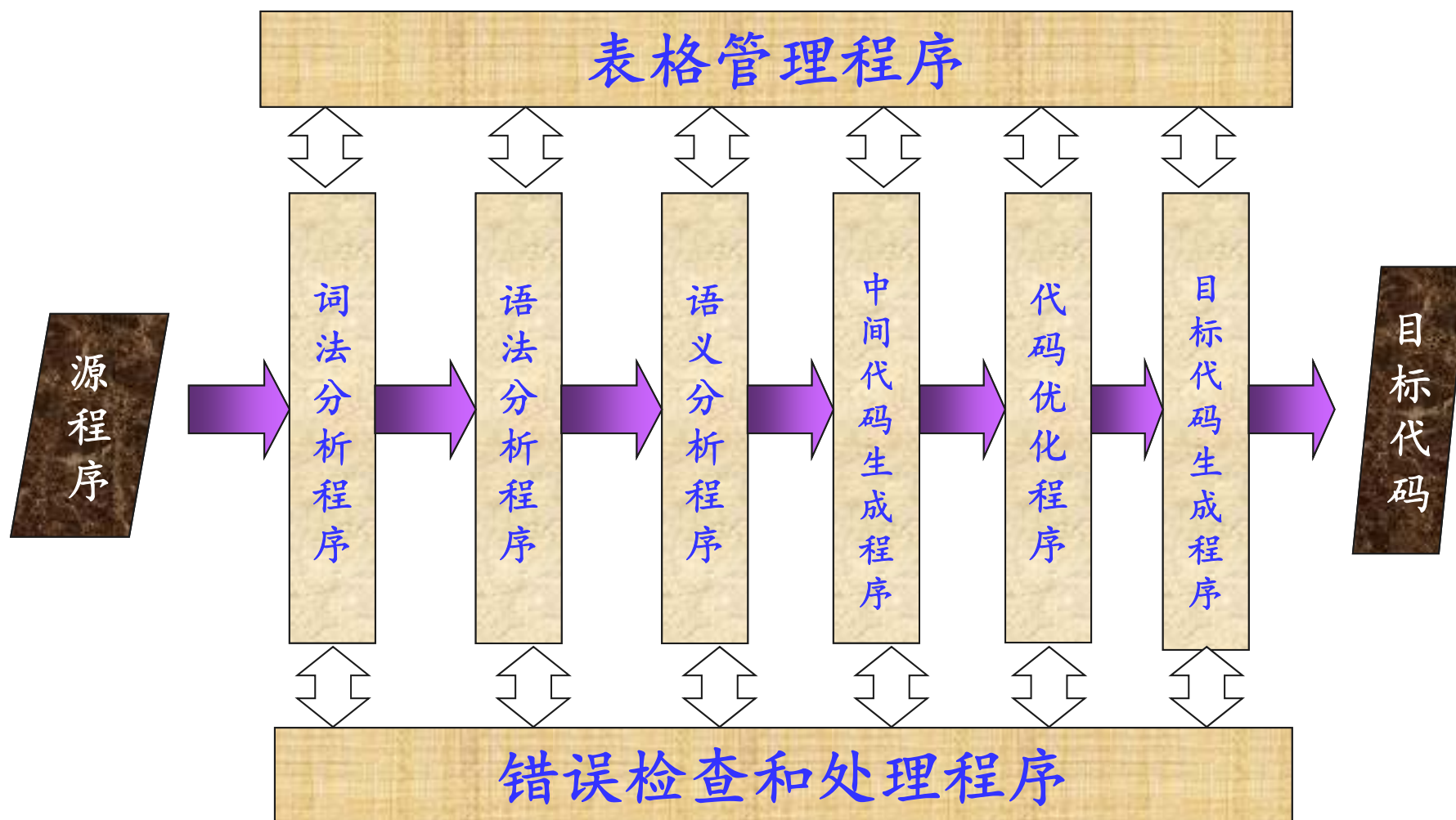


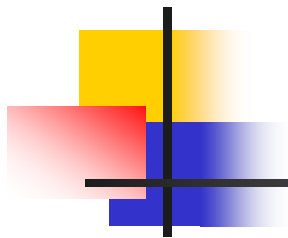


编译原理

武汉大学计算机学院
编译原理课程组

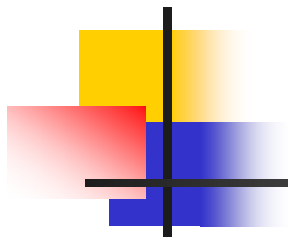
编译程序的结构





前述内容回顾 —— 优先分析法

- 基本思想 定义文法符号被归约的优先级
- 简单优先分析法 简单优先关系的构造
- 算符优先分析法 算符优先关系的构造



第7章 自下而上LR(k)分析方法

- 基本思想
- 存在的问题
- 解决方法
- LR分析方法
- 二义性文法的LR分析

Donald. E. Knuth (高德纳)



《The Art of Computer Programming》

- ◆ 第1卷/基本算法
- ◆ 第2卷/半数值算法
- ◆ 第3卷/排序和查找
- ◆ 第4卷/组合算法
- ◆ 第5卷/语法算法——词法分析、语法分析
- ◆ 第6卷/语言理论——文法、自动机
- ◆ 第7卷/编译程序



1965年, Knuth提出了LR(k)分析法。

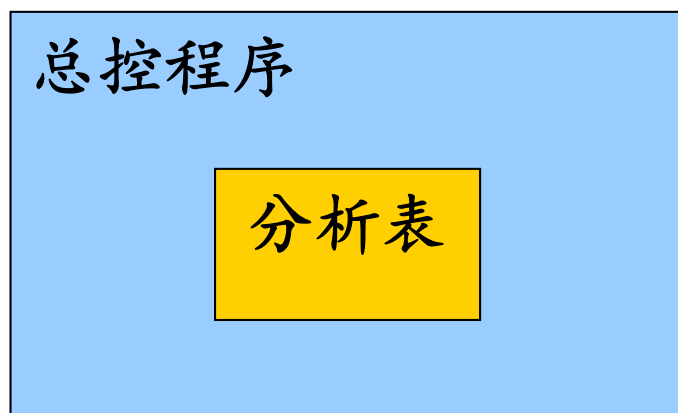




7.1 LR分析法和LR分析程序

1. LR分析程序的逻辑结构

包括两部分：一个总控(驱动)程序和一张分析表。



7.1 LR分析法和LR分析程序

1. LR分析程序的逻辑结构

设置一个栈，栈中每个元素包含两部分：状态与文法符号。

状态栈	q_0	q_1	q_2	...	q_m
符号栈	#	X_1	X_2	...	X_m

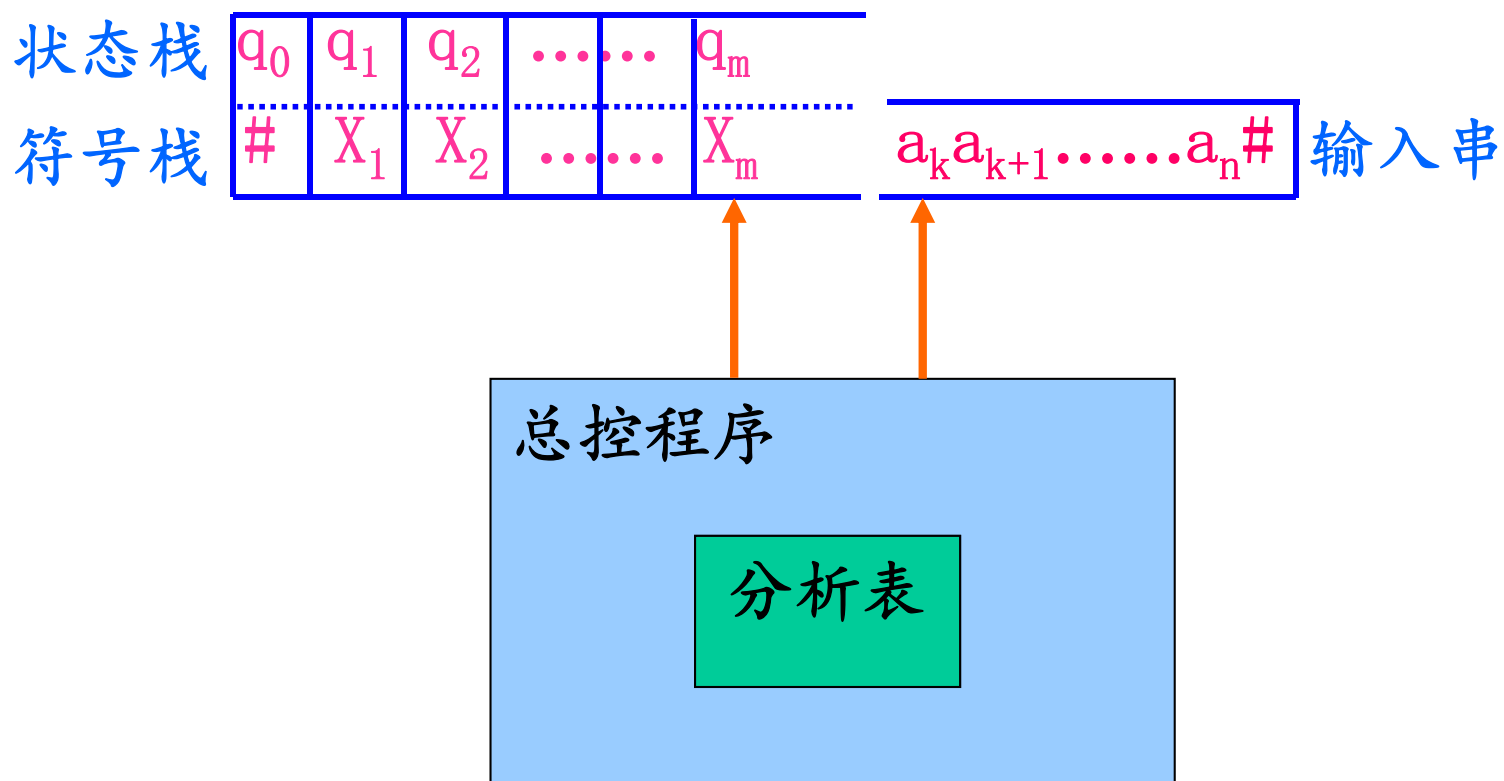
总控程序

分析表

7.1 LR分析法和LR分析程序

1. LR分析程序的逻辑结构

输入符号串，总控程序根据分析表分析并给出判断结果。





7.1 LR分析法和LR分析程序

1. LR分析程序的逻辑结构——分析表

分析表是LR分析程序的核心部分，它有“动作”(ACTION)和“状态转换”(GOTO)两部分。

ACTION和GOTO都是二维数组：

ACTION[q, a]——当状态q面临输入符号a时应采取什么动作；

GOTO[q, X]——状态q面对文法符号X时下一状态是什么。



7.1 LR分析法和LR分析程序

1. LR分析程序的逻辑结构——分析表

G[E]:

1 $E \rightarrow E+T$

2 $E \rightarrow T$

3 $T \rightarrow T * F$

4 $T \rightarrow F$

5 $F \rightarrow (E)$

6 $F \rightarrow i$

G[E]:

1 $E \rightarrow E+T$

2 $E \rightarrow T$

3 $T \rightarrow T * F$

4 $T \rightarrow F$

5 $F \rightarrow (E)$

6 $F \rightarrow i$

状态	ACTION						GOTO		
	i	+	*	()	#	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			



7.1 LR分析法和LR分析程序

1. LR分析程序的逻辑结构——分析表

所有的LR分析器的总控程序都是一样的，只是分析表因文法不同而各有不同。主要解决分析表产生器是怎样生成分析表的。

常见的构造LR分析表的方法有四种：LR(0)分析表构造，SLR分析表构造，LR(1)分析表构造法，LALR分析表构造法。相应地，利用这四种不同分析表可得到四种不同的LR分析法。



7.1 LR分析法和LR分析程序

2. LR分析过程

总控程序在分析的每一步，按照状态栈顶状态 q 和当前输入符号 a ，查阅LR分析表，并执行其中ACTION[q, a]和GOTO部分规定的操作。

用一个三元式表示分析的每一步栈中状态 q 、文法符号 X 和输入符号串 a 的变化情况，如下所示：

$$(q_0 q_1 \dots q_i, \#X_1 X_2 \dots X_i, a_k a_{k+1} \dots a_n \#)$$

初始状态： $(q_0, \#, a_1 a_2 \dots a_n \#)$



7.1 LR分析法和LR分析程序

2. LR分析过程

分析到某个时刻: $(q_0 q_1 \dots q_i, \#X_1 X_2 \dots X_i, a_k a_{k+1} \dots a_n \#)$

i. 若 $\text{ACTION}[q_i, a_k] = S_j$, 则三元式变为

$(q_0 q_1 \dots q_i q_j, \#X_1 X_2 \dots X_i a_k, a_{k+1} \dots a_n \#)$

ii. 若 $\text{ACTION}[q_i, a_k] = r_j$, 且第j条产生式为 $U \rightarrow x$, $|x|=m$, 且LR分析表中有 $\text{GOTO}[q_{i-m}, U] = q_t$, 则三元式变为

$(q_0 q_1 \dots q_{i-m} q_t, \#X_1 X_2 \dots X_{i-m} U, a_k a_{k+1} \dots a_n \#)$

iii. 若 $\text{ACTION}[q_i, a_k] = \text{acc}$, 表示接受, 三元式不变, 分析成功。

iv. 若 $\text{ACTION}[q_i, a_k] = \text{ERROR}$, 表示出错, 中止三元式的变化。



7.1 LR分析法和LR分析程序

3. LR分析举例

G[E]:

1 $E \rightarrow E+T$

2 $E \rightarrow T$

3 $T \rightarrow T * F$

4 $T \rightarrow F$

5 $F \rightarrow (E)$

6 $F \rightarrow i$

G[E]:

1 $E \rightarrow E+T$

2 $E \rightarrow T$

3 $T \rightarrow T * F$

4 $T \rightarrow F$

5 $F \rightarrow (E)$

6 $F \rightarrow i$

$i * i + i$

状态	ACTION						GOTO		
	i	+	*	()	#	E	T	F
0	S ₅			S ₄			1	2	3
1		S ₆				acc			
2		r ₂	S ₇		r ₂	r ₂			
3		r ₄	r ₄		r ₄	r ₄			
4	S ₅			S ₄			8	2	3
5		r ₆	r ₆		r ₆	r ₆			
6	S ₅			S ₄				9	3
7	S ₅			S ₄					10
8		S ₆			S ₁₁				
9		r ₁	S ₇		r ₁	r ₁			
10		r ₃	r ₃		r ₃	r ₃			
11		r ₅	r ₅		r ₅	r ₅			



7.2 LR(0) 分析表的构造

1. LR分析法的基本原理

例如，对文法G[S]:

$$S \rightarrow aAcBe$$
$$A \rightarrow b$$
$$A \rightarrow Ab$$
$$B \rightarrow d$$

分析符号串abbcde。

遇到的两个问题：

问题1) 选择哪个子串作为句柄？

问题2) 选择哪个产生式进行归约？



7.2 LR(0) 分析表的构造

1. LR分析法的基本原理

例如，对文法G[S]:

$S \rightarrow aAcBe$

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$

分析符号串abbcde。

句柄的识别是一个符号一个符号得到的，若将从左到右每识别得到句柄的一个符号就对应着一个状态，则可将n个符号的句柄的识别分成n+1个状态。



7.2 LR(0) 分析表的构造

1. LR分析法的基本原理

句柄的识别是一个符号一个符号得到的，若将从左到右每识别得到句柄的一个符号就对应着一个状态，则可将 n 个符号的句柄的识别分成 $n+1$ 个状态。

用LR(0)项目来表示一个句柄的所有识别状态。



7.2 LR(0) 分析表的构造

LR(0)项目：

文法的产生式右部某位置加一个‘•’。

产生式右部有n个符号，就有n+1个对应的**LR(0)项目**，

它表示该句柄（即该产生式右部的串）的所有识别状态。

从左到右，每识别句柄的一个符号就对应一个状态，识别过程即状态转换过程（GOTO）。—— **DFA**

例如：

$A \rightarrow xyz$ 对应了4个LR(0)项目。

————→

[$A \rightarrow \bullet xyz$]

[$A \rightarrow x \bullet yz$]

[$A \rightarrow xy \bullet z$]

[$A \rightarrow xyz \bullet$]



7.2 LR(0) 分析表的构造

1. LR分析法的基本原理

句柄的识别是一个符号一个符号得到的，若将从左到右每识别得到句柄的一个符号就对应着一个状态，则可将 n 个符号的句柄的识别分成 $n+1$ 个状态。

用LR(0)项目来表示一个句柄的所有识别状态。

在得到一个规范句型的完整句柄（可归前缀入栈）之前所识别的符号串称为规范句型的活前缀（规范前缀）。

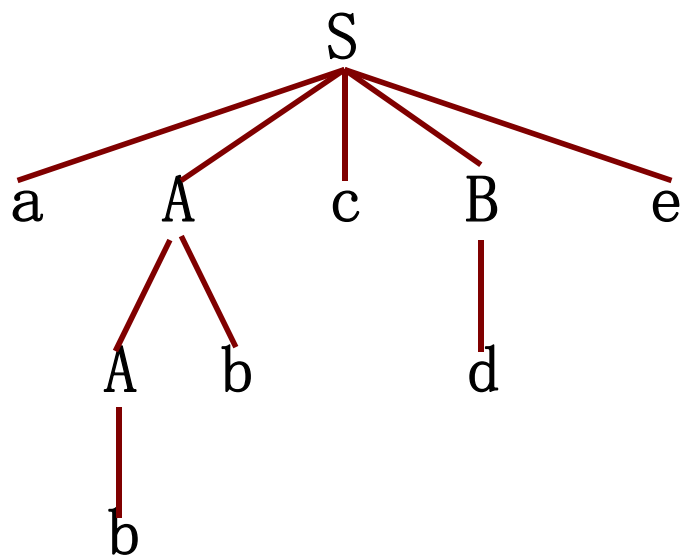
7.2 LR(0) 分析表的构造

$G[S]: S \rightarrow aAcBe$

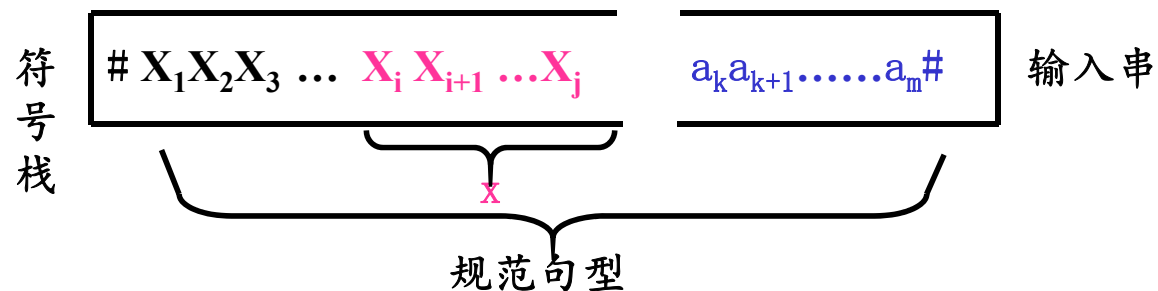
$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$



abbcbde对应的语法树



规范句型	句柄	活前缀
a bbcbde	b	ϵ , a, ab
a A bbcbde	Ab	ϵ , a, aA, aAb
aAc d e	d	ϵ , a, aA, aAc, aAcd
aAcBe	aAcBe	ϵ , a, aA, aAc, aAcB, aAcBe



7.2 LR(0) 分析表的构造

1. LR分析法的基本原理

句柄的识别是一个符号一个符号得到的，若将从左到右每识别得到句柄的一个符号就对应着一个状态，则可将 n 个符号的句柄的识别分成 $n+1$ 个状态。

用LR(0)项目来表示一个句柄的所有识别状态。

在得到一个规范句型的完整句柄之前所识别的符号串称为规范句型的活前缀。

只要分析的过程中已经识别得到的符号串是一个活前缀，就说明分析过程是正确的。



7.2 LR(0) 分析表的构造

1. LR分析法的基本原理

对句柄的识别过程就是对规范句型的活前缀的识别过程。

如果能够构造出一个文法的所有规范句型的所有活前缀，就可以用这些活前缀来指导分析过程的进行。

用一个确定的有穷自动机描述出一个文法的所有规范句型的活前缀，然后将该DFA转换成一张LR分析表，来指导实际的语法分析工作。



7.2 LR(0) 分析表的构造

2. 构造识别规范句型活前缀的DFA

◆ 拓广文法

引入一个新的识别符号 S' 和产生式 $S' \rightarrow S$ 。

◆ LR(0)项目集

$U \rightarrow x.$ —— 归约项目；

$S' \rightarrow S.$ —— 接受项目；

$U \rightarrow x. ay$ (x, y 为符号串, $a \in V_T$) —— 移进项目；

$U \rightarrow x. Vy$ ($V \in V_N$) —— 待约项目。

识别规范句型活前缀的DFA的构造思想

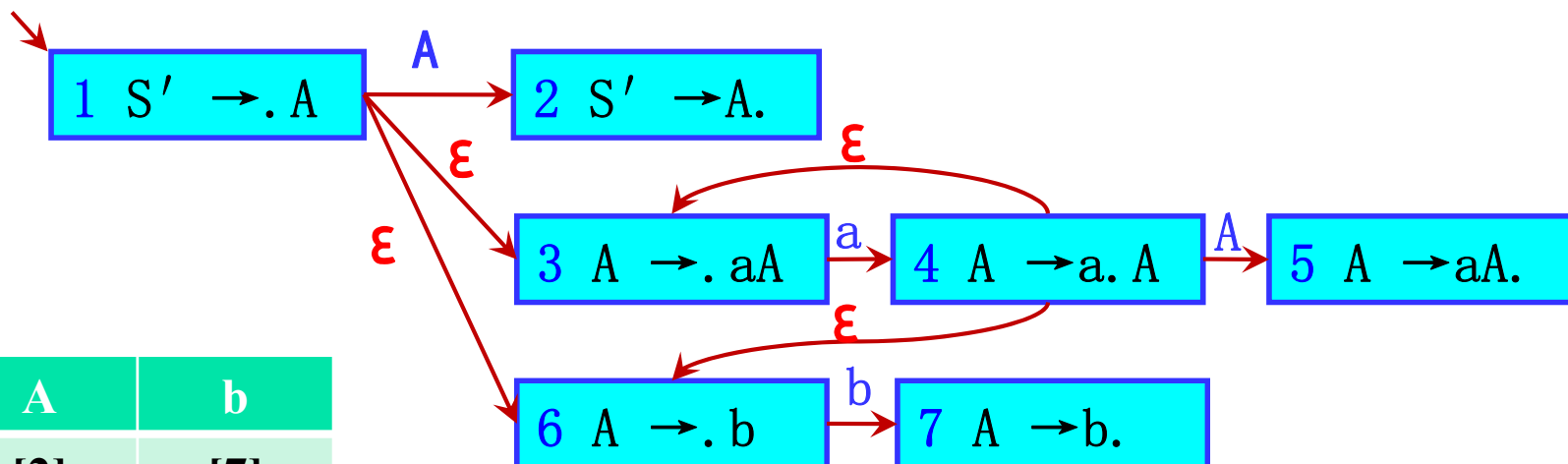
$G[A]$:

[0] $S' \rightarrow A$

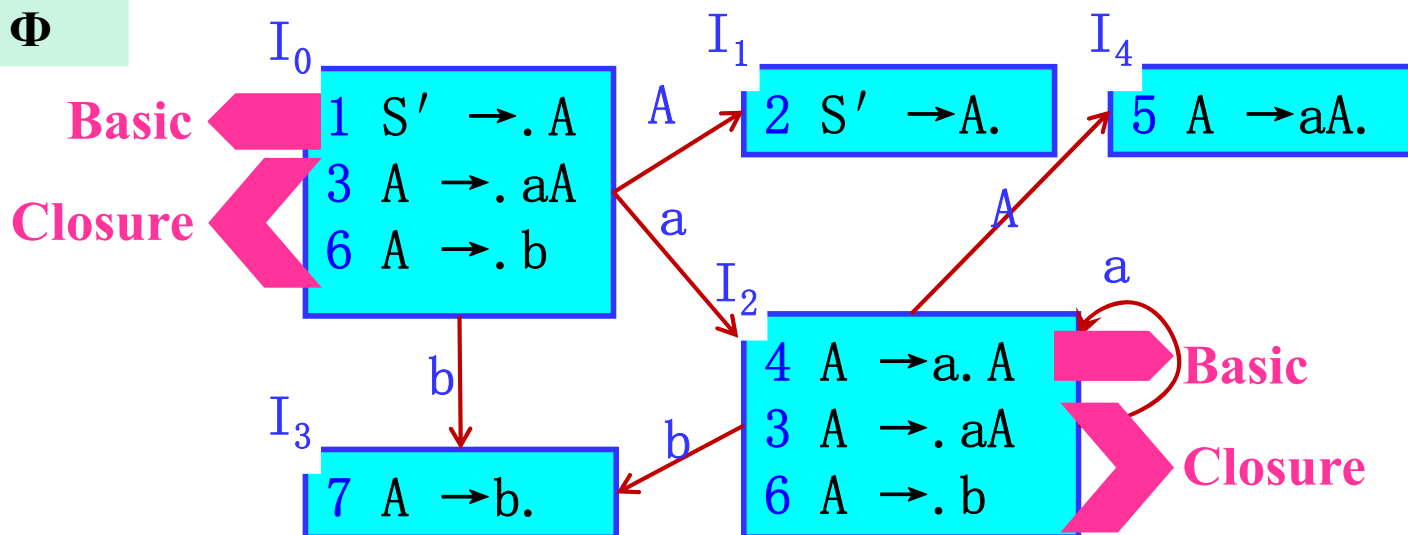
[1] $A \rightarrow aA$

[2] $A \rightarrow b$

	a	A	b
[136]	[436]	[2]	[7]
[436]	[436]	[5]	[7]
[2]	Φ	Φ	Φ
[7]	Φ	Φ	Φ
[5]	Φ	Φ	Φ



确定化





7.2 LR(0) 分析表的构造

2. 构造识别规范句型活前缀的DFA

◆ 构造一个状态

假设 I 为文法的任一项目集(开始时仅包含 $S' \rightarrow \cdot x$, S' 为拓广文法的识别符号), 重复下述步骤求 $CLOSURE(I)$:

- i. I 的任何项目均属于 $CLOSURE(I)$
- ii. 如果 $A \rightarrow x \cdot B y$ 属于 $CLOSURE(I)$, 则所有 $B \rightarrow \cdot z$ 也属于 $CLOSURE(I)$ 。

上述工作重复到 $CLOSURE(I)$ 不再扩大为止, 此 $CLOSURE(I)$ 即为所求的一个项目子集, 将其作为DFA的一个状态。



7.2 LR(0) 分析表的构造

2. 构造识别规范句型活前缀的DFA

◆ 由DFA的一个状态求其它状态，即构造DFA的映射

i. 若状态 I_i 中含有项目 $[A \rightarrow x.Xy]$ ($X \in V$, x, y 为任一符号串), 则从 I_i 出发, 以 X 为标记作一弧, 引出一新状态 I_j

$$I_j = \text{CLOSURE}(\{[A \rightarrow xX.y] \mid [A \rightarrow x.Xy] \text{ 在 } I_i \text{ 中}\})$$

I_i 和 I_j 之间的这种转换关系用函数 $\text{GOTO}(I_i, X) = I_j$ 表示。

ii. 若 I_i 和 I_j 相同, 则取消 I_j , 而 GOTO 函数为 $\text{GOTO}(I_i, X) = I_i$ 。



7.2 LR(0) 分析表的构造

2. 构造识别规范句型活前缀的DFA

◆ 拓广文法 G' 的项目集规范族 C

$$C = \cup I_i$$

$$I_0 = \text{CLOSURE}(\{[S' \rightarrow \bullet S]\})$$

C 即从初态 I_0 出发，可达的所有状态的集合。

构造算法：

从 I_0 出发，不断构造后继状态，直至不再产生新状态为止，所产生的全部状态的集合即为 C 。



7.2 LR(0) 分析表的构造

2. 构造识别规范句型活前缀的DFA——举例

G[S]:

[0] $S \rightarrow E$

[1] $E \rightarrow aA$

[2] $E \rightarrow bB$

[3] $A \rightarrow cA$

[4] $A \rightarrow d$

[5] $B \rightarrow cB$

[6] $B \rightarrow d$

$G[S]$:

[0] $S \rightarrow E$

[1] $E \rightarrow aA$

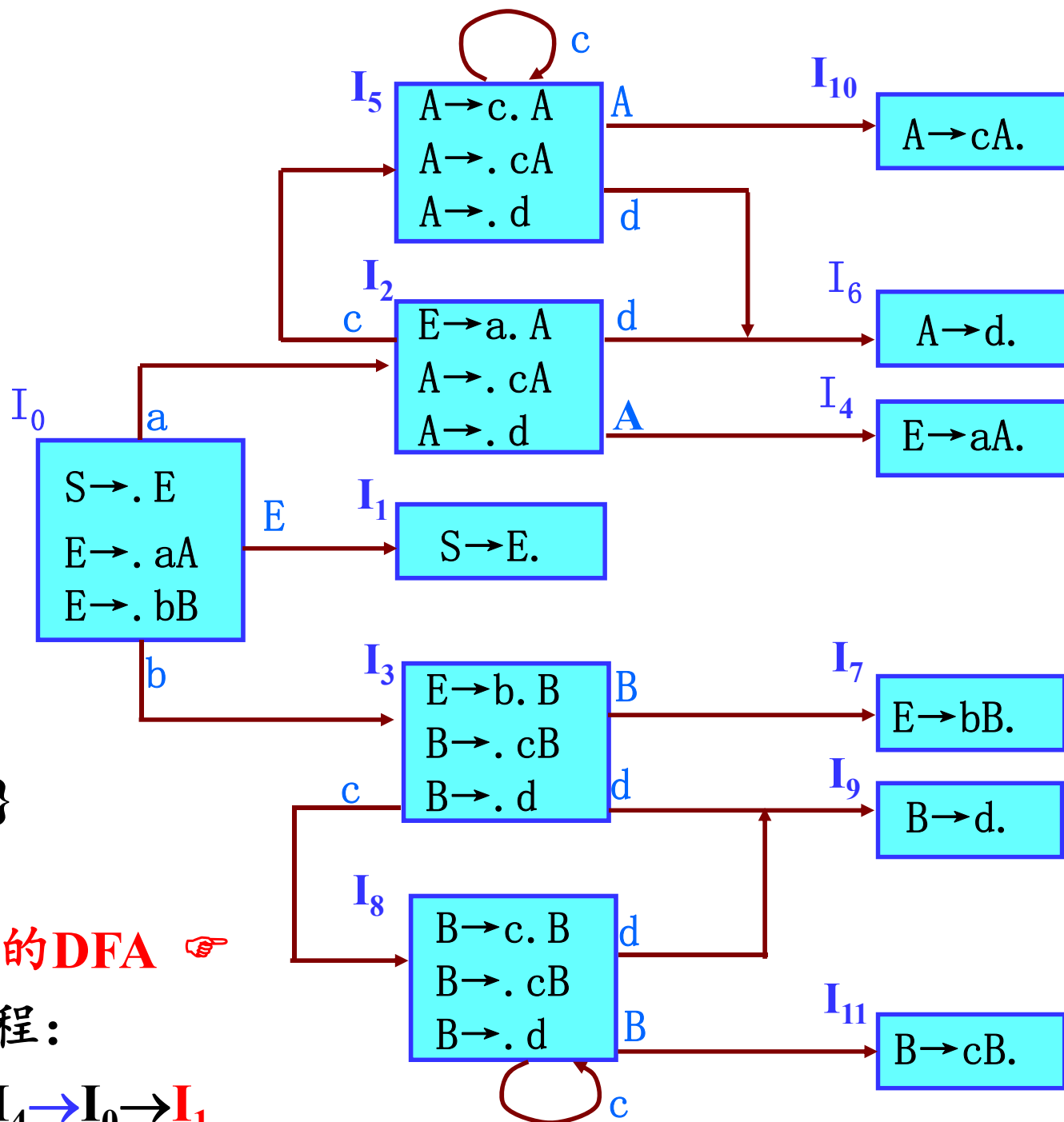
[2] $E \rightarrow bB$

[3] $A \rightarrow cA$

[4] $A \rightarrow d$

[5] $B \rightarrow cB$

[6] $B \rightarrow d$



LR(0)项目集规范族:

$C = \{ I_0, I_1, \dots, I_{11} \}$

识别 $G[S]$ 的全部活前缀的DFA

例: 'ad'的LR分析过程:

$I_0 \rightarrow I_2 \rightarrow I_6 \rightarrow I_2 \rightarrow I_4 \rightarrow I_0 \rightarrow I_1$

项目 $[A \rightarrow x.y]$ 称为对某活前缀 ux 是有效的 (valid)

iff 存在规范推导过程:

$$S \xRightarrow{*} uAv \Rightarrow uxyv$$

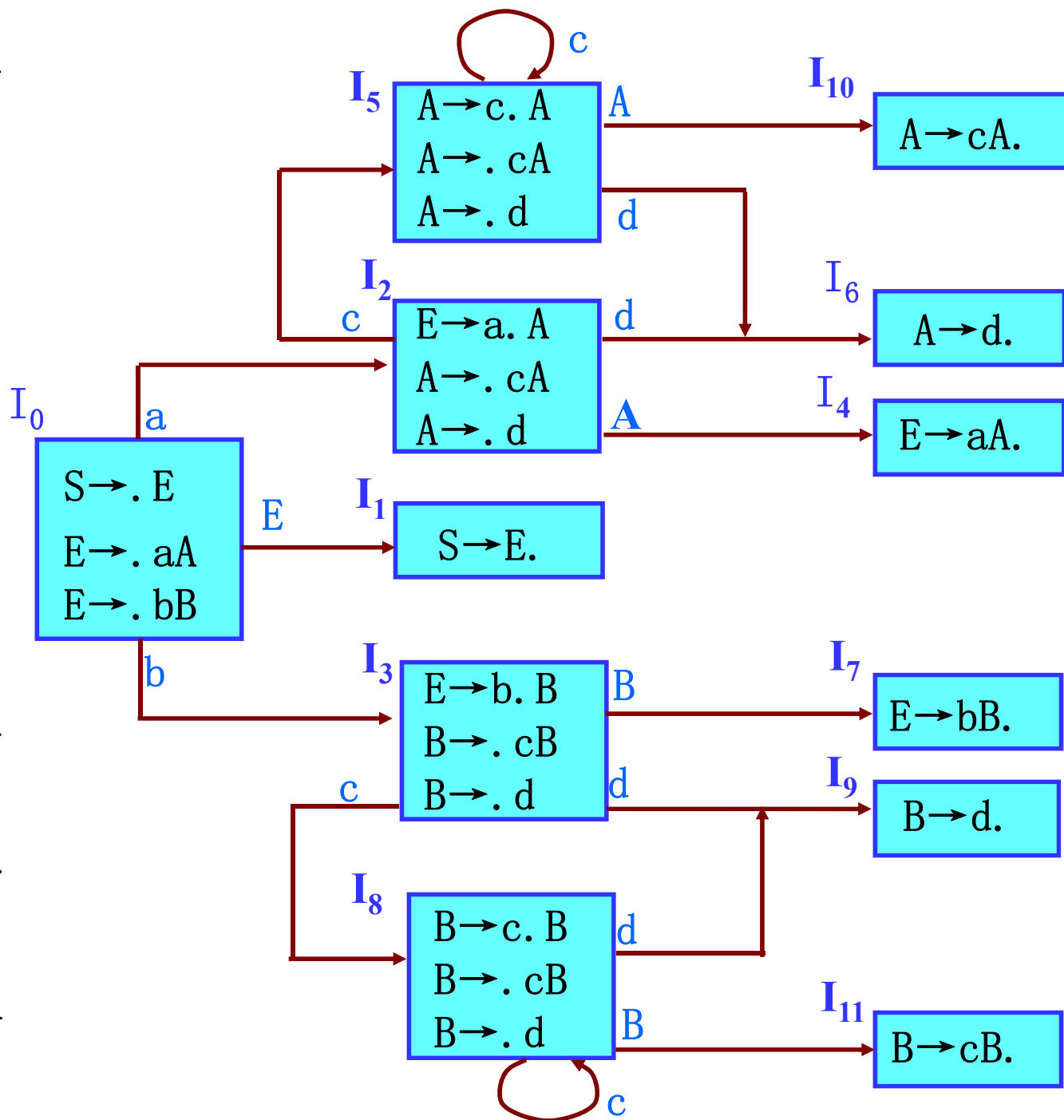
其中, $v \in V_T^*$

xy 是规范句型 $uxyv$ 的句柄。

一般地,

同一LR项目可能对多个活前缀是有效的 (如 I_9 中的 $[B \rightarrow d.]$ 项目对活前缀 bd 和 bcd 都是有效的) ;

也可能同一活前缀存在多个有效项目。





7.2 LR(0) 分析表的构造

3. LR(0) 文法

如果一个项目集(一个状态)中既有移进项目又含有归约项目，或着一个项目集中含有两个以上的不同归约项目，则称此项目为冲突项目。

一个文法，如果由它构造的识别规范句型活前缀的DFA中，每个项目集(状态)均不含冲突项目，则称这种文法G为LR(0)文法。



7.2 LR(0) 分析表的构造

4. 构造LR(0)分析表

① 若 $U \rightarrow x \cdot a y \in I_i$, 且 $GOTO(I_i, a) = I_j$, $a \in V_T$,

则 $ACTION[i, a] = "S_j"$ 。

② 若 $S' \rightarrow S \cdot \in I_i$, 则置 $ACTION[i, \#] = "acc"$ 。

③ 若 $U \rightarrow x \cdot \in I_i$, 则对任意终结符 a 和 $\#$,

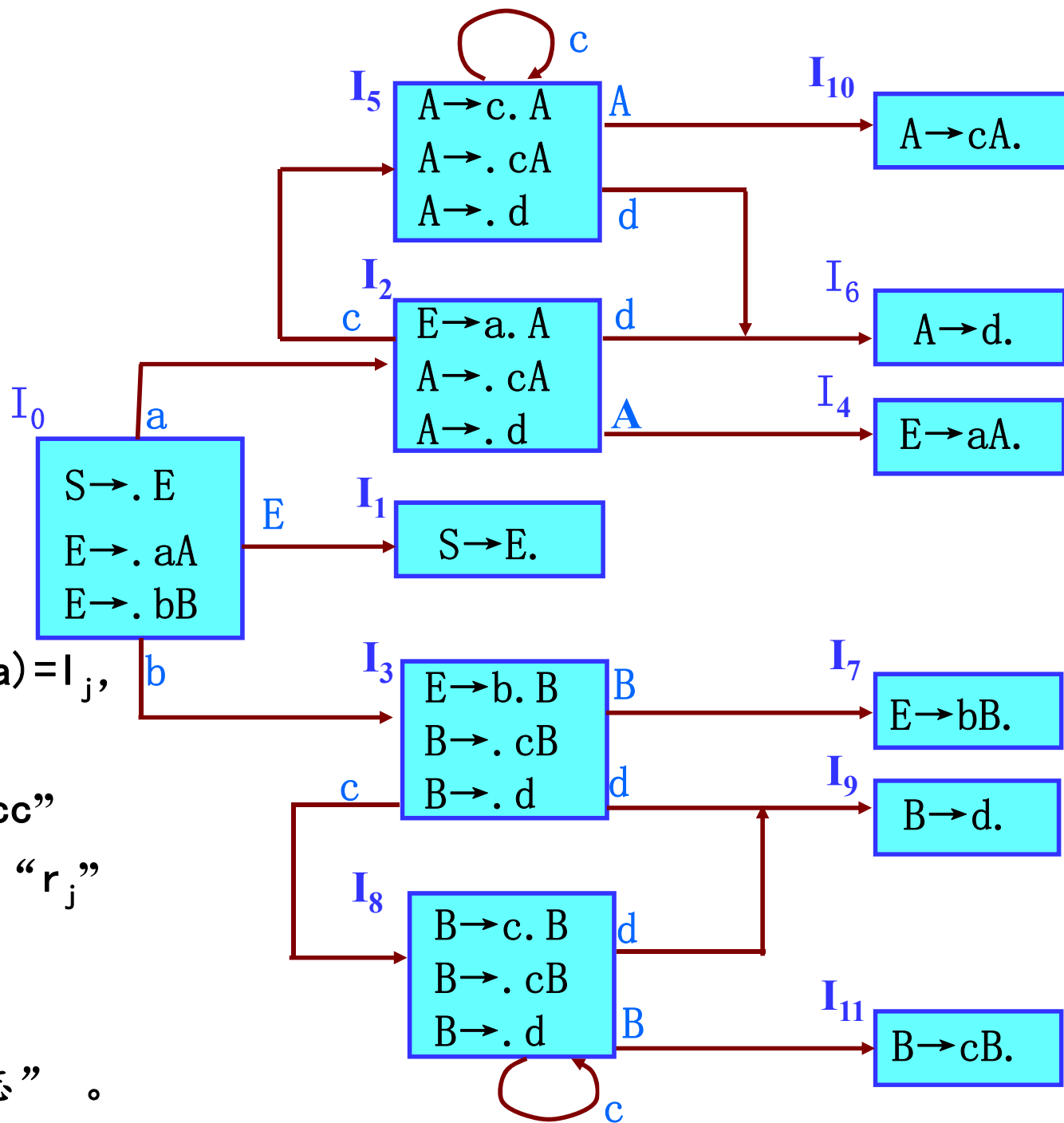
均置 $ACTION[i, a] = "r_j"$ 或 $ACTION[i, \#] = "r_j"$ 。

④ 若 $U \rightarrow x \cdot V y \in I_i$, 且 $GOTO(I_i, V) = I_j$, 其中 $V \in V_N$,

则置 $GOTO[i, V] = "j"$ 。

⑤ 除上述方法得到的分析表元素外, 其余元素均置“报错标志”。

$G[S]$: [0] $S \rightarrow E$
 [1] $E \rightarrow aA$
 [2] $E \rightarrow bB$
 [3] $A \rightarrow cA$
 [4] $A \rightarrow d$
 [5] $B \rightarrow cB$
 [6] $B \rightarrow d$



- ① $U \rightarrow x \cdot ay \in I_i$, $GOTO(I_i, a) = I_j$,
 $ACTION[i, a] = "S_j"$.
- ② $S' \rightarrow S \cdot$, $ACTION[i, \#] = "acc"$
- ③ $U \rightarrow x \cdot$, $ACTION[i, a/\#] = "r_j"$
- ④ $GOTO(I_i, V) = I_j$,
 $GOTO[i, V] = "j"$.
- ⑤ 其余元素均置“报错标志”。



7.2 LR(0) 分析表的构造

4. 构造LR(0)分析表——举例

$G[S]:$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow a$

$G[S']:$

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow a$



7.2 LR(0) 分析表的构造

4. 构造LR(0)分析表——举例

$G[S]:$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \epsilon$

$G[S']:$

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

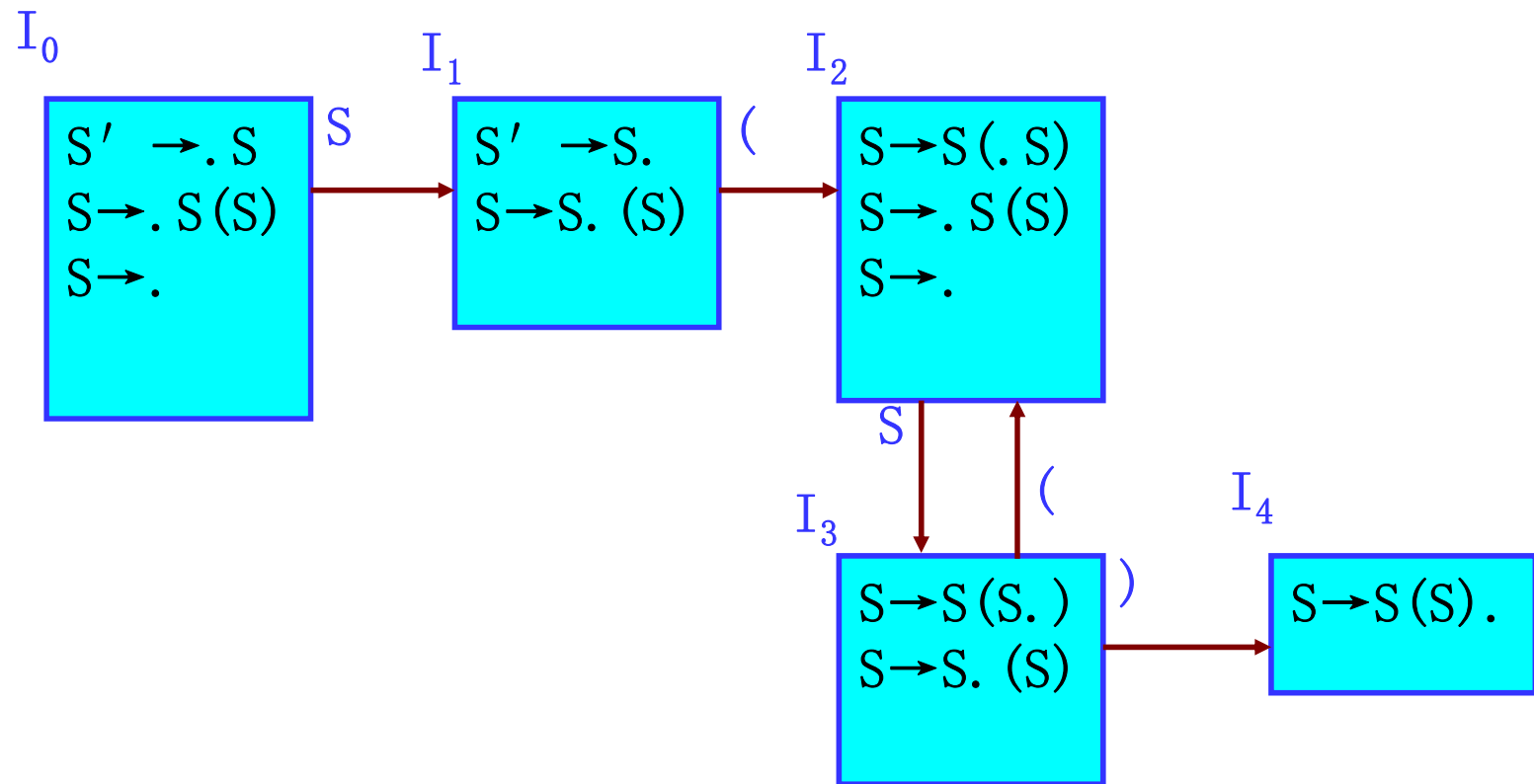
[2] $S \rightarrow \epsilon$

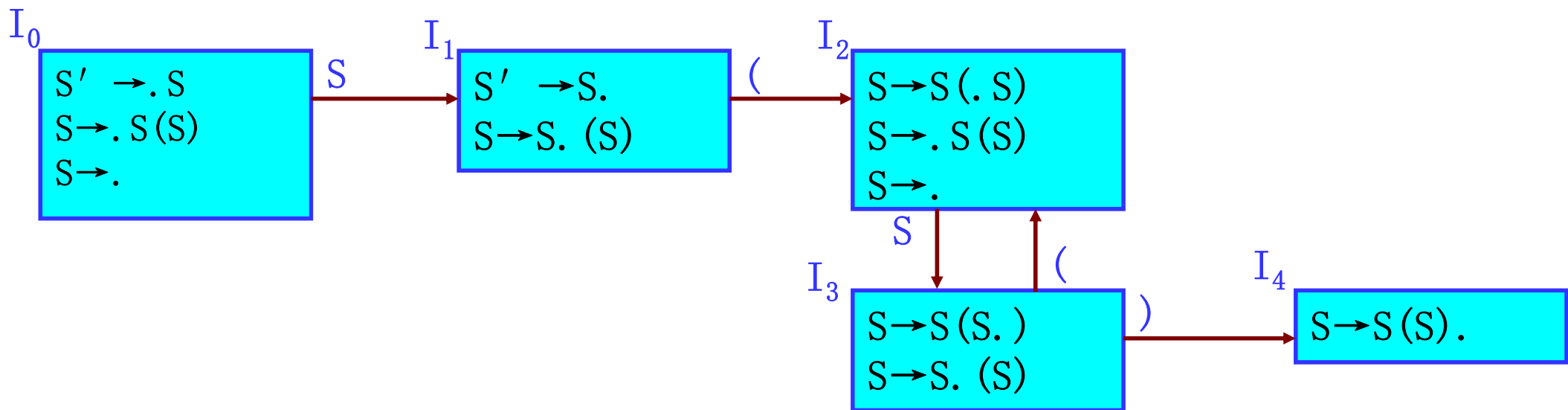
$G[S] :$

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \epsilon$





状态	ACTION			GOTO
	()	#	S
0	r_2	r_2	r_2	1
1	S_2		acc	
2	r_2	r_2	r_2	3
3	S_2	S_4		
4	r_1	r_1	r_1	

$G[S']$:

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \epsilon$



7.2 LR(0) 分析表的构造

思考：LR(0)分析法如何解决Shift-Reduce Parsing面临的两个问题？

例如，对文法G[S]：

$S \rightarrow aAcBe$

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$

分析符号串abbcde。

问题1. 如何确定句柄？

问题2. 如何确定归约的产生式？



7.3 SLR(1) 分析表的构造

1. LR(0)方法的不足

LR(0)分析表的构造方法实际上隐含了这样一个要求：构造出的识别规范句型活前缀的有穷自动机中，每个状态均不能有冲突项目，这显然对文法的要求太高，也限制了该方法的应用。



7.3 SLR(1) 分析表的构造

1. LR(0)方法的不足

例如，对文法 $G[E']$:

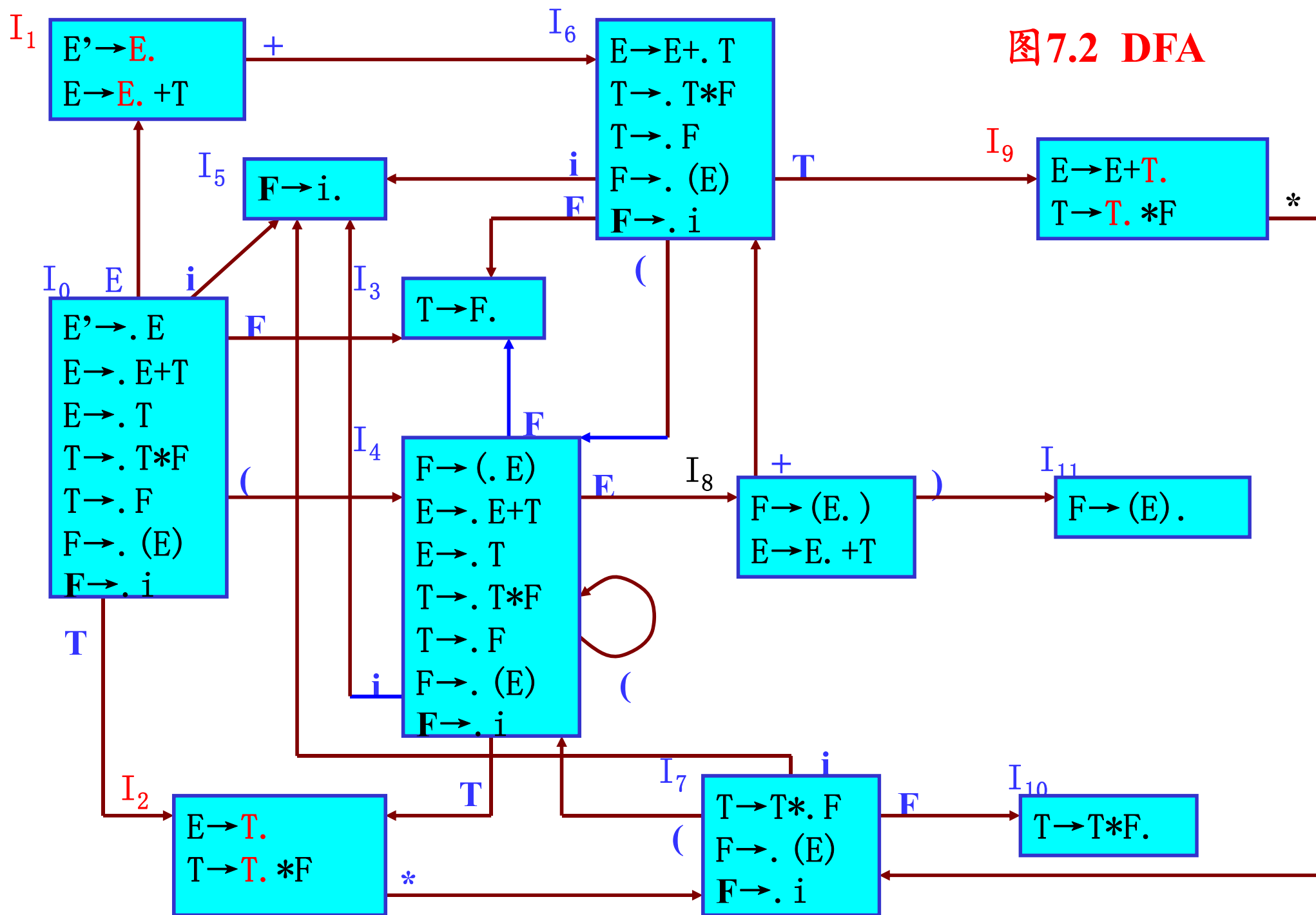
- 0 $E' \rightarrow E$
- 1 $E \rightarrow E + T$
- 2 $E \rightarrow T$
- 3 $T \rightarrow T * F$
- 4 $T \rightarrow F$
- 5 $F \rightarrow (E)$
- 6 $F \rightarrow i$

文法(7.1)' 不是LR(0)文法

构造LR(0)分析表的步骤:

1. G'
2. I_0
3. C (CLOSURE, GOTO)
4. DFA
5. 分析表 (ACTION, GOTO)

图7.2 DFA



G[E']:

$$\mathbf{0} \quad \mathbf{E}' \rightarrow \mathbf{E}$$

1 E→E+T

2 E→T

3 $T \rightarrow T * F$

4 T→F

5 $\mathbf{F} \rightarrow (\mathbf{E})$

6 $\mathbf{F} \rightarrow \mathbf{i}$

[illegible]



7.3 SLR(1) 分析表的构造

2. LR(0)方法的问题

分析状态 I_2 :

$\{ [E \rightarrow T.] , [T \rightarrow T.*F] \}$

该状态含有移进-归约冲突，在分析表中表现为多重定义。



7.3 SLR(1) 分析表的构造

2. LR(0)方法的问题

① 若 $U \rightarrow x.ay \in I_i$, 且 $GOTO(I_i, a) = I_j$, $a \in V_T$,

则 $ACTION[i, a] = "S_j"$ 。

② 若 $S' \rightarrow S. \in I_i$, 则置 $ACTION[i, \#] = "acc"$ 。

③ 若 $U \rightarrow x. \in I_i$, 则对任意终结符 a 和 $\#$,

均置 $ACTION[i, a] = "r_j"$ 或 $ACTION[i, \#] = "r_j"$ 。

④ 若 $U \rightarrow x.Vy \in I_i$, 且 $GOTO(I_i, V) = I_j$, 其中 $V \in V_N$,

则置 $GOTO[i, V] = "j"$ 。

⑤ 除上述方法得到的分析表元素外, 其余元素均置“报错标志”。



7.3 SLR(1) 分析表的构造

3. 解决问题的方法

所以在状态 I_2 :

$\{ E \rightarrow T., T \rightarrow T.*F \}$

中，只需要向前看下一输入符号是 $*$ ，还是 $FOLLOW(E)$ 中的符号，便可解决冲突了。

7.3 SLR(1) 分析表的构造

3. 解决问题的方法

假设一个状态 I_i 中含有冲突项目

$$I_i = \{ [U_1 \rightarrow x.b_1 y_1], [U_2 \rightarrow x.b_2 y_2], \dots, [U_m \rightarrow x.b_m y_m], \\ \dots, [V_1 \rightarrow x.], [V_2 \rightarrow x.], \dots, [V_n \rightarrow x.] \}$$

若 $\{b_1, b_2, \dots, b_m\}$ 和 $\text{FOLLOW}(V_1)$ 、 $\text{FOLLOW}(V_2)$ 、 \dots 、 $\text{FOLLOW}(V_n)$ 两两互不相交，就可以采用向前看一个输入符号的方法来解决状态中的冲突。当向前看到的输入符号属于集合 $\{b_1, b_2, \dots, b_m\}$ 时，执行“移进”动作；当向前看到的输入符号属于 $\text{FOLLOW}(V_i)$ 时，则按归约项目 $V_i \rightarrow x.$ 相应的产生式进行“归约”。

7.3 SLR(1) 分析表的构造

4. SLR(1) 分析表构造算法

③ 若 $U \rightarrow x. \in I_i$, 则对任意的终结符 a 和 $\#$, 均置

$ACTION[i, a] = "r_j"$ 或 $ACTION[i, \#] = "r_j"$ 。



③ 若 $U \rightarrow x. \in I_i$, 则对 $FOLLOW(U)$ 中的终结符 a 或 $\#$, 均置

$ACTION[i, a] = "r_j"$ $a \in FOLLOW(U)$

或 $ACTION[i, \#] = "r_j"$ $\# \in FOLLOW(U)$

G[E'] :

0 E' → E

1 E → E + T

2 E → T

3 T → T * F

4 T → F

5 F → (E)

6 F → i

状态	ACTION						GOTO		
	i	+	*	()	#	E	T	F
0	S ₅			S ₄			1	2	3
1		S ₆				acc			
2		r ₂	S ₇		r ₂	r ₂			
3		r ₄	r ₄		r ₄	r ₄			
4	S ₅			S ₄			8	2	3
5		r ₆	r ₆		r ₆	r ₆			
6	S ₅			S ₄				9	3
7	S ₅			S ₄					10
8		S ₆			S ₁₁				
9		r ₁	S ₇		r ₁	r ₁			
10		r ₃	r ₃		r ₃	r ₃			
11		r ₅	r ₅		r ₅	r ₅			



7.4 LR(1) 分析表的构造

1. SLR(1)方法的不足

SLR(1) 方法要求：当一个状态中出现冲突项目时，若向前搜索符号集 $\{b_1, b_2, \dots, b_m\}$ 和 $\text{FOLLOW}(V_1)$ 、 $\text{FOLLOW}(V_2)$ 、...、 $\text{FOLLOW}(V_n)$ 两两互不相交，则可以通过向前读一个输入符号，看这个输入符号属于哪个集合来解决分析表中的冲突定义。

若文法不满足这一条件呢？ --- 有相交怎么办？

7.4 LR(1) 分析表的构造

2. 对SLR(1)分析表构造算法的分析

③ 若 $U \rightarrow x. \in I_i$, 则对任意终结符 a 和 $\#$, 均置

$ACTION[i, a] = "r_j"$ 或 $ACTION[i, \#] = "r_j"$ 。



③ 若 $U \rightarrow x. \in I_i$, 则对 **FOLLOW(U)** 中的终结符 a 和 $\#$, 均置

$ACTION[i, a] = "r_j"$ 或 $ACTION[i, \#] = "r_j"$ 。

例: 文法(7.2)

G[Z]:

[0] $Z \rightarrow S$

[1] $S \rightarrow L=R$

[2] $S \rightarrow R$

[3] $R \rightarrow L$

[4] $L \rightarrow *R$

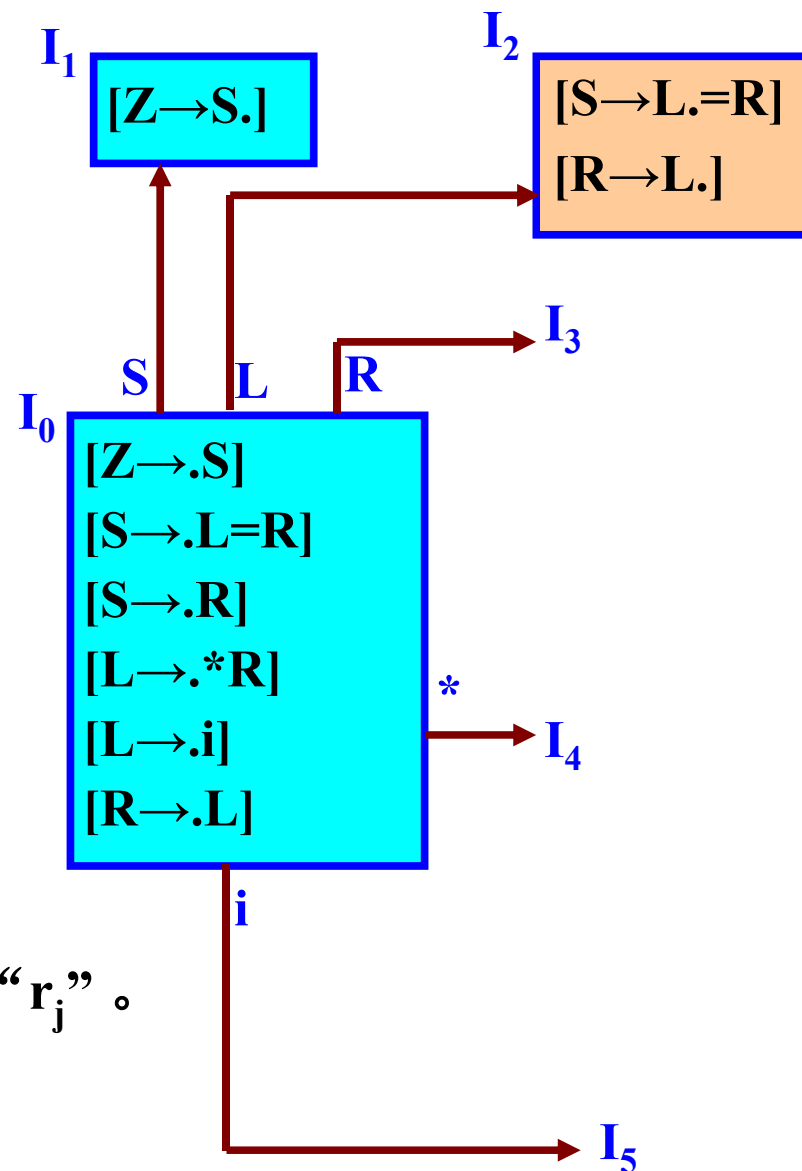
[5] $L \rightarrow i$

③ 若 $U \rightarrow x. \in I_i$,

则对 **FOLLOW(U)** 中的终结符 **a** 和 **#**, 均置

$\text{ACTION}[i, a] = "r_j"$ 或 $\text{ACTION}[i, \#] = "r_j"$ 。

FOLLOW(R) = {#, =}



7.4 LR(1) 分析表的构造

2. 对SLR(1)分析表构造算法的分析

例：文法(7.2) 不是SLR(1)文法

I_2

$[S \rightarrow L.=R]$

$[R \rightarrow L.]$

$I_2 = \{ [S \rightarrow L.=R], [R \rightarrow L.] \}$

考虑项目 $[S \rightarrow L.=R]$: $\text{ACTION}(2, =) = S_6$

考虑项目 $[R \rightarrow L.]$: 因为 $\text{FOLLOW}(R) = \{ \#, = \}$

所以 $\text{ACTION}(2, =) = r_3$

用 $R \rightarrow L$ 归约

分析：此时当输入符号是 ‘=’ 时，不能用 $R \rightarrow L$ 归约，只能移进。

当遇到 ‘ $L=...$ ’ 时，用 ‘ $R \rightarrow L$ ’ 将 L 归约为 R ，得到的符号串

‘ $R=...$ ’ 不是句型（不合适的归约）！故当遇到 ‘ $L=...$ ’ 时不能将 L 做句柄。

$G[Z]$:

[0] $Z \rightarrow S$

[1] $S \rightarrow L=R$

[2] $S \rightarrow R$

[3] $R \rightarrow L$

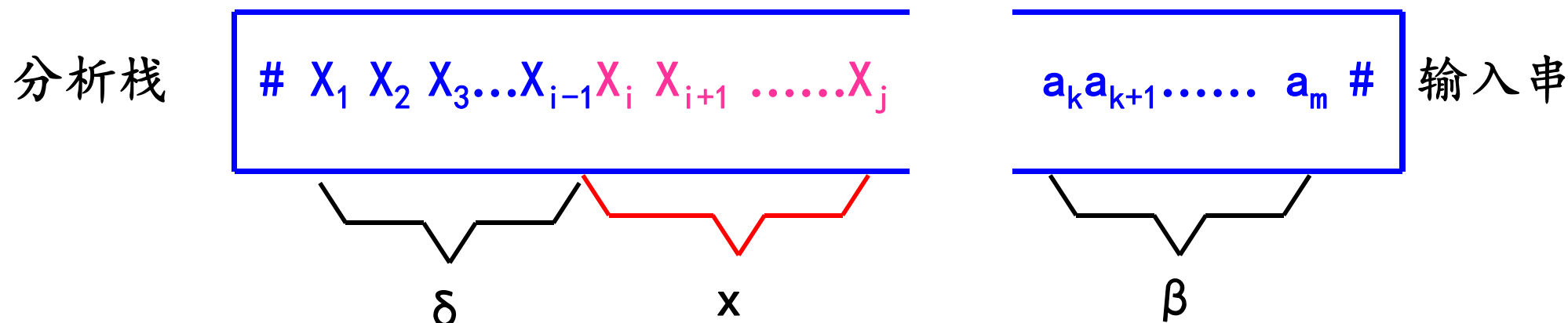
[4] $L \rightarrow *R$

[5] $L \rightarrow i$

7.4 LR(1) 分析表的构造

3. 解决问题的方法

句柄一定和某条产生式的右部相同，但并非和产生式右部相同的符号串都是句柄。





7.4 LR(1) 分析表的构造

3. 解决问题的方法

句柄一定和某条产生式的右部相同，但并非和产生式右部相同的符号串都是句柄。

分析文法 $G[S]$ 中所有以 T 作为句柄的句型：

当 T 作为一个句型的句柄时， T 后的符号一定在 $FOLLOW(E)$ 中，但并不是 $FOLLOW(E)$ 中的每一个符号都可以出现在每一个句柄 T 后。

• 如何记住每一个句柄的上下文？ ---- 重新定义项目：LR(1)项目

使得LR的每个状态都能指明：当哪些输入符号紧跟在句柄 α 之后时，才允许选用产生式 $A \rightarrow \alpha$ 将 α 归约到 A 。



7.4 LR(1) 分析表的构造

4. LR(1) 项目

$[U \rightarrow v.w, a]$

含义：已经从输入符号串中获得了句柄vw中的v，还希望从输入符号串中进一步获得w，并且当在分析栈顶获得完整的vw后（即进入含归约项目 $[U \rightarrow vw., a]$ 的状态），只有下一个输入符号是a时，才可将vw归约成U，否则不能归约。

向前看符a只对归约项目 $[U \rightarrow vw., a]$ 有意义。

在其它项目中，a只是起一个传递作用。



7.4 LR(1) 分析表的构造

5. 构造识别规范句型活前缀的DFA

◆ 构造一个状态 I_0

假设 I 为文法的任一项目集(开始时仅包含 $[S' \rightarrow .S, \#]$, S' 为扩充文法的识别符号), 重复下述步骤求 $CLOSURE(I)$:

i. I 的任何项目均属于 $CLOSURE(I)$

ii. 如果 $[A \rightarrow \alpha.B \beta, a]$ 属于 $CLOSURE(I)$,

则所有 $[B \rightarrow \cdot\gamma, b]$ 也属于 $CLOSURE(I)$,

这里终结符 $b \in FIRST(\beta a)$ 。

上述工作重复到 $CLOSURE(I)$ 不再扩大为止, 此 $CLOSURE(I)$ 即为所求的一个项目子集, 将其作为DFA的一个状态。



7.4 LR(1) 分析表的构造

5. 构造识别规范句型活前缀的DFA

◆ 构造DFA的映射

i. 若状态 I_i 中含有项目 $[A \rightarrow \alpha \cdot X \beta, a]$ ($X \in V$, α, β 为任一符号串), 则从 I_i 出发, 以 X 为标记作一弧, 引出一新状态 I_j

$$I_j = \text{CLOSURE}(\{[A \rightarrow \alpha X \cdot \beta, a] \mid [A \rightarrow \alpha \cdot X \beta, a] \text{ 在 } I_i \text{ 中}\})$$

I_i 和 I_j 之间的这种转换关系用函数 $\text{GOTO}(I_i, X) = I_j$ 表示。

ii. 若 I_i 和 I_j 相同, 则取消 I_j , 而 GOTO 函数为 $\text{GOTO}(I_i, X) = I_i$ 。



7.4 LR(1) 分析表的构造

6. 构造识别规范句型活前缀的DFA——**举例 文法(7.2)**
不是SLR(1)文法

G[Z]:

[0] $Z \rightarrow S$

[1] $S \rightarrow L = R$

[2] $S \rightarrow R$

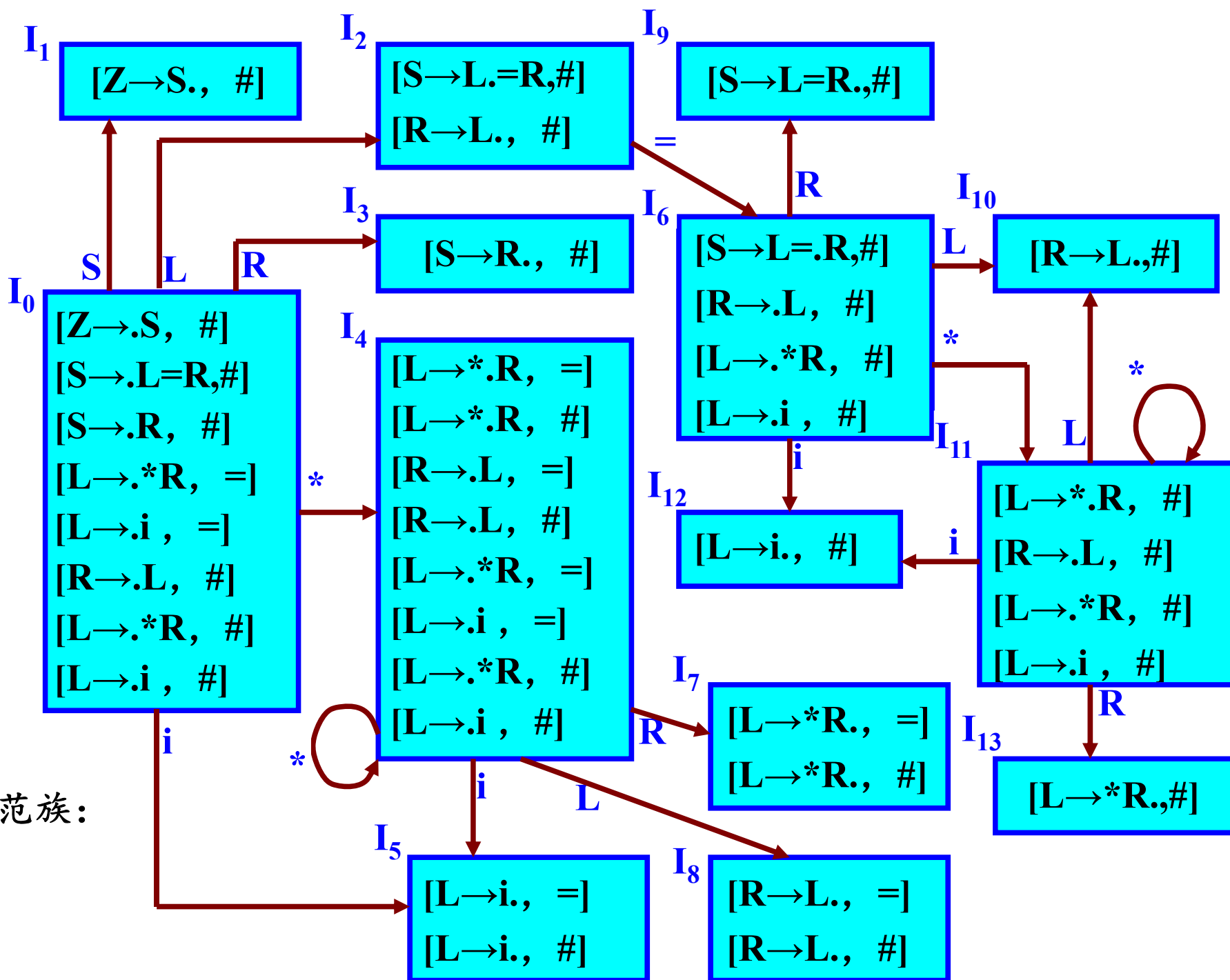
[3] $R \rightarrow L$

[4] $L \rightarrow *R$

[5] $L \rightarrow i$

$G[Z]$:

- [0] $Z \rightarrow S$
- [1] $S \rightarrow L=R$
- [2] $S \rightarrow R$
- [3] $R \rightarrow L$
- [4] $L \rightarrow *R$
- [5] $L \rightarrow i$



LR(1)项目集规范族:

$\{I_0, I_1, \dots, I_{13}\}$

$G[Z]$:

[0] $Z \rightarrow S$

[1] $S \rightarrow L=R$

[2] $S \rightarrow R$

[3] $R \rightarrow L$

[4] $L \rightarrow *R$

[5] $L \rightarrow i$

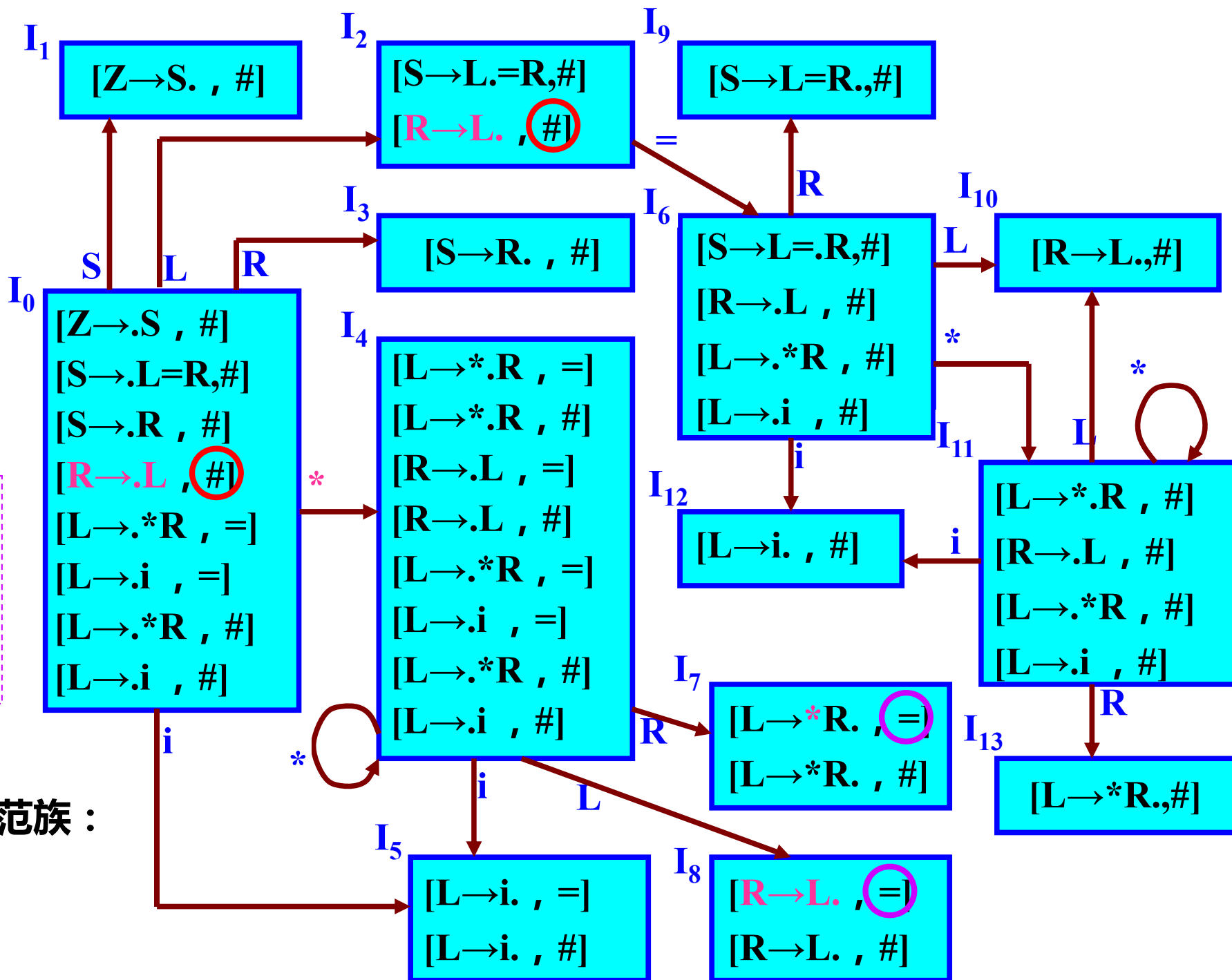
句型 " $*R=...$ "

非句型 " $R=...$ "

句型 " $R\#$ "

LR(1)项目集规范族 :

$\{I_0, I_1, \dots, I_{13}\}$





7.4 LR(1) 分析表的构造

7. 构造LR(1)分析表

- ① 若 $[U \rightarrow x \cdot a y, b] \in I_i$, $\text{GOTO}(I_i, a) = I_j$, $a \in V_T$, 则 $\text{ACTION}[i, a] = "S_j"$ 。
- ② 若 $[S' \rightarrow S \cdot, \#] \in I_i$, 则置 $\text{ACTION}[i, \#] = "acc"$ 。
- ③ 若 $[U \rightarrow x \cdot, a] \in I_i$, 则置 $\text{ACTION}[i, a] = "r_j"$ 。
- ④ 若 $\text{GOTO}(I_i, V) = I_j$, 其中 $V \in V_N$, 则置 $\text{GOTO}[i, V] = "j"$ 。
- ⑤ 除上述方法得到的分析表元素外, 其余元素均置“报错标志”。



7.4 LR(1) 分析表的构造

7. 构造LR(1)分析表——

举例 文法(7.2)

不是SLR(1)文法

G[Z]:

[0] $Z \rightarrow S$

[1] $S \rightarrow L = R$

[2] $S \rightarrow R$

[3] $R \rightarrow L$

[4] $L \rightarrow *R$

[5] $L \rightarrow i$

$G[Z]$:

[0] $Z \rightarrow S$

[1] $S \rightarrow L=R$

[2] $S \rightarrow R$

[3] $R \rightarrow L$

[4] $L \rightarrow *R$

[5] $L \rightarrow i$

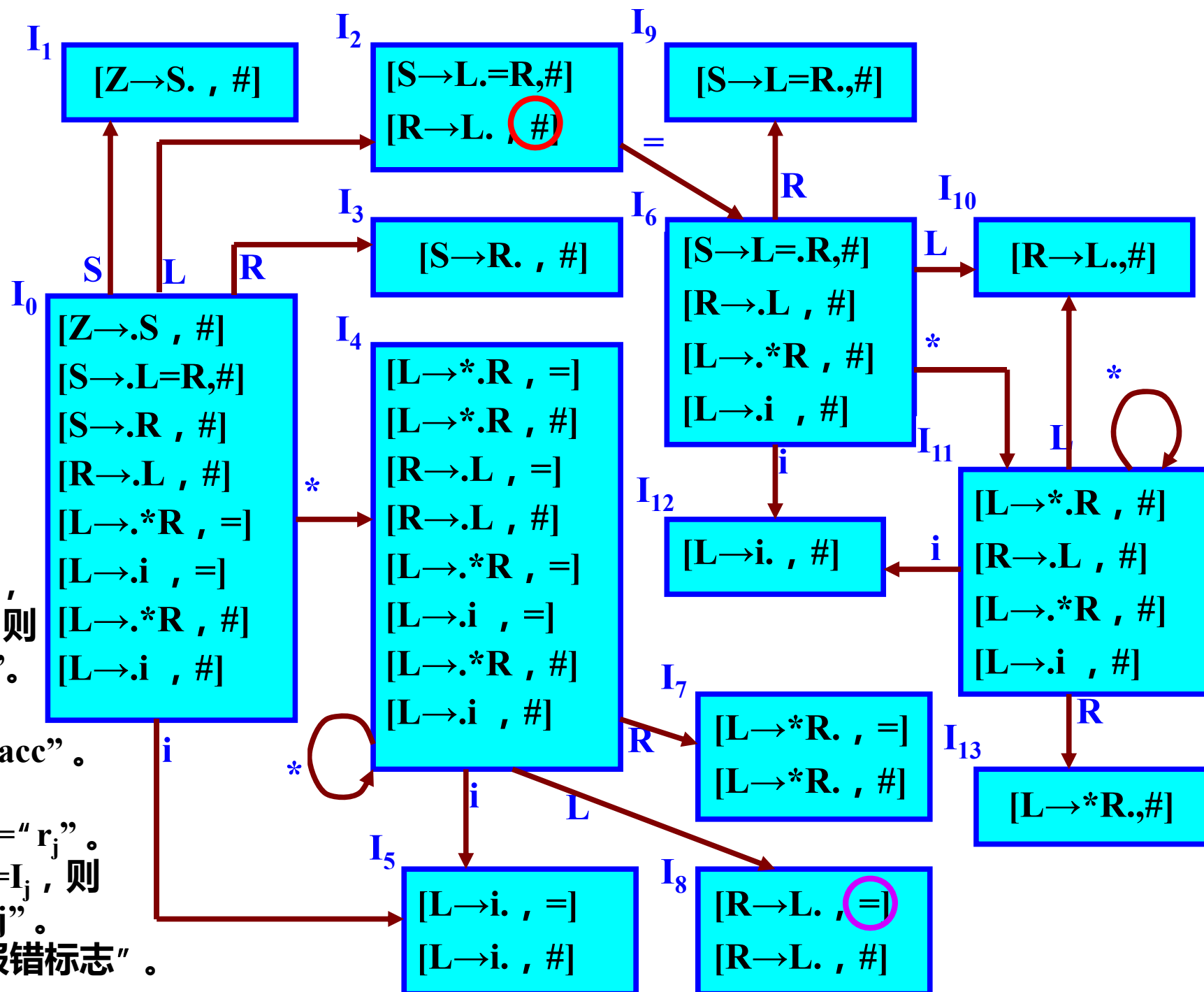
① 若 $[U \rightarrow x.ay, b] \in I_i$,
GOTO(I_i , a)= I_j , 则
ACTION[i, a]=" S_j ".

② 若 $[S' \rightarrow S., \#] \in I_i$,
则 ACTION[$i, \#$]="acc".

③ 若 $[U \rightarrow x., a] \in I_i$,
则置 ACTION[i, a]=" r_j ".

④ 若 GOTO(I_i , V)= I_j , 则
置 GOTO[i , V]=" j ".

⑤ 其余元素均置“报错标志”。



状态	ACTION				GOTO		
	i	*	=	#	S	L	R
0	S ₅	S ₄			1	2	3
1				acc			
2			S ₆	r ₃			
3				r ₂			
4	S ₅	S ₄				8	7
5			r ₅	r ₅			
6	S ₁₂	S ₁₁				10	9
7			r ₄	r ₄			
8			r ₃	r ₃			
9				r ₁			
10				r ₃			
11	S ₁₂	S ₁₁				10	13
12				r ₅			
13				r ₄			



7.4 LR(1) 分析表的构造

8. LR(1) 文法

如果按算法构造的分析表不含多重定义的表项（即无冲突动作出现），则称此分析表为**规范LR(1)分析表**。

利用规范LR(1)分析表的分析器称为**规范LR(1)分析器**。

能构造出规范LR(1)分析表的文法，称为**LR(1)文法**。



7.4 LR(1) 分析表的构造

8. LR(1)文法 ——

举例

文法(7.2)

不是SLR(1)文法,是LR(1)文法

G[Z]:

[0] $Z \rightarrow S$

[1] $S \rightarrow L=R$

[2] $S \rightarrow R$

[3] $R \rightarrow L$

[4] $L \rightarrow *R$

[5] $L \rightarrow i$

LR(1) 分析方法:

除了历史、现实, 还考虑
展望信息 —— 确实是规
范句型中跟在句柄之后的
终结符a才进行归约。



7.4 LR(1) 分析表的构造

9. LR(k)文法

—— 向前查看k个符号，确定使用哪个产生式进行归约

文法G[S]，对于规范句型的规范推导

$S \xRightarrow{*} uAv \Rightarrow uxyv$ 的每一推导步，

仅通过扫描 uxy 和至多向前查看 v 中的k个符号，
就能唯一确定选用产生式 $A \rightarrow xy$ 。

则此文法G为LR(k)文法。

其中， $v \in V_T^*$ ， xy 是规范句型 $uxyv$ 的句柄。



7.4 LR(1) 分析表的构造

非LR结构

例： $L = \{ ww^R \mid w \in \{a,b\}^* \}$

G[S]:

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \epsilon$



7.5 LALR(1) 分析表的构造

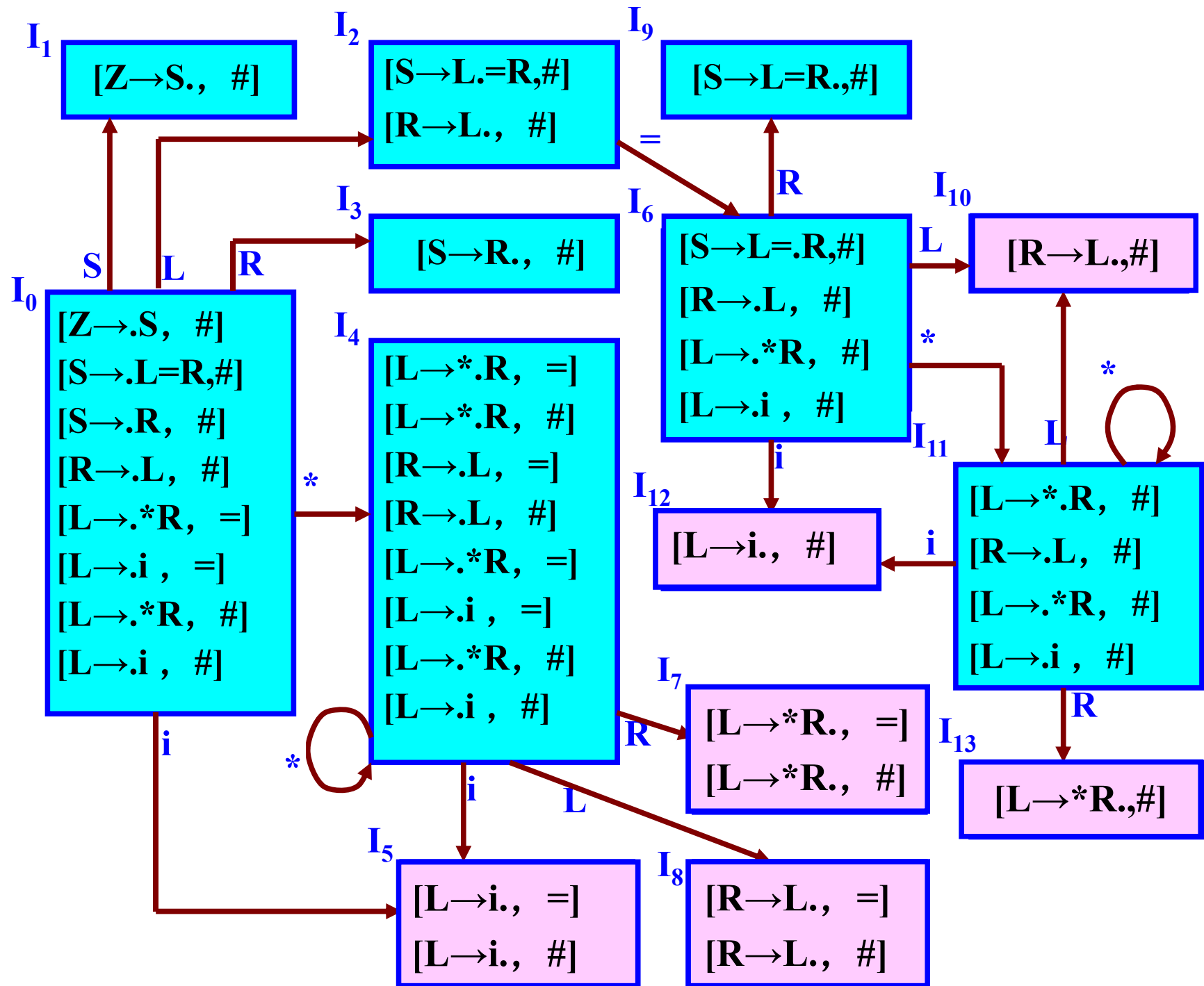
1. LR(1)方法的不足

—— 分析表太庞大。

如何解决存储空间的有效使用问题？

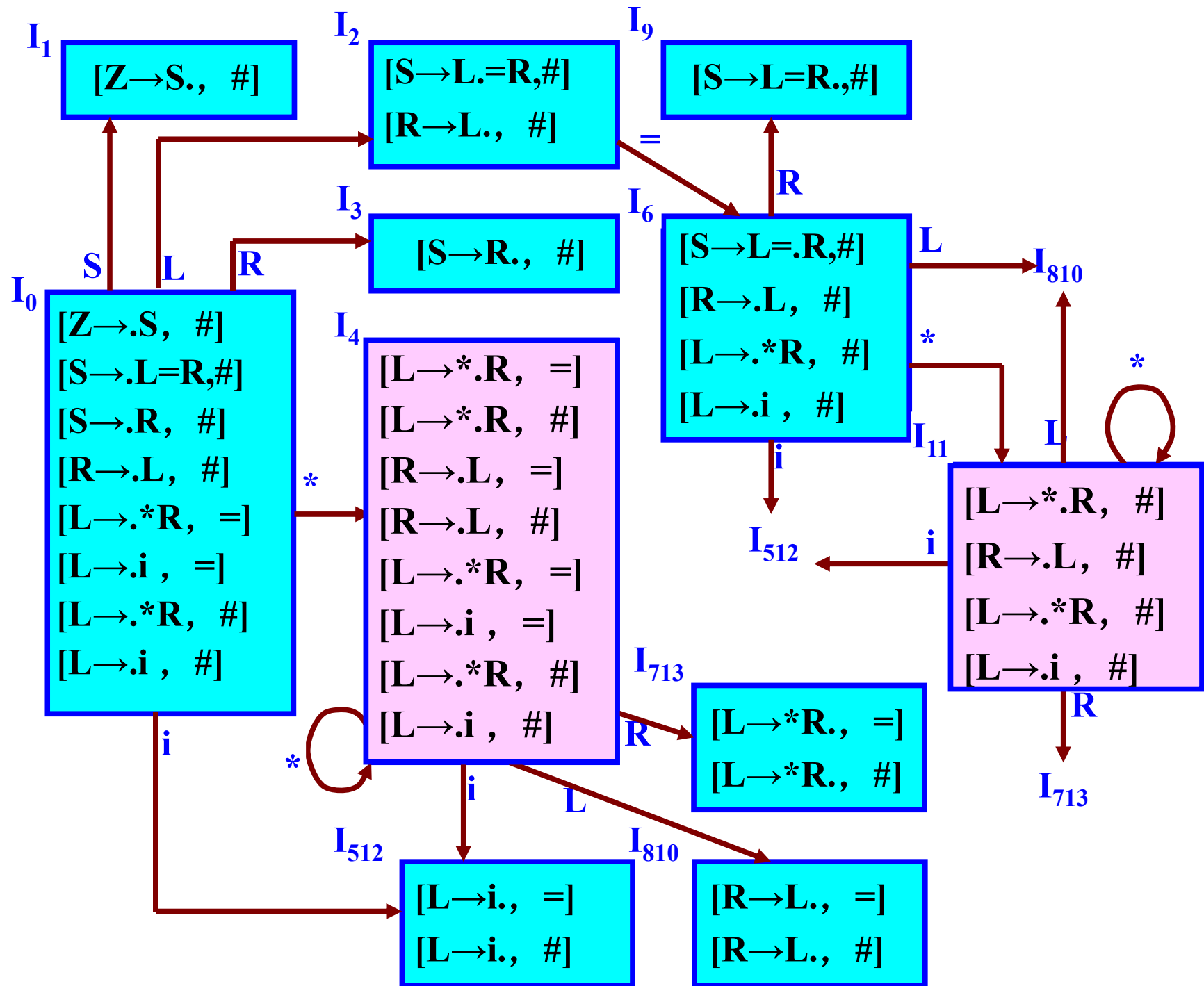
G[Z]:

- [0] $Z \rightarrow S$
- [1] $S \rightarrow L=R$
- [2] $S \rightarrow R$
- [3] $R \rightarrow L$
- [4] $L \rightarrow *R$
- [5] $L \rightarrow i$



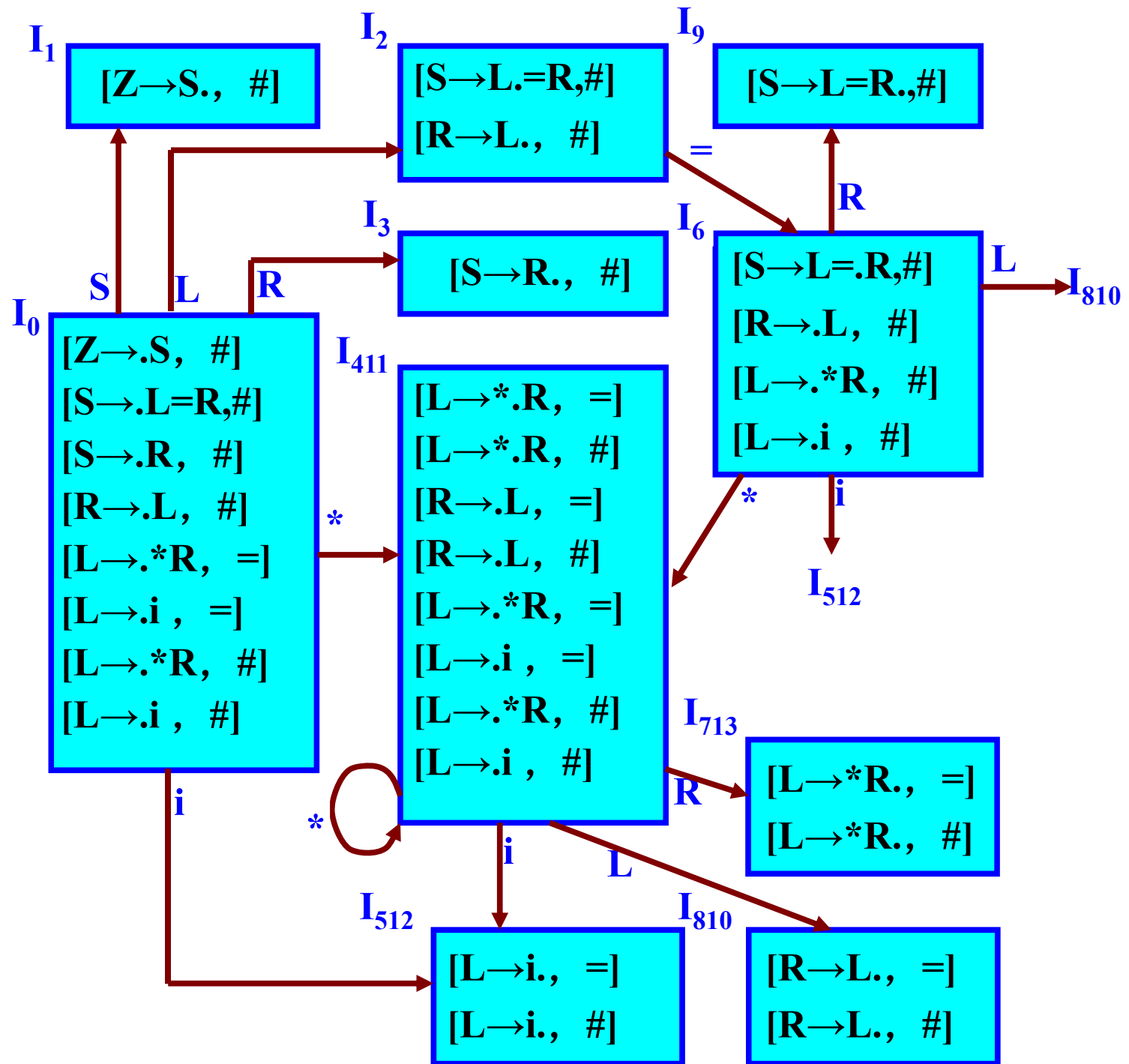
G[Z]:

- [0] $Z \rightarrow S$
- [1] $S \rightarrow L=R$
- [2] $S \rightarrow R$
- [3] $R \rightarrow L$
- [4] $L \rightarrow *R$
- [5] $L \rightarrow i$



G[Z]:

- [0] $Z \rightarrow S$
- [1] $S \rightarrow L=R$
- [2] $S \rightarrow R$
- [3] $R \rightarrow L$
- [4] $L \rightarrow *R$
- [5] $L \rightarrow i$



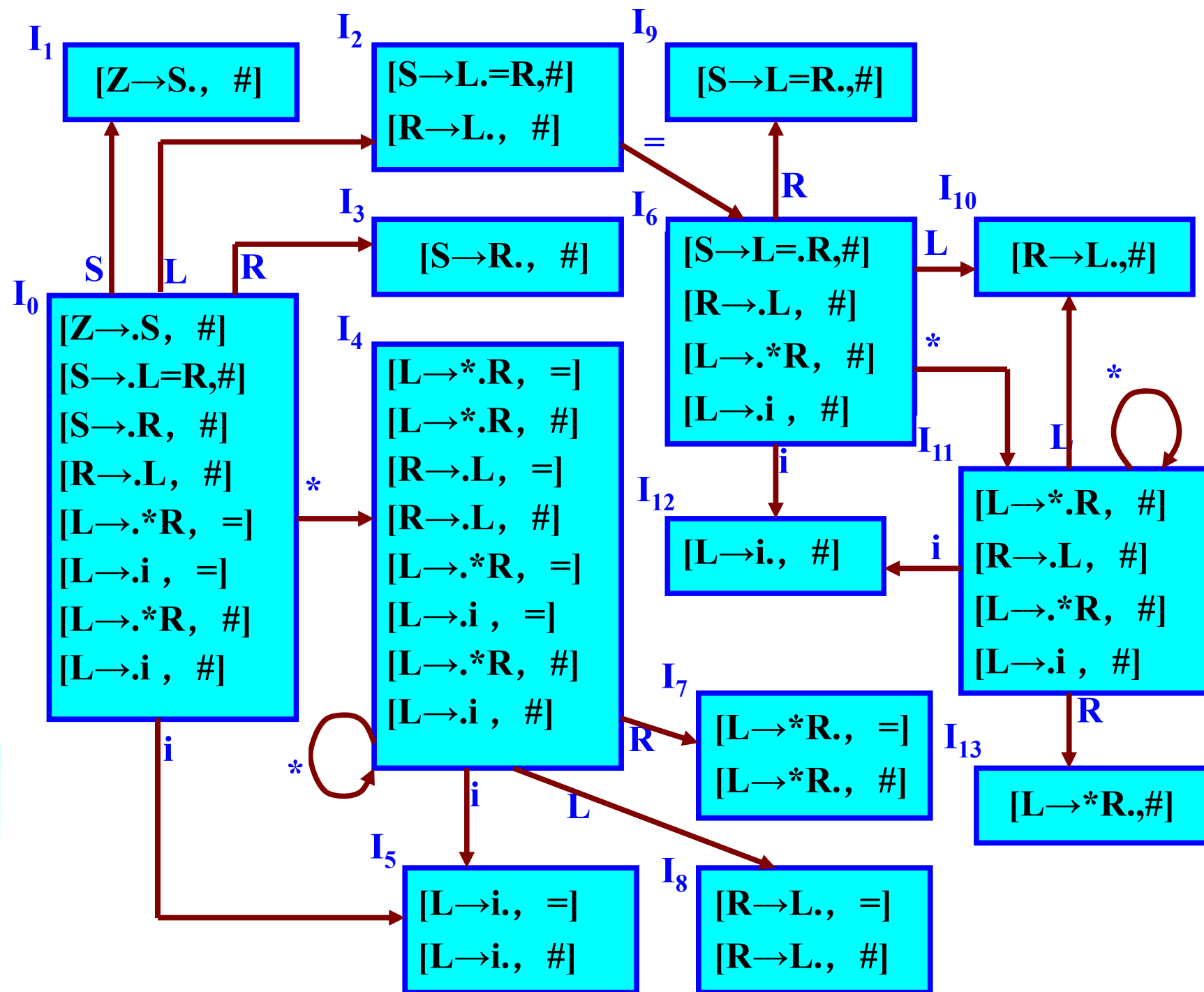
状态	ACTION				GOTO		
	i	*	=	#	S	L	R
0	S ₅	S ₄			1	2	3
1				acc			
2			S ₆	r ₃			
3				r ₂			
4	S ₅	S ₄				8	7
<u>5</u>			r ₅	r ₅			
6	S ₁₂	S ₁₁				10	9
<u>7</u>			r ₄	r ₄			
<u>8</u>			r ₃	r ₃			
9				r ₁			
<u>10</u>				r ₃			
<u>11</u>	S ₁₂	S ₁₁				10	13
<u>12</u>				r ₅			
<u>13</u>				r ₄			

LALR(1)分析表

状态	ACTION				GOTO		
	i	*	=	#	S	L	R
0	S ₅	S ₄			1	2	3
1				acc			
2			S ₆	r ₃			
3				r ₂			
4 <u>11</u>	S _{5<u>12</u>}	S _{4<u>11</u>}				8 <u>10</u>	7 <u>13</u>
5 <u>12</u>			r ₅	r ₅			
6	S ₁₂	S ₁₁				8 <u>10</u>	9
7 <u>13</u>			r ₄	r ₄			
8 <u>10</u>			r ₃	r ₃			
9				r ₁			

$G[Z]$:

- [0] $Z \rightarrow S$
- [1] $S \rightarrow L=R$
- [2] $S \rightarrow R$
- [3] $R \rightarrow L$
- [4] $L \rightarrow *R$
- [5] $L \rightarrow i$



LR(1)分析法

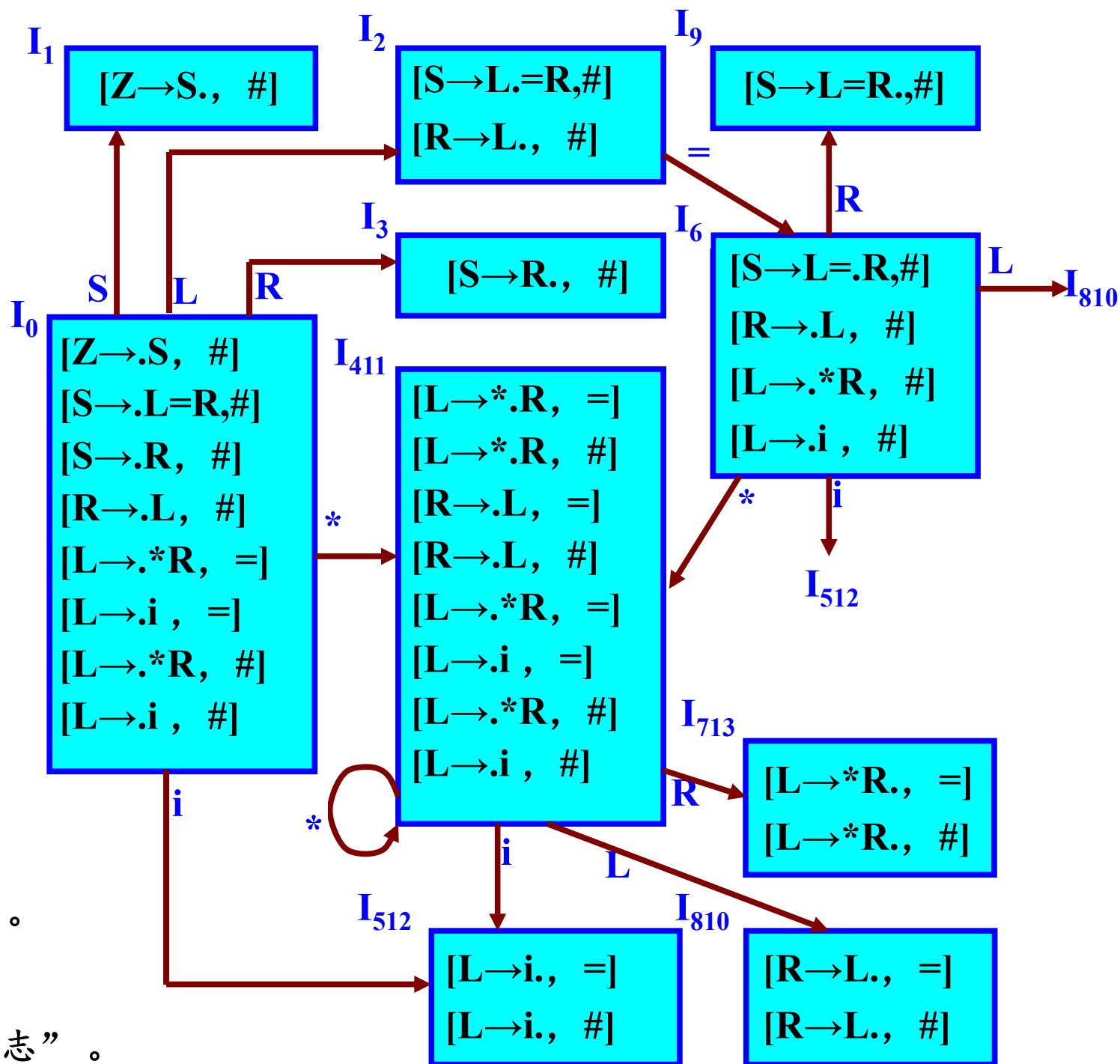
分析串 ' $i=i=$ '

$G[Z]:$

[0] $Z \rightarrow S$
 [1] $S \rightarrow L=R$
 [2] $S \rightarrow R$
 [3] $R \rightarrow L$
 [4] $L \rightarrow *R$
 [5] $L \rightarrow i$

LALR(1)分析法 分析串' $i=i=$ '

- ① 若 $[U \rightarrow x.ay, b] \in I_i$,
 $GOTO(I_i, a) = I_j$, 则
 $ACTION[i, a] = "S_j"$ 。
- ② 若 $[S' \rightarrow S., \#] \in I_i$,
 则 $ACTION[i, \#] = "acc"$ 。
- ③ 若 $[U \rightarrow x., a] \in I_i$,
 则置 $ACTION[i, a] = "r_j"$ 。
- ④ 若 $GOTO(I_i, V) = I_j$, 则
 置 $GOTO[i, V] = "j"$ 。
- ⑤ 其余元素均置“报错标志”。





7.5 LALR(1)分析表的构造

LALR(1)分析器与LR(1)分析器的比较

LALR(1)分析器行为与合并前的LR(1)分析器的行为类似，只是可能延迟报错的时间，但决不会放过错误。事实上，在移进下一符号之前，错误仍将被查出来。

LALR(1)方法合并过程中，在ACTION表中可能将归约与出错动作合并了，这是LALR将规定它做归约动作。可见，与LR(1)相比，LALR(1)放松了报错条件。但由于移进能力没有减弱，所以在下一个符号进栈之前总能报错，故它对错误的定向能力没有减弱。

7.5 LALR(1) 分析表的构造

2. LALR(1) 文法

冲突动作

不可能是“移进-归约”冲突

只可能是“归约-归约”冲突

LR(1) 文法合并后不可能存在
“移进-归约”冲突！

$[A \rightarrow \alpha. , a]$
 $[B \rightarrow \beta.a\gamma, b]$

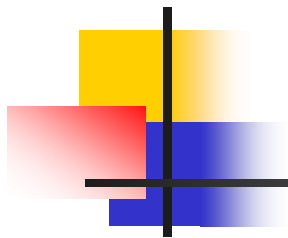
$G[S']:$

$S' \rightarrow S$

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

$A \rightarrow c$

$B \rightarrow c$



7.6 几个结论

1. 几种LR方法的比较——LR(0)与SLR(1)方法的比较

G[S]:

[1] $S \rightarrow S(S)$

[2] $S \rightarrow a$

G[S']:

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

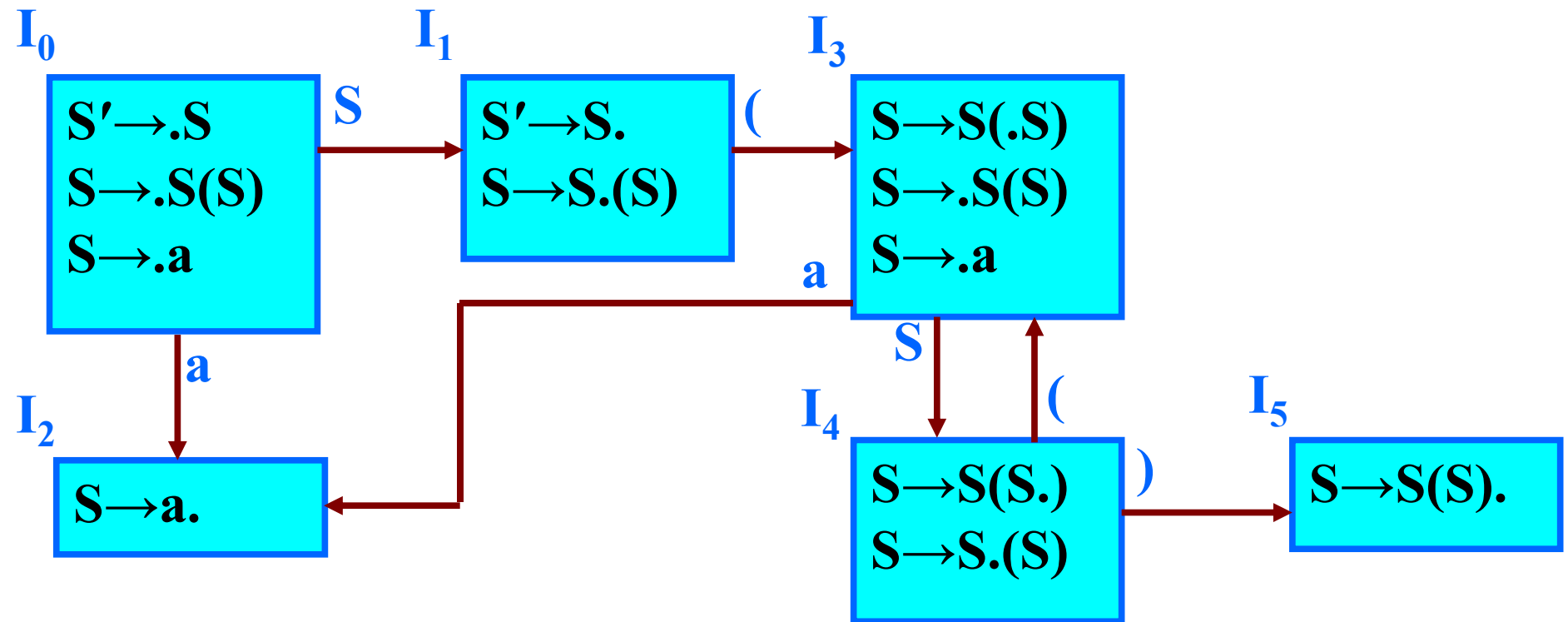
[2] $S \rightarrow a$

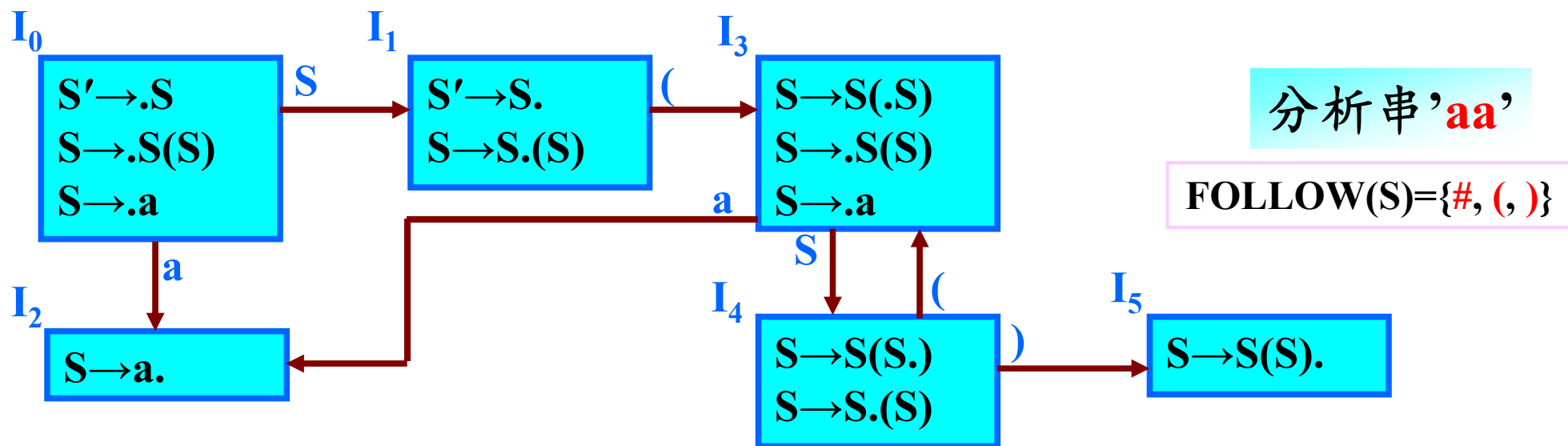
G[S]:

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow a$



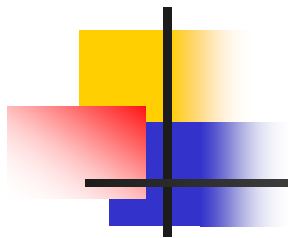


LR(0)分析表

状态	ACTION				GOTO
	a	()	#	
0	S ₂				1
1		S ₃		acc	
2	r ₂	r ₂	r ₂	r ₂	
3	S ₂				4
4		S ₃	S ₅		
5	r ₁	r ₁	r ₁	r ₁	

SLR(1)分析表

状态	ACTION				GOTO
	a	()	#	
0	S ₂				1
1		S ₃		acc	
2	error	r ₂	r ₂	r ₂	
3	S ₂				4
4		S ₃	S ₅		
5	error	r ₁	r ₁	r ₁	



7.6 几个结论

2. 几种LR方法的比较——LR(0)、SLR(1)与LR(1)方法的比较

$G[S]:$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \epsilon$

$G[S']:$

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

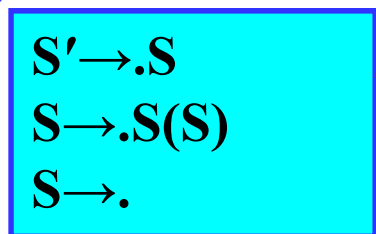
[2] $S \rightarrow \epsilon$

LR (0) 分析法, SLR (1) 分析法 DFA相同

G[S']:

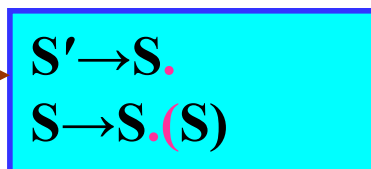
- [0] $S' \rightarrow S$
- [1] $S \rightarrow S(S)$
- [2] $S \rightarrow \varepsilon$

I₀



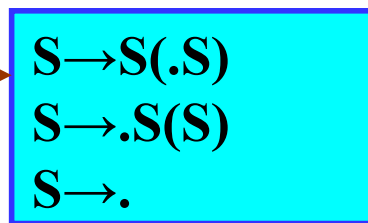
S

I₁



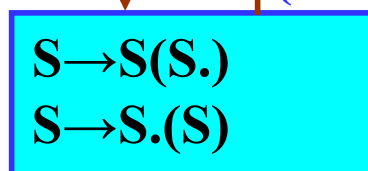
(

I₂



S

I₃



(

I₄



FOLLOW(S)={#, (,)}

LR(0)分析表

状态	ACTION			GOTO
	()	#	
0	r ₂	r ₂	r ₂	1
1	S ₂		acc	
2	r ₂	r ₂	r ₂	3
3	S ₂	S ₄		
4	r ₁	r ₁	r ₁	

SLR(1)分析表

状态	ACTION			GOTO
	()	#	
0	r ₂	r ₂	r ₂	1
1	S ₂		acc	
2	r ₂	r ₂	r ₂	3
3	S ₂	S ₄		
4	r ₁	r ₁	r ₁	

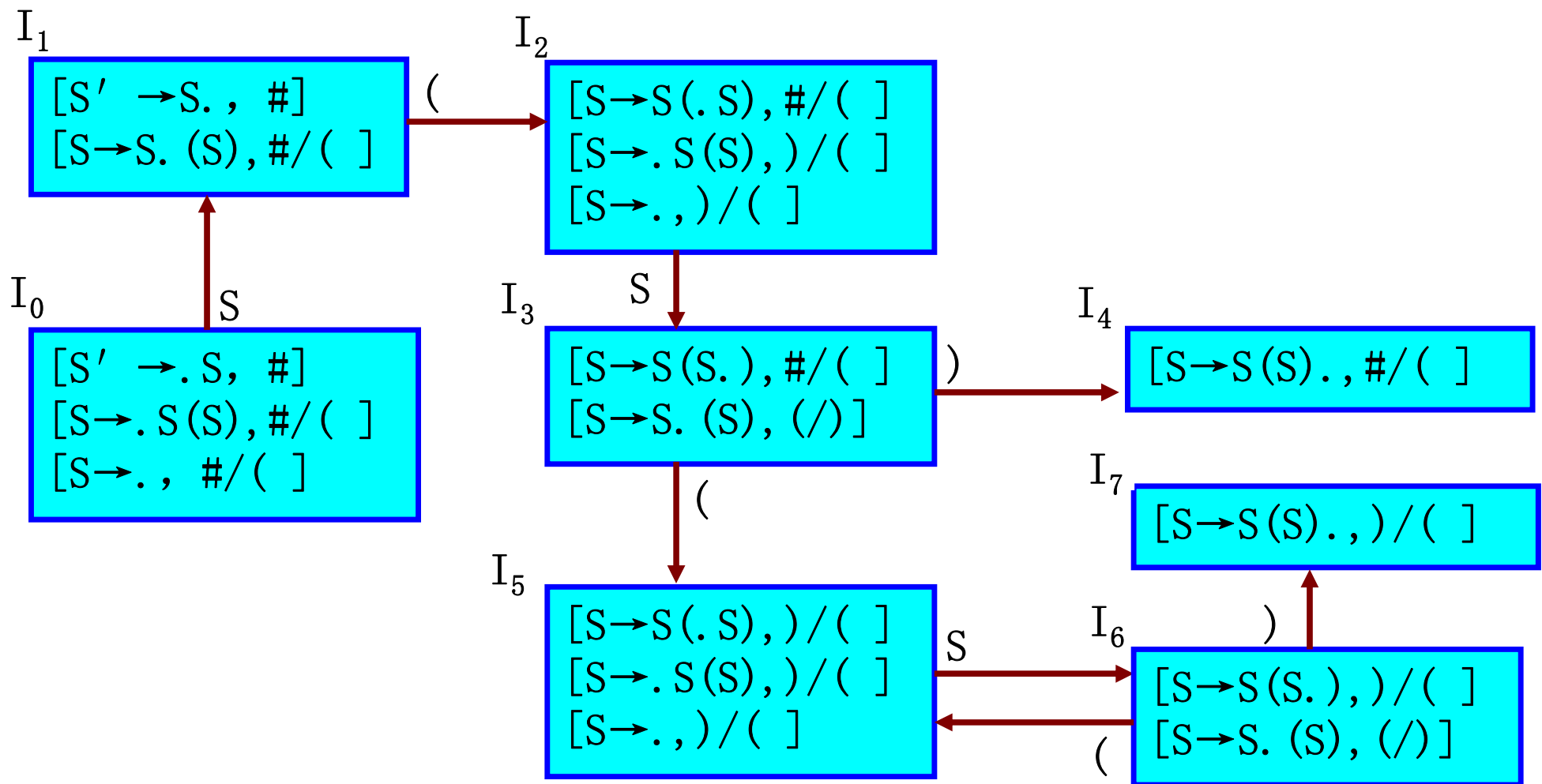
$G[S]$:

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \varepsilon$

LR (1) 分析法



LR(1)分析表

状态	ACTION			GOTO
	()	#	S
0	r ₂		r ₂	1
1	S ₂		acc	
2	r ₂	r ₂		3
3	S ₅	S ₄		
4	r ₁		r ₁	
5	r ₂	r ₂		6
6	S ₅	S ₇		
7	r ₁	r ₁		

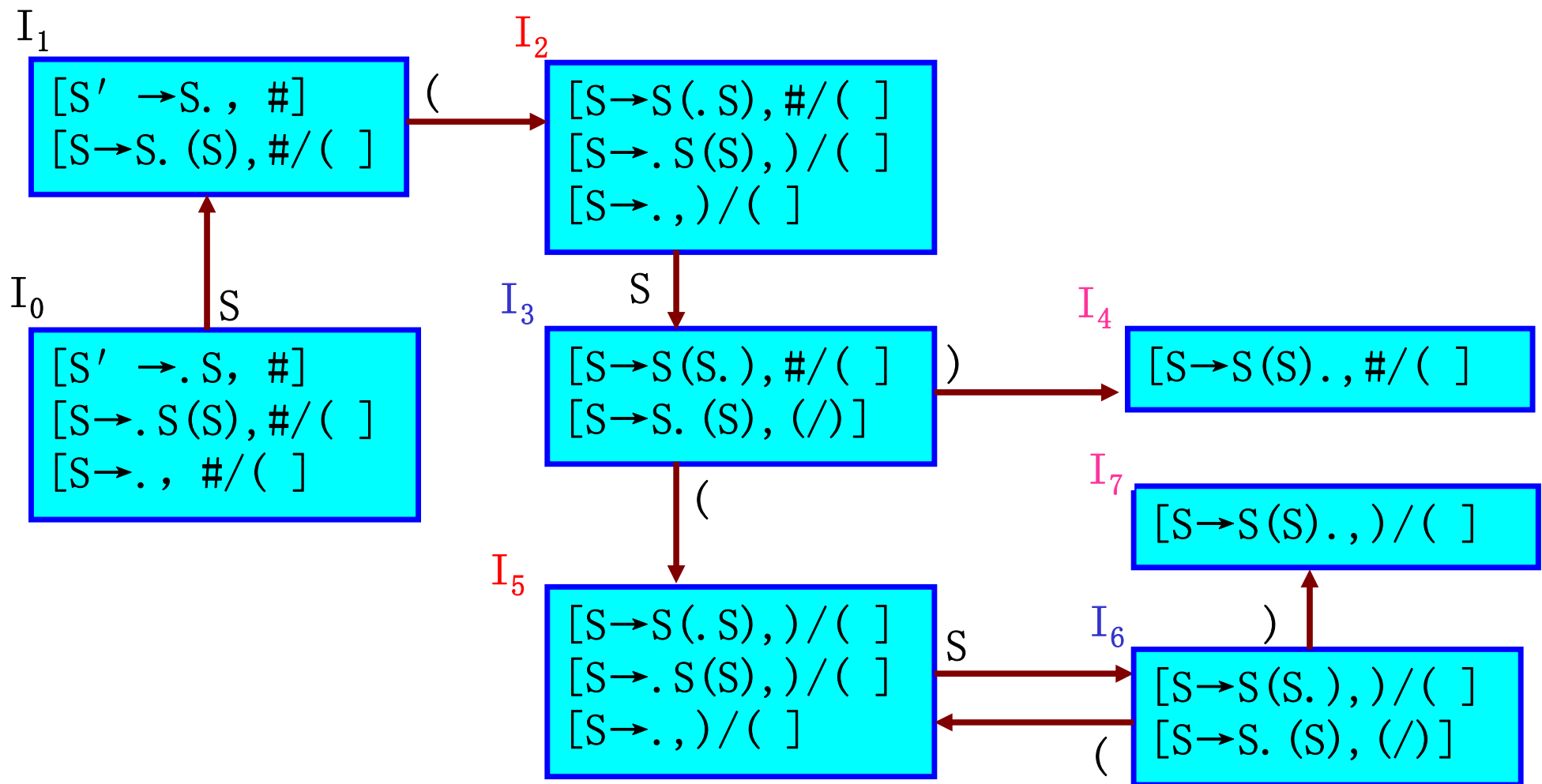
$G[S]:$

LR (1) 分析法

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \varepsilon$



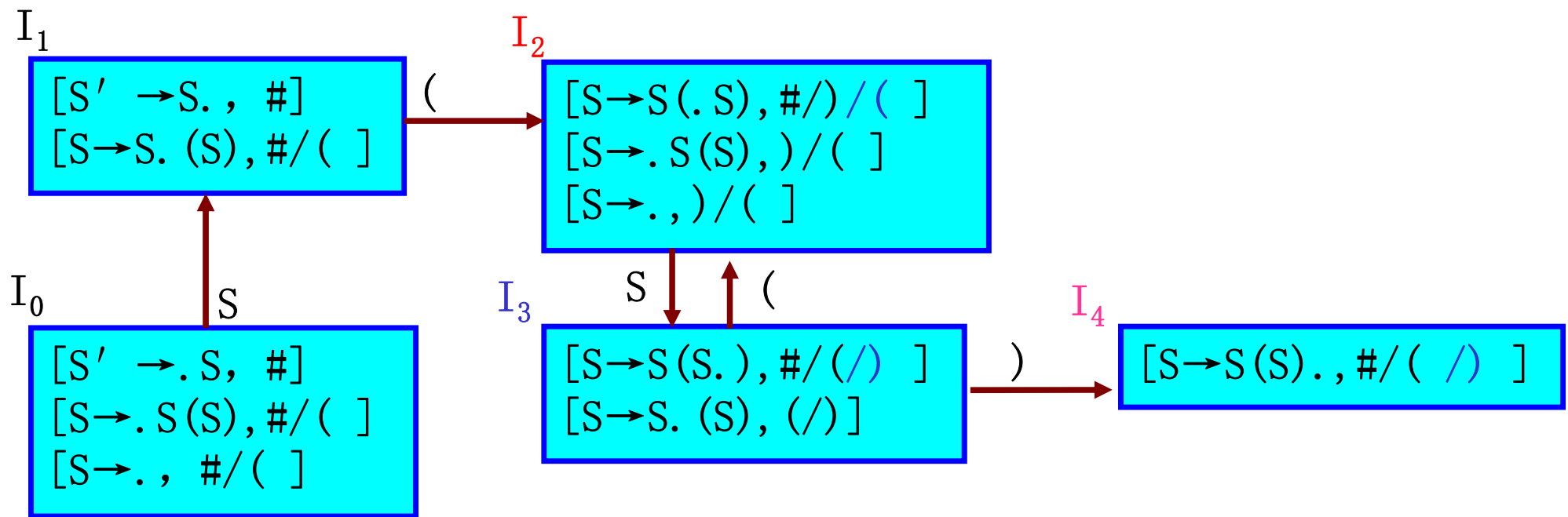
$G[S]$:

[0] $S' \rightarrow S$

[1] $S \rightarrow S(S)$

[2] $S \rightarrow \varepsilon$

LALR (1) 分析法



LR(1)分析表

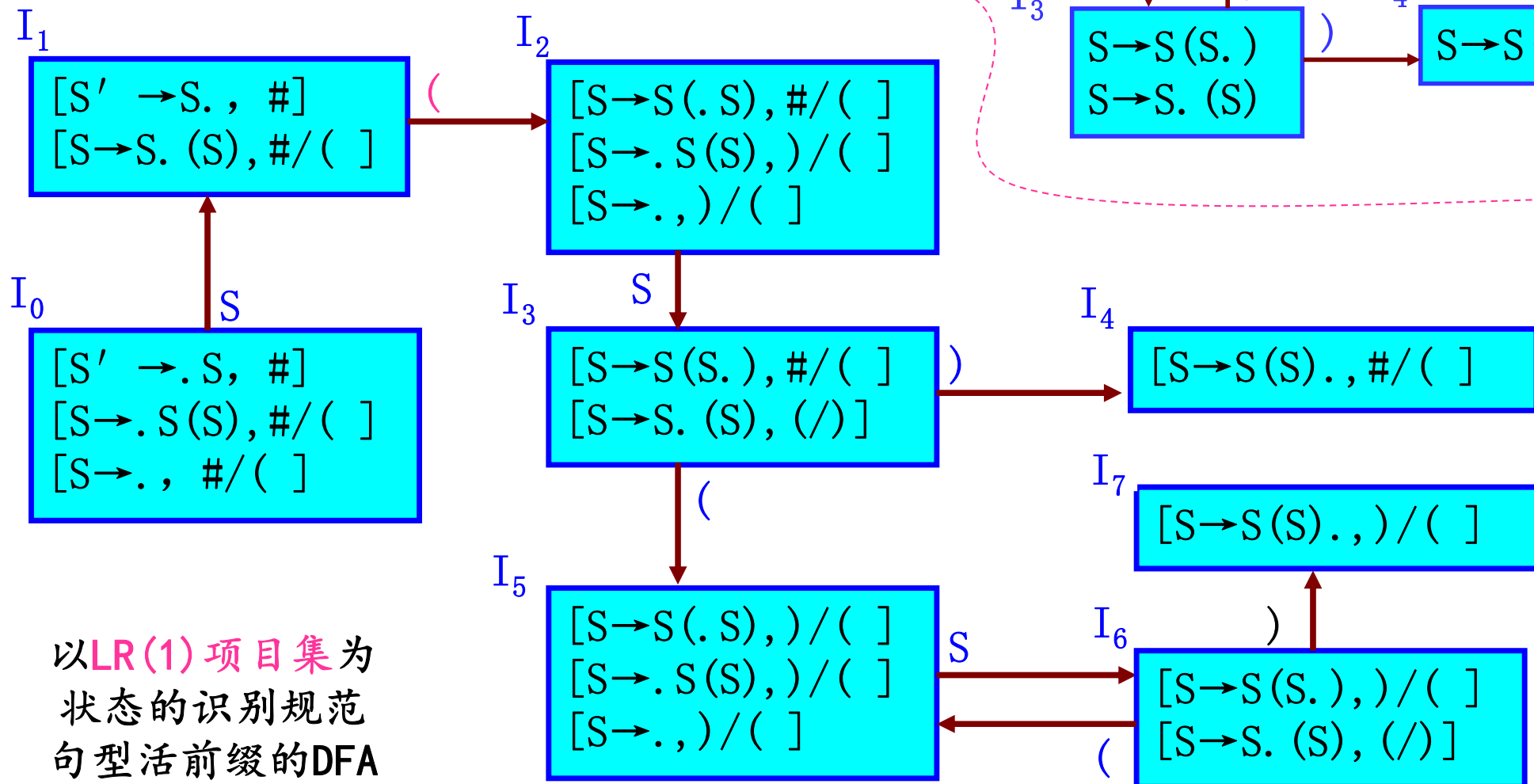
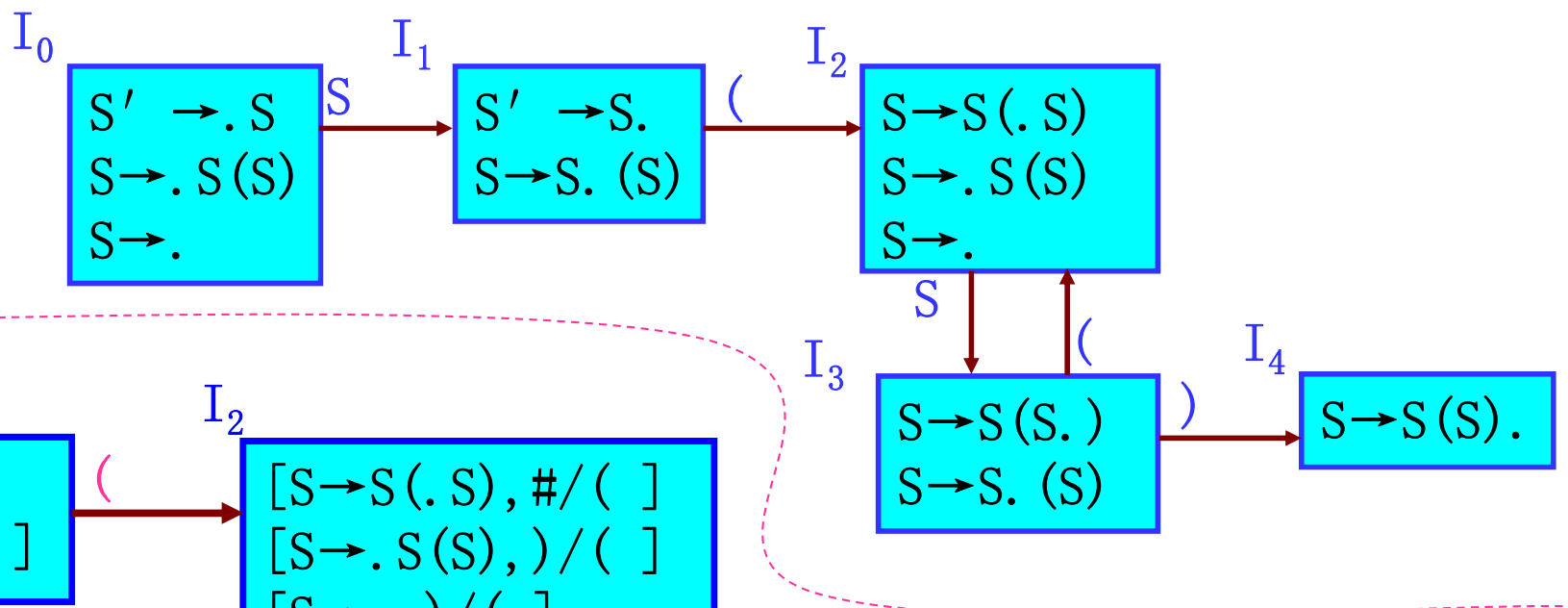
状态	ACTION			GOTO
	()	#	S
0	r ₂		r ₂	1
1	S ₂		acc	
2	r ₂	r ₂		3
3	S ₅	S ₄		
4	r ₁	error	r ₁	
5	r ₂	r ₂		6
6	S ₅	S ₇		
7	r ₁	r ₁	error	

LALR(1)分析表

状态	ACTION			GOTO
	()	#	S
0	r ₂		r ₂	1
1	S ₂		acc	
2	r ₂	r ₂		3
3	S ₂	S ₄		
4	r ₁	r ₁	r ₁	

用LALR(1) / LR(1)分析串'()'，
分析串'()'，

以LR(0)项目集为
状态的识别规范
句型活前缀的DFA



以LR(1)项目集为
状态的识别规范
句型活前缀的DFA

LR(0)、SLR(1)分析表

状态	ACTION			GOTO
	()	#	
0	r ₂	r ₂	r ₂	1
1	S ₂		acc	
2	r ₂	r ₂	r ₂	3
3	S ₂	S ₄		
4	r ₁	r ₁	r ₁	

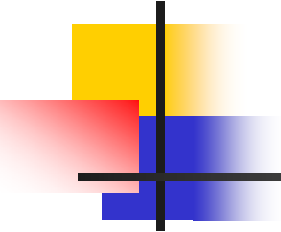
状态	ACTION			GOTO
	()	#	
0	r ₂	error	r ₂	1
1	S ₂		acc	
2	r ₂	r ₂	error	3
3	S ₂	S ₄		
4	r ₁	r ₁	r ₁	

LALR(1)分析表

LR(1)分析表

状态	ACTION			GOTO
	()	#	
0	r ₂	error	r ₂	1
1	S ₂		acc	
2	r ₂	r ₂	error	3
3	S ₅	S ₄		
4	r ₁	error	r ₁	
5	r ₂	r ₂	error	6
6	S ₅	S ₇		
7	r ₁	r ₁	error	

用SLR / LALR(1)分析串')...', '(



7.6 几个结论

2. 几种LR方法的比较——LR(0)、SLR(1)与LR(1)方法的比较

LR(0)分析表和SLR(1)分析表的状态数一样多，而LALR(1)分析表的状态数少于LR(1)分析表。

LR(1)方法中细致的状态划分，明确地指出了每一个句型中句柄后的符号，使得它的分析能力是四种方法中最强的，LALR(1)方法的能力弱于LR(1)方法，但强于SLR(1)方法，而LR(0)方法的能力最弱。

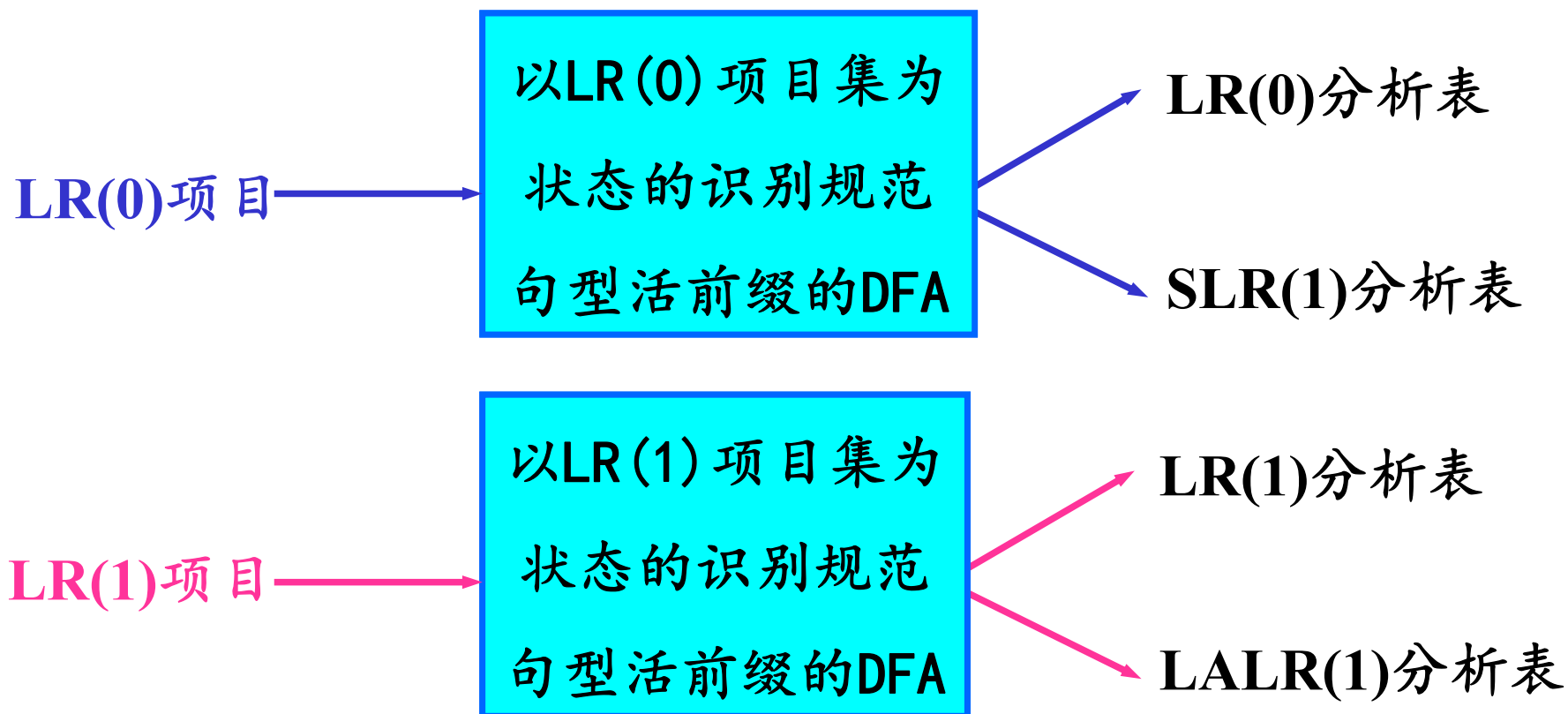
$\text{LR}(0)\text{文法} \subset \text{SLR}(1)\text{文法} \subset \text{LALR}(1)\text{文法} \subset \text{LR}(1)\text{文法}$

对于所有的k都有：

$\text{LR}(k)\text{文法} \subset \text{LR}(k+1)\text{文法}$

7.6 几个结论

2. 几种LR方法的比较——LR(0)、SLR(1)与LR(1)方法的比较





7.6 几个结论

3. LL(k)与LR(k)的比较

共同点:

- ①两者都借助于疑似句柄左部的全部符号及向右看k个符号来确定所应执行的唯一动作，识别过程严格从左到右，无回溯，功效高。
- ②都能及时察觉错误。
- ③识别程序都能自动生成。



7.6 几个结论

3. LL(k)与LR(k)的比较

区别：

- ①LR分析技术利用的是最右推导（最左归约）（由R隐指），
LL(k)分析技术利用的是最左推导（由第二个L隐指）。
- ②LL(k)要求文法无左递归，满足无回溯的条件，LR分析法则无此限制。
- ③LL(k)是自上而下构造推导，而LR(k)是自下而上构造归约。



7.6 几个结论

4. 关于LR(k)方法的几个结论

① LR(k) 文法是无二义性的（唯一的规范推导、最左归约）

LR(0) 文法 \subset SLR(1) 文法 \subset LALR(1) 文法 \subset LR(1) 文法

对于所有的k都有： LR(k) 文法 \subset LR(k+1) 文法

② 对于文法G和k，“G是否是LR(k) 文法”是可判定的

③ 对于文法G，“是否存在k，使得G是LR(k) 文法”是不可判定的

文法类的谱系

Unambiguous Grammars

LL(k)

LR(k)

LR(1)

LL(1)

LALR

SLR

LL(0)

LR(0)

Ambiguous Grammars



7.7 二义文法的LR分析

文法的二义性在分析表中体现为一个表项的冲突定义，从而导致语法分析程序无法依据分析表进行语法分析。

- LR文法是无二义性的
- 通过无二义性规则的使用，使对一些二义性文法也可以进行LR分析。



7.7 二义文法的LR分析

典型的无二义性规则有：

- ◆ 运算符优先级的规定
- ◆ 运算符结合顺序的规定
- ◆ 规定产生式和终结符号的优先级
- ◆ 规定终结符号的结合顺序
- ◆ 实际应用中，还可由用户指定优先级和结合性。
- ◆ 遇到“移进-归约”冲突时，采用移进
- ◆ 遇到“归约-归约”冲突时，优先使用列在前面的产生式进行归约



7.7 二义文法的LR分析

在为二义性文法构造LR分析表时，可先构造出识别规范句型活前缀的有穷自动机，得到有冲突的LR分析表，然后根据无二义性规则去除分析表中的冲突定义即可。



7.7 二义文法的LR分析

举例

$G[S]:$

0 $S' \rightarrow S$

1 $S \rightarrow iSeS$

2 $S \rightarrow iS$

3 $S \rightarrow a$

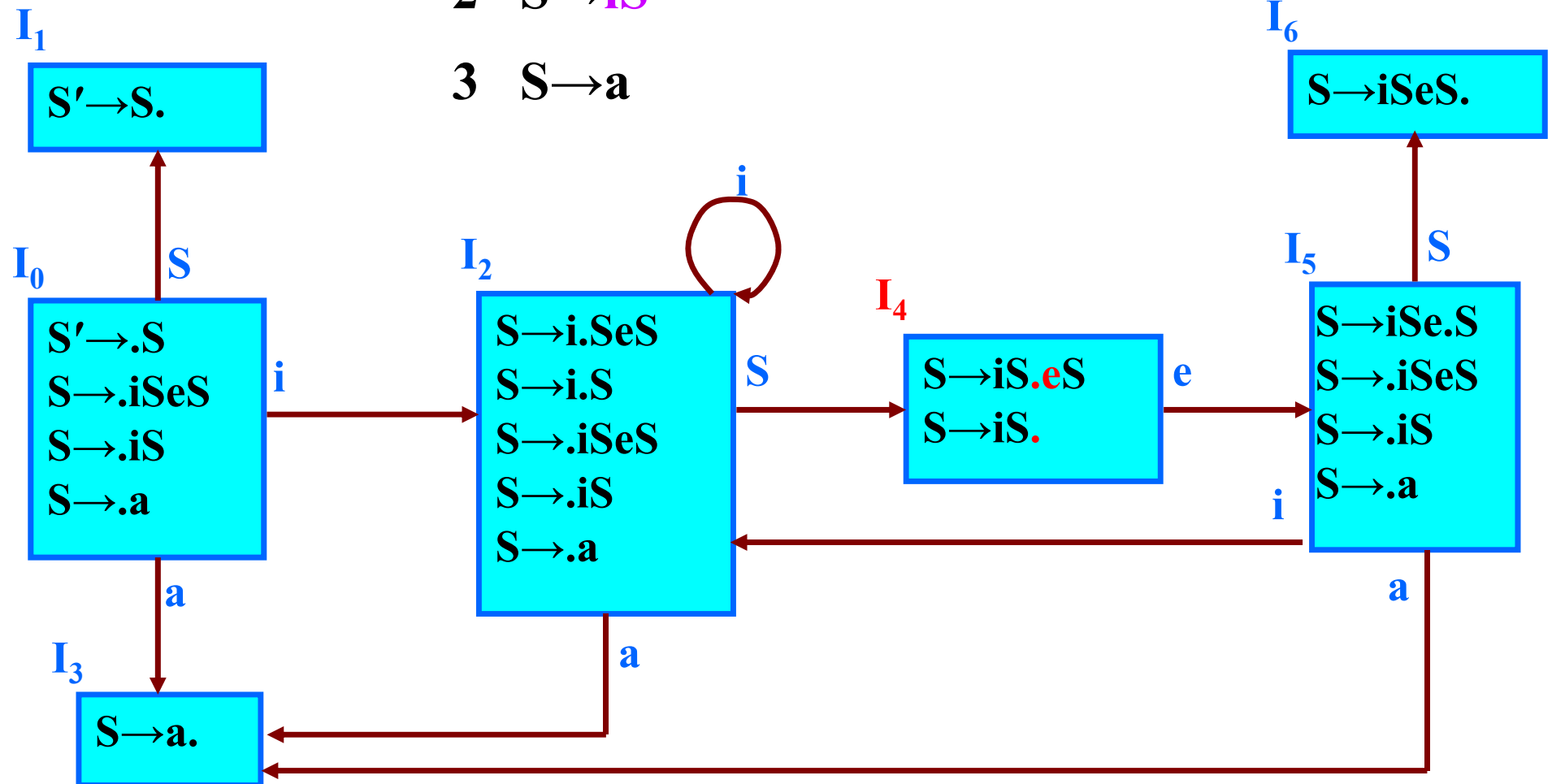
$G[S]:$

0 $S \rightarrow S$

1 $S \rightarrow iSeS$

2 $S \rightarrow iS$

3 $S \rightarrow a$

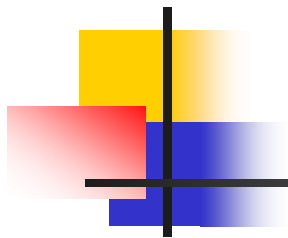


有冲突表项 ACTION(4, e) 的 LR 分析表

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

无冲突表项的 LR 分析表

无二义性规则（优先较长的产生式，采取“移进”）



本章内容回顾

- 基本思想
- 存在的问题
- 解决方法
- LR分析方法
- 二义性文法的LR分析



下章内容简介 —— 第8章

语法制导翻译