



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



# 《编译原理与技术》

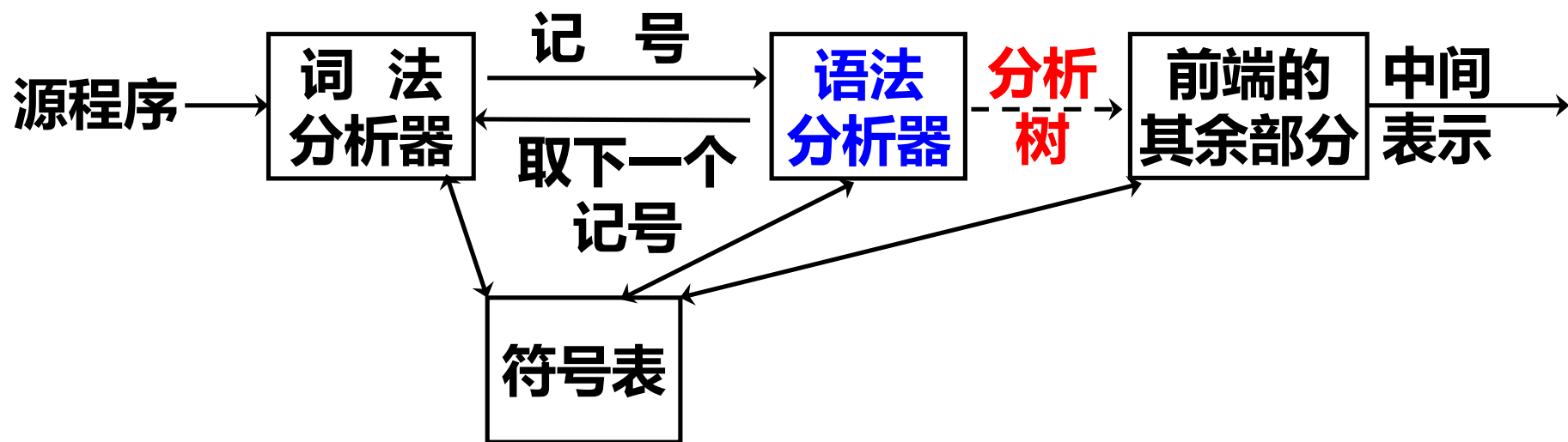
## 语法分析IV

计算机科学与技术学院

李 诚

25/09/2019

- Lab2在9.30课后发布
- 10.12 有一节课
- 今天晚上西区学活有一场SmartX公司的招聘会
- 国庆以后，不定期会有学校的老师来听课
- 国庆以后安排关于课后习题和LLVM IR的  
Tutorial



## □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)

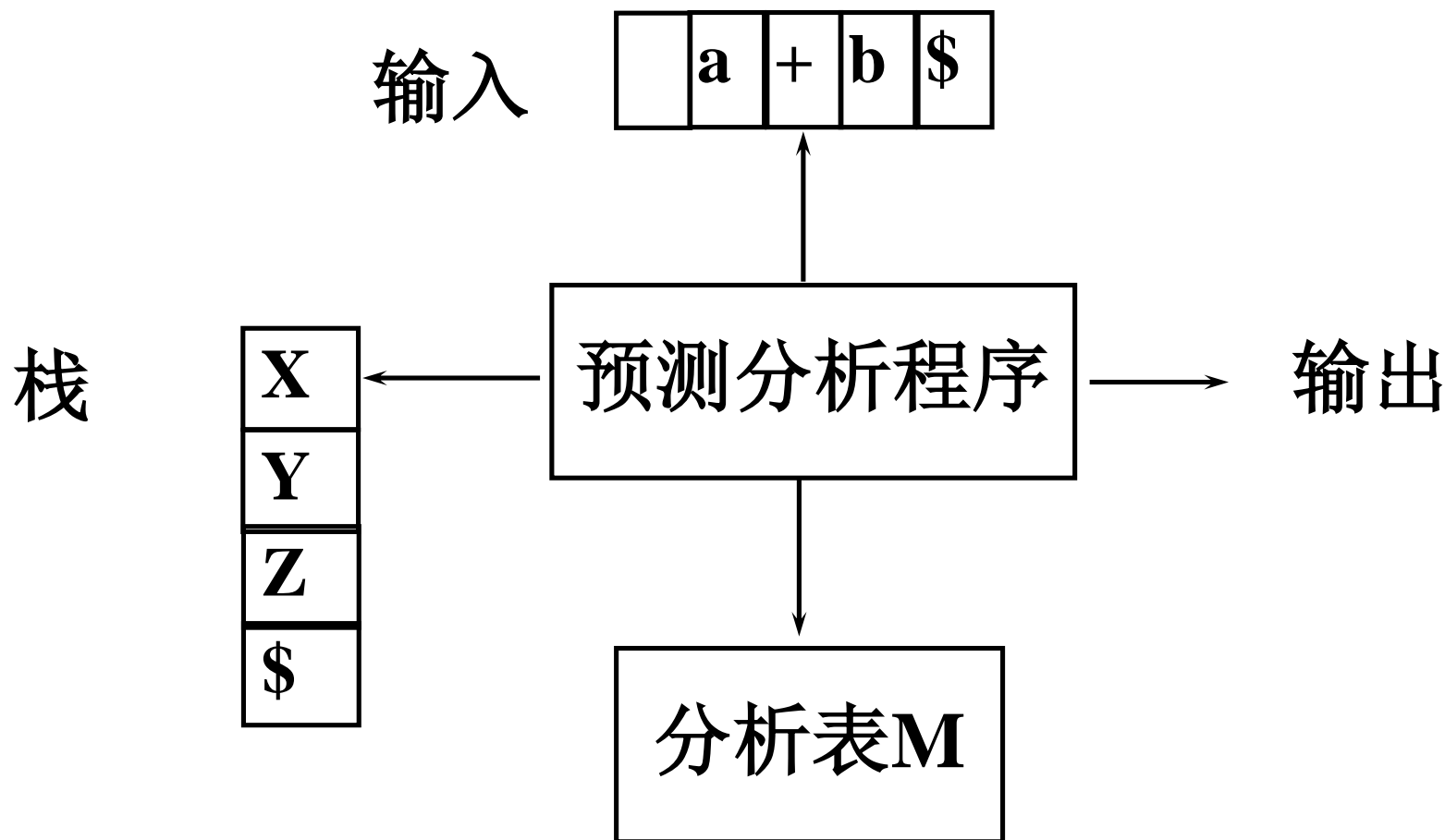
- ❖ 针对输入串，从文法的开始符号出发，尝试根据产生式规则 **推导 (derive)** 出该输入串。
- ❖ **LL(1)** 文法及非递归预测分析方法
- ❖ **left-to-right scan** + **leftmost derivation**

## □ 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





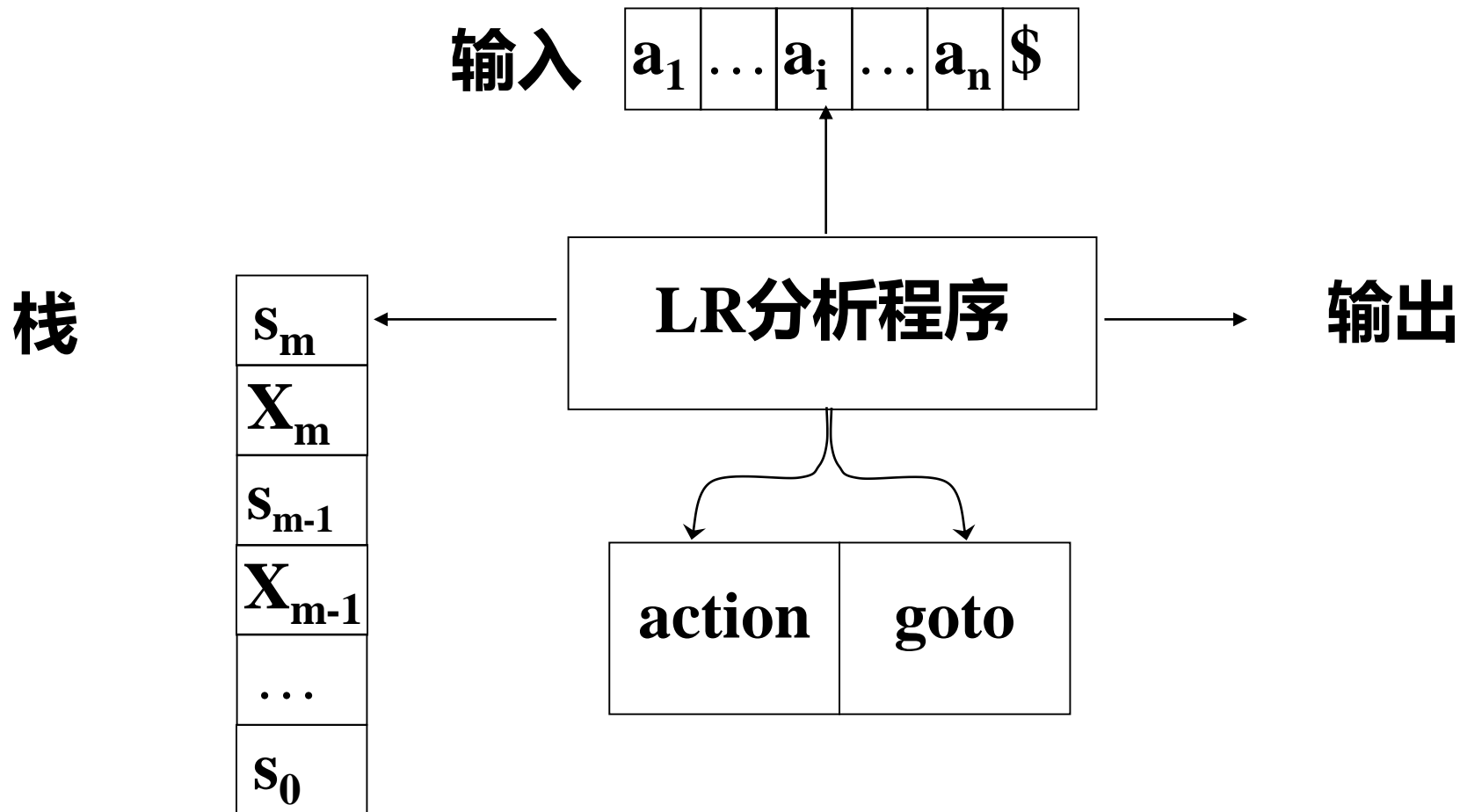
# 复习：LL(1)非递归分析



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

□行：非终结符；列：终结符或\$；单元：产生式

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



$s_j$ : 总结了栈中该状态以下的信息

$X_i$ : 代表文法符号

栈

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

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)





# LR分析算法：举例



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

*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



# LR分析算法：举例



栈	输 入	动 作
<b>0</b>	<b>id * id + id \$</b>	



# 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 $action[5, *]$  => 归约
2. 执行归约( $F \rightarrow \alpha$ ):
  - 从栈中弹出 $|\alpha|$ 个<状态, 符号>对
  - 查 $goto[0, F]$  => 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 <i>F</i> 3	* 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



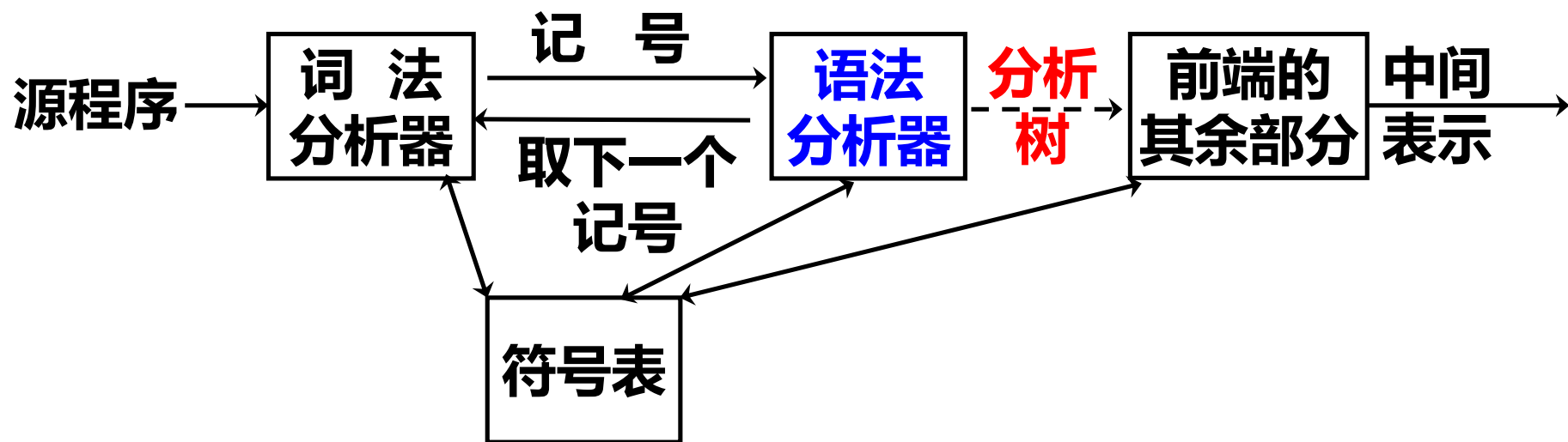
## □关键在于构造LR分析表

### ❖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



## □ LR语法分析的每一步都形成一个格局config

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

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$
- ❖  $X_1X_2\ldots X_m$  是最右句型的一个前缀
- ❖ 每一个前缀都对应一个状态，因此，找出所有可能出现在栈里出现的前缀，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

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

栈的内容

尚未处理的输入

- ❖ 代表最右句型  $X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$
- ❖  $X_1X_2\ldots X_m$  是最右句型的一个前缀
- ❖ 每一个前缀都对应一个状态，因此，找出所有可能出现在栈里出现的前缀，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

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

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  $p$  不是句柄且存在  $p' = pa$
  - 何时归约？当  $p$  为句柄时





## □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：

$a$   
 $ab$   
 $aA$   
 $aAb$   
 $aAbc$   
 $aAd$   
 $aAB$   
 $aABe$   
 $S$

$S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

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：

$S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

$a$

$a\underline{b}$

← 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：

$S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

$a$

$ab$

$a\bar{A}$

$a\bar{A}b$

$aAbc$

$aAd$

$a\bar{AB}$

$aABe$

$S$

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	<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

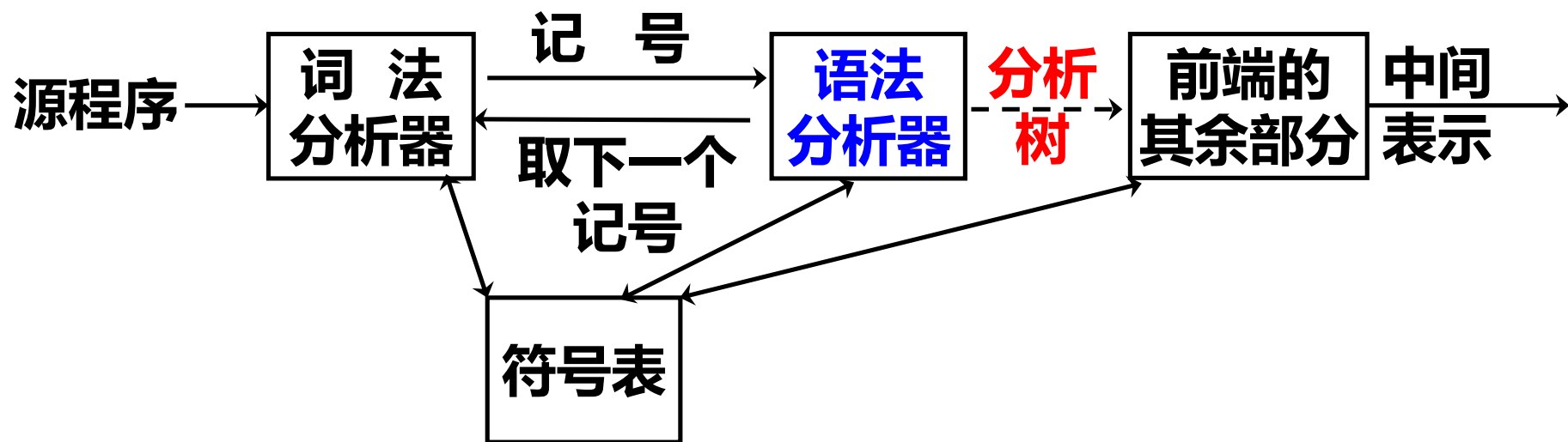


- 栈中的文法符号总是形成一个活前缀
- 分析表的转移函数本质上是识别活前缀的DFA
- 栈顶的状态符号包含确定句柄所需的一切信息

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）



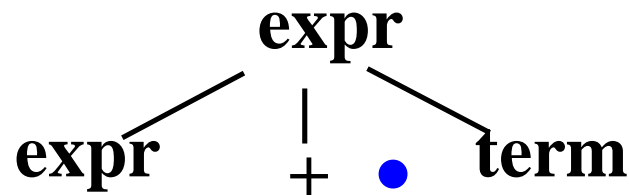


□SLR (Simple LR)

□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



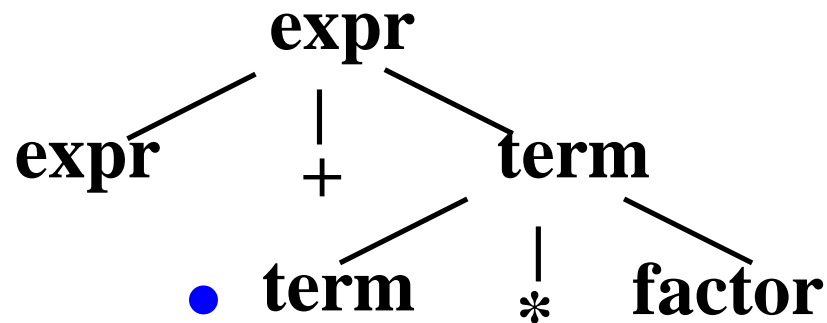


□SLR (Simple LR)

□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



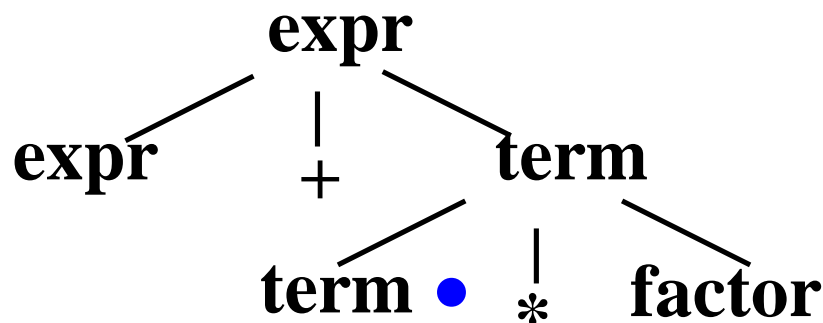


□SLR (Simple LR)

□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

$A \rightarrow \cdot$



□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)

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

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

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



## 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

$I_0$ :

$E' \rightarrow E$

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 $B \rightarrow \cdot \gamma$ 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

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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

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



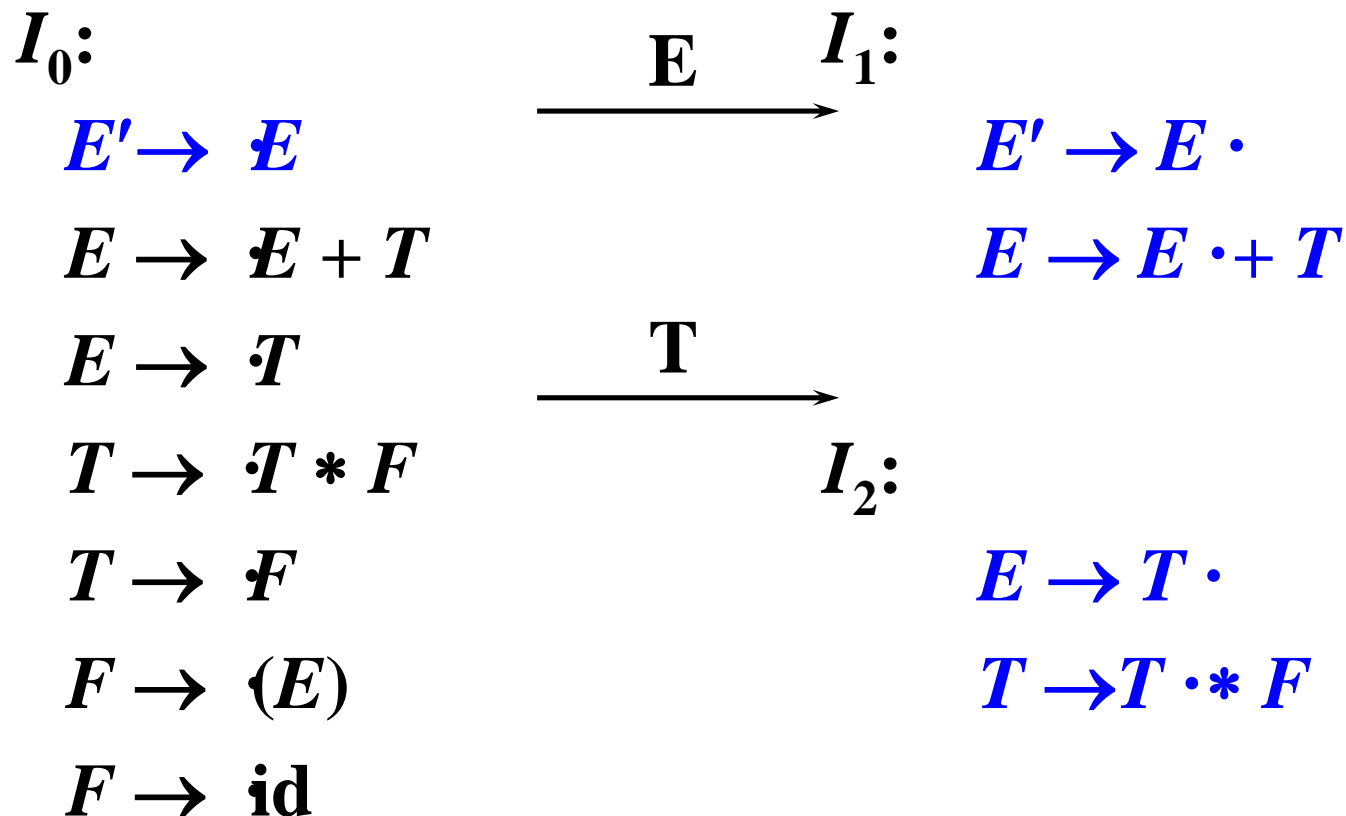
## 2. 构造LR(0)项目集规范族

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

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

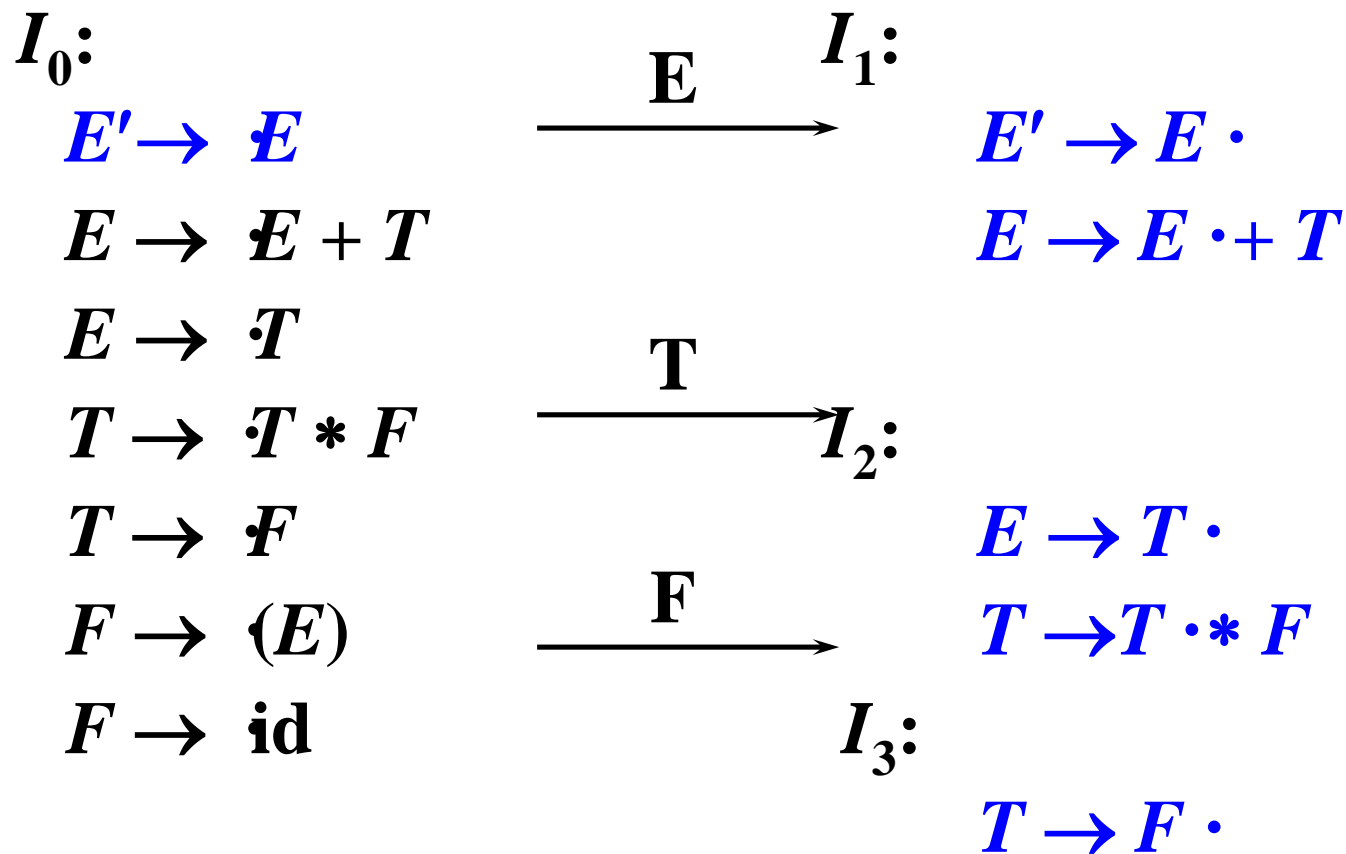


## 2. 构造LR(0)项目集规范族



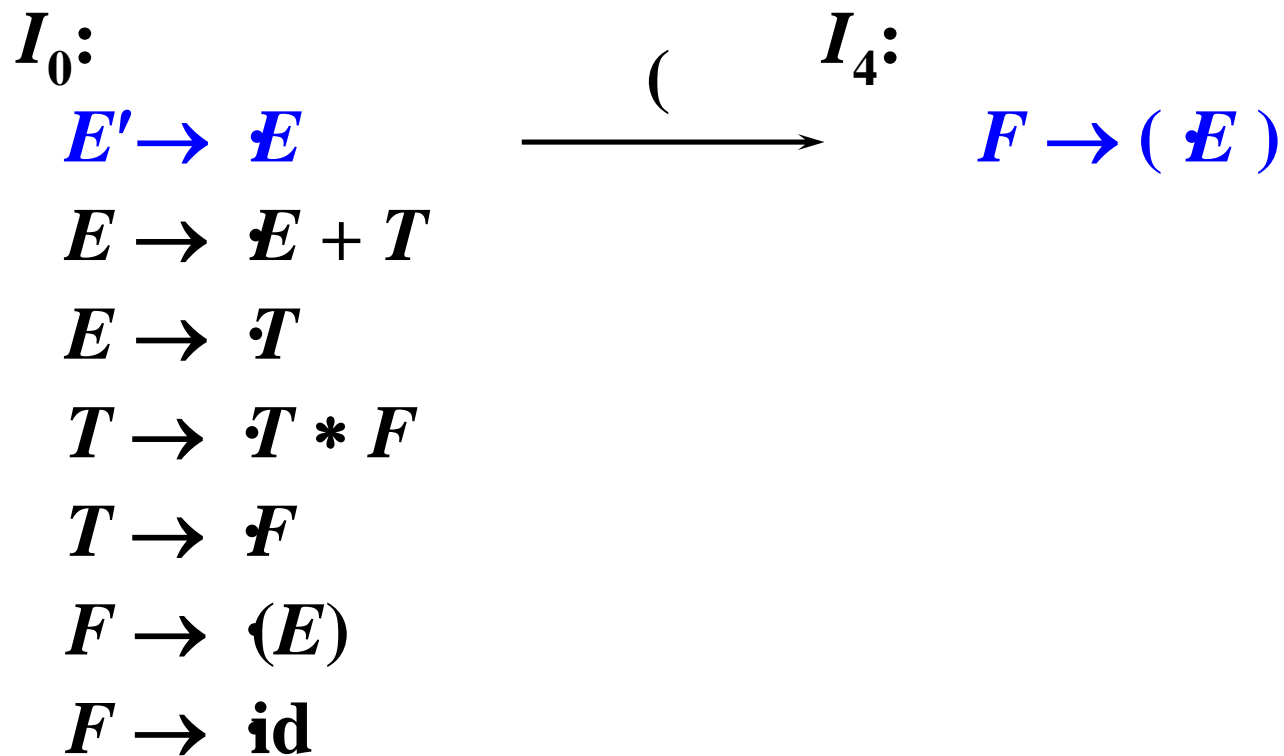


## 2. 构造LR(0)项目集规范族





## 2. 构造LR(0)项目集规范族





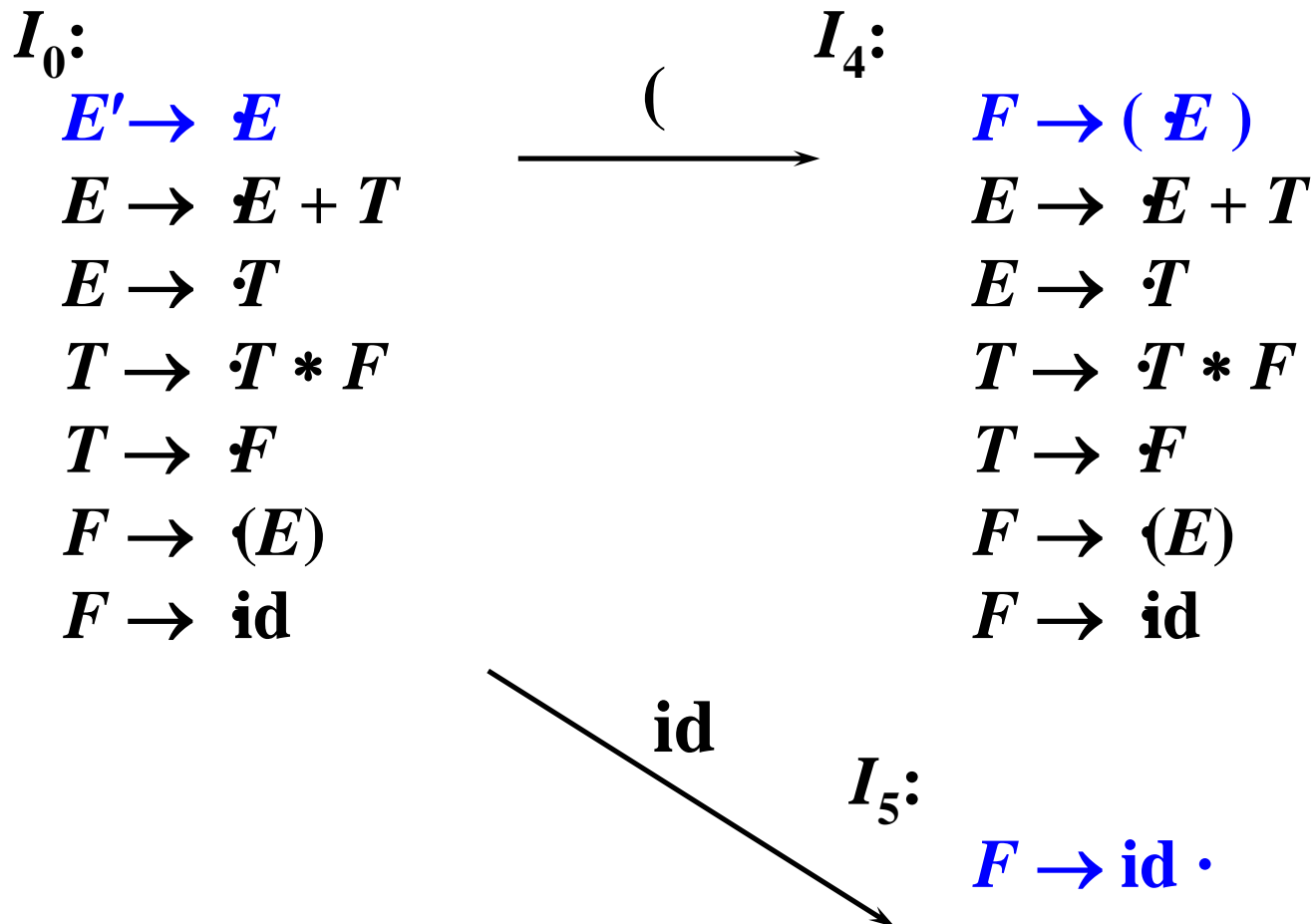


## 2. 构造LR(0)项目集规范族

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

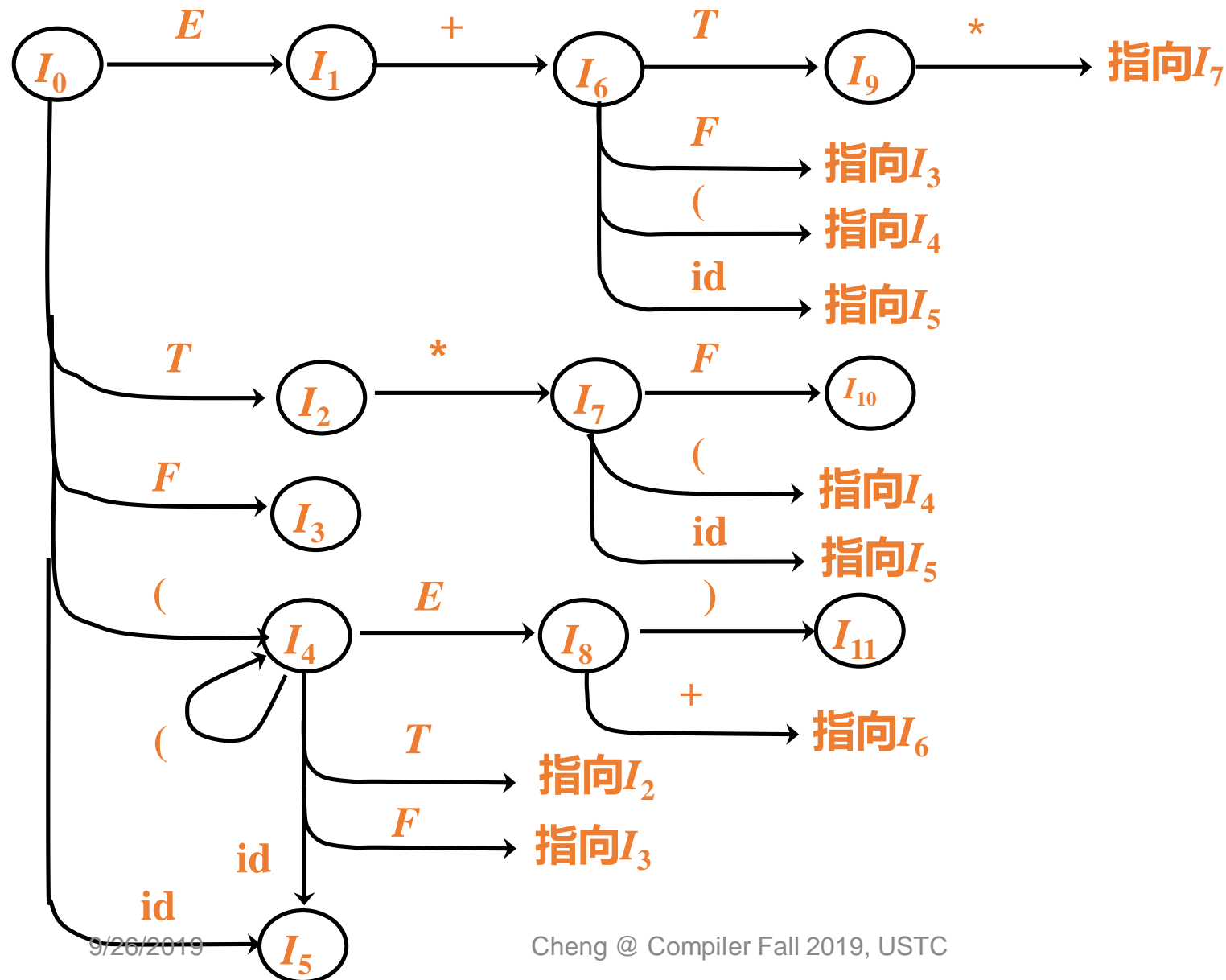


## 2. 构造LR(0)项目集规范族



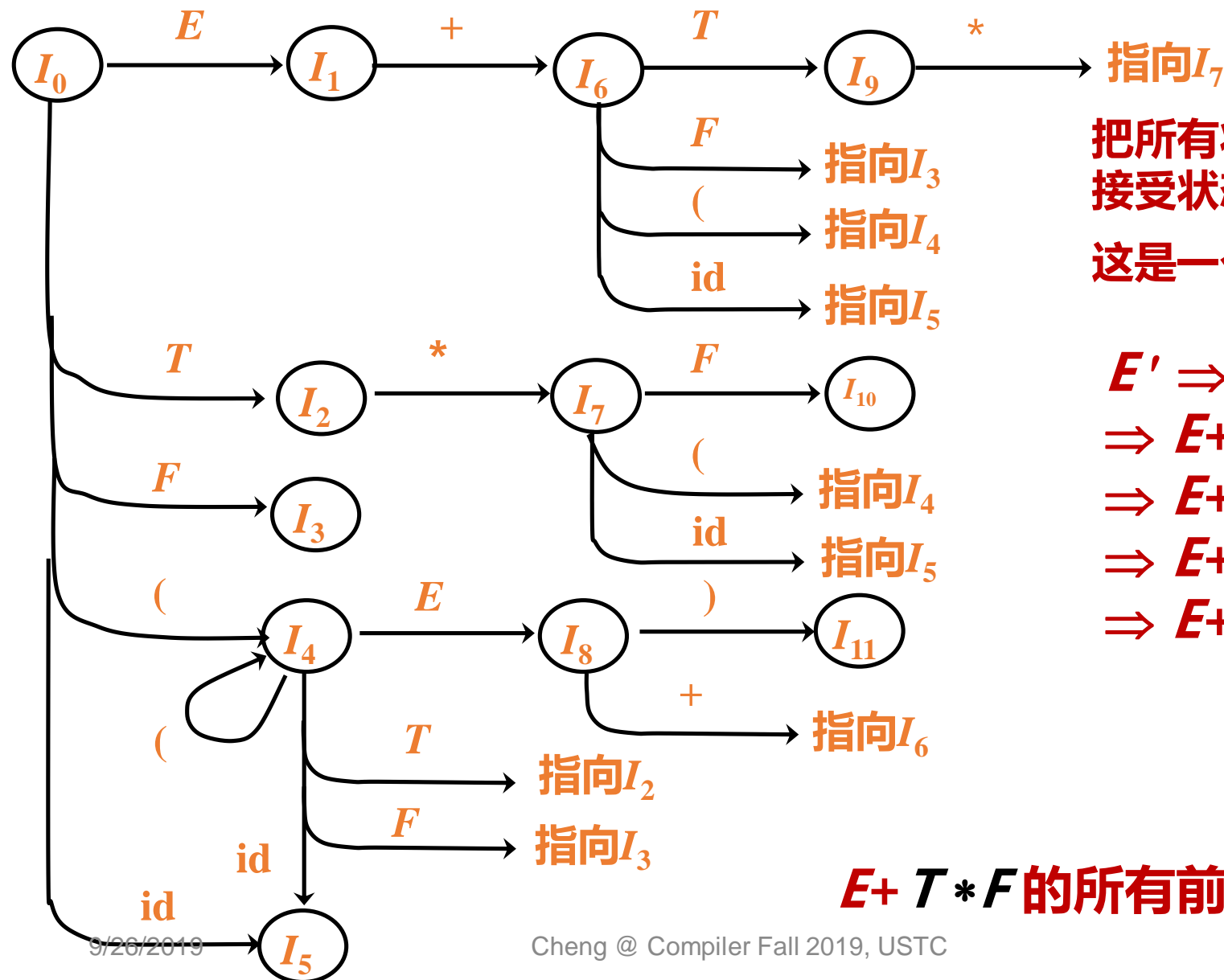


# 构造识别活前缀的DFA





# 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



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

$A \rightarrow \beta_1 \beta_2$  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

$$E' \rightarrow E$$

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

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

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

$E \rightarrow E + T$  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  $S' \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta_1 \beta_2 w$ , 那么就说项目

$A \rightarrow \beta_1 \beta_2$  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 $S' \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm} \alpha \beta_1 \beta_2 w$ ，那么就说项目

$A \rightarrow \beta_1 \beta_2$ 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 $I_7$

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

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

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





每一个项目一个状态

$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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



$I_0:$

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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

每一个项目一个状态

$E' \rightarrow \cdot E$



# 构造识别活前缀的NFA



$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow T \cdot$

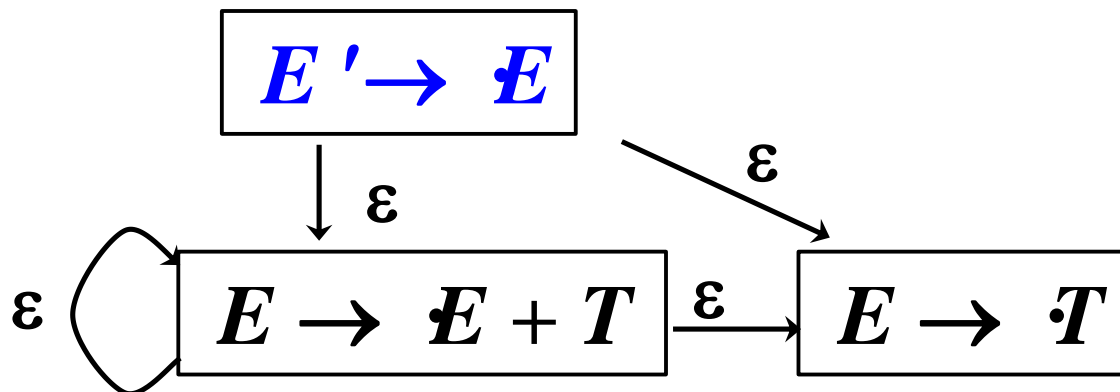
$T \rightarrow \cdot T * F$

$T \rightarrow F \cdot$

$F \rightarrow ( \cdot E )$

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

每一个项目一个状态





$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow T \cdot$

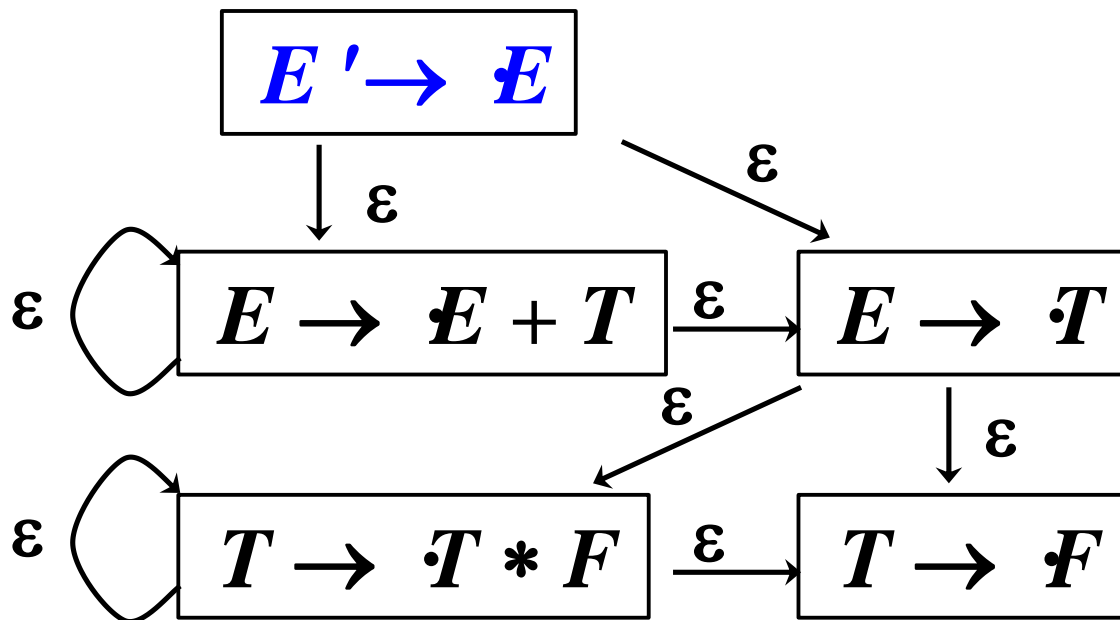
$T \rightarrow \cdot T * F$

$T \rightarrow F \cdot$

$F \rightarrow ( \cdot E )$

$F \rightarrow id \cdot$

每一个项目一个状态





$I_0$ :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

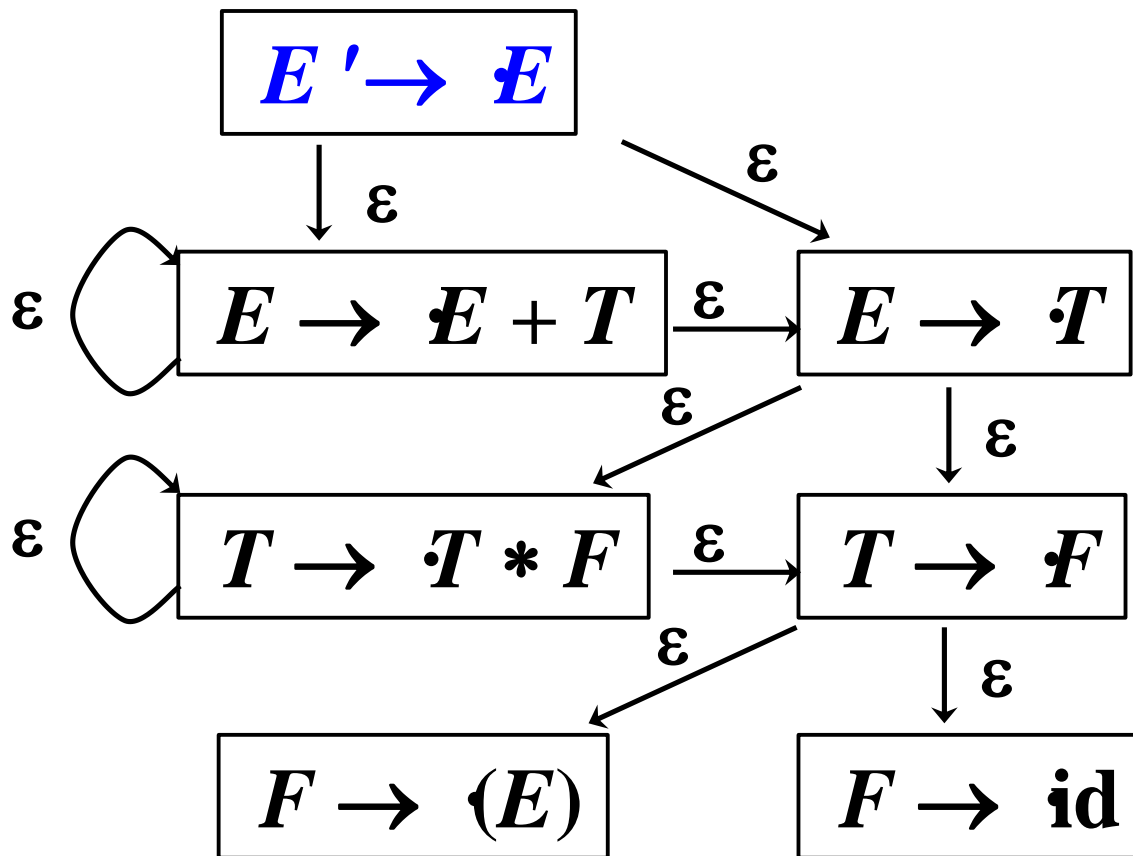
$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

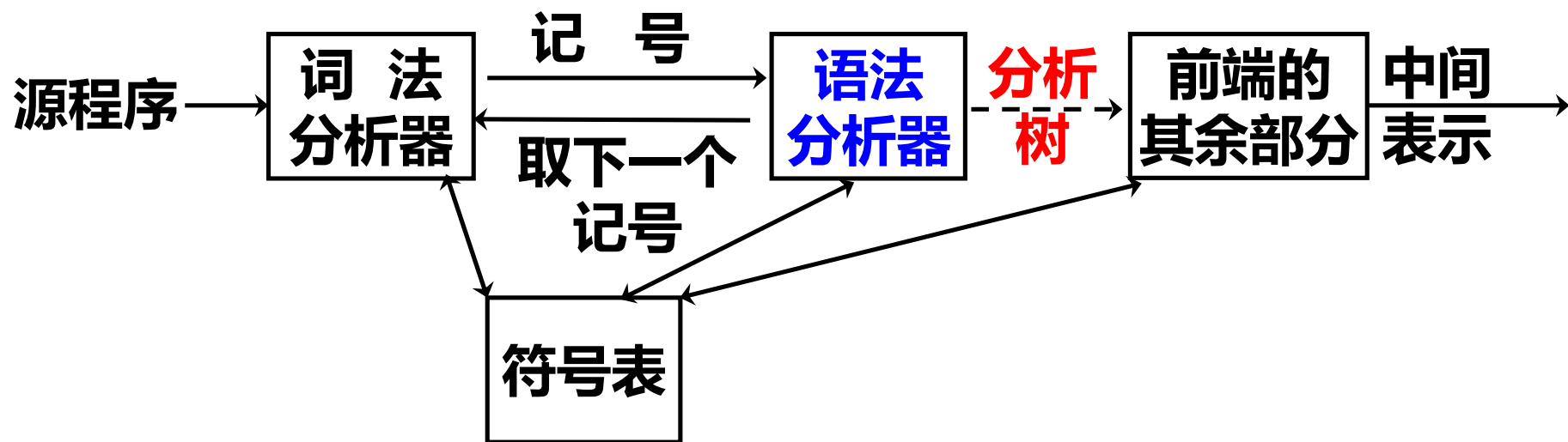
$F \rightarrow \cdot (E)$

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

每一个项目一个状态



由NFA通过子集构造法可以得到一个DFA。



## □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）



□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  $\text{goto}(I_i, a) = I_j$ , 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  $\text{goto}(I_i, A) = I_j$ ，那么  
 $\text{goto}[i, A] = j$



□ 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  $I_2$ :

$$E \rightarrow T \cdot$$

$$T \rightarrow T \cdot * F$$

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

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

❖  $\text{action}[2, *] = s7$



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



# 《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***