



中国科学技术大学  
University of Science and Technology of China



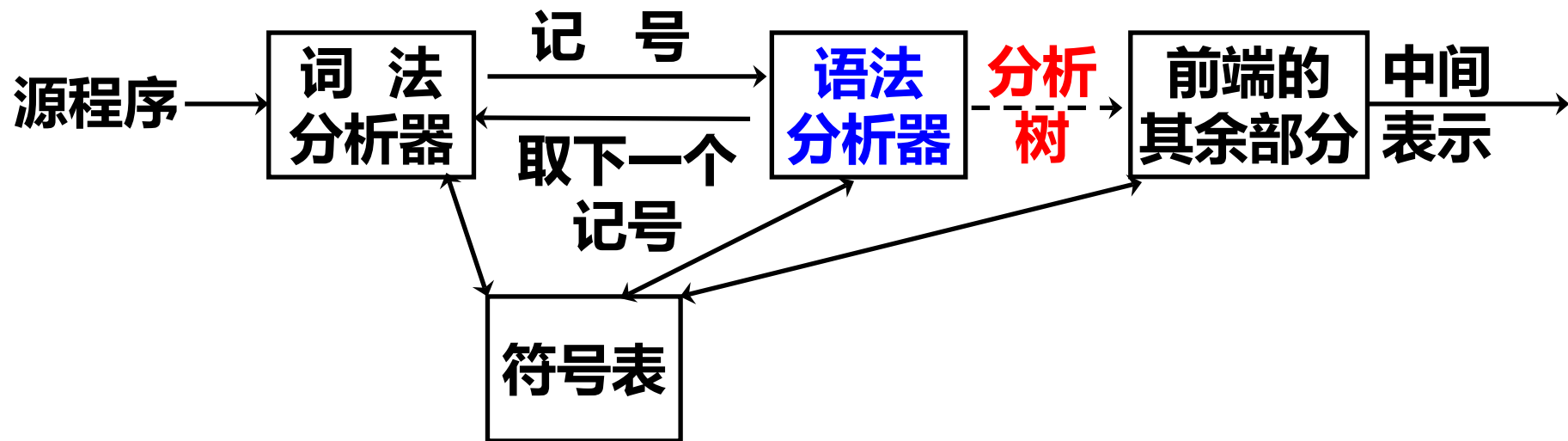
# 《编译原理与技术》

## 语法分析IV

计算机科学与技术学院

李 诚

2021-09-22



## □LR(k)分析技术

### ❖LR分析器的简单模型

➢ action, goto函数

### ❖简单的LR方法（简称SLR）

➢ 活前缀，识别活前缀的DFA/NFA，SLR算法

### ❖规范的LR方法

### ❖向前看的LR方法（简称LALR）



## □ 自顶向下 (Top-down)

- ❖ 针对输入串，从文法的开始符号出发，尝试根据产生式规则 **推导 (derive)** 出该输入串。
- ❖ **LL(1)** 文法及非递归预测分析方法
- ❖ **left-to-right scan** + **leftmost derivation**

## □ 自底向上 (Bottom-up)

- ❖ 针对输入串，尝试根据产生式规则 **归约 (reduce)** 到文法的开始符号。
- ❖ **LR(k)** 文法及其分析器
- ❖ **left-to-right scan** + **rightmost derivation**



$s_j$ : 总结了栈中该状态以下的信息

$X_i$ : 代表文法符号

栈

$S_m$
$X_m$
$S_{m-1}$
$X_{m-1}$
...
$S_0$

输入

$a_1$	...	$a_i$	...	$a_n$	$\$$
-------	-----	-------	-----	-------	------

LR分析程序

输出

action

goto

$action[s_m, a_i]$ : 移进 | 归约 | 接受 | 出错  
 $goto[s_{m-r}, A]=s_j$ : 移进  $A$  和  $s_j$  (归约后使用)



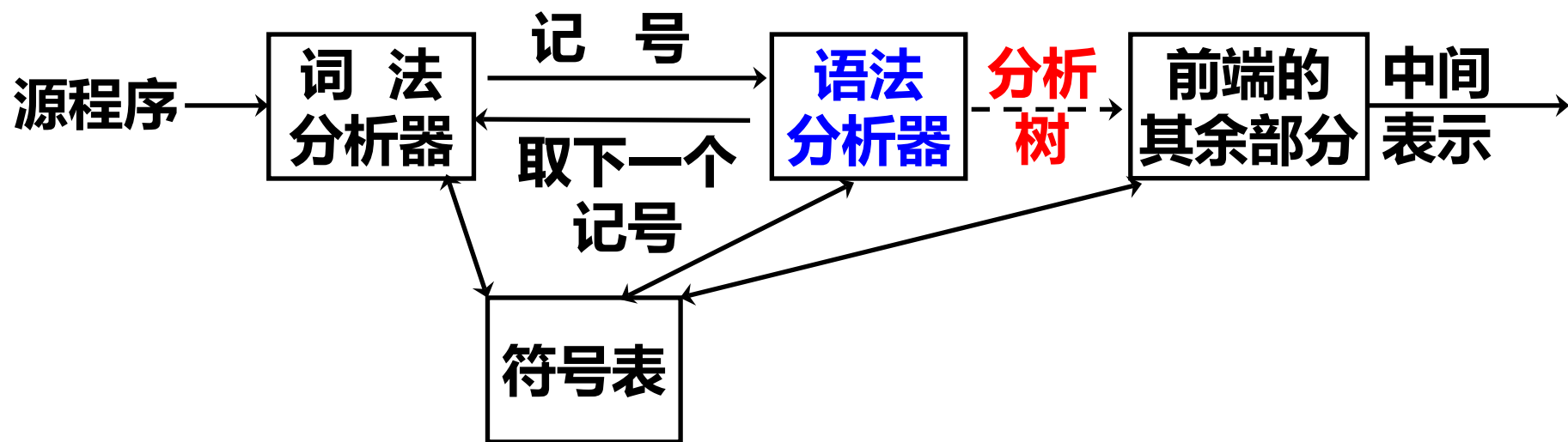
## □关键在于构造LR分析表

### ❖计算所有可能的状态

- 每一个状态描述了语法分析过程中所处的位置
- 可确定正在分析的产生式集合
- 可确定句柄形成的中间步骤

### ❖明确状态之前的**跳转关系**

### ❖明确状态与输入之间对应的**移进或者归约操作**



## □LR(k)分析技术

❖ LR分析器的简单模型

❖ 简单的LR方法（简称SLR）

➤ 活前缀，识别活前缀的DFA/NFA，SLR算法

❖ 规范的LR方法

❖ 向前看的LR方法（简称LALR）



## □ LR语法分析的每一步都形成一个格局config

$(s_0X_1s_1X_2s_2\cdots X_ms_m, a_ia_{i+1}\cdots a_n\$)$

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1X_2\cdots X_ma_ia_{i+1}\cdots a_n$
- ❖  $X_1X_2\cdots X_m$  是最右句型的一个前缀
- ❖ 每一个前缀都对应一个状态，因此，找出所有可能在栈里出现的前缀，就可以确定所有的状态
- ❖ 状态之间的转换  $\Leftrightarrow$  前缀之间的转换
- ❖ 在栈顶为  $s$ ，下一个字符为  $a$  的格局下，前缀为  $p$ 
  - 何时移进？当  $p$  包含句柄的一部分且存在  $p' = pa$
  - 何时归约？当  $p$  包含整个句柄时



□ 活前缀或可行前缀 (viable prefix):

❖ 最右句型的前缀, 该前缀不超过最右句柄的右端

$$S \Rightarrow_{rm}^* \gamma A w \Rightarrow_{rm} \gamma \beta w$$

❖  $\gamma\beta$ 的任何前缀 (包括 $\varepsilon$ 和 $\gamma\beta$ 本身) 都是活前缀

❖ 都出现在栈顶





## 栈中可能出现的串：

$a$

$ab$

$aA$

$aAb$

$aAbc$

$aAd$

$aAB$

$aABe$

$S$

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

活前缀：

最右句型的前缀，该前缀不超过最右句柄的右端

$$S \Rightarrow_{rm}^* \gamma A w \Rightarrow_{rm} \gamma \beta w$$

$\gamma\beta$  的任何前缀（包括 $\varepsilon$ 和 $\gamma\beta$ 本身）都是一个活前缀。



## 栈中可能出现的串：

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

$a$

$a\underline{b}$

← 出现句柄 (对应  $A \rightarrow b$ )

$aA$

$aAb$

$a\underline{Abc}$

← 出现句柄 (对应  $A \rightarrow Abc$ )

$aA\underline{d}$

← 出现句柄 (对应  $B \rightarrow d$ )

$aAB$

$\underline{aABe}$

← 出现句柄 (对应  $S \rightarrow aABe$ )

$S$

- 活前缀已含有句柄，表明产生式  $A \rightarrow \beta$  的右部  $\beta$  已出现在栈顶。



## 栈中可能出现的串:

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

$a$

$ab$

$a\underline{A}$

$a\underline{A}b$

$aAbc$

$aAd$

$a\underline{AB}$

$aABe$

$S$

出现产生式  $A \rightarrow Abc$  右端的一部分,  
期望从输入串中看到  $bc$

出现产生式  $A \rightarrow Abc$  右端的一部分,  
期望从输入串中看到  $c$

出现产生式  $S \rightarrow aABe$  的右端一部分,  
期望从输入串中看到  $e$

- 活前缀已含有句柄, 表明产生式  $A \rightarrow \beta$  的右部  $\beta$  已出现在栈顶。
- 活前缀只含句柄的一部分符号如  $\beta_1$  表明  $A \rightarrow \beta_1\beta_2$  的右部子串  $\beta_1$  已出现在栈顶, 当前期待从输入串中看到  $\beta_2$  推出的符号。



□ 栈中的文法符号总是形成一个活前缀

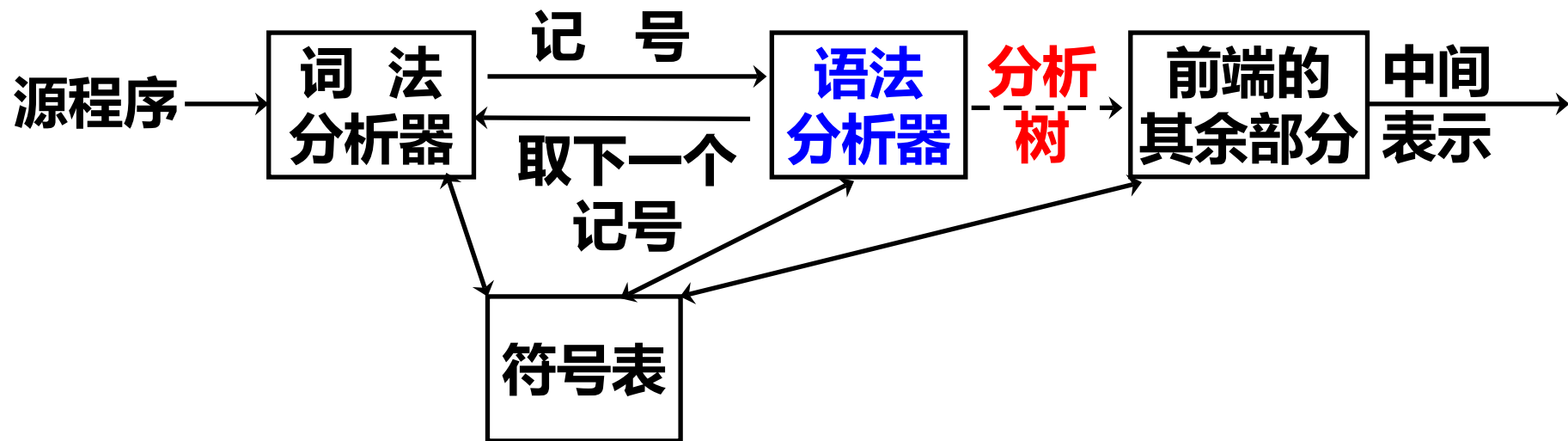
□ 分析表的转移函数本质上是识别活前缀的DFA

下表蓝色部分构成识别活前缀DFA的状态转换表

状态	动 作						转 移		
	id	+	*	(	)	\$	<i>E</i>	<i>T</i>	<i>F</i>
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



- 栈中的文法符号总是形成一个活前缀
- 分析表的转移函数本质上是识别活前缀的DFA
- 栈顶的状态符号包含确定句柄所需的一切信息
- 是已知的最一般的无回溯的移进-归约方法
- 能分析的文法类是预测分析法能分析的文法类的真超集
- 能及时发现语法错误
- 手工构造分析表的工作量太大



## □LR(k)分析技术

❖ LR分析器的简单模型

❖ 简单的LR方法（简称SLR）

➤ 活前缀，识别活前缀的DFA/NFA，SLR算法

❖ 规范的LR方法

❖ 向前看的LR方法（简称LALR）



□SLR (Simple LR)

□LR(0)项目 (简称项目)

项代表了一个可能的  
前缀

❖在右部的某个地方加点的产生式

❖加点的目的是用来表示分析过程中的状态

□例  $A \rightarrow XYZ$  对应应有四个项目

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

$A \rightarrow XYZ \cdot$

点的左边代表历史信息,  
点的右边代表展望信息。

□例  $A \rightarrow \varepsilon$  只有一个项目和它对应

$A \rightarrow \cdot$



□从文法构造识别活前缀的DFA

□从上述DFA构造分析表





## 1. 拓（增）广文法 (augmented grammar)

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow ( E ) \mid \text{id}$$

当且仅当分析器使用  $E' \rightarrow E$   
归约时，宣告分析成功



## 2. 构造LR(0)项目集规范族

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot \text{id}$

求项目集的闭包closure(I)

闭包函数closure(I)

1、I的每个项目均加入closure(I)

2、如果 $A \rightarrow \alpha \cdot B \beta$ 在 closure(I)中，且 $B \rightarrow \gamma$ 是产生式，那么如果项目 $B \rightarrow \cdot \gamma$ 还不在于closure(I)中的话，那么把它加入。



## 2. 构造LR(0)项目集规范族

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot \text{id}$

核心项目: 初始项目 ( $E' \rightarrow \cdot E$ ) 或者  
点不在最左边的项

非核心项目: 不是初始项, 且点在最左边

可以通过对核心项目求闭包来获得  
为节省存储空间, 可省去



## 2. 构造LR(0)项目集规范族

$$\begin{array}{ccc}
 I_0: & \xrightarrow{E} & I_1: \\
 E' \rightarrow \cdot E & & E' \rightarrow E \cdot \\
 E \rightarrow \cdot E + T & & E \rightarrow E \cdot + T \\
 E \rightarrow \cdot T & & \\
 T \rightarrow \cdot T * F & & \\
 T \rightarrow \cdot F & & \\
 F \rightarrow \cdot (E & & \\
 F \rightarrow \cdot id & & 
 \end{array}$$

$$I_1 := \text{goto}(I_0, E)$$

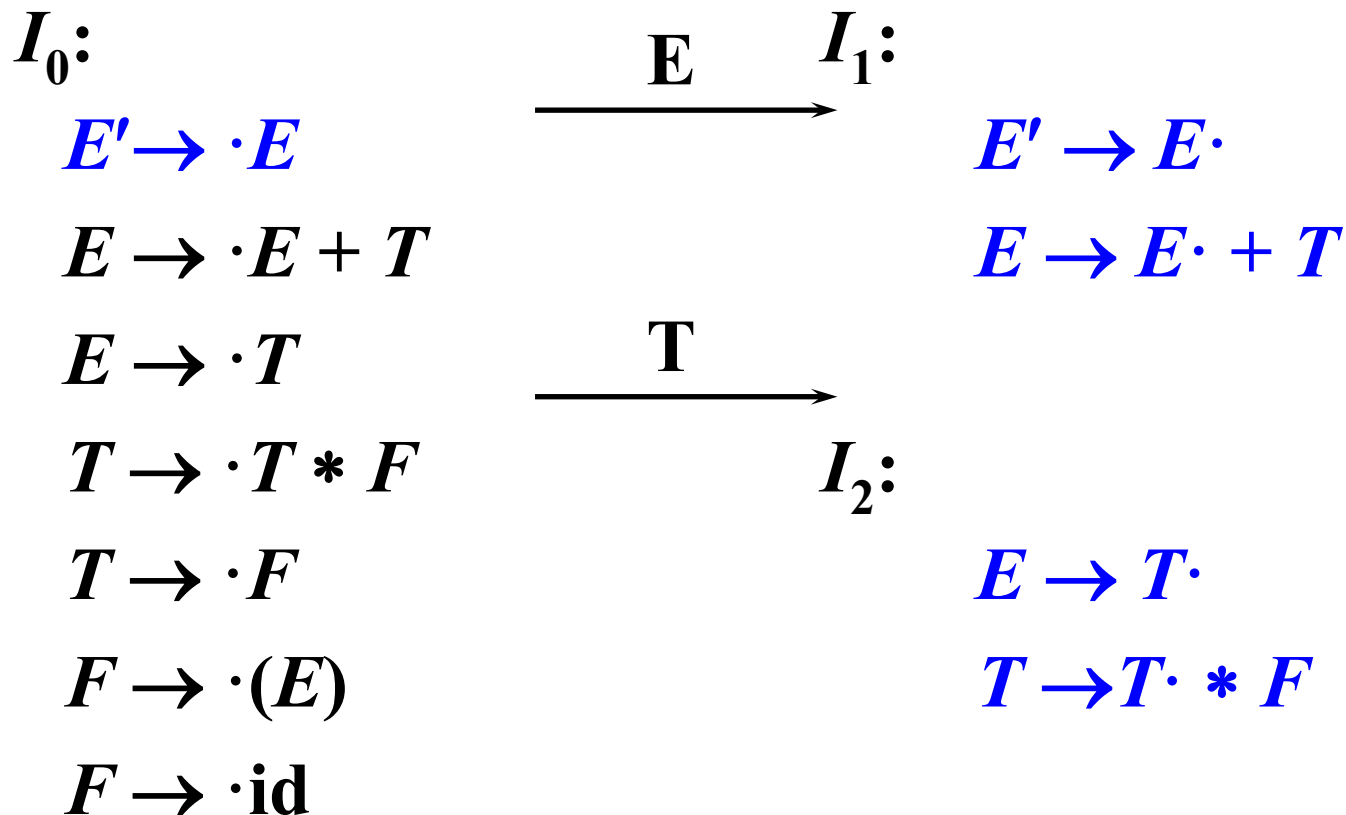
求项目集I和文法符号X的  $I' = \text{goto}(I, X)$   
(当输入为X时离开I状态后的转换)

1、  $I' = \emptyset$

2、对于I中的每一项  $A \rightarrow \alpha \cdot X \beta$ ,  
 $I' = I' \cup \text{closure}(A \rightarrow \alpha X \cdot \beta)$

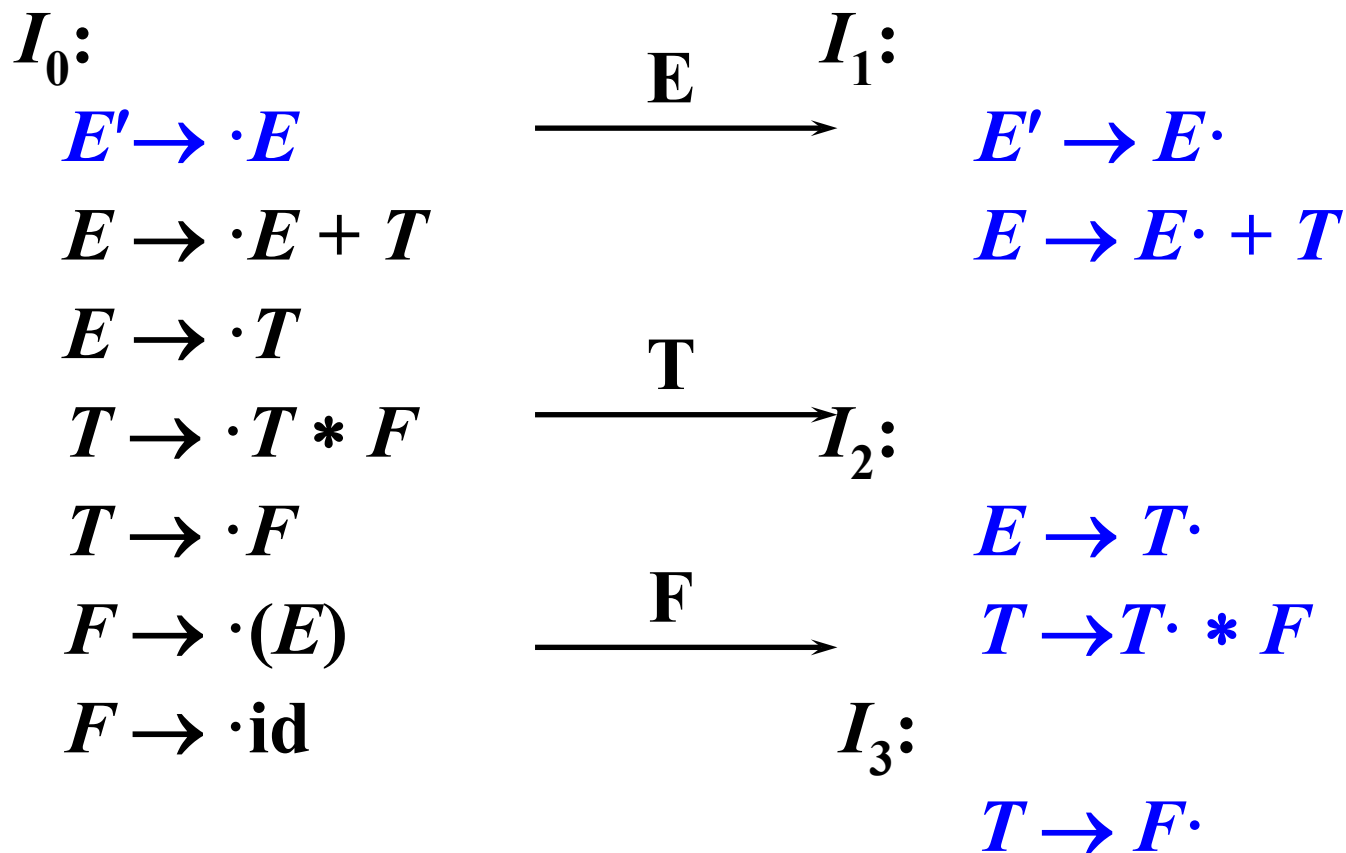


## 2. 构造LR(0)项目集规范族





## 2. 构造LR(0)项目集规范族



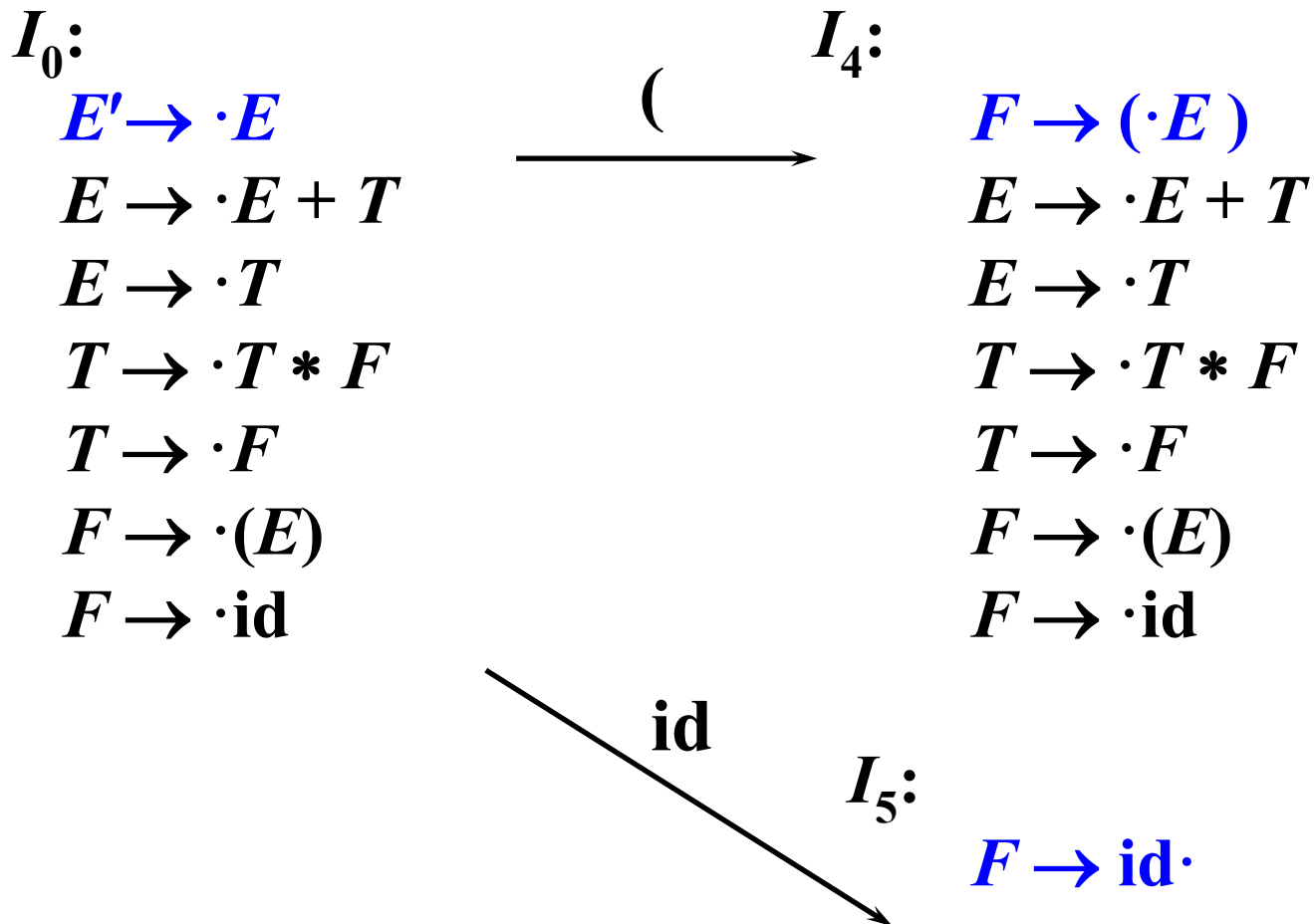


## 2. 构造LR(0)项目集规范族

$I_0:$	$($	$I_4:$
$E' \rightarrow \cdot E$	$\longrightarrow$	$F \rightarrow (\cdot E)$
$E \rightarrow \cdot E + T$		$E \rightarrow \cdot E + T$
$E \rightarrow \cdot T$		$E \rightarrow \cdot T$
$T \rightarrow \cdot T * F$		$T \rightarrow \cdot T * F$
$T \rightarrow \cdot F$		$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$		$F \rightarrow \cdot (E)$
$F \rightarrow \cdot \text{id}$		$F \rightarrow \cdot \text{id}$



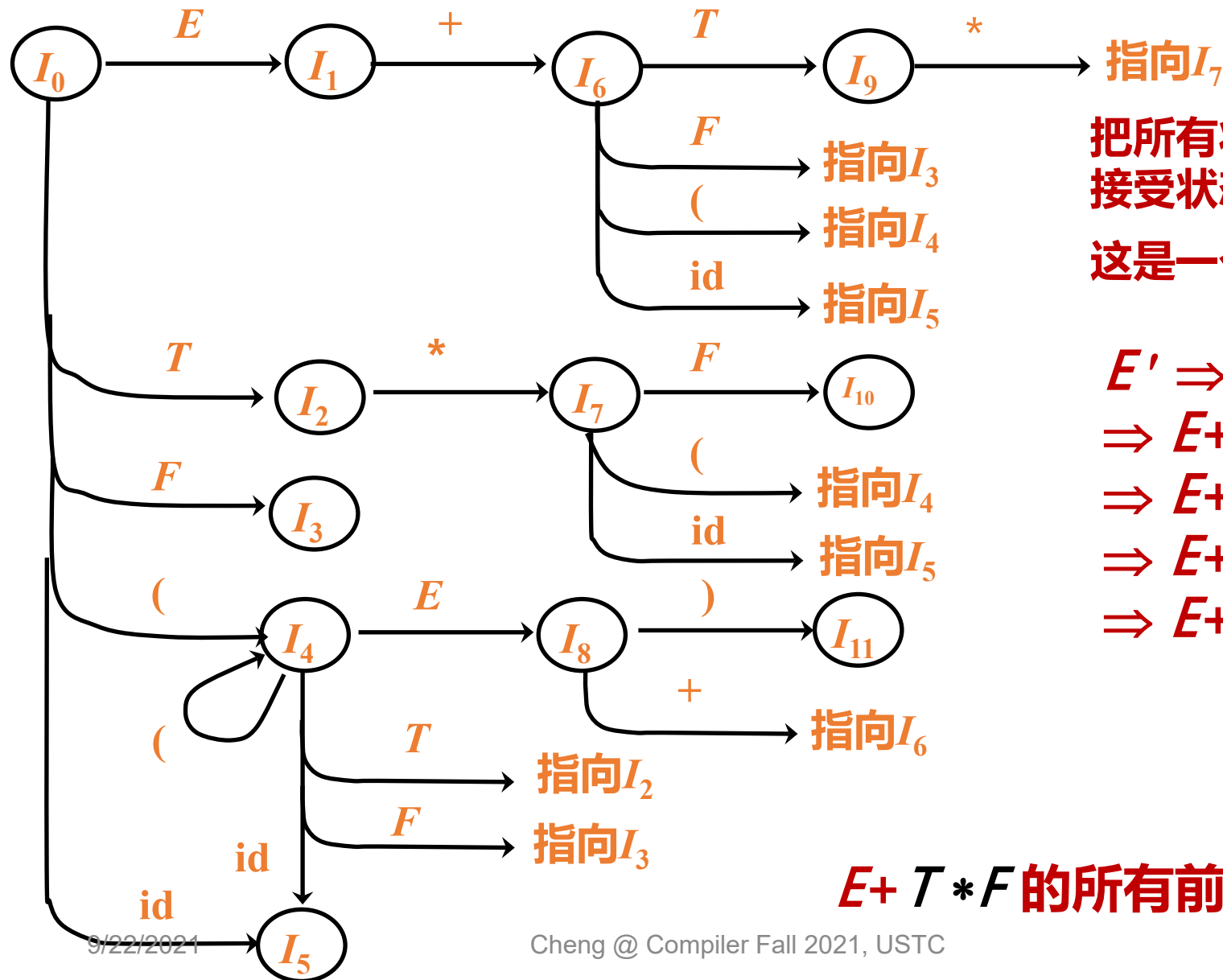
## 2. 构造LR(0)项目集规范族







# 构造识别活前缀的DFA



把所有状态都作为  
接受状态

这是一个DFA

$E' \Rightarrow E$   
 $\Rightarrow E + T$   
 $\Rightarrow E + T * F$   
 $\Rightarrow E + T * id$   
 $\Rightarrow E + T * F * id$

$E + T * F$  的所有前缀都可接受



□从文法构造识别活前缀的DFA

□从上述DFA构造分析表



□ 状态  $i$  从  $I_i$  构造，它的 **action 函数** 如下确定：

- ❖ 如果  $[A \rightarrow \alpha \cdot a \beta]$  在  $I_i$  中，并且  $\text{goto}(I_i, a) = I_j$ ，那么置  $\text{action}[i, a]$  为  $sj$
- ❖ 如果  $[A \rightarrow \alpha \cdot]$  在  $I_i$  中，那么对  $\text{FOLLOW}(A)$  中的所有  $a$ ，置  $\text{action}[i, a]$  为  $rj$ ， $j$  是产生式  $A \rightarrow \alpha$  的编号
- ❖ 如果  $[S' \rightarrow S \cdot]$  在  $I_i$  中，那么置  $\text{action}[i, \$]$  为接受  $\text{acc}$
- ❖ 上面的  $a$  是终结符

□ 如果出现动作冲突，那么该文法就不是SLR(1)文法



□ 状态  $i$  从  $I_i$  构造，它的 *action* 函数如下确定：

❖ 此处省略，参见上页

□ 使用下面规则构造状态  $i$  的 *goto* 函数：

❖ 对所有的非终结符  $A$ ，如果  $goto(I_i, A) = I_j$ ，那么  
 $goto[i, A] = j$



□ 状态  $i$  从  $I_i$  构造，它的 *action* 函数如下确定：

❖ 此处省略，参见上页

□ 使用下面规则构造状态  $i$  的 *goto* 函数：

❖ 此处省略，参见上页

□ 分析器的**初始状态**是包含  $[S' \rightarrow \cdot S]$  的项目集对应的状态

不能由上面两步定义的条目都置为 **error**



例 (1)  $E \rightarrow E + T$  (2)  $E \rightarrow T$   
 (3)  $T \rightarrow T * F$  (4)  $T \rightarrow E$   
 (5)  $F \rightarrow ( E )$  (6)  $F \rightarrow \text{id}$

*si* 移进当前输入符号和状态 *i*  
*rj* 按第 *j* 个产生式进行归约  
*acc* 接受

状态	动作 action						转移 goto		
	id	+	*	(	)	\$	<i>E</i>	<i>T</i>	<i>F</i>
0	<i>s5</i>				<i>s4</i>		1	2	3
1		<i>s6</i>				<i>acc</i>			
2		<i>r2</i>	<i>s7</i>		<i>r2</i>	<i>r2</i>			
3		<i>r4</i>	<i>r4</i>		<i>r4</i>	<i>r4</i>			
4	<i>s5</i>				<i>s4</i>		8	2	3
5		<i>r6</i>	<i>r6</i>		<i>r6</i>	<i>r6</i>			
6	<i>s5</i>				<i>s4</i>			9	3



- 一个上下文无关文法 $G$ ，通过上述算法构造出SLR语法分析表，且表项中**没有移进/归约或者归约/归约冲突**，那么 $G$ 就是SLR(1)文法。
- 1代表了当看到某个产生式右部时，只需要再向前看1个符号就可决定是否用该式进行归约。
- 通常可以省略1，写作SLR文法



□依据上述SLR(1)分析表

□参照slide 9 -25的分析方法

❖文法符号栈

❖输入缓冲区

❖选择的行为

➤移进、归约、接受、报错





# LR分析算法：举例



栈	输 入	动 作
0	id * id + id \$	



# LR分析算法：举例



栈	输 入	动 作
0	<b>id * id + id \$</b>	<b>移进 (查action表)</b>



# LR分析算法：举例



栈	输 入	动 作
0	id * id + id \$	移进
0 id 5	* id + id \$	



# LR分析算法：举例



栈	输入	动作
0	id * id + id \$	移进
0 id 5	* id + id \$	按 $F \rightarrow id$ 归约
		<p>1. 查 <math>action[5, *] \Rightarrow</math> 归约</p> <p>2. 执行归约 (<math>F \rightarrow \alpha</math>):</p> <ul style="list-style-type: none"> <li>• 从栈中弹出 <math> \alpha </math> 个 <math>\langle \text{状态}, \text{符号} \rangle</math> 对</li> <li>• 查 <math>goto[0, F] \Rightarrow 3</math></li> <li>• 将 <math>(F, 3)</math> 压入栈</li> </ul>



□例  $I_2$ :

(2)  $E \rightarrow T \cdot$

(3)  $T \rightarrow T \cdot * F$

❖ 归约：因为  $\text{FOLLOW}(E) = \{\$, +, )\}$ ,

所以  $\text{action}[2, \$] = \text{action}[2, +] = \text{action}[2, )] = r2$

❖ 移进：因为圆点在中间，且点后面是终结符，  
所以， $\text{action}[2, *] = s7$



- LR(0)自动机刻画了可能出现在文法符号栈中的所有串;
- 栈中的内容一定是某个最右句型的前缀;
- 但是不是所有前缀都会出现在栈中。

$$E \Rightarrow_{rm}^* F * \text{id} \Rightarrow_{rm} (E) * \text{id}$$

- 栈中只能出现(, (E, (E), 而不会出现(E)\*
  - ❖ 因为看到\*时, (E)是句柄, 会被归约成为F



□如果  $S' \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta_1 \beta_2 w$ , 那么就说明项目

$A \rightarrow \beta_1 \cdot \beta_2$  对活前缀  $\alpha \beta_1$  是有效的

❖ 一个项目可能对好几个活前缀都是有效的

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \text{id}$$

项  $E \rightarrow \cdot E + T$  对  $\varepsilon$  和  $($  这两个活前缀都有效

$$E' \Rightarrow E \Rightarrow E + T \quad (\alpha, \beta_1 \text{ 都为空})$$

$$E' \Rightarrow E \Rightarrow (E) \Rightarrow (E + T) \quad (\alpha = "(", \beta_1 \text{ 为空})$$

该DFA读过 $\varepsilon$ 和 $($ 后到达不同的状态,  
那么项目  $E \rightarrow \cdot E + T$  就出现在对应的不同项目集中



□如果  $S' \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta_1 \beta_2 w$ , 那么就说明项目

$A \rightarrow \beta_1 \cdot \beta_2$  对活前缀  $\alpha \beta_1$  是有效的

❖ 一个项目可能对好几个活前缀都是有效的

➤ 如果  $\beta_2 \neq \epsilon$ , 应该移进

➤ 如果  $\beta_2 = \epsilon$ , 应该用产生式  $A \rightarrow \beta_1$  归约





□如果 $S' \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta_1 \beta_2 w$ , 那么就说明项目

$A \rightarrow \beta_1 \cdot \beta_2$ 对活前缀 $\alpha \beta_1$ 是有效的

❖ 一个项目可能对好几个活前缀都是有效的

❖ 一个活前缀可能有多个有效项目

一个活前缀 $\gamma$ 的**有效项目集**就是

从这个DFA的初态出发, 沿着标记为 $\gamma$ 的路径到达的那个项目集 (状态)

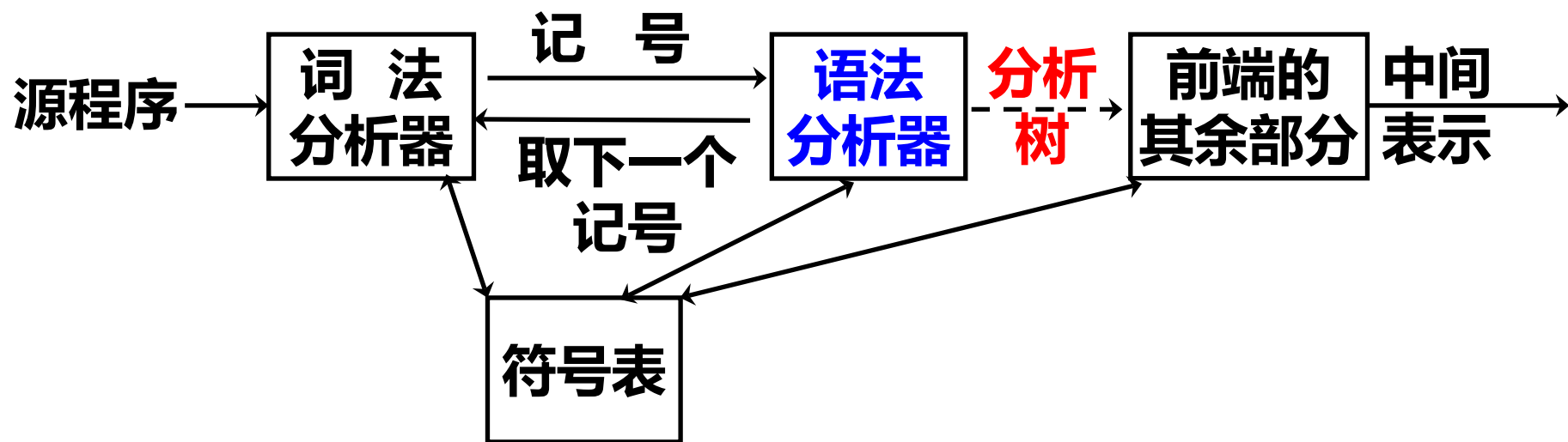


□例 串 $E + T *$ 是活前缀，读完它后，DFA处于状态 $I_7$

$I_7: \quad T \rightarrow T * \cdot F, F \rightarrow \cdot (E), F \rightarrow \cdot \text{id}$

$E' \Rightarrow E$	$E' \Rightarrow E$	$E' \Rightarrow E$
$\Rightarrow E + T$	$\Rightarrow E + T$	$\Rightarrow E + T$
$\Rightarrow E + T * F$	$\Rightarrow E + T * F$	$\Rightarrow E + T * F$
$\Rightarrow E + T * \text{id}$	$\Rightarrow E + T * (E)$	$\Rightarrow E + T * \text{id}$
$\Rightarrow E + T * F * \text{id}$		

包含活前缀的最右推导，且  
 $I_7$ 中所有的项目对该活前缀是有效的



## □LR(k)分析技术

- ❖ LR分析器的简单模型
- ❖ 简单的LR方法（简称SLR）
  - 活前缀，识别活前缀的DFA/NFA，SLR算法
- ❖ 规范的LR方法
- ❖ 向前看的LR方法（简称LALR）



□目标：在识别活前缀DFA的状态中，增加信息，排除一些不正确的归约操作

□方法：添加了前向搜索符

❖一个项目  $A \rightarrow \alpha \cdot \beta$ ，如果最终用这个产生式进行归约之后，期望看见的符号是  $a$ ，则这个加点项的前向搜索符是  $a$ 。

❖上述项目可以写成：  $A \rightarrow \alpha \cdot \beta, a$

□与SLR(1)分析的区别

❖项目集的定义发生了改变：  $LR(0) \Rightarrow LR(1)$

❖ $\text{closure}(I)$  和  $\text{GOTO}$  函数需要修改



## □LR(1)项目:

$$[A \rightarrow \alpha \cdot \beta, a]$$

- ❖ 当项目由两个分量组成，第一分量为SLR中的项，第二分量为搜索符（向前看符号）
- ❖ LR(1)中的1代表了搜索符 $a$ 的长度

## □使用注意事项:

- ❖ 当 $\beta$ 不为空时， $a$ 不起作用
- ❖ 当 $\beta$ 为空时，如果下一个输入符号是 $a$ ，将按照 $A \rightarrow \alpha$ 进行归约
  - $a$ 的集合是FOLLOW( $A$ )的子集



## □LR(1)项目:

$$[A \rightarrow \alpha \cdot \beta, a]$$

- ❖当项目由两个分量组成，第一分量为SLR中的项，第二分量为搜索符（向前看符号）
- ❖LR(1)中的1代表了搜索符 $a$ 的长度

## □LR(1)项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对活前缀 $\gamma$ 有效:

- ❖如果存在着推导 $S \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$ ，其中：
  - $\gamma = \delta \alpha$ ;
  - $a$ 是 $w$ 的第一个符号，或者 $w$ 是 $\epsilon$ 且 $a$ 是 $\$$



□例  $S \rightarrow BB$

$B \rightarrow bB \mid a$

LR(1)项目  $[A \rightarrow \alpha \cdot \beta, a]$  对活前缀  $\gamma$  有效：  
存在着推导  $S \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$ ，其中：  
 $\gamma = \delta \alpha$ ;  
 $a$  是  $w$  的第一个符号，或者  $w$  是  $\epsilon$  且  $a$  是  $\$$

从最右推导  $S \Rightarrow_{rm}^* bbBba \Rightarrow_{rm} bbbBba$  看出：

令  $A = B, \alpha = b, \beta = B, \delta = bb, \gamma = \delta \alpha = bbb, w = ba$

$[B \rightarrow b \cdot B, b]$  对活前缀  $\gamma = bbb$  是有效的



## □构造LR(1)项目集规范族

❖也就是构造识别活前缀的DFA

## □构造规范的LR分析表

❖状态之间的转换关系





## □基础运算1: 计算闭包CLOSURE(I)

❖ I中的任何项目都属于CLOSURE(I)

❖ 若有项目  $[A \rightarrow \alpha \cdot B \beta, a]$  在CLOSURE(I)中, 而  $B \rightarrow \gamma$  是文法中的产生式,  $b$  是FIRST( $\beta a$ )中的元素, 则  $[B \rightarrow \cdot \gamma, b]$  也属于CLOSURE(I)

保证在用  $B \rightarrow \gamma$  进行归约后,

- 出现的输入字符  $b$  是句柄  $\alpha B \beta$  中  $B$  的后继符号
- 或者是  $\alpha B \beta$  归约为  $A$  后可能出现的终结符。



## □基础运算2：通过GOTO(I,X)算CLOSURE(J)

- ❖ 将J置为空集
- ❖ 若有项目  $[A \rightarrow \alpha \cdot X \beta, a]$  在I中，那么将项目  $[A \rightarrow \alpha X \cdot \beta, a]$  放入J中
- ❖ 计算并返回CLOSURE(J)

注意：GOTO(I,X)中的X可以是终结符或非终结符



## □具体算法

❖ 初始项目集  $I_0$ :

$I_0 = \text{CLOSURE}(/S' \rightarrow \cdot S, \$/)$  将\$作为向前的搜索符

❖ 设C为最终返回的项目集族, 初始为  $C = \{I_0\}$

❖ 重复以下步骤

➤ 对C中的任意项目集I, 重复

- 对每一个文法符号X(终结符或非终结符)

- 如果  $\text{GOTO}(I, X) \neq \emptyset$  且  $\text{GOTO}(I, X) \notin C$ , 那么将  $\text{GOTO}(I, X)$  放入C

- 注: 上述  $\text{GOTO}(I, X)$  是上一页ppt中计算闭包的GOTO

➤ 当C中项目集不再增加为止



# 构造LR(1)项目集族： 举例



中国科学技术大学  
University of Science and Technology of China

$$S' \rightarrow \cdot S, \$ \quad I_0$$

步骤一：从初始项开始



# 构造LR(1)项目集族： 举例



中国科学技术大学  
University of Science and Technology of China

$$\begin{array}{l} S' \rightarrow \cdot S, \$ \\ S \rightarrow \cdot BB \end{array} \quad I_0$$

步骤二：计算非核心项目的  
第一个分量



# 构造LR(1)项目集族：举例



中国科学技术大学  
University of Science and Technology of China

$$\begin{array}{l} S' \rightarrow \cdot S, \$ \\ S \rightarrow \cdot BB, \$ \end{array} I_0$$

步骤三：通过FIRST( $\epsilon$ \$) 计算非核心项目的第二个分量



# 构造LR(1)项目集族： 举例



中国科学技术大学  
University of Science and Technology of China

$$\begin{array}{l} S' \rightarrow \cdot S, \$ \\ S \rightarrow \cdot BB, \$ \\ B \rightarrow \cdot bB \\ B \rightarrow \cdot a \end{array} \quad I_0$$

步骤二：计算非核心项目的  
第一个分量



# 构造LR(1)项目集族： 举例



中国科学技术大学  
University of Science and Technology of China

$S' \rightarrow \cdot S, \$$   $I_0$

$S \rightarrow \cdot BB, \$$

$B \rightarrow \cdot bB, b/a$

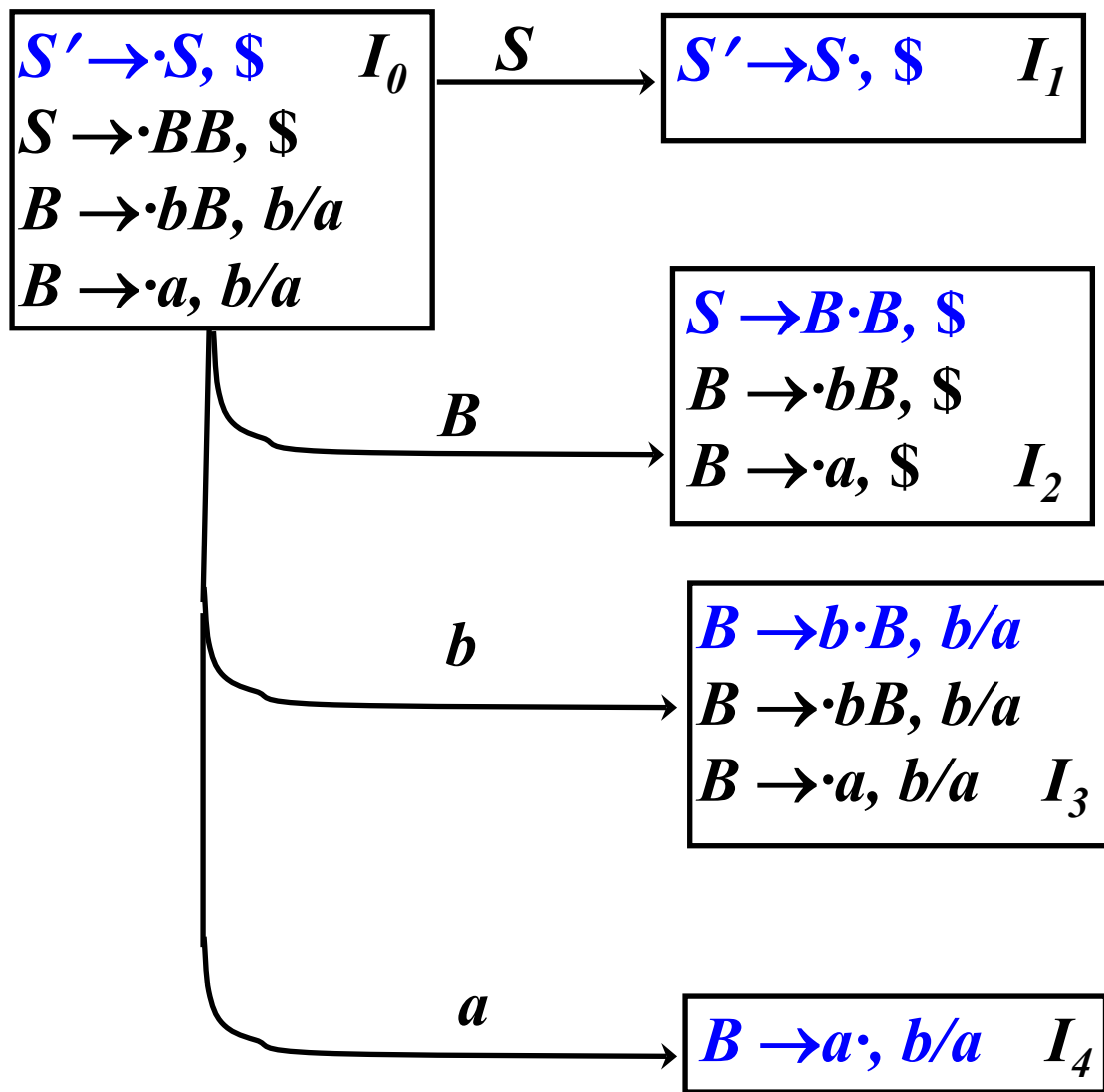
$B \rightarrow \cdot a, b/a$

步骤三：通过FIRST(B\$) 计算非核心项目的第二个分量



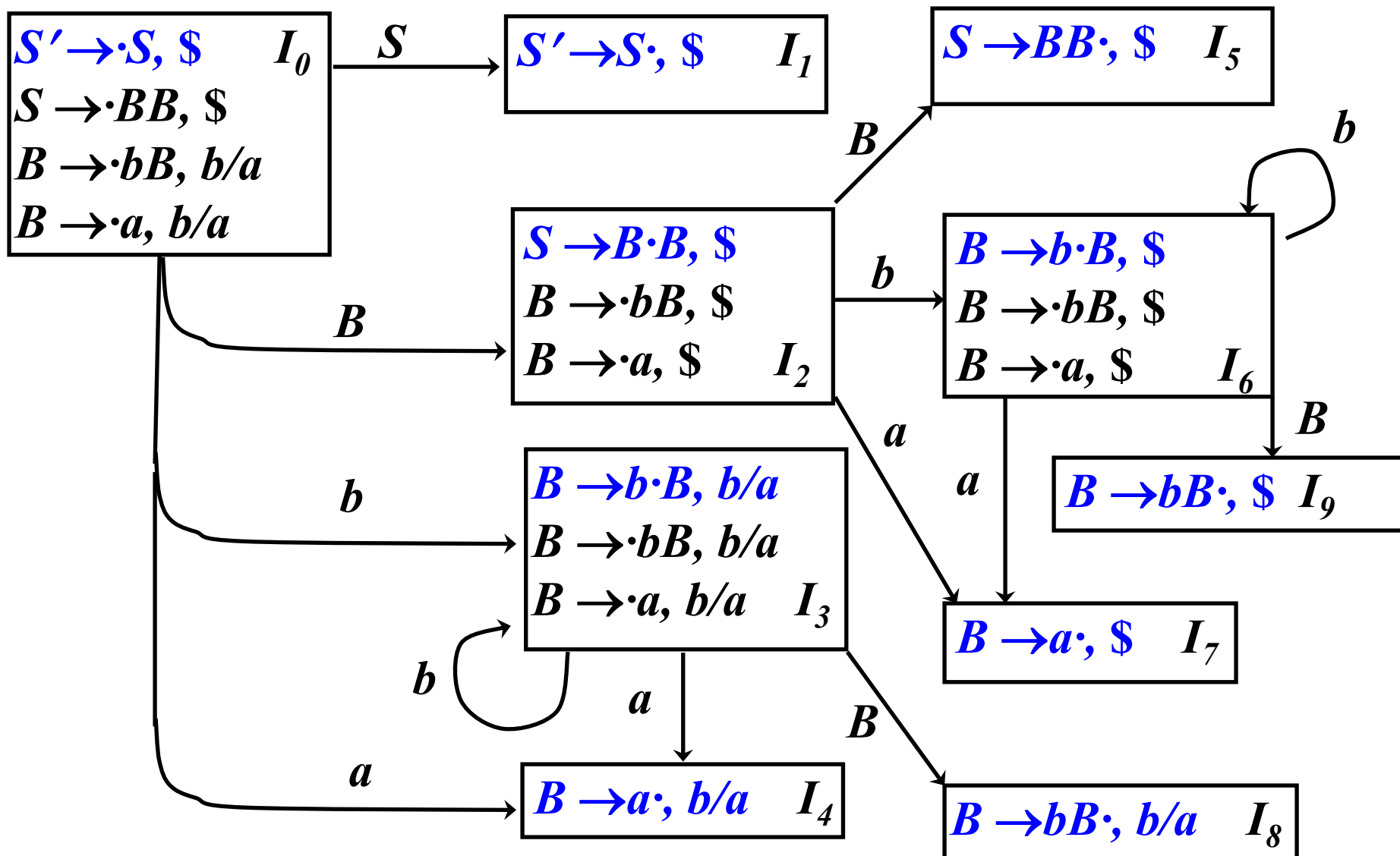


# 构造LR(1)项目集族：举例





# 构造LR(1)项目集族：举例





## □构造识别拓广文法 $G'$ 活前缀的DFA

❖基于LR(1)项目族来构造

## □状态 $i$ 的 $action$ 函数如下确定:

❖如果 $[A \rightarrow \alpha \cdot a\beta, b]$ 在 $I_i$ 中, 且 $goto(I_i, a) = I_j$ , 那么置 $action[i, a]$ 为 $sj$  (此时, 不看 $b$ )

❖如果 $[A \rightarrow \alpha \cdot, a]$ 在 $I_i$ 中, 且 $A \neq S'$ , 那么置 $action[i, a]$ 为 $rj$  (此时, 不再看 $FOLLOW(A)$ )

❖如果 $[S' \rightarrow S \cdot, \$]$ 在 $I_i$ 中, 那么置 $action[i, \$] = acc$

如果上述构造出现了冲突, 那么文法就不是LR(1)的



□构造识别拓广文法 $G'$ 活前缀的DFA

□状态 $i$ 的 $action$ 函数如下确定:

❖参见上页ppt

□状态 $i$ 的 $goto$ 函数如下确定:

❖如果 $goto(I_i, A) = I_j$ , 那么 $goto[i, A] = j$



□构造识别拓广文法 $G'$ 活前缀的DFA

□状态 $i$ 的 $action$ 函数如下确定:

❖参见上页ppt

□状态 $i$ 的 $goto$ 函数如下确定:

❖如果 $goto(I_i, A) = I_j$ , 那么 $goto[i, A] = j$

□分析器的初始状态是包含 $[S' \rightarrow \cdot S, \$]$ 的项目集对应的状态

用上面规则未能定义的所有条目都置为 **error**



# 非SLR(1)文法-revisit



$S \rightarrow V = E$   
 $S \rightarrow E$   
 $V \rightarrow * E$   
 $V \rightarrow \text{id}$   
 $E \rightarrow V$

$I_0$ :  
 $S' \rightarrow \cdot S$   
 $S \rightarrow \cdot V = E$   
 $S \rightarrow \cdot E$   
 $V \rightarrow \cdot * E$   
 $V \rightarrow \cdot \text{id}$   
 $E \rightarrow \cdot V$

$V$

$I_2$ :  
 $S \rightarrow V \cdot = E$   
 $E \rightarrow V \cdot$

$=$

$I_6$ :  
 $S \rightarrow V = \cdot E$   
 $E \rightarrow \cdot V$   
 $V \rightarrow \cdot * E$   
 $V \rightarrow \cdot \text{id}$

项目  $S \rightarrow V \cdot = E$  使得  
 $\text{action}[2, =] = \text{s6}$

项目  $E \rightarrow V \cdot$  使得  
 $\text{action}[2, =] = \text{r5}$   
因为  $\text{Follow}(E) = \{=, \$\}$

产生移进-归约冲突,  
但该文法不是二义的。



# 非SLR(1)但是LR(1)文法



$S \rightarrow V = E$

$S \rightarrow E$

$V \rightarrow * E$

$V \rightarrow \text{id}$

$E \rightarrow V$

$I_0:$   
 $S' \rightarrow \cdot S, \$$



# 非SLR(1)但是LR(1)文法



$S \rightarrow V = E$

$S \rightarrow E$

$V \rightarrow * E$

$V \rightarrow \text{id}$

$E \rightarrow V$

$I_0:$   
 $S' \rightarrow \cdot S, \$$

计算闭包:

定义里:  $[A \rightarrow \alpha \cdot B \beta, a]$

这里:  $[S' \rightarrow \epsilon \cdot S \epsilon, \$]$

$\text{FIRST}(\beta a)$



$\text{FIRST}(\epsilon \$) = \{\$ \}$





# 非SLR(1)但是LR(1)文法



$$S \rightarrow V = E$$

$$S \rightarrow E$$

$$V \rightarrow * E$$

$$V \rightarrow \text{id}$$

$$E \rightarrow V$$

$$I_0:$$

$$S' \rightarrow \cdot S, \$$$

$$S \rightarrow \cdot V = E, \$$$

$$S \rightarrow \cdot E, \$$$

计算闭包:

定义里:  $[A \rightarrow \alpha \cdot B \beta, a]$

这里:  $[S' \rightarrow \epsilon \cdot S \epsilon, \$]$

$\text{FIRST}(\beta a)$



$\text{FIRST}(\epsilon \$) = \{\$ \}$



# 非SLR(1)但是LR(1)文法



$$S \rightarrow V = E$$

$$S \rightarrow E$$

$$V \rightarrow * E$$

$$V \rightarrow \text{id}$$

$$E \rightarrow V$$

$$I_0:$$

$$S' \rightarrow \cdot S, \$$$

$$S \rightarrow \cdot V = E, \$$$

$$S \rightarrow \cdot E, \$$$

计算闭包:

定义里:  $[A \rightarrow \alpha \cdot B \beta, a]$

这里:  $[S \rightarrow \varepsilon \cdot V = E, \$]$

FIRST( $\beta a$ )



FIRST( $= E \$$ ) =  $\{=\}$



# 非SLR(1)但是LR(1)文法



$$S \rightarrow V = E$$

$$S \rightarrow E$$

$$V \rightarrow * E$$

$$V \rightarrow \text{id}$$

$$E \rightarrow V$$

$I_0$ :

$$S' \rightarrow \cdot S, \$$$

$$S \rightarrow \cdot V = E, \$$$

$$S \rightarrow \cdot E, \$$$

$$V \rightarrow \cdot * E, =$$

$$V \rightarrow \cdot \text{id}, =$$

计算闭包:

定义里:  $[A \rightarrow \alpha \cdot B \beta, a]$

这里:  $[S \rightarrow \varepsilon \cdot V = E, \$]$

FIRST( $\beta a$ )



FIRST( $= E \$$ ) =  $\{=\}$



# 非SLR(1)但是LR(1)文法



$$S \rightarrow V = E$$

$$S \rightarrow E$$

$$V \rightarrow * E$$

$$V \rightarrow \text{id}$$

$$E \rightarrow V$$

$I_0$ :

$$S' \rightarrow \cdot S, \$$$

$$S \rightarrow \cdot V = E, \$$$

$$S \rightarrow \cdot E, \$$$

$$V \rightarrow \cdot * E, =$$

$$V \rightarrow \cdot \text{id}, =$$

计算闭包:

定义里:  $[A \rightarrow \alpha \cdot B \beta, a]$

这里:  $[S \rightarrow \varepsilon \cdot V = E, \$]$

$\text{FIRST}(\beta a)$



$\text{FIRST}(= E\$) = \{=\}$



# 非SLR(1)但是LR(1)文法



$$S \rightarrow V = E$$

$$S \rightarrow E$$

$$V \rightarrow * E$$

$$V \rightarrow \text{id}$$

$$E \rightarrow V$$

$I_0$ :

$$S' \rightarrow \cdot S, \$$$

$$S \rightarrow \cdot V = E, \$$$

$$S \rightarrow \cdot E, \$$$

$$V \rightarrow \cdot * E, =$$

$$V \rightarrow \cdot \text{id}, =$$

$$E \rightarrow \cdot V, \$$$

计算闭包:

定义里:  $[A \rightarrow \alpha \cdot B \beta, a]$

这里:  $[S \rightarrow \epsilon \cdot E \epsilon, \$]$

$\text{FIRST}(\beta a)$



$\text{FIRST}(\epsilon \$) = \{\$ \}$



# 非SLR(1)但是LR(1)文法



$S \rightarrow V = E$

$S \rightarrow E$

$V \rightarrow * E$

$V \rightarrow \text{id}$

$E \rightarrow V$

$I_0:$

$S' \rightarrow \cdot S, \$$

$S \rightarrow \cdot V = E, \$$

$S \rightarrow \cdot E, \$$

$V \rightarrow \cdot * E, =$

$V \rightarrow \cdot \text{id}, =$

$E \rightarrow \cdot V, \$$

$V \rightarrow \cdot * E, \$$

$V \rightarrow \cdot \text{id}, \$$



# 非SLR(1)但是LR(1)文法


$$S \rightarrow V = E$$
$$S \rightarrow E$$
$$V \rightarrow * E$$
$$V \rightarrow \text{id}$$
$$E \rightarrow V$$

$I_0$ :

$$S' \rightarrow \cdot S, \$$$
$$S \rightarrow \cdot V = E, \$$$
$$S \rightarrow \cdot E, \$$$
$$V \rightarrow \cdot * E, =/\$$$
$$V \rightarrow \cdot \text{id}, =/\$$$
$$E \rightarrow \cdot V, \$$$

可通过合并搜索符简化



# 非SLR(1)但是LR(1)文法



$S \rightarrow V = E$   
 $S \rightarrow E$   
 $V \rightarrow * E$   
 $V \rightarrow \text{id}$   
 $E \rightarrow V$

$I_0:$   
 $S' \rightarrow \cdot S, \$$   
 $S \rightarrow \cdot V = E, \$$   
 $S \rightarrow \cdot E, \$$   
 $V \rightarrow \cdot * E, =/\$$   
 $V \rightarrow \cdot \text{id}, =/\$$   
 $E \rightarrow \cdot V, \$$

$V \longrightarrow$

$I_2:$   
 $S \rightarrow V \cdot = E, \$$   
 $E \rightarrow V \cdot, \$$

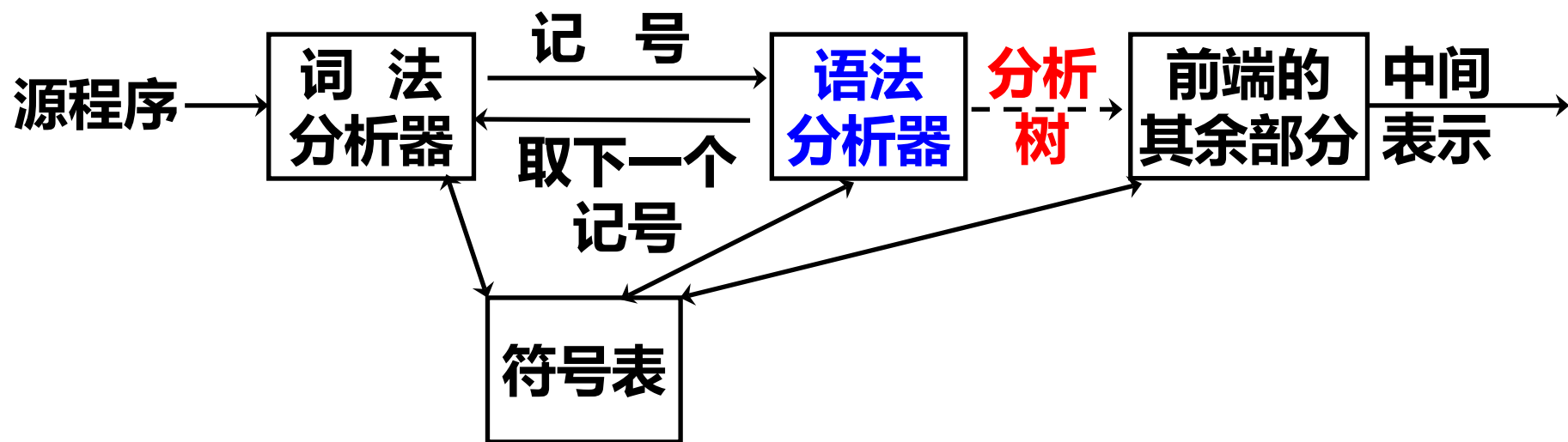
项目  $[S \rightarrow V \cdot = E, \$]$  使得  
 $\text{action}[2, =] = \text{s6}$

项目  $[E \rightarrow V \cdot, \$]$  使得  
 $\text{action}[2, \$] = \text{r5}$

因为  $\{\$, \$\}$  是  $\text{Follow}(E) = \{=, \$\}$  的真子集

每一个SLR(1)文法都是LR(1)的





## □LR(k)分析技术

- ❖ LR分析器的简单模型
- ❖ 简单的LR方法（简称SLR）
  - 活前缀，识别活前缀的DFA/NFA，SLR算法
- ❖ 规范的LR方法
- ❖ 向前看的LR方法（简称LALR）



## □研究LALR的原因

规范LR分析表的状态数偏多

## □LALR特点

- ❖ LALR和SLR的分析表有同样多的状态，比规范LR分析表要小得多
- ❖ LALR的能力介于SLR和规范LR之间
- ❖ LALR的能力在很多情况下已经够用

## □LALR分析表构造方法

- ❖ 通过合并规范LR(1)项目集来得到



□ 合并识别 LR(1)文法的活前缀的DFA中的相同  
核心项目集(**同心项目集**, 注意: 不是项)

□ 同心的LR(1)项目集

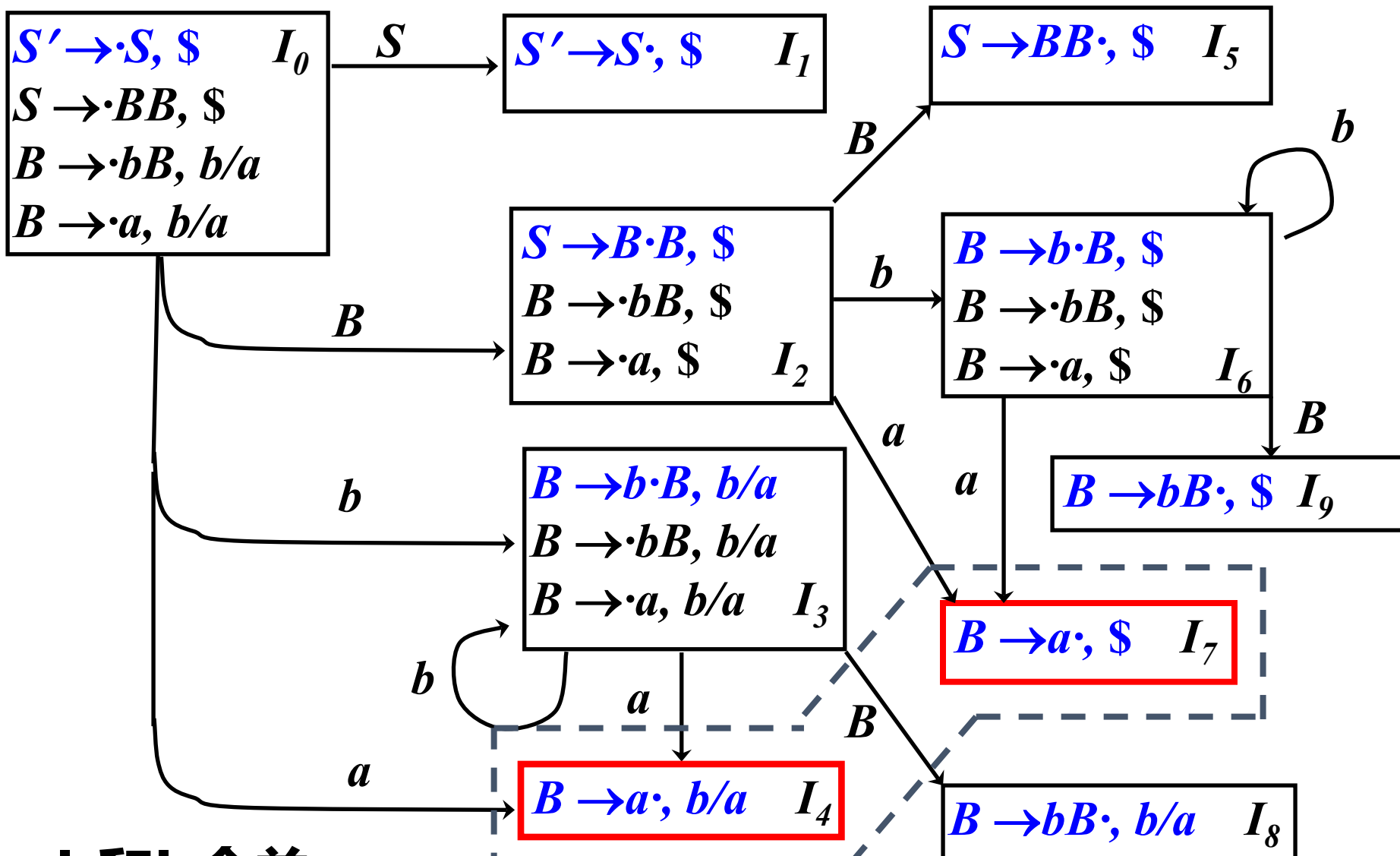
❖ **核心**: 项目集中第一分量的集合

❖ 略去搜索符后它们是相同的集合

❖ 例:  $[B \rightarrow \cdot bB, \$]$  与  $[B \rightarrow \cdot bB, b/a]$

第一分量

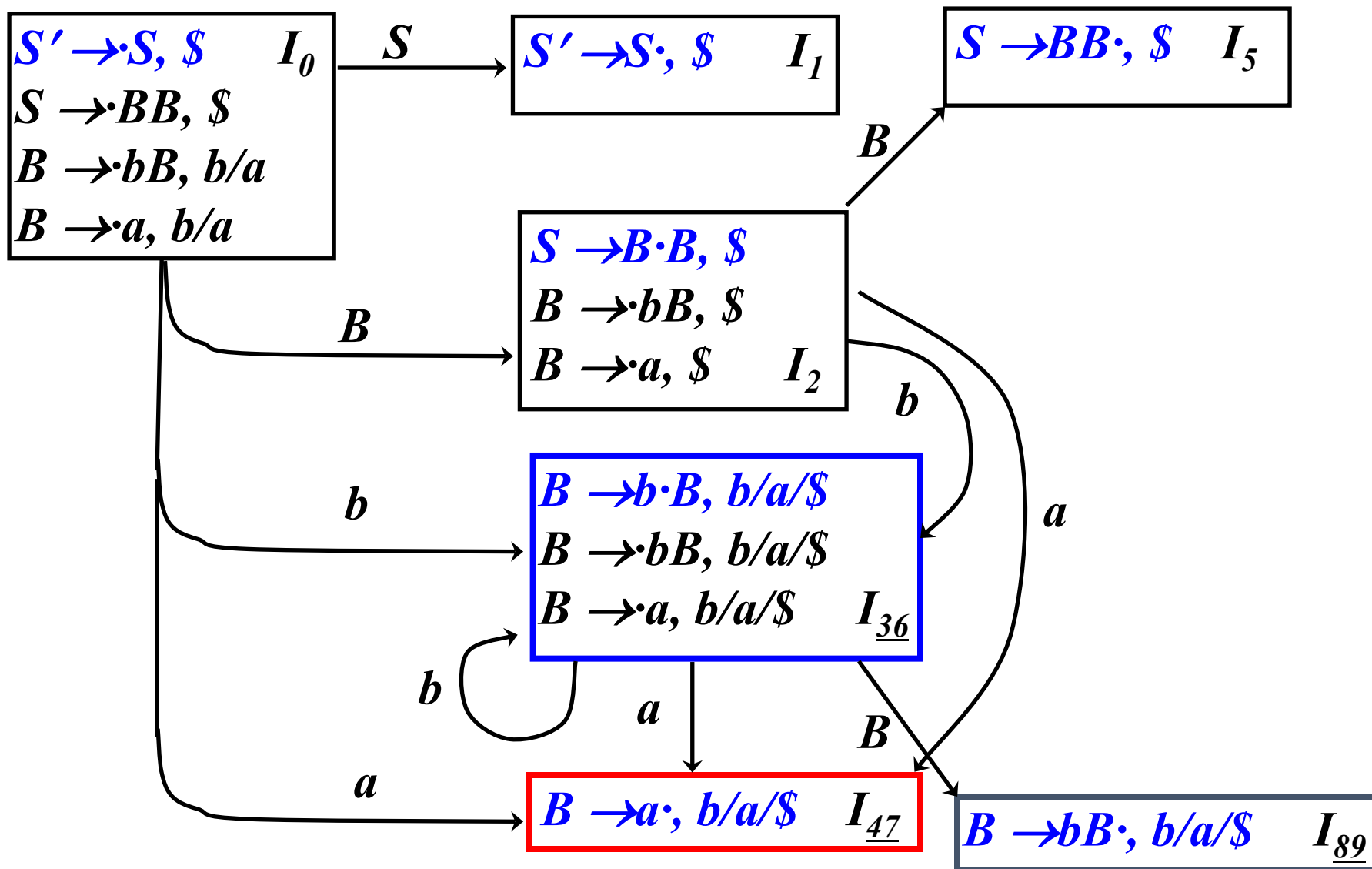
第二分量



$I_4$ 和 $I_7$ 合并



# 合并同心项目集





## 2、构造LALR(1)分析表

- ❖ 构造LR(1)项目集规范族  $C = \{I_0, I_1, \dots, I_n\}$
- ❖ 构造LALR(1)项目集规范族  $C' = \{J_0, J_1, \dots, J_k\}$ ,  
其中任意项目集  $J_i = I_n \cup I_m \cup \dots \cup I_t$ 
  - $I_n, I_m, \dots, I_t \in C$  且具有共同的核心
- ❖ 按构造规范LR(1)分析表的方式构造分析表

如没有语法分析动作冲突，那么给定文法就是  
**LALR(1)文法**

## □合并同心项目集可能会引起冲突

❖同心集的合并不会引起新的移进-归约冲突

项目集1

$[A \rightarrow \alpha \cdot, a]$

$[B \rightarrow \beta \cdot a \gamma, c]$

...

项目集2

$[B \rightarrow \beta \cdot a \gamma, b]$

$[A \rightarrow \alpha \cdot, d]$

...

如果有移进归约冲突，则合并前就有冲突



## □合并同心项目集可能会引起冲突

- ❖同心集的合并不会引起新的移进-归约冲突
- ❖同心集的**合并有可能产生**新的归约-归约冲突

$S' \rightarrow S$

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

$A \rightarrow c$

$B \rightarrow c$

对 $ac$ 有效的项目集

$A \rightarrow c ; d$ $B \rightarrow c ; e$
--

对 $bc$ 有效的项目集

$A \rightarrow c ; e$ $B \rightarrow c ; d$
--

合并同心集后

$A \rightarrow c ; d/e$ $B \rightarrow c ; d/e$
--

该文法是LR(1)的  
但不是LALR(1)的

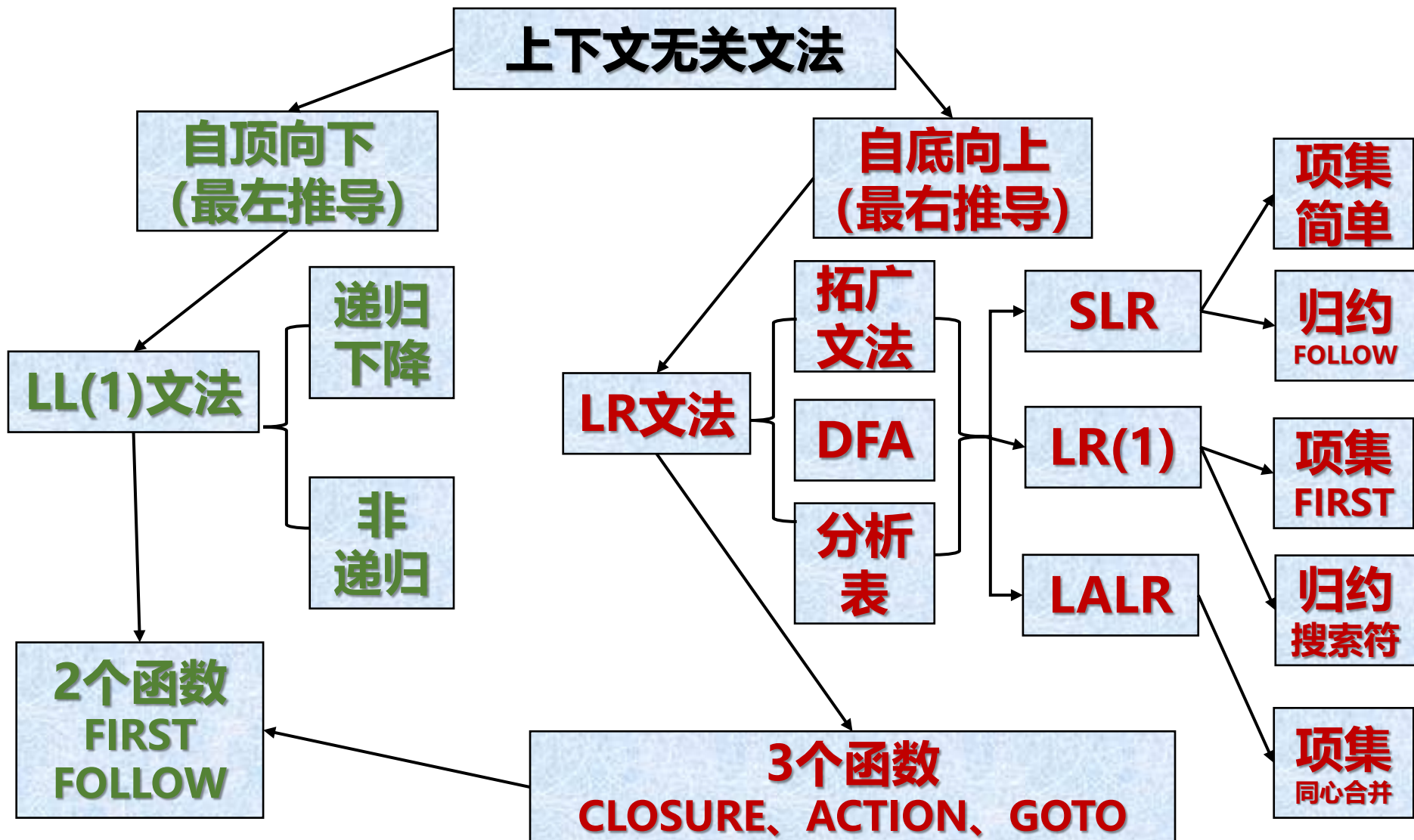




# LR分析法总结



		SLR	LALR	LR(1)
初始状态		$[S' \rightarrow \cdot S]$	$[S' \rightarrow \cdot S, \$]$	$[S' \rightarrow \cdot S, \$]$
项目集		LR(0) CLOSURE(I)	合并LR(1)项目集 族的同心项目集	LR(1), CLOSURE(I) 搜索符考虑 <b>FISRT</b> ( $\beta a$ )
动作	移进	$[A \rightarrow \alpha a \beta] \in I_i$ $GOTO(I_i, a) = I_j$ <b>ACTION</b> $[i, a] = sj$	与LR(1)一致	$[A \rightarrow \alpha a \beta, b] \in I_i$ $GOTO(I_i, a) = I_j$ <b>ACTION</b> $[i, a] = sj$
	归约	$[A \rightarrow \alpha] \in I_i, A \neq S'$ $a \in FOLLOW(A)$ <b>ACTION</b> $[i, a] = rj$	与LR(1)一致	$[A \rightarrow \alpha, a] \in I_i$ $A \neq S'$ <b>ACTION</b> $[i, a] = rj$
	接受	$[S' \rightarrow S \cdot] \in I_i$ <b>ACTION</b> $[i, \$] = acc$	与LR(1)一致	$[S' \rightarrow S \cdot, \$] \in I_i$ <b>ACTION</b> $[i, \$] = acc$
	出错	空白条目	与LR(1)一致	空白条目
GOTO		$GOTO(I_i, A) = I_j$ <b>GOTO</b> $[i, A] = j$	与LR(1)一致	$GOTO(I_i, A) = I_j$ <b>GOTO</b> $[i, A] = j$
状态量		少(几百)	与SLR一样	多(几千)





# LR和LL分析方法的比较



中国科学技术大学  
University of Science and Technology of China

	LR(1)方 法	LL(1)方 法
建立分析树	自底而上	自顶而下
归约or推导	规范归约	最左推导
决定使用产生式的时机	看见产生式整个右部推出的串后(句柄)	看见产生式推出的第一个终结符后
对文法的限制	无	无左递归、无公共左因子
分析表	状态 $\times$ 文法符号, 大	非终结符 $\times$ 终结符, 小
分析栈	状态栈, 信息更多	文法符号栈
确定句柄	根据栈顶状态和下一个符号便可以确定句柄和归约所用产生式	无句柄概念
语法错误	决不会将出错点后的符号移入分析栈	和LR一样, 决不会读过出错点而不报错