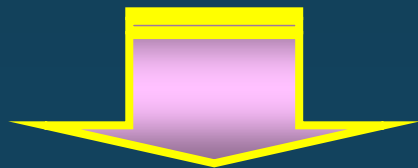


分析功能: $\text{LR}(1) \supset \text{SLR}(1) \supset \text{LR}(0)$

分析器开销: $\text{LR}(1) \uparrow \text{SLR}(1) \text{LR}(0) \downarrow$



存储开销 $\Rightarrow \text{LR}(0), \text{SLR}(1)$

分析功能 $\Rightarrow \text{LR}(1)$





LALR(1)分析表状态数 = LR(0)的C

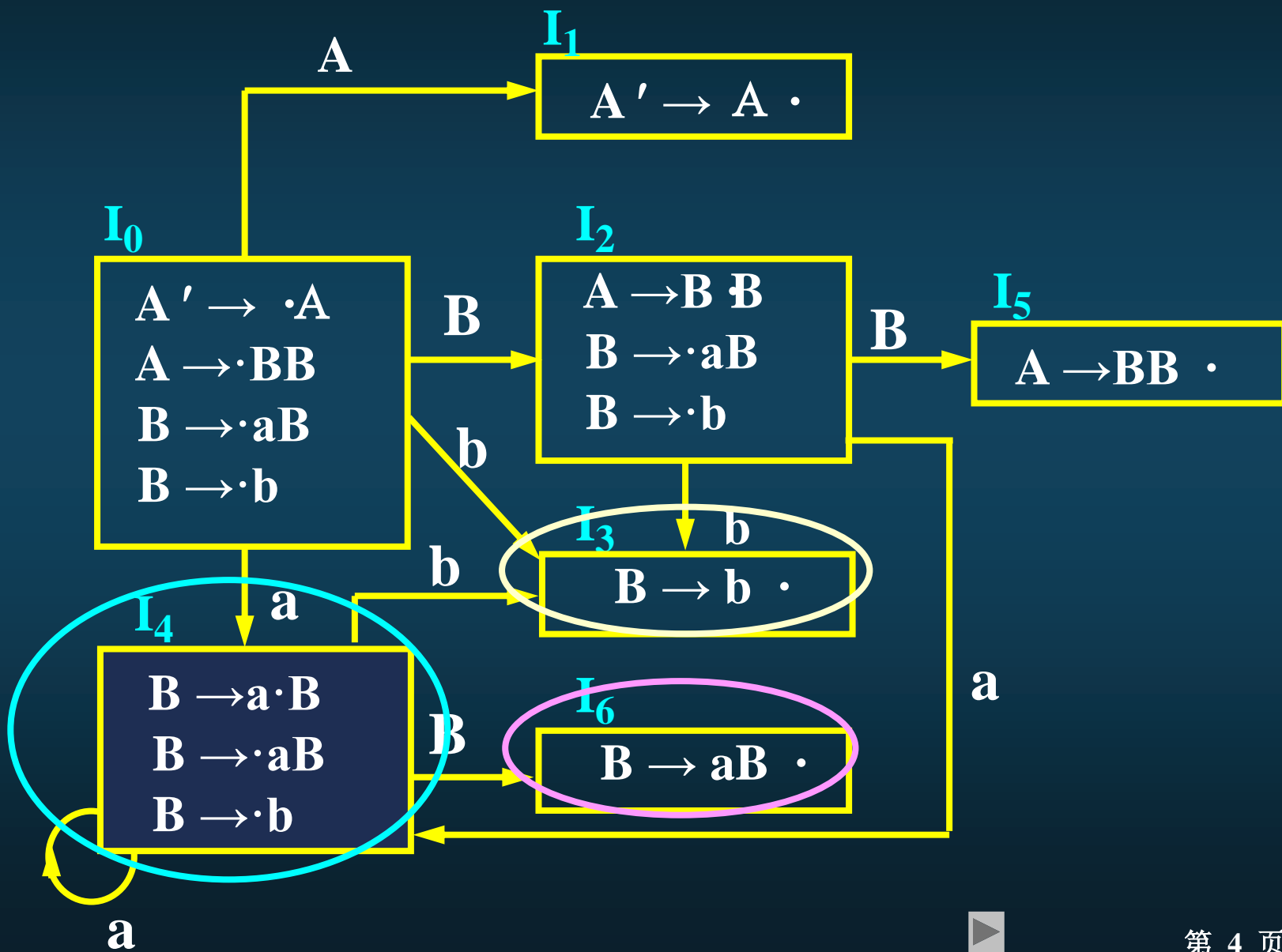
LALR(1)分析功能 \subseteq LR(1)

例5.17 设有文法G(A)

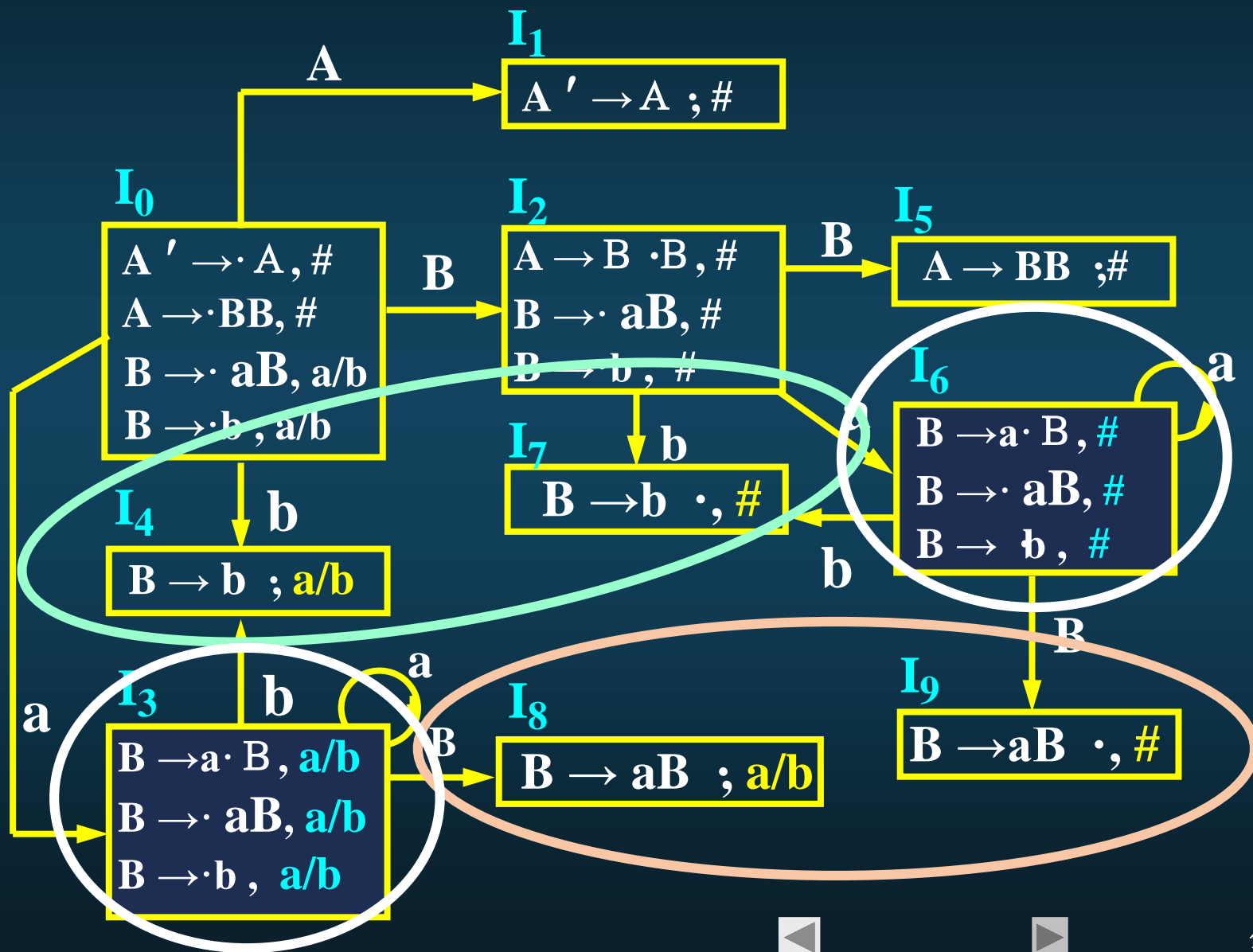
- (1) $A' \rightarrow A$
- (2) $A \rightarrow BB$
- (3) $B \rightarrow aB$
- (4) $B \rightarrow b$

$$L(G(A)) = \{ a^*ba^*b \}$$

文法G(A)的LR(0)项目集规范族



文法G(A)的LR(1)项目集规范族



对比文法G(A)的LR(1)的C和LR(0)的C

LR(0)的C

 I_4

分裂为

 I_6

分裂为

 I_3

分裂为

LR(1)的C

 I_3 、 I_6 I_8 、 I_9 I_4 、 I_7

每个项目集
仅搜索符不
同，LR(0)项
目相同

■ 定义5.16 （同心项目集）

对文法 G 的 LR(1) 项目集规范族，若存在两个（或两个以上）项目集 I_0 、 I_1 ，其中 I_0 、 I_1 项目集中的**LR(0)项目相同，仅搜索符不同**，则称 I_0 、 I_1 为 G 的LR(1)的同心项目集。或称 I_0 、 I_1 具有相同的心。

合并LR(1)的C中的同心项目集 \Rightarrow LALR (1)的C



缩小LR (1)的C分析的状态数

$I_0: \{ A' \rightarrow \cdot a, \#; A \cdot BB, \#; B \rightarrow \cdot aB, a/b; \\ B \rightarrow \cdot b, a / b \}$

$I_1: \{ A' \rightarrow A \cdot, \# \}$

$I_2: \{ A \rightarrow B \cdot B, \#; B \rightarrow \cdot aB, \#; B \rightarrow \cdot b, \# \}$

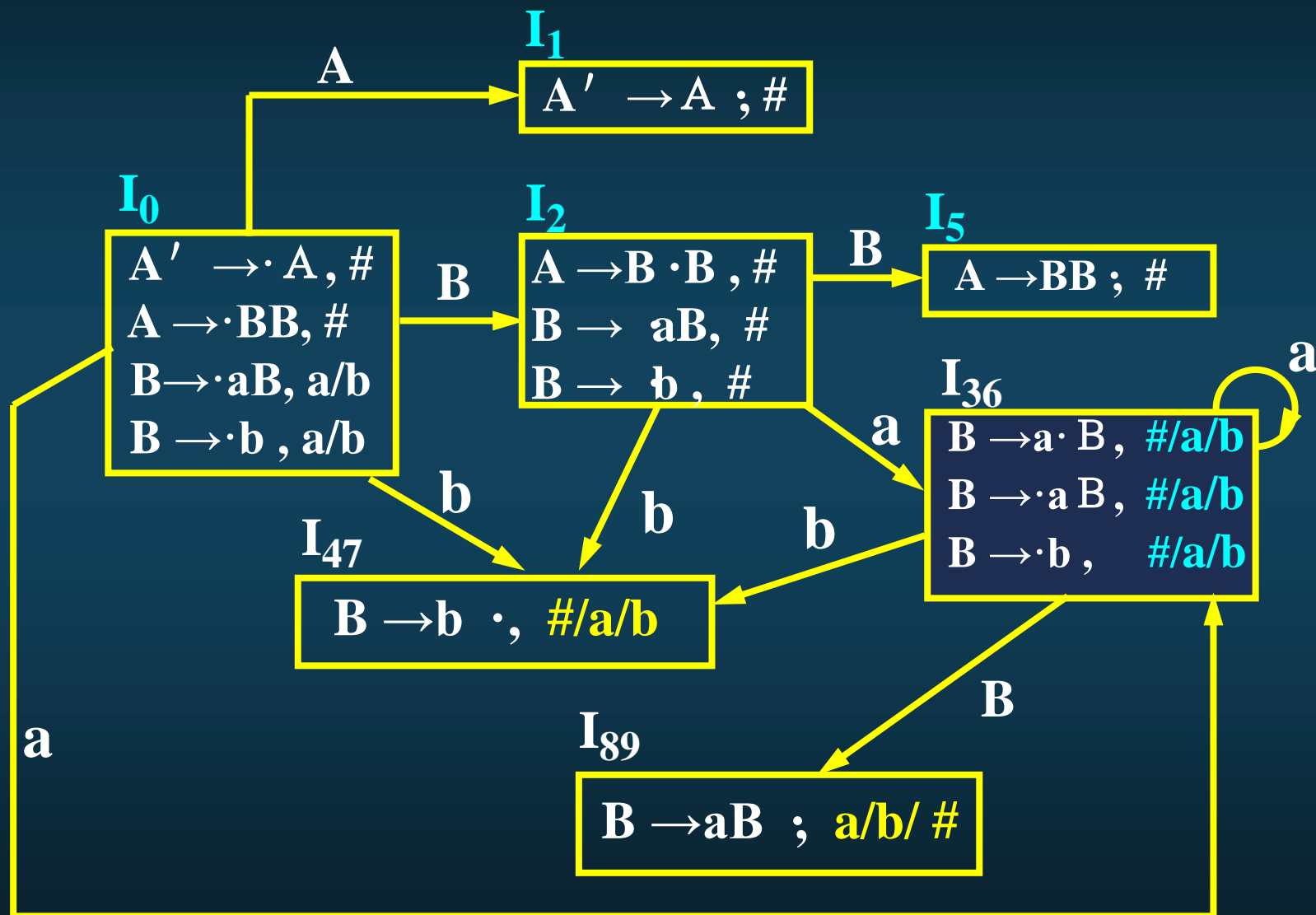
$I_{36}: \{ B \rightarrow a \cdot B, a/b/\#; B \rightarrow \cdot aB, a/b/\#; \\ B \rightarrow \cdot b, a/b/\# \}$

$I_{47}: \{ B \rightarrow b \cdot, a/b/\# \}$

$I_5: \{ A \rightarrow BB \cdot, \# \}$

$I_{89}: \{ B \rightarrow aB, a/b/\# \}$

文法G(A)的LALR(1)项目集规范族



文法G(A)的GO函数

	A'	A	B	a	b	#
0		1	2	<u>36</u>	<u>47</u>	
2			5	<u>36</u>	<u>47</u>	
<u>36</u>			<u>89</u>	<u>36</u>	<u>47</u>	

* 注意：1, 5, 47, 89为归约态。

💧 注意：

1. LR(1)的C无冲突 \Rightarrow LALR(1)的C

LALR(1)的C { 无冲突：构造LALR(1)分析表
有冲突：构造LR(1)的分析表

2. LALR(1)的C成立，分析能力同LR(1)。
确定\$中错误前会比LR(1)多产生若干步
归约。

■ 算法5.11 (LALR(1)分析表构造)

输入: 拓广的文法 G' 及文法 G' 的LALR(1)项目集规范族 C 和GO函数

输出: 文法 G' 的LALR(1)分析表

方法: 设 $C' = \{I_0, I_1, \dots, I_n\}$ 为文法 G 的LALR(1)项目集族

① 若 $[A \rightarrow \alpha \cdot a\beta, b] \in I_k$ 且 $GO(I_k, a) = I_j$, $a \in V_T$, 则置 $action(K, a) = S_j$;

② 若 $[A \rightarrow \alpha \cdot, a] \in I_k$, 则置 $action(K, a) = r_j$, 其中 j 表示 $A \rightarrow \alpha$ 为文法 G 的第 j 个产生式;

③ 若 $[S' \rightarrow S \cdot, \#] \in I_k$, 则置 $action(K, \#) = acc$;

④ 若 $GO(I_k, A) = I_j$, 则置 $GOTO[K, A] = j$ 。

⑤ 分析表中不能用①至④规则填入信息的元素, 则置“出错标志”。

文法G(A)LALR(1)分析表

state	ACTION表			GOTO表	
	a	b	#	A	B
0	S ₃₆	S ₄₇		1	2
1			acc		
2	S ₃₆	S ₄₇			5
36	S ₃₆	S ₄₇			89
47	r ₄	r ₄	r ₄		
5			r ₂		
89	r ₃	r ₃	r ₃		

■ 定义5.16

按照LALR(1)的项目集规范族构造的文法G的LALR(1)分析表，如果每个入口不含多重定义，则称它为G的LALR(1)分析表。具有LALR(1)分析表的文法G称为LALR(1)文法。使用LALR(1)分析表的语法分析器称作LALR(1)分析器。

🔥 综述:

1. 构造文法G的LR(1)的项目集规范族C

$$C = \{ C_0, C_1, C_2, \dots, C_n \}$$

2. 合并C中的所有同心的LR(1)项目集, 用M表示合并后的新项目集

$$M = \{ M_0, M_1, M_2, \dots, M_m \} \quad \text{其中: } m \leq n$$

M称为LALR(1)的项目集规范族。

3. M若没有冲突, 据M构造文法G的LALR(1)的分析表。

LALR(1)的项目集规范族若存在冲突，只能是归约—归约冲突。（尽管原来的LR(1)的C不冲突）

证明：

设原LR(1)的项目集规范族C中有如下项目集

$$\begin{array}{ll} I_k: [A \rightarrow \alpha \cdot, W_1] & I_j: [A \rightarrow \alpha \cdot, W_2] \\ [B \rightarrow \beta \cdot a\gamma, b] & [B \rightarrow \beta \cdot a\gamma, c] \end{array}$$

并设 I_k 与 I_j 均无冲突，故有

$$W_1 \cap \{a\} = \emptyset$$

$$W_2 \cap \{a\} = \emptyset$$

从而 $(W_1 \cup W_2) \cap \{a\} = \emptyset$

现将 I_k 与 I_j 合并, 有

$I_{k/j}$:

$[A \rightarrow \alpha \cdot, W_1 \cup W_2]$

$[B \rightarrow \beta \cdot a \gamma, \{b\} \cup \{c\}]$

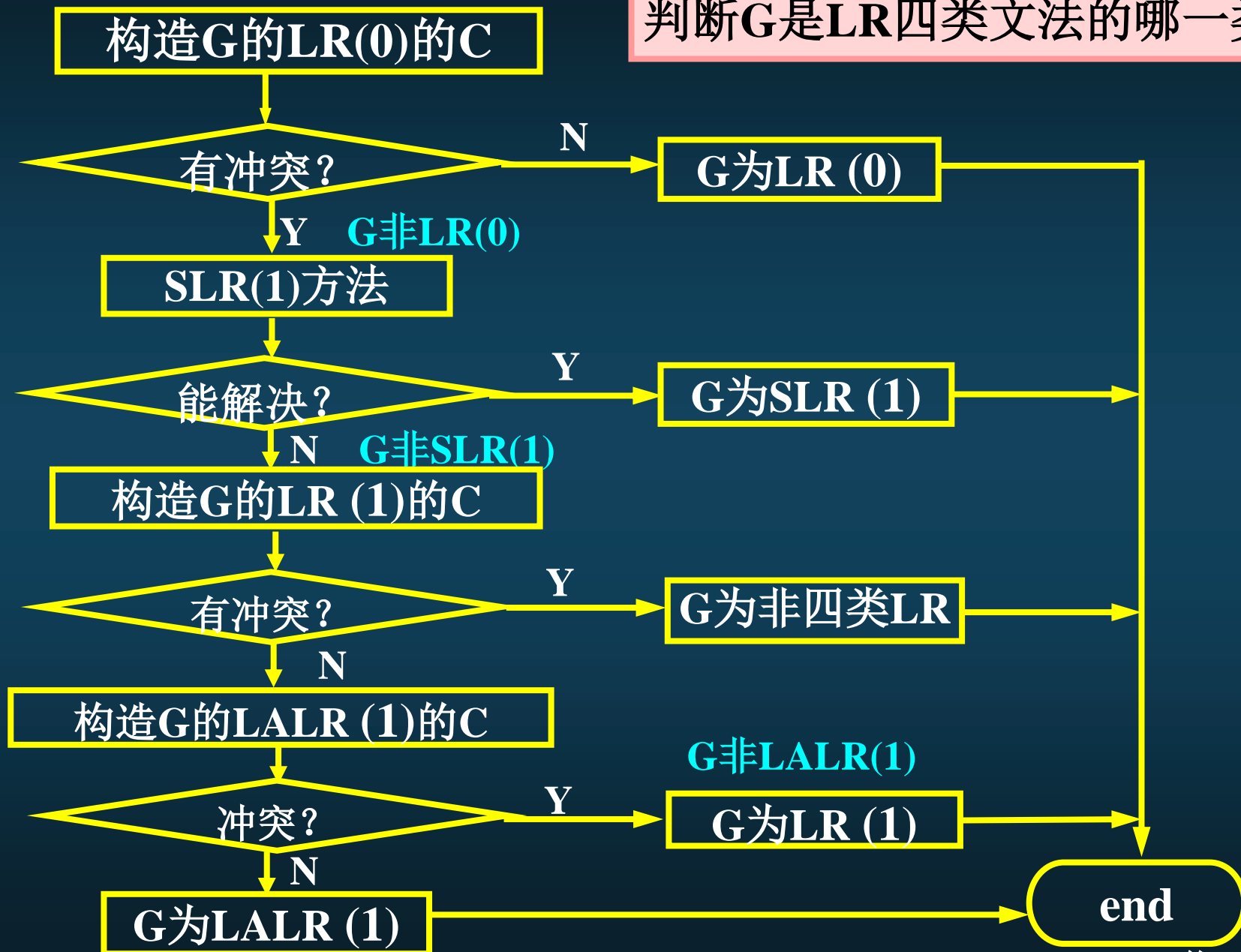
若此时 $I_{k/j}$ 有“移进—归约”冲突, 则必有

$$(W_1 \cup W_2) \cap \{a\} \neq \emptyset$$

与 I_k 和 I_j 无冲突的假设矛盾。

\therefore 合并同心集后不会引入新的“移进—归约”冲突。证毕。

判断G是LR四类文法的哪一类



例5.18 设有下列文法

$S \rightarrow Aa \mid bAc \mid Bc \mid bBa$

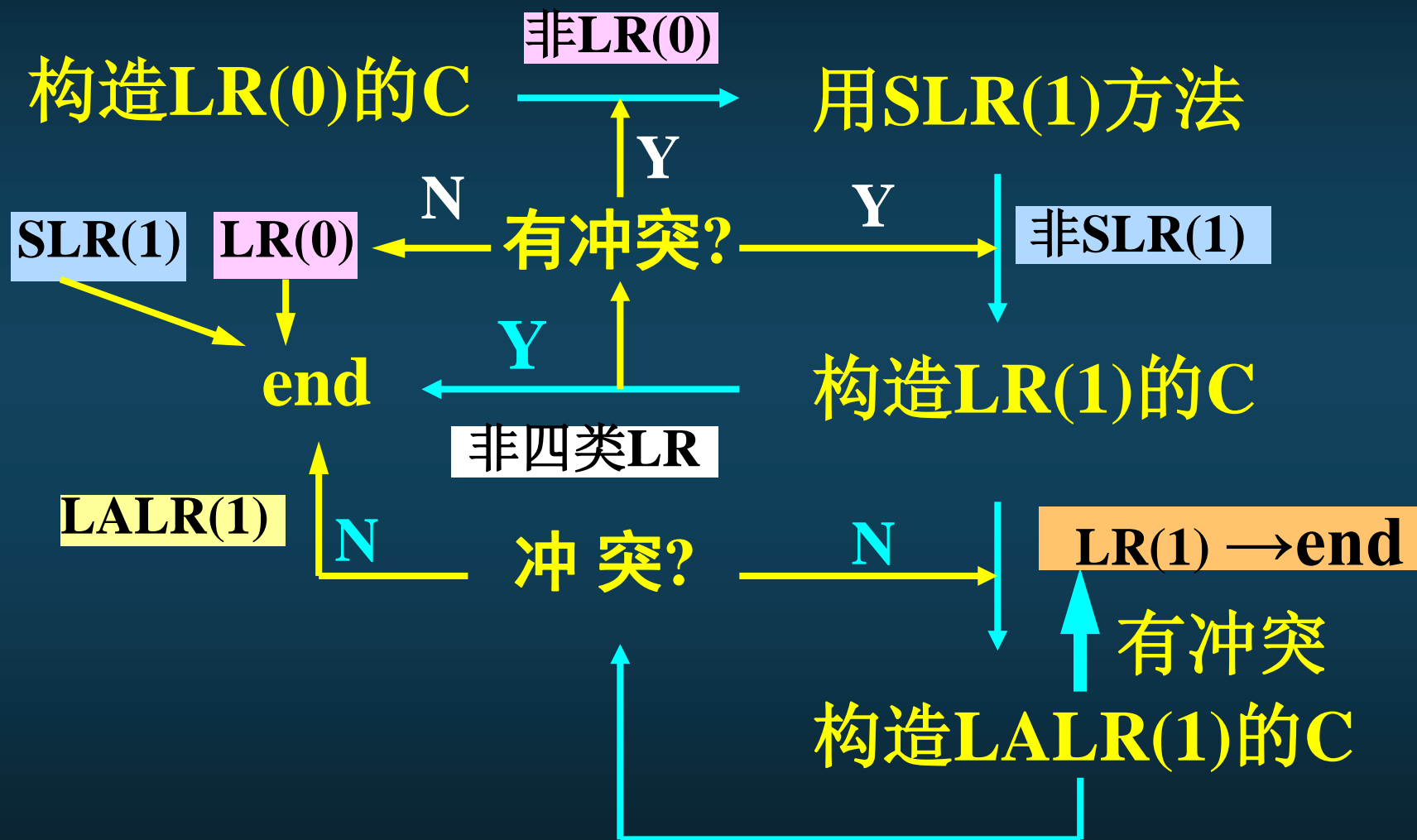
$A \rightarrow d$

$B \rightarrow d$

判断该文法是哪类LR文法？



LR(0)?SLR(1)?LR(1)?LALR(1)?

解题思路:

I_0

$S' \rightarrow \bullet S$
 $S \rightarrow \bullet Aa \mid \bullet bAc \mid$
 $\quad \bullet Bc \mid \bullet bBa$
 $A \rightarrow \bullet d$
 $B \rightarrow \bullet d$

d

 I_1

$A \rightarrow d \bullet$
 $B \rightarrow d \bullet$

归约—归约冲突

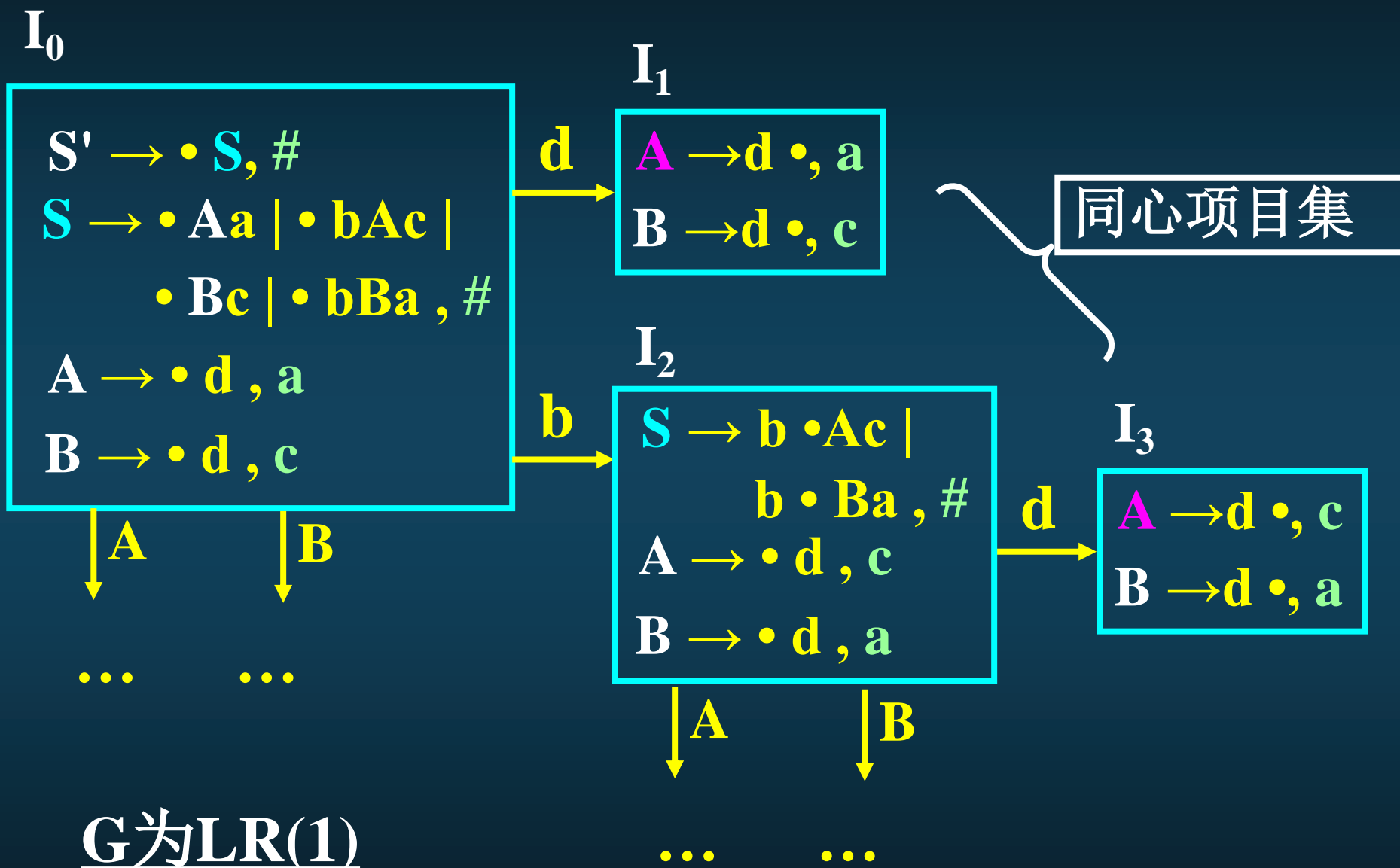
G为非LR(0)

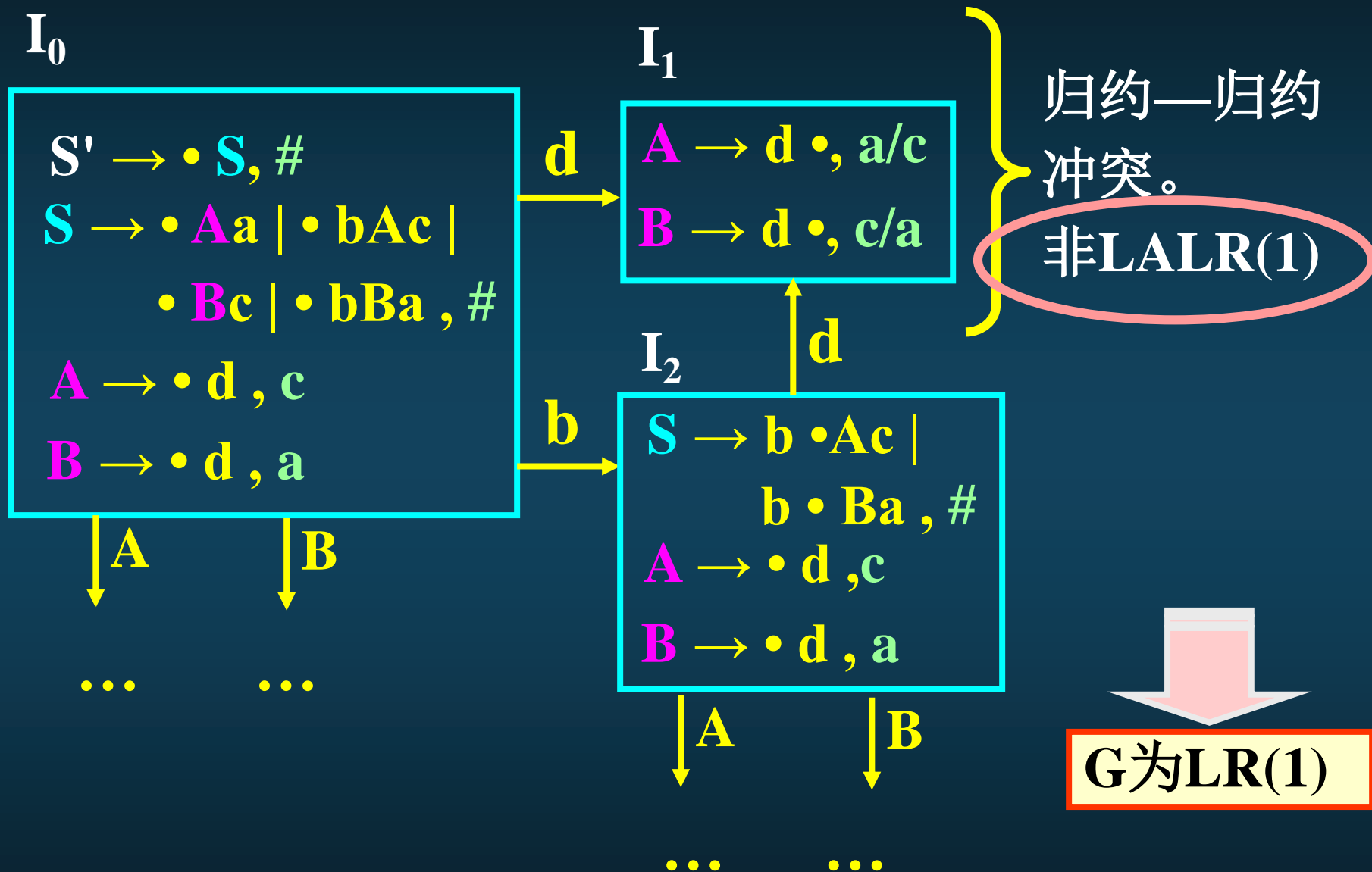
用SLR(1)方法

 $\text{FOLLOW}(A) = \{a, c\}$ $\text{FOLLOW}(B) = \{a, c\}$ $\cap \neq \emptyset$

构造LR(1)的C

G为非SLR(1)





■ 定理5.2

任何一个二义文法都不是一个LR文法。

- ** 二义文法会导致语法分析的二义性。
- ** 二义文法的有用之处在于可以缩小文法的规模，降低分析器开销。

例如,

$$G: E \rightarrow E+E \mid E * E \mid (E) \mid i$$
$$G': E \rightarrow T \mid E+T$$
$$T \rightarrow F \mid T * F$$
$$F \rightarrow (E) \mid i$$

例如,

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

或 $G': S'' \rightarrow S'$

$$S' \rightarrow iS' \mid eS' \mid a \mid \varepsilon$$

例5.19 设有文法 $G(E)$:

$$E \rightarrow E + E \quad ①$$

$$E \rightarrow E * E \quad ②$$

$$E \rightarrow (E) \quad ③$$

$$E \rightarrow i \quad ④$$

LR(0)的C
参见P128

L

I_1

$$\begin{array}{l} E' \rightarrow E \cdot \\ E \rightarrow E + E \\ E \rightarrow E * E \end{array}$$

I_7

$$\begin{array}{l} E \rightarrow E + E \cdot \\ E \rightarrow E + E \\ E \rightarrow E * E \end{array}$$

I_8

$$\begin{array}{l} E \rightarrow E * E \cdot \\ E \rightarrow E + E \\ E \rightarrow E * E \end{array}$$

$FOLLOW(E') =$
 $\{ \# \} \cap \{ + \} \cap \{ * \} = \varnothing$

$FOLLOW(E) =$
 $\{ +, *,), \# \} \cap \{ + \} \cap \{ * \} \neq \varnothing$

例5.19 设有文法 $G(E)$:

$E \rightarrow E + E$ ①

$E \rightarrow E * E$ ②

$E \rightarrow (E)$ ③

$E \rightarrow i$ ④

LR(0)的C
参见P128

L

I_1

$E' \rightarrow E \cdot$
 $E \rightarrow E + E$
 $E \rightarrow E * E$

I_7

$E \rightarrow E + E \cdot$
 $E \rightarrow E + E$
 $E \rightarrow E * E$

I_8

$E \rightarrow E * E \cdot$
 $E \rightarrow E + E$
 $E \rightarrow E * E$

$FOLLOW(E') =$
 $\{ \# \} \{ + \} \{ * \} \cap = \varnothing$

$FOLLOW(E) =$
 $\{ +, *,), \# \}, \{ + \}, \{ * \} \cap \neq \varnothing$

文法G(E)的LR分析表

状 态	ACTION						GOTO
	i	+	*	()	#	
0	s ₃			s ₂			1
1		s ₄	s ₅			acc	
2	s ₃			s ₂			6
3	r ₄	r ₄	r ₄	r ₄	r ₄	r ₄	
4	s ₃			s ₂			7
5	s ₃			s ₂			8
6		s ₄	s ₅		s ₉		
7	r ₁	r ₁	s ₅	r ₁	r ₁	r ₁	
8	r ₂	r ₂	r ₂	r ₂	r ₂	r ₂	
9	r ₃	r ₃	r ₃	r ₃	r ₃	r ₃	

\$: i+i+i #

↑
F

action(7, +) = r₁

7	E	← P } E
4	+	
1	E	
0	#	

\$: i+i*i #

↑
F

action(7, *) = S₅

5	*	
7	E	← P
4	+	
1	E	
0	#	

例5.20 设有文法G:

$$S' \rightarrow S \quad \textcircled{1}$$

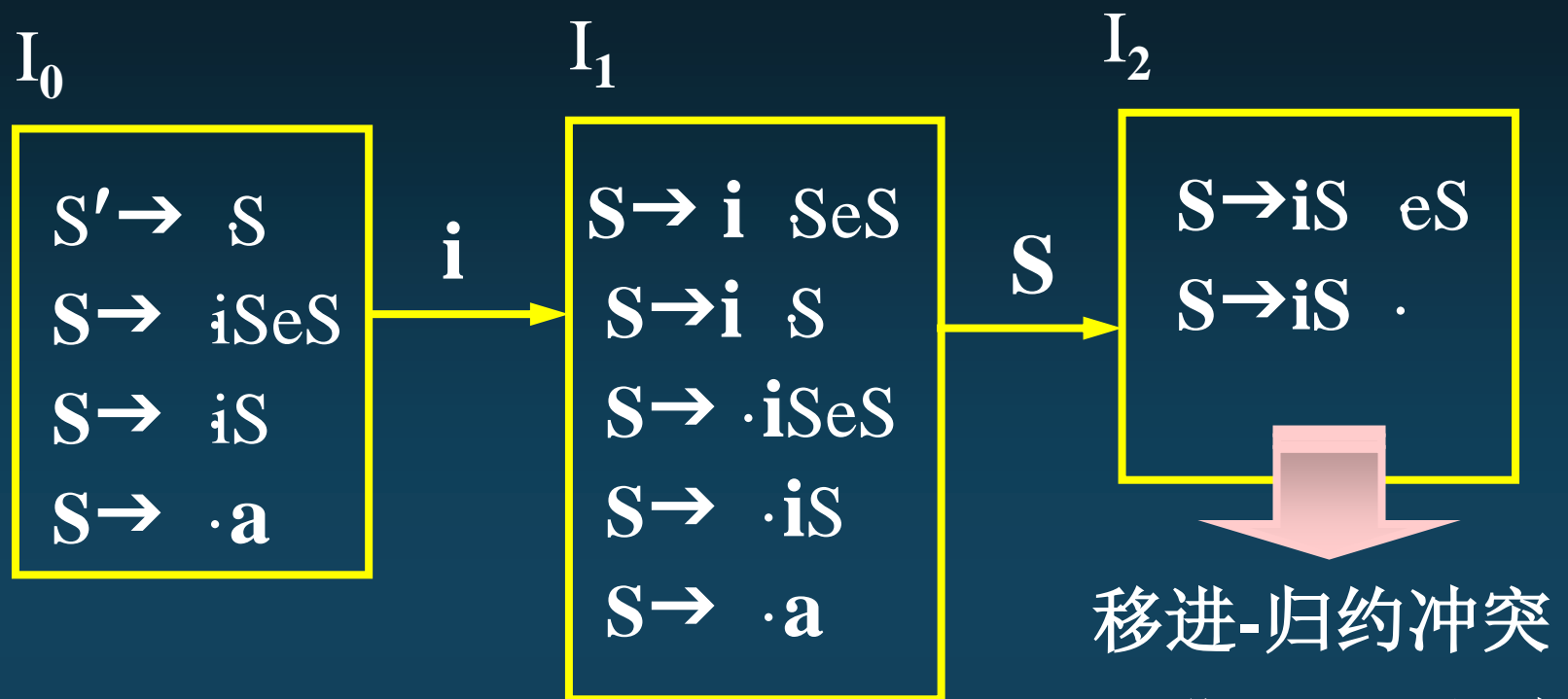
$$S \rightarrow iSeS \mid iS \mid a \quad \textcircled{2} \mid \textcircled{3} \mid \textcircled{4}$$

其中:

i : if e_r then

e: else

a, S: 语句

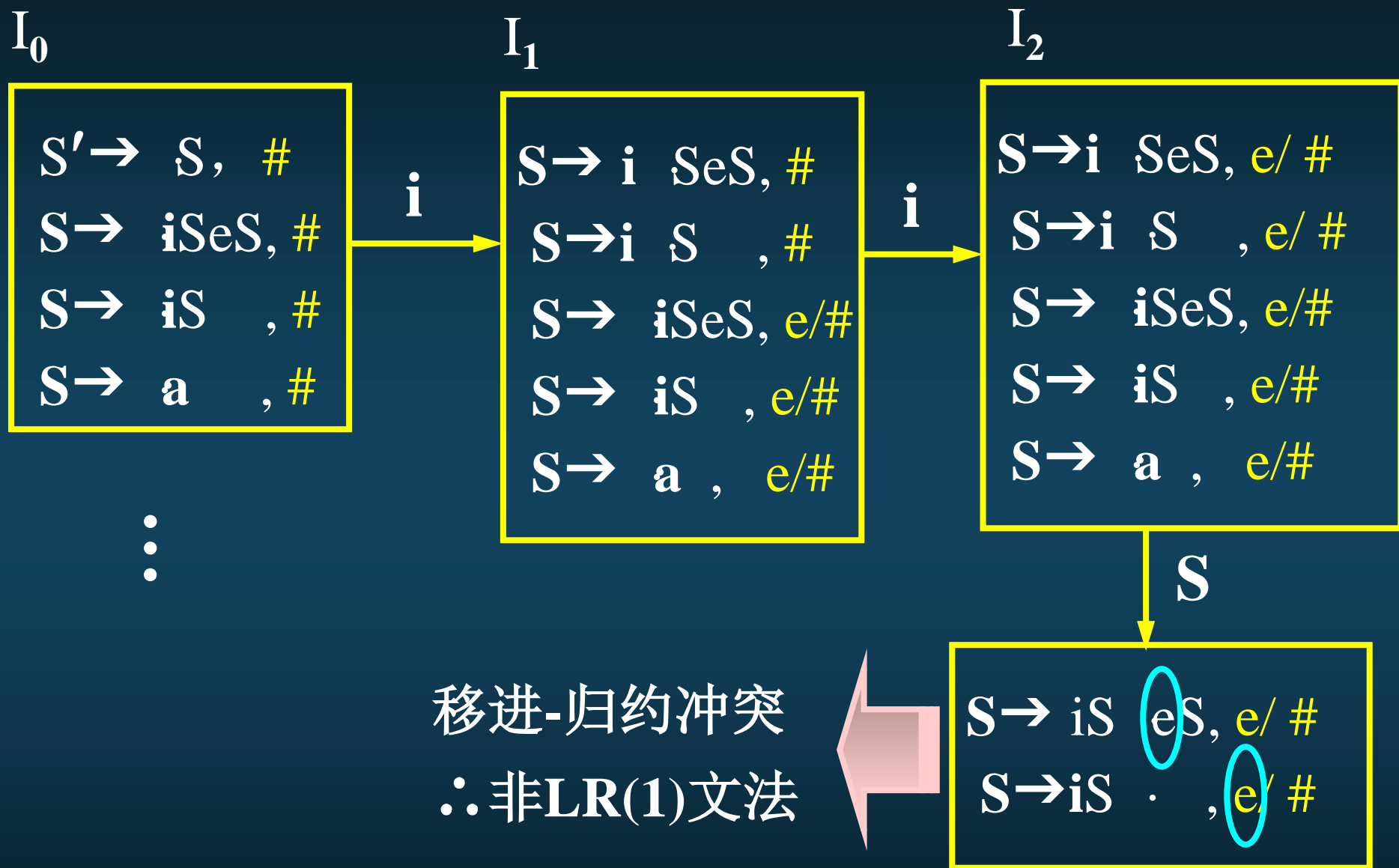


移进-归约冲突

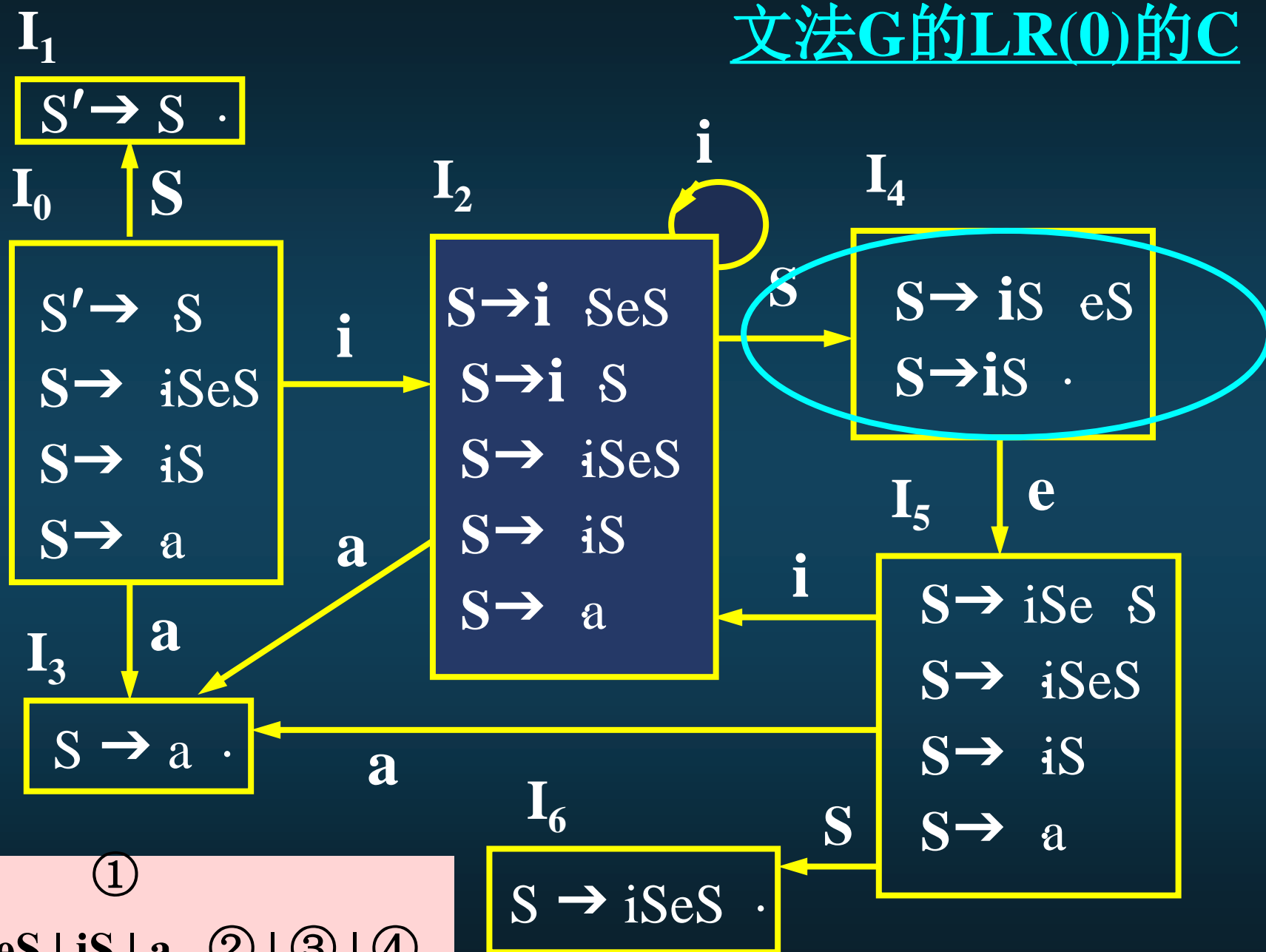
\therefore 非LR(0)文法

$\text{FOLLOW}(S) = \{e, \#\} \cap \{e\} \neq \Phi$

\therefore 非SLR(1)文法



文法G的LR(0)的C



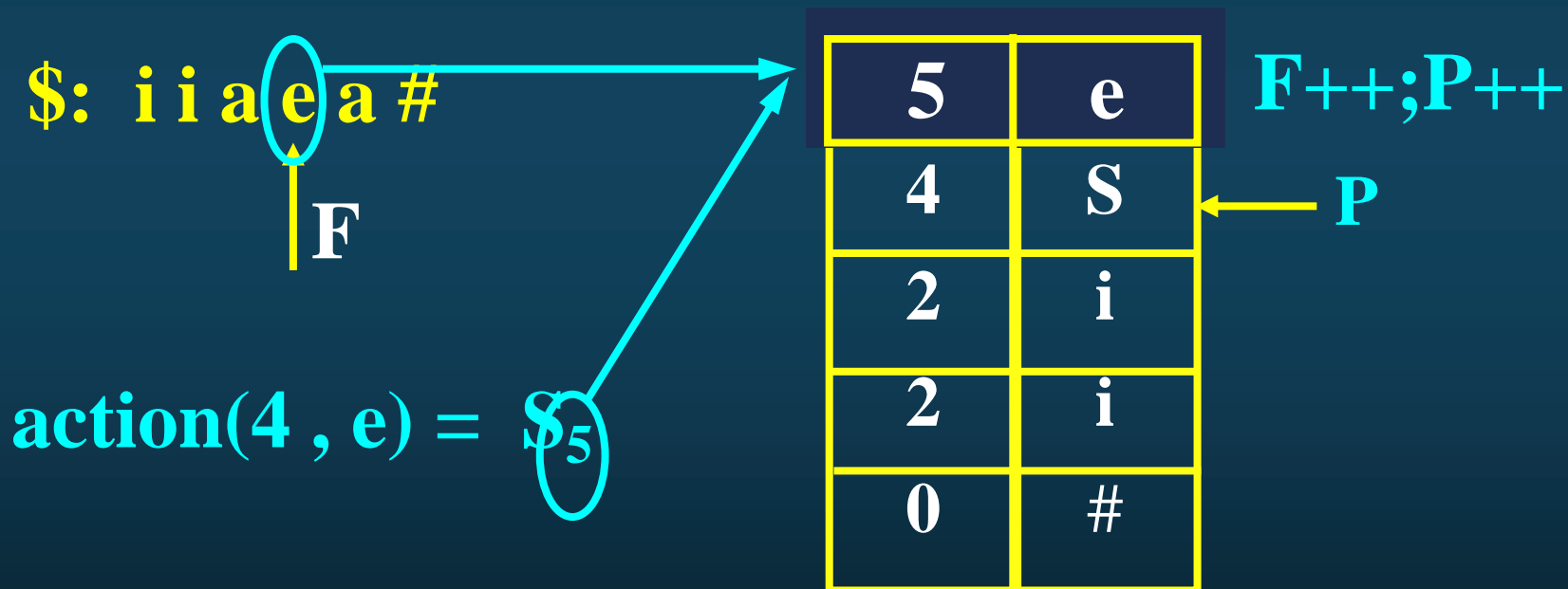
文法G的LR分析表

	i	e	a	#	S
0	S ₂		S ₃		1
1				acc	
2	S ₂		S ₃		4
3	r ₄	r ₄	r ₄	r ₄	
4	r ₃	S ₅	r ₃	r ₃	
5	S ₂		S ₃		6
6	r ₂	r ₂	r ₂	r ₂	



对 $\$ = \mathbf{iaea\#}$ 进行分析

if e_1 then	if e_2 then	S_1	else	S_2
i	i	a	e	a



■ 错误存在的必然性

ch5.9 错误处理

问题复杂性、程序员素质、输入错、系统环境生疏 ...

■ 编译中的错误种类

1. 词法错误；
2. 语法错误；
3. 语义错误(静态、动态)；
4. 违反环境限制的错误；

■ 错误处理基本目标

1. 清晰准确地**报告**错误（错误的定性和定位）；
2. 迅速从每个错误中**恢复**过来，以便诊断后面的错误；
3. 不使正确程序的处理**效率**降低。

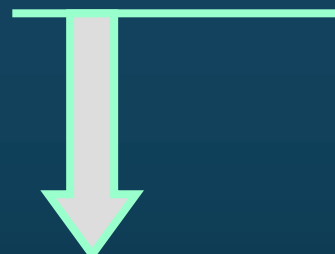
■ 错误恢复的困难

1. 源程序书写的灵活性;
2. 错误发生的随机性;
3. 可能校正途径的多样性。

■ 错误处理与恢复策略

1. 紧急恢复方式

编译器每次发现错误时，抛弃一个输入符号，直到输入符号属于某个指定的同步符号集合为止。



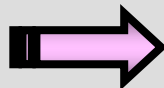
例如，界限符：

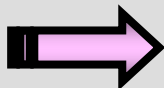
； 、 }、 end ...

2. 短语级恢复

发现错误时，对剩余符号串作局部校正。即使用可以是编译器继续工作的输入串代替剩余输入的前缀。

* 关键：选择合适的替换串。

(i+i ;  (i+i) ;

i i*i ;  i + i*i ;

3. 出错产生式

扩充语言的文法，增加产生错误结构的产生式。分析中可以直接识别处理错误。

4. 全局纠正

获取全局最小代价纠正。

■ 局部化紧急恢复的一般性技术

1. 在输入串的**出错点**采用**插入、删除或修改**的方法。
2. 在分析到某一含有错误的短语时，该短语不能与语法任一个非终结符能推出的符号串匹配，则采取将后续的输入字符移进栈内，实际是**跳过部分源程序**，直至找到能推出该短语的非终结符号的跟随字符为止，其实质是将含有错误的短语局部化。

例5.19 设有文法G(E):

$$E \rightarrow E + E \quad \textcircled{1}$$

$$E \rightarrow E * E \quad \textcircled{2}$$

$$E \rightarrow (E) \quad \textcircled{3}$$

$$E \rightarrow i \quad \textcircled{4}$$

构造文法G(E)的LR(0)项目集规范族

参 P128

e1: 处于状态0、2、4、5时，要求输入符号为运算对象 **i** 或 **(E)**，此时若遇到+、*、#，则调用e1。

e1功能: 将假设的 **i** 和状态**3**入栈；(**shift**)

e1出错信息: “缺少运算对象”。

link

e2: 处于状态0、1、2、4、5时，若遇到“）”，则调用e2。

e2功能: 删除输入的“）”；(**F++**)

e2出错信息: “括号不配对”。

link

e3: 处于状态1或6时，期望下面输入符为运算符或“)”，但遇到“i”或“(”时，则调用e3。

e3功能: 将假设的“+”和状态4入栈, (P++);

e3出错信息: “缺少运算符”。

link

e4: 处于状态6时，期望下面输入符为运算符或“)”，但遇到“#”时，则调用e4。

e4功能: 将假设的“)”和状态9入栈, (P++);

e4出错信息: “缺少 ‘)’”。

link

文法G(E)的LR分析表

状 态	ACTION						GOTO
	i	+	*	()	#	
0	s ₃	e1	e1	s ₂		e1	1
1		s ₄	s ₅			acc	
2	s ₃	e1	e1	s ₂		e1	6
3	r ₄	r ₄	r ₄	r ₄	r ₄	r ₄	
4	s ₃	e1	e1	s ₂		e1	7
5	s ₃	e1	e1	s ₂		e1	8
6		s ₄	s ₅		s ₉		
7	r ₁	r ₁	s ₅	r ₁	r ₁	r ₁	
8	r ₂	r ₂	r ₂	r ₂	r ₂	r ₂	
9	r ₃	r ₃	r ₃	r ₃	r ₃	r ₃	

文法G(E)的LR分析表

状 态	ACTION						GOTO
	i	+	*	()	#	E
0	s ₃	e1	e1	s ₂	e2	e1	1
1		s ₄	s ₅		e2	acc	
2	s ₃	e1	e1	s ₂	e2	e1	6
3	r ₄	r ₄	r ₄	r ₄	r ₄	r ₄	
4	s ₃	e1	e1	s ₂	e2	e1	7
5	s ₃	e1	e1	s ₂	e2	e1	8
6		s ₄	s ₅		s ₉		
7	r ₁	r ₁	s ₅	r ₁	r ₁	r ₁	
8	r ₂	r ₂	r ₂	r ₂	r ₂	r ₂	
9	r ₃	r ₃	r ₃	r ₃	r ₃	r ₃	

文法G(E)的LR分析表

状 态	ACTION						GOTO
	i	+	*	()	#	
0	s ₃	e1	e1	s ₂	e2	e1	1
1	e3	s ₄	s ₅	e3	e2	acc	
2	s ₃	e1	e1	s ₂	e2	e1	6
3	r ₄	r ₄	r ₄	r ₄	r ₄	r ₄	
4	s ₃	e1	e1	s ₂	e2	e1	7
5	s ₃	e1	e1	s ₂	e2	e1	8
6	e3	s ₄	s ₅	e3	s ₉		
7	r ₁	r ₁	s ₅	r ₁	r ₁	r ₁	
8	r ₂	r ₂	r ₂	r ₂	r ₂	r ₂	
9	r ₃	r ₃	r ₃	r ₃	r ₃	r ₃	

文法G(E)的LR分析表(带出错处理)

状 态	ACTION						GOTO
	i	+	*	()	#	E
0	s ₃	e1	e1	s ₂	e2	e1	1
1	e3	s ₄	s ₅	e3	e2	acc	
2	s ₃	e1	e1	s ₂	e2	e1	6
3	r ₄	r ₄	r ₄	r ₄	r ₄	r ₄	
4	s ₃	e1	e1	s ₂	e2	e1	7
5	s ₃	e1	e1	s ₂	e2	e1	8
6	e3	s ₄	s ₅	e3	s ₉	e4	
7	r ₁	r ₁	s ₅	r ₁	r ₁	r ₁	
8	r ₂	r ₂	r ₂	r ₂	r ₂	r ₂	
9	r ₃	r ₃	r ₃	r ₃	r ₃	r ₃	

\$1: (i+i # $\xrightarrow{\text{F}}$ \$: (i+i) # $\xrightarrow{\text{F}}$ action(9, #) = r₃

E {

7	E
4	+
6	E
2	(
0	#

P ←

E {

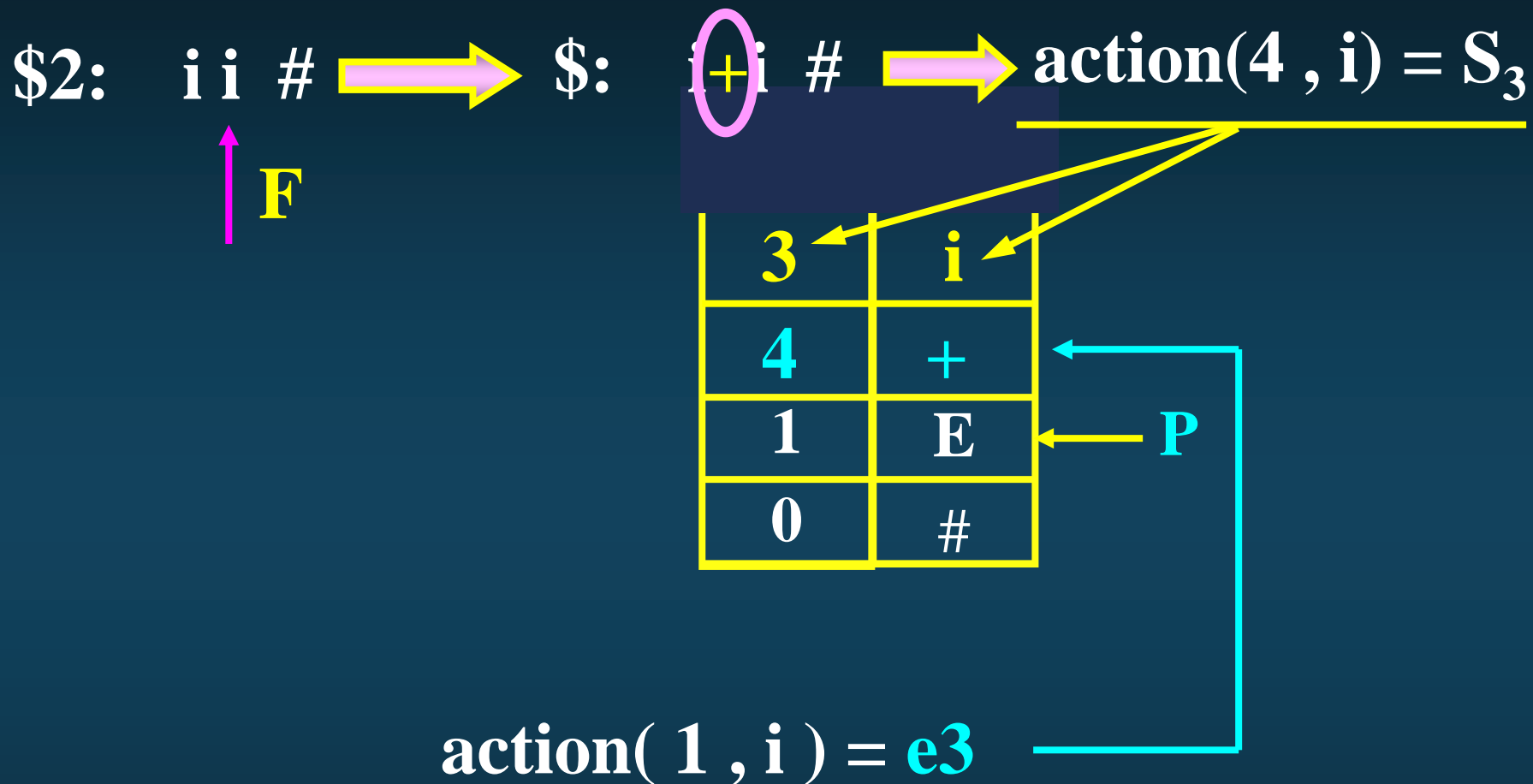
9)
6	E
2	(
0	#

P ←

goto(2, E) = 6

action(6, #) = e4

action(7, #) = r₁



YACC (**Y**et **A**nother **C**ompiler-**C**ompiler)
parser generator

Bison (GNU工程推出，与YACC向上兼容。)

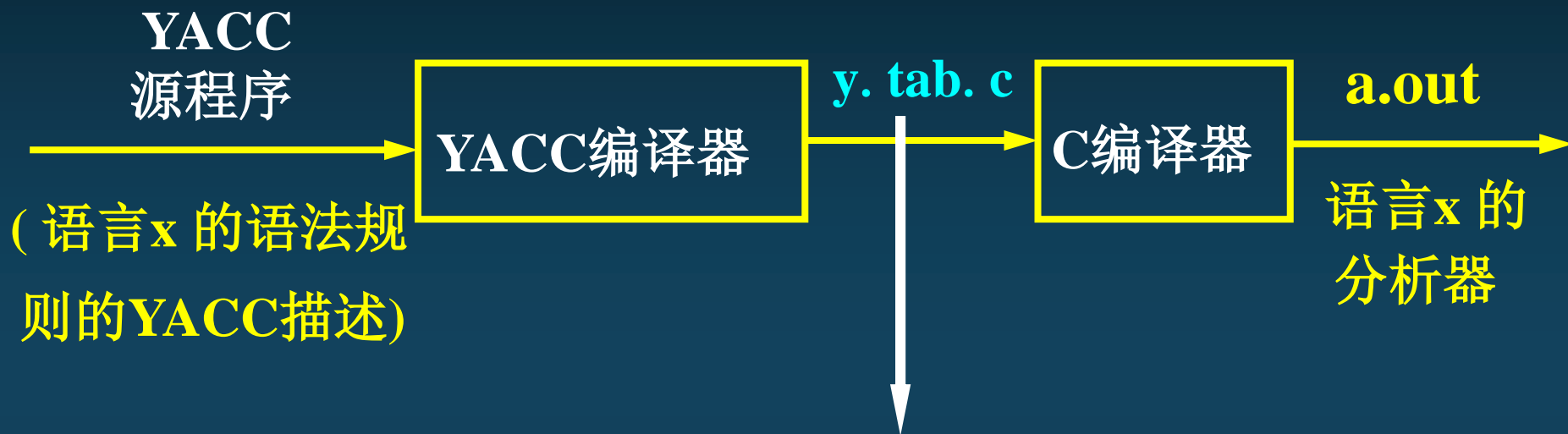
1. 是一个LALR(1)分析器自动生成器；
2. 与LEX有标准接口；
3. 适应二义文法的LALR(1)分析；
4. 可带语义处理。

■ Compiler — Compiler思想



$G \longrightarrow \text{YACC描述}$

■ YACC自动构造分析器的模式

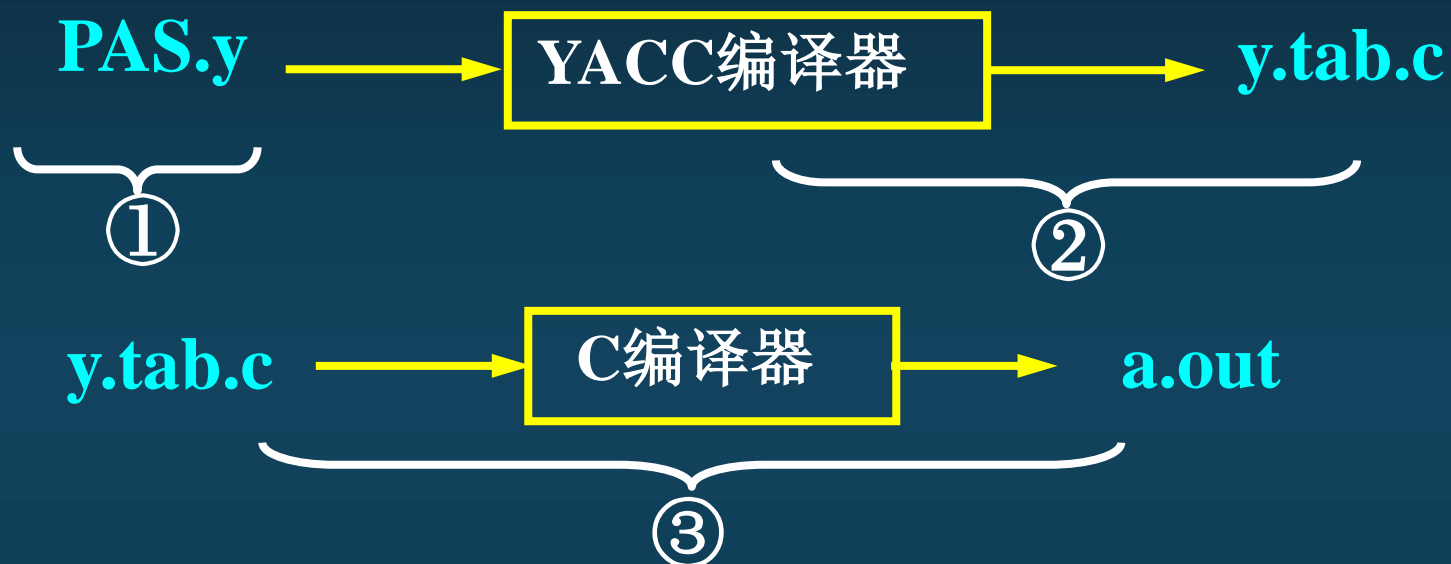


UNIX命令 (编译y.tab.c)

cc y.tab.c -ly

其中：ly表示使用LR分析器的库（ly随系统而定）。

■ 使用YACC步骤

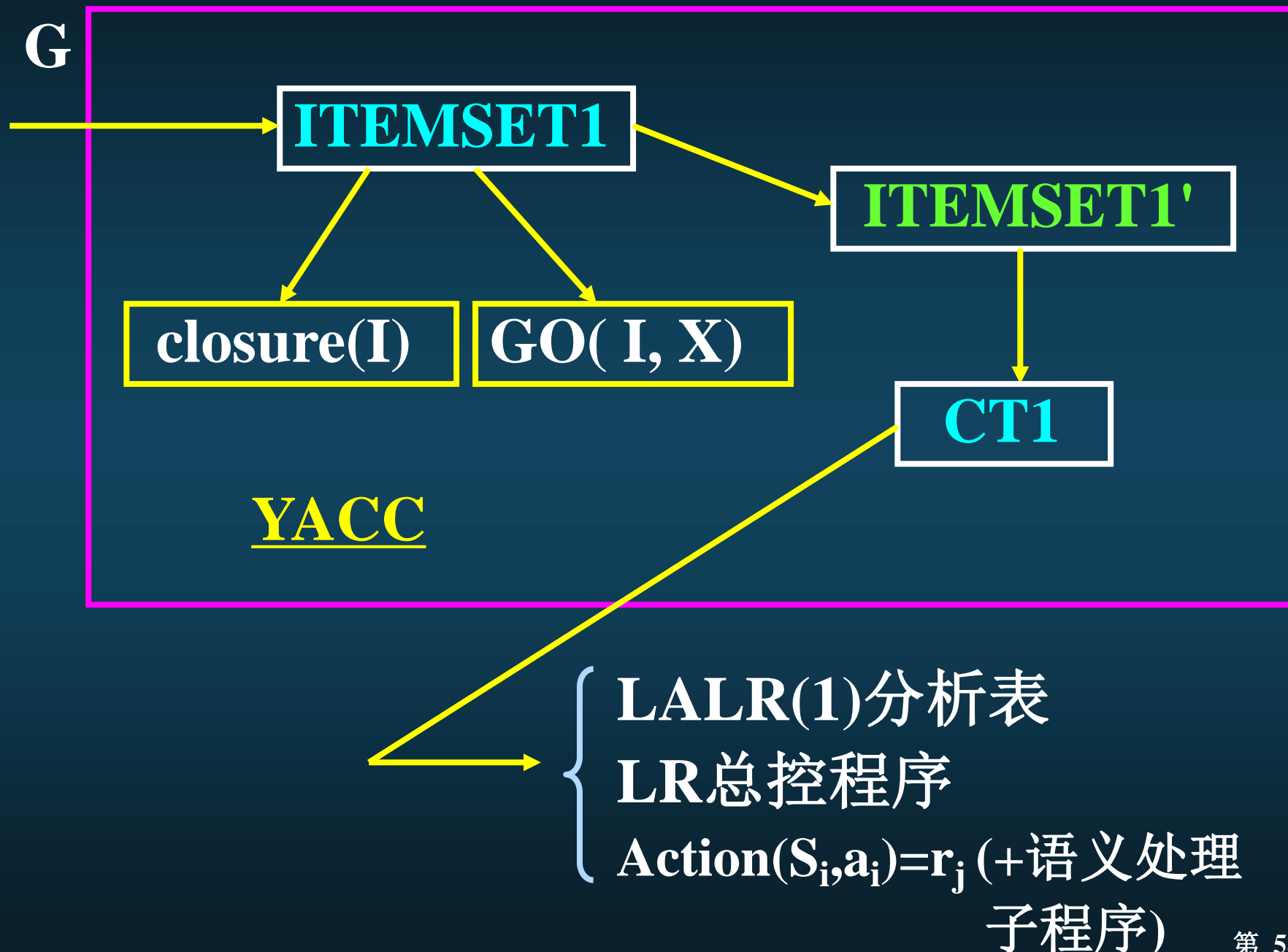


- ① 编辑YACC源程序 (例如, 生成文本格式的PAS语言的YACC源文件PAS.y);
- ② 使用命令 `yacc PAS.y` 运行YACC, 正确则输出 `y.tab.c`;
- ③ 调用C编译器编译 `cc y.tab.c`, 并与其它C模块连接产生执行文件; 调试执行文件, 直至获得正确输出。

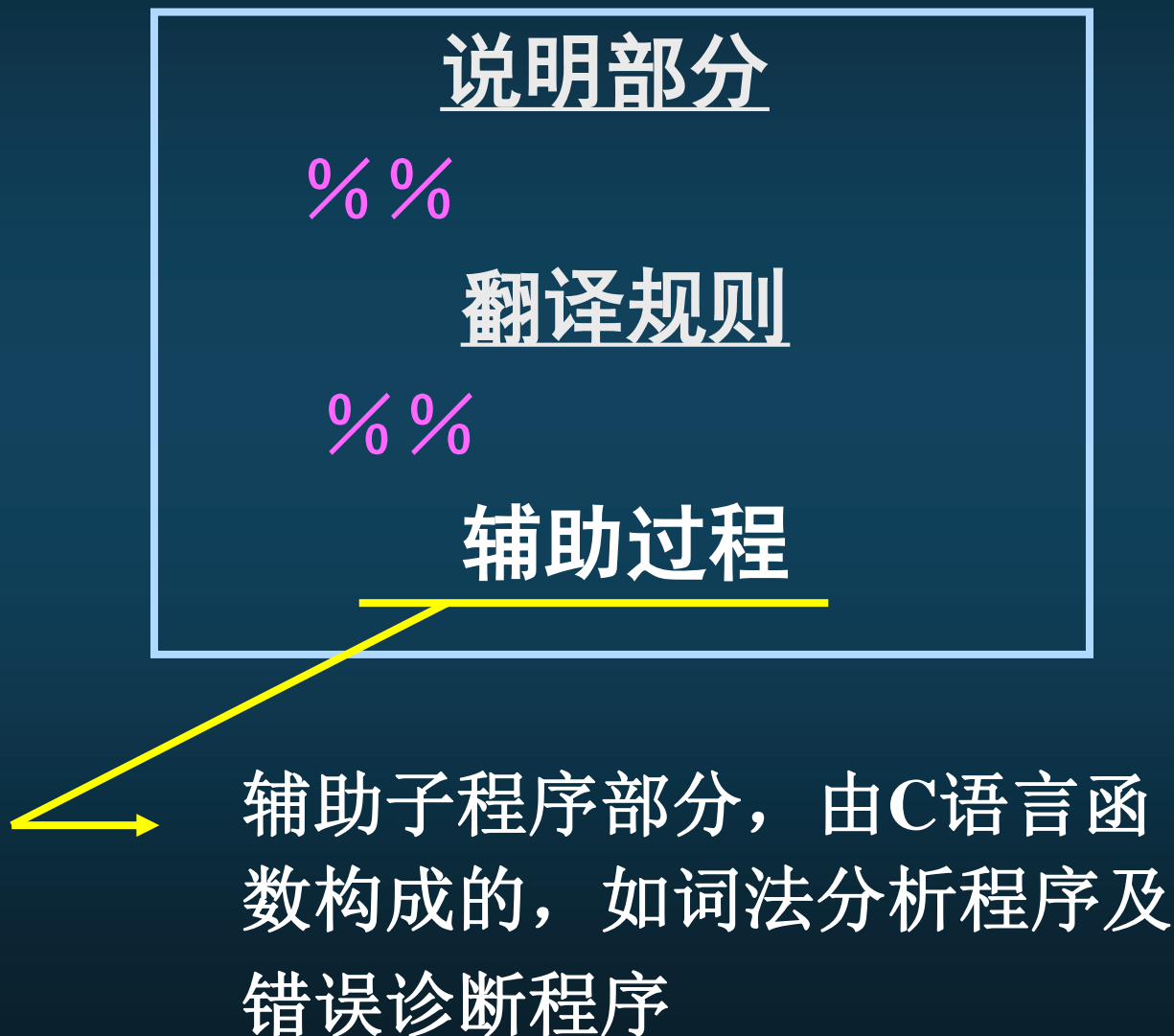
YACC体系 { YACC语言
YACC 编译器

YACC实现 (语法分析器自动生成原理)

1. 构造LR(1)的C (ITEMSET1 — 算法5.9) ;
2. 查找、合并同心项目集, 构造LALR(1)的C (ITEMSET1') ;
3. 构造LALR(1)的分析表 (CT1— 算法5.11);



■ YACC源程序结构



■ 说明部分

- `%{ %}` C语言程序的常规说明 (头文件 / 宏定义 ...)
- 文法符号(一般为终结符)和文法规则的说明
(对文法规则说明的一些限定规则和条件的声明)
 - `% 说明内容1`
 - `% 说明内容2`
 - `.....`

常规说明

%{

include <ctype.h>

include <stdio.h>

define YYSTYPE double /* YACC栈定义为double类型 */

%}

% token NUMBER

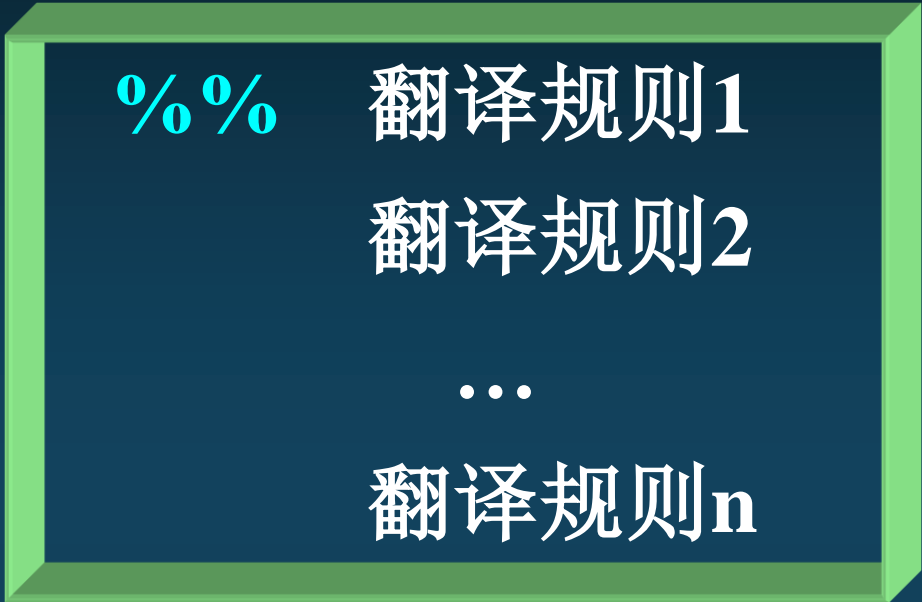
% terminal '+' '-' Left

% terminal '*' '/' Left

% Right UMINUS

文法符号和文法规则的说明

■ 翻译规则部分



%% 翻译规则1
翻译规则2
...
翻译规则n

翻译规则 i : 文法产生式 { 语义动作 }

BNF

<左部文法符号>

→ <候选式1> | <候选式2> | ... | <候选式 n >

用YACC描述的一般形式为:

<左部文法符号>: <候选式1>
 | <候选式2>

 | <候选式 n >
 ;

{语义动作1}

{语义动作2}

{语义动作 n }

例5.21 设文法 $G(E)$ 为 $A \rightarrow E+E \mid E * E \mid \text{number}$
文法 $G(A)$ 的YACC源程序如下:

```
%{
    #include <ctype.h>
    #include <stdio.h>
%}
% token number

%%
    lines: lines expr '\n'
    expr : expr '+' expr
          : expr '*' expr
          : number
    ;
%%
```

```
{ printf("%g\n", $2); }
{ $$ = $1+$3; }
{ $$ = $1*$3; }
```

(辅助过程)

例5.22 设文法G(E)为

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid -E \mid \text{NUMBER}$

%{

include <ctype.h>

include <stdio.h>

define YYSTYPE double /* YACC栈定义为double类型 */

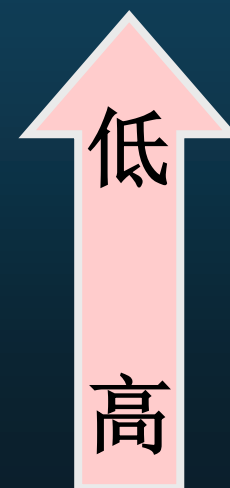
%}

% token NUMBER

% terminal '+' '-' Left

% terminal '*' '/' Left

% Right UMINUS



%%

lines: lines expr '\n'

{ printf("%g\n", \$\$);

| lines '\n'

| /* ϵ */

;

expr: expr '+' expr { \$\$=\$1+\$3;}

| expr '-' expr { \$\$=\$1-\$3;}

| expr '*' expr { \$\$=\$1*\$3;}

| expr '/' expr { \$\$=\$1/\$3;}

| '(' expr ')' { \$\$=\$2;}

| '-' expr % prec UMINUS { \$\$=-\$2;}

| NUMBER

;