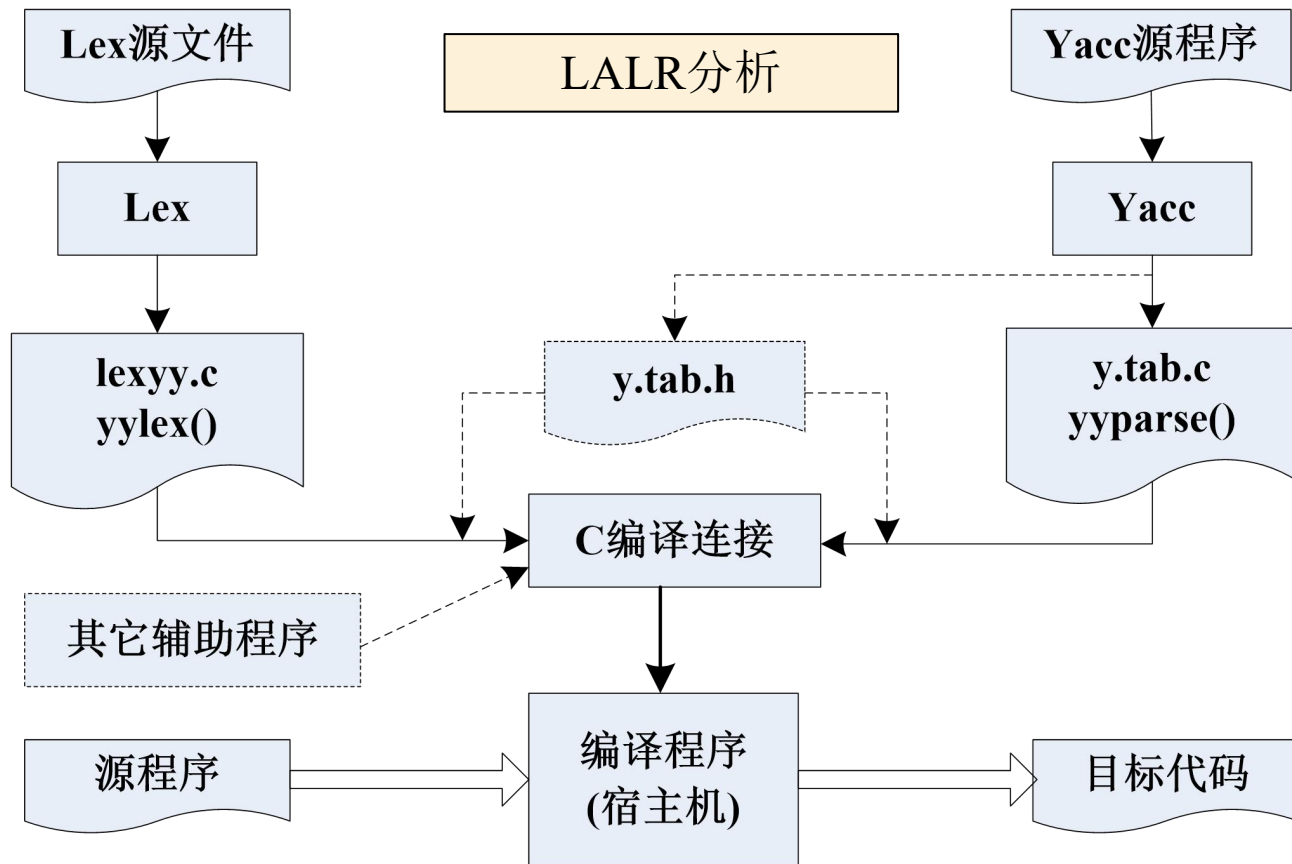
用户自定义函数，创建
树的若干结点

4.6 语法分析程序的自动生成工具Yacc

➤ Yacc源程序的用户函数部分

- 可选的，由一些例行的C语言程序组成，主要包括：
- 主程序main()
- 词法分析程序yylex() /*如果是自定义的*/
- 出错处理子程序yyerror()
- 语义动作所调用的用户自定义函数及其他辅助函数等。

用Yacc和Lex合建编译程序



语法分析小结

➤ 自顶向下分析

➤ 核心问题：产生式的选择

➤ 目标：无回溯的确定分析，每一步都选择唯一确定的产生式进行扩展

➤ 解决办法：

➤ 预测分析法：使用最左推导，并参考当前输入符号，唯一确定推导产生式。

➤ LL(1)文法的判定 (FIRST集, FOLLOW集, SELECT集)

语法分析小结

➤ 自顶向下分析

➤ 方法：

➤ 前提：无左递归、无左公因子

➤ 递归的预测分析法

➤ 非递归的LL(1)分析法

} 核心：预测分析表构造

➤ 表驱动的预测分析器构成与分析过程

➤ 对分析树的先根深度优化遍历。（出栈顺序）

语法分析小结

➤ 自底向上分析

➤ 核心问题：如何寻找正确的句柄进行归约？

➤ 目标：每一步都正确地选择句柄进行归约

➤ 解决办法：

➤ 优先法：预定义文法符号之间的归约优先级

➤ 状态法：使用状态追踪句柄的识别进度

➤ LR(0) 项、LR(1) 项

语法分析小结

➤ 自底向上分析

➤ 方法:

➤ 算符优先法

➤ LR分析法: LR(0) SLR(1) LR(1) LALR(1)

➤ LR分析器模型与分析过程

➤ 构造LR自动机, 构造LR分析表 (动作表和转移表)

➤ LR(0)项集、LR(1)项集、闭包函数和GOTO函数

➤ 规范句型、句柄、活前缀、有效项

➤ 对分析树的后根深度优化遍历。(进栈顺序)

实验一 实验指导

➤ 语法树的创建

➤ 语义动作中调用自定义函数创建语法树

➤ 确定多叉树的存储结构

➤ 为不同文法符号定义不同的属性值类型

➤ 终结符号类型就是其词法值的类型

➤ 非终结符号属性值类型是指向子结点的指针

➤ 独立C文件管理自定义代码

➤ 将产生式右部符号作为节点插入多叉树的函数

➤ 遍历并输出语法分析树的函数

```
typedef struct node {  
    int elem;        // 语法单元编号  
    struct node *child[16]; // 子节点  
    int cnt;         // 子节点数量  
    union lexval lex_val; // 词法值  
} node;
```

```
union lexval { // 词法值类型  
    int int_lex;  
    float float_lex;  
    char id_lex[17]; // char *id_lex;  
}
```

实验一 实验指导

```
F : num { $$ = MakeLeafNode(NUM, lex_val) }  
    | id { 语义动作2 }
```

```
T : T * F { star = MakeLeafNode(STAR, lex_val) ;  
           child[0] = $1; child[1] = star;  
           child[3] = $3  
           $$ = MakeNode(child, 3) }
```

```
| F { $$ = $1; }
```

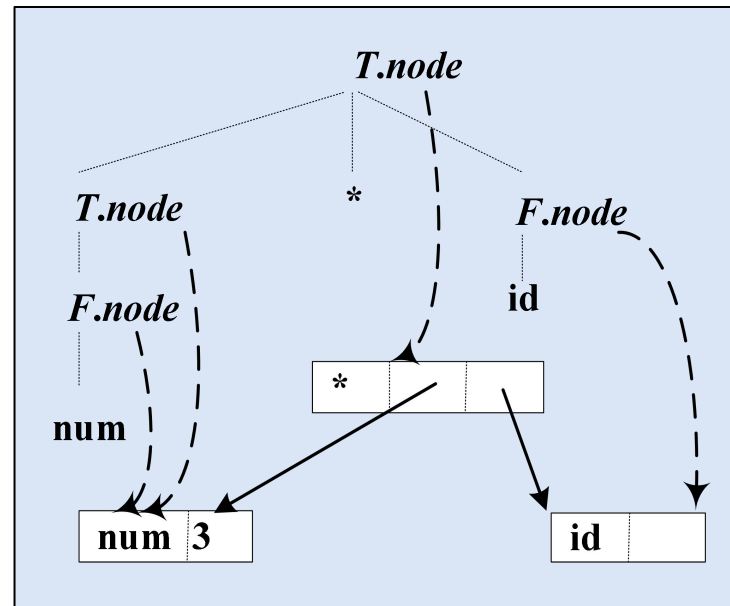
自定义函数:

/*生成叶节点函数。参数: 编号和词法值*/

```
node* MakeLeafNode ( int token,  
                     union lexval lex_val)
```

/*生成中间节点函数, 参数: 归约产生式, 孩子数*/

```
node * MakeNode (struct node *child, int c )
```





结束

