



第4章 语法分析 (3)

Syntax Analysis (3)

【对应教材 4.5-4.6】



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：自顶向下的语法分析

- 自顶向下分析是从文法的开始符号出发，试构造出一个最左推导，从左至右匹配输入的单词串。
- LL(1)文法：对于任意两个不同的产生式 $A \rightarrow \alpha | \beta$
 - $\text{First}(\alpha) \cap \text{First}(\beta) = \Phi$;
 - 如果 $\varepsilon \in \text{First}(\beta)$, 那么 $\text{First}(\alpha) \cap \text{Follow}(A) = \Phi$; 反之亦然。
- LL(1)文法的判别：计算First和Follow集合
- LL(1)文法的分析：首先构造LL(1)分析表
 - 编写不含回溯的递归分析子程序
 - 表格驱动的（非递归）预测分析方法



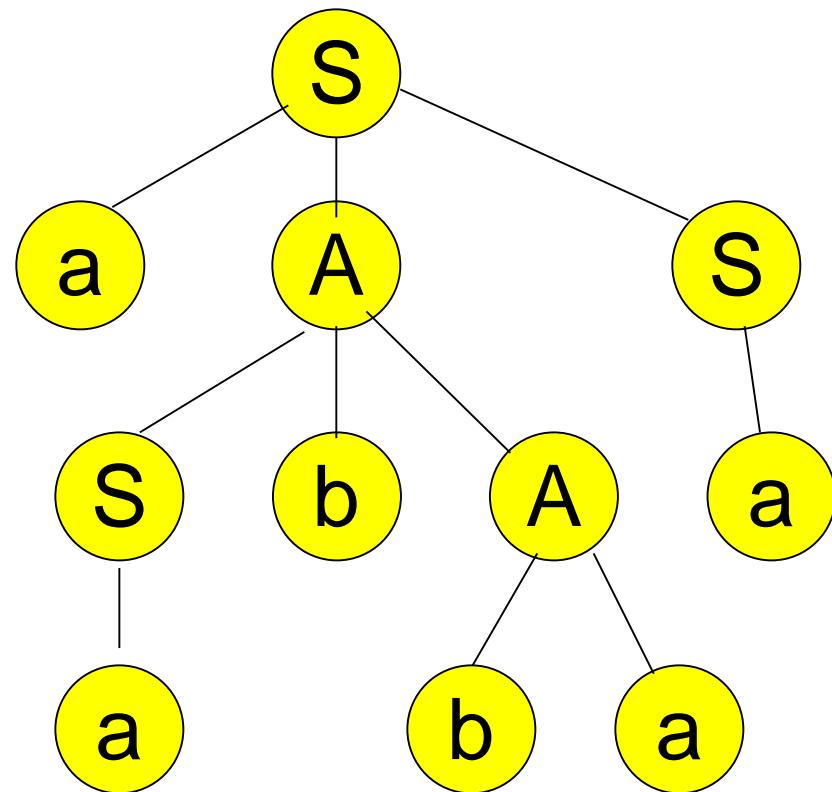
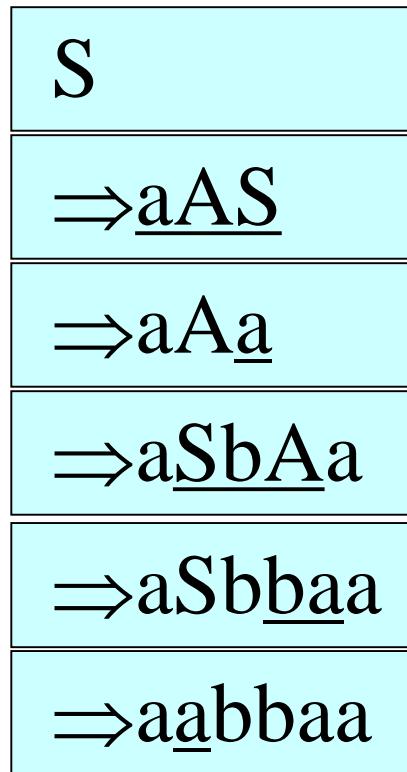
自底向上的语法分析

- 为一个输入串构造语法分析树的过程
- 从叶子（输入串中的终结符号，将位于分析树的底端）开始，向上到达根结点
 - 在实际的语法分析过程中并不会真的构造出相应的分析树，但是分析树概念可以方便理解
- 自底向上语法分析的通用框架
 - “移进-归约”分析
- LR：最大的可以构造出移进-归约语法分析器的语法类

自底向上分析的例子

文法: $S \rightarrow aAS \mid a \quad A \rightarrow SbA \mid SS \mid ba$

句子: aabbbaa





归约 (Reduce)

- 可以把自底向上语法分析过程看成从串 **w** “**归约**” 为文法开始符号 **S** 的过程
- 归约步骤：
 - 一个与某产生式右部相匹配的特定子串被替换为该产生式左部的非终结符号
- 问题：
 - 何时归约（归约哪些符号串）？
 - 归约到哪个非终结符号？



归约的例子

- **id * id** 的归约过程
 - **id * id <= F*id <= T*id <= T*F <= T <= E**
- 对于句型**T*id**, 有两个子串和某产生式右部匹配
 - T是E→T的右部
 - id是F→id的右部
 - 为什么选择将id归约为F, 而不是将T归约为E?
 - 原因: T归约为E之后, E*id不再是句型
 - 问题: 如何确定这一点?



句柄剪枝

- 对输入从左到右扫描，并进行自底向上语法分析，实际上可以反向构造出一个最右推导
- 句柄 (handle)：
 - 最右句型中和某个产生式右部匹配的子串，对它的归约代表了该最右句型的最右推导的最后一步
 - 定义：如果 $S \xrightarrow{^*_{\text{rm}}} aAw \xrightarrow{^*_{\text{rm}}} a\beta w$ ；那么紧跟 a 之后的 β 是 (对应 $A \rightarrow \beta$ 的) 一个句柄
- 在一个最右句型中，句柄右边只有终结符号
- 如果文法是无二义性的，那么每个句型都有且仅有一个句柄。



句柄的例子

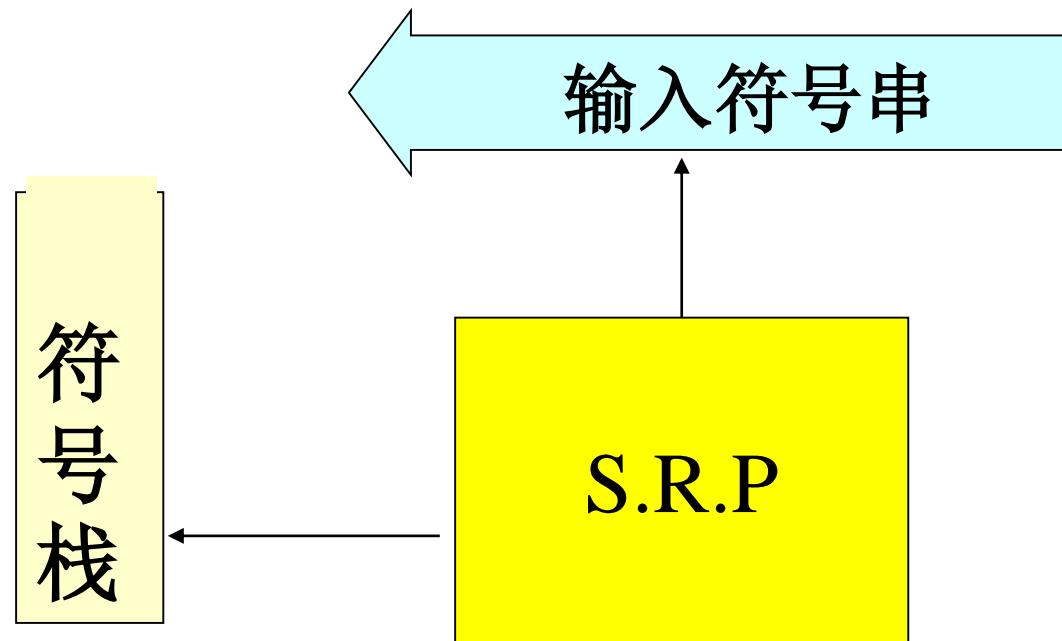
□ 输入： $\text{id} * \text{id}$

最右句型	句柄	归约用的产生式
$\text{id}_1 * \text{id}_2$	id_1	$F \rightarrow \text{id}$
$F * \text{id}_2$	F	$T \rightarrow F$
$T * \text{id}_2$	id_2	$F \rightarrow \text{id}$
$T * F$	$T * F$	$T \rightarrow T * F$



移进-归约分析 (Shift-Reduce Parsing)

- 建立符号栈，用来记录分析的历史和现状，并根据所面临的状态，确定下一步动作是移进还是归约。





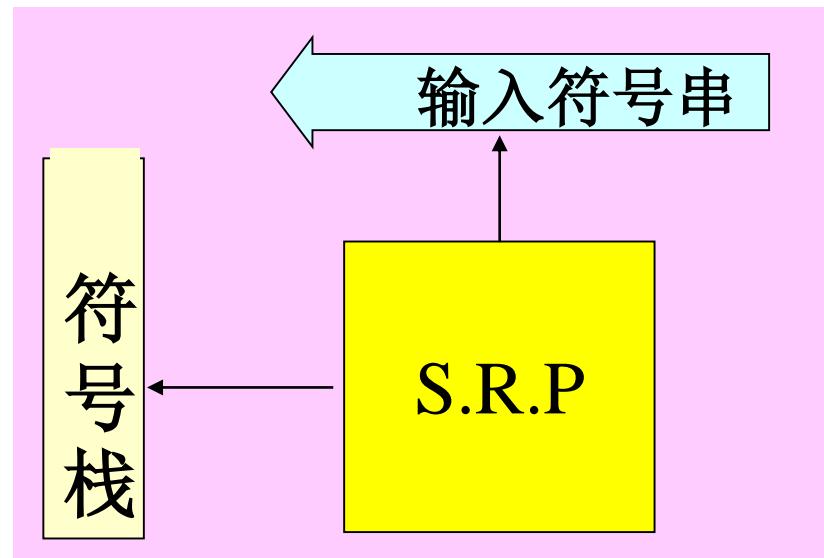
主要分析动作

- 移进：将下一个输入符号移动到栈顶
- 归约：将句柄归约为相应的非终结符号
 - 句柄总是在栈顶
 - 具体操作时弹出句柄，压入被归约到的非终结符号
- 接受：宣布分析过程成功完成
- 报错：发现语法错误，调用错误恢复子程序



分析过程

- 开始时刻：栈中只包含\$，而输入为w\$
- 成功结束时刻：栈中\$S，而输入\$
- 在分析过程中，不断地移进符号，并在识别到句柄时进行归约
- 句柄被识别时总是出现在栈的顶部





移进-归约分析过程的例子

栈	输入	动作
\$	$\text{id}_1 * \text{id}_2 \$$	移入
\$ id_1	$* \text{id}_2 \$$	按照 $F \rightarrow \text{id}$ 归约
\$ F	$* \text{id}_2 \$$	按照 $T \rightarrow F$ 归约
\$ T	$* \text{id}_2 \$$	移入
\$ $T *$	$\text{id}_2 \$$	移入
\$ $T * \text{id}_2$	\$	按照 $F \rightarrow \text{id}$ 归约
\$ $T * F$	\$	按照 $T \rightarrow T + T$ 归约
\$ T	\$	按照 $E \rightarrow T$ 归约
\$ E	\$	accept

栈内符号串 +
未处理输入符号串
= 当前句型

句柄都在栈顶

图 4-28 一个移入 - 归约语法分析器
在处理输入 $\text{id}_1 * \text{id}_2$ 时经历的格局

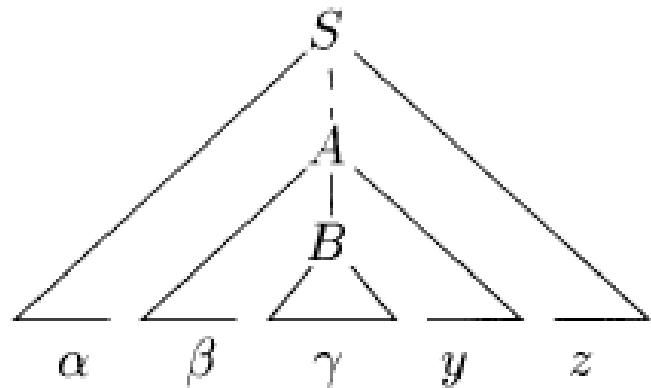
为什么句柄总是在栈顶？

□ 分别考虑最右推导的两个连续步骤的两种情况

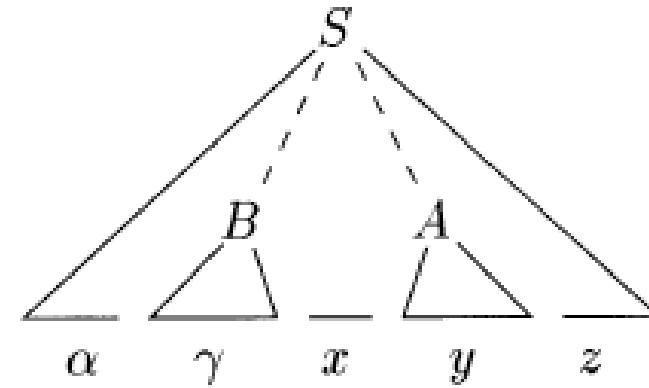
$$1) \ S \xrightarrow[\text{rm}]{*} \alpha A z \xrightarrow[\text{rm}]{*} \alpha \beta B y z \xrightarrow[\text{rm}]{*} \alpha \beta \gamma y z$$

$$2) \ S \xrightarrow[\text{rm}]{*} \alpha B x A z \xrightarrow[\text{rm}]{*} \alpha B x y z \xrightarrow[\text{rm}]{*} \alpha y x y z$$

- $x, y, z \in V_T^*$
- $\gamma \in (V_T \cup V_N^*)$



情况 (1)



情况 (2)



移进-归约分析中的冲突

- 有时候，即使知道了栈中所有内容、以及下面k个输入符号，仍然无法知道是否该进行归约，或者不知道按照什么产生式进行归约
- 分别对应两种冲突情形：
 - “移进-归约”冲突
 - “归约-归约”冲突
- “移进-归约”冲突的例子：
 - 设栈中符号串是 $a\beta$ ，接下来的k个符号是x，产生移进/归约冲突的原因是存在y和 y' 使得 $a\beta xy$ 是最右句型且 β 是句柄，而 $a\beta xy'$ 也是最右句型，但是句柄还在右边。



归约/归约冲突的例子

□ 输入为 **id (id , id)**

□ 冲突时的格局：

- 栈：...**id (id**

- 输入：**, id) ...**

(1)	<i>stmt</i>	\rightarrow	<i>id (parameter-list)</i>
(2)	<i>stmt</i>	\rightarrow	<i>expr := expr</i>
(3)	<i>parameter-list</i>	\rightarrow	<i>parameter-list , parameter</i>
(4)	<i>parameter-list</i>	\rightarrow	<i>parameter</i>
(5)	<i>parameter</i>	\rightarrow	<i>id</i>
(6)	<i>expr</i>	\rightarrow	<i>id (expr-list)</i>
(7)	<i>expr</i>	\rightarrow	<i>id</i>
(8)	<i>expr-list</i>	\rightarrow	<i>expr-list , expr</i>
(9)	<i>expr-list</i>	\rightarrow	<i>expr</i>

某个语言中，
多维数组的
表示方法。

图 4-30 有关过程调用和数组引用的产生式



LR 分析简介

- 也称为LR(k)分析，是一种常用的自底向上分析。
 - L 指的是从左向右扫描输入符号串
 - R 指的是构造最右推导的逆过程（即规范归约）
 - k 指的是决定动作时向前看的符号个数，通常取0或1
- LR分析的过程是进行规范归约，即每次归约的都是句柄。
- LR 分析的主要特点：
 - 1) 适合文法类足够大：比LL(k)文法的范围大
 - 2) 分析效率高（无回溯）
 - 3) 报错及时
 - 4) 可以自动生成



LR(k) 项 (LR(k) Item)

定义: $A \rightarrow a \cdot \beta, \gamma$ 称为文法 G 的一个 LR(k) 项。

- 其中, $A \rightarrow a\beta$ 是 G 的一个产生式;
- γ 是一个长度为 k 的由终结符和 \$ 构成的符号串, 称为搜索字符串。
- 其中的圆点 • 表示语法分析程序已读到 (可能经过若干次归约) 的符号串 a , 而构成圆点后面的符号串 β (可能经过归约) 尚未读到。
- 搜索串 γ 用于辅助确定是否在栈顶形成了句柄。
- k 取 0 时, LR(0) 项 简写为 $A \rightarrow a \cdot \beta$ 。
- 产生式 $A \rightarrow a$ 相应的项有 $|a| + 1$ 个。



LR(0)项

- 产生式 $A \rightarrow XYZ$, 有下面四个项:

$$A \rightarrow \bullet XYZ \quad A \rightarrow X \bullet YZ$$

$$A \rightarrow XY \bullet Z \quad A \rightarrow XYZ \bullet$$

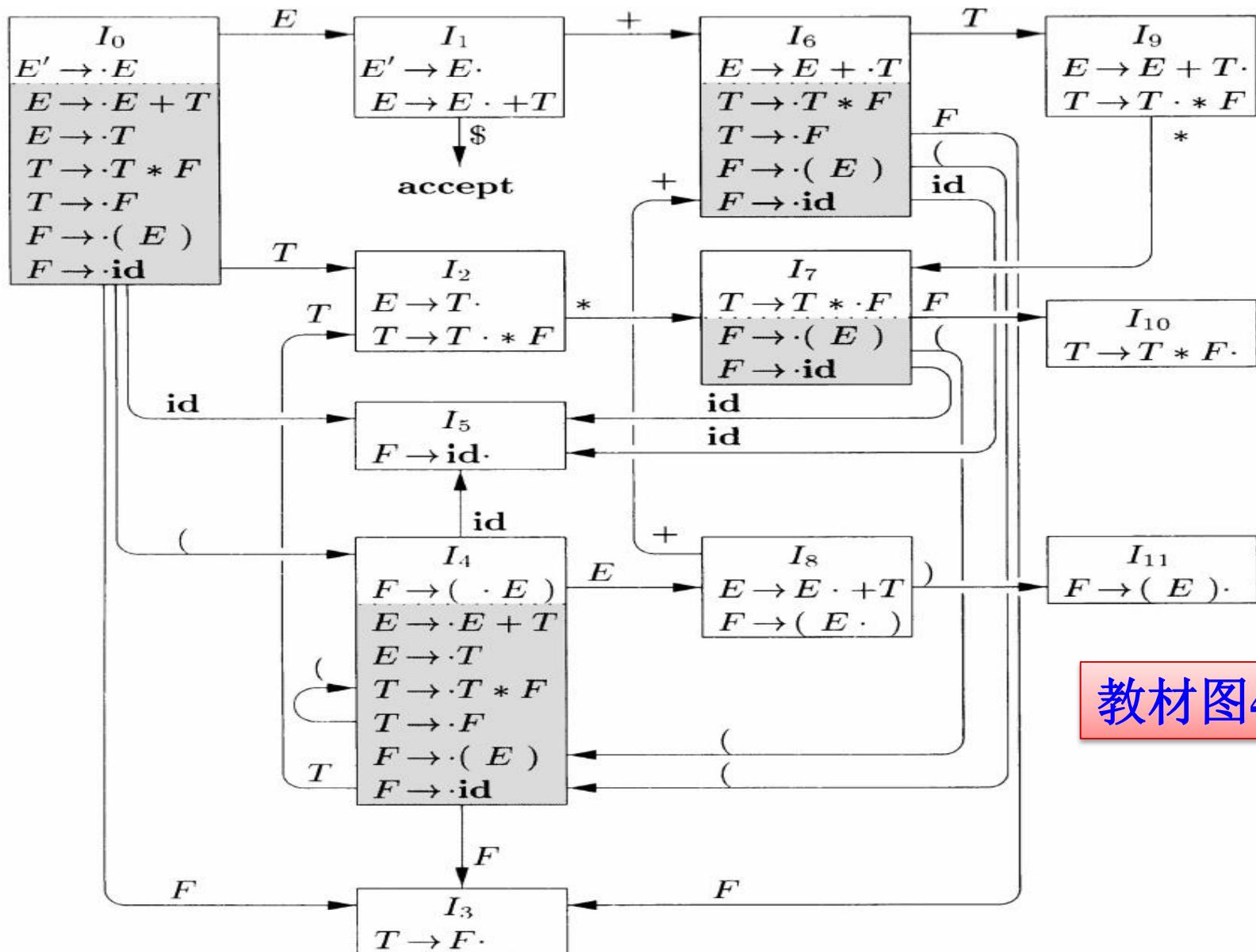
- LR(0)项可分为四类:

- $A \rightarrow \alpha \bullet a\beta, \quad a \in V_T$ 移进项
- $A \rightarrow \alpha \bullet B\beta, \quad B \in V_N$ 待归约项
- $A \rightarrow \alpha \bullet$ 归约项
- $S' \rightarrow S \bullet$ 接受项
- 对应于产生式 $A \rightarrow \varepsilon$ 的唯一项是 $A \rightarrow \bullet$, 它是归约项。

- 项也可以用一对整数表示。

- (i, j) 表示第 i 条规则, 点位于右部第 j 个位置

构造LR(0)项集族与LR(0)自动机



教材图4-31



规范LR(0)项集族的构造

- 拓广文法：
 - 增加一条产生式 $S' \rightarrow S$ 拓广文法 G
 - 对文法中的产生式进行编号
- 规范LR(0)项集族即LR(0)自动机的状态集合；
- 构造过程中用到的子函数
 - CLOSURE(I)： I 的项集闭包
 - 对应于DFA化算法的 ϵ -CLOSURE
 - GOTO(I, X)： I 的 X 后继
 - 对应于DFA化算法的转换边



计算 closure(I)

1. 把I中每一个项都加到初始为空的closure(I)中
2. 若项 $A \rightarrow \alpha \cdot B\beta \in \text{closure}(I)$ 且 $B \rightarrow \gamma \in P$
则把 $B \rightarrow \cdot \gamma$ 加进 closure(I)中。
3. 反复执行(2)直到closure不再增大为止。

上面第二步的分析含义：

- 项 $A \rightarrow \alpha \cdot B\beta$ 表示期望在接下来的输入中归约到 B 。
- 显然，要归约到 B ，首先要扫描归约到 B 的某个产生式的右部；
- 因此对每个产生式 $B \rightarrow \gamma$ ，加入 $B \rightarrow \cdot \gamma$ ；
 - 表示它期望能够扫描归约到 γ 。



构造闭包算法的伪代码描述

```
SetOfItems CLOSURE( $I$ ) {
     $J = I;$ 
    repeat
        for ( $J$  中的每个项  $A \rightarrow \alpha \cdot B\beta$ )
            for ( $G$  的每个产生式  $B \rightarrow \gamma$ )
                if (项  $B \rightarrow \cdot\gamma$  不在  $J$  中)
                    将  $B \rightarrow \cdot\gamma$  加入  $J$  中;
    until 在某一轮中没有新的项被加入到  $J$  中;
    return  $J$ ;
}
```



闭包构造的例子

例： (0) $E' \rightarrow E$

(1) $E \rightarrow E + T$

(2) $E \rightarrow T$

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

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow id$

项集 $\{[E' \rightarrow \bullet E]\}$ 的闭包

$E' \rightarrow \bullet E$
 $E \rightarrow \bullet E + T$
 $E \rightarrow \bullet T$
 $T \rightarrow \bullet T^* F$
 $T \rightarrow \bullet F$
 $F \rightarrow \bullet (E)$
 $F \rightarrow \bullet id$



LR(0)项集中的内核项和非内核项

- 在算法中，在加入某个项 $B \rightarrow \cdot\gamma$ 时，所有以B为左部的产生式所对应的（点在最左边的）项都会被加入
- **内核项：**初始项 $S' \rightarrow \cdot S$ 、以及所有点不在最左边的项
- **非内核项：**除了 $S' \rightarrow \cdot S$ 之外、点在最左边的项；
- 实现算法时可以考虑只保存相应的非终结符号；甚至可以只保存内核项，而在要使用非内核项时调用**CLOSURE**函数重新计算。



GOTO函数

- **GOTO(I,X)**: 项集 $\{[A \rightarrow \alpha X \cdot \beta] \mid [A \rightarrow \alpha \cdot X \beta] \in I\}$ 的闭包
 - 根据项的的含义, $GOTO(I,X)$ 表示读取输入中的X 或者归约到一个X之后的情况。
 - $GOTO(I,X)$ 定义了LR(0)自动机中状态I在X之上的转换。
- 例如:
 - $I = \{[E' \rightarrow E \cdot], [E \rightarrow E \cdot + T]\};$
 - $GOTO(I,+)$ 计算如下:
 - I 中只有一个项的点后面跟着+, 对应的项为 $[E \rightarrow E \cdot + \cdot T]$
 - $CLOSURE(\{[E \rightarrow E \cdot + T]\}) = \{[E \rightarrow E \cdot + T], [T \rightarrow \cdot T^* F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id]\}.$



求LR(0)项集规范族的算法

```
void items( $G'$ ) {  
     $C = \text{CLOSURE}(\{[S' \rightarrow \cdot S]\})$ ;  
    repeat  
        for ( $C$  中的每个项集  $I$ )  
            for (每个文法符号  $X$ )  
                if (  $\text{GOTO}(I, X)$  非空且不在  $C$  中)  
                    将  $\text{GOTO}(I, X)$  加入  $C$  中;  
    until 在某一轮中没有新的项集被加入到  $C$  中;  
}
```

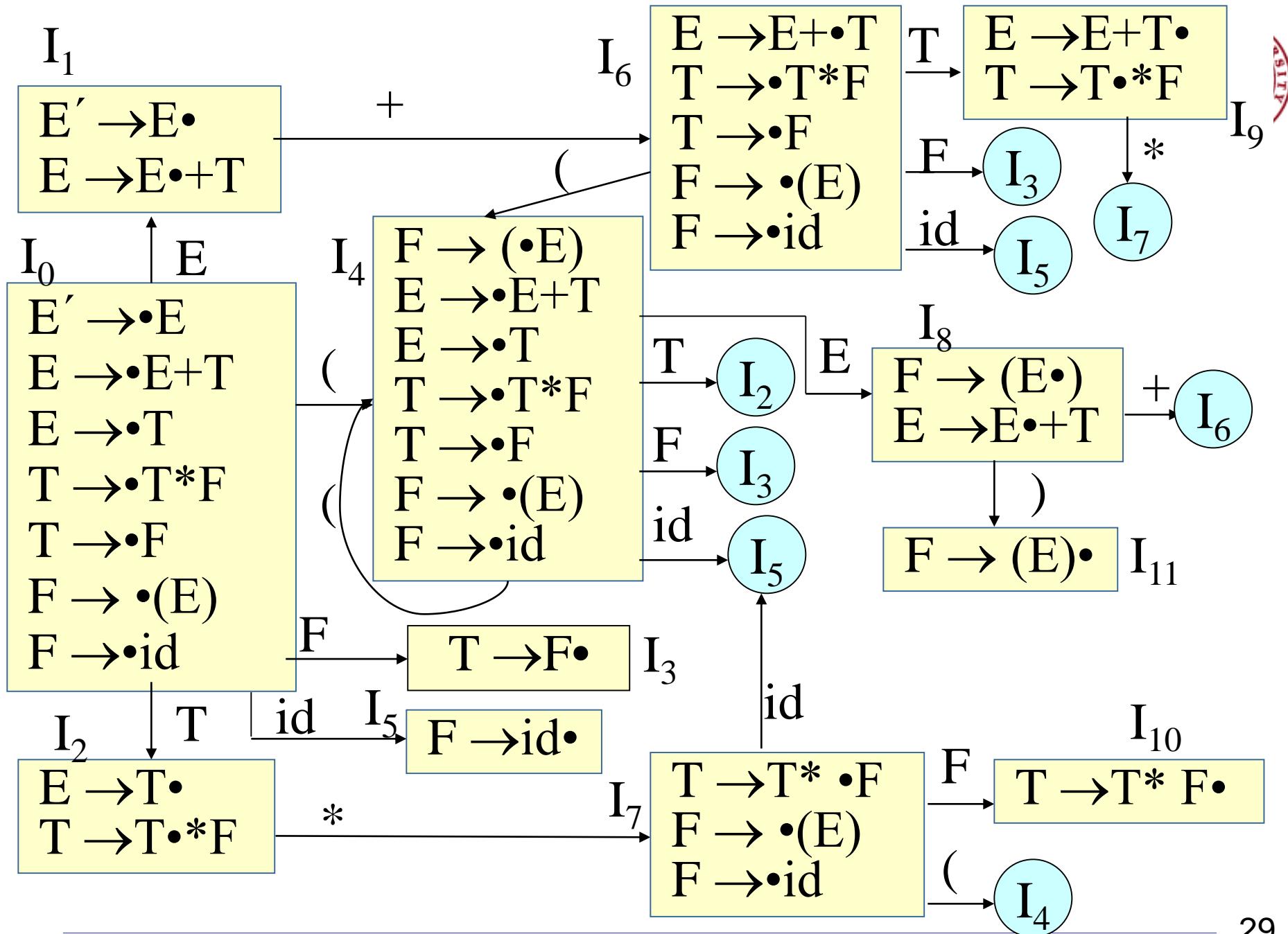
- 从初始项集开始，不断计算各种可能的后继，直到生成所有的项集。
- 可以和NFA的子集构造算法类比



LR(0)自动机的构造

□ 构造方法：

- 规范LR(0)项集族中的项集可以作为LR(0)自动机的状态
- **GOTO(I,X)=J**, 则从I到J有一个标号为X的转换
- 初始状态为**CLOSURE({S' →•S})**对应的项集
- 接受状态：包含形如**A → α•**的项集对应的状态





LR(0)自动机的作用 (1)

- 假设文法符号串 γ 使LR(0)自动机从开始状态运行到状态 (项集) j
- 如果 j 中有一个形如 $A \rightarrow a \cdot$ 的项, 那么
 - 在 γ 之后添加一些终结符号可以得到一个最右句型
 - a 是 γ 的后缀, 且 $A \rightarrow a$ 是这个句型的句柄
 - 表示可能找到了当前最右句型的句柄
- 如果 j 中存在一个项 $B \rightarrow a \cdot X \beta$, 那么
 - 在 γ 之后添加 $X \beta$, 然后再添加一个终结符号串可得到一个最右句型
 - 在这个句型中 $B \rightarrow a X \beta$ 是句柄
 - 此时表示还没有找到句柄, 需要移进



LR(0)自动机的作用 (2)

□ LR(0)自动机的使用

- 移进-归约时，LR(0)自动机被用于识别句型
- 已经归约/移进得到的文法符号序列对应于LR(0)自动机的一条路径
- 如果路径到达接受状态，表明栈上端的某个符号串可能是句柄

□ 不需要每次用归约/移进得到的串来运行LR(0)自动机

- 路径被放到栈中；且文法符号可以省略，因为由LR(0)状态可以确定相应文法符号
- 在移进后，根据原来的栈顶状态即可知道新的状态
- 在归约时，根据归约产生式的右部长度弹出相应状态，仍然可以根据此时的栈顶状态计算得到新状态



LR(0)的作用演示：分析id*id

行号	栈	符号	输入	动作
(1)	0	\$	id * id \$	移入到 5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow id$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \rightarrow F$ 归约
(4)	0 2	\$ T	* id \$	移入到 7
(5)	0 2 7	\$ T *	id \$	移入到 5
(6)	0 2 7 5	\$ T * id	\$	按照 $F \rightarrow id$ 归约
(7)	0 2 7 10	\$ T * F	\$	按照 $T \rightarrow T * F$ 归约
(8)	0 2	\$ T	\$	按照 $E \rightarrow T$ 归约
(9)	0 1	\$ E	\$	接受

- 根据LR(0)自动机状态和符号的对应关系，可以得到符号栏中的字符串。

使用图4-31的LR(0)自动机



LR分析的结构

□ LR分析表（依赖于具体文法）

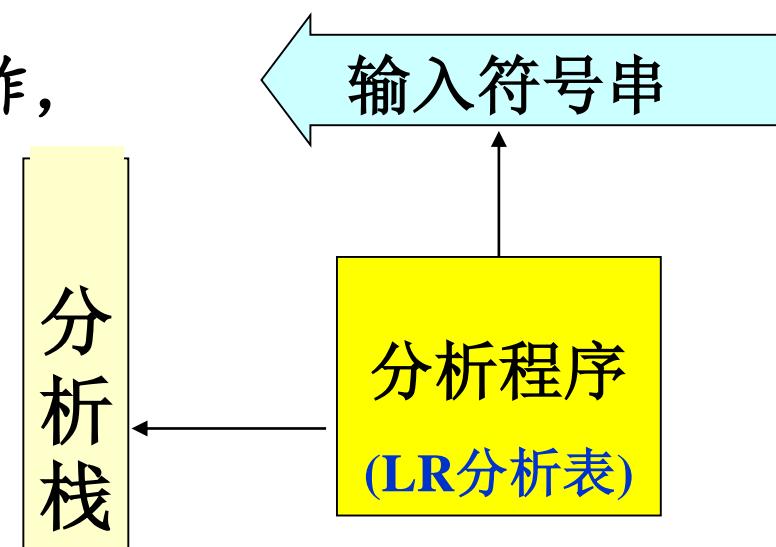
- 由两个矩阵组成，其功能是指示分析器的动作是移进还是归约，根据不同的文法类要采用不同的构造方法

□ 驱动程序

- 执行分析表所规定的动作，对栈进行操作

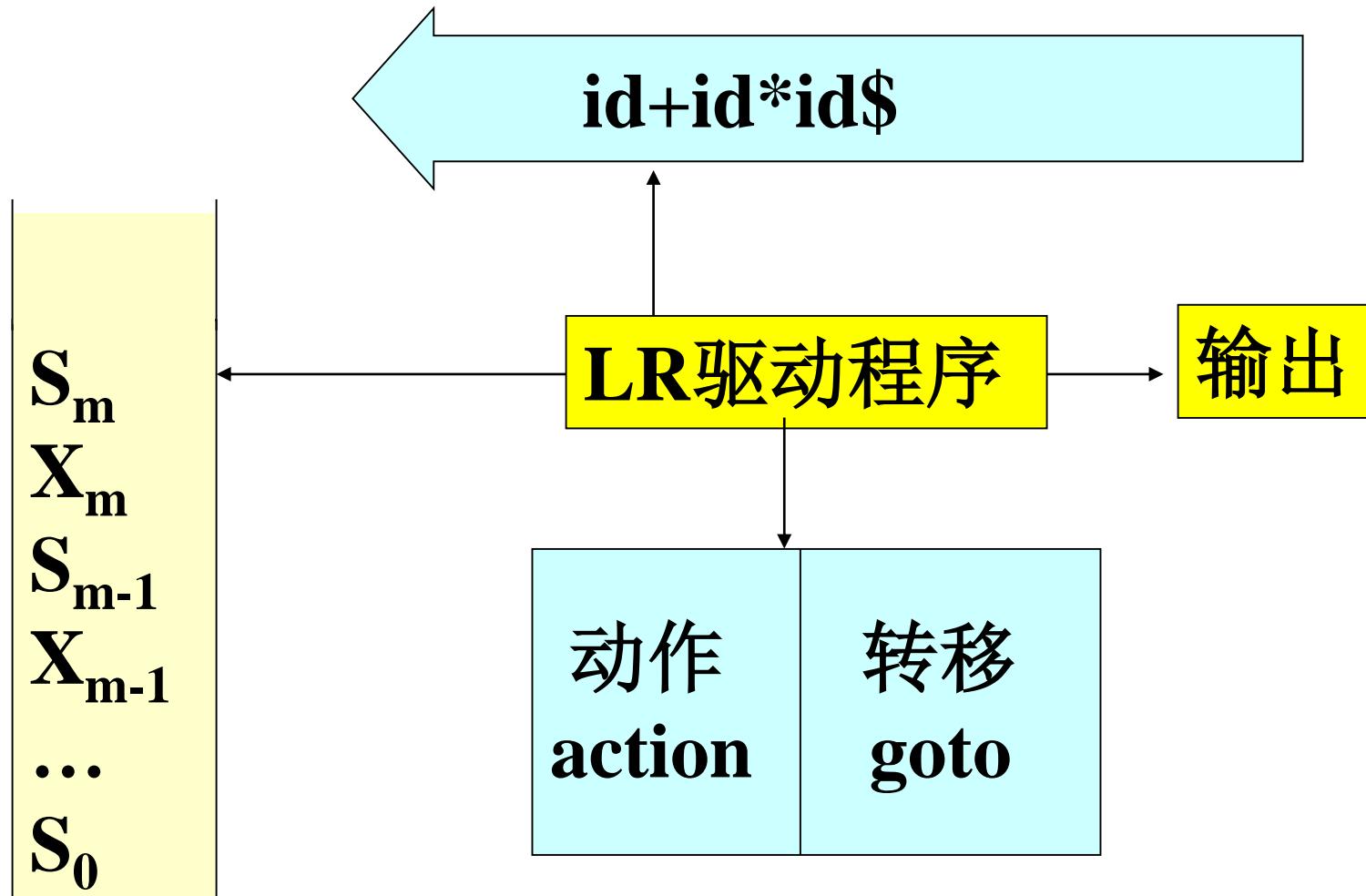
□ 分析栈

- 放置分析器状态和文法符号





LR分析器的结构





LR分析表

- 分析表的第一列是状态、第二栏是action部分，由 $(|T|+1)$ 列构成、第三栏是goto部分，由 $|V|$ 列构成：

$$\text{action}[s,a] = \begin{cases} \text{移进 } S_i & a \text{ 和状态 } i \text{ 进栈} \\ \text{归约 } r_j : & \left\{ \begin{array}{l} \text{出栈 } k \text{ 项} \\ A \text{ 和 } \text{goto}[s', A] \text{ 进栈} \end{array} \right. \\ \text{接受} \\ \text{出错} \end{cases}$$

其中， s 是状态， a 是读入的终结符（单词）或 $\$$ ；
 k 是 j 号产生式 $A \rightarrow \beta$ 右部(β)的长度；
 s' 是出栈 k 项后新的栈顶元素中的状态。

1. $E \rightarrow E + T$ 2. $E \rightarrow T$
 3. $T \rightarrow T^* F$ 4. $T \rightarrow F$
 5. $F \rightarrow (E)$ 6. $F \rightarrow id$

表达式文法的SLR分析表

状态	ACTION					GOTO			
	id	+	*	()	\$	E	T	F
0	S_5			S_4			1	2	3
1		S_6				acc			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5			S_4			9	3	
7	S_5			S_4				10	
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

龙书图4-37

使用图4-31的
LR(0)自动机



LR分析器的驱动程序

- 把状态 0 和符号 \$ 压入初始为空的栈里。
- 设栈顶元素中的状态为 s, 当前读入的符号为 a。
- 反复执行以下各动作, 直到分析成功或发现语法错误为止:
 - 1 (移进) 若 $\text{action}[s, a] = S_i$, 则把 a 和状态 i 压进栈,
读下一个输入符号到 a 中。
 - 2 (归约) 若 $\text{action}[s, a] = r_j$ ($j: A \rightarrow x_{m-k+1}x_{m-k+2}\dots x_m$),
则出栈 k 项, 把 A 和 $s_{\text{new}} = \text{goto}[s', A]$ 进栈。其中
 s' 是出栈 k 项后新的栈顶元素中的状态。
 - 3 (接受) 若 $\text{action}[s, \$] = \text{accept}$, 则分析成功, 结束。
 - 4 (出错) 若 $\text{action}[s, a] = \text{error}$, 则转出错处理程序。

LR分析过程示例

使用龙书图4.37的分析表



栈	输入	动作
\$0	id+id*id\$	S ₅
\$0id5	+id*id\$	r ₆ F→id
\$0F3	+id*id\$	r ₄ T→F
\$0T2	+ id*id\$	r ₂ E→T
\$0E1	+id *id\$	S ₆
\$0E1+6	id *id\$	S ₅
\$0E1+6id5	*id\$	r ₆ F→id
\$0E1+6F3	*id\$	r ₄ T→F
\$0E1+6T9	*id\$	S ₇



LR分析过程示例(续) 使用龙书图4.37的分析表

栈	输入	动作
\$0E1+6T9*7	id \$	S ₅
\$0E1 +6T9*7 id5	\$	r ₆ F→id
\$0E 1+6T9*7 F <u>10</u>	\$	r ₃ T→T*T
\$0E1+6T9	\$	r ₁ E→E+T
\$0E1	\$	accept

LR分析的关键是构造LR(k)分析表。



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



构造LR(0)分析表

□ 根据DFA构造LR(0)分析表M:

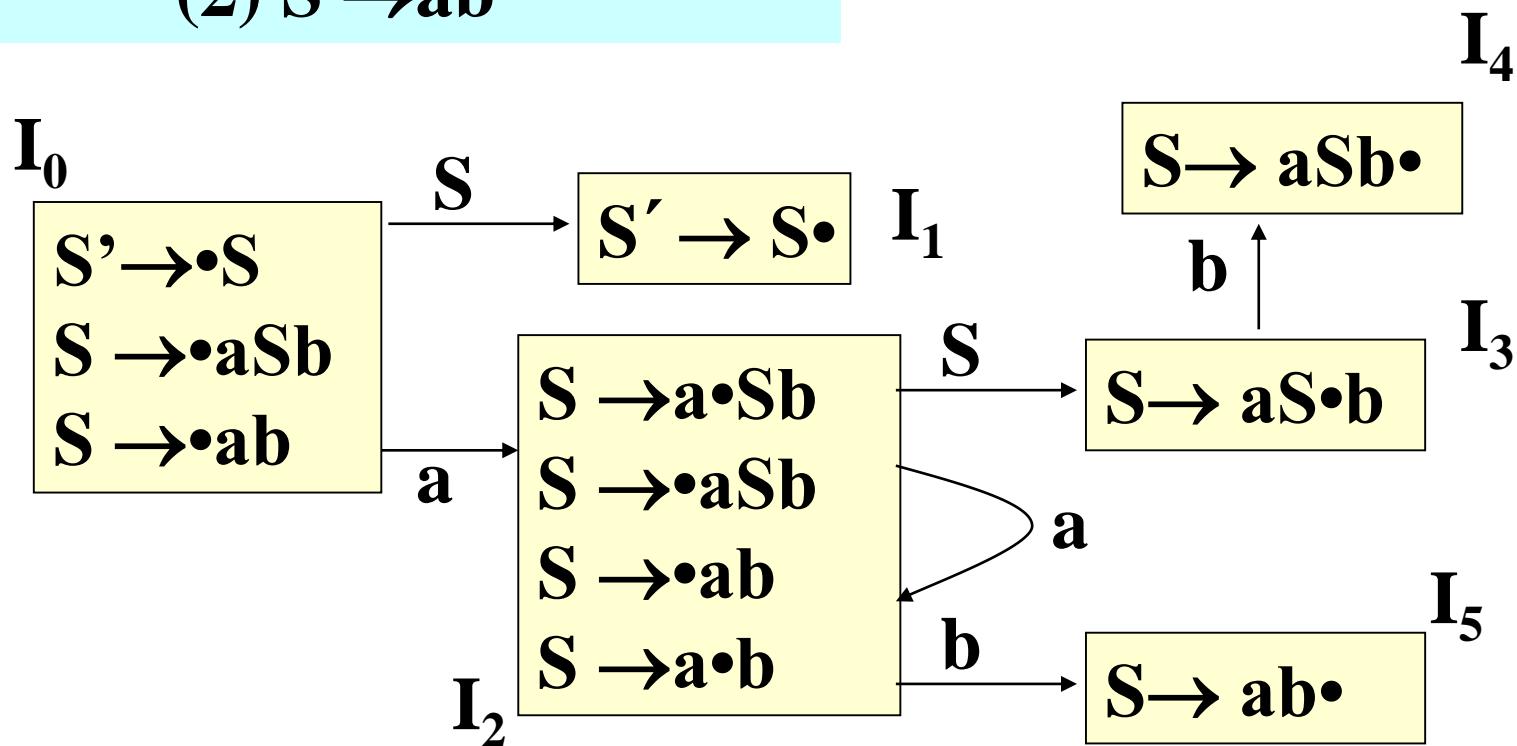
- DFA中的每个状态对应分析表中的一行
- 对于DFA中的每一个从状态 i 到状态 j 的转移
 - 如果转移符号为终结符 a : 在相应的表项 $M[i, a]$ 中填写移进动作 S_j
 - 如果转移符号为非终结符 A : 在相应的表项 $M[i, A]$ 中填写要转移到的状态 j
- 对于包含归约项 $A \rightarrow \alpha \bullet$ 的状态 i
 - 对于所有终结符 a , 在相应的表项 $M[i, a]$ 中填写归约动作 (r_k) , 其中 k 是产生式 $A \rightarrow \alpha$ 的编号。

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(0)分析表



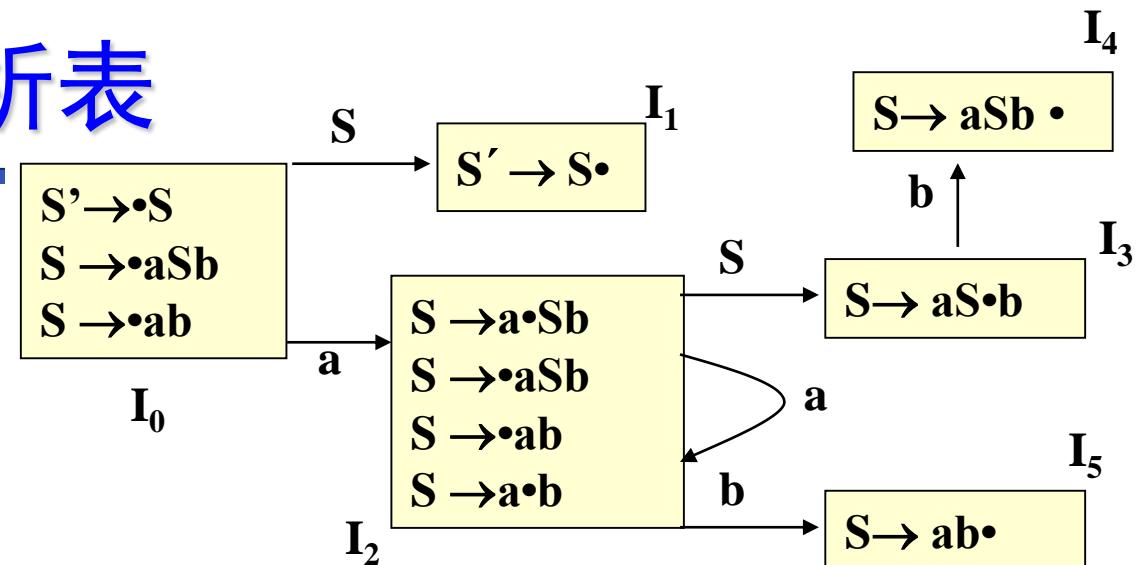
例：LR(0)分析表的构造

- 例：
- (0) $S' \rightarrow S$
 - (1) $S \rightarrow aSb$
 - (2) $S \rightarrow ab$



例：LR(0)分析表

文法：
 (0) $S' \rightarrow S$
 (1) $S \rightarrow aSb$
 (2) $S \rightarrow ab$

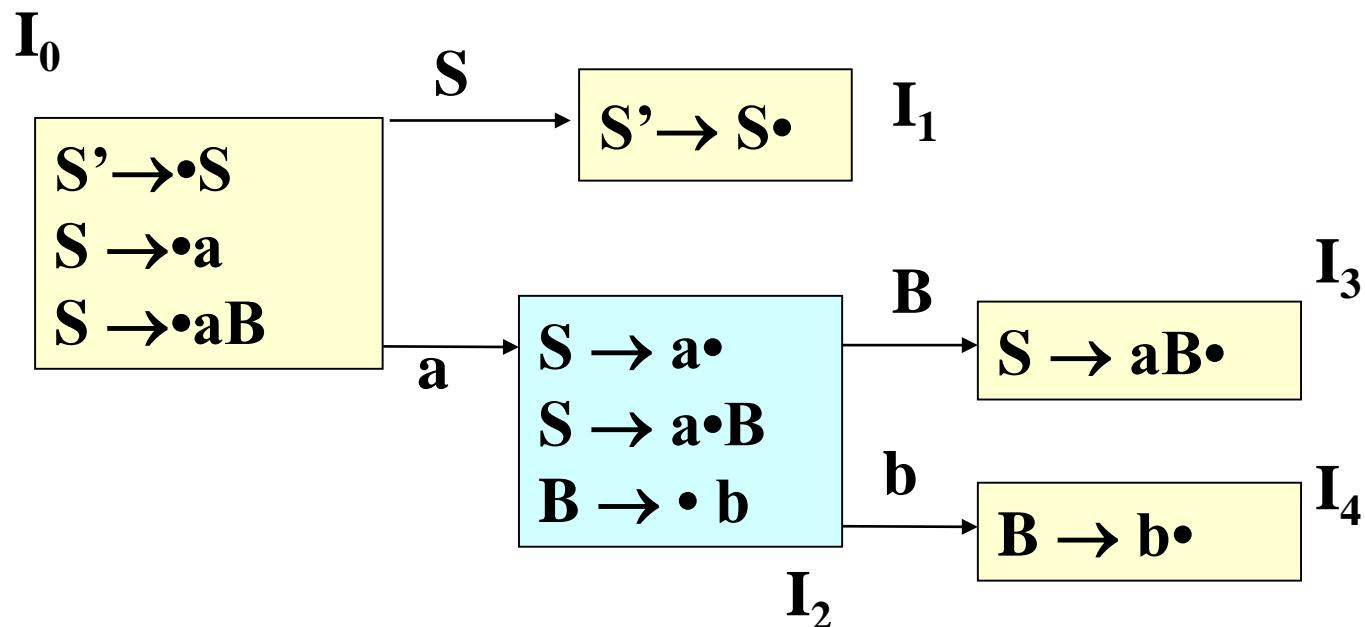


	a	b	\$	S
0	S_2			1
1			acc	
2	S_2	S_5		3
3		S_4		
4	r_1	r_1	r_1	
5	r_2	r_2	r_2	



LR(0)分析表中的冲突

- 假设我们为 $S \rightarrow a \mid aB \quad B \rightarrow b$ 构造 LR(0)项集。
 (0) $S' \rightarrow S$ (1) $S \rightarrow a$ (2) $S \rightarrow aB$ (3) $B \rightarrow b$
- I_2 状态中出现了移进-归约冲突。





解决办法：SLR分析表

- 根据Follow集来选择是否要进行归约。
- 如果 $I=\{X \rightarrow \alpha \cdot b \beta, A \rightarrow \alpha \cdot, B \rightarrow \alpha \cdot\}$
若{b}, Follow(A), Follow(B)两两不交,
则面对当前读入符号a, 状态I的解决方法：
 1. 若 $a=b$, 则移进。
 2. 若 $a \in \text{Follow}(A)$, 则用 $A \rightarrow \alpha$ 进行归约。
 3. 若 $a \in \text{Follow}(B)$, 则用 $B \rightarrow \alpha$ 进行归约。
 4. 此外, 报错。
- 这种解决方法比较简单, 因此称作**SLR分析**,
由此构造的分析表, 称作**SLR分析表**。

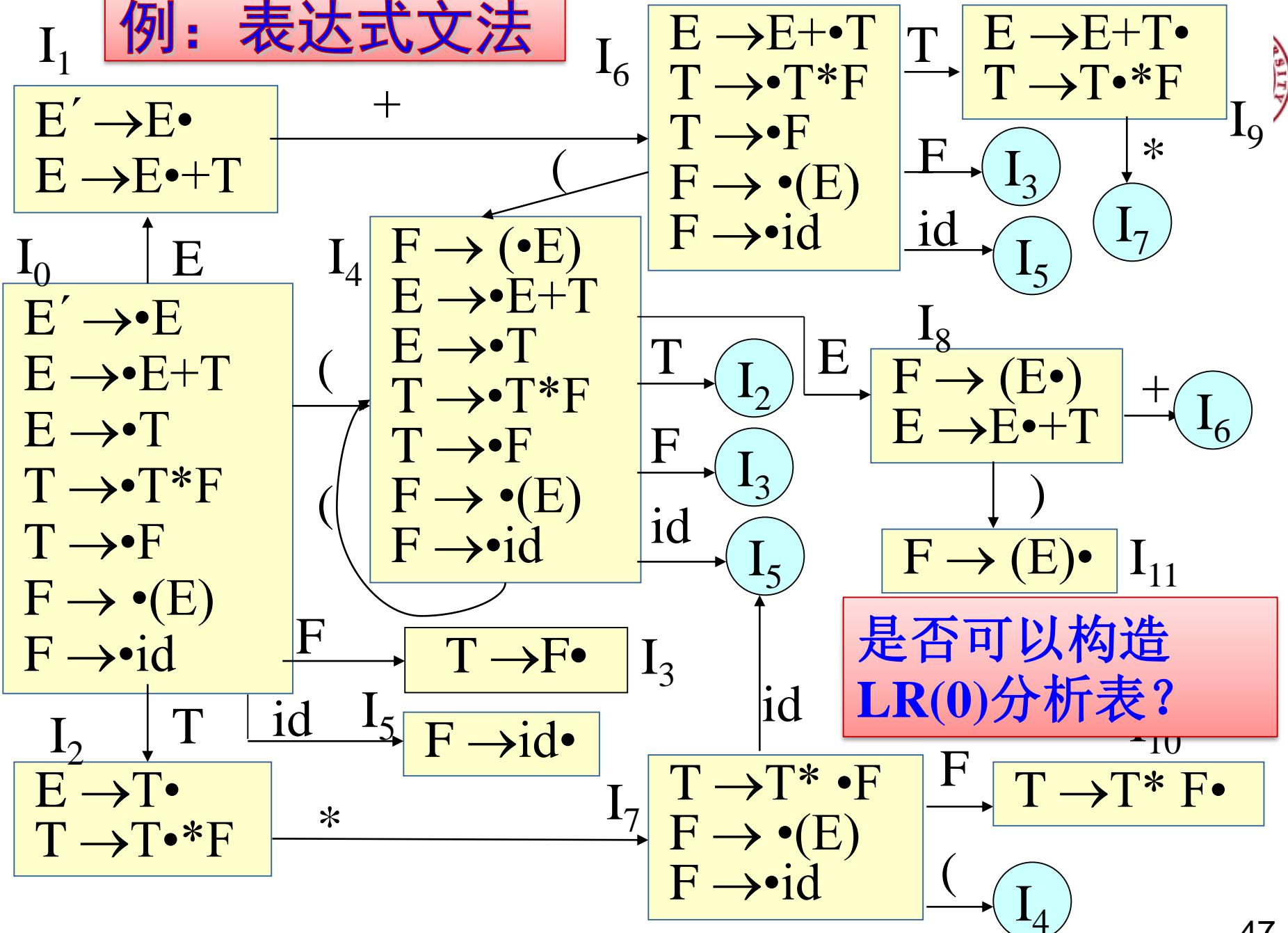


构造SLR分析表

- 根据LR(0)的DFA构造SLR分析表M：
 - 为所有非终结符计算相应的Follow集合
 - 对于包含归约项 $A \rightarrow \alpha^*$ 的状态 i
 - 为所有终结符 $a \in \text{Follow}(A)$, 在相应的表项 $M[i, a]$ 中填写归约动作 r_i , 其中 i 是产生式 $A \rightarrow \alpha$ 的编号。
 - 所有其他表项都按照与LR(0)分析表相同的方式

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SLR分析表

例：表达式文法





计算Follow集

存在冲突的项集：

$$I_1: \{ E' \rightarrow E^* \quad E \rightarrow E^* + T \}$$

$$I_2: \{ E \rightarrow T^* \quad T \rightarrow T^* * F \}$$

$$I_9: \{ E \rightarrow E + T^* \quad T \rightarrow T^* * F \}$$

文法的Follow集：

E'	\$		
E	\$)	+
T	\$)	+
F	\$)	*

该文法的SLR分析表参见龙书图4.37。



SLR的原理：活前缀 (1)

- LR(0)自动机刻画了可能出现在语法分析栈中的文法符号串。
 - 直接把项当作状态，可以构造得到一个NFA。
 - 然后确定化得到DFA就是LR(0)自动机
- 活前缀 (Viable Prefix, 也译作可行前缀):
 - 可以出现在移进-归约语法分析器栈中的右句型前缀
 - 定义：某个右句型的前缀，且没有越过该句型的句柄的右端。
- 有效项：
 - 如果存在 $S \Rightarrow^* \alpha A w \Rightarrow^* \alpha \beta_1 \beta_2 w$, 那么我们说项 $A \rightarrow \beta_1 \cdot \beta_2$ 对 $\alpha \beta_1$ 有效。



SLR的原理：活前缀 (2)

- 如果我们知道 $A \rightarrow \beta_1 \cdot \beta_2$ 对 $\alpha\beta_1$ 有效，
 - 当 β_2 不等于空，表示句柄尚未出现在栈中，应该移进或者等待归约；
 - 如果 β_2 等于空，表示句柄出现在栈中，应该归约。
- 如果某个时刻存在两个有效项要求执行不同的动作，那么就应该设法解决冲突。
 - 冲突实际上表示可行前缀可能是两个最右句型的前缀，第一个包含了句柄，而另一个尚未包含句柄
 - $E + T + id$
 - $E + T^* id$

也可能都认为包含句柄，但是规则不一样
 - SLR解决冲突的思想：假如要按照 $A \rightarrow \beta$ 进行归约，那么得到的新句型中 A 后面跟着的是下一个输入符号。因此只有当下一个输入在 $FOLLOW(A)$ 中时才可以归约。



SLR的原理：活前缀 (3)

- 如果在文法G的LR(0)自动机中，从初始状态出发，沿着标号为 γ 的路径到达一个状态，那么这个状态对应的项集就是 γ 的有效项集。
- 回顾确定分析动作的方法，就可以知道我们实际上是按照有效项来确定的。
 - 为了避免冲突，归约时要求下一个输入符号在 $\text{FOLLOW}(A)$ 中。

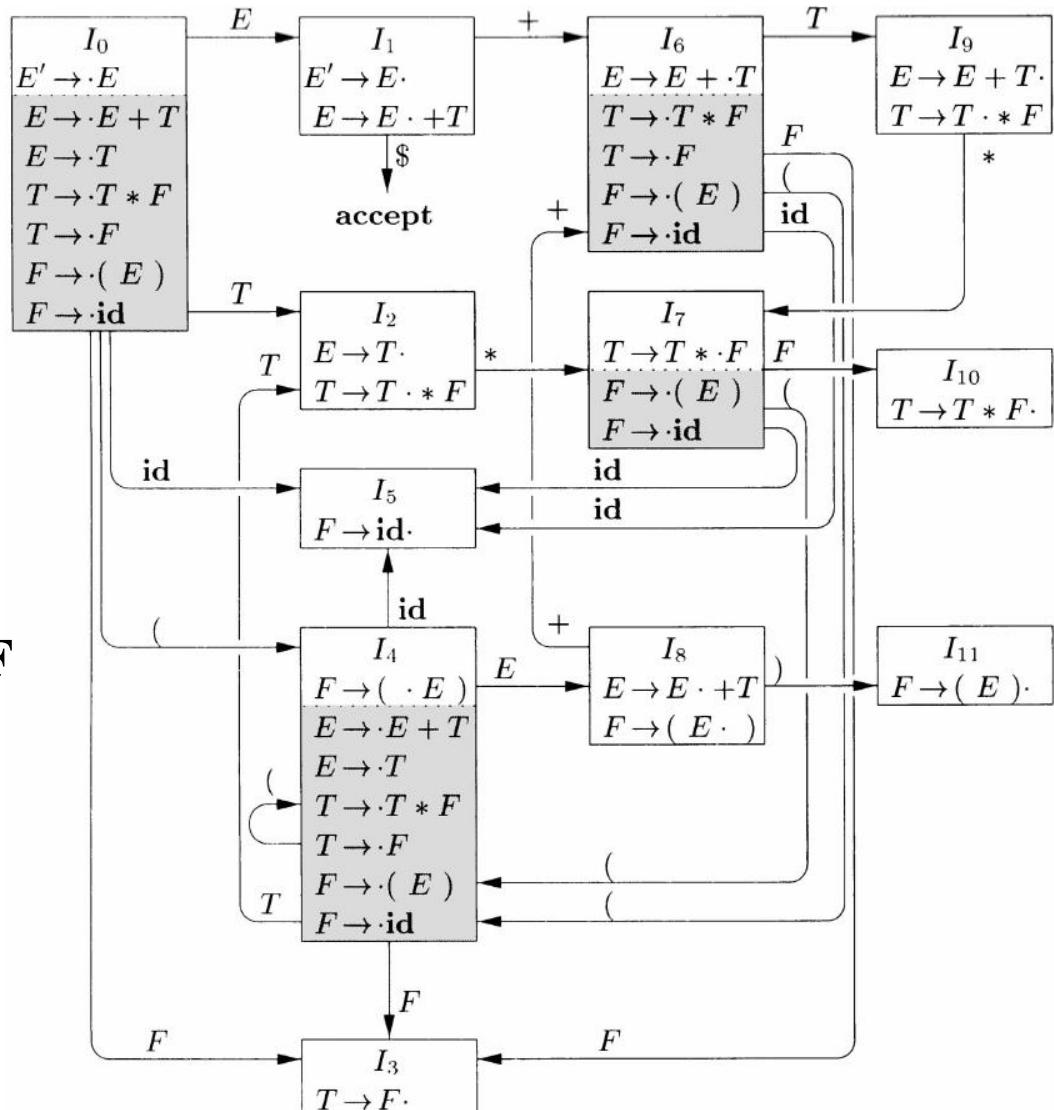
活前缀/有效项的例子

- 活前缀 $E + T^*$
- 对应的 LR(0) 项

- $T \rightarrow T^* \cdot F$
- $F \rightarrow \cdot (E)$
- $F \rightarrow \cdot id$

- 对应的最右推导

- $E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T^* F$
- $E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T^* F \Rightarrow E + T^*(E)$
- $E' \Rightarrow E \Rightarrow E + T \Rightarrow E + T^* F \Rightarrow E + T^* id$





SLR语法分析器的弱点（1）

□ SLR技术解决冲突的方法：

- 项集中包含 $[A \rightarrow a \cdot]$ 时，按照 $A \rightarrow a$ 进行归约的条件是下一个输入符号 a 可以在某个句型中跟在 A 之后。
 - 如果对于 a 还有其它的移进/归约操作，则出现冲突
- 假设此时栈中的符号串为 βa ，
 - 如果 $\beta A a$ 不可能是某个最右句型的前缀，那么即使 a 在某个句型中跟在 A 之后，仍然不应该按照 $A \rightarrow a$ 归约。
 - 进行归约的条件更加严格可以降低冲突的可能性



SLR语法分析器的弱点 (2)

- $[A \rightarrow \alpha \cdot]$ 出现在项集中的条件：
 - 首先 $[A \rightarrow \cdot \alpha]$ 出现在某个项集中，然后逐步读入/归约到 α 中的符号，点不断后移，到达末端。
 - 而 $[A \rightarrow \cdot \alpha]$ 出现的条件是 $B \rightarrow \beta \cdot A \gamma$ 出现在项集中
 - 期望首先按照 $A \rightarrow \alpha$ 归约，然后将 $B \rightarrow \beta \cdot A \gamma$ 中的点后移到 A 之后。
 - 显然，在按照 $A \rightarrow \alpha$ 归约时要求下一个输入符号是 γ 的第一个符号。
 - 但是从 LR(0) 项集中不能确定这个信息。



作业

- 10月23日交
- LR分析
 - 4.5.1, 4.5.2
- LR(0), SLR
 - 4.6.2, 4.6.3, 4.6.5