



第4章 语法分析 (4)

Syntax Analysis

【对应教材 4.6-4.9】



内容提要

- 语法分析简介
- 上下文无关文法
- 文法的设计方法
- 自顶向下的语法分析
- 自底向上的语法分析
 - 简单LR分析: LR(0), SLR
 - **更强大的LR分析: LR(1), LALR**
 - 二义性文法的使用
- 语法分析器生成工具YACC



回顾：自底向上的分析方法

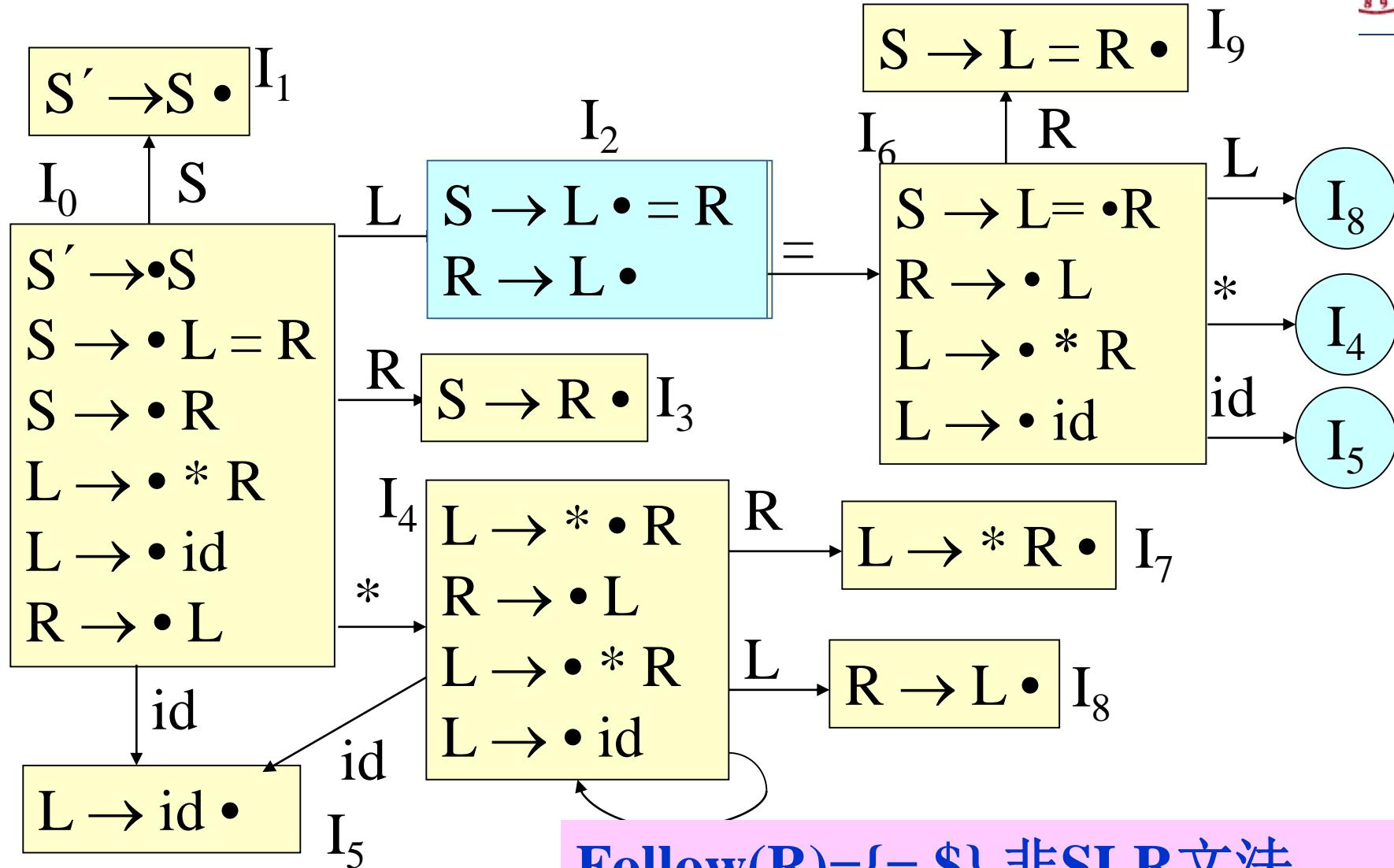
- 自底向上的语法分析：把给定的符号串归约为开始符号S的过程（最右推导的逆过程）
- （无回溯的）自底向上分析的关键：每一步寻找当前句型中的句柄进行归约
 - 句柄：当前最右句型中最右推导的最后一步
- 最常用的自底向上分析方法：**LR分析**
 - 基于栈的分析方法，采用通用的分析驱动程序
 - LR分析的关键是构造相应**LR分析表**
 - 首先需要构造**LR(0)**自动机：**LR(0)**项集规范族与Goto函数



LR 分析表的种类

- LR(0) 分析表
 - 过于简单，只能用于最简单的文法，不实用
- SLR分析表 (Simple LR)
 - 构造简单，最易实现，大多数上下文无关文法都可以构造出SLR分析表，具有较高的实用价值
- LR(1)分析表 (Canonical LR)
 - 适用文法类最大，几乎所有上下文无关文法都能构造出LR分析表，但其分析表体积太大，实用价值不大
- LALR(1)分析表 (LookAhead LR)
 - 这种表适用的文法类及其实现上难易在上面两种之间，较为实用

非SLR(1)文法: (0) $S' \rightarrow S$ (1) $S \rightarrow L = R$ (2) $S \rightarrow R$
 (3) $L \rightarrow^* R$ (4) $L \rightarrow id$ (5) $R \rightarrow L$

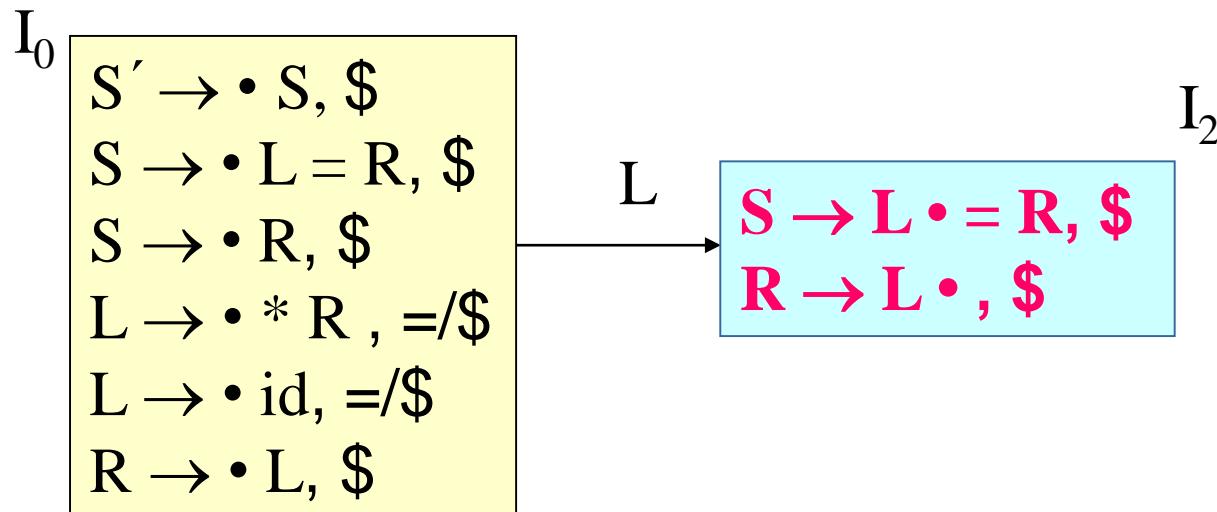


Follow(R)={=,\$}, 非SLR文法。



LR(1)分析

- 问题：栈中已识别出一个L，此时，a是“=”，不能用“R→L”进行归约，因为不存在“R=”的规范句型。
- 解决方法：添加搜索符号串。
 - 使每个项中含有更多的展望信息，使得能确切知道何时能进行归约。





LR(k)项

□ LR(k) 项

形式: $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \dots a_k]$

- $\beta \neq \varepsilon$ 时, 是移进或待归约项, $a_1 a_2 \dots a_k$ 不直接起作用。
- 对归约项 $[A \rightarrow \alpha \cdot, a_1 a_2 \dots a_k]$, 仅当前输入符号串开始的前 k 个符号是 $a_1 a_2 \dots a_k$ 时, 才能用 $A \rightarrow \alpha$ 进行归约。
- $a_1 a_2 \dots a_k$ 称为向前搜索符号串。



LR(1)分析表的构造

□ LR(1)有效项：

如果存在一个规范推导

$$S \Rightarrow^*_{rm} \delta Aw \Rightarrow_{rm} \delta\alpha\beta w$$

则称一个LR(1)项 $[A \rightarrow \alpha \cdot \beta, a]$ 对活前缀 $\gamma = \delta\alpha$ 是有效的。

- 其中或w的第一个符号为a, 或w=ε而a为\$。

例：考虑文法G: $S \rightarrow CC \quad C \rightarrow cC \mid d$

因为有规范推导

$$S \Rightarrow^*_{rm} ccCc d \Rightarrow_{rm} cccCc d$$

故项 $[C \rightarrow c \cdot C, c]$ 对活前缀 ccc 是有效的。



如何推导LR(1)的有效项

- 若项 $[A \rightarrow \alpha \cdot B\beta, a]$ 对活前缀 $\gamma = \delta\alpha$ 是有效的，则存在一个规范推导

$$S \Rightarrow^*_{rm} \delta Aax \Rightarrow_{rm} \delta\alpha B\beta ax$$

- 假定 $B\beta ax \Rightarrow^*_{rm} by$ ，则对每一个形如 $B \rightarrow \xi$ 的产生式，我们有规范推导

$$S \Rightarrow^*_{rm} \delta\alpha Bby \Rightarrow_{rm} \delta\alpha\xi by$$

- 从而项 $[B \rightarrow \cdot \xi, b]$ 对于活前缀 $\gamma = \delta\alpha$ 也是有效的。
- 注意到 b 或者是从 β 推出的第一个终结符号，或者 $\beta \Rightarrow^*_{rm} \varepsilon$ 而 $b = a$ 。这两种可能性结合在一起，则 $b \in \text{first}(\beta a)$ 。



LR(1)项集的构造

- 设I是G的一个LR(1)项集, $\text{closure}(I)$ 是从I出发用下面三个规则构造的项集：
 1. 每一个I中的项都属于 $\text{closure}(I)$ 。
 2. 若项 $[A \rightarrow \alpha \bullet B\beta, a]$ 属于 $\text{closure}(I)$ 且 $B \rightarrow \gamma \in P$, 则对任何 $b \in \text{FIRST}(\beta a)$, 把 $[B \rightarrow \bullet \gamma, b]$ 加到 $\text{closure}(I)$ 中。
 3. 重复执行(2)直到 $\text{closure}(I)$ 不再增大为止。
- 设I是G的一个LR(1)项集, X是一个文法符号, 定义

$$\text{GOTO}(I, X) = \text{closure}(J),$$

其中 $J = \{[A \rightarrow \alpha X \bullet \beta, a] \mid \text{当 } [A \rightarrow \alpha \bullet X \beta, a] \in I \text{ 时}\}$



LR(1)项集的CLOSURE算法

```
SetOfItems CLOSURE( $I$ ) {  
    repeat  
        for (  $I$  中的每个项  $[A \rightarrow \alpha \cdot B\beta, a]$  )  
            for (  $G'$  中的每个产生式  $B \rightarrow \gamma$  )  
                for ( FIRST( $\beta a$ ) 中的每个终结符号  $b$  )  
                    将  $[B \rightarrow \cdot \gamma, b]$  加入到集合  $I$  中;  
    until 不能向  $I$  中加入更多的项 ;  
    return  $I$ ;  
}
```

注意添加 $[B \rightarrow \cdot \gamma, b]$ 时， b 的取值范围。



LR(1)项集的GOTO算法

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

和LR(0)项集的GOTO算法基本相同



LR(1)项集簇的主构造算法

```
void items( $G'$ ) {
```

 将 C 初始化为 CLOSURE($\{[S' \rightarrow \cdot S, \$]\}\}$)

 repeat

 for (C 中的每个项集 I)

 for (每个文法符号 X)

 if (GOTO(I, X) 非空且不在 C 中)

 将 GOTO(I, X) 加入 C 中 ;

 until 不再有新的项集加入到 C 中 ;

}

和LR(0)项集簇的构造算法相同



LR(1)项集及规范族的构造过程

例4_9 $G(S')$: (0) $S' \rightarrow S$ (1) $S \rightarrow CC$
(2) $C \rightarrow cC$ (3) $C \rightarrow d$

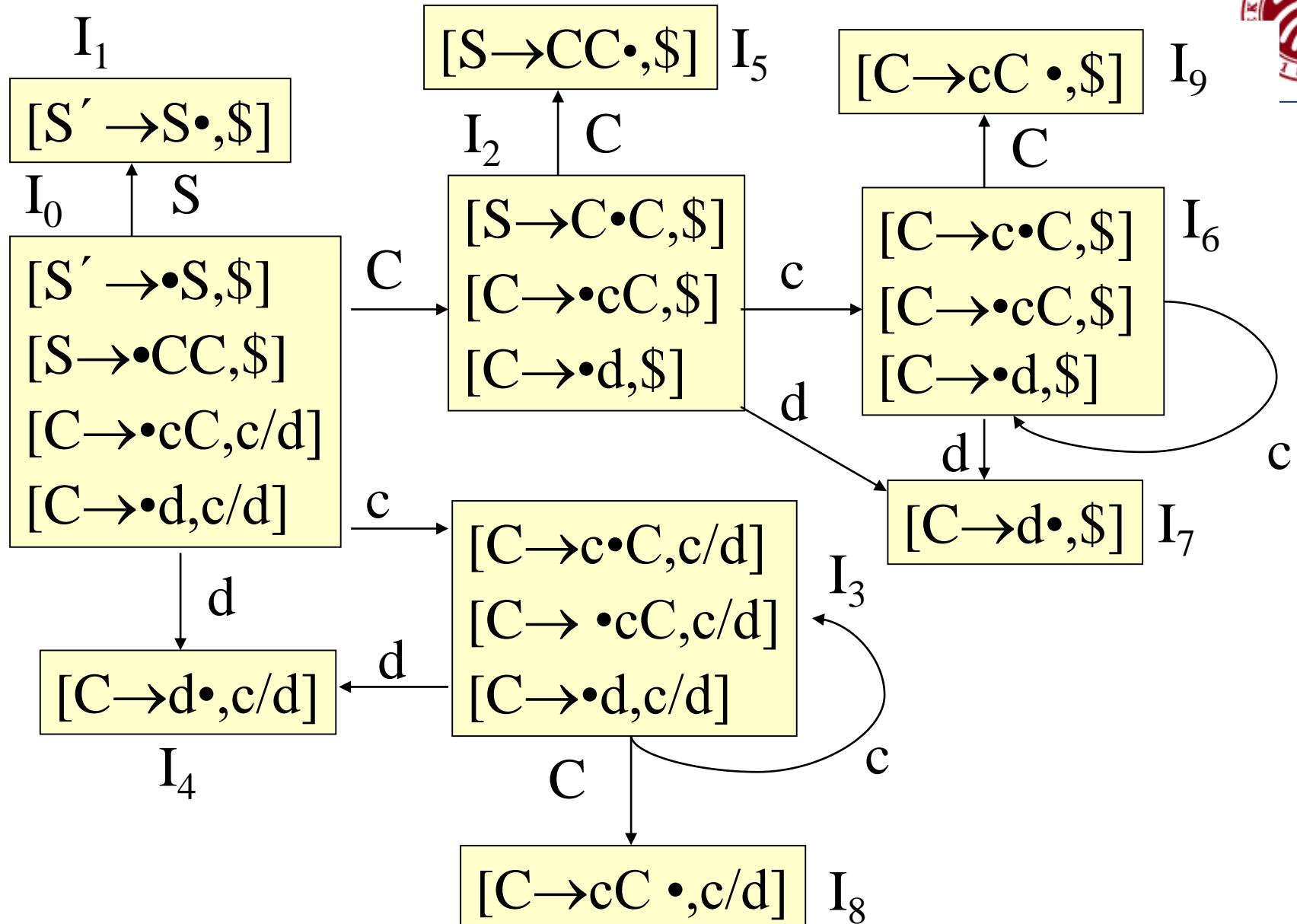
若 $[A \rightarrow \alpha \bullet B\beta, a] \in I$

且 $B \rightarrow \gamma \in P$

则 $[B \rightarrow \bullet \gamma, b] \in I$

其中 $b \in \text{FIRST}(\beta a)$

I_0
 $[S' \rightarrow \bullet S, \$]$
 $[S \rightarrow \bullet CC, \$]$
 $[C \rightarrow \bullet cC, c/d]$
 $[C \rightarrow \bullet d, c/d]$





构造LR(1)分析表

□ 根据DFA构造LR(1)分析表M:

- DFA中的每个状态对应分析表中的一行
- 对于DFA中的每一个从状态*i*到状态*j*的转移
 - 如果转移符号为终结符a: 在相应的表项 $M[i, a]$ 中填写移进动作 S_j
 - 如果转移符号为非终结符A: 在相应的表项 $M[i, A]$ 中填写要转移到的状态 *j*
- 对于包含归约项 [$A \rightarrow \alpha^*, a$] 的状态 *i*
 - 在相应的表项 $M[i, a]$ 中填写归约动作 r_k , 其中 *k* 是产生式 $A \rightarrow \alpha$ 的编号。

如果按照上面的步骤填写的分析表中没有冲突（即每个单元格中只包含一个动作），那么得到的就是合法的LR(1)分析表



例：LR(1)分析表

	c	d	\$	S	C
0	S_3	S_4		1	2
1			acc		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

回顾: (0) $S' \rightarrow S$

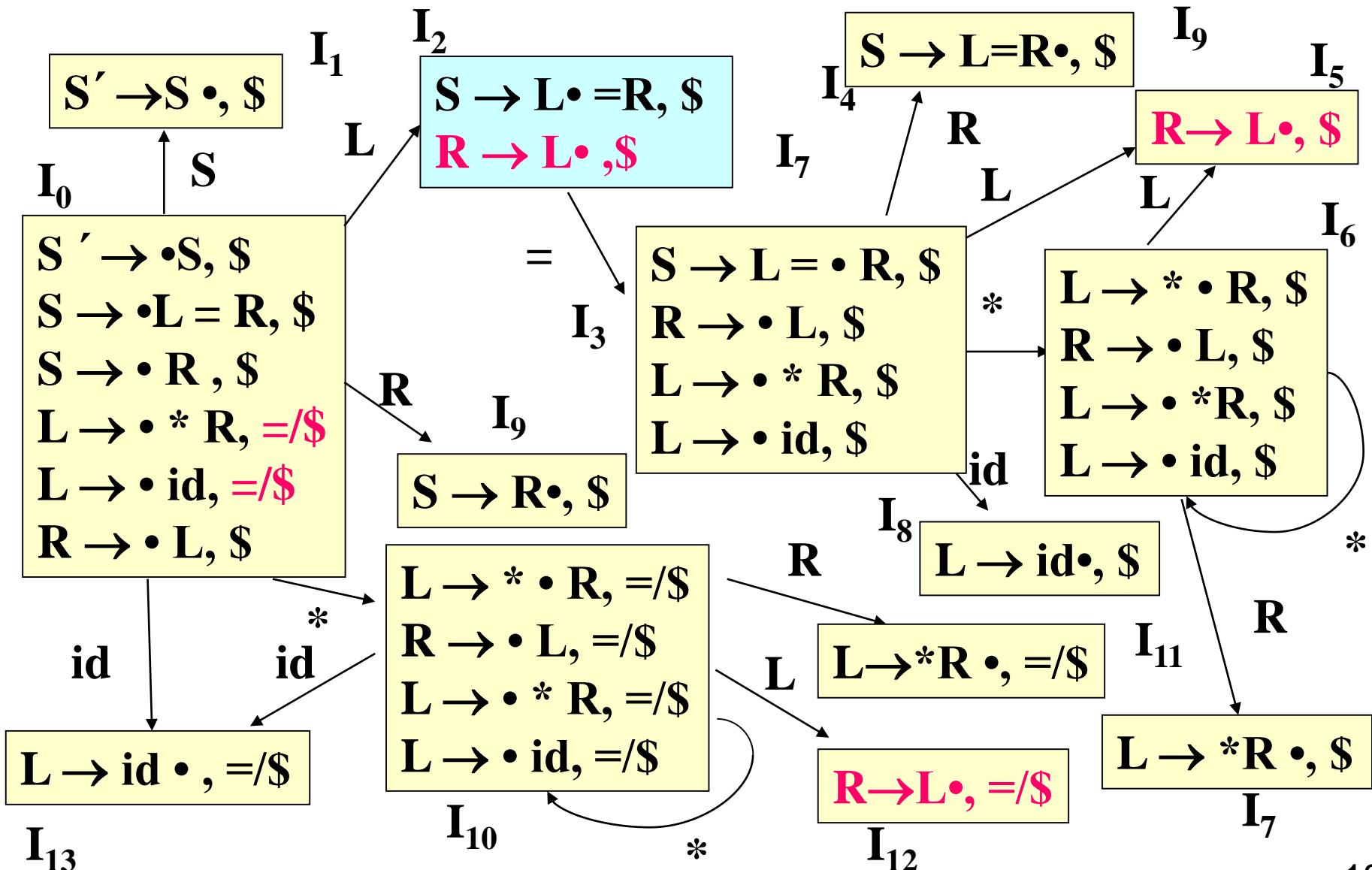
(1) $S \rightarrow L = R$

(2) $S \rightarrow R$

(3) $L \rightarrow^* R$

(4) $L \rightarrow id$

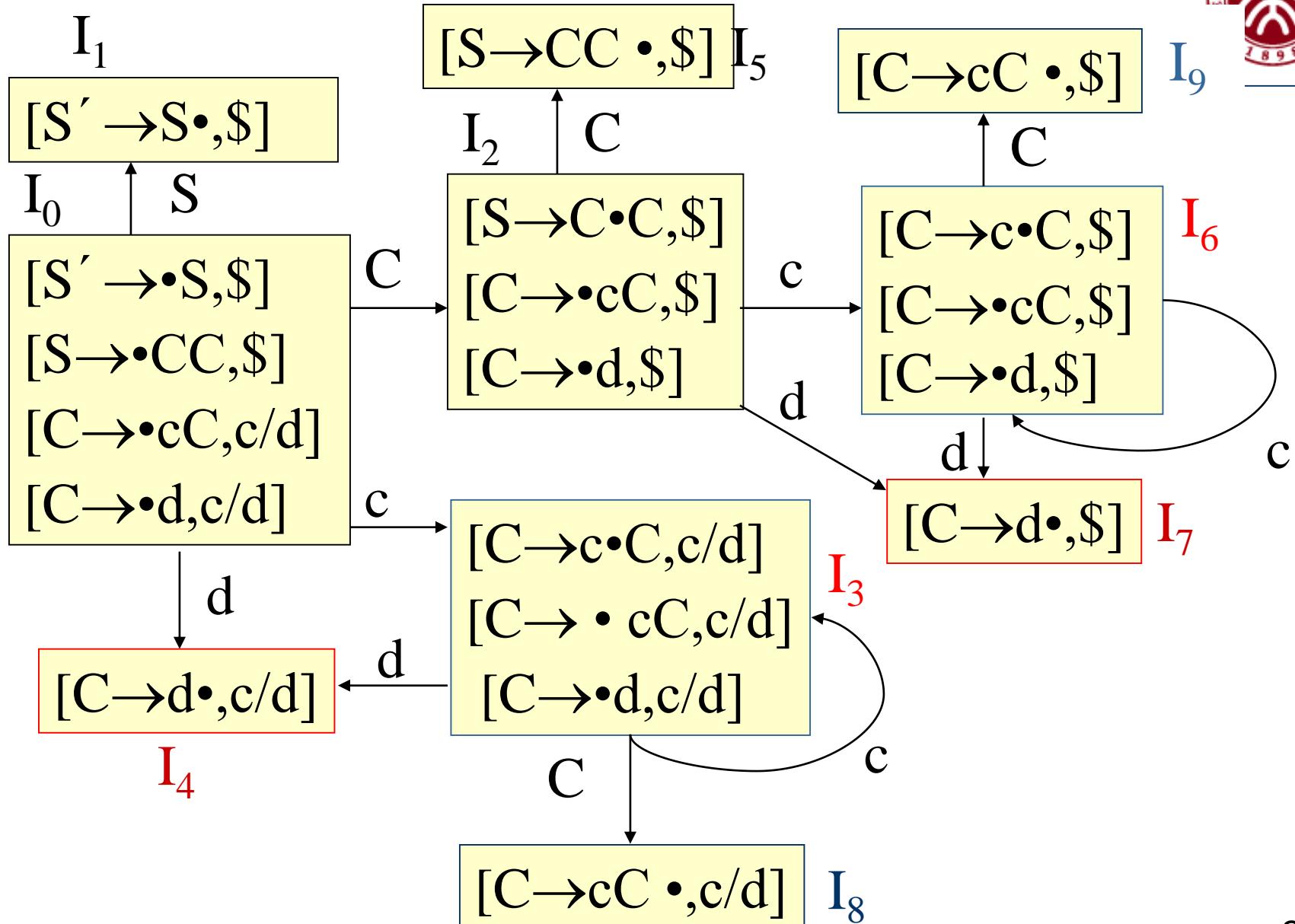
(5) $R \rightarrow L$

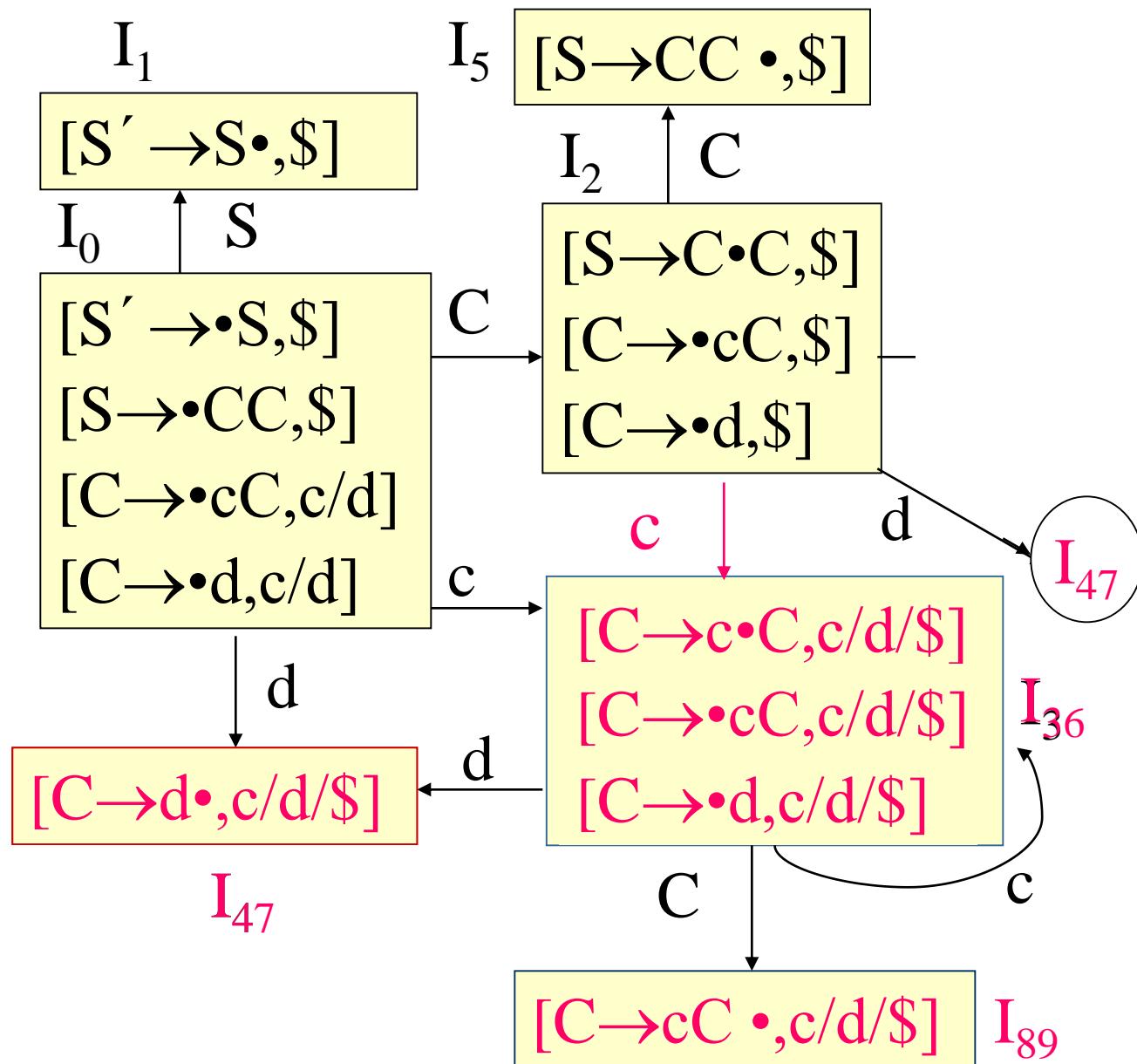




LALR文法

- LR(1)分析表常常会含有比较多的状态，在实际使用中不是很常用。
- 同心集：如果两个LR(1)项集去掉搜索符之后是相同的，则称这两个项集具有相同的核（core）。
- 合并同心集（合并搜索字符串）后，像构造LR(1)分析表一样构造出的LR分析表称作是**LALR(1)分析表**。







对应的LALR 分析表

	c	d	\$	S	C
0	S_3	S_4		1	2
1			acc		
2	S_6	S_7			5
3(6)	S_3	S_4			8
4(7)	r_3	r_3	r_3		
5			r_1		
6	S_6	S_7			9
7			f_3		
8(9)	r_2	r_2	r_2		
9			f_2		



LALR分析表的高效构造算法

- LALR的项集可以看作是在LR(0)项集上添加了适当的向前看符号。
- 基本方法
 - 只使用内核项来表示LR(0)或LR(1)项集。只使用 $[S' \rightarrow \cdot S]$ 或 $[S' \rightarrow \cdot S, \$]$ ，以及点不在最左端的项
 - 使用传播和自发生成的过程来生成向前看符号，得到LALR(1)内核项
 - 使用CLOSURE函数，求出内核的闭包。然后得到LALR分析表

详细算法参见龙书第4.7.5节



LALR(1)的讨论

1. 由于 $\text{goto}(I, X)$ 仅仅依赖于 I 的核心，因此 $\text{LR}(1)$ 项集合并后的转换函数 $\text{go}(I, X)$ 随自身的合并而得到。
2. 动作 action 应当进行修改，反映各被合并集合的既定动作。
3. 把同心的项集合并的时候，有可能导致冲突，这种冲突不会是移进-归约冲突；但可能引起归约-归约冲突。

$I_k : \{ [A \rightarrow \alpha^*, u_1] \quad [B \rightarrow \beta \cdot a\gamma, b] \} \quad a \cap u_1 = \emptyset$

$I_j : \{ [A \rightarrow \alpha^*, u_2] \quad [B \rightarrow \beta \cdot a\gamma, c] \} \quad a \cap u_2 = \emptyset$

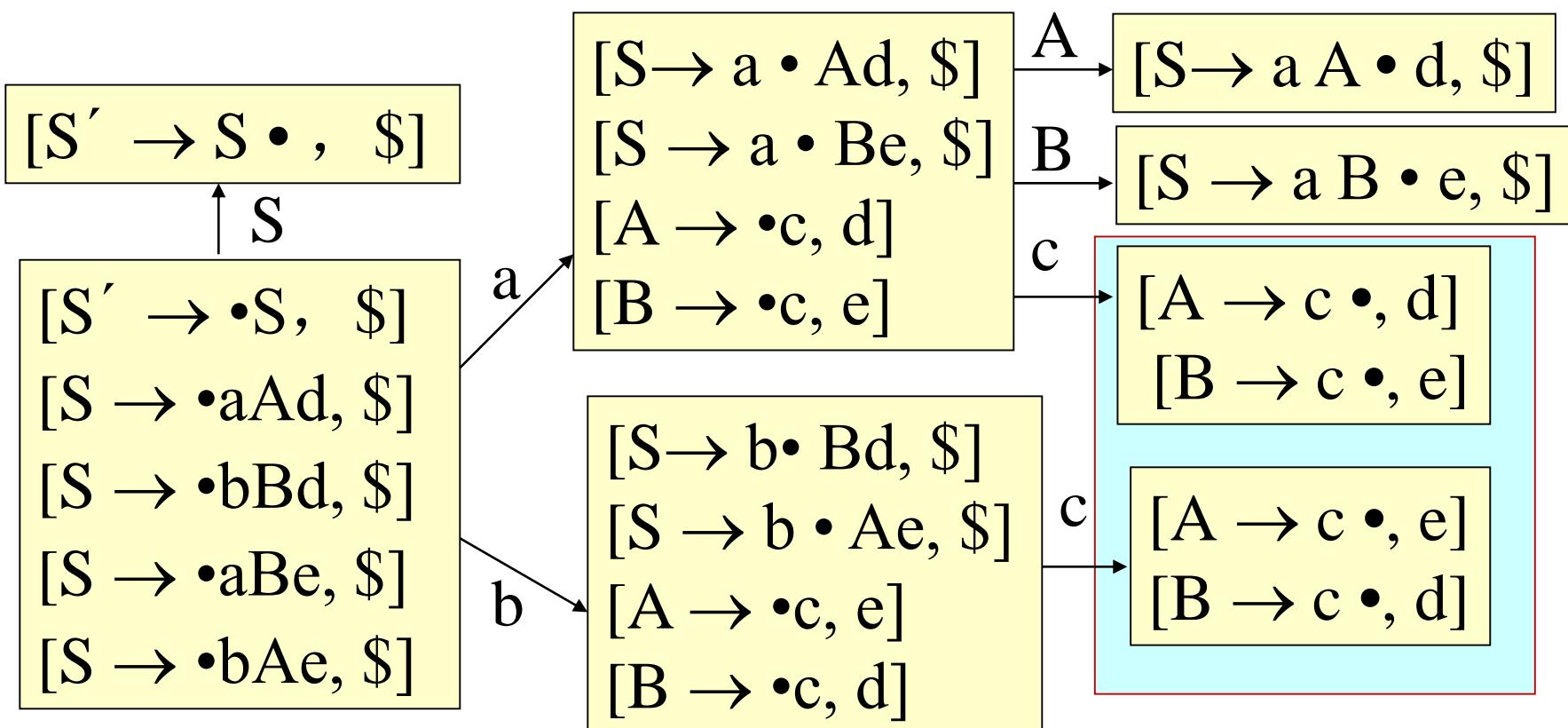
$I_{kj} : [A \rightarrow \alpha^*, u_1/u_2] \quad [B \rightarrow \beta \cdot a\gamma, b/c] \} \quad a \cap \{u_1, u_2\} = \emptyset$



非LALR(1)的LR(1)文法

例: $S' \rightarrow S$
 $A \rightarrow c$

$S \rightarrow aAd \mid bBd \mid aBe \mid bAe$
 $B \rightarrow c$





LR(1)和LALR(1)

LR(1)和LALR(1)在分析上的差别

输入: ccd\$

LR: 0 c 3 c 3 d 4

报错

LALR: 0 c 36 c 36 d 47

0 c 36 c 36 C 89

0 c 36 C 89

0 C 2

报错



二义性文法的使用

- 二义性文法都不是LR的。
- 某些二义性文法是有用的
 - 可以简洁地描述某些结构；
 - 隔离某些语法结构，对其进行特殊处理。
- 对于某些二义性文法
 - 可以通过消除二义性规则来保证每个句子只有一棵语法分析树。
 - 且可以在LR分析器中实现这个规则。



利用优先级/结合性消除冲突

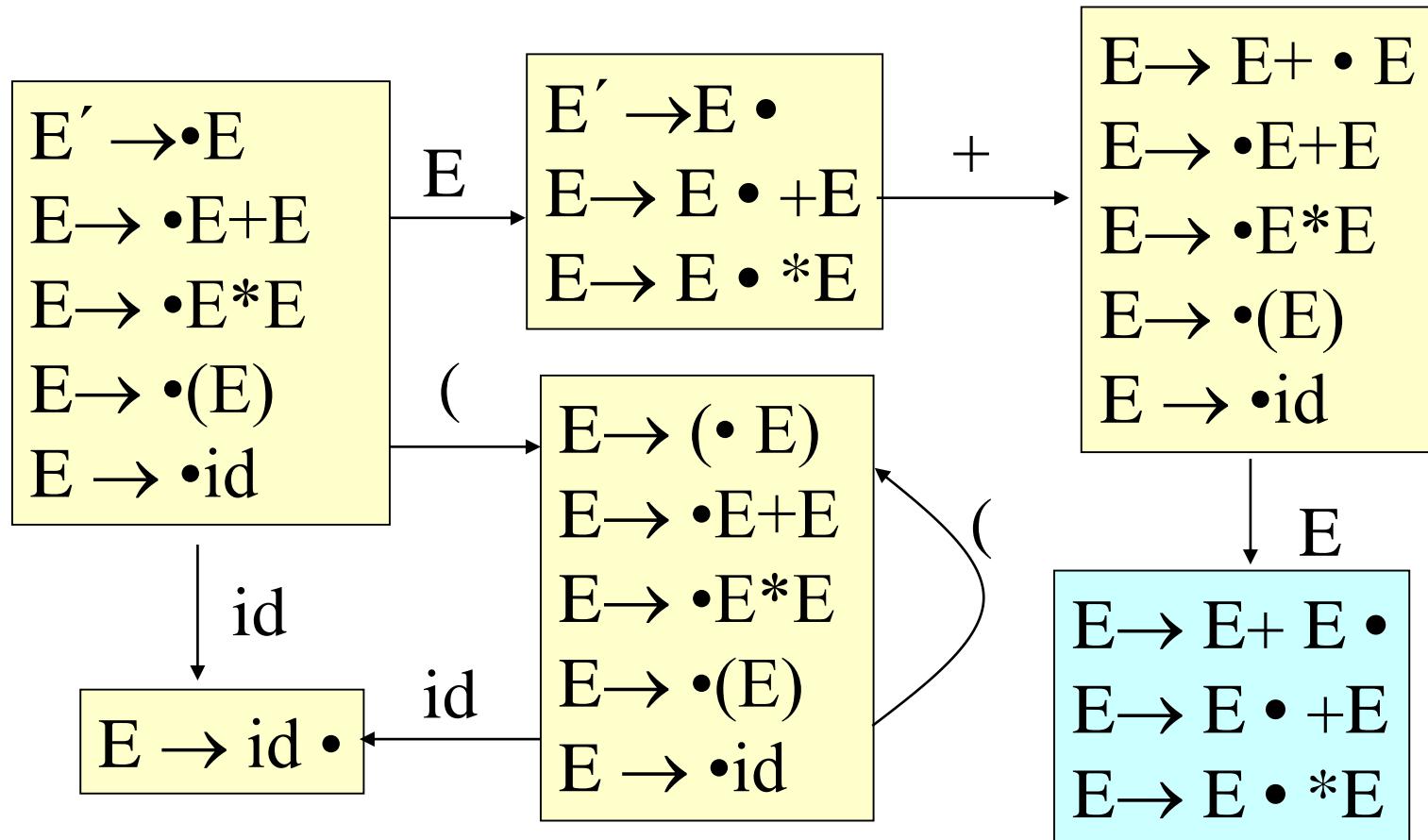
- 二义性文法: $E \rightarrow E+E \mid E*E \mid (E) \mid id$
- 等价于: $E \rightarrow E+T \mid T$ $T \rightarrow T*F \mid F$
 $F \rightarrow (E) \mid id$
- 二义性文法的优点:
 - 容易修改算符的优先级和结合性。
 - 简洁: 如果有多个优先级, 那么无二义性文法
讲引入太多的非终结符号
 - 高效: 不需要处理 $E \rightarrow T$ 这样的归约。



LR分析方法对二义文法的应用

龙书4.8节

表达式文法: $E \rightarrow E+E \mid E^*E \mid (E) \mid id$





悬空else的二义性

□ 文法：

$S' \rightarrow S$

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

$$\begin{aligned} I_0: \quad S' &\rightarrow \cdot S \\ &S \rightarrow \cdot iSeS \\ &S \rightarrow \cdot iS \\ &S \rightarrow \cdot a \end{aligned}$$

$$I_1: \quad S' \rightarrow S \cdot$$

$$\begin{aligned} I_2: \quad S &\rightarrow i \cdot SeS \\ &S \rightarrow i \cdot S \\ &S \rightarrow \cdot iSeS \\ &S \rightarrow \cdot iS \\ &S \rightarrow \cdot a \end{aligned}$$

$$\begin{aligned} I_3: \quad S &\rightarrow a \cdot \\ I_4: \quad S &\rightarrow iS \cdot eS \\ &S \rightarrow iS \cdot \\ I_5: \quad S &\rightarrow iSe \cdot S \\ &S \rightarrow \cdot iSeS \\ &S \rightarrow \cdot iS \\ &S \rightarrow \cdot a \\ I_6: \quad S &\rightarrow iSeS \cdot \end{aligned}$$

- I_4 包含冲突，它出现在栈顶时，栈中符号为 iS
- 如果下一个符号为 e ，因栈中的 i 尚未匹配，因此应该移进 e 。
- 否则，如果下一个符号属于 $\text{Follow}(S)$ ，就归约。



LR分析总结：四种LR分析的对比

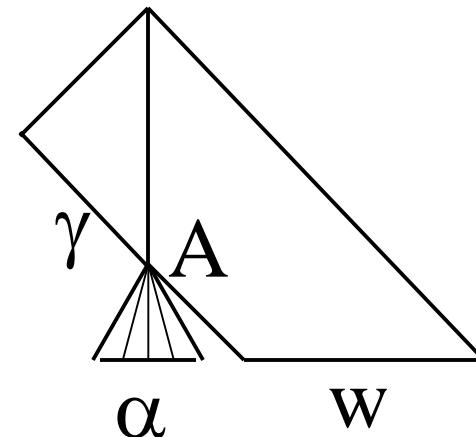
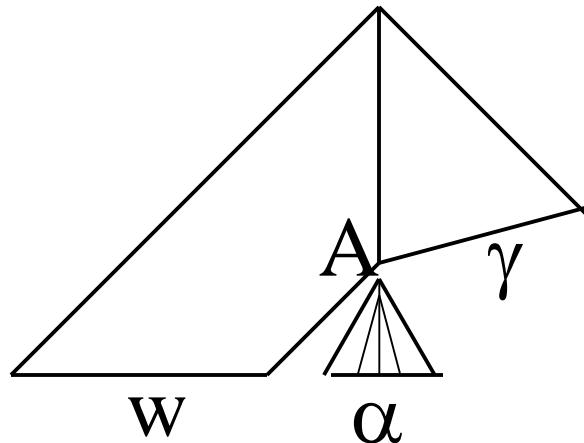
- 如果构造LR(0)的DFA
 - 没有归约冲突就是**LR(0)**文法
 - 有冲突但是可以通过Follow集合解决冲突就是**SLR**文法
 - 否则不是**SLR**文法。
- 如果构造LR(1)的DFA
 - 没有冲突就是**LR(1)**文法
 - 如果合并同心集之后也没有冲突，那么就是**LALR(1)**文法。
- 包含关系：**LR(0) < SLR < LALR < LR(1)**



LR(k)和LL(k)的比较 (1)

1. $A \rightarrow \alpha_1 \mid \alpha_2$

LL(k)根据FIRST(α_i)确定使用哪条产生式；而LR(k)是在识别出整个 α 后，再往后看k个符号，然后确定使用哪条产生式。





LR(k)和LL(k)的比较 (2)

2. LL(k)文法都是LR(k)文法。都能用形式化方法实现。

3. 存在非LR的结构

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

$G[S]: S \rightarrow aSa \mid bSb \mid \epsilon$

4. LR(k)分析用手工构造是不可能的

- 类Pascal语言的LR(1)分析表，需要数千个状态；
- 由于存在自动生成工具（比如YACC），LR分析受到广泛重视。



LR分析的错误恢复

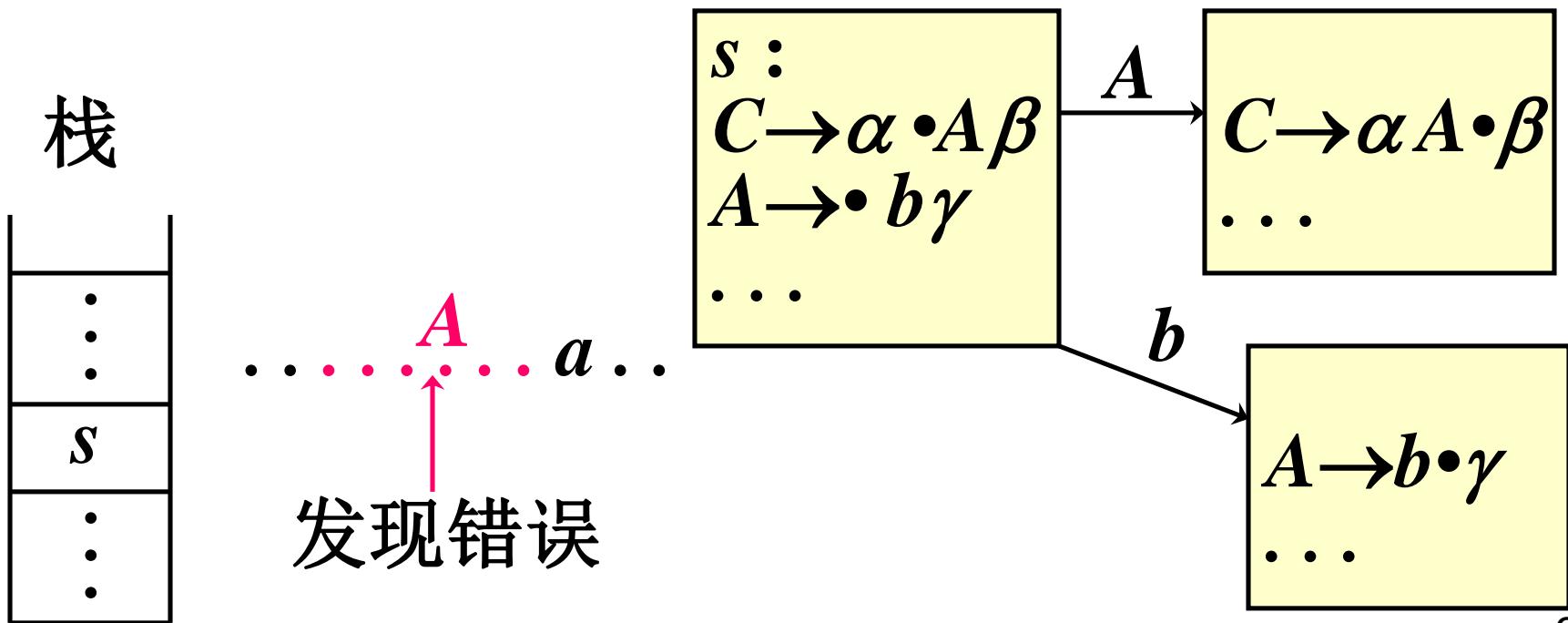
□ LR分析器在什么情况下发现错误

- 访问动作表时若遇到出错条目
- 访问转移表时它决不会遇到出错条目
- 决不会把不正确的后继移进栈
- 规范的LR分析器甚至在报告错误之前决不做任何无效归约



紧急方式错误恢复

- 退栈，直至出现状态 s ，它有预先确定的 A 的转移
- 抛弃若干个输入符号，直至找到 a ，它是 A 的合法后继
- 再把 A 和状态 $goto[s, A]$ 压进栈，恢复正常分析

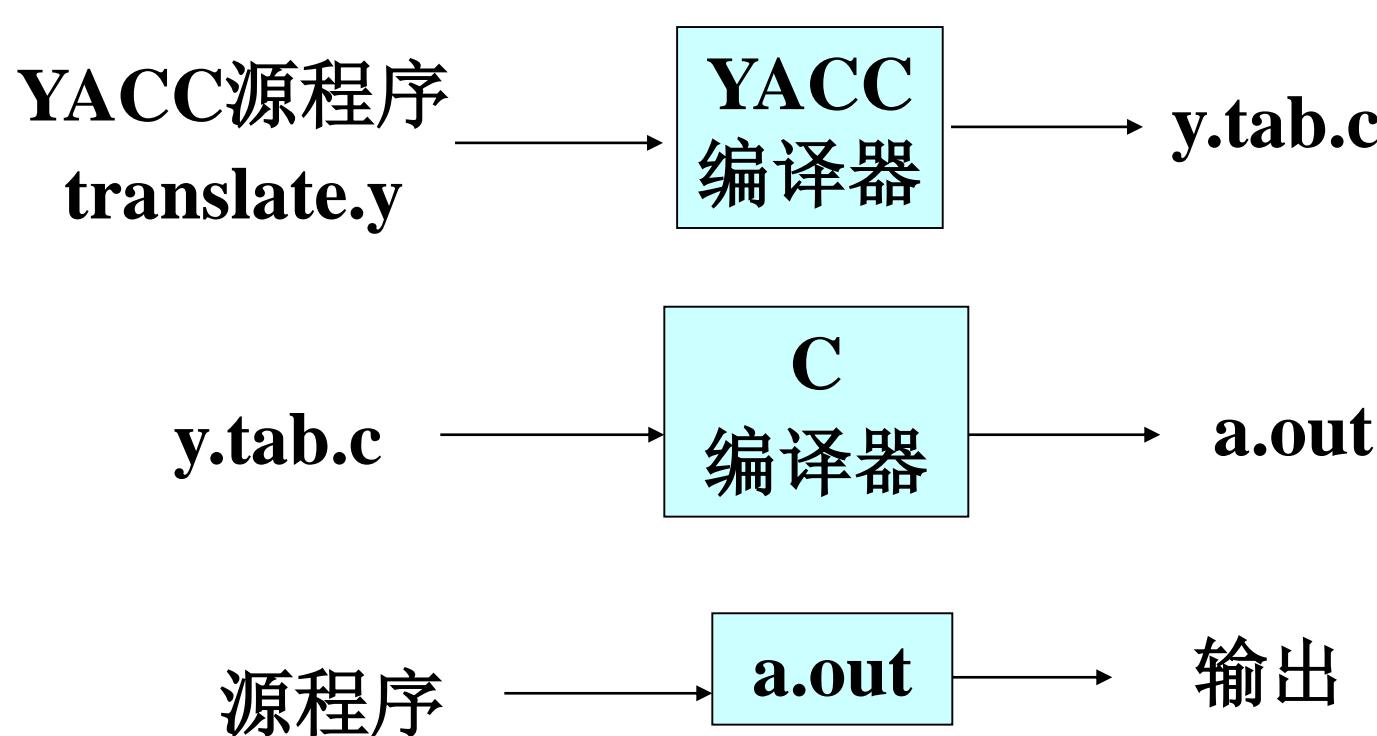




YACC: Yet Another Compiler-Compiler 语法分析器生成工具YACC



用生成器YACC构造翻译器的过程





YACC源程序的结构

□ YACC源程序有三部分组成

声明

%%

翻译规则

%%

C写的支持例程

例：台式计算器

$G[E]: E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow (E) \mid \text{digit}$

读入一个整数表达式，计算它的值并输出。



YACC源程序示例 (1/2)

```
%{  
$ include <ctype.h>  
%}  
% token digit  
  
%%  
line    : expr' '\n'      {printf("%d\n",$1);} ;  
expr   : expr '+' term   {$$=$1+$3;} ;  
       : term ;
```



YACC源程序示例 (2/2)

```
term      : term '*' factor    {$$=$1*$3;}  
          : factor  
          ;  
factor   : '(' expr ')'       {$$=$2;}  
          : digit  
          ;  
%%  
yylex(){ int c;  
          c=getchar();  
          if ( isdigit(c) ) { yyval = c-'0';  
                               return digit; }  
          return c; }
```



YACC源文件的说明

□ 声明部分

- 有任选的两节。
- 第一节处于分界符%{和%}之间，它是一些普通的C的声明；第二节是文法记号的声明。

□ 翻译规则部分

- 每条翻译规则由一个文法产生式和有关的语义动作组成。

□ 支持例程部分

- 一些C写的支持例程。
- 例：词法分析器，错误恢复例程等。



YACC中的冲突处理

- 缺省的处理方法
 - 对于归约/归约冲突，选择列在前面的产生式
 - 对于移进/规约冲突，总是移进（悬空-else的解决）
- 运行选项-v可以在文件y.output中描述冲突及其解决方法
- 可以通过一些命令来确定终结符号的优先级/结合性，解决移进/归约冲突。
 - 结合性：`%left` `%right` `%nonassoc`
 - 产生式的优先级
 - 它的最右终结符号的优先级。
 - 或者加标记`%prec<终结符号>`，指明产生式的优先级
 - 移进符号a/按照 $A \rightarrow a$ 归约：比较a和 $A \rightarrow a$ 的优先级，根据结合性，选择不同的操作。



YACC的错误恢复

- 使用错误产生式的方式来完成语法错误恢复
 - 错误产生式 $A \rightarrow \text{error } \alpha$
 - 例如: $\text{stmt} \rightarrow \text{error};$
- 首先定义哪些“主要”非终结符号有错误恢复动作;
 - 比如: 表达式, 语句, 块, 函数定义等非终结符号
- 当语法分析器碰到错误时
 - 不断弹出栈中状态, 直到栈顶状态包含 $A \rightarrow \cdot \text{error } \alpha;$
 - 分析器将 error 移到栈中。
 - 如果 α 为空, 分析器直接执行归约, 并调用相关的语义动作; 否则向前跳过一些符号, 找到可以归约为 α 的串为止。



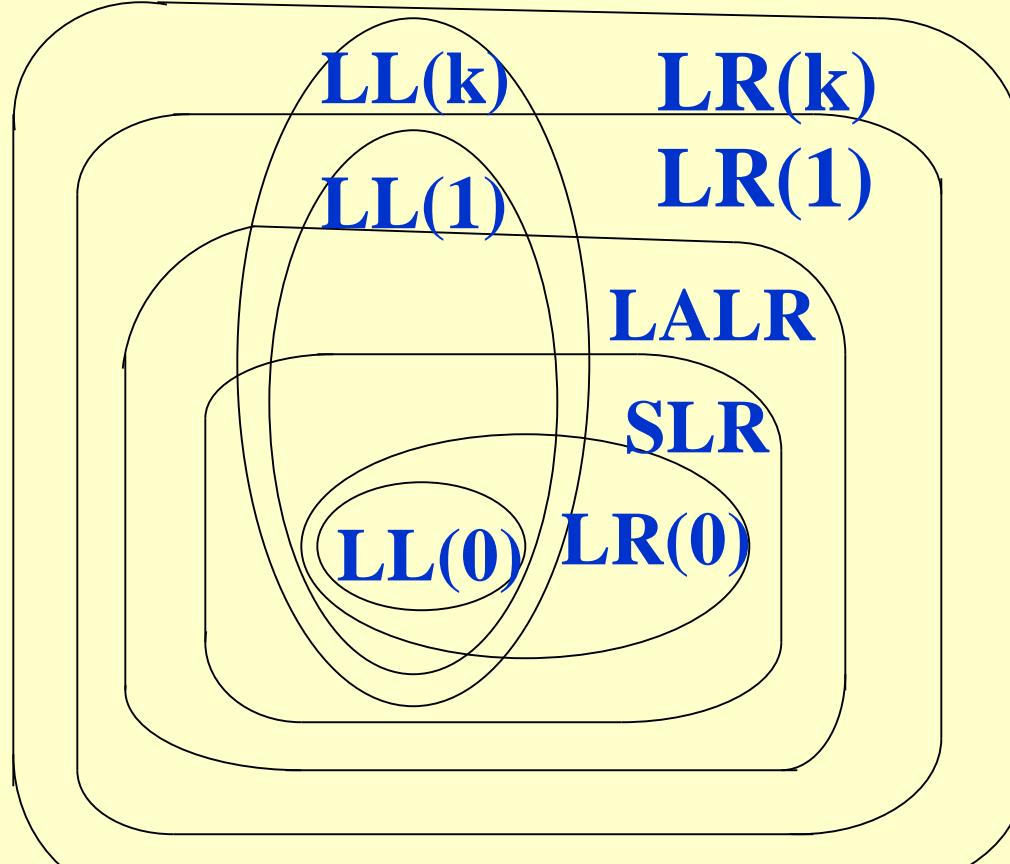
本章小结

- 上下文无关文法
- 自顶向下分析
 - 递归预测分析（递归子程序法）
 - 非递归预测分析——LL(1)
 - 注意：首先消除左递归和提取左公因子。
- 自底向上分析
 - LR分析: SLR(1), LR(1), LALR(1)



文法类的谱系

无二义文法



二义文法



作业

- 10月25日交
- LR(1), LALR(1)
 - 4.7.1, 4.7.4, 4.7.5