✧ 自底向上（Bottom-Up）语法分析

自底向上语法分析

- 自底向上分析思想
- 移进–归约分析
- LR 分析
- 二义文法在 LR 分析中的应用
- LR 分析中的出错处理
- LR(*K*)文法（选讲）
- 几类分析文法之间的关系（选讲）

自底向上分析思想

✧ 语法分析

– 核心问题：识别（*recognition*）与解析（*parsing*）

对任意上下文无关文法$G = (V, T, P, S)$和任意$w \in T^*$，是否有$w \in L(G)$？ 若成立，则给出分析树或（最左/右）推导步骤；否则，进行报错处理。

– 两种实现途径

自顶向下（*top-down*）分析
自底向上（*bottom-up*）分析

自底向上分析思想

✧ 自底向上分析的一般过程

- 从所要分析的终结符串开始进行归约；
- 每一步归约是在当前串中找到与某个产生式的右部相匹配的子串，然后将该子串用这一产生式的左部非终结符进行替换；如果找不到这样的子串，则回退到上一步归约前的状态，选择不同的子串或不同的产生式重试；
- 重复上一步骤，直到归约至文法开始符号；
- 如果不存在任何一个这样的归约，则表明该终结符串存在语法错误

自底向上分析思想

✧ 自底向上分析举例

– 单词序列 aaab 的一个自底向上分析过程

文法 G（S）：

$S \to AB$

$A \to aA \mid \varepsilon$

$B \to b \mid bB$

aaab	$(A \to \varepsilon)$
$\Leftarrow$ aaaAb	$(A \to aA)$
$\Leftarrow$ aaAb	$(A \to aA)$
$\Leftarrow$ aAb	$(B \to b)$
$\Leftarrow$ aAB	$(A \to aA)$
$\Leftarrow$ AB	$(S \to AB)$
$\Leftarrow$ S	

自底向上分析思想

✧ 自底向上分析中的非确定性

- 在每一步归约中，选择哪一个产生式以及匹配哪一个位置上的子串都可能是非确定的
- 这些非确定性导致分析过程会有很高的复杂性

自底向上分析思想

✧ 改进的方法

– 选择“可归约串”进行归约

在实用的自底向上分析中，总是选择某个“可归约串”进行归约，可大大减少回溯

对于一个句型而言，“可归约串” 一定是该句型的短语

对于文法G[S]，若 $S \overset{*}{\Rightarrow} \alpha A \delta$ 且 $A \overset{+}{\Rightarrow} \beta$，
则称β是句型αβδ相对于非终结符A的短语

自底向上分析思想

✧ 举例：短语

– 对于右边的文法G(S),

句子 aaab 的短语有：

ε：aaaεb;

a：aaaεb;

aa：aaaεb;

aaa：aaaεb;

aaab：aaaεb

b：aaab

句型aaAb的短语有：

aA：aaAb;

aaA：aaAb;

aaAb：aaAb

b：aaAb

文法 G（S）：

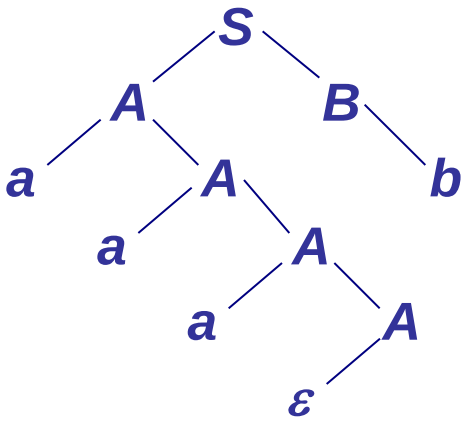
（1）S → AB

（2）A → aA

（3）A → ε

（4）B → b

（5）B → bB



自底向上分析思想

✧ 直接短语

– 对于文法 $G = (V_N, V_T, P, S)$，以及 $\alpha, \beta, \delta \in (V_N \cup V_T)^*$

若 $S \overset{*}{\Rightarrow} \alpha A \delta$ 且 $A \Rightarrow \beta$，则称

β是句型αβδ相对于非终结符A的直接短语

✧ 直接短语的作用

– 作为当前句型的一步“可归约串”

自底向上分析思想

✧ 举例：直接短语

– 对于右边的文法G(S)

句子 aaab 的直接短语有：

ε：aaaεb;

b：aaab

句型aaAb的直接短语有：

aA：aaAb;

b：aaAb

文法 G（S）：

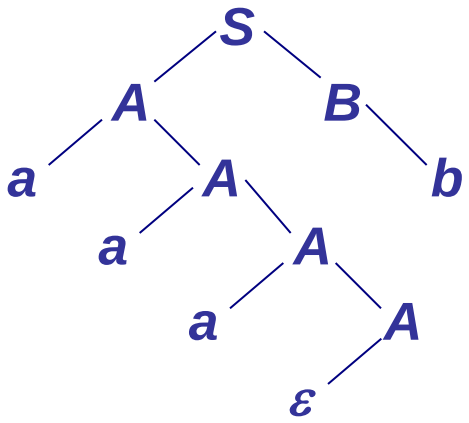
（1）S → AB

（2）A → aA

（3）A → ε

（4）B → b

（5）B → bB



自底向上分析思想

✧ 句柄

– 对于文法 $G=(V_N, V_T, P, S)$，以及

$\alpha,\beta\in(V_N\cup V_T)^*$，$w\in V_T^*$

若 $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w$ 且 $A \Rightarrow \beta$，则称

β是右句型 αβw 相对于非终结符 A 的句炳

✧ 句柄的作用

– 当前句型从左到右最先出现的“一步可归约串”

自底向上分析思想

✧ 举例：句柄

– 对于右边的文法G(S),

文法 G（S）：

（1）S → AB
（2）A → aA
（3）A → ε
（4）B → b
（5）B → bB

句子 aaab 的直接短语有：

ε：aaaεb;
b：aaab

aaab 的句柄：ε

右句型aaAb的直接短语有：

aA：aaAb;
b：aaAb

aaAb 的句柄：aA

$$S \underset{rm}{\Rightarrow} AB \underset{rm}{\Rightarrow} Ab$$
$$\underset{rm}{\Rightarrow} aAb \underset{rm}{\Rightarrow} aaAb$$
$$\underset{rm}{\Rightarrow} aaaAb \underset{rm}{\Rightarrow} aaab$$

自底向上分析思想

✧ 举例：句柄不一定唯一

– 对于右边的文法G(S),

句子 aaab 的直接短语有：

ε：aaaεb;

b：aaab

aaab 的句柄：ε

右句型aaAb的直接短语有：

aA：aaAb;

aaA：aaAb;

b：aaAb

aaAb 的句柄：aA，aaA

文法 G（S）：

（1）S → AB
（2）A → aA
（3）A → aaA
（4）A → ε
（5）B → b
（6）B → bB

不唯一的原因：
G（S）是二义文法，右句型的最右推导有多个

自底向上分析思想

✧ 举例：最右推导与最左归约

$G(S)$: $S \to (L) \mid a$
$L \to L,S \mid S$

S	短语	直接短语	句柄
$\Rightarrow_{rm}$ (L)	(L)	(L)	(L)
$\Rightarrow_{rm}$ (L,S)	(L,S) L,S	L,S	L,S
$\Rightarrow_{rm}$ $(L,(L))$	$(L,(L))$ $L,(L)$ (L)	(L)	(L)
$\Rightarrow_{rm}$ $(L,(L,S))$	$(L,(L,S))$ $L,(L,S)$ (L,S) L,S	L,S	L,S
$\Rightarrow_{rm}$ $(L,(L,a))$	$(L,(L,a))$ $L,(L,a)$ (L,a) L,a a	a	a
$\Rightarrow_{rm}$ $(L,(S,a))$	$(L,(S,a))$ $L,(S,a)$ (S,a) S,a a S	a S	S
$\Rightarrow_{rm}$ $(L,(a,a))$	$(L,(a,a))$ $L,(a,a)$ (a,a) a,a a a	a a	a
$\Rightarrow_{rm}$ $(S,(a,a))$	$(S,(a,a))$ $S,(a,a)$ (a,a) a,a a a S	a a S	S
$\Rightarrow_{rm}$ $(a,(a,a))$	$(a,(a,a))$ $a,(a,a)$ (a,a) a,a a a a	a a a	a

自底向上分析思想

✧ 自底向上分析的实现技术

– 移进–归约（*shift-reduce*）分析技术

LR分析和算符优先分析（参见清华教材第 3 版第 5 章）采用移进–归约分析技术

自底向上分析思想

✧ 与自顶向下技术相比

– 功能较强大

原因在于推导和归约过程有如下差别：推导时仅观察可推导出的输入串的一部分，而归约时可归约的输入串整体已全部出现

– 利于出错处理

输入符号查看后才被移进

– 构造较复杂

手工构造有难度

但存在很好的自动构造技术

（如 Yacc 工具采用 LALR 分析技术）

移进-归约分析

✧ 基本原理

– 借助一个下推栈（分析栈）和一个基于有限状态控制的分析引擎

分析引擎根据当前状态、下推栈当前状态/内容、剩余输入单词序列来确定如下动作之一，然后进入新状态：

- *Reduce*: 依确定的方式对位于栈顶的短语进行归约
- *Shift*: 从输入序列移进一个单词
- *Error*: 发现语法错误，进行错误处理/恢复
- *Accept*: 分析成功

移进–归约分析

✧ 移进–归约分析的一个例子

文法 $G[E]$:

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$

待分析输入串:
$v+v*d\#$

步骤	分析栈	余留输入串	动作
(0)		$v+v*d\#$	Shift
(1)	v	$+v*d\#$	Reduce(6)
(2)	F	$+v*d\#$	Reduce(4)
(3)	T	$+v*d\#$	Reduce(2)
(4)	E	$+v*d\#$	Shift
(5)	$E+$	$v*d\#$	Shift
(6)	$E+v$	$*d\#$	Reduce(6)
(7)	$E+F$	$*d\#$	Reduce(4)
(8)	$E+T$	$*d\#$	Shift
(9)	$E+T*$	$d\#$	Shift
(10)	$E+T*d$	$\#$	Reduce(7)
(11)	$E+T*F$	$\#$	Reduce(3)
(12)	$E+T$	$\#$	Reduce(1)
(13)	E	$\#$	Accept

移进-归约分析

(接上页)

– 对应一个最右推导

将分析栈中的符号串和余留输入串并置，若逆向观察从步骤（13）至步骤（1）的每一个归约步骤，则对应一个最右推导，也称为规范推导（*canonical derivation*）

此为LR分析过程句柄作为“可归约串”

步骤	分析栈	余留输入串	动作
(0)		*v* + *v* ∗ *d* #	*Shift*
(1)	*v*	+ *v* ∗ *d* #	*Reduce(6)*
(2)	*F*	+ *v* ∗ *d* #	*Reduce(4)*
(3)	*T*	+ *v* ∗ *d* #	*Reduce(2)*
(4)	*E*	+ *v* ∗ *d* #	*Shift*
(5)	*E* +	*v* ∗ *d* #	*Shift*
(6)	*E* + *v*	∗ *d* #	*Reduce(6)*
(7)	*E* + *F*	∗ *d* #	*Reduce(4)*
(8)	*E* + *T*	∗ *d* #	*Shift*
(9)	*E* + *T* ∗	*d* #	*Shift*
(10)	*E* + *T* ∗ *d*	#	*Reduce(7)*
(11)	*E* + *T* ∗ *F*	#	*Reduce(3)*
(12)	*E* + *T*	#	*Reduce(1)*
(13)	*E*	#	*Accept*

移进-归约分析

✧ 移进-归约分析的另一个例子

文法 $G[E]$:

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$

步骤	分析栈	余留输入串	动作
(0)		$v + v * d \#$	Shift
(1)	v	$+ v * d \#$	Reduce
(2)	F	$+ v * d \#$	Shift
(3)	$F +$	$v * d \#$	Shift
(4)	$F + v$	$* d \#$	Reduce
(5)	$F + F$	$* d \#$	Shift
(6)	$F + F *$	$d \#$	Shift
(7)	$F + F * d$	$\#$	Reduce
(8)	$F + F * F$	$\#$	Reduce
(9)	$F + T$	$\#$	Reduce
(10)	E	$\#$	

对应的推导过程不一定是规范推导

待分析输入串:
$v+v*d\#$

可对应于算符优先分析过程
最左素短语作为“可归约串”

移进–归约分析

✧ 分析过程确定化的关键：解决两类冲突

– 移进–归约（*shift-reduce*）冲突

到达一个不能确定下一步应该移进还是应该归约的状态

例如，有产生式

S → if E then S | if E then S else S

考虑对于如下串进行移进–归约分析

if E then if E then S else S

当 if E then if E then S 出现在分析栈中时，是移进 else，还是归约 if E then S？

移进–归约分析

✧ 分析过程确定化的关键：解决两类冲突

– 归约–归约（*reduce-reduce*）冲突

到达这样的状态：有对多于一个短语进行归约的选择

例如，有产生式

$$A \rightarrow aA \mid aaA$$

考虑对于串 aaab 进行移进–归约分析

当分析到某一步时，aaA出现在分析栈中（**b** 位于剩余输入区），是用产生式 $A \rightarrow aA$ 归约 aA，还是用产生式 $A \rightarrow aaA$ 归约 aaA？

✧ 表驱动方法

– 借助于分析表

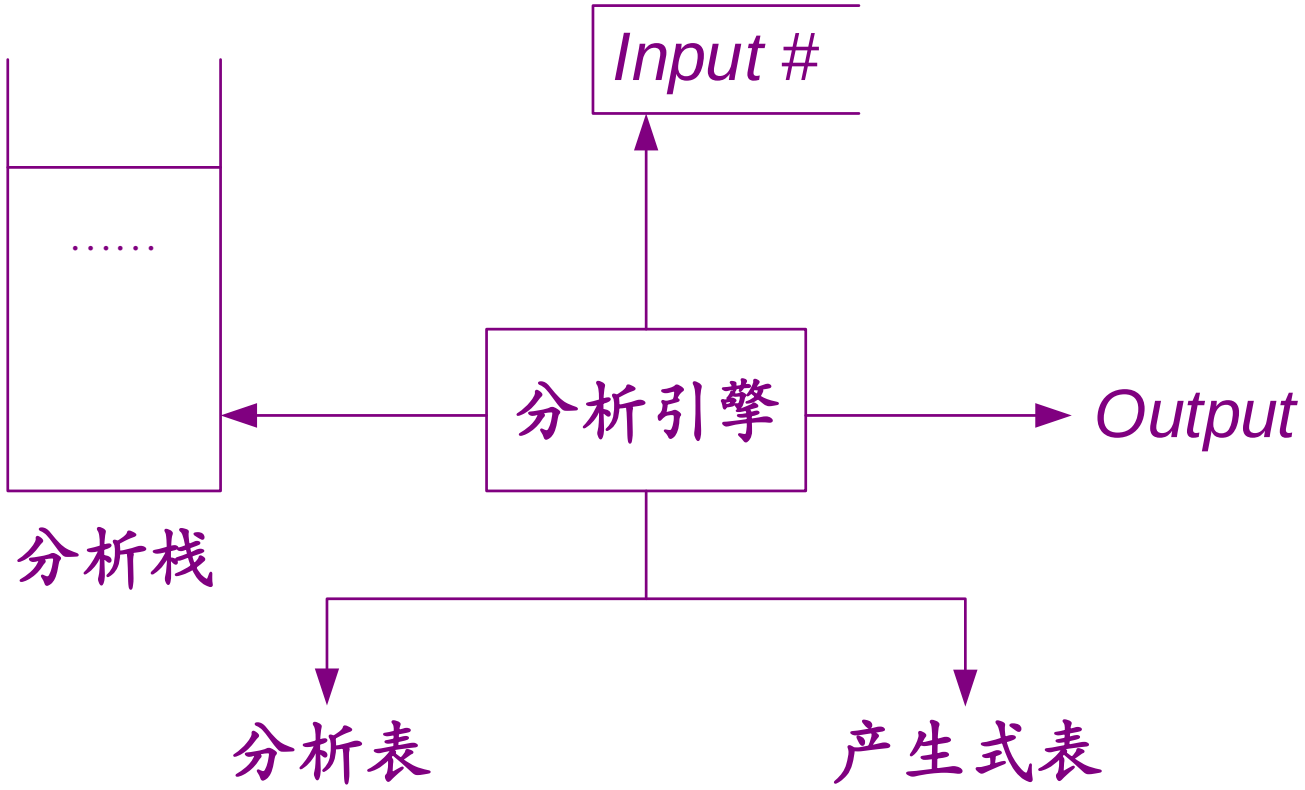
多数移进–归约分析的实现都是基于表驱动方法

分析引擎根据当前状态、输入单词查询分析表，确定 *Reduce*，*Shift*，*Error* 和*Accept* 等动作

分析表应当可以体现出移进–归约冲突和归约–归约冲突的解决方法

LR分析中的LR分析表以及算符优先分析中的算符优先分析表可用于上述目的

✧ 表驱动移进-归约分析模型



LR分析

- LR分析基础
- LR（0）分析
- SLR（1）分析
- LR（1）分析
- LALR（1）分析

LR 分析基础

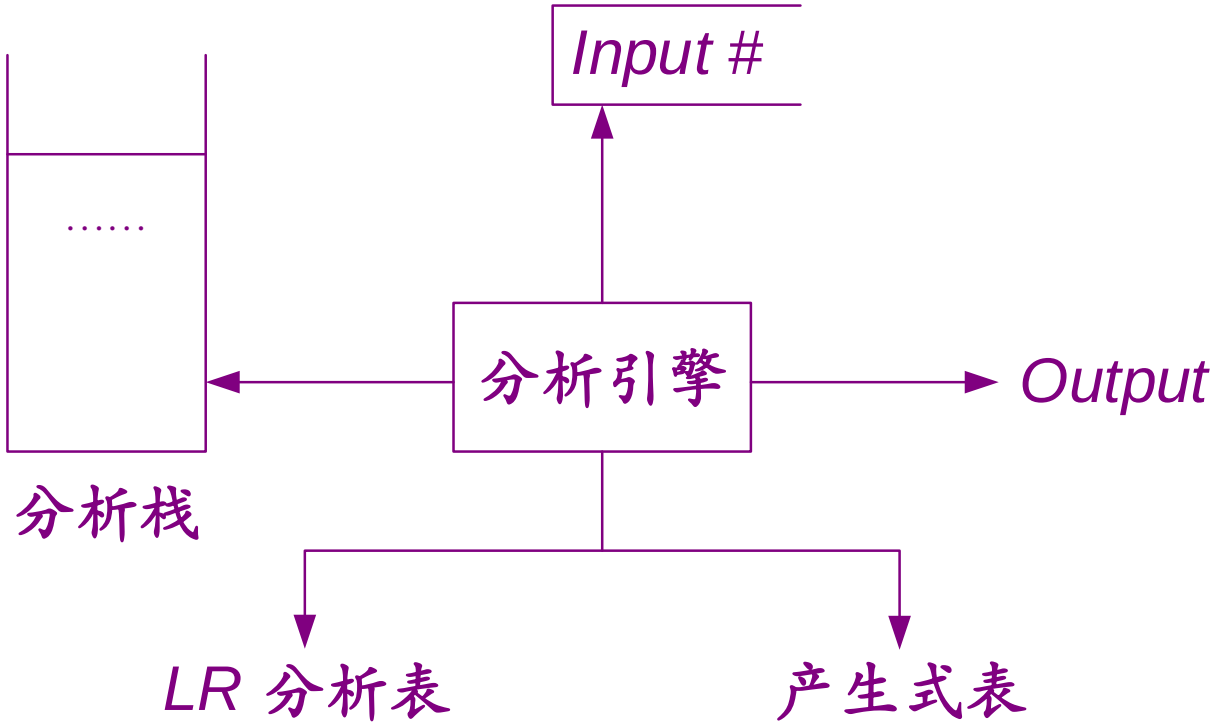
✧ LR的含义

- “L”, 代表从左（*Left*）向右扫描输入单词序列
- “R”,代表产生的是最右（*Rightmost*）推导

LR 分析基础

✧ LR 分析模型

– LR 分析是一种表驱动的移进–归约分析



LR 分析基础

✧ 主要学习四种 LR 分析技术

– LR（0）分析

适用于 LR（0）文法

– SLR（1）分析

适用于 SLR（1）文法

– LR（1）分析

适用于 LR（1）文法

– LALR（1）分析

适用于 LALR（1）文法

“0”– 向前查看 0 个符号

Simple LR(1)

“1”– 向前查看 1 个符号

LookAhead LR(1)

LR 分析基础

✧ LR 分析表

- LR 分析表的构造是LR 分析的基础
- LR（0），SLR（1），LR（1）和 LALR（1）四种分析方法可共享同样的LR 分析表

  本讲的LR 分析表专指此类LR 分析表

LR 分析基础

✧ LR 分析表举例

– 文法： $G[E]$

(1) $E \to E+T$ (2) $E \to T$
(3) $T \to T*F$ (4) $T \to F$
(5) $F \to (E)$ (6) $F \to v$ (7) $F \to d$

栈顶状态	ACTION							GOTO		
	v	*d*	*	+	(	)	#	*E*	*T*	*F*
0	s5	s6			s4			1	2	3
1				s7			acc			
2			s8	r2		r2	r2			
3			r4	r4		r4	r4			
4	s5	s6			s4			9	2	3
5			r6	r6		r6	r6			
6			r7	r7		r7	r7			
7	s5	s6			s4				10	3
8	s5	s6			s4					11
9				s7		s12				
10			s8	r1		r1	r1			
11			r3	r3		r3	r3			
12			r5	r5		r5	r5			

LR 分析基础

✧ LR 分析表

– 使用两张表

- ACTION 表 告诉分析引擎：在栈顶状态为*k*，当前输入符号是 **a** 时做什么
  ACTION [*k*,a]=*si*， *Shift*：状态 *i* 移进栈顶
  ACTION [*k*,a]=*rj*， *Reduce*：按第 *j* 条产生式归约
  ACTION [*k*,a]=*acc*， *Accept*：分析完成
  ACTION [*k*,a]=*err*，*Error*：发现错误（常标为空白）
- GOTO 表 GOTO[*i*,*A*]=*j* 告诉分析引擎：
  在依产生式 $A\rightarrow\beta$ 归约之后，栈顶状态为*i* 时，要将新状态 *j* 移进栈顶
  （依产生式 $A\rightarrow\beta$ 归约时，要将栈顶的 $|\beta|$ 个状态弹出）

LR 分析基础

✧ LR 分析表举例

– 栈顶状态

(1) $E \to E+T$ (2) $E \to T$
(3) $T \to T*F$ (4) $T \to F$
(5) $F \to (E)$ (6) $F \to v$ (7) $F \to d$

- 初始状态 记为“0” = { <ε, E>, <ε, E+T>, <ε, T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d> }

若“E”入栈，则转移至{ <E, ε>, <E, +T> }，记为状态“1”

若“T”入栈，则转移至{ <T, ε>, <T, *F> }，记为状态“2”

若“F”入栈，则转移至{ <F, ε> }，记为状态“3”

若“(”入栈，则转移至{ <(, E)>, <ε, E+T>, <ε, T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d> }，记为状态“4”

若“v”入栈，则转移至{ <v, ε> }，记为状态“5”

若“d”入栈，则转移至{ <d, ε> }，记为状态“6”

LR 分析基础

✧ LR 分析表举例

– 栈顶状态

(1) $E \to E+T$ (2) $E \to T$
(3) $T \to T*F$ (4) $T \to F$
(5) $F \to (E)$ (6) $F \to v$ (7) $F \to d$

- 状态“1” = { <E, ε>, <E, +T> }

  若“+”入栈，则转移至 {<E+, T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d>}，记为状态“7”

- 状态“2” = { <T, ε>, <T, *F> }

  若“*”入栈，则转移至 { <T*, F>, <ε, (E)> , <ε, v> , <ε, d> }，记为状态“8”

LR 分析基础

✧ LR 分析表举例

– 栈顶状态

(1) $E \to E+T$ (2) $E \to T$
(3) $T \to T*F$ (4) $T \to F$
(5) $F \to (E)$ (6) $F \to v$ (7) $F \to d$

- 状态“4” = { <(, E)>, <ε, T>, <ε, E+T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d> }

若“E”入栈，则转移至{ <(E,)> , <E, +T> }，记为状态“9”

若“T”入栈，则转移至{ <T, ε>, <T,*F> }，即为状态“2”

若“F”入栈，则转移至{ <F, ε> }，即为状态“3”

若“(”入栈，则转移至{ <(, E)>, <ε, E+T>, <ε, T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d> }，即为状态“4”

若“v”入栈，则转移至{ <v, ε> }，即为状态“5”

若“d”入栈，则转移至{ <d, ε> }，即为状态“6”

LR 分析基础

✧ LR 分析表举例

– 栈顶状态

（1）$E \to E+T$　（2）$E \to T$
（3）$T \to T*F$　（4）$T \to F$
（5）$F \to (E)$　（6）$F \to v$　（7）$F \to d$

- 状态“7” = { <E+, T>, <ε, T*F> , <ε, F>,
  <ε, (E)> , <ε, v> , <ε, d> }

若“T”入栈，则转移至{< E+T, ε>, <T, *F>, 记为状态“10”

若“F”入栈，则转移至{ <F, ε> }，即为状态“3”

若“(”入栈，则转移至{ <(, E)>, <ε, E+T>, <ε, T>, <ε, T*F> ,
<ε, F>, <ε, (E)> , <ε, v> , <ε, d> } ，即为状态“4”

若“v”入栈，则转移至{ <v, ε> }，即为状态“5”

若“d”入栈，则转移至{ <d, ε> }，即为状态“6”

LR 分析基础

✧ LR 分析表举例

– 栈顶状态

（1）$E \to E+T$　（2）$E \to T$
（3）$T \to T*F$　（4）$T \to F$
（5）$F \to (E)$　（6）$F \to v$　（7）$F \to d$

- 状态“8” = { <T*, F>, <ε, (E)> , <ε, v> , <ε, d> }

  若“F”入栈，则转移至{ < T*F, ε> }，记为状态“11”

  若“(”入栈，则转移至{ <(, E)>, <ε, E+T>, <ε, T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d> } ，即为状态“4”

  若“v”入栈，则转移至{ <v, ε> }，即为状态“5”

  若“d”入栈，则转移至{ <d, ε> }，即为状态“6”

LR 分析基础

✧ LR 分析表举例

– 栈顶状态

（1）$E \to E+T$　（2）$E \to T$
（3）$T \to T*F$　（4）$T \to F$
（5）$F \to (E)$　（6）$F \to v$　（7）$F \to d$

- 状态“9” = { <(E,)> , <E, +T> }

  若“)”入栈，则转移至 { < (E), ε> }，记为状态“12”

  若“+”入栈，则转移至 {<E+, T>, <ε, T*F> , <ε, F>, <ε, (E)> , <ε, v> , <ε, d>}，即为状态“7”

- 状态“10” = { < E+T, ε>, <T, *F> }

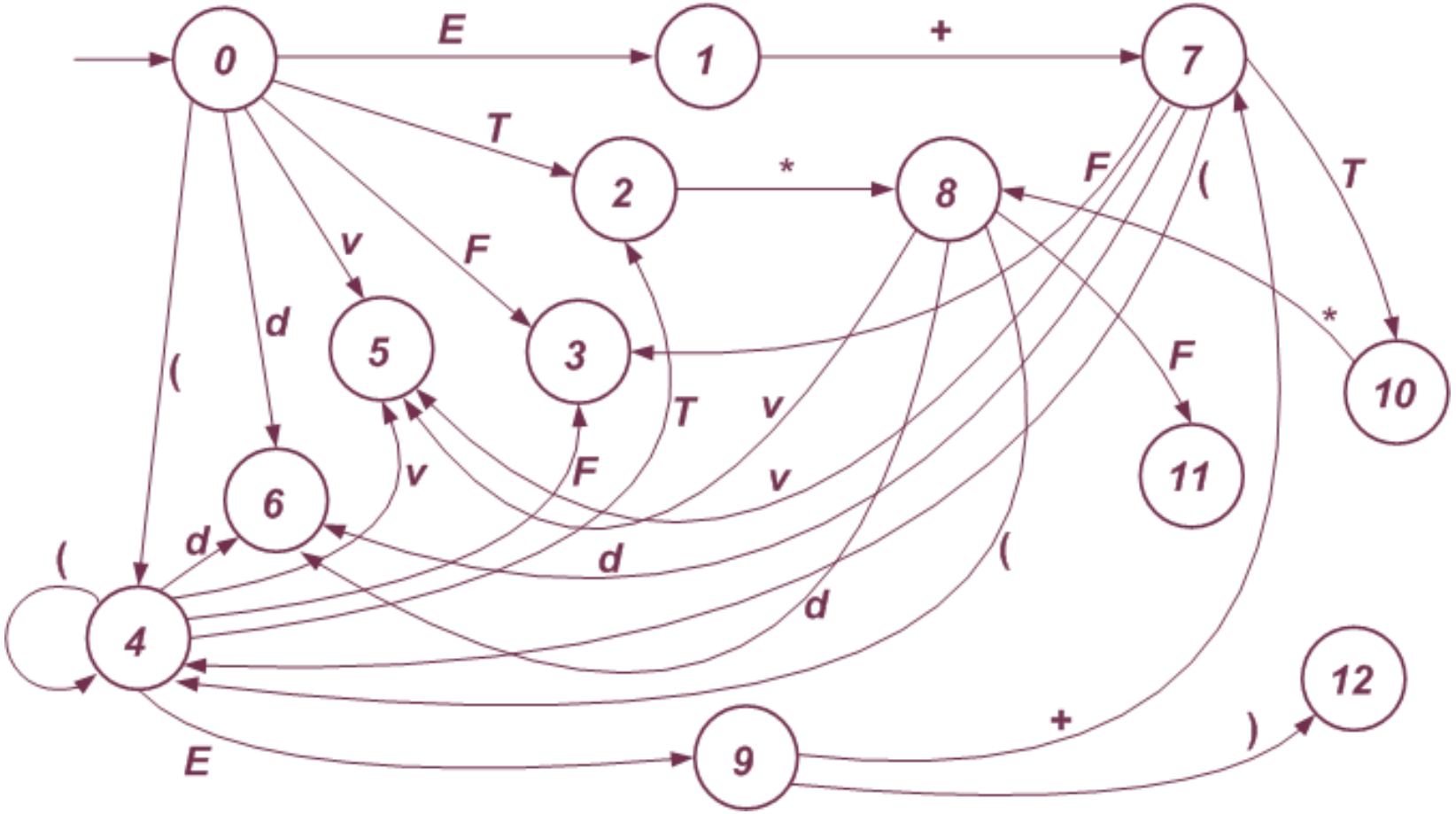
  若“*”入栈，则转移至 { <T*, F>, <ε, (E)> , <ε, v> , <ε, d> }，即为状态“8”

LR 分析基础

✧ LR 分析表举例

– 状态转移图

(1) $E \to E+T$ (2) $E \to T$
(3) $T \to T*F$ (4) $T \to F$
(5) $F \to (E)$ (6) $F \to v$ (7) $F \to d$



LR 分析基础

✧ LR 分析算法

– 置 *ip* 指向输入串 *w* 的首符号，置初始栈顶状态为 *0*
令 *i* 为栈顶状态，a 是 *ip* 指向的符号，重复如下步骤:

```
if (ACTION[i,a]=sj ) {
    PUSH j ; /*进栈*/      ip 前进 ; /*指向下一输入符号*/
}
else if (ACTION[i,a]=rj ) {     /*第 j 条产生式为 A→β*/
        POP |β| 项; /*位于栈顶部的 |β| 个状态退栈*/
        令当前栈顶状态为 k;  PUSH GOTO[k,A];
    }
    else if (ACTION[i,a]=acc ) return;  /*成功*/
        else error;     /*报错/错误恢复*/
```

LR 分析基础

✧ LR 分析过程举例

– 文法：$G[E]$

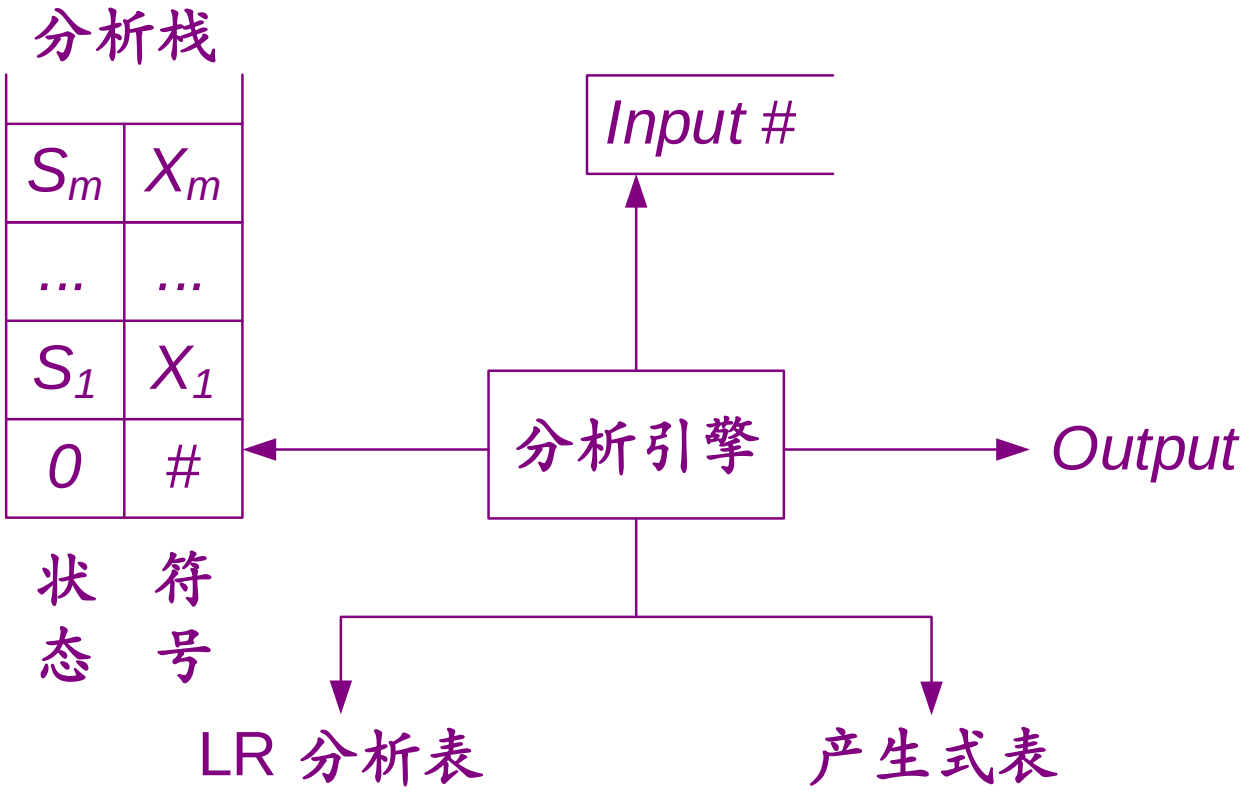
– 输入串：$v+v*d$

（1）$E\rightarrow E+T$　（2）$E\rightarrow T$
（3）$T\rightarrow T*F$　（4）$T\rightarrow F$
（5）$F\rightarrow (E)$　（6）$F\rightarrow v$　（7）$F\rightarrow d$

分析栈	余留输入串	分析动作
0	$v+v*d\#$	ACTION [0,v]=s5
0 5	$+v*d\#$	ACTION [5,+]=r6, GOTO [0,F]=3
0 3	$+v*d\#$	ACTION [3,+]=r4, GOTO [0,T]=2
0 2	$+v*d\#$	ACTION [2,+]=r2, GOTO [0,E]=1
0 1	$+v*d\#$	ACTION [1,+]=s7
0 1 7	$v*d\#$	ACTION [7,v]=s5
0 1 7 5	$*d\#$	ACTION [5,*]=r6, GOTO [7,F]=3
0 1 7 3	$*d\#$	ACTION [3,*]=r4, GOTO [7,T]=10
0 1 7 10	$*d\#$	ACTION [10,*]=s8
0 1 7 10 8	$d\#$	ACTION [8,d]=s6
0 1 7 10 8 6	$\#$	ACTION [6,#]=r7, GOTO [8,F]=11
0 1 7 10 8 11	$\#$	ACTION [11,#]=r3, GOTO [7,T]=10
0 1 7 10	$\#$	ACTION [10,#]=r1, GOTO [0,E]=1
0 1	$\#$	ACTION [1,#]=acc

LR 分析基础

✧ 带符号栈的LR 分析模型



LR 分析基础

✧ 带符号栈的 LR 分析算法

– 置 ***ip*** 指向输入串 ***w*** 的首符号，置状态栈顶为 *0*，状态栈顶为 #，令 ***i*** 为栈顶状态，a 是 ***ip*** 指向的符号，重复如下步骤：

```
if (ACTION[i,a]=sj ) {
    PUSH j, a; /*进栈*/      ip 前进；/*指向下一输入符号*/
}
else if (ACTION[i,a]=rj ) {     /*第 j 条产生式为 A→β*/
        POP |β| 项; /*位于两个栈顶部的 |β| 个元素退栈*/
        令当前状态栈顶为 k;  PUSH GOTO[k,A], A ;
    }
    else if (ACTION[i,a]=acc ) return;  /*成功*/
        else error;     /*报错/错误恢复*/
```

LR 分析基础

✧ 带分析栈的LR 分析过程举例

– 文法：$G[E]$

– 输入串： $v+v*d$

（1）$E\to E+T$　（2）$E\to T$
（3）$T\to T*F$　（4）$T\to F$
（5）$F\to(E)$　（6）$F\to v$　（7）$F\to d$

分析栈	余留输入串	分析动作
0#	$v+v*d\#$	ACTION [0,v]=s5
0# 5v	$+v*d\#$	ACTION [5,+]=r6, GOTO [0,F]=3
0# 3F	$+v*d\#$	ACTION [3,+]=r4, GOTO [0,T]=2
0# 2T	$+v*d\#$	ACTION [2,+]=r2, GOTO [0,E]=1
0# 1E	$+v*d\#$	ACTION [1,+]=s7
0# 1E 7+	$v*d\#$	ACTION [7,v]=s5
0# 1E 7+ 5v	$*d\#$	ACTION [5,*]=r6, GOTO [7,F]=3
0# 1E 7+ 3F	$*d\#$	ACTION [3,*]=r4, GOTO [7,T]=10
0# 1E 7+ 10T	$*d\#$	ACTION [10,*]=s8
0# 1E 7+ 10T 8*	$d\#$	ACTION [8,d]=s6
0# 1E 7+ 10T 8* 6d	$\#$	ACTION [6,#]=r7, GOTO [8,F]=11
0# 1E 7+ 10T 8* 11F	$\#$	ACTION [11,#]=r3, GOTO [7,T]=10
0# 1E 7+ 10T	$\#$	ACTION [10,#]=r1, GOTO [0,E]=1
0# 1E	$\#$	ACTION [1,#]=acc

LR 分析基础

- 如何获得 LR 分析表
  - LR（0），SLR（1），LR（1）和 LALR（1）四种分析方法分别讨论

LR（0）分析

✧ 核心概念

– 增广文法（augmented grammar）

对于文法 $G = (V_N, V_T, P, S)$，增加如下产生式

$$S' \rightarrow S$$

其中，$S' \notin V_N \cup V_T$，得到 G 的增广文法

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

注：增广文法等价于原文法；增广文法的开始符号不会出现在任何产生式的右部

LR（0）分析

✧ 核心概念

– 活前缀（*viable prefix*）

- 对于文法 $G=(V_N, V_T, P, S)$，以及

$$\alpha,\beta\in(V_N\cup V_T)^*,\ w\in V_T^*$$

若 $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha Aw$ 且 $A \Rightarrow \beta$，即 β 为句柄，

则 $\alpha\beta$ 的任何前缀 γ 都是文法 G 的活前缀

– 增广文法的活前缀

- 若 $G'[S']$ 是上述 $G[S]$ 的增广文法（增加 $S' \rightarrow S$）

由于 $S' \overset{*}{\underset{rm}{\Rightarrow}} S'$ 且 $S' \Rightarrow S$，故 S 是 G' 的活前缀

LR（0）分析

✧ 活前缀举例

– 对于右边的文法G(S),

句子 aaab 是一个右句型，其唯一的句柄为：

ε： aaaεb;

所以 aaa 的任何前缀都是文法的活前缀：ε， a ， aa ， aaa

右句型 aaAb 的唯一的句柄为：

aA：aaAb;

所以 aaA 的任何前缀都是文法的活前缀：ε， a ， aa ， aaA

文法 G（S）：

（1）$S \rightarrow AB$

（2）$A \rightarrow aA$

（3）$A \rightarrow \varepsilon$

（4）$B \rightarrow b$

（5）$B \rightarrow bB$

LR（0）分析

✧ 活前缀与句柄的关系

一个活前缀是某一右句型的前缀，它不超过该右句型的某个句柄

- 活前缀已含有该句柄的全部符号，表明该句柄对应的产生式 $A\rightarrow\alpha$ 的右部 α 已出现在栈顶
- 活前缀只含该句柄的一部分符号，表明该句柄对应的产生式 $A\rightarrow\alpha_1\alpha_2$ 的右部子串 α_1 已出现在栈顶，期待从输入串中看到 α_2 推导出的符号串
- 活前缀不含有该句柄的任何符号，此时期待从输入串中看到该句柄对应的产生式 $A\rightarrow\alpha$ 的右部所推导出的符号串

LR（0）分析

✧ 核心概念

– LR（0）FSM

- 每个上下文无关文法 G 都对应一个LR（0）FSM
- 由 G 的增广文法 G' 直接构造其 LR（0）FSM
- 文法 $G = (V_N, V_T, P, S)$ 的 LR（0）FSM 可以看作一个字母表为 $V_N \cup V_T$ 的 DFA

LR（0）分析

✧ LR（0）FSM 的构造

– LR（0）FSM 的状态

- LR（0）FSM 的状态是一个特殊的 LR（0）项目（*item*）集
- 一个LR（0）项目是在右端某一位置有圆点的产生式

如，产生式 **A→xyz** 对应如下 4 个 LR（0）项目：

A→.xyz
A→x.yz
A→xy.z
A→xyz.

圆点标志着已分析过的串与该产生式匹配的位置

LR（0）分析

✧ LR（0）FSM 的构造

– LR（0）项目解析

设 $G'[S']$ 是文法 $G=(V_N, V_T, P, S)$的增广文法

根据圆点所在的位置和圆点后是终结符还是非终结符或为空，把项目分为以下几种：

移进项目：形如 $A \to \alpha .a\beta$，其中$a \in V_T$, $\alpha, \beta \in (V_N \cup V_T)^*$

待约项目：形如 $A \to \alpha .B\beta$

归约项目：形如 $A \to \alpha .$

接受项目：形如 $S' \to S .$

LR（0）分析

✧ LR（0）FSM 的构造

– LR（0）FSM 的状态

- LR（0）FSM 的状态是一个 LR（0）项目集的闭包（*closure*）
- 计算LR（0）项目集 I 的闭包 CLOSURE(I)的算法：

```
function  CLOSURE(I)
{  J := I;
   repeat  for  J 中的每个项目A →α .Bβ 和产生式 B→ γ
                do 若 B→ .γ 不在 J 中，则加 B→ .γ 到 J 中
   until  上一次循环不再有新项目加到J中
   return  J
};
```

LR（0）分析

✧ LR（0）FSM 的构造

– LR（0）FSM 的初态

设文法 $G[S]$ 的增广文法为 $G'[S']$, 则 G' 的LR（0）FSM 的初态　$\mathbf{I_0 = CLOSURE(\{S' \rightarrow .S\})}$

例 右边文法$G[E]$的增广文法为 $G'[E']$ ，其 LR（0）FSM 的初态

$\mathbf{I_0} = \{ E' \rightarrow .E,$
$E \rightarrow .E+T,$
$E \rightarrow .T,$
$T \rightarrow .(E),$
$T \rightarrow .d$
$\}$

$G[E]$:

(1) $E \rightarrow E+T$
(2) $E \rightarrow T$
(3) $T \rightarrow (E)$
(4) $T \rightarrow d$

LR（0）分析

✧ LR（0）FSM 的构造

– LR（0）FSM 的状态转移函数

GO (I,X) = CLOSURE(J)

其中，**I**为LR（0）FSM 的状态（闭包的项目集），**X** 为文法符号，**J={ A→αX.β | A→α.Xβ ∈I}**

– 从 LR（0）FSM 的初态出发，应用上述转移函数，可逐步构造出完整的 LR（0）FSM

LR（0）分析

✧ LR（0）FSM 的构造

– 计算 LR（0）FSM 的所有状态的集合

设文法 *G*[*S*] 的增广文法为 *G'*[*S'*], 则 **LR（0）FSM** 的所有状态的集合 **C** 可由如下算法计算:

C:= { CLOSURE ({S'→.S})}

Repeat

For C 中每一项目集 **I** 和每一文法符号**X**

Do if **GO(I,X)** 非空且不属于**C**

Then 把 **GO(I,X)** 放入**C**中

Until **C** 不再增大

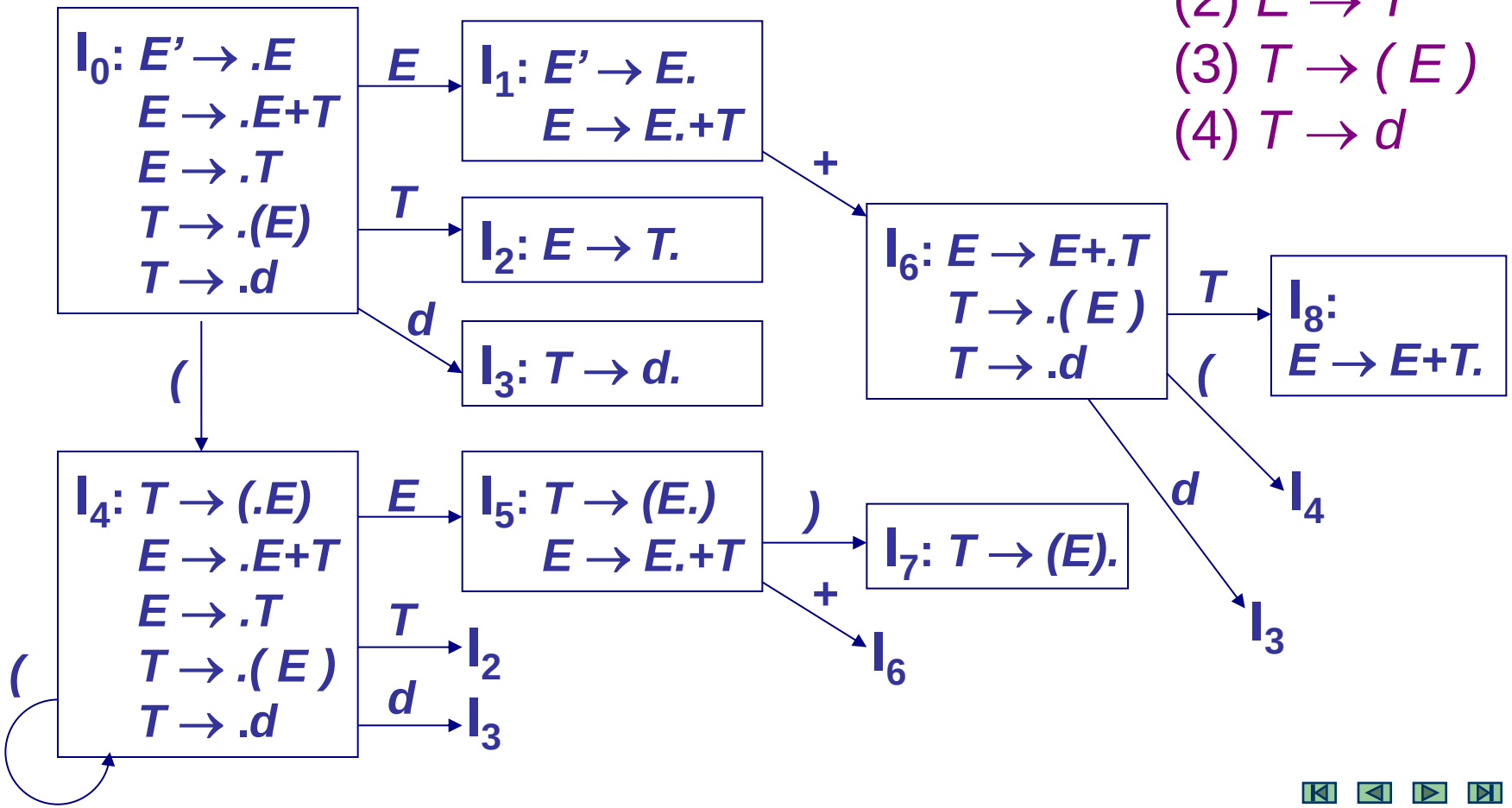
LR（0）分析

✧ LR（0）FSM 的构造举例

– G[E]的增广文法G'[E'] 的 LR（0）FSM

G[E]:

(1) $E \rightarrow E+T$
(2) $E \rightarrow T$
(3) $T \rightarrow (\ E\ )$
(4) $T \rightarrow d$



LR（0）分析

✧ LR（0）FSM 的语言

– 结论

- 文法 $G=(V_N, V_T, P, S)$ 的 LR（0）FSM 可以看作一个字母表为 $V_N \cup V_T$ 的 DFA（所有状态都是终态；严格地说，还应该有一个死状态，它不是终态）. 设G的增广文法为G'. 可以证明:

  该 DFA 的语言是 G'的所有活前缀的集合

  (证明略)

- 由此可知，对任何句型，我们不会错过任何可归约的句柄，或者说不会错过任何最右推导

LR（0）分析

✧ 进一步理解 LR（0）FSM 的状态

– 增广文法的每个活前缀有唯一对应的状态

- 从初态经这些活前缀可达该状态
- 该状态中所有项目针对这些活前缀均是有效的

注：对一个文法 $G = (V_N, V_T, P, S)$，如果存在最右推导

$$S \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha\beta_1\beta_2 w,$$

我们称 项目 $A \to \beta_1 . \beta_2$ 针对活前缀$\alpha\beta_1$是有效的（valid）

因此，该状态中若有项目 $A' \to \beta_1'. \beta_2'$，则一定有

$$S' \Rightarrow_{rm}^* \alpha' A' w' \Rightarrow_{rm} \alpha'\beta_1'\beta_2' w',$$

这里，$\alpha'\beta_1'$ 是对应于该状态的活前缀之一

- 针对这些活前缀的所有有效项目均隶属于该状态

（上述结论的证明超出本课程范围）

LR（0）分析

✧ LR（0）分析表的构造

– 假定**C**={I_0, I_1,…，I_n}，令状态I_k对应的 LR（0）分析表的栈顶状态为k；令含有项目$S'\rightarrow .S$ 的状态为I_0，因此 0 为初态。ACTION 表项和 GOTO 表项可按如下方法构造：

- 若项目$A\rightarrow \alpha .a\beta$属于 I_k 且 GO (I_k, a)= I_j, a 为终结符，则置 ACTION[k, a] 为“把状态j和符号a移进栈”，简记为“sj”；
- 若项目$A\rightarrow \alpha .$ 属于I_k，那么，对任何终结符a，置ACTION[k, a]为“用产生式$A\rightarrow \alpha$进行归约”，简记为“rj”;其中，假定$A\rightarrow \alpha$为文法*G* 的第j个产生式；
- 若项目$S'\rightarrow S.$ 属于I_k，则置ACTION[k, #]为“接受”，简记为“acc”；
- 若GO (I_k, A)= I_j，A为非终结符，则置GOTO(k, A)=j；
- 分析表中凡不能用上述规则填入信息的空白格均置上“出错标志”

LR（0）分析

✧ LR（0）分析表的构造举例

– 增广文法：　（0）$E'\to E$　（1）$E\to E+T$
$G'[E']$　（2）$E\to T$　（3）$T\to (E)$　（4）$T\to d$

栈顶状态	ACTION					GOTO	
	d	+	(	)	#	*E*	*T*
0	s3		s4			1	2
1		s6			acc		
2	r2	r2	r2	r2	r2		
3	r4	r4	r4	r4	r4		
4	s3		s4			5	2
5		s6		s7			
6	s3		s4				8
7	r3	r3	r3	r3	r3		
8	r1	r1	r1	r1	r1		

LR（0）分析

- LR（0）文法
  - 按上述算法构造的分析表，如果各表项均无多重定义，则称该文法为一个LR（0）文法
  - LR（0）文法的LR（0）FSM中，每个状态（闭包项目集）都满足：
    - 不同时含有移进项目和归约项目
    - 不含有两个以上归约项目

SLR（1）分析

✧ LR（0）分析的局限性

– 满足 LR（0）要求的文法不多

如：文法中含有产生式 $A \to \varepsilon$ 通常会遇到问题，对应的项目 $A \to .$ 是归约项目，容易引起移进–归约冲突

将会看到，对上述 LR（0）文法的例子作很小的扩充就会变成非 LR（0）文法

– 只根据栈顶的当前状态确定下一步动作

根据栈顶状态，就可确定进行移进还是归约：

- ACTION 表同一行中，不会既有移进又有归约；
- 同一行中，归约动作同时存在且都是一样的

SLR（1）分析

✧ 不是 LR（0）的文法举例

– 验证如下文法不是 LR（0）的

文法 $G[E]$:

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$

$G[E]$ 的增广文法 $G'[S]$:

(0) $S \to E$
(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$

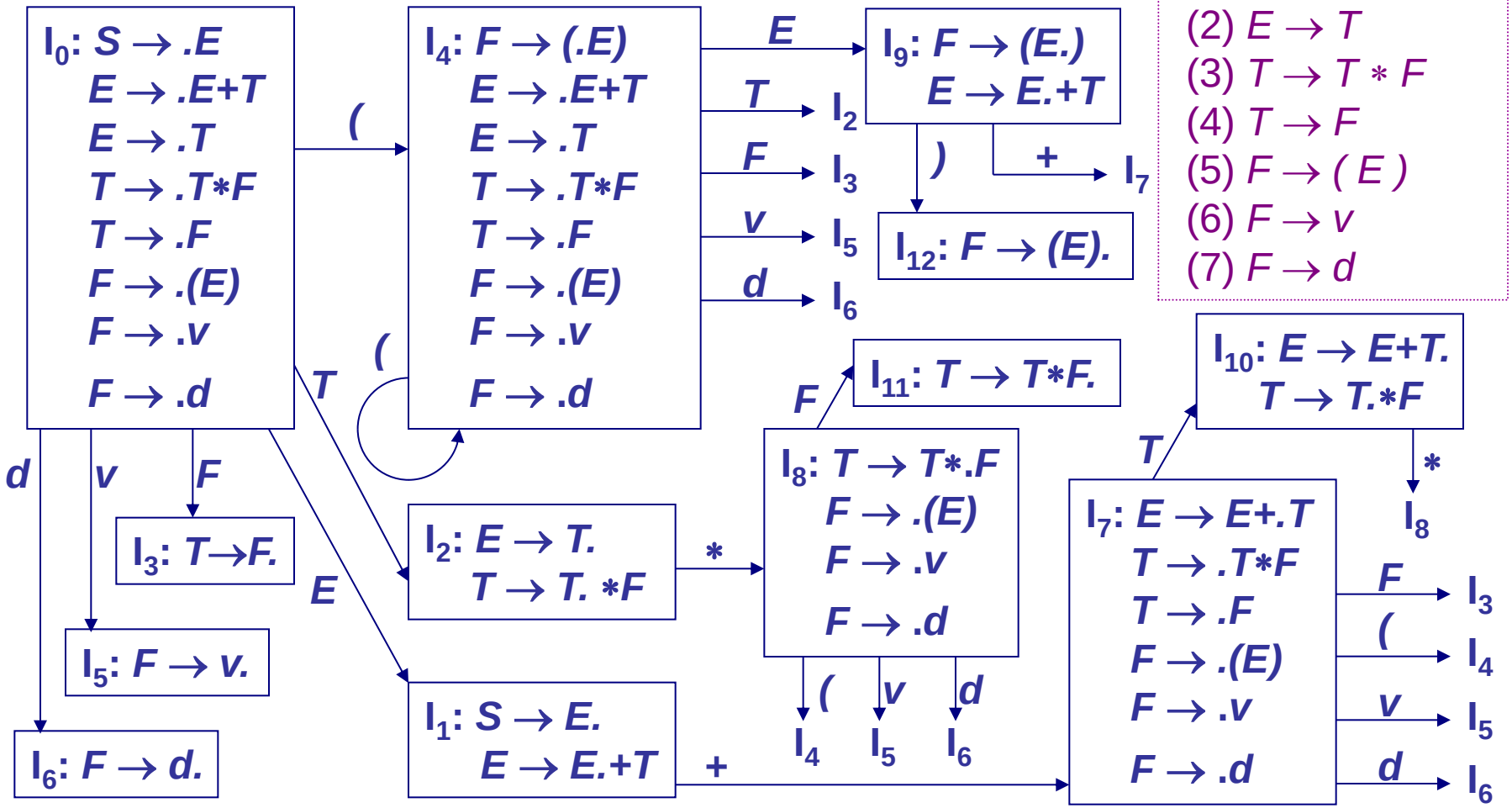
SLR（1）分析

✧ 验证文法G不是LR（0）文法

– 构造G[E]的增广文法G'[S]的LR（0）FSM

增广文法G'[S]:

(0) $S \to E$
(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$



SLR（1）分析

✧ 验证文法G不是LR（0）文法

– 从前一页的LR（0）FSM可以发现如下两个状态（项目集）存在移进–归约冲突

I_2: $E \to T .$
$T \to T . * F$

I_{10}: $E \to E + T .$
$T \to T . * F$

增广文法G'[S]:

(0) $S \to E$
(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$

– 注意：由于 $S \to E .$ 是接受项目，所以如下状态不存在冲突

I_1: $S \to E .$
$E \to E . + T$

SLR（1）分析

- 向前查看一个符号可解决冲突
  - 文法 G' 中， Follow(E) = { +,),# }

在如下存在移进–归约冲突的状态 I_2 和 I_{10} 中，可以根据下一个输入符号是否属于 Follow(E) 来决定是否进行归约，同时可以根据下一个输入符号是否为 * 来决定是否移进

I_2: $E \to T\ .$
$T \to T\ .\ * F$

I_{10}: $E \to E + T\ .$
$T \to T\ .\ * F$

增广文法$G'[S]$:

(0) $S \to E$
(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$

SLR（1）分析

✧ SLR（1）分析思想

- 向前查看一个符号来改进对LR（0）状态（项目集）中移进–归约和归约–归约冲突的解决
- 根据下一个输入符号是否属于要归约的非终结符的 Follow 集来决定是否进行归约
- 如果LR（0）状态（项目集）中的所有归约项中要归约的非终结符的 Follow 集互不相交，则可以解决归约–归约冲突
- 如果LR（0）状态（项目集）中的所有归约项中要归约的非终结符的 Follow 集与所有移进项目要移进的符号集互不相交，则可以解决移进–归约冲突

SLR（1）分析

- SLR（1）分析思想
  - SLR（1）分析表的构造也基于LR（0）FSM
  - 只需对 LR（0）分析表进行简单修改

    使得归约表项只适用于相应非终结符Follow 集中的输入符号

SLR（1）分析

✧ SLR（1）分析表的构造

– 假定$G[S]$的增广文法为$G'[S]$，其LR（0）FSM 的状态集为$\mathbf{C}=\{I_0, I_1, \ldots, I_n\}$；令状态$I_k$对应的 SLR（1）分析表的栈顶状态为k；并令含有项目$S'\rightarrow .S$ 的项目集为I_0，因此0为初态. ACTION表项和GOTO表项可按如下方法构造:

- 若项目$A\rightarrow \boldsymbol{\alpha}.a\boldsymbol{\beta}$属于 I_k 且 GO (I_k, a)= I_j, a 为终结符，则置ACTION[k, a] 为“把状态j和符号a移进栈”，简记为“sj”;
- 若项目$A\rightarrow \boldsymbol{\alpha}.$ 属于I_k，那么，对任何$a\in$Follow(A)，置ACTION[k, a]为“用产生式$A\rightarrow\boldsymbol{\alpha}$进行归约”，简记为“rj”;其中，假定$A\rightarrow\boldsymbol{\alpha}$为文法$G'$的第j个产生式;
- 若项目$S'\rightarrow S.$ 属于I_k，则置ACTION[k, #]为“接受”，简记为“acc”;
- 若GO (I_k, A)= I_j，A为非终结符，则置GOTO(k, A)=j;
- 分析表中凡不能用上述规则填入信息的空白格均置上“出错标志”

SLR（1）分析

✧ SLR（1）分析表的构造举例

– 增广文法：$G'[S]$

（0）$S \to E$ （1）$E \to E+T$ （2）$E \to T$
（3）$T \to T*F$ （4）$T \to F$
（5）$F \to (E)$ （6）$F \to v$ （7）$F \to d$

栈顶状态	ACTION							GOTO		
	v	*d*	*	+	(	)	#	*E*	*T*	*F*
0	s5	s6			s4			1	2	3
1				s7			acc			
2			s8	r2		r2	r2			
3			r4	r4		r4	r4			
4	s5	s6			s4			9	2	3
5			r6	r6		r6	r6			
6			r7	r7		r7	r7			
7	s5	s6			s4				10	3
8	s5	s6			s4					11
9				s7		s12				
10			s8	r1		r1	r1			
11			r3	r3		r3	r3			
12			r5	r5		r5	r5			

SLR（1）分析

✧ SLR（1）文法

- 按上述算法构造的分析表，如果各表项均无多重定义，则称该文法为一个SLR（1）文法
- SLR（1）文法的LR（0）FSM中，每个状态都满足：
  - 对该状态的任何项目A→u.av（a为终结符），不存在项目 B→w. 使得 a∈Follow(B)
  - 对该状态的任何两个项目A→u.和B→v.，满足
    $Follow(A) \cap Follow(B) = \Phi$

SLR（1）分析

- 比较 LR（0）表和 SLR（1）表
  - 在 LR（0）表的 ACTION 表中，归约表项总是整行出现的，即一个归约对于所有输入符号都适用；不会既有移进又有归约
  - 而在 SLR（1）表的ACTION 表中。归约表项只适用于相应非终结符Follow 集中的输入符号；可以既有移进又有归约

LR（1）分析

✧ SLR（1）分析的局限性

- 只考虑到所归约非终结符的 Follow 符号

  虽然是向前查看一个输入符号，但只要输入符号属于所归约非终结符的 Follow 集合，就可进行归约

- 未考虑非终结符 Follow 集中的符号是否也是句柄的 Follow 符号

  一个输入符号属于所归约非终结符的 Follow 集合，未必就是句柄可以后跟的符号

LR（1）分析

✧ SLR（1）分析的局限性举例

– 验证如下文法不是 SLR（1）的

文法 $G[E]$:

(1) $E \to (L\ ,\ E\ )$
(2) $E \to F$
(3) $L \to L\ ,\ E$
(4) $L \to E$
(5) $F \to (\ F\ )$
(6) $F \to d$

$G[E]$ 的增广文法 $G'[S]$:

(0) $S \to E$
(1) $E \to (L\ ,\ E\ )$
(2) $E \to F$
(3) $L \to L\ ,\ E$
(4) $L \to E$
(5) $F \to (\ F\ )$
(6) $F \to d$

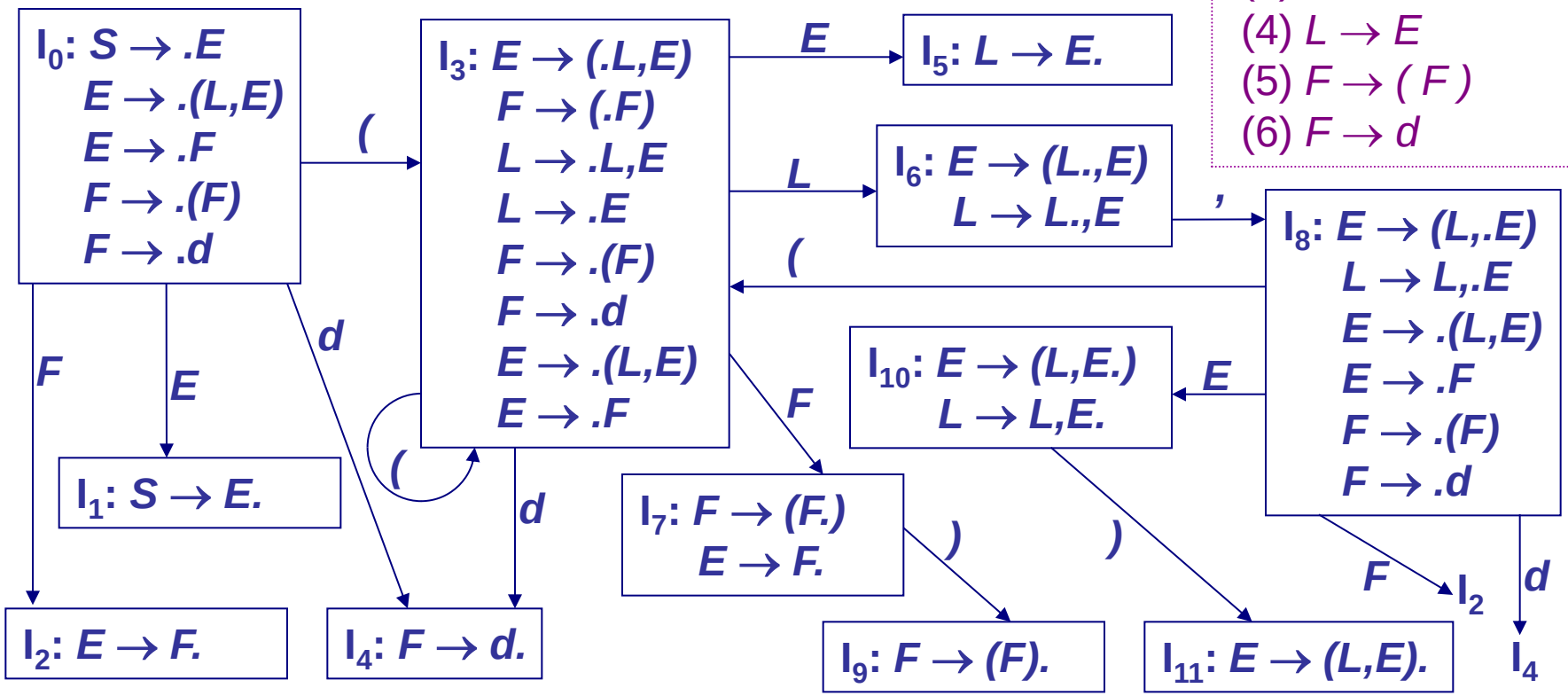
LR（1）分析

增广文法$G'[S]$:

(0) $S \to E$
(1) $E \to (L\ ,\ E\ )$
(2) $E \to F$
(3) $L \to L\ ,\ E$
(4) $L \to E$
(5) $F \to (\ F\ )$
(6) $F \to d$

✧ SLR（1）分析的局限性举例

– 构造文法G[S] 的一个LR（0）FSM



LR（1）分析

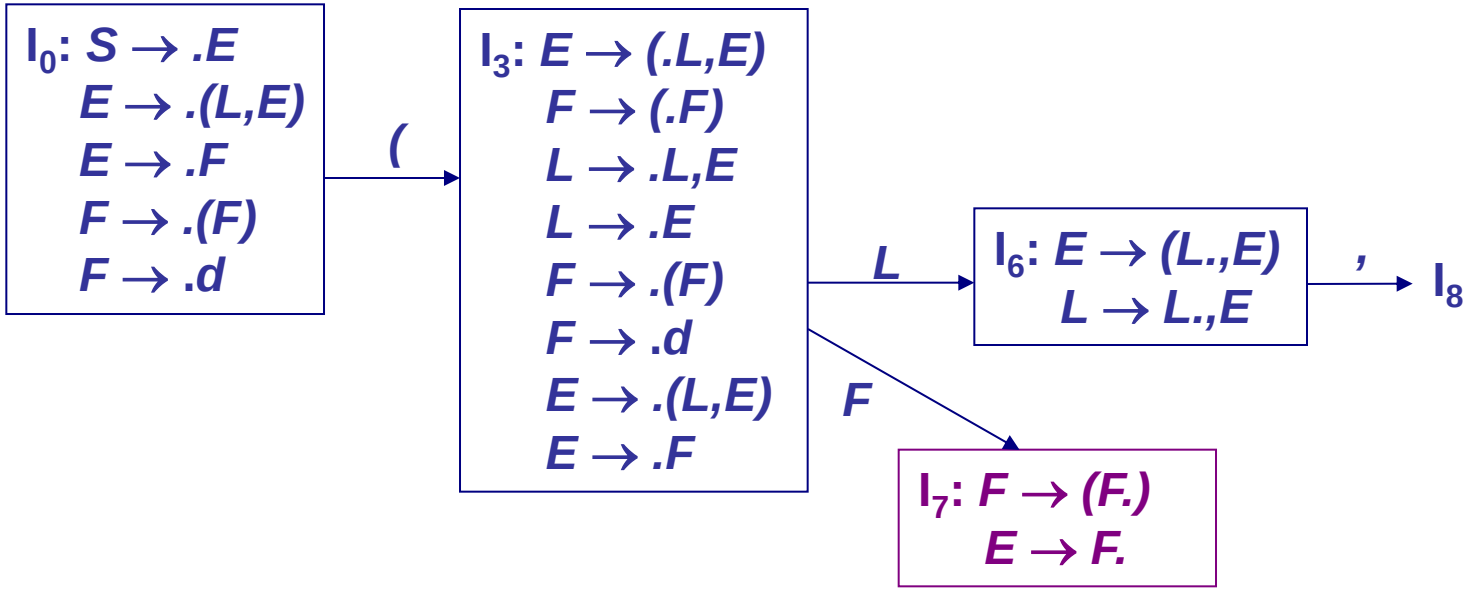
增广文法$G'[S]$:

(0) $S \to E$
(1) $E \to (L\ ,\ E\ )$
(2) $E \to F$
(3) $L \to L\ ,\ E$
(4) $L \to E$
(5) $F \to (\ F\ )$
(6) $F \to d$

✧ SLR（1）分析的局限性举例

– 文法$G[E]$ 不是SLR（1）文法

状态 I_7 的移进–归约冲突无法用 SLR（1）分析方法解决　注意：$) \in$ Follow(E)



LR（1）分析

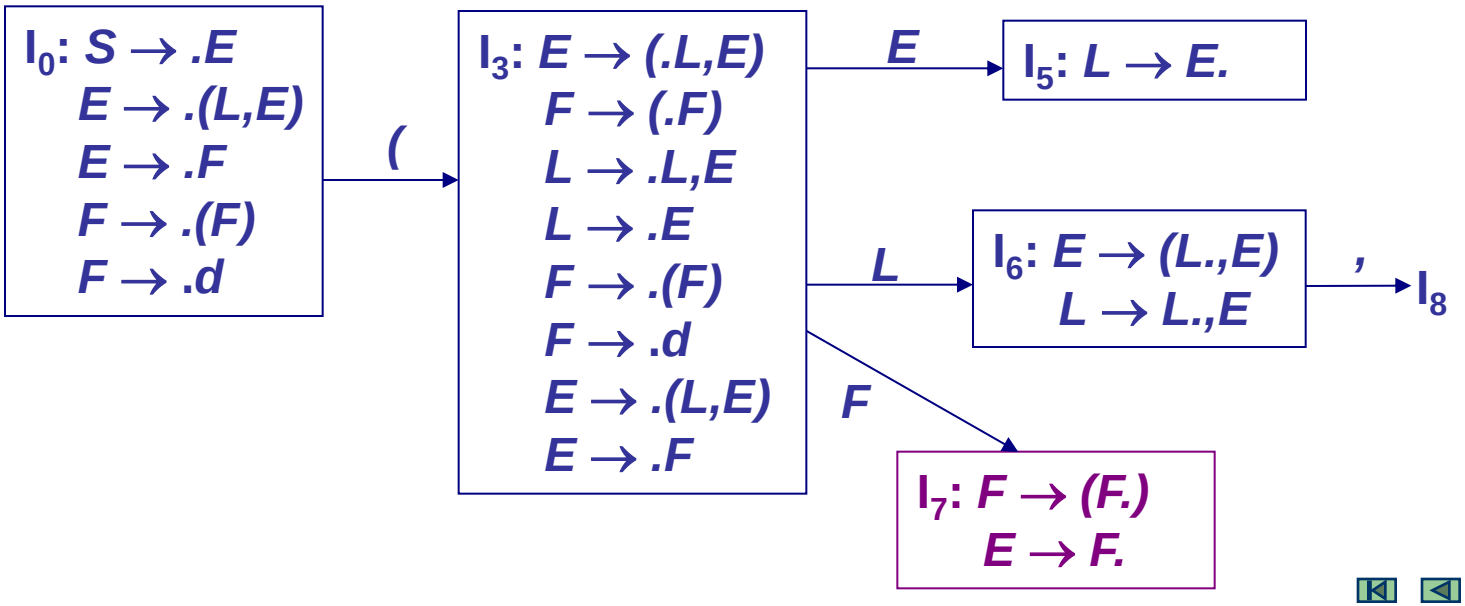
✧ SLR（1）分析的局限性举例

– 状态 I_7 的冲突是可以解决的

从 I_3，I_5和 I_6 容易看出，到达状态 I_7 时，句柄 ***F*** 所期望的下一个 输入符号实际上是，，而不是 ***)***

增广文法*G'*[*S*]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\,,E\,)$
(2) $E \rightarrow F$
(3) $L \rightarrow L\,,E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\,F\,)$
(6) $F \rightarrow d$



LR（1）分析

✧ LR（1）项目

– 在 LR（0）项目基础上增加一个终结符

所增加的终结符称为向前搜索符（*lookahead*）
表示产生式右端完整匹配后所允许在余留符号串中的下一个终结符

– LR（1）项目形如：

$$A \rightarrow \alpha . \beta , a$$

$A \rightarrow \alpha . \beta$ 同 LR（0）项目， a 为向前搜索符

这里，a 或为终结符，或为输入结束标志符 #

LR（1）分析

✧ LR（1）项目解析

– 对于形如：

$$A \rightarrow \alpha .,\ a$$

的 LR（1）项目，对应 LR（0）的归约项目，但只有当下一个输入符是 a 时才能进行归约

对于其它形式的LR（1）项目， a 只起到信息传承的作用

（参见随后的 LR（1） FSM 构造过程）

LR（1）分析

✧ 记号

– 若形如

$$A \to \alpha . \beta,\ a_1$$
$$A \to \alpha . \beta,\ a_2$$
$$\ldots$$
$$A \to \alpha . \beta,\ a_m$$

的 LR（1）项目序列需要出现在同一个项目集中时，将其简记为

$$A \to \alpha . \beta, a_1 / a_2 / \ldots / a_m$$

LR（1）分析

- LR（1）FSM
  - 类似于 LR（0）FSM，可以在 LR（1）项目的基础上为上下文无关文法 G 构造一个类似的有限状态机，称为 LR（1）FSM

LR（1）分析

✧ LR（1）FSM的构造

– LR（1）FSM的状态

- LR（1） FSM 的状态是 LR（1）项目集的闭包（*closure*）
- 计算LR（1）项目集 **I** 的闭包 CLOSURE**(I)**的算法：

```
function  CLOSURE(I)
{  J:= I;
   repeat  for  J 中的每个项目 [A→α . Bβ , a] 和产生式 B→ γ
           do 将所有形如 [B→ .γ , b] 的项目加到 J 中, 这里
              b ∈ First（βa）
   until  上一次循环不再有新项目加到 J 中
   return  J
};
```

LR（1）分析

✧ LR（1）FSM的构造

– LR（1）FSM的初态

设文法 $G[S]$ 的增广文法为 $G'[S']$，则 G'的 LR（1）FSM 的初态　$\mathbf{I_0}$ = CLOSURE({ **[S'→.S，#]** })

例 设右边文法$G[S]$的增广文法为 $G'[S]$，其 LR（1）FSM 的初态

$\mathbf{I_0}$：
$S \to .E\ ,\ \#$
$E \to .(L\ ,\ E)\ ,\ \#$
$E \to .F\ ,\ \#$
$F \to .(\ F\ )\ ,\ \#$
$F \to .\ d\ ,\ \#$

文法 $G[E]$:

(1) $E \to (L\ ,\ E\ )$
(2) $E \to F$
(3) $L \to L\ ,\ E$
(4) $L \to E$
(5) $F \to (\ F\ )$
(6) $F \to d$

LR（1）分析

✧ LR（1）FSM的构造

– LR（1）FSM的状态转移函数

GO (I,X) = CLOSURE(J)

其中，**I**为LR（1）FSM的状态（闭包的项目集），**X** 为文法符号，**J={ [A→αX . β , a] | [A→α . Xβ , a] ∈ I}**

– 从 LR（1）FSM 的初态出发，应用上述转移函数，可逐步构造出完整的 LR（1）FSM

LR（1）分析

✧ LR（1）FSM的构造

– 计算 LR（1）项目集规范族

设文法 *G*[*S*] 的增广文法为 *G'* [*S'*], 则 *G'* 的LR（1）项目集规范族 **C** 可由如下算法计算：

C:= { CLOSURE ({ [S'→**.**S, #] }) }

Repeat

For **C** 中每一项目集 **I** 和每一文法符号**X**

Do if **GO(I,X)** 非空且不属于 **C**

Then 把 **GO(I,X)** 放入 **C** 中

Until **C** 不再增大

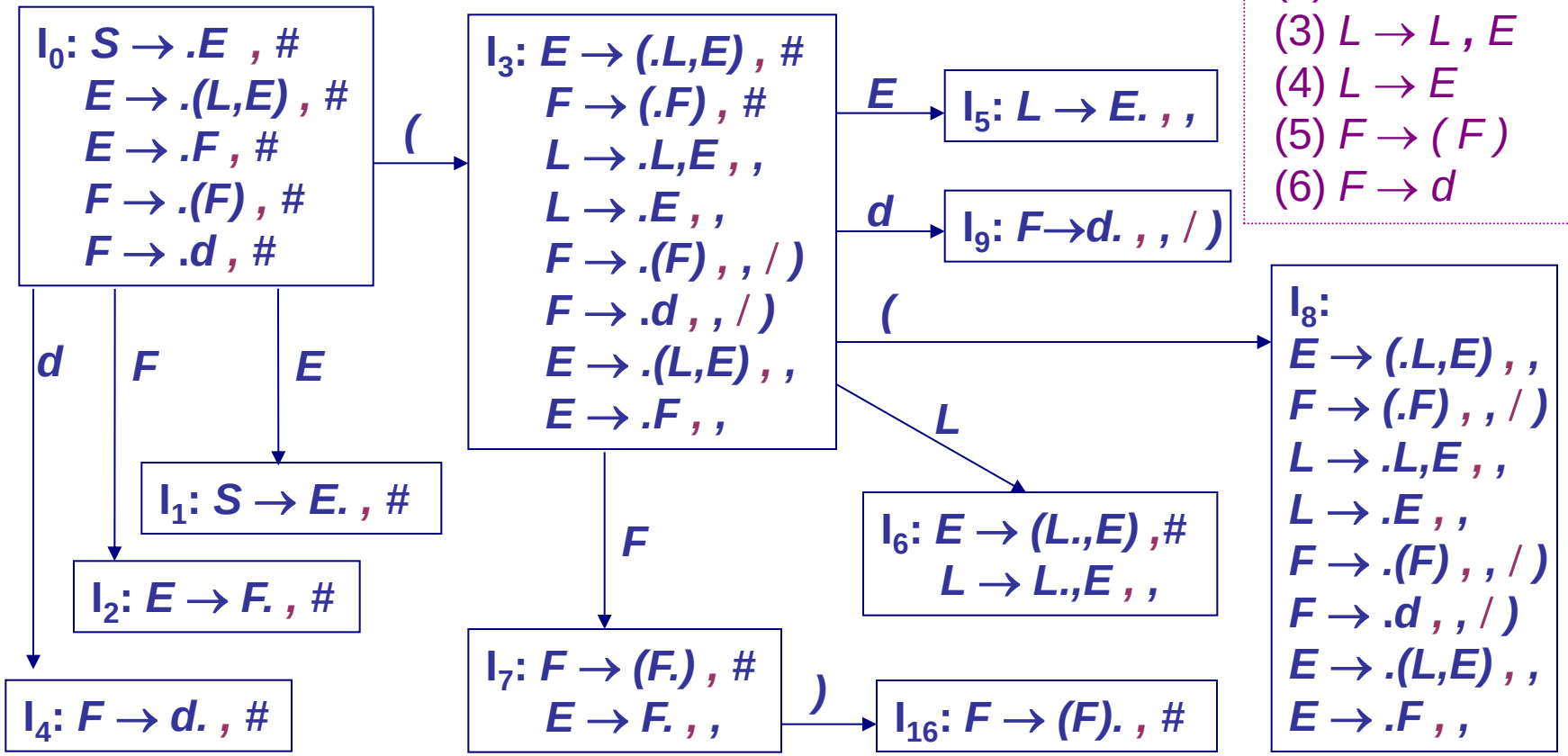
LR（1）分析

✧ LR（1）FSM的构造举例

– 构造文法G[E] 的 LR（1）FSM

增广文法G'[S]:

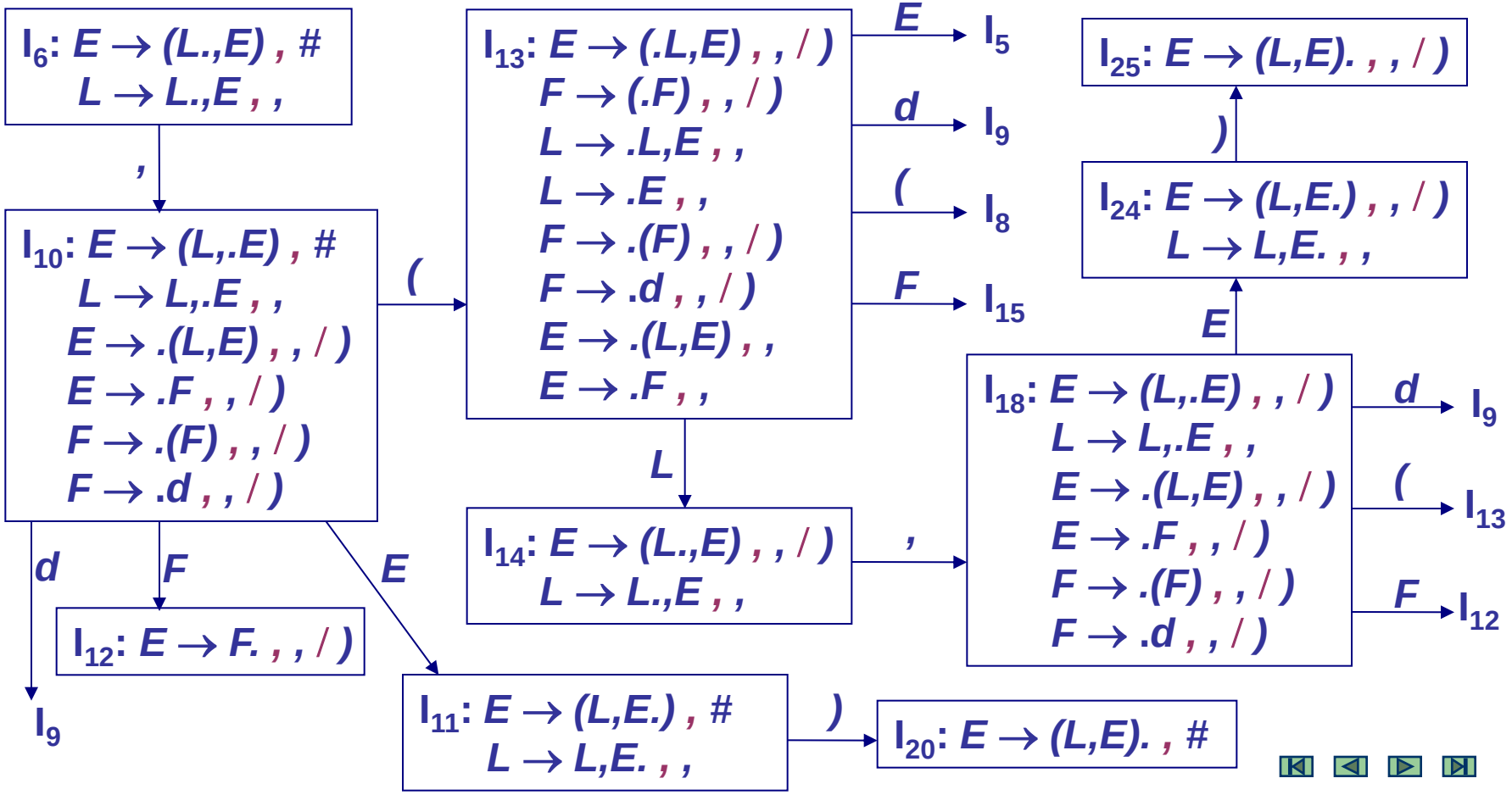
(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$



LR（1）分析

✧ LR（1）FSM的构造举例

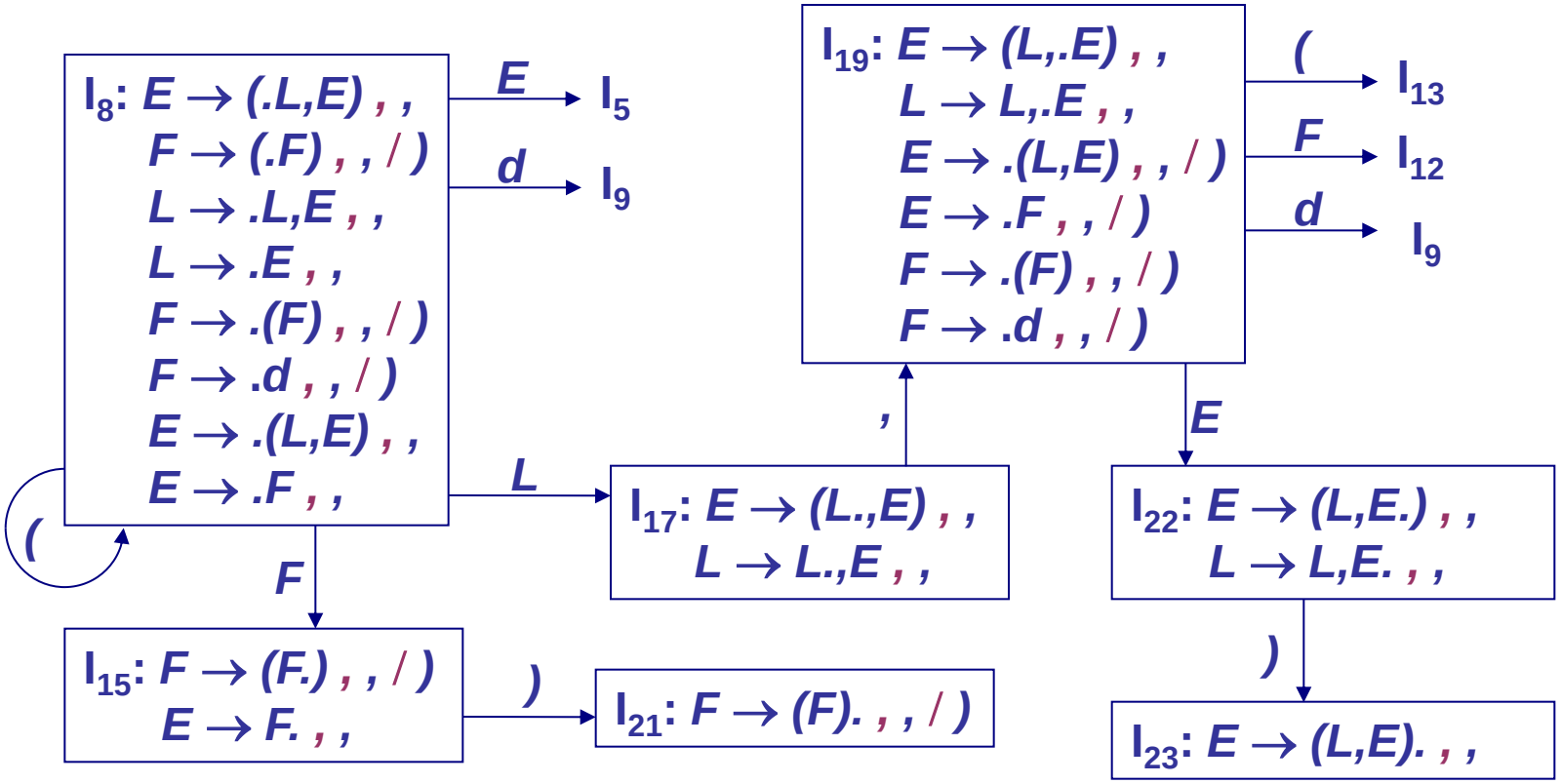
– 构造文法G[E] 的 LR（1）FSM



LR（1）分析

✧ LR（1）FSM的构造举例

– 构造文法*G*[*E*] 的 LR（1）FSM



LR（1）分析

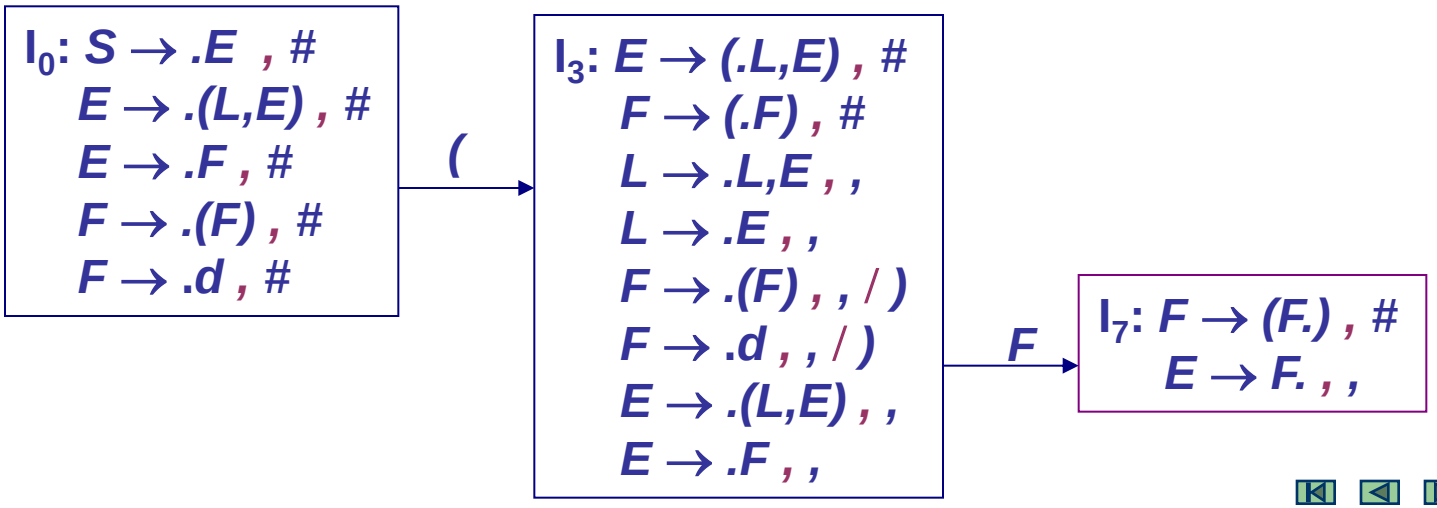
✧ 解决前例SLR（1）分析中的冲突

– 当到达状态 I_7（栈上的活前缀为**(F**）时，句柄 **F** 所期望的下一个输入符号只有**,**，没有 **)**，因而该状态下不存在移进–归约冲突

– 可以验证，对本例中*G'*[S] 的 LR（1），任何状态都不存在（移进–归约或归约–归约）冲突

增广文法*G'*[S]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$



LR（1）分析

✧ LR（1）分析表的构造

– 假定**C**={I_0, I_1,…，I_n}，令状态I_k对应的 LR（1）分析表的栈顶状态为k；令含有项目[*S*'→.*S*，#]的状态为I_0，因此 0 为初态。ACTION 表项和 GOTO 表项可按如下方法构造：

- 若项目[A→**α**.a**β**, b]属于 I_k 且 GO (I_k, a)= I_j, a 为终结符，则置 ACTION[k, a] 为“把状态j和符号a移进栈”，简记为“sj”
- 若项目[A→**α**. , b] 属于I_k，那么置ACTION[k, b]为“用产生式A→**α**进行归约”，简记为“rj”；这里，假定A→**α**为文法*G*'的第j个产生式
- 若项目[*S*'→*S*., #] 属于I_k，则置ACTION[k, #]为“接受”，记为“acc”
- 若GO (I_k, A)= I_j，A为非终结符，则置GOTO(k, A)=j;
- 分析表中凡不能用上述规则填入信息的空白格均置上“出错标志”

LR（1）分析

✧ LR（1）分析表的构造举例

– 增广文法：（0）$S \to E$　（1）$E \to (L,E)$　（2）$E \to F$
$G'[S]$　（3）$L \to L,E$　（4）$L \to E$　（5）$F \to (F)$　（6）$F \to d$

栈顶状态	ACTION					GOTO		
	d	,	(	)	#	E	L	F
0	s4		s3			1		2
1					acc			
2					r2			
3	s9		s8			5	6	7
4					r6			
5		r4						
6		s10						
7		r2		s16				
8	s9		s8			5	17	15
9		r6		r6				
10	s9		s13			11		12
11		r3		s20				
12		r2		r2				

LR（1）分析

✧ LR（1）分析表的构造举例（续）

– 增广文法：（0）$S \to E$　（1）$E \to (L,E)$　（2）$E \to F$
$G'[S]$　（3）$L \to L,E$　（4）$L \to E$　（5）$F \to (F)$　（6）$F \to d$

栈顶状态	ACTION					GOTO		
	d	,	(	)	#	E	L	F
13	s9		s8			5	14	15
14		s18						
15		r2		s21				
16					r5			
17		s19						
18	s9		s13			24		12
19	s9		s13			22		12
20					r1			
21		r5		r5				
22		r3		s23				
23		r1						
24		r3		s25				
25		r1		r1				

LR（1）分析

✧ LR（1）文法

- 按上述算法构造的分析表，如果各表项均无多重定义，则称该文法为一个LR（1）文法
- LR（1）文法的LR（1）FSM中，每个状态都满足：
  - 如果该状态含有项目 [A → u.av , b] ，a是终结符，那么就不会有项目[B → w., a] ；反之亦然
  - 该状态里所有归约项目的向前搜索符不相交，即不能同时含有项目[A → u., a] 和 [B → v., a]

LR（1）分析

- 可推广到 LR（k）分析
  - LR（k）项目

    形如：$[A \to \alpha . \beta, a_1 a_2 \ldots a_k]$,其中$a_1 a_2 \ldots a_k$为向前搜索符号串

    移进项目形如：$[A \to \alpha . a\beta, a_1 a_2 \ldots a_k]$
    待约项目形如：$[A \to \alpha . B\beta, a_1 a_2 \ldots a_k]$
    归约项目形如：$[A \to \alpha., a_1 a_2 \ldots a_k]$
  - 只有理论意义（LR（1）FSM 的状态数已经太大，k>1的情形难以想象）
  - 对任意k>0, 可证明LR（k）的语言类是相同的

LALR（1）分析

- LR（1）和SLR（1）分析技术的折衷
  - LR（1）分析比SLR（1）分析强大

    可以采用SLR（1）分析的文法一定可以采用LR（1）分析；但反之不成立
  - LR（1）分析的复杂度比SLR（1）分析高

    通常，一个SLR（1）文法的LR（0）FSM状态数目要比它的LR（1）FSM状态数目少得多
  - 合并LR（1）FSM的相似状态

    LALR（1）分析中将同芯的LR（1）FSM状态合并，得到与LR（0）FSM相同数目的状态，但保留了LR（1）的部分向前搜索能力

LALR（1）分析

- LR（1）FSM的同芯状态
  - LR（1）项目的“芯”（*core*）

    LR（1）项目 [**A** → α . β , **a**] 中的 **A** → α . β 部分称为该项目的“芯”
  - 同芯状态

    对于LR（1）FSM 的两个状态，如果只考虑每个项目的“芯”，它们是完全相同的项目集合，那么这两个状态就是同芯的状态

LALR（1）分析

✧ LR（1）FSM 同芯状态举例

– 文法*G'*[*S*] 的 LR（1）FSM中的同芯状态（共 9 组）

增广文法*G'*[*S*]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$

I_2: $E \rightarrow F.\ ,\ \#$

I_{12}: $E \rightarrow F.\ ,\ ,\ /\ )$

I_4: $F \rightarrow d.\ ,\ \#$

I_9: $F \rightarrow d.\ ,\ ,\ /\ )$

I_3: $E \rightarrow (.L,E)\ ,\ \#$
$F \rightarrow (.F)\ ,\ \#$
$L \rightarrow .L,E\ ,\ ,$
$L \rightarrow .E\ ,\ ,$
$F \rightarrow .(F)\ ,\ ,\ /\ )$
$F \rightarrow .d\ ,\ ,\ /\ )$
$E \rightarrow .(L,E)\ ,\ ,$
$E \rightarrow .F\ ,\ ,$

I_8: $E \rightarrow (.L,E)\ ,\ ,$
$F \rightarrow (.F)\ ,\ ,\ /\ )$
$L \rightarrow .L,E\ ,\ ,$
$L \rightarrow .E\ ,\ ,$
$F \rightarrow .(F)\ ,\ ,\ /\ )$
$F \rightarrow .d\ ,\ ,\ /\ )$
$E \rightarrow .(L,E)\ ,\ ,$
$E \rightarrow .F\ ,\ ,$

I_{13}: $E \rightarrow (.L,E)\ ,\ ,\ /\ )$
$F \rightarrow (.F)\ ,\ ,\ /\ )$
$L \rightarrow .L,E\ ,\ ,$
$L \rightarrow .E\ ,\ ,$
$F \rightarrow .(F)\ ,\ ,\ /\ )$
$F \rightarrow .d\ ,\ ,\ /\ )$
$E \rightarrow .(L,E)\ ,\ ,$
$E \rightarrow .F\ ,\ ,$

LALR（1）分析

✧ LR（1）FSM 同芯状态举例

– 文法*G'*[*S*] 的 LR（1）FSM中的同芯状态（共 9 组）

I_6: $E \to (L.,E)$,#
$L \to L.,E$, ,

I_{14}: $E \to (L.,E)$, , /)
$L \to L.,E$, ,

I_{17}: $E \to (L.,E)$, ,
$L \to L.,E$, ,

I_7: $F \to (F.)$, #
$E \to F.$, ,

I_{15}: $F \to (F.)$, , /)
$E \to F.$, ,

I_{10}: $E \to (L,.E)$, #
$L \to L,.E$, ,
$E \to .(L,E)$, , /)
$E \to .F$, , /)
$F \to .(F)$, , /)
$F \to .d$, , /)

I_{18}: $E \to (L,.E)$, , /)
$L \to L,.E$, ,
$E \to .(L,E)$, , /)
$E \to .F$, , /)
$F \to .(F)$, , /)
$F \to .d$, , /)

I_{19}: $E \to (L,.E)$, ,
$L \to L,.E$, ,
$E \to .(L,E)$, , /)
$E \to .F$, , /)
$F \to .(F)$, , /)
$F \to .d$, , /)

LALR（1）分析

✧ LR（1）FSM 同芯状态举例

– 文法*G'*[S] 的 LR（1）FSM中的同芯状态（共 9 组）

I_{11}: $E \rightarrow (L,E.)\ ,\ \#$
$L \rightarrow L,E.\ ,\ ,$

I_{22}: $E \rightarrow (L,E.)\ ,\ ,$
$L \rightarrow L,E.\ ,\ ,$

I_{24}: $E \rightarrow (L,E.)\ ,\ ,\ /\ )$
$L \rightarrow L,E.\ ,\ ,$

I_{16}: $F \rightarrow (F).\ ,\ \#$

I_{21}: $F \rightarrow (F).\ ,\ ,\ /\ )$

I_{20}: $E \rightarrow (L,E).\ ,\ \#$

I_{23}: $E \rightarrow (L,E).\ ,\ ,$

I_{25}: $E \rightarrow (L,E).\ ,\ ,\ /\ )$

LALR（1）分析

✧ LR（1）FSM的同芯状态合并

- 同芯项目的搜索符号集合进行合并
- 举例 上例中的 9 组同芯状态合并为9个新状态

$I_{2\text{-}12}$
$E \to F.\ ,\ ,\ /\ )\ /\ \#$

$I_{4\text{-}9}$
$F \to d.\ ,\ ,\ /\ )\ /\ \#$

$I_{10\text{-}18\text{-}19}$
$E \to (L,.E)\ ,\ ,\ /\ )\ /\ \#$
$L \to L,.E\ ,\ ,$
$E \to .(L,E)\ ,\ ,\ /\ )$
$E \to .F\ ,\ ,\ /\ )$
$F \to .(F)\ ,\ ,\ /\ )$
$F \to .d\ ,\ ,\ /\ )$

$I_{6\text{-}14\text{-}17}$
$E \to (L.,E)\ ,\ ,\ /\ )\ /\ \#$
$L \to L.,E\ ,\ ,$

$I_{3\text{-}8\text{-}13}$
$E \to (.L,E)\ ,\ ,\ /\ )\ /\ \#$
$F \to (.F)\ ,\ ,\ /\ )\ /\ \#$
$L \to .L,E\ ,\ ,$
$L \to .E\ ,\ ,$
$F \to .(F)\ ,\ ,\ /\ )$
$F \to .d\ ,\ ,\ /\ )$
$E \to .(L,E)\ ,\ ,$
$E \to .F\ ,\ ,$

$I_{7\text{-}15}$
$F \to (F.)\ ,\ ,\ /\ )\ /\ \#$
$E \to F.\ ,\ ,$

$I_{16\text{-}21}$
$F \to (F).\ ,\ ,\ /\ )\ /\ \#$

$I_{11\text{-}22\text{-}24}$
$E \to (L,E.)\ ,\ ,\ /\ )\ /\ \#$
$L \to L,E.\ ,\ ,$

$I_{20\text{-}23\text{-}25}$
$E \to (L,E).\ ,\ ,\ /\ )\ /\ \#$

LALR（1）分析

✧ LALR（1）文法

- 一个LR（1）文法，如果将其LR（1）FSM的同芯状态合并后所得到的新状态无归约–归约冲突，则该文法是一个LALR（1）文法
- 注：
  - 由于是LR（1）文法，所以未合并的状态无冲突
  - 合并同芯状态后，不会产生新的移进–归约冲突（分析一下为什么？）
  - 合并同芯状态后，可能产生新的归约–归约冲突

LALR（1）分析

✧ LALR（1）FSM的构造

– “brute-force”方法

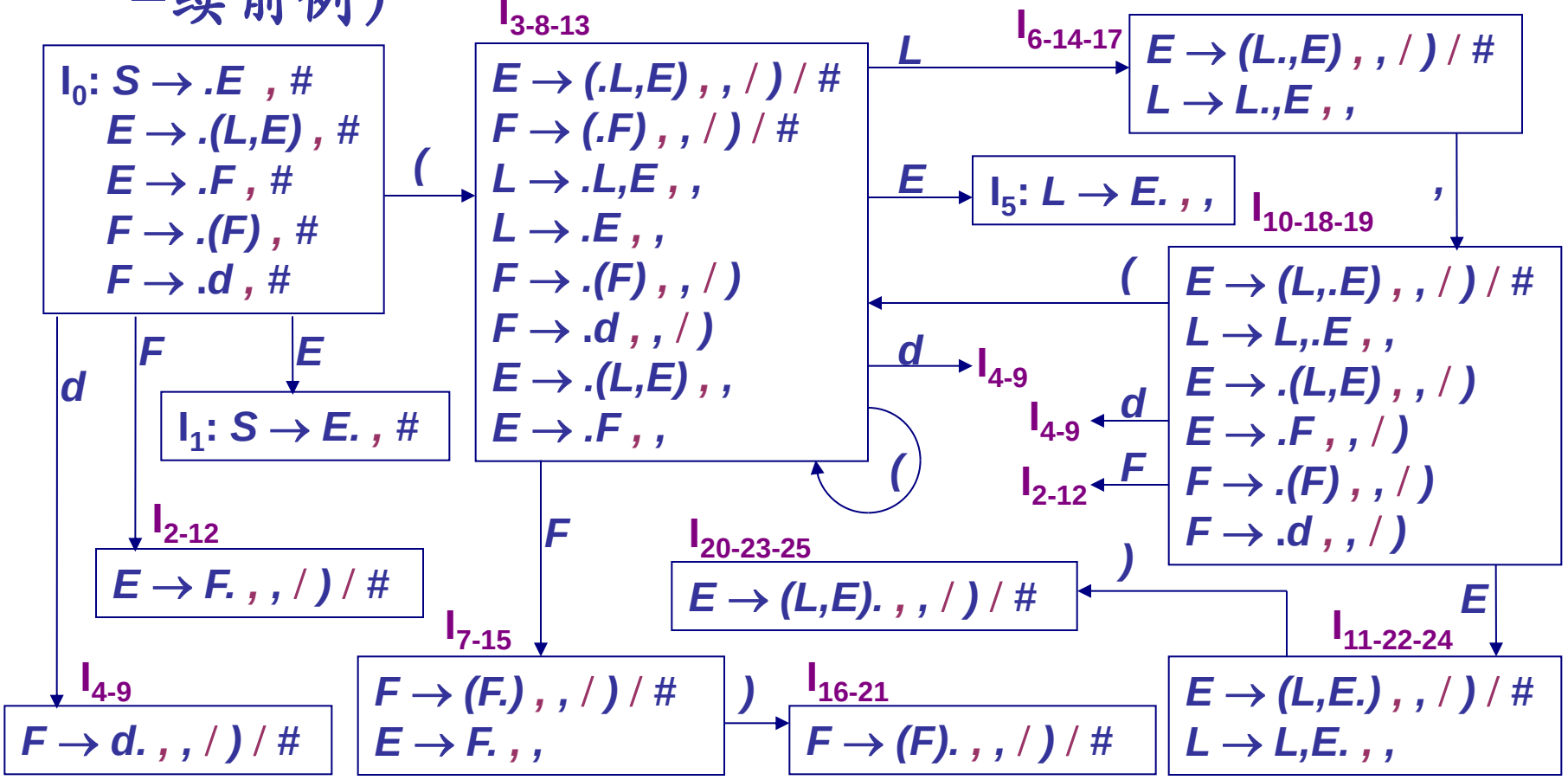
- 构造增广文法的 LR（1）FSM 状态
- 合并“同芯”的状态（同芯项目的搜索符集合相并）得到LALR(1) FSM 的状态
- LALR(1) FSM 状态由 GO 函数得到的后继状态是所有被合并的“同芯”状态的后继状态之并

  请思考：若两个状态“同芯”，那么其原来的后继状态也一定是“同芯”的

LALR（1）分析

✧ LALR（1）FSM的构造

– “brute-force”方法举例（–续前例）



LALR（1）分析

- LALR（1）分析表
  - 构造方法同LR（1）分析表

LALR（1）分析

✧ LALR（1）分析表的构造举例

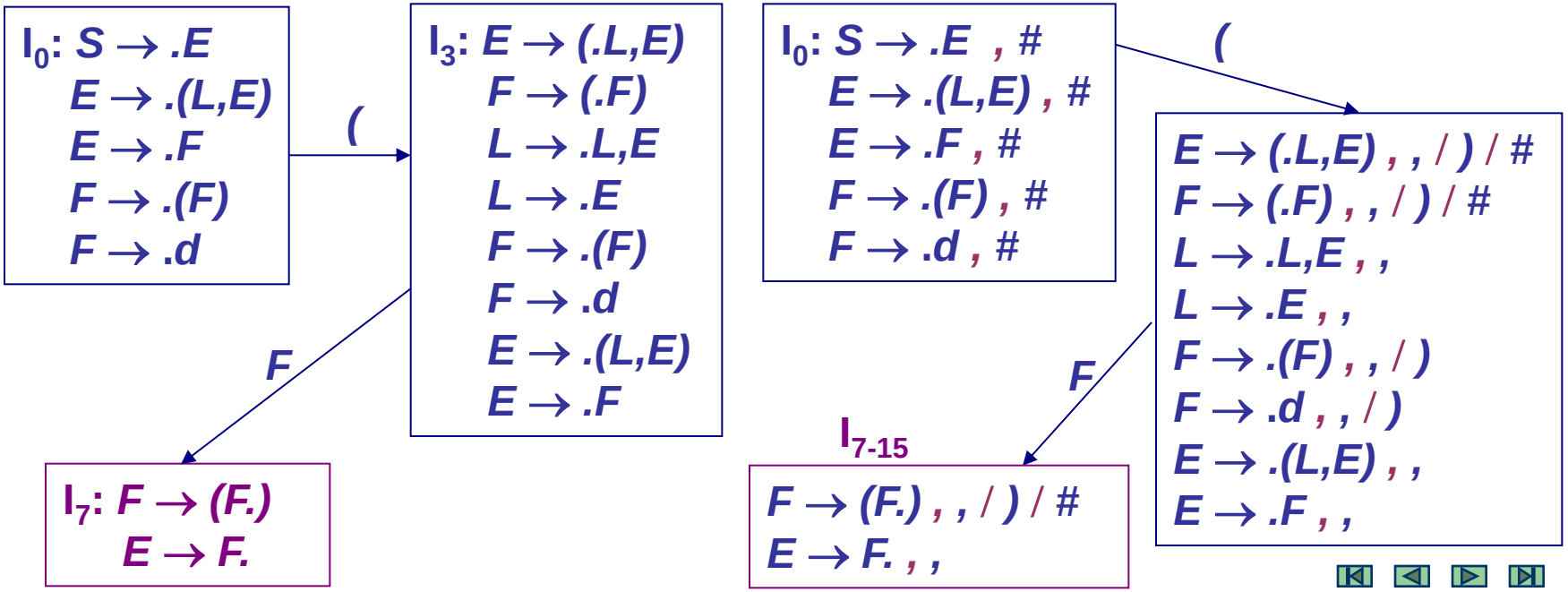
– 增广文法：（0）$S \to E$　（1）$E \to (L,E)$　（2）$E \to F$
$G'[S]$　（3）$L \to L,E$　（4）$L \to E$　（5）$F \to (F)$　（6）$F \to d$

栈顶状态	ACTION					GOTO		
	d	,	(	)	#	E	L	F
0	s4-9		s3-8-13			1		2-12
1					acc			
2-12		r2		r2	r2			
3-8-13	s4-9		s3-8-13			5	6-14-17	7-15
4-9		r6		r6	r6			
5		r4						
6-14-17		s10-18-19						
7-15		r2		s16-21				
10-18-19	s4-9		s3-8-13			11-22-24		2-12
11-22-24		r3		s20-23-25				
16-21		r5		r5	r5			
20-23-25		r1		r1	r1			

LALR（1）分析

- 与SLR（1）分析相比
  - LALR（1）FSM 的状态数与 LR（0）FSM 相同
  - LALR（1）分析强于SLR（1）分析

比较如下 LR（0）FSM 和 LALR（1）FSM 的片断



二义文法在LR分析中的应用

- ✧ 某些二义文法可以构造出高效的LR 分析器
  - 二义性文法不是LR文法，但是对某些二义性文法，人为地给出合理的限定规则，可能构造出高效的LR分析器
  - 例：规定优先级和结合性可构造右边文法的 SLR（1）分析器
  - 规定最近嵌套匹配原则可构造如下文法的SLR（1）分析器:

    $S \rightarrow \underline{\text{if}}\ S\ \underline{\text{else}}\ S \mid \underline{\text{if}}\ S \mid a$

    （留作练习）

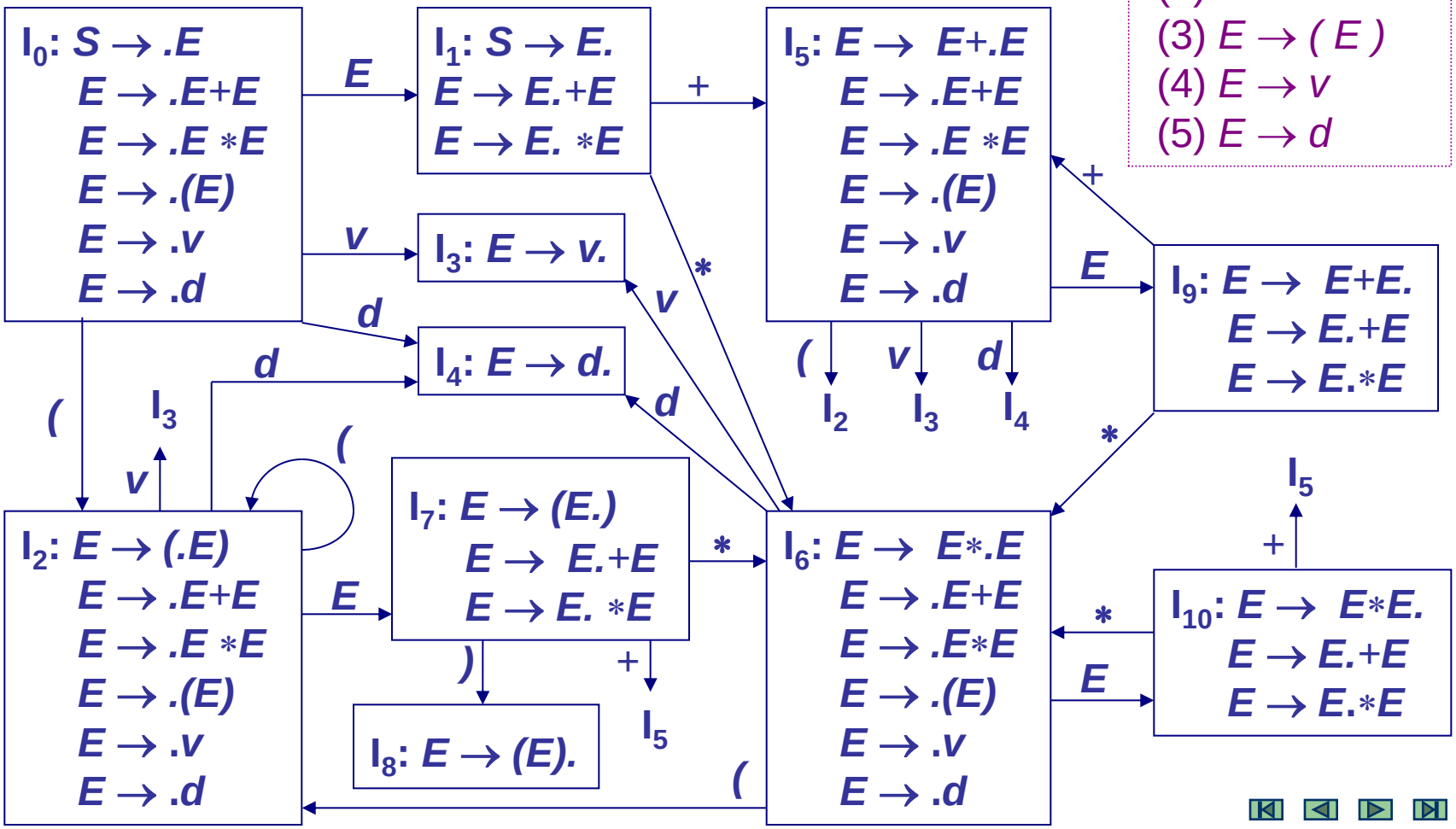
增广文法$G'[S]$:

(0) $S \rightarrow E$
(1) $E \rightarrow E + E$
(2) $E \rightarrow E * E$
(3) $E \rightarrow (\ E\ )$
(4) $E \rightarrow v$
(5) $E \rightarrow d$

二义文法在LR分析中的应用

✧ 例：对右边文法G'[S]，
先构造其 LR（0）FSM

(0) $S \to E$
(1) $E \to E + E$
(2) $E \to E * E$
(3) $E \to (E)$
(4) $E \to v$
(5) $E \to d$



二义文法在LR分析中的应用

✧ 例：右边文法$G'[S]$的LR（0）FSM中，
因为+, *∈Follow(E)={+, *,),#},
状态I_9和I_{10}存在移进–归约冲突，
所以，该文法不是SLR（1）文法

但如果规定 * 的优先级高于+，
* 和 +都服从左结合性,则可以解决I_9和I_{10}中的移进–归约冲突：

- 对于I_9

  若遇*，则移进；若遇+，则归约

- 对于I_{10}

  无论遇*，还是遇+，都归约

文法$G'[S]$
(0) $S \rightarrow E$
(1) $E \rightarrow E + E$
(2) $E \rightarrow E * E$
(3) $E \rightarrow (E)$
(4) $E \rightarrow v$
(5) $E \rightarrow d$

I_9: $E \rightarrow E+E.$
$E \rightarrow E.+E$
$E \rightarrow E.*E$

I_{10}: $E \rightarrow E*E.$
$E \rightarrow E.+E$
$E \rightarrow E.*E$

二义文法在LR分析中的应用

✧ 例：对右边文法G'[S]，从其LR（0）FSM和前述移进–归约冲突的解决方法，可构造该文法的LR分析表如下

(0) $S \rightarrow E$
(1) $E \rightarrow E + E$
(2) $E \rightarrow E * E$
(3) $E \rightarrow (E)$
(4) $E \rightarrow v$
(5) $E \rightarrow d$

栈顶状态	ACTION							GOTO
	v	*d*	*	+	(	)	#	*E*
0	s3	s4			s2			1
1			s6	s5			acc	
2	s3	s4			s2			7
3			r4	r4		r4	r4	
4			r5	r5		r5	r5	
5	s3	s4			s2			9
6	s3	s4			s2			10
7			s6	s5		s8		
8			r3	r3		r3	r3	
9			s6	r1		r1	r1	
10			r2	r2		r2	r2	

LR 分析中的出错处理

✧ 简单的LR 分析出错处理

- LR分析表的空表项对应一个出错位置
- 可根据相应的堆栈状态和输入符号设置报错信息，进行简单的恢复工作

LR 分析中的出错处理

✧ 简单的LR 分析出错处理举例

- 可能的报错信息　*e1*–缺少运算数
  e2–右括号未匹配　*e3*–缺少运算符　*e4*–缺少右括号
- 可能的恢复措施 *e1*? *e2*? *e3*? *e4*?

栈顶状态	ACTION							GOTO
	v	*d*	*	+	(	)	#	*E*
0	s3	s4	e1	e1	s2	e2	e1	1
1	e3	e3	s6	s5	e3	e2	acc	
2	s3	s4	e1	e1	s2	e2	e1	7
3	e3	e3	r4	r4	e3	r4	r4	
4	e3	e3	r5	r5	e3	r5	r5	
5	s3	s4	e1	e1	s2	e2	e1	9
6	s3	s4	e1	e1	s2	e2	e1	10
7	e3	e3	s6	s5	e3	s8	e4	
8	e3	e3	r3	r3	e3	r3	r3	
9	e3	e3	s6	r1	e3	r1	r1	
10	e3	e3	r2	r2	e3	r2	r2	

LR(K)文法（选讲）

✧ LR(K)文法的条件

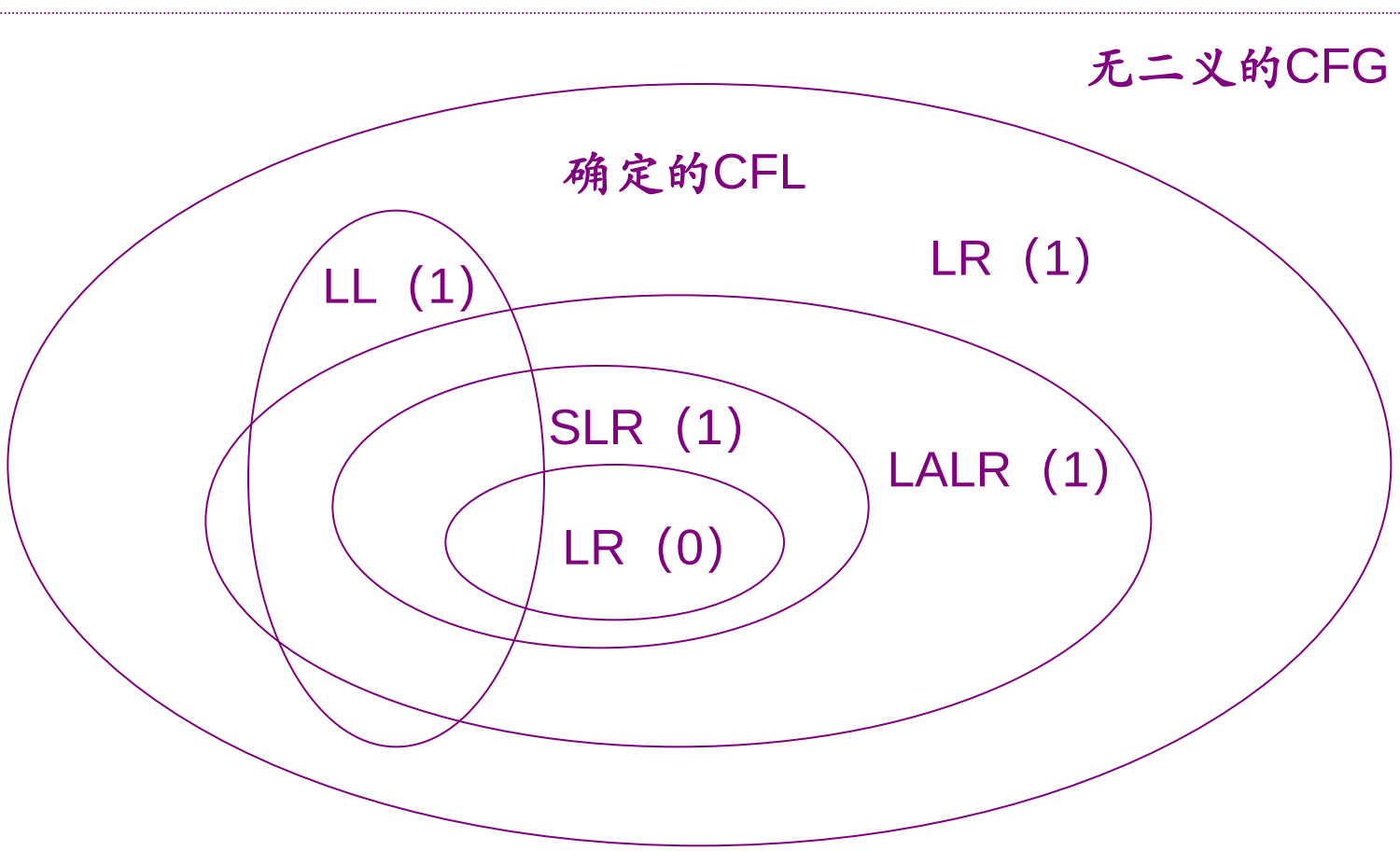
– 任何分析树对应的句柄，可由句型中该句柄左边的符号串以及其右边的k个终结符构成的串（不足k个时以输入结束符补足）唯一确定

LR(K)文法（选讲）

✧ 一些重要的结论

- 给定$k>0$, 某CFG是否为LR(k) 文法是可判定的
- 对于一个CFG，是否存在一个整数$k>0$使得该文法是LR(k) 文法，是不可判定的
- LR(k) 文法是无二义文法
- 如果G是一个LR(k)文法，且 L = L(G)，则一定存在某个DPDA，其语言为L；如果语言L是确定的（是某个DPDA的语言），则定有某一LR(1)文法G，L = L(G)
- 两个LR(k)文法的语言是否相等是可判定的
- 任何LL(k)文法都是LR(k)文法

几类分析文法之间的关系（这讲）



课后作业

参见网络学堂公告：“第二次书面作业”

That's all for today.

Thank You