



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

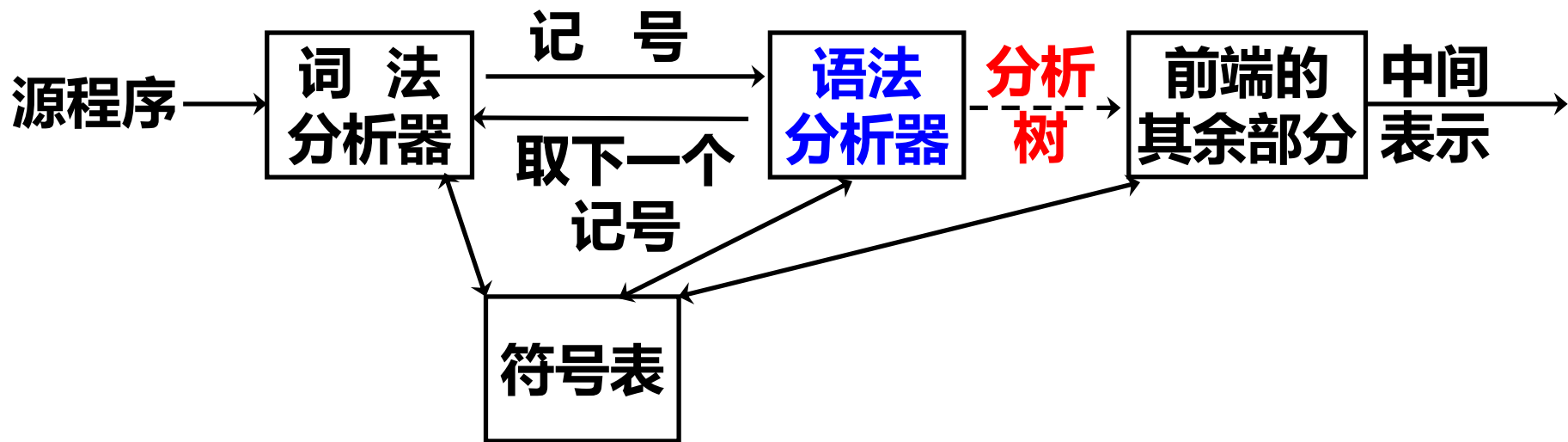
计算机科学与技术学院

李诚

29/09/2018



- **书面作业不允许补交，只判对错，不评分，但质量差的有诫勉谈话**
- **Lab作业允许迟交和补交，成绩会受到影响**
  - ❖ **Advice: submitting an incomplete version better than nothing, and optimizing it later**



## □ 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)

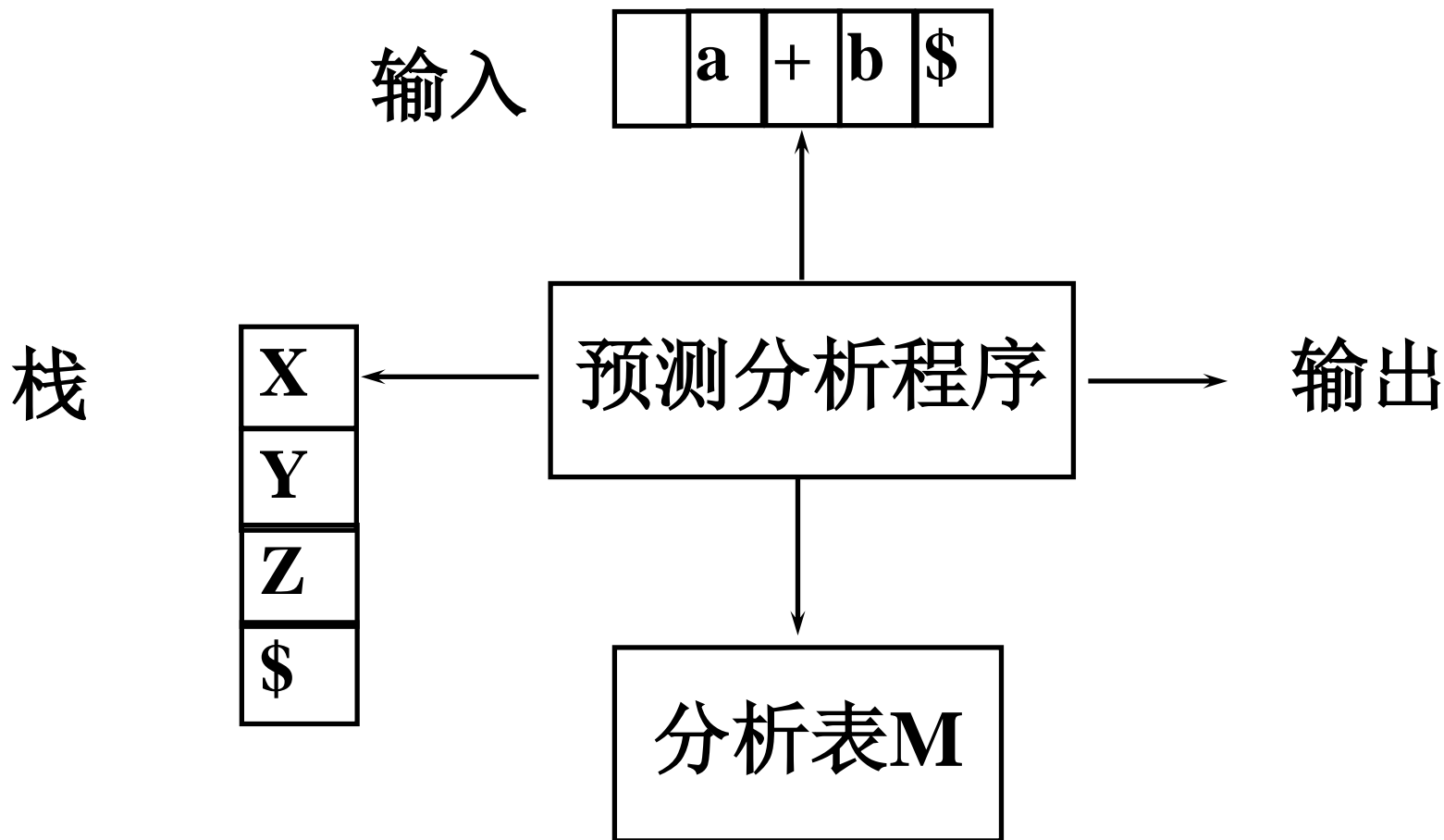
- ❖ 针对输入串，从文法的开始符号出发，尝试根据产生式规则**推导 (derive)**出该输入串。
- ❖ **LL(1)**文法及非递归预测分析方法
- ❖ **left-to-right scan** + **leftmost derivation**

## □ 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**



# 复习: LL(1)非递归分析



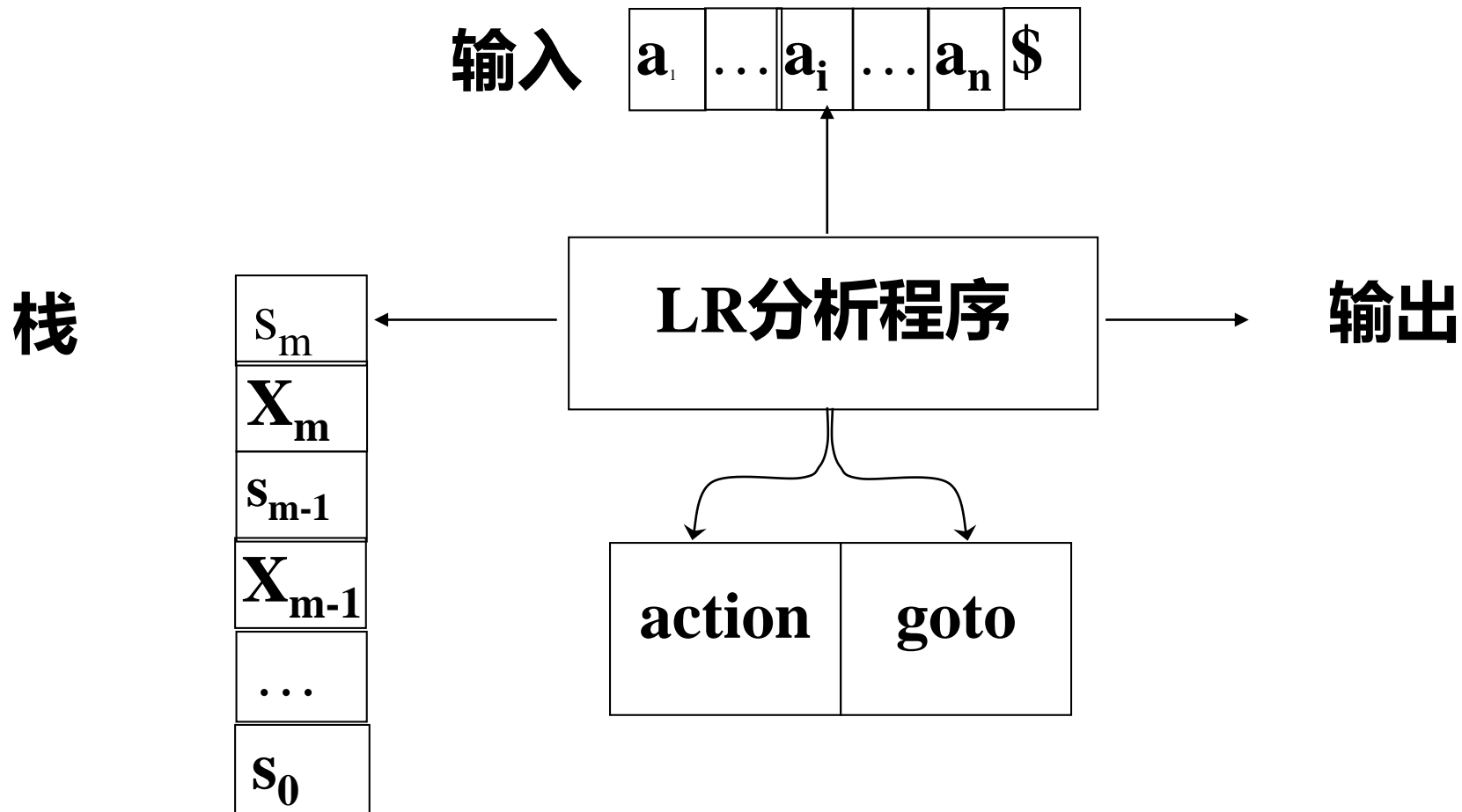


# 复习: LL(1)非递归分析



□行: 非终结符; 列: 终结符或\$; 单元: 产生式

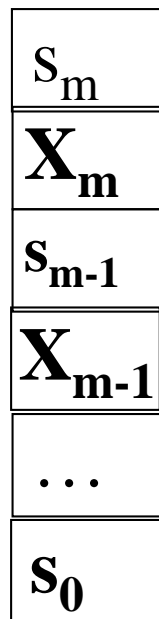
非终结符	输入符号					
	id	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F$	$F \rightarrow id$			$F \rightarrow (E)$		



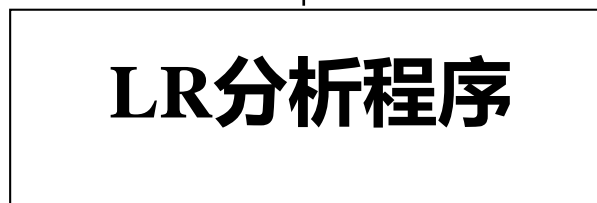
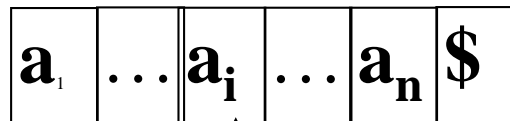
$s_j$ : 总结了栈中该状态以下的信息

$X_i$ : 代表文法符号

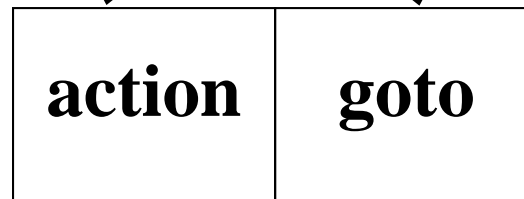
栈



输入



输出



$action[s_m, a_i]$ : 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)





# LR分析算法：举例



例 (1)  $E \rightarrow E + T$  (2)  $E \rightarrow T$  *si* 移进当前输入符号和状态 *i*  
 (3)  $T \rightarrow T * F$  (4)  $T \rightarrow E$  *rj* 按第 *j* 个产生式进行归约  
 (5)  $F \rightarrow ( E )$  (6)  $F \rightarrow id$  *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



# 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

1. 查  $action[5, *] \Rightarrow$  归约  
 2. 执行归约 ( $F \rightarrow \alpha$ ):

- 从栈中弹出  $|\alpha|$  个  $\langle$ 状态, 符号 $\rangle$  对
- 查  $goto[0, F] \Rightarrow 3$
- 将  $(F, 3)$  压入栈



# 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
0 <b><math>F</math> 3</b>	* 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
0 $F$ 3	* id + id \$	按 $T \rightarrow F$ 归约



# 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
0 $F$ 3	* id + id \$	按 $T \rightarrow F$ 归约
0 $T$ 2	* 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进



# 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进



# 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约



# 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
0 $F$ 3	* id + id \$	按 $T \rightarrow F$ 归约
0 $T$ 2	* id + id \$	移进
0 $T$ 2 * 7	id + id \$	移进
0 $T$ 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 $T$ 2 * 7 $F$ 10	+ 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约



# 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约
...	...	...





# LR分析算法：举例



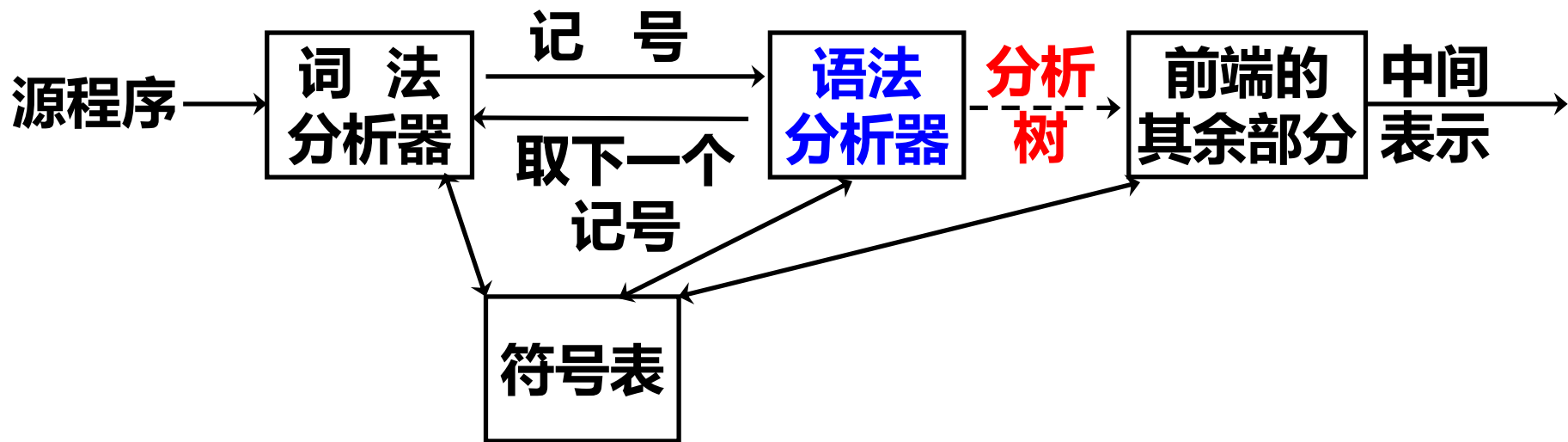
栈	输入	动作
0	id * id + id \$	移进
0 id 5	* id + id \$	按 $F \rightarrow id$ 归约
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约
...	...	...
0 E 1	\$	



# 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
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约
...	...	...
0 E 1	\$	接受



## □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）

## □关键在于构造LR分析表

- ❖ 计算所有可能的状态
- ❖ 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语法分析的每一步都形成一个格局config

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

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1X_2\dots X_ma_ia_{i+1}\dots a_n$
- ❖  $X_1X_2\dots X_m$  是最右句型的一个前缀



□ LR语法分析的每一步都形成一个格局config

$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \$)$

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n$
- ❖  $X_1 X_2 \dots X_m$  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



□ LR语法分析的每一步都形成一个格局config

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

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1X_2\dots X_ma_ia_{i+1}\dots a_n$
- ❖  $X_1X_2\dots X_m$  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



## □ LR语法分析的每一步都形成一个格局config

$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \$)$

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n$
- ❖  $X_1 X_2 \dots X_m$  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  $p$  不是句柄且存在  $p' = pa$
  - 何时归约？当  $p$  为句柄时





## □活前缀 (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



## 栈中可能出现的串:

$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

$a$

$ab$

$aA$

$aAb$

$aAbc$

$aAd$

$aAB$

$aABe$

$S$

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:

$S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

$a$

$a\underline{b}$

← 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  $B \rightarrow d$ )

$aAB$

$a\underline{ABe}$

← 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。



# 活前缀与句柄的关系



## 栈中可能出现的串：

$S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

$a$

$ab$

$a\underline{A}$

$a\underline{Ab}$

$aAbc$

$aAd$

$a\underline{AB}$

$aABe$

$S$

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>
<b>0</b>	<b>s5</b>			<b>s4</b>		<b>1</b>	<b>2</b>	<b>3</b>
<b>1</b>		<b>s6</b>						<b>acc</b>
<b>2</b>		<b>r2</b>	<b>s7</b>		<b>r2</b>	<b>r2</b>		
<b>3</b>		<b>r4</b>	<b>r4</b>		<b>r4</b>	<b>r4</b>		
<b>4</b>	<b>s5</b>			<b>s4</b>		<b>8</b>	<b>2</b>	<b>3</b>

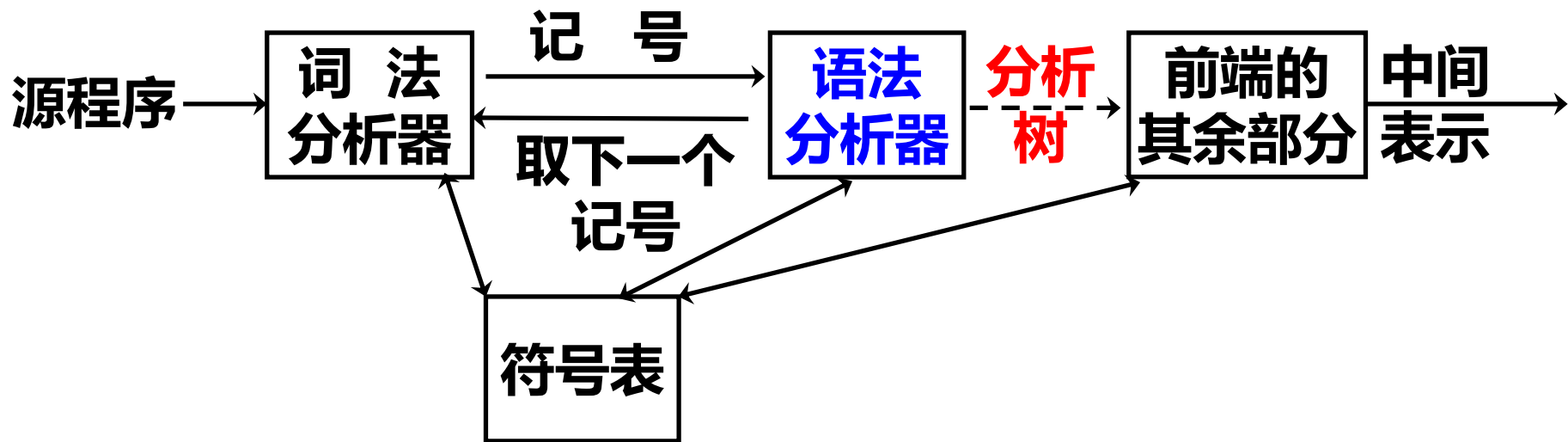


- 栈中的文法符号总是形成一个活前缀
- 分析表的转移函数本质上是识别活前缀的DFA
- 栈顶的状态符号包含确定句柄所需的一切信息

栈	输入	动作
0	id * id + id \$	移进
...	...	...
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约



- 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）



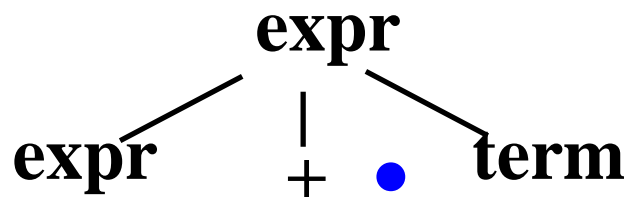


□SLR (Simple LR)

□LR(0)项目 (简称项目)

❖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



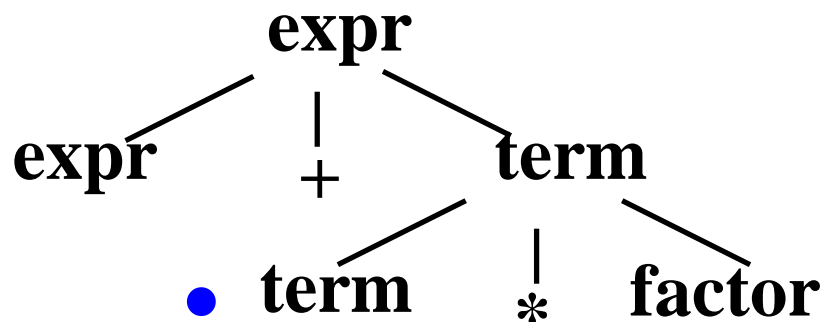


□SLR (Simple LR)

□LR(0)项目 (简称项目)

❖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



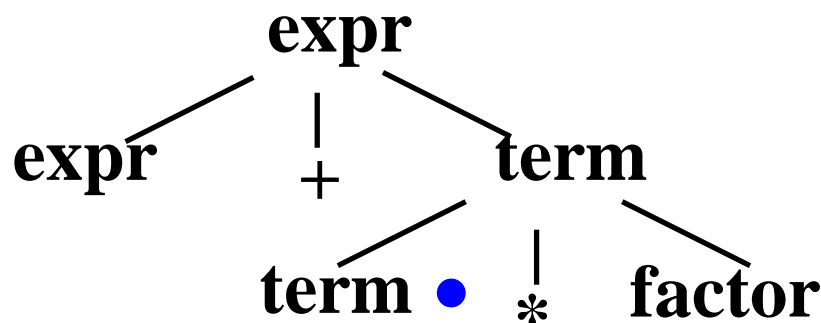


□SLR (Simple LR)

□LR(0)项目 (简称项目)

❖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





□ 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

$A \rightarrow \cdot$



# SLR分析表的构造



- 从文法构造识别活前缀的DFA
- 从上述DFA构造分析表



## 1. 拓广文法 (augmented grammar)

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

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

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



## 1. 拓广文法 (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

$I_0$ :

$E' \rightarrow \cdot E$

项集族是若干可能前缀的集合，对应DFA的状态





## 2. 构造LR(0)项目集规范族

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

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

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

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 $B \rightarrow \gamma$ 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

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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

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 $B \rightarrow \gamma$ 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

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

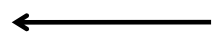
$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

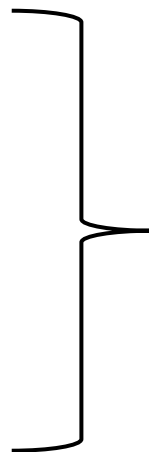
$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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



核心项目，初始项+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

$I_0:$

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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

$\xrightarrow{E}$

$I_1:$

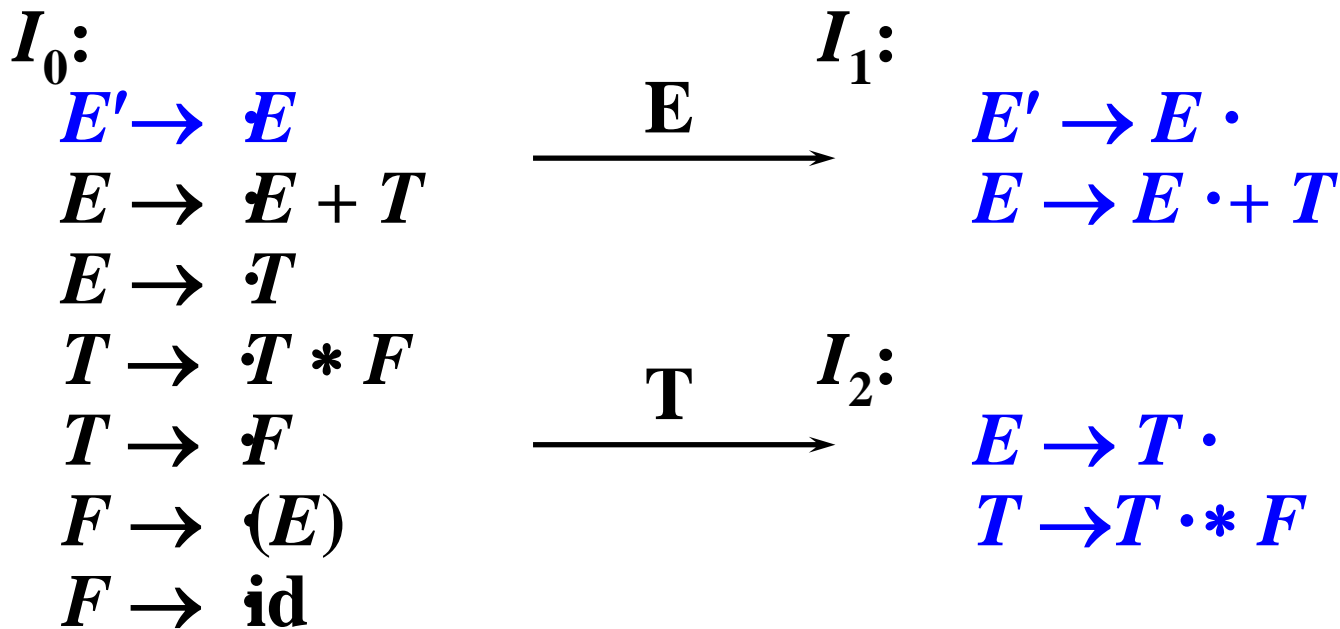
$E' \rightarrow E \cdot$

$E \rightarrow E \cdot + T$

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

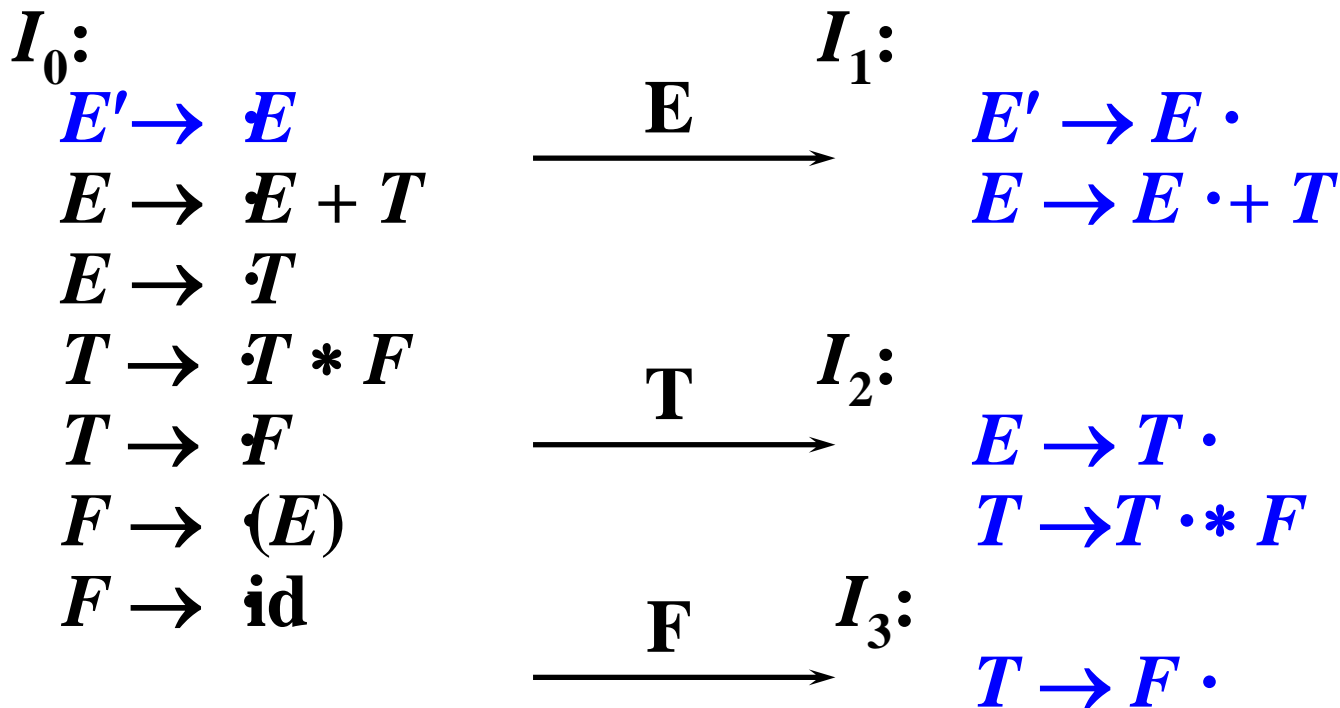


## 2. 构造LR(0)项目集规范族





## 2. 构造LR(0)项目集规范族





## 2. 构造LR(0)项目集规范族

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

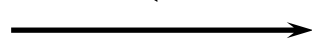
$F \rightarrow \cdot (E)$

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

$I_4$ :

$F \rightarrow (\cdot E)$

(



)





## 2. 构造LR(0)项目集规范族

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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

$I_4$ :

$F \rightarrow (\cdot E)$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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



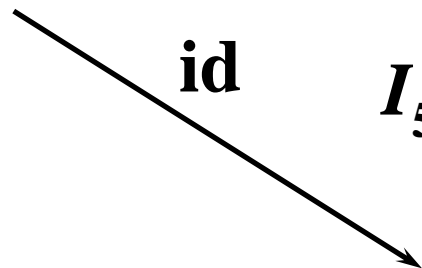
## 2. 构造LR(0)项目集规范族

$I_0$ :

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

$I_4$ :

$F \rightarrow (\cdot E)$   
 $E \rightarrow \cdot E + T$   
 $E \rightarrow \cdot T$   
 $T \rightarrow \cdot T * F$   
 $T \rightarrow \cdot F$   
 $F \rightarrow \cdot (E)$   
 $F \rightarrow \cdot \text{id}$

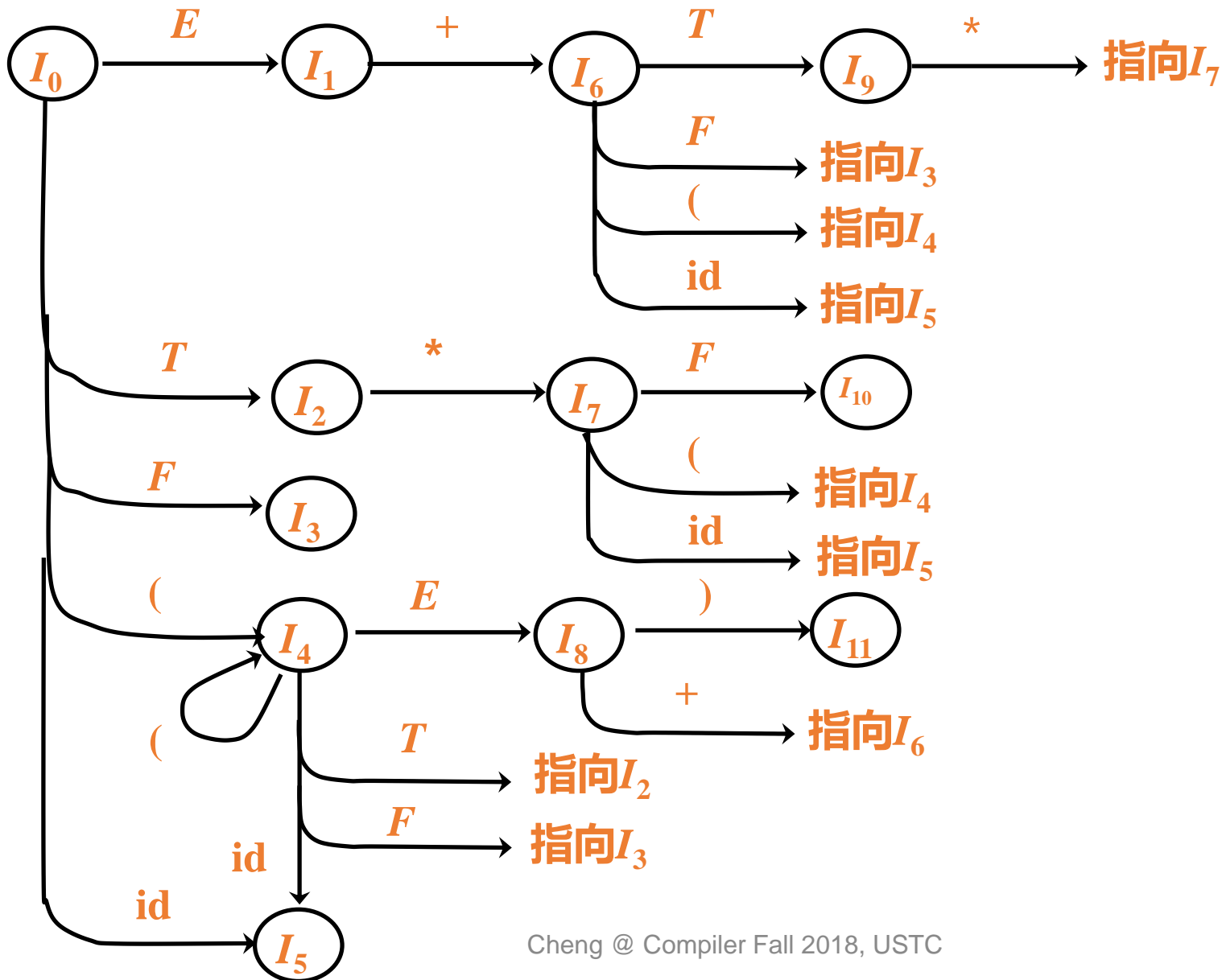


$I_5$ :

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

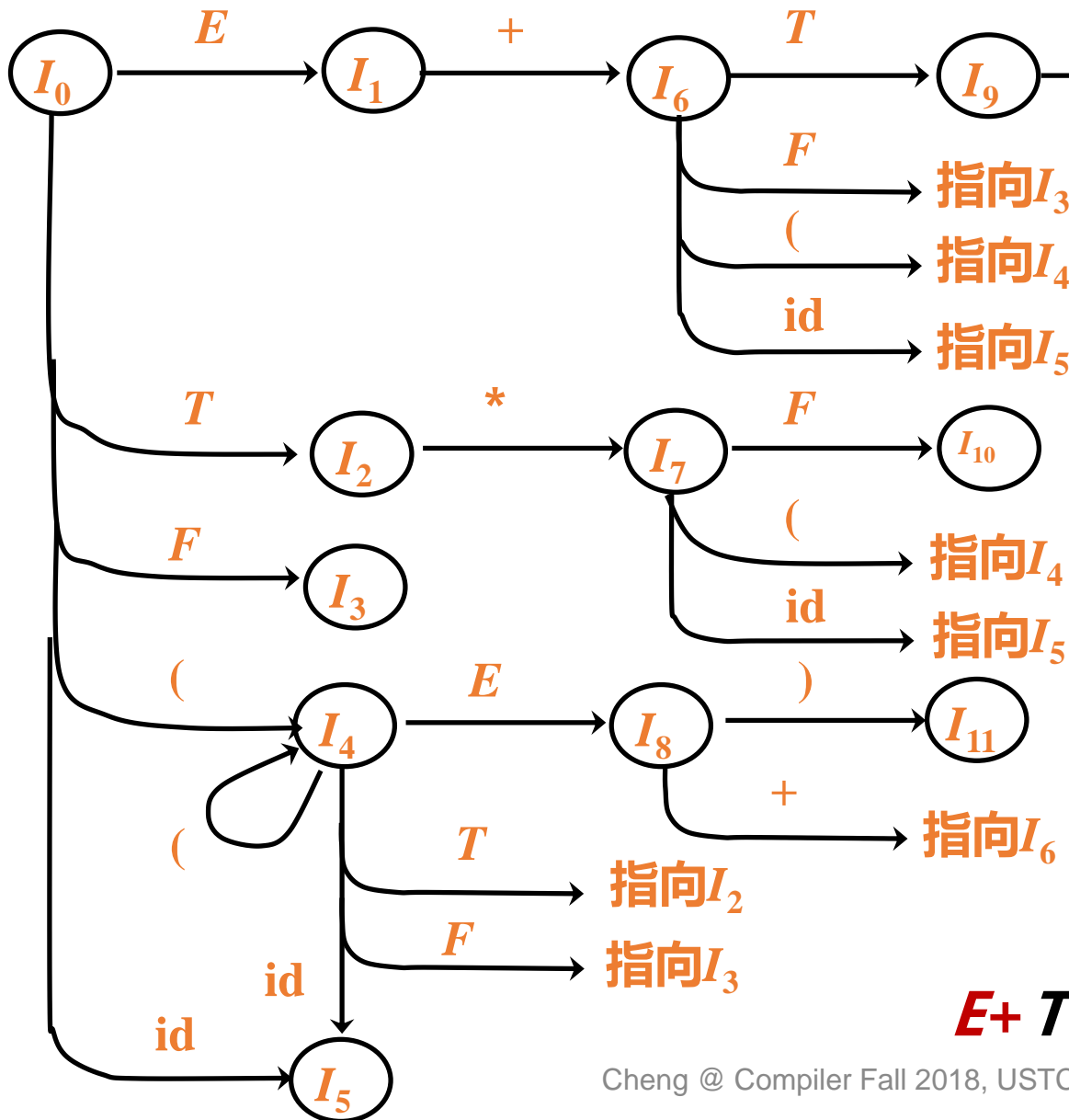


# 构造识别活前缀的DFA





# 构造识别活前缀的DFA



指向  $I_7$   
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**



# SLR分析表的构造



- 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 $[A \rightarrow \alpha a \beta]$ 在 $I_i$ 中，并且 $goto(I_i, a) = I_j$ ，那么置 $action[i, a]$ 为 $sj$
- ❖ 如果 $[A \rightarrow \alpha \cdot]$ 在 $I_i$ 中，那么对 $FOLLOW(A)$ 中的所有 $a$ ，置 $action[i, a]$ 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 $action[i, \$]$ 为接受 $acc$

□ 如果出现动作冲突，那么该文法就不是SLR(1)



□ 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

□ 不能由上面两步定义的条目都置为error

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





例 (1)  $E \rightarrow E + T$  (2)  $E \rightarrow T$   $si$  移进当前输入符号和状态  $i$   
 (3)  $T \rightarrow T * F$  (4)  $T \rightarrow E$   $rj$  按第  $j$  个产生式进行归约  
 (5)  $F \rightarrow ( E )$  (6)  $F \rightarrow id$   $acc$  接受

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



# 《编译原理与技术》

## 语法分析IV

**The Pessimist Sees Difficulty In Every Opportunity.  
The Optimist Sees Opportunity In Every Difficulty.**

**—— *Winston Churchill***