

编译原理

第六章 LR语法分析技术 (1)

方徽星

扬州大学 信息工程学院(505)

fanghuixing@yzu.edu.cn

2018年春季学期

本章主要内容

一. 自下向上语法分析

二. LR分析

- SLR
- LR(1)
- LALR

三. 使用二义性文法

四. Yacc

第一节 自下向上语法分析

1.1 归约

- 为输入串构造分析树：
 - 从叶子结点开始，朝着根结点方向逆序前进
 - 即把输入串**归约**成文法的**开始符号**
- 每步归约：当**子串**和某**产生式体**匹配就可用该产生式头代替子串

1.1 归约

- 例：考虑文法

- $S \rightarrow aABe$

- $A \rightarrow Abc \mid b$

- $B \rightarrow d$

把句子 $abbcde$ 进行归约

1.1 归约

• 例：考虑文法

- $S \rightarrow aABe$

- $A \rightarrow Abc \mid b$

- $B \rightarrow d$

把句子 $abbcde$ 进行归约

$$a\textcolor{red}{b}bcde \xrightarrow[\text{归约}]{A \rightarrow b} a\textcolor{red}{A}bcde$$

1.1 归约

• 例：考虑文法

- $S \rightarrow aABe$

- $A \rightarrow Abc \mid b$

- $B \rightarrow d$

把句子 $abbcde$ 进行归约

$$a\mathbf{A}bcde \xrightarrow[\text{归约}]{A \rightarrow Abc} a\mathbf{A}de$$

1.1 归约

• 例：考虑文法

- $S \rightarrow aABe$

- $A \rightarrow Abc \mid b$

- $B \rightarrow d$

把句子 $abbcde$ 进行归约

$$aA\textcolor{red}{d}e \xrightarrow[\text{归约}]{\textcolor{red}{B \rightarrow d}} aA\textcolor{red}{B}e$$

1.1 归约

• 例：考虑文法

- $S \rightarrow aABe$

- $A \rightarrow Abc \mid b$

- $B \rightarrow d$

把句子 $abbcde$ 进行归约

$$aABe \xrightarrow[\text{归约}]{S \rightarrow aABe} S$$

1.1 归约

- 例：考虑文法
 - $S \rightarrow aABe$
 - $A \rightarrow Abc \mid b$
 - $B \rightarrow d$

*abbcede*的最左归约：

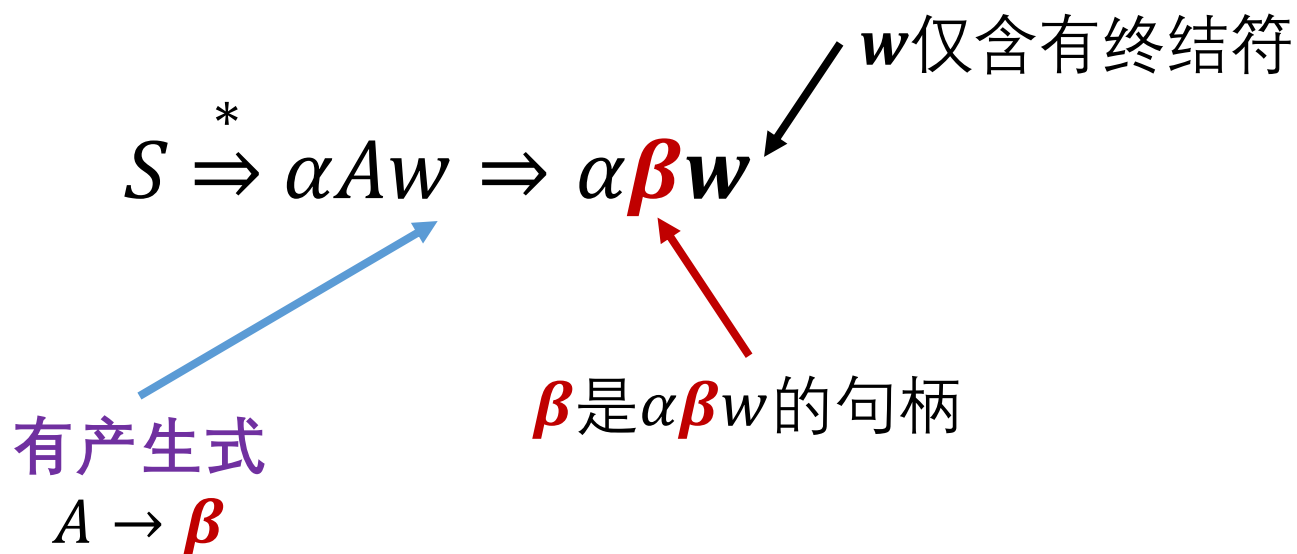
abbcede \rightarrow *aAbcde* \rightarrow *aAde* \rightarrow *aABe* \rightarrow *S*

*abbcede*的最右推导：

S \Rightarrow *aABe* \Rightarrow *aAde* \Rightarrow *aAbcde* \Rightarrow *abbcede*

1.2 句柄

- 考虑最右句型（最右推导可得到的句型）



1.2 句柄

- 考虑文法

- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow (E)$
- $E \rightarrow \mathbf{id}$

每个右句型的句柄
使用下划线标记



最右推导： $E \Rightarrow \underline{E * E}$
 $\Rightarrow E * \underline{E + E}$
 $\Rightarrow E * E + \underline{\mathbf{id}}$
 $\Rightarrow E * \underline{\mathbf{id}} + \mathbf{id}$
 $\Rightarrow \underline{\mathbf{id}} * \mathbf{id} + \mathbf{id}$

句柄右边的串
仅含终结符

1.2 用栈实现分析

- 用**栈保存文法符号**，用输入缓冲区保存要分析的串 w ，用\$标记栈底和输入串的右端
- 初始：栈仅含符号\$，串 w 在输入中

栈	输入
\$	w \$

1.2 用栈实现分析

- 然后：**移动**若干个(可以0个)输入符号**入栈**，直到**句柄** β 出现在栈顶，把 β **归约**成恰当的产生式左部
- 重复上述过程，直到发现错误或者栈中只含有开始符号且输入串为空

栈	输入
$\$S$	$\$$

↑
分析成功

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

[illegible]

1.2 用栈实现分析

- 例：分析输入串**id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	

1.2 用栈实现分析

- 例：分析输入串**id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow id$ 归约
\$ E	* id + id \$	

1.2 用栈实现分析

- 例：分析输入串**id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow id$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow id$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	移进
\$ E * id	+id \$	

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow \text{id}$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	移进
\$ E * id	+ id \$	按 $E \rightarrow \text{id}$ 归约
\$ E * E	+ id \$	

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow \text{id}$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	移进
\$ E * id	+id \$	按 $E \rightarrow \text{id}$ 归约
\$ E * E	+id \$	移进
\$ E * E +	id \$	

1.2 用栈实现分析

- 例：分析输入串**id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow \text{id}$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	移进
\$ E * id	+id \$	按 $E \rightarrow \text{id}$ 归约
\$ E * E	+id \$	移进
\$ E * E +	id \$	移进
\$ E * E + id	\$	

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow \text{id}$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	移进
\$ E * id	+id \$	按 $E \rightarrow \text{id}$ 归约
\$ E * E	+id \$	移进
\$ E * E +	id \$	移进
\$ E * E + id	\$	按 $E \rightarrow \text{id}$ 归约
\$ E * E + E	\$	

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

栈	输入	动作
\$	id * id + id \$	移进
\$ id	* id + id \$	按 $E \rightarrow \text{id}$ 归约
\$ E	* id + id \$	移进
\$ E *	id + id \$	移进
\$ E * id	+id \$	按 $E \rightarrow \text{id}$ 归约
\$ E * E	+id \$	移进
\$ E * E +	id \$	移进
\$ E * E + id	\$	按 $E \rightarrow \text{id}$ 归约
\$ E * E + E	\$	按 $E \rightarrow E + E$ 归约
\$ E * E	\$	

1.2 用栈实现分析

- 例：分析输入串 **id * id + id**

栈	输入	动作
\$	id * id + id\$	移进
\$id	* id + id\$	按 $E \rightarrow id$ 归约
\$E	* id + id\$	移进
\$E *	id + id\$	移进
\$E * id	+id\$	按 $E \rightarrow id$ 归约
\$E * E	+id\$	移进
\$E * E +	id\$	移进
\$E * E + id	\$	按 $E \rightarrow id$ 归约
\$E * E + E	\$	按 $E \rightarrow E + E$ 归约
\$E * E	\$	按 $E \rightarrow E * E$ 归约
\$E	\$	接受

1.3 移进-归约分析的冲突

- 有些上下文无关文法不能使用移进-归约分析
- 例：考虑如下文法

$stmt \rightarrow$ **if** *expr* **then** *stmt*
 | **if** *expr* **then** *stmt* **else** *stmt*
 | **other**

如果移进-归约分析过程中出现下面的情况：

栈	输入
... if <i>expr</i> then <i>stmt</i>	else ...\$

无法知道**if** *expr* **then** *stmt*是否为句柄，产生移进-归约冲突

1.3 移进-归约分析的冲突

- 例：考虑如下文法

- $stmt \rightarrow \mathbf{id}(p_list) \mid expr = expr$
- $p_list \rightarrow p_list, para \mid para$
- $para \rightarrow \mathbf{id}$
- $expr \rightarrow \mathbf{id}(expr_list) \mid \mathbf{id}$
- $expr_list \rightarrow expr_list, expr \mid expr$

若 $p(i, j)$ 经过词法分析变为序列： $\mathbf{id}(\mathbf{id}, \mathbf{id})$ ，然后交给语法分析器移进前三个词法单元进栈后，栈顶 \mathbf{id} 必须归约：

栈	输入
... $\mathbf{id}(\mathbf{id}$, $\mathbf{id}) \dots$

- 如果 p 是过程则按照 $para \rightarrow \mathbf{id}$ 进行归约
- 如果 p 是表达式，则按照 $expr \rightarrow \mathbf{id}$ 进行归约

第二节 LR分析

2 LR分析

- **LR(k)分析技术**

- L：从左向右(**Left to right**)扫描输入
- R：反向构造最右推导(**Rightmost Derivation**)序列
- k：决定分析动作时向前看k个输入符号

- **三种LR分析表构造技术**

- 简单LR(**Simple LR**)：容易实现，功能最弱
- 规范LR(**Canonical LR**)：功能最强，代价最大
- 向前搜索的LR(**Look-Ahead LR**)：介于二者之间

2.1 项和LR(0)自动机

一个移进-归约语法分析器如何确定

何时移进，何时归约？

2.1 项和LR(0)自动机

- LR语法分析器通过维护一些**状态**，用这些状态来**表明**我们在语法分析过程中**所处的位置**，从而做出移进-归约决定；状态是“**项**”的集合
- 文法G的一个LR(0)**项**：G的一个产生式P再加上一个位于P的体中某处的点D
- 例：产生式 $A \rightarrow XYZ$ 产生了四个项：
 1. $A \rightarrow \bullet XYZ$
 2. $A \rightarrow X \bullet YZ$
 3. $A \rightarrow XY \bullet Z$
 4. $A \rightarrow XYZ \bullet$

2.1 项和LR(0)自动机

- 项指明了在语法分析过程中，我们**已经**看到了一个产生式的**哪些部分**
- 例：
 1. 项 $A \rightarrow \bullet XYZ$ 表明**希望**接下来在**输入**中看到一个从**XYZ**推导得到的串
 2. 项 $A \rightarrow X \bullet YZ$ 说明刚刚(**已经**)**在输入中看到了一个可以由X推导得到的串**，且**希望**接下来看到一个能从**YZ**推导得到的串
 3. 项 $A \rightarrow XYZ \bullet$ 表示我们**已经**看到了产生式体**XYZ**，已经是时候把**XYZ**归约为**A**了

2.1 项和LR(0)自动机

- 基于一组**项集**可以构造一个**确定的有限自动机**, 用于做出语法分析**决定**
 - 项集：**规范LR(0)项集族**(Canonical LR(0) Collection)
 - 确定的有限自动机：**LR(0)自动机**
 - LR(0)自动机的每个**状态**代表了规范LR(0)项集族中的一个**项集**
- 构造文法的规范LR(0)项集族，需要定义
 - 增广文法(Augmented Grammar)
 - CLOSURE函数：**项集闭包**
 - GOTO函数：**转移**

2.1 项和LR(0)自动机

- 文法 G (其中 S 为开始符号)的增广文法 G' :
 - 在 G 中加上新开始符号 S' 和产生式 $S' \rightarrow S$
 - 引入 S' 后 :

输入符号串
被接受

iff

要使用规则
 $S' \rightarrow S$
进行归约

2.1 项和LR(0)自动机

- 项集闭包：如果 I 是文法 G 的一个项集，那么 **$CLOSURE(I)$** 可由如下规则构建
 - 一开始，将 I 中的各个项加入到 $CLOSURE(I)$ 中
 - 如果项 $A \rightarrow \alpha \bullet B \beta \in CLOSURE(I)$ ，有 $B \rightarrow \gamma$ 且 $B \rightarrow \bullet \gamma \notin CLOSURE(I)$ ，则将项 $B \rightarrow \bullet \gamma$ 放入 $CLOSURE(I)$



- ✓ 从 $B\beta$ 推导得到的子串的某个前缀可以从 B 推导得到
- ✓ 从 B 推导时必然用到 B 的某个产生式
- ✓ 因此加了 B 的产生式对应的项

2.1 项和LR(0)自动机

- 例：考虑增广的表达式文法

1. $E' \rightarrow E$

2. $E \rightarrow E + T \mid T$

3. $T \rightarrow T * F \mid F$

4. $F \rightarrow (E) \mid \mathbf{id}$

如果 $i = \{[E' \rightarrow \bullet E]\}$, 则 $CLOSURE(i) = ?$

2.1 项和LR(0)自动机

- 例：考虑增广的表达式文法

1. $E' \rightarrow E$

2. $E \rightarrow E + T \mid T$

3. $T \rightarrow T * F \mid F$

4. $F \rightarrow (E) \mid \mathbf{id}$

如果 $i = \{[E' \rightarrow \bullet E]\}$, 则 $CLOSURE(i) = ?$

- 因为 $E' \rightarrow \bullet E \in i$, 由规则1,

$$E' \rightarrow \bullet E \in CLOSURE(i)$$

2.1 项和LR(0)自动机

- 例：考虑增广的表达式文法

1. $E' \rightarrow E$

2. $E \rightarrow E + T \mid T$

3. $T \rightarrow T * F \mid F$

4. $F \rightarrow (E) \mid \mathbf{id}$

如果 $i = \{[E' \rightarrow \bullet E]\}$, 则 $CLOSURE(i) = ?$

- 再由规则2, 及 $E \rightarrow E + T \mid T$

$$E \rightarrow \bullet E + T \in CLOSURE(i)$$

$$E \rightarrow \bullet T \in CLOSURE(i)$$

2.1 项和LR(0)自动机

- 例：考虑增广的表达式文法

1. $E' \rightarrow E$

2. $E \rightarrow E + T \mid T$

3. $T \rightarrow T * F \mid F$

4. $F \rightarrow (E) \mid \mathbf{id}$

如果 $i = \{[E' \rightarrow \bullet E]\}$, 则 $CLOSURE(i) = ?$

- 再由规则2, 及 $T \rightarrow T * F \mid F$

$$T \rightarrow \bullet T * F \in CLOSURE(i)$$

$$T \rightarrow \bullet F \in CLOSURE(i)$$

2.1 项和LR(0)自动机

- 例：考虑增广的表达式文法

1. $E' \rightarrow E$

2. $E \rightarrow E + T \mid T$

3. $T \rightarrow T * F \mid F$

4. $F \rightarrow (E) \mid \mathbf{id}$

如果 $i = \{[E' \rightarrow \bullet E]\}$, 则 $CLOSURE(i) = ?$

- 再由规则2, 及 $F \rightarrow (E) \mid \mathbf{id}$

$$F \rightarrow \bullet(E) \in CLOSURE(i)$$

$$F \rightarrow \bullet \mathbf{id} \in CLOSURE(i)$$

结束！

2.1 项和LR(0)自动机

- 例：考虑增广的表达式文法

1. $E' \rightarrow E$

2. $E \rightarrow E + T \mid T$

3. $T \rightarrow T * F \mid F$

4. $F \rightarrow (E) \mid \mathbf{id}$

如果 $i = \{[E' \rightarrow \bullet E]\}$, 则

$CLOSURE(i) =$

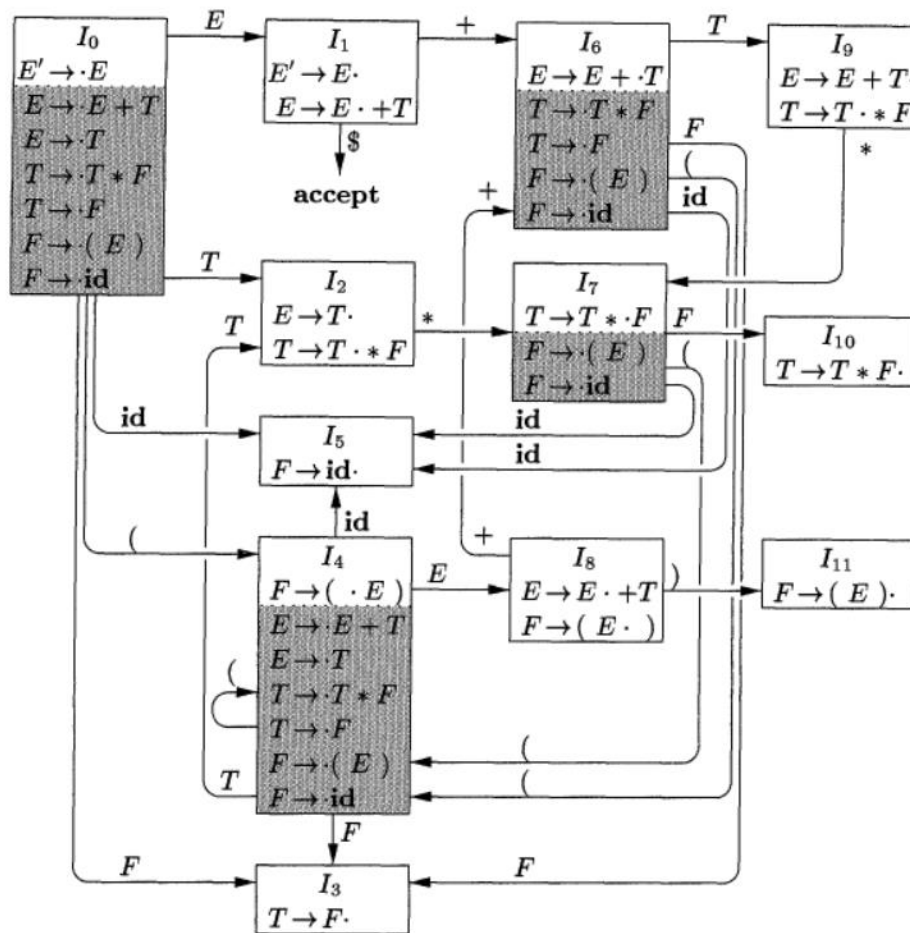
$$\{ [E' \rightarrow \bullet E], [E \rightarrow \bullet E + T], \\ [E \rightarrow \bullet T], [T \rightarrow \bullet T * F], \\ [T \rightarrow \bullet F], [F \rightarrow \bullet (E)], \\ [F \rightarrow \bullet \mathbf{id}] \}$$

2.1 项和LR(0)自动机

- 项的分类
 - 内核项：初始项 $S' \rightarrow \bullet S$ 以及点不在最左端的所有项
 - 非内核项：除了 $S' \rightarrow \bullet S$ 之外的点在最左端的所有项
- 感兴趣的每个项集都是某个内核项集合的闭包(?)
- 计算闭包时加入的项不是内核项，可以省略所有非内核项以减少内存(因为可以计算出来)

2.1 项和LR(0)自动机

表达式文法的LR(0)自动机



阴影部分表示
非内核项

2.1 项和LR(0)自动机

- **GOTO函数：**

$$GOTO(\overset{\text{项集}}{\downarrow} \textcolor{red}{I}, \overset{\text{文法符号}}{\downarrow} \textcolor{violet}{X})$$

定义LR(0)自动机的一个转换：

当输入为 $\textcolor{violet}{X}$ 时，从 $\textcolor{red}{I}$ 对应的状态出发的转换

如果有形如 $[A \rightarrow \alpha \bullet \textcolor{violet}{X} \beta] \in \textcolor{red}{I}$ ，则

$$GOTO(\textcolor{red}{I}, \textcolor{violet}{X}) \stackrel{\text{def}}{=} \text{K的闭包}$$

其中 $\text{K} \stackrel{\text{def}}{=} \text{所有形如}[A \rightarrow \alpha \textcolor{violet}{X} \bullet \beta]$ 的项组成的项集

2.1 项和LR(0)自动机

- 例：令 $i = \{[E' \rightarrow E\bullet], [E \rightarrow E\bullet + T]\}$,
求 $GOTO(i, +)$

2.1 项和LR(0)自动机

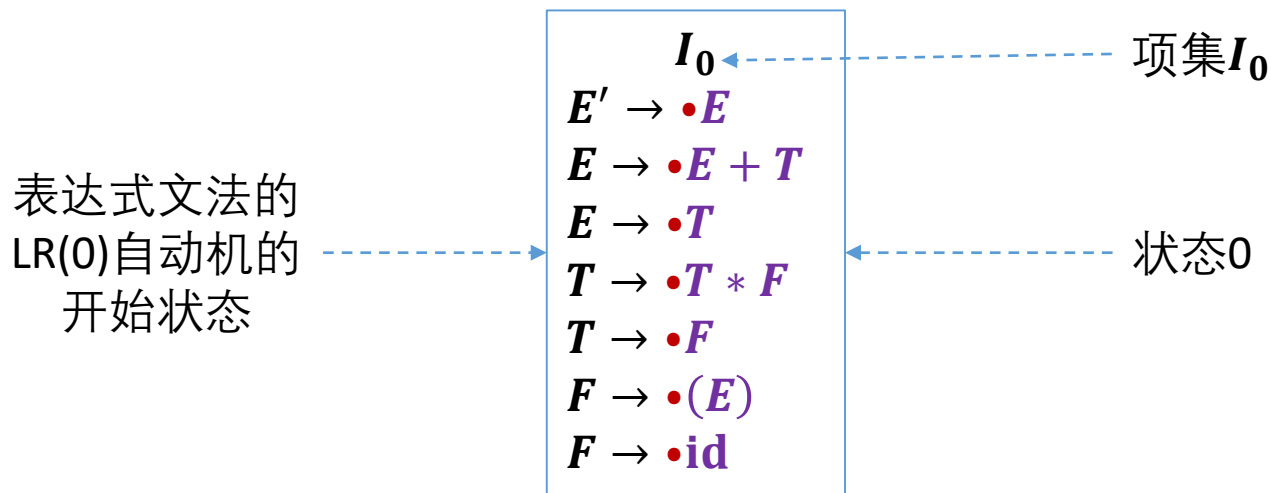
- 有了CLOSURE和GOTO之后，就可以计算项集族

```
void items( $G'$ ) {  
     $C = \{CLOSURE(\{[S' \rightarrow \bullet S]\})\}$  ;  
    repeat  
        for ( $C$  中的每个项集  $i$ ) {  
            for (每个文法符号  $X$ ) {  
                if ( $GOTO(i, X) \neq \emptyset \wedge GOTO(i, X) \notin C$ ) {  
                     $C = C \cup \{GOTO(i, X)\}$  ;  
                }  
            }  
        }  
    until 在某轮循环中没有新的项被加入  $C$  中 ;  
}
```

规范LR(0)项集族的计算

2.1 项和LR(0)自动机

- SLR分析技术的中心思想是根据文法构造出LR(0)自动机
 - 自动机的状态是规范LR(0)项集族中的元素
 - 自动机的转换由GOTO函数给出



LR(0)自动机如何帮助做出移进-归约决定的呢?

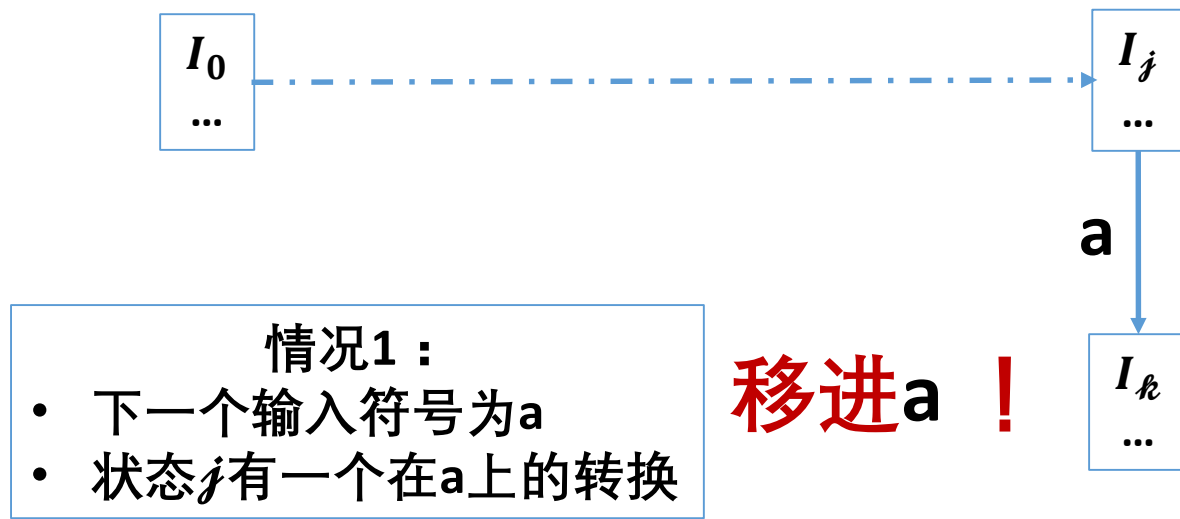
2.1 项和LR(0)自动机

- 移进-归约决定可以按照如下方式做出
 - 假设文法符号串 γ 使LR(0)自动机从开始状态0运行到某个状态 j



2.1 项和LR(0)自动机

- 移进-归约决定可以按照如下方式做出
 - 假设文法符号串 γ 使LR(0)自动机从开始状态0运行到某个状态 j



2.1 项和LR(0)自动机

- 移进-归约决定可以按照如下方式做出
 - 假设文法符号串 γ 使LR(0)自动机从开始状态0运行到某个状态 j



情况2：

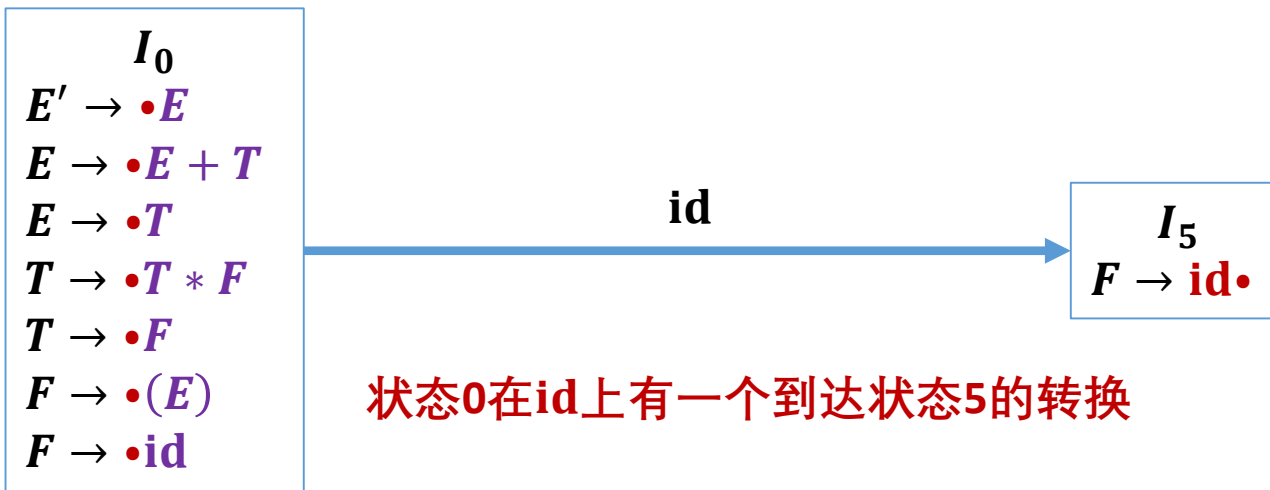
- 不满足情况1的条件时
- 状态 j 的项将告诉我们使用哪个产生式进行归约

归约！

2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

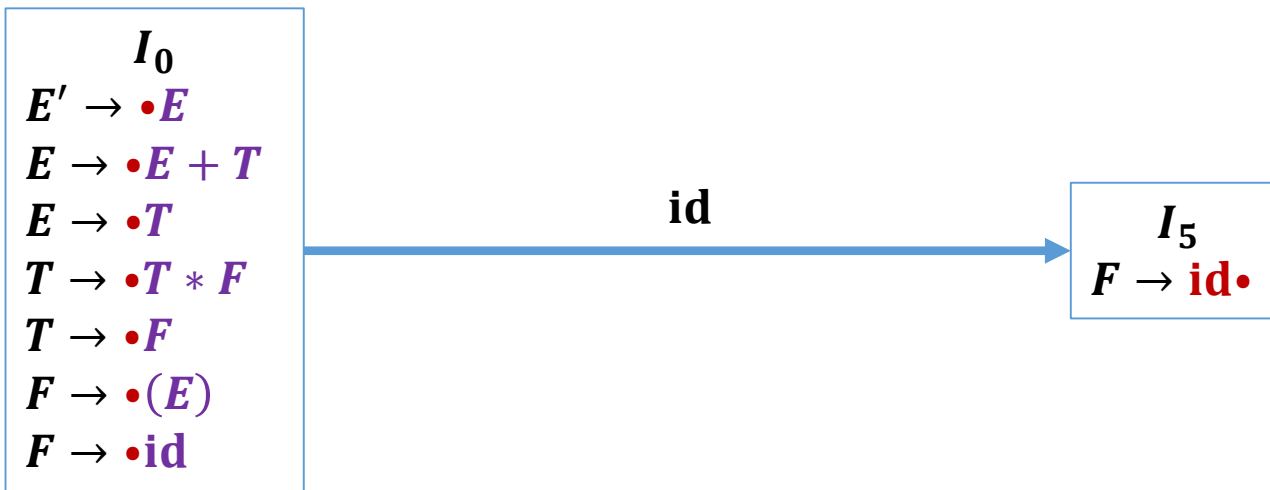
行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	



2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

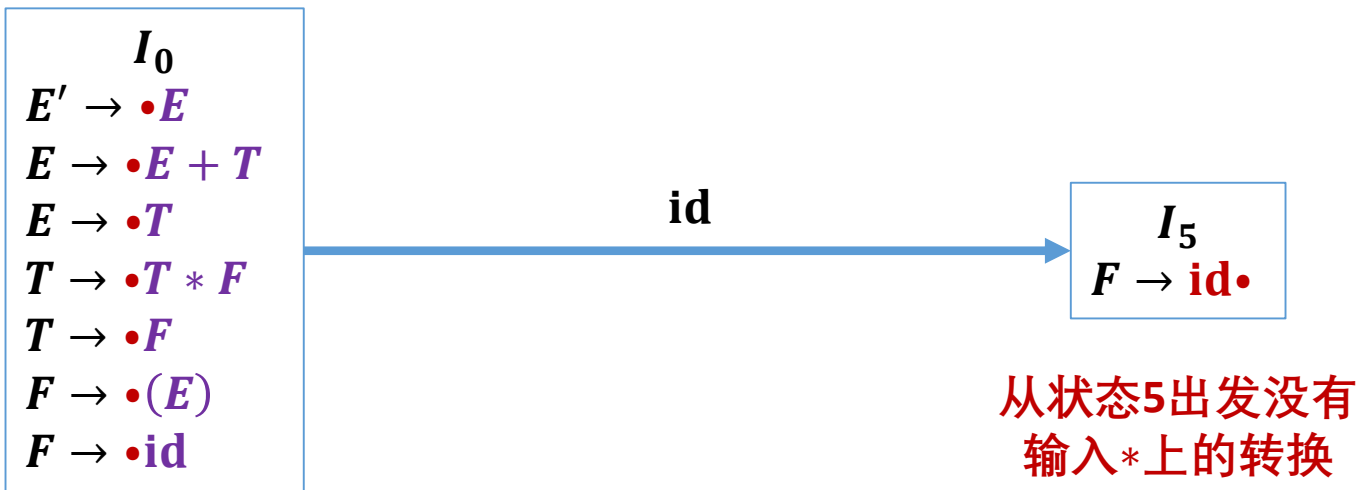
行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	



2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow \text{id}$ 归约



2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串
id * id

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 <i>F</i> → id 归约

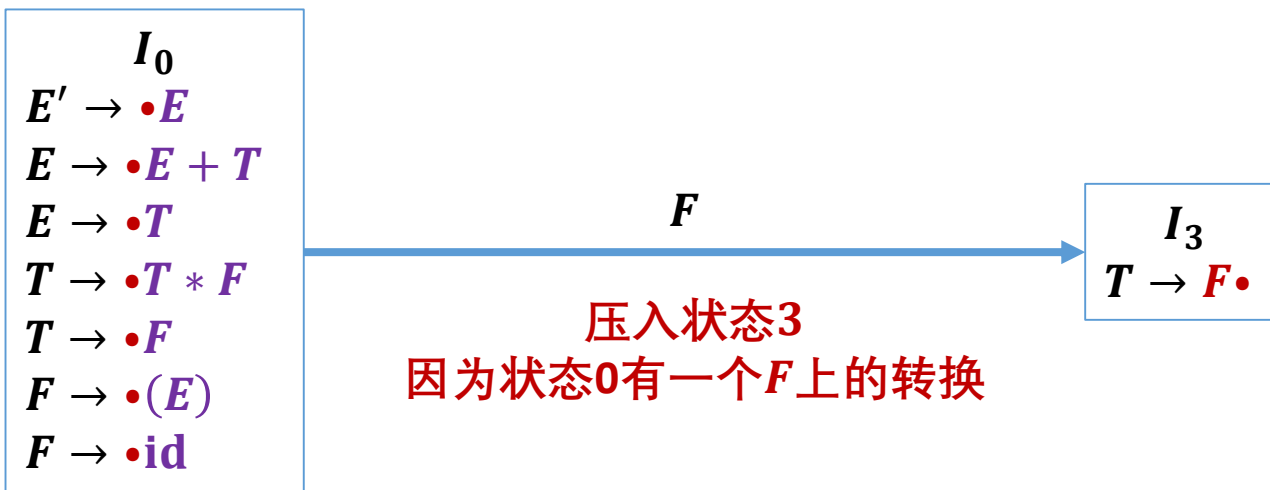
弹出**id**
对应的状态**5**

弹出产生式体**id**
压入产生式头***F***

2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

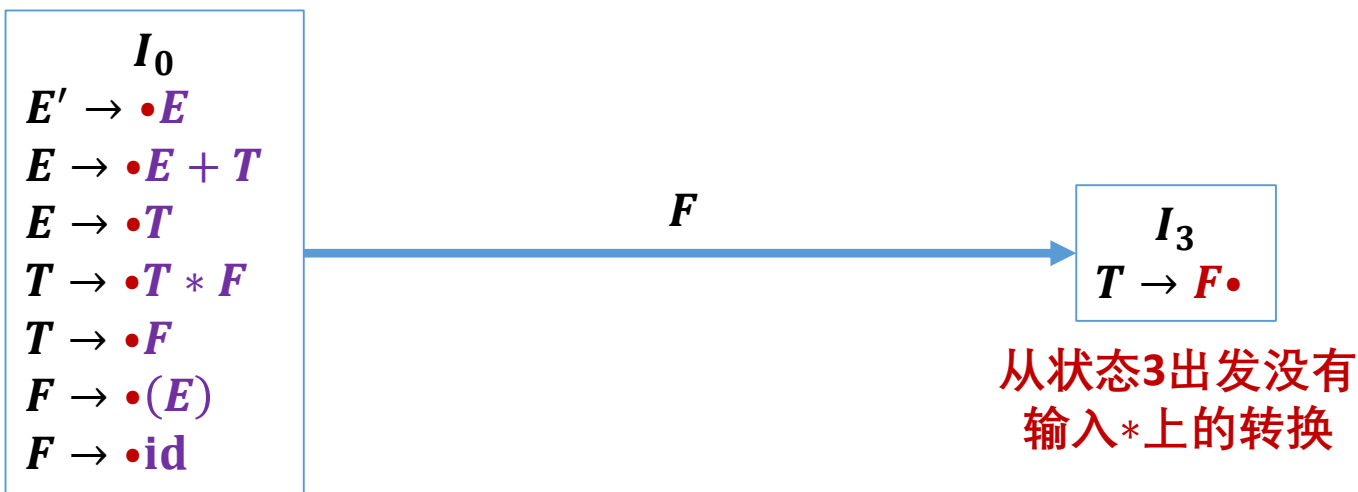
行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow \text{id}$ 归约
(3)	0	\$ F	* id \$	



2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 <i>F</i> → id 归约
(3)	0 3	\$ <i>F</i>	* id \$	按照 <i>T</i> → <i>F</i> 归约



2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 <i>F</i> → id 归约
(3)	0 3	\$ <i>F</i>	* id \$	按照 <i>T</i> → <i>F</i> 归约

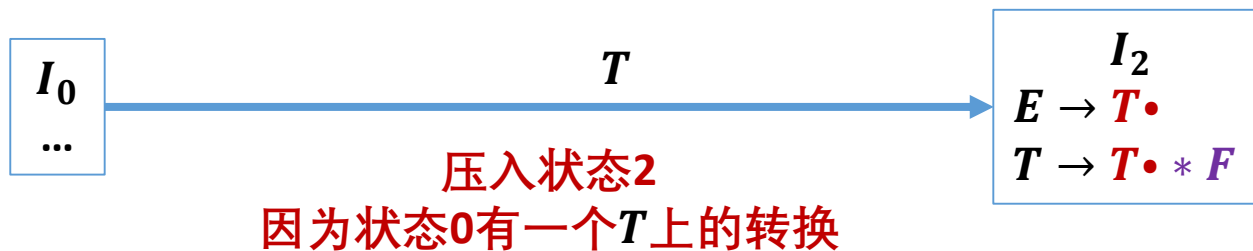
弹出***F***
对应的状态3

弹出产生式体***F***
压入产生式头***T***

2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

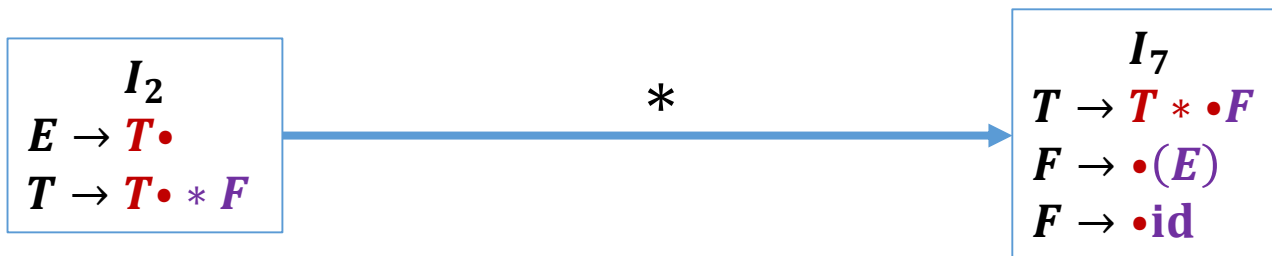
行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow \text{id}$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \rightarrow F$ 归约
(4)	0	\$ T	* id \$	



2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow \text{id}$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \rightarrow F$ 归约
(4)	0 2	\$ T	* id \$	

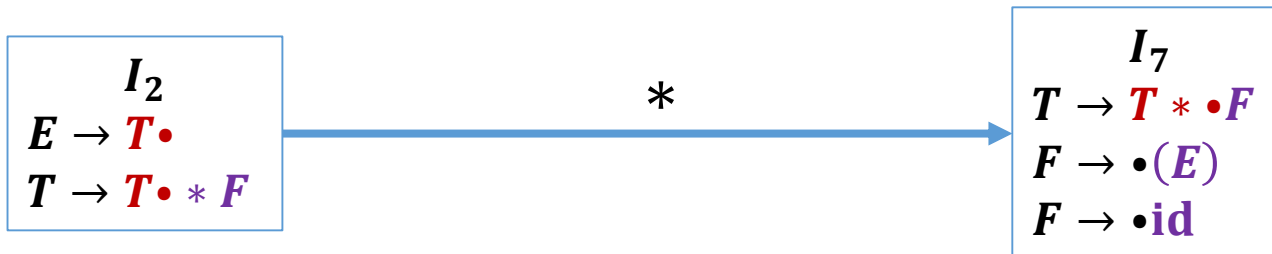


状态2在 $*$ 上有一个到达状态7的转换

2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow \text{id}$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \rightarrow F$ 归约
(4)	0 2	\$ T	* id \$	移进到7



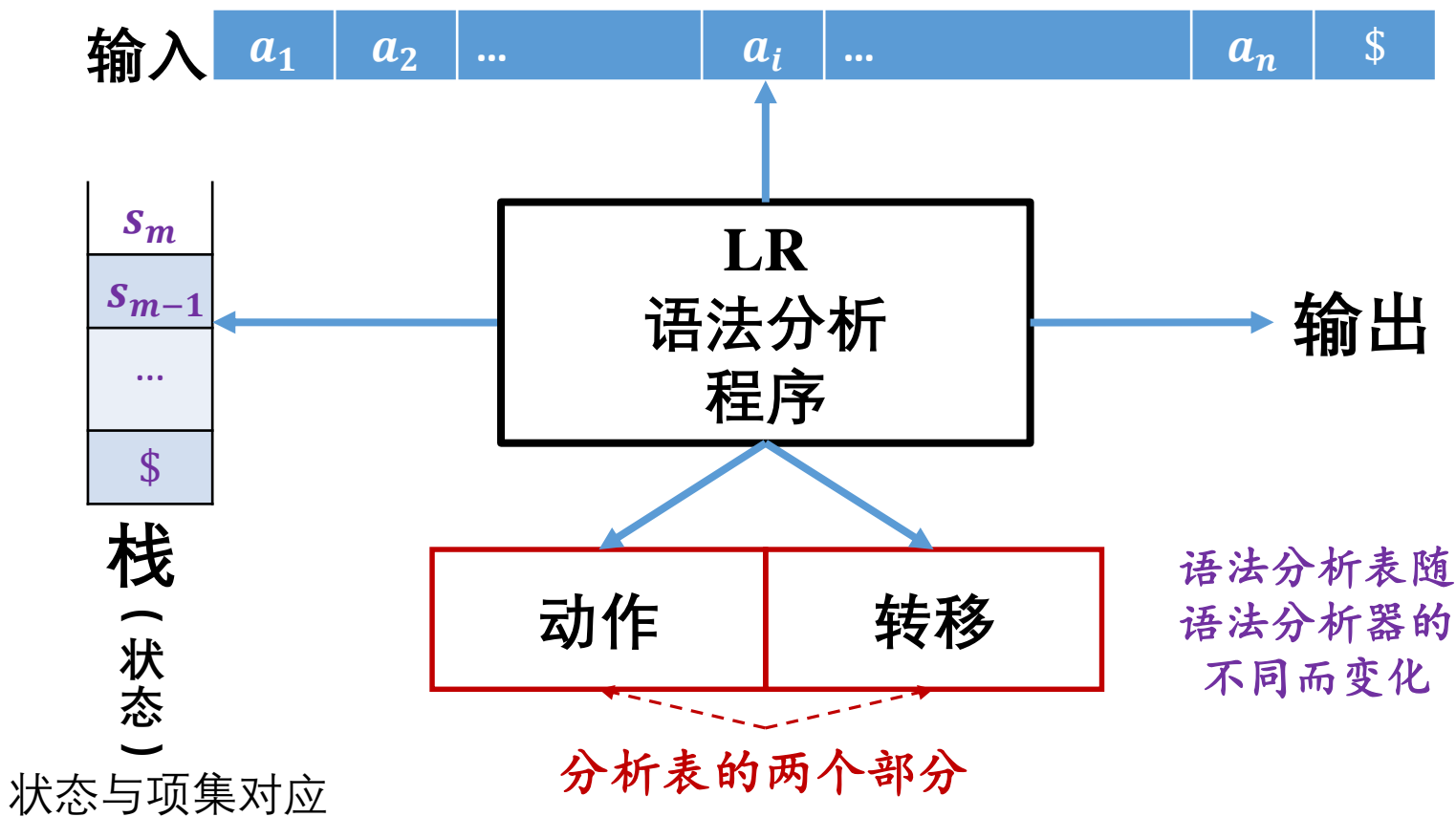
2.1 项和LR(0)自动机

- 例：使用表达式文法的LR(0)自动机分析输入串 **id * id**

行号	栈(状态)	符号	输入	动作
(1)	0	\$	id * id \$	移进到5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow \text{id}$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \rightarrow F$ 归约
(4)	0 2	\$ T	* id \$	移进到7
(5)	0 2 7	\$ T *	id \$...
...
	0 1	\$ E	\$	接受

2.2 LR分析算法

一个LR语法分析器的模型：



2.2 LR分析算法

- LR语法分析表的结构：

动作函数(ACTION) + 转换函数(GOTO)

ACTION有两个参数：

1. 状态 i
2. 终结符号 a (或输入结束标记)

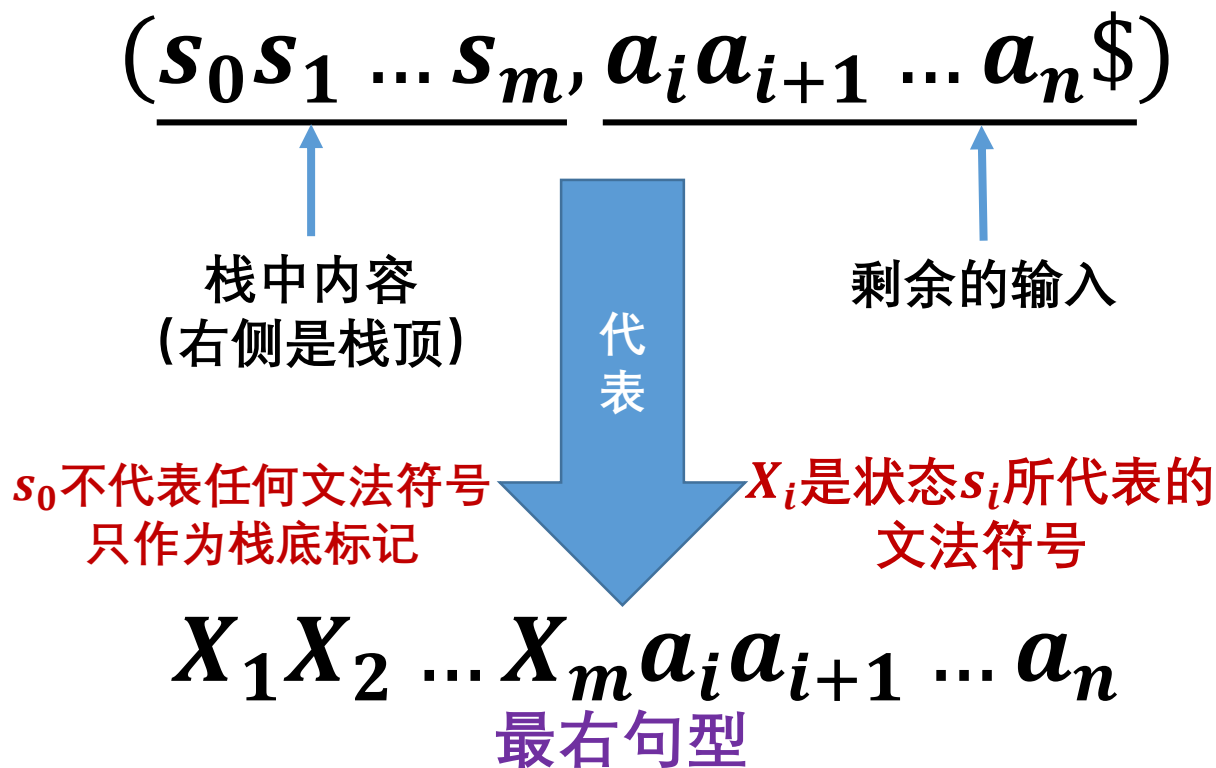
ACTION $[i, a]$ 的取值可能有：

1. 移进状态 j ，将输入符号 a 移进栈，使用状态 j 代表 a
2. 归约 $A \rightarrow \beta$ ，将栈顶 β 归约为产生式头 A
3. 接受，接受输入完成语法分析过程
4. 报错，发现错误并执行纠正动作

GOTO：如果**GOTO** $(I_i, A) = I_j$ ，则**GOTO**也把状态 i 和一个非终结符 A 映射到状态 j

2.2 LR分析算法

- LR语法分析器的格局(Configuration)



2.2 LR分析算法

- **LR语法分析器的行为**：读入当前输入符号 a_i 和栈顶的状态 s_m ，然后在分析动作表中查询条目 $ACTION[s_m, a_i]$

Step-1. 如果 $ACTION[s_m, a_i]$ =移进状态 s ，则语法分析器执行一次移进动作；它将下一个状态 s 移进栈中，进入格局：

$$\left(\underline{s_0 s_1 \dots s_m s}, \underline{a_{i+1} \dots a_n \$} \right)$$

当前的输入符号是 a_{i+1}

2.2 LR分析算法

Step-2. 如果 $ACTION[s_m, a_i]$ =归约 $A \rightarrow \beta$ ，则语法分析器执行一次归约动作，进入格局：

$$\left(\underline{s_0 s_1 \dots s_{m-r} \mathbf{s}}, \underline{a_i a_{i+1} \dots a_n \$} \right)$$

其中 $r = |\beta|$ ， $\mathbf{s} = GOTO[s_{m-r}, A]$

- 语法分析器首先将 r 个状态符号弹出栈，使状态 s_{m-r} 位于栈顶，然后将 \mathbf{s} 压入栈
- 在一个归约动作中，当前输入符号不变
- 出栈状态的文法符号序列 $X_{m-r+1} \dots X_m = \beta$

2.2 LR分析算法

Step-3. 如果 $ACTION[s_m, a_i]$ =接受, 则分析过程完成

Step-4. 如果 $ACTION[s_m, a_i]$ =报错, 则说明分析器发现了一个语法错误, 并调用一个错误恢复例程

- 所有LR语法分析器均按照上述方式(Step-1~4)执行
- 分析器间唯一区别在于ACTION表项和GOTO表项中包含的信息不同

2.2 LR分析算法

LR语法分析算法

输入：一个输入串 w 和文法 G 的一个LR语法分析表

输出：如果 w 在 $L(G)$ 中，则输出 w 的自下向上语法分析过程中的**归约步骤**；否则给出一个**错误提示**

方法：最初，语法分析器栈中的内容为**初始状态** s_0 ，输入缓冲区中的内容为 $w\$$ ，然后执行程序(右图)

```
令 $a$ 为 $w\$$ 的第一个符号；
while(1) {
    令 $s$ 是栈顶的状态；
    if ( $ACTION[s, a] = \text{移进}t$ ) {
        将 $t$ 压入栈中；
        令 $a$ 为下一个输入符号；
    }
    else if ( $ACTION[s, a] = \text{归约}A \rightarrow \beta$ ) {
        从栈中弹出 $|\beta|$ 个符号；
        令 $t$ 为当前的栈顶状态；
        将 $GOTO[t, A]$ 压入栈中；
        输出产生式 $A \rightarrow \beta$ ；
    }
    else if ( $ACTION[s, a] = \text{接受}$ ) break;
    else 调用错误恢复例程；
}
```

2.2 LR分析算法

- 例：表达式文法

1. $E \rightarrow E + T$

2. $E \rightarrow T$

3. $T \rightarrow T * F$

4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow \mathbf{id}$

- s : shift
- r : reduce
- acc : accept

状态	ACTION						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
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

2.2 LR分析算法

- 例：表达式文法

1. $E \rightarrow E + T$

2. $E \rightarrow T$

3. $T \rightarrow T * F$

4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow \mathbf{id}$

状态	ACTION					
	id	+	*	()	\$
0	s5			s4		

	栈	符号	输入	动作
(1)	0		id * id+id\$	移进

2.2 LR分析算法

- 例：表达式文法

1. $E \rightarrow E + T$

2. $E \rightarrow T$

3. $T \rightarrow T * F$

4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow \mathbf{id}$

状态	ACTION					
	id	+	*	()	\$
5		r6	r6		r6	r6

需从栈中弹出|id|个符号

	栈	符号	输入	动作
(1)	0		id * id+id\$	移进
(2)	0 5	id	* id+id\$	根据 $F \rightarrow \mathbf{id}$ 归约

2.2 LR分析算法

- 例：表达式文法

1. $E \rightarrow E + T$

2. $E \rightarrow T$

3. $T \rightarrow T * F$

4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow \mathbf{id}$

状态	GOTO		
	E	T	F
0	1	2	3

需将 $GOTO[0, F]$ 压入栈

	栈	符号	输入	动作
(1)	0		id * id+id\$	移进
(2)	0 5	id	* id+id\$	根据 $F \rightarrow \mathbf{id}$ 归约
(3)	0	F	* id+id\$	

2.2 LR分析算法

- 例：表达式文法

1. $E \rightarrow E + T$

2. $E \rightarrow T$

3. $T \rightarrow T * F$

4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow \mathbf{id}$

状态	ACTION					
	id	+	*	()	\$
3		r4	r4		r4	r4

	栈	符号	输入	动作
(1)	0		id * id+id\$	移进
(2)	0 5	id	* id+id\$	根据 $F \rightarrow \mathbf{id}$ 归约
(3)	0 3	F	* id+id\$	根据 $T \rightarrow F$ 归约

2.2 LR分析算法

- 例：表达式文法

$$1. E \rightarrow E + T$$

$$2. E \rightarrow T$$

$$3. T \rightarrow T * F$$

$$4. T \rightarrow F$$

$$5. F \rightarrow (E)$$

$$6. F \rightarrow \text{id}$$

	栈	符号	输入	动作
(1)	0		id * id+id\$	移进
(2)	0 5	id	* id+id\$	根据 $F \rightarrow \text{id}$ 归约
(3)	0 3	F	* id+id\$	根据 $T \rightarrow F$ 归约
...
	0 1	E	\$	接受

状态	ACTION					
	id	+	*	()	\$
1		s6				acc