Outline

- LR(1) 分析*
- LALR(1) 分析*
- 二义文法
- 语义分析
- 语法制导定义*

LR（1）分析

✧ LR（1）A的构造

– 计算 LR（1）项目集规范族

设文法 *G*[*S*] 的拓广文法为 *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

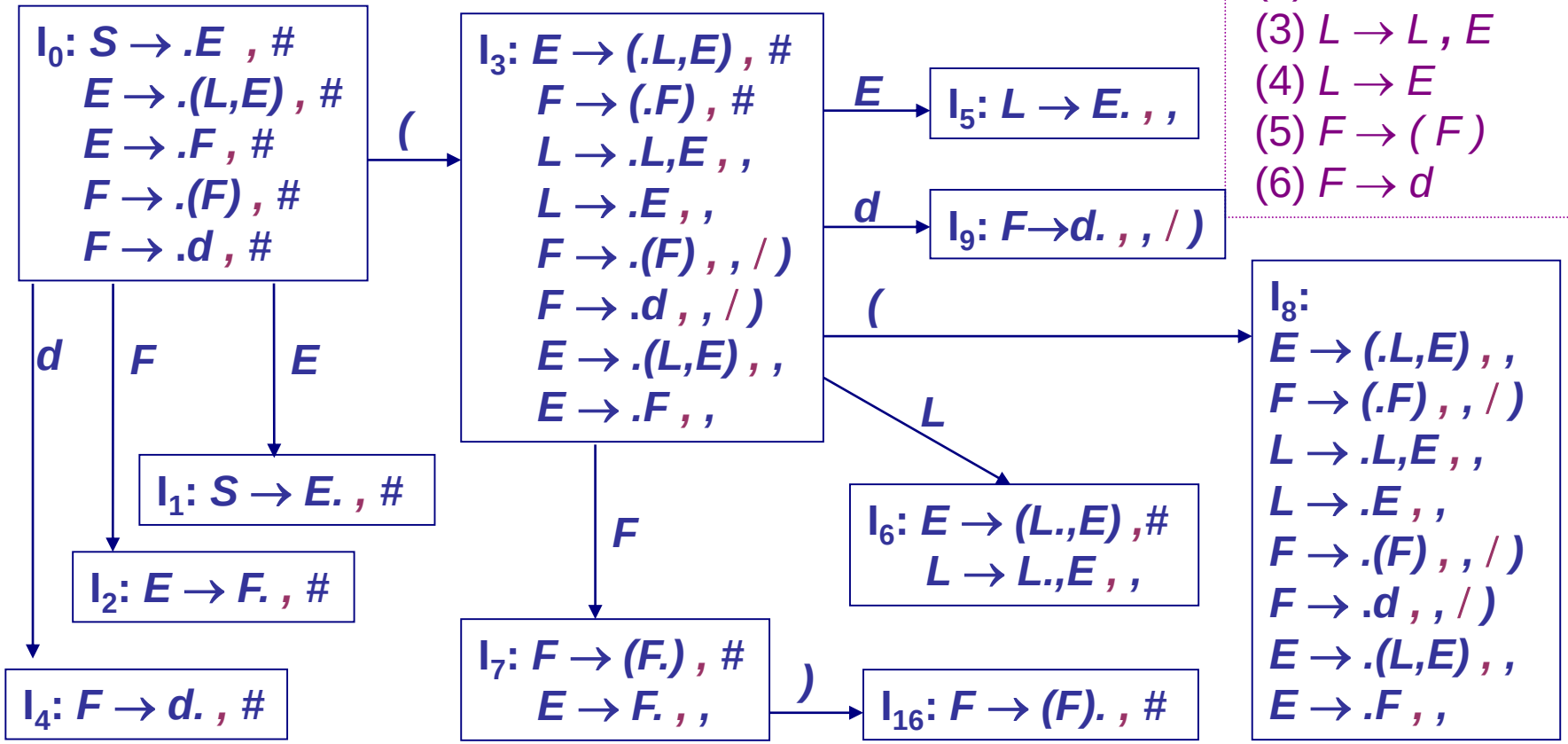
LR（1）分析

✧ LR（1）A的构造举例

– 构造拓广文法*G'*[S] 的 LR（1）A

拓广文法*G'*[S]:

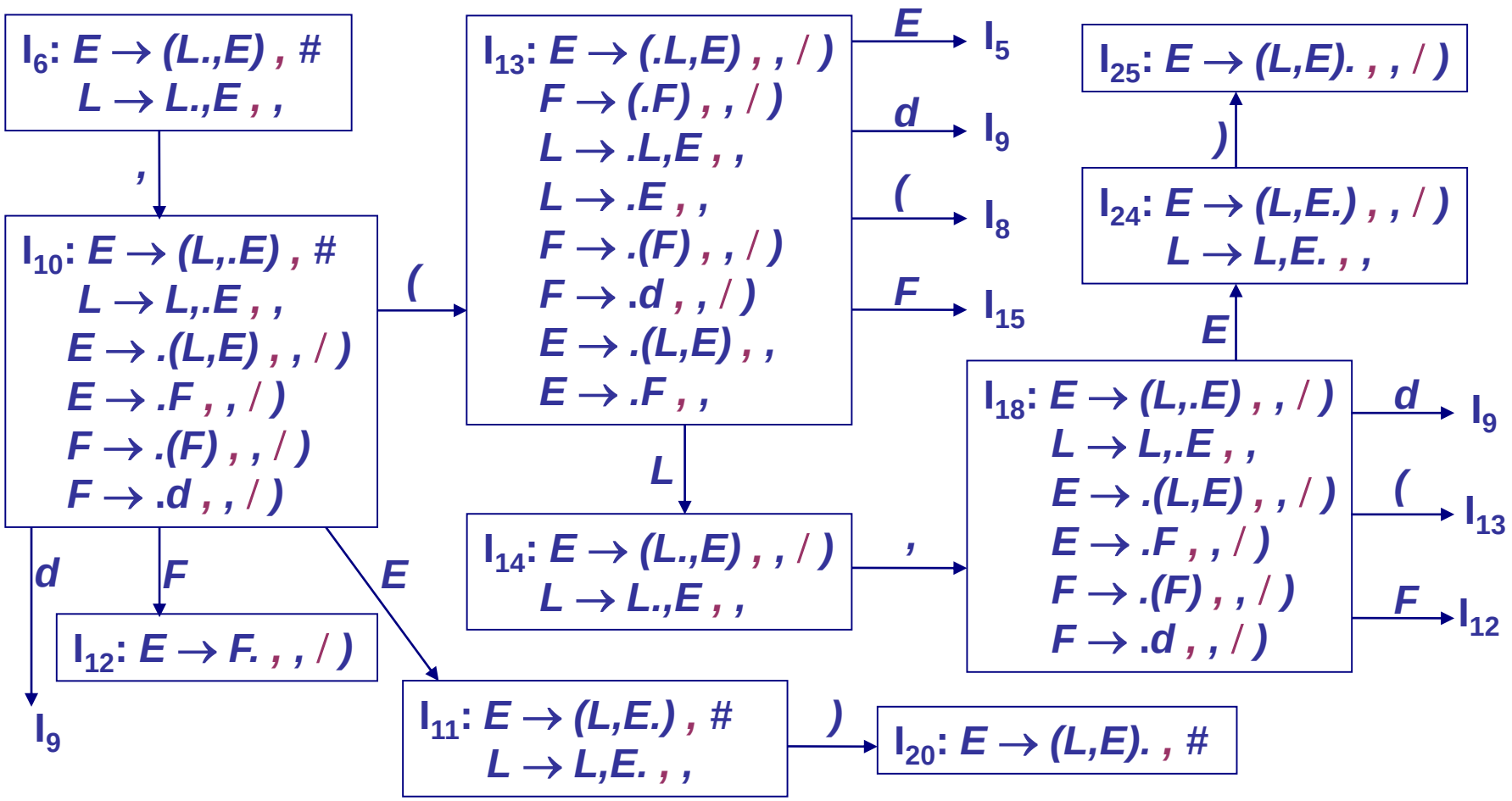
(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$



LR（1）分析

✧ LR（1）A的构造举例

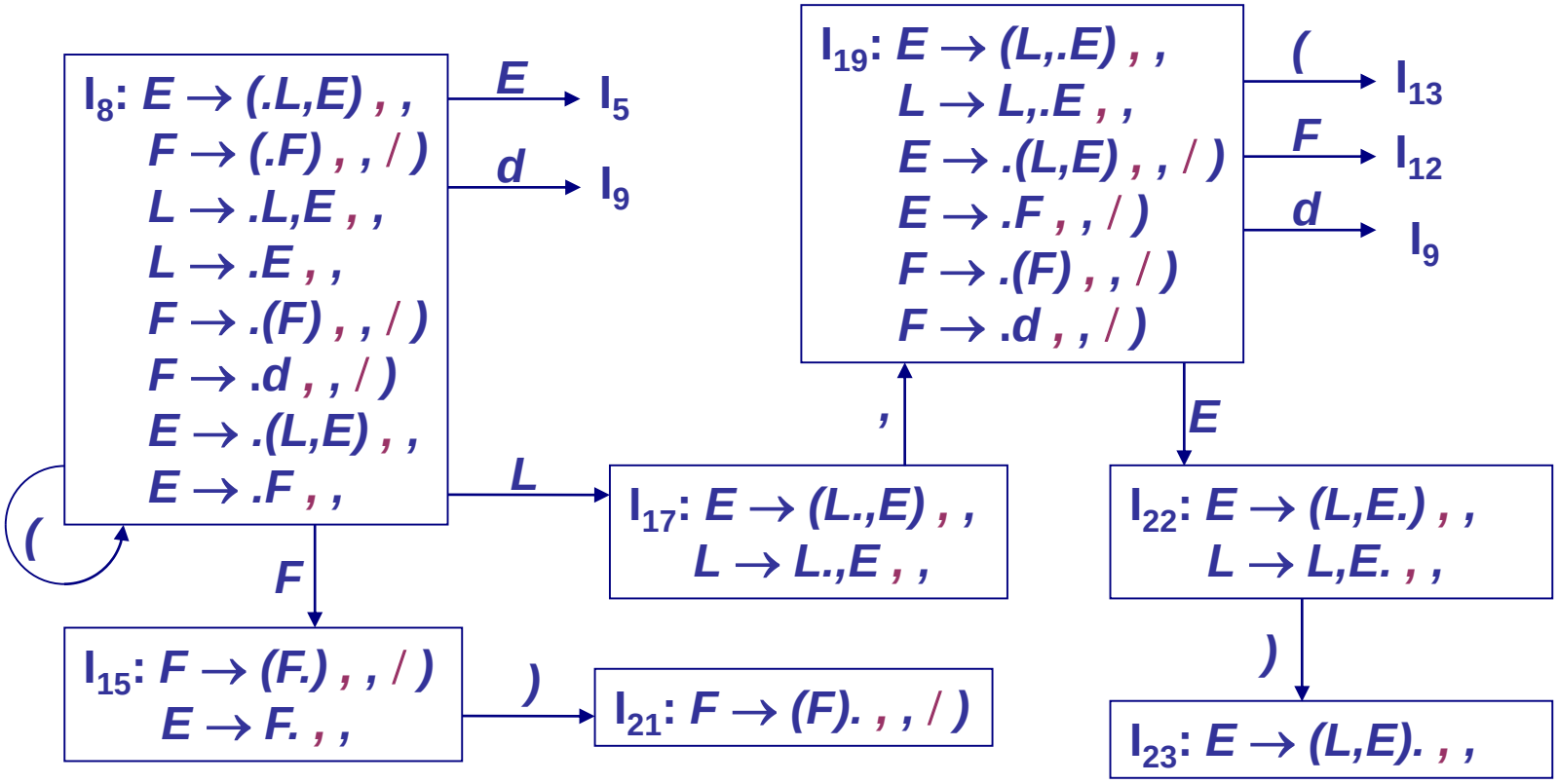
– 构造拓广文法*G'* [S] 的 LR（1）A



LR（1）分析

✧ LR（1）A的构造举例

– 构造拓广文法*G'* [S] 的 LR（1）A



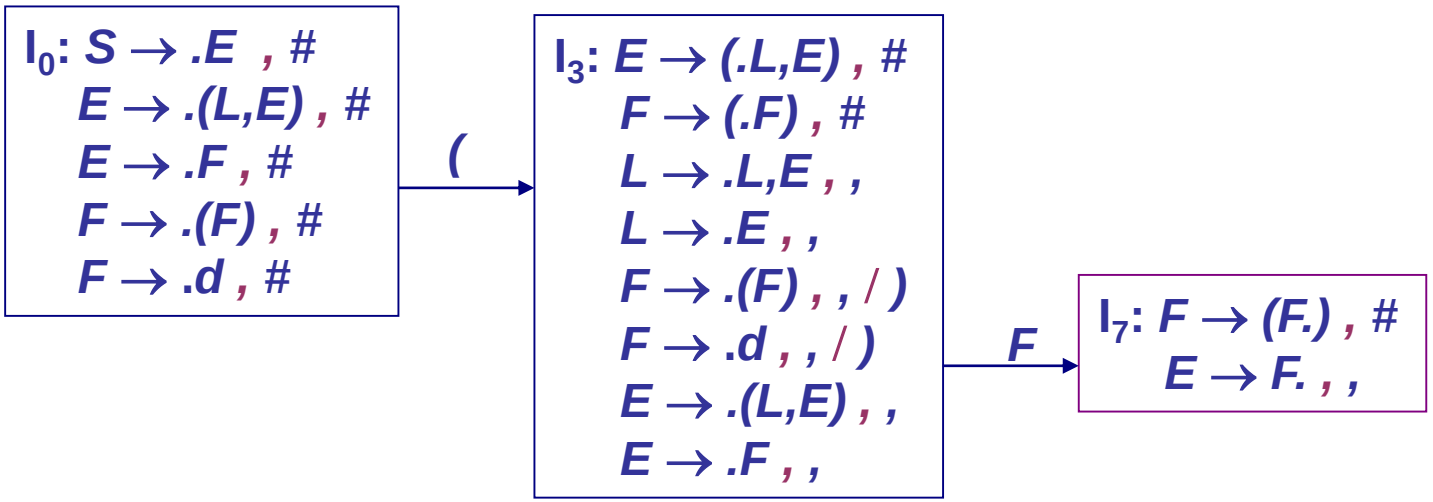
LR（1）分析

✧ 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）A，任何状态都不存在（移进–归约或归约–归约）冲突

拓广文法*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.aβ, 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

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

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	16	14
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

– 拓广文法：（0）$S \rightarrow E$ （1）$E \rightarrow (L,E)$ （2）$E \rightarrow F$

$G'[S]$ （3）$L \rightarrow L,E$ （4）$L \rightarrow E$ （5）$F \rightarrow (F)$ （6）$F \rightarrow d$

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		11
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，则称它为文法 *G* 的一张 LR（1）表，并称 *G* 为一个 LR（1）文法
- LR（1）文法的LR（1）A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

- 可推广到更强大的LR（k）分析
  - LR（k）项目

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

    移进项目形如：[$A \rightarrow \alpha . a\beta$， $a_1\ a_2 \ldots a_k$]
    待约项目形如：[$A \rightarrow \alpha . B\beta$， $a_1\ a_2 \ldots a_k$]
    归约项目形如：[$A \rightarrow \alpha .$ ， $a_1\ a_2 \ldots a_k$]
  - 只有理论意义（ LR（1）A 的状态数已经太大，k>1的情形难以想象）

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）A 状态数目要比它的 LR（1）A状态数目少得多
  - 合并LR（1）A的相似状态

    LALR（1）分析中将同芯/核心（**core**）的LR（1）A状态合并，得到与LR（0）A 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）A的同芯状态

– LR（1）项目的“芯”或“核心”（*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）A 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）A 同芯状态举例

– 文法*G'*[S] 的 LR（1）A中的同芯状态（共 9 组）

拓广文法*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）A 同芯状态举例

– 文法*G'*[S] 的 LR（1）A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）A 同芯状态举例

– 文法*G'*[S] 的 LR（1）A中的同芯状态（共 9 组）

I_{11}: *E* → *(L,E.)* , #
L → *L,E.* , ,

I_{22}: *E* → *(L,E.)* , ,
L → *L,E.* , ,

I_{24}: *E* → *(L,E.)* , , /)
L → *L,E.* , ,

I_{16}: *F* → *(F).* , #

I_{21}: *F* → *(F).* , , /)

I_{20}: *E* → *(L,E).* , #

I_{23}: *E* → *(L,E).* , ,

I_{25}: *E* → *(L,E).* , , /)

LALR（1）分析

✧ LR（1）A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）A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）A的构造

– “brute-force”方法

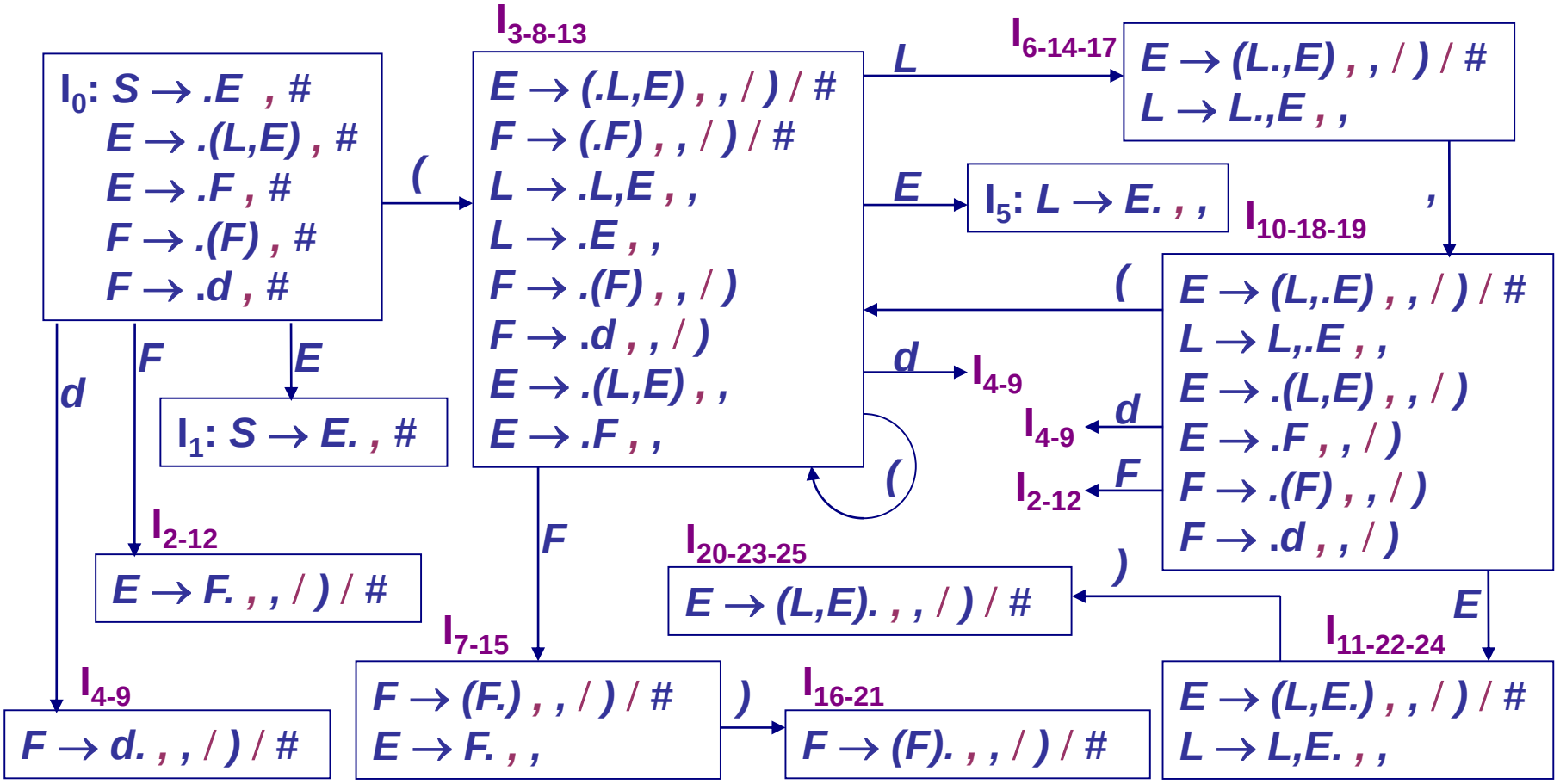
- 构造拓广文法的 LR（1）A 状态
- 合并“同芯”的状态（同芯项目的搜索符集合相并）得到LALR(1) A 的状态
- LALR(1) A 状态由 GO 函数得到的后继状态是所有被合并的“同芯”状态的后继状态之并

  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）A的构造

– “brute-force”方法举例（续前例）



LALR（1）分析

- LALR（1）A的构造
  - “on-the-fly”方法或逐步合并法（*step-by-step merging*）

    比“brute-force”的方法更有效

    - LALR（1）A的状态和 GO 函数的构造过程同LR（1）A，但每产生一个新状态后，需判断其是否与已有状态“同芯”；若是，就并到“同芯”的状态，否则才加入此状态

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

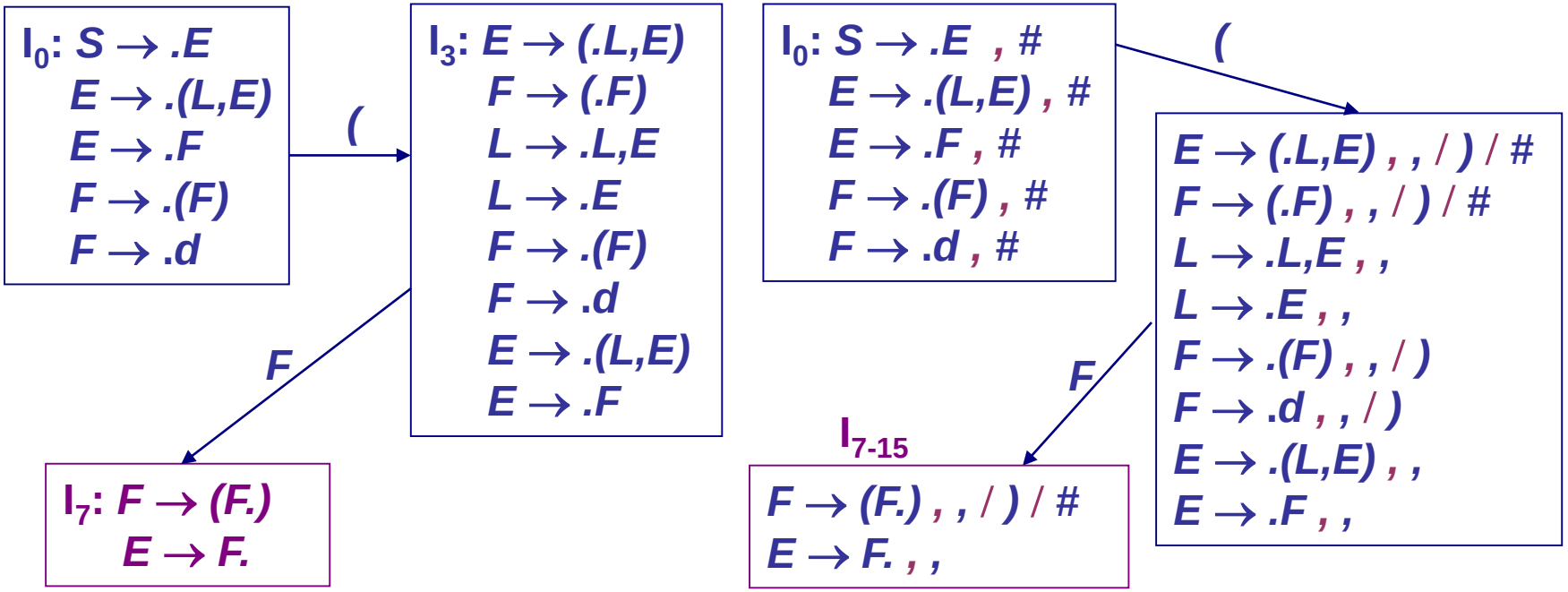
– 拓广文法：（0）$S \rightarrow E$ （1）$E \rightarrow (L,E)$ （2）$E \rightarrow F$
$G'[S]$ （3）$L \rightarrow L,E$ （4）$L \rightarrow E$ （5）$F \rightarrow (F)$ （6）$F \rightarrow d$

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）A 的状态数目与 LR（0）A 相同
  - LALR（1）分析强于SLR（1）分析

比较如下 LR（0）A （左）和 LALR（1）A（右）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下述文法的SLR（1）分析器

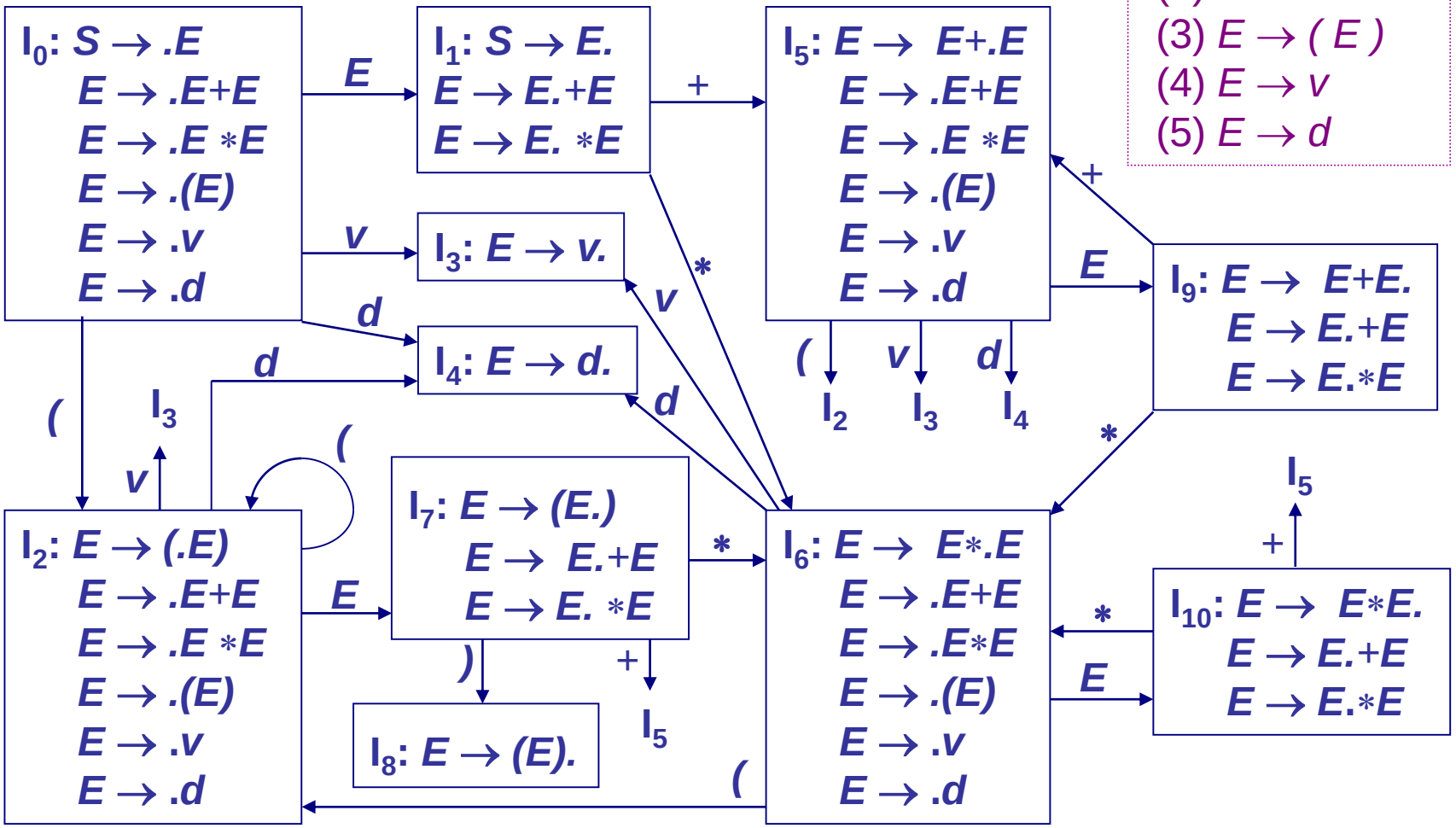
拓广文法$G'[S]$:

(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）A

(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）A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} 中存在移进–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）A 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 分析中的错误处理

✧ 简单的LR 分析错误处理

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

LR 分析中的错误处理

✧ 简单的LR 分析错误处理举例

– 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	

算符优先分析（参见龙书第1版）

- 算符优先文法的定义
- 算符优先关系表的构造
- 算符优先分析法的特点

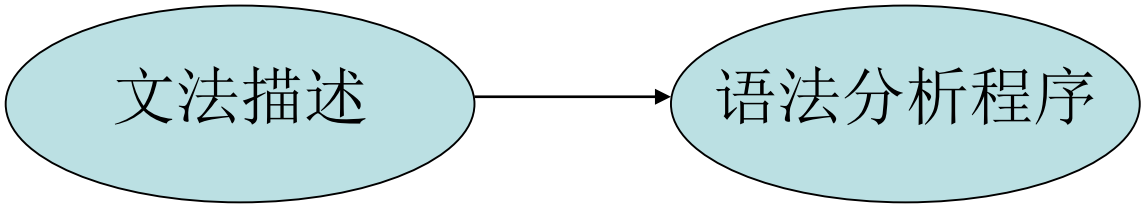
见夏天航、李胜涛同学讲解。

YACC 概述

- **YACC是一个语法分析程序的自动产生系统**

YACC源程序 ⟶ YACC ⟶ parser_tab.c文件

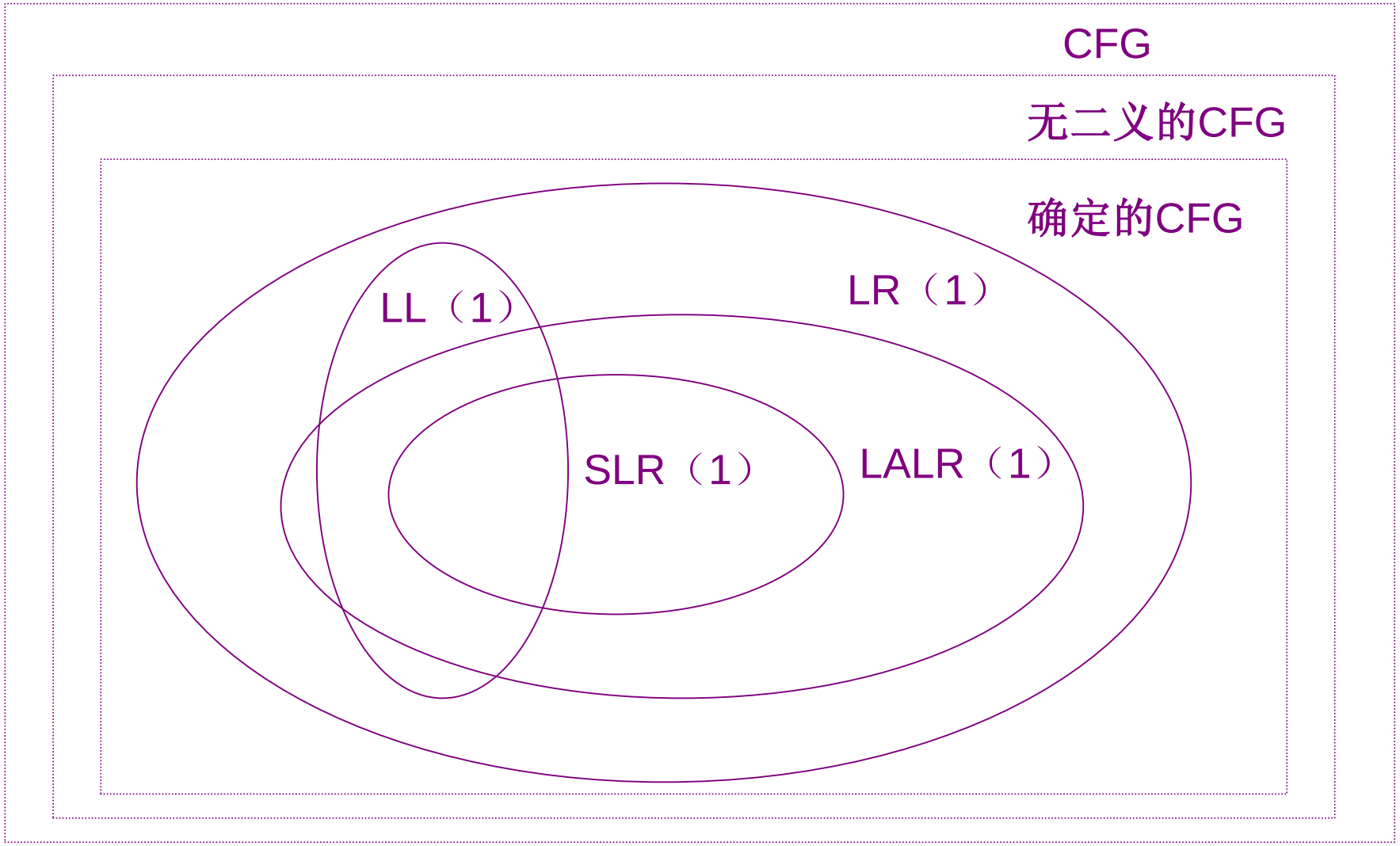
- **YACC的工作原理：**



- **YACC的处理能力：可以用LALR(1)文法表示的上下文无关文法。**

- **见张小健、张嘉睿同学讲解。**

几类分析文法之间的关系



Chapter 5
Syntax-Directed Translation

王朝坤

IISE@Tsinghua

Outline

- ✧ 语义处理概述
- ✧ 语法制导定义
- ✧ 基于语法制导定义的语义处理
- ✧ 翻译方案

✧ 语义处理在编译程序中的逻辑位置

✧ 语义处理在编译程序中的逻辑位置

- 两项工作：

  静态语义分析：主要工作如类型检查、名字的作用域分析等

  中间/目标代码生成：从语法分析的结果生成中间代码（个别编译程序可能直接生成目标代码）

- 跨分析和综合两个阶段

  分析阶段：理解源程序，挖掘源程序的语义

  综合阶段：生成与源程序语义上等价的目标程序

- 跨编译程序的前端和后端

语义处理概述

✧ 语义处理技术

– 语法制导的语义处理

编译程序的设计中，语义分析和中间（目标）代码生成的实现多采用语法制导的语义处理（许多时候直接称之为语法制导的翻译）

语法制导语义处理技术的基础是应用语法制导定义

✧ 现实生活中的语义

语法制导定义

✧ 概念

– 语法制导定义（*Syntax-Directed Definition, SDD*）
– 属性文法（*Attribute Grammar*）
在上下文无关文法
的基础上进行如下扩展：

- 为每个文法符号关联多个属性（*Attribute*）
- 为文法的每个产生式关联一个语义规则集合

语法制导定义

✧ 概念

- 属性（*Attribute*）可用来刻画一个文法符号的任何我们所关心的特性，如：符号的值，符号的名字串，符号的类型，符号的存储地址，符号被赋予的寄存器，代码片断，等等…
- 记号

  文法符号 ***X*** 关联属性 ***a*** 的属性值可通过 ***X.a*** 访问

语法制导定义

✧ 概念

– 语义规则（*Semantic Rule*）

在**SDD**中，每个产生式 $A \rightarrow \alpha$ 都关联一个语义规则的集合，用于描述如何计算当前产生式中文法符号的属性值

– **SDD**中允许两种语义规则

- 复写（*copy*）规则，形如

$$X.a := Y.b$$

- 基于语义函数（*semantic function*）的规则，形如

$b:=f(c_1, c_2, \ldots, c_k)$ 或 $f(c_1, c_2, \ldots, c_k)$

其中，$b, c_1, c_2, \ldots, c_k$是该产生式中文法符号的属性

– 实践中，对语义函数的限制会适当放宽

语法制导定义

✧ 有两种属性：综合属性和继承属性

- 综合属性（*synthesized attribute*）

  用于“自下而上”传递信息

  对关联于产生式 $A \rightarrow \alpha$ 的语义规则 $b:=f(c_1, c_2, \ldots, c_k)$，如果 ***b*** 是 ***A*** 的某个属性, 则称 ***b*** 是 ***A*** 的一个综合属性

- 继承属性（*inherited attribute*）

  用于“自上而下”传递信息

  对关联于产生式 $A \rightarrow \alpha$ 的语义规则 $b:=f(c_1, c_2, \ldots, c_k)$，如果 ***b*** 是产生式右部某个文法符号 ***X*** 的某个属性，则称 ***b*** 是文法符号 ***X*** 的一个继承属性

语法制导定义

✧ 语法制导定义举例

– 仅含综合属性的例子（开始符号S）

产生式	语义规则
$S \rightarrow E$	{ *print(E.val)* }
$E \rightarrow E_1 + T$	{ *E.val* := E_1*.val* + *T.val* }
$E \rightarrow T$	{ *E.val* := *T.val* }
$T \rightarrow T_1 * F$	{ *T.val* := T_1*.val* × *F.val* }
$T \rightarrow F$	{ *T.val* := *F.val* }
$F \rightarrow (E)$	{ *F.val* := *E.val* }
$F \rightarrow d$	{ *F.val* := *d.lexval* }

注：*d.lexval* 是词法分析程序确定的属性值

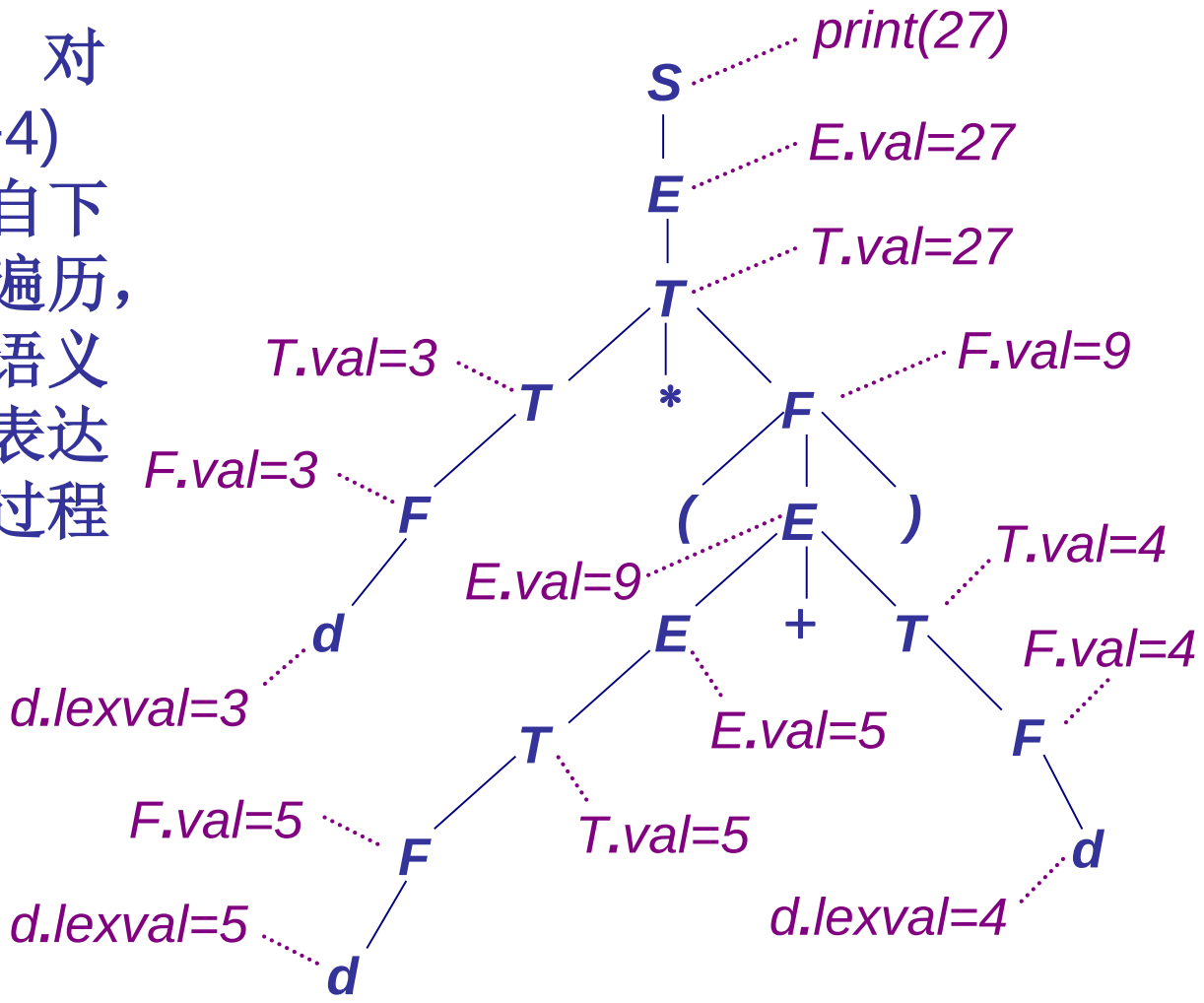
继承属性：⇩

综合属性：⇧

语法制导定义

✧ 综合属性代表自下而上传递的信息

– 接上页的例子，对表达式 3 * (5+4) 的分析树进行自下而上（后序）遍历，并执行相应的语义规则，得到该表达式的一种求值过程



语法制导定义

✧ 语法制导定义举例

– 含继承属性的例子（开始符号D）

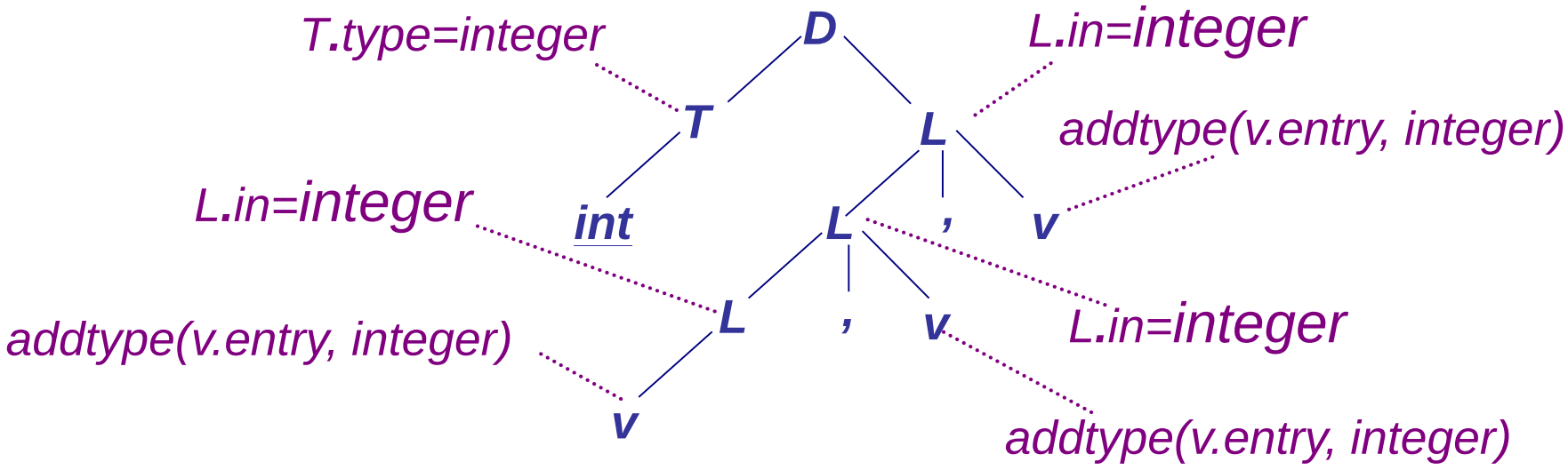
产生式	语义规则
$D \rightarrow T\ L$	{ *L.in* := *T.type* }
$T \rightarrow \underline{int}$	{ *T.type* := *integer* }
$T \rightarrow \underline{real}$	{ *T.type* := *real* }
$L \rightarrow L_1\ ,\ v$	{ L_1*.in* := *L.in* *addtype(v.entry,L.in)* }
$L \rightarrow v$	{ *addtype(v.entry,L.in)* }

注：语义函数 *addtype(v.entry,L.in)* 表示将属性值 *L.in* 填入 *v* 的符号表项中的 *type* 域

继承属性：⇩
综合属性：⇧

✧ 继承属性代表自上而下传递的信息

– 接上页的例子，对声明语句 int x,y,z 的分析树进行遍历，自下而上执行综合属性相应的语义规则，自上而下执行继承属性相应的语义规则，可以得到所有属性值的一个求值过程



语法制导定义

✧ 语法制导定义举例

– 更复杂的例子（开始符号N）

产生式	语义规则
$N \rightarrow S_1.S_2$	$\{ N.v := S_1.v+S_2.v;\ S_1.f := 1;\ S_2.f := 2^{-S_2.l} \}$
$S \rightarrow S_1B$	$\{ S_1.f := 2S.f;\ B.f := S.f;\ S.v := S_1.v+B.v;\ S.l := S_1.l +1 \}$
$S \rightarrow B$	$\{ S.l := 1 ;\ S.v := B.v ;\ B.f := S.f \}$
$B \rightarrow 0$	$\{ B.v := 0 \}$
$B \rightarrow 1$	$\{ B.v := B.f \}$

该语法制导定义可用于

将二进制无符号小数转化为十进制小数

继承属性：⇩

综合属性：⇧

Conclusions

- ✧ LR(1) 分析
- ✧ LALR(1)分析
- ✧ 二义文法
- ✧ LR 分析错误处理
- ✧ 语义处理概述
- ✧ 语法制导定义

- § 4.7 More Powerful LR Parsers
- § 4.8 Using Ambiguous Grammars
- § 4.9 Parser Generators
- § 5.1 Syntax-Directed Definition

课外学习建议

✧ Floyd, R. W. Syntactic Analysis and Operator Precedence. J. ACM. 10:3, 316-333. 1961

✧ Engelfriet, J. Attribute Evaluation Methods. *Methods and Tools for Compiler Construction*. 103-138. 1984

Thank you!