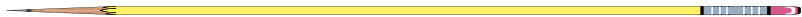




# 第4章 语法分析



知识点：预测分析方法、LL(1)分析程序  
移进-归约分析方法、LR分析程序  
SLR(1)、LR(1)、LALR(1)分析表

# 教学目标与要求

- 掌握自顶向下分析方法对文法的要求；
- 理解递归调用预测分析程序的构造方法；
- 掌握LL(1)分析程序的模型及工作过程，掌握LL(1)分析表的构造方法；
- 理解“移进-归约”分析方法的工作过程；
- 掌握LR(1)分析程序的模型及工作过程；掌握SLR(1)、LR(1)、LALR(1)分析表的构造方法。
- 理解并分析语法分析的需求，能够基于自动机设计语法分析程序，并对符号串进行语法分析。

# 本章内容

4.1 语法分析简介

4.2 自顶向下分析方法

4.3 自底向上分析方法

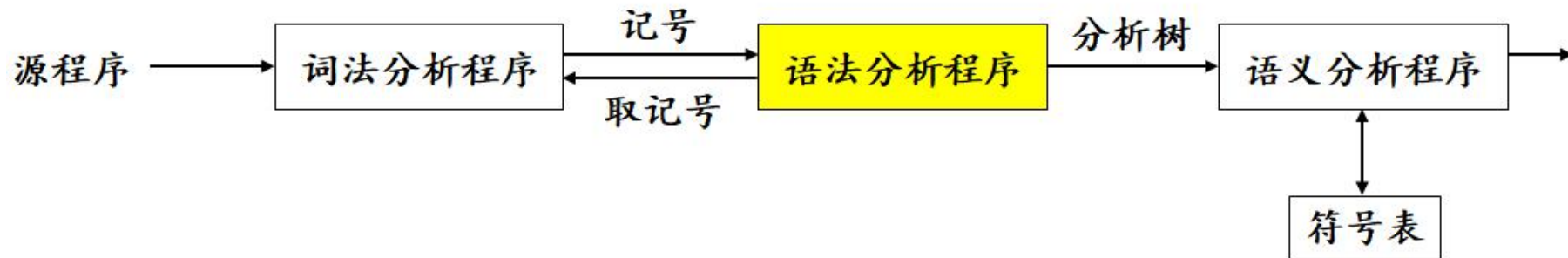
4.4 LR分析方法

4.5 软件工具YACC (\*)

小结

# 4.1 语法分析简介

- 编译程序的核心工作
- 由语法分析程序完成
- 工作依据：源语言的语法规则
- 语法分析程序的任务
  - 从源程序记号序列中识别出各类语法成分
  - 进行语法检查
- 语法分析程序的地位：



# 语法分析简介

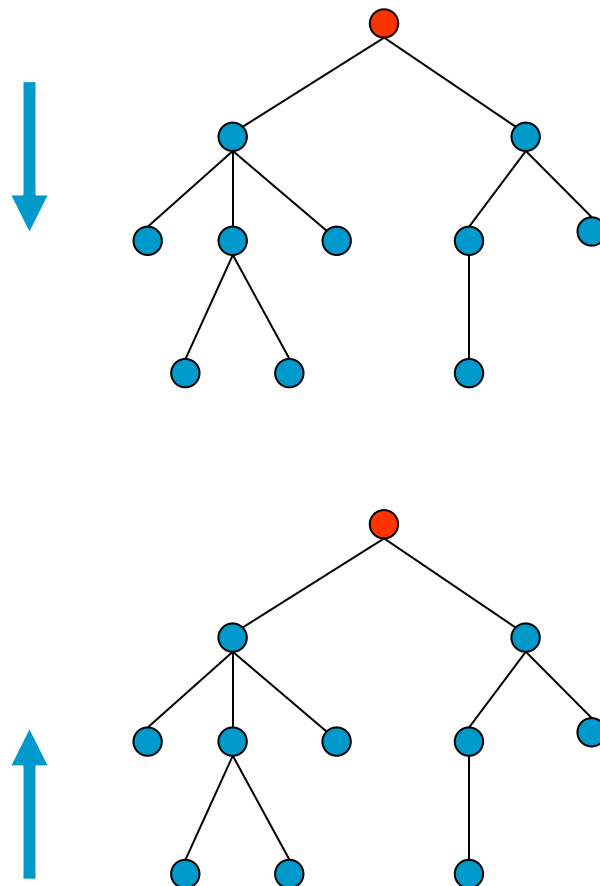
## ■ 语法分析程序

- 输入：记号流/记号序列
- 工作依据：语法规则
- 功能：将记号组合成语法成分、进行语法检查
- 输出：分析树
- 错误处理

## ■ 常用的分析方法

- 自顶向下的方法：  
从树根到叶子来建立分析树
- 自底向上的方法：  
从树叶到树根来建立分析树

## ■ 对输入符号串的扫描顺序：自左向右



# 语法错误的处理

## ■ 错误处理目标

- 清楚而准确地报告发现的错误，如错误的位置和性质。
- 迅速地从错误中恢复过来。
- 不应该明显地影响编译程序对正确程序的处理效率。

## ■ 错误恢复策略

- 紧急恢复
- 短语级恢复
- 出错产生式
- 全局纠正

- 简单，适用于大多数分析程序。
- 做法：一旦发现错误，分析程序每次抛弃一个输入记号，直到扫描到的记号属于某个指定的同步记号集合为止。
- 同步记号通常是定界符，如语句结束符分号、语句起始符、块结束标识END等。



## 4.2 自顶向下分析方法

4.2.1 递归下降分析

4.2.2 递归调用预测分析

4.2.3 非递归预测分析

## 4.2.1 递归下降分析

- 从文法的开始符号出发，进行推导，试图推出要分析的输入串的过程。
- 对给定的输入符号串，从对应于文法开始符号的根结点出发，自顶向下地为输入串建立一棵分析树。
- 试探过程，反复使用不同产生式谋求匹配输入符号串的过程。
- 例：试分析输入符号串  $\omega=abbcde$  是否为如下文法的一个句子。

$S \rightarrow aAcBe$

$A \rightarrow b \mid Ab$

$B \rightarrow d$

(文法4.1)



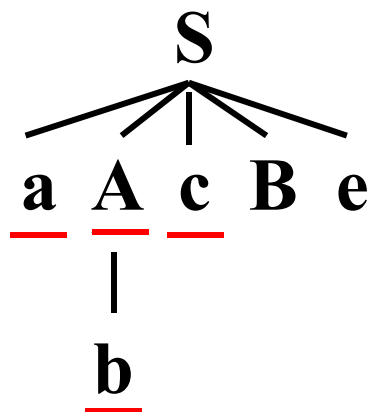
# 示例：abbcbde 的递归下降分析

$S \rightarrow aAcBe$

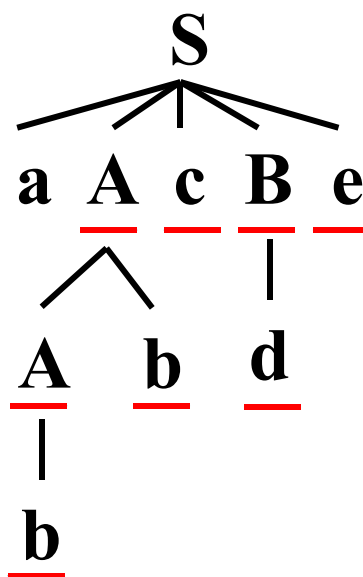
$A \rightarrow b \mid Ab$

$B \rightarrow d$

a b b c d e \$  
          ↑



a b b c d e \$  
                          ↑



推导?

$\Rightarrow$

$S \Rightarrow aAcBe \Rightarrow aAbcBe \Rightarrow abbcBe \Rightarrow abbcde$

试图为输入符号串建立一个最左推导序列的过程。

# 递归下降分析

## ■ 为什么采用最左推导？

- 因为对输入串的扫描是自左至右进行的，只有使用最左推导，才能保证按扫描的顺序匹配输入串。

## ■ 递归下降分析方法的实现

- 文法的每一个非终结符号对应一个递归过程，即可实现这种带回溯的递归下降分析方法。
- 每个过程作为一个布尔过程，一旦发现它的某个产生式与输入串匹配，则用该产生式展开分析树，并返回true，否则分析树不变，返回false。

## ■ 实践中存在的困难和缺点

- 左递归的文法，可能导致分析过程陷入死循环。
- 回溯
- 工作的重复
- 效率低、代价高：穷尽一切可能的试探法。

## 4.2.2 递归调用预测分析

■ 一种确定的、不带回溯的递归下降分析方法

一、如何克服回溯？

二、对文法的要求

三、预测分析程序的构造

# 一、如何克服回溯？

- 能够根据所面临的输入符号准确地指派一个候选式去执行任务。
- 该选择的工作结果是确信无疑的。
- 例：

$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_I \mid \dots \mid \alpha_n$

当前输入符号：a

指派 $\alpha_i$ 去匹配输入符号串

## 二、对文法的要求

左递归?

### 1. 不含左递归

$$A \Rightarrow \dots \Rightarrow A\alpha$$

### 2. $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset \quad (i \neq j)$

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

$$\text{FIRST}(\alpha_i) = \{ a \mid \alpha_i \xRightarrow{*} a\beta, a \in V_T, \alpha_i, \beta \in (V_T \cup V_N)^* \}$$

如果  $\alpha_i \xRightarrow{*} \varepsilon$ , 则规定  $\varepsilon \in \text{FIRST}(\alpha_i)$ 。

非终结符号A的所有候选式的开头终结符号集两两互不相交

# 示例:

- 有如下产生PASCAL类型子集的文法:

$type \rightarrow simple \mid \uparrow id \mid array[simple] \text{ of } type$

$simple \rightarrow integer \mid char \mid num \text{ dotdot } num$

分析输入串:  $array \ [num \ dotdot \ num] \ \text{of} \ char$

- $type$ 的三个候选式, 有:

$FIRST(simple) = \{ integer, char, num \}$

$FIRST(\uparrow id) = \{ \uparrow \}$

$FIRST(array[simple] \text{ of } type) = \{ array \}$

- $simple$ 的三个候选式, 有:

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

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

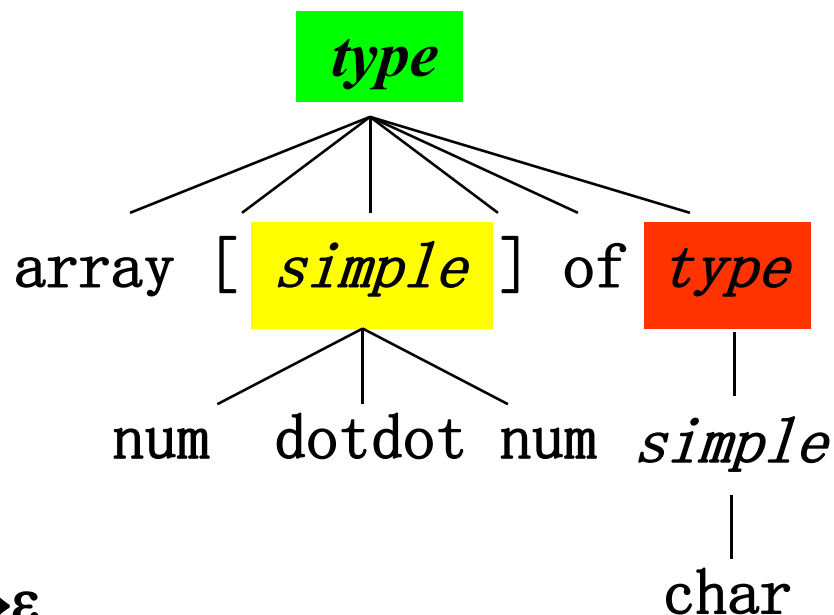
$FIRST(num \ dotdot \ num) = \{ num \}$

# 示例

输入: array [num dotdot num] of char

↑            ↑                            ↑

树:



产生式:  $A \rightarrow \epsilon$

缺席匹配

当递归下降分析程序没有适当候选式时, 可以用一个  $\epsilon$ -候选式, 表示其它候选式可以缺席。

# 三、预测分析程序的构造

- 预测分析程序的转换图
- 转换图的工作过程
- 转换图的化简
- 预测分析程序的实现



# 预测分析程序的转换图

- 为预测分析程序建立转换图作为其实现蓝图
- 每一个非终结符号有一张图
- 边的标记可以是终结符号，也可以是非终结符号。
- 在一个非终结符号A上的转移意味着对相应A的过程的调用。
- 在一个终结符号a上的转移，意味着下一个输入符号若为a，则应做此转移。

# 从文法构造转换图

## ■ 改写文法

- 重写文法

- 消除左递归

- 提取左公因子



改写文法?

$\Rightarrow$

## ■ 对每一个非终结符号A，做如下工作：

- 创建一个初始状态和一个终结状态。

- 对每一个产生式  $A \rightarrow X_1 X_2 \dots X_n$  创建一条从初态到终态的路径，有向边的标记依次为  $X_1, X_2, \dots, X_n$ 。

# 示例

- 为如下文法构造预测分析程序转换图

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \text{id} \quad (\text{文法4.3})$$

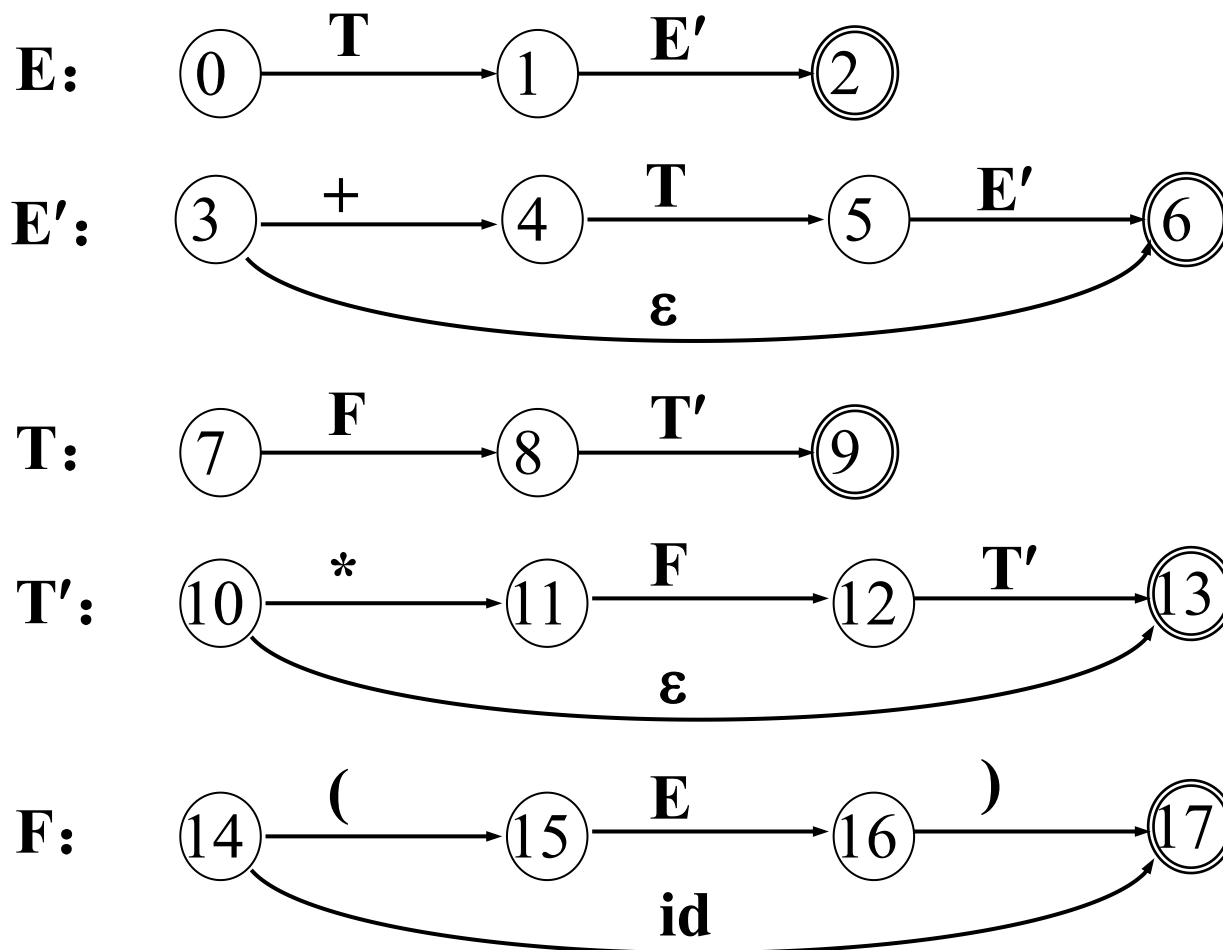
- 消除文法中存在的左递归，得到

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E) \mid \text{id} \quad (\text{文法4.4})$$

# 示例

- 为每个非终结符号构造转换图：

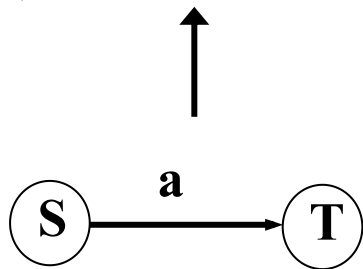
$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \varepsilon$   
 $F \rightarrow (E) \mid id$



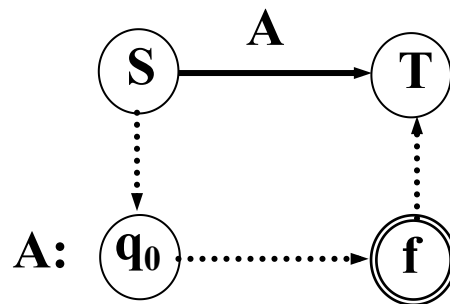
# 转换图的工作过程

- 从文法开始符号所对应的转换图的开始状态开始分析
- 经过若干动作之后，处于状态S  
扫描指针指向符号a，即当前输入符号

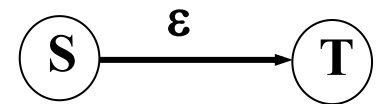
输入串: ... a ...



输入串: ... a ..... b ...

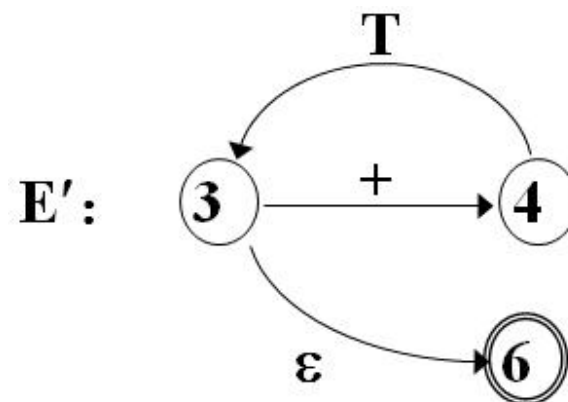
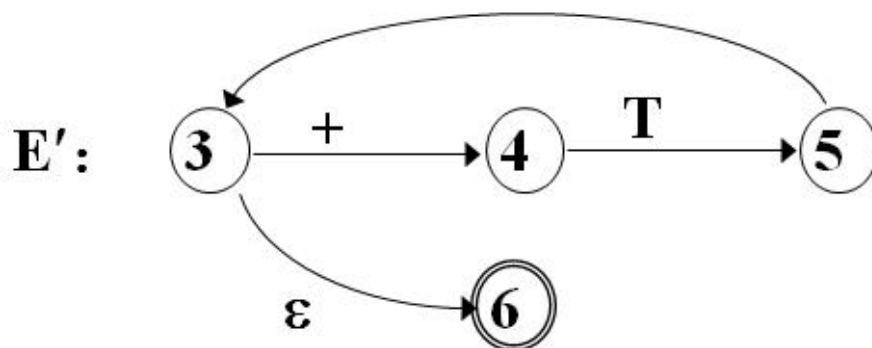
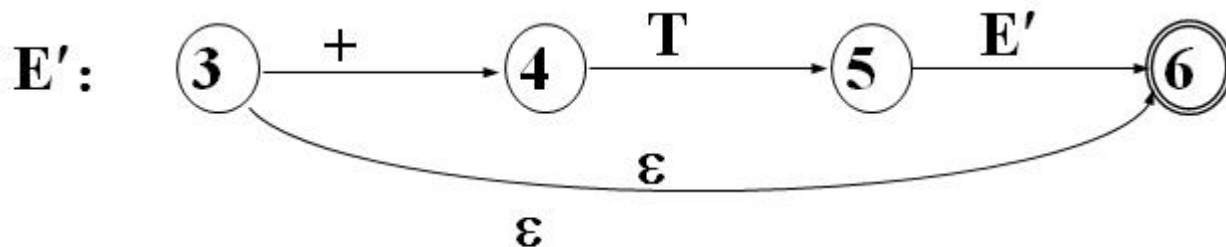


输入串: ... a ...

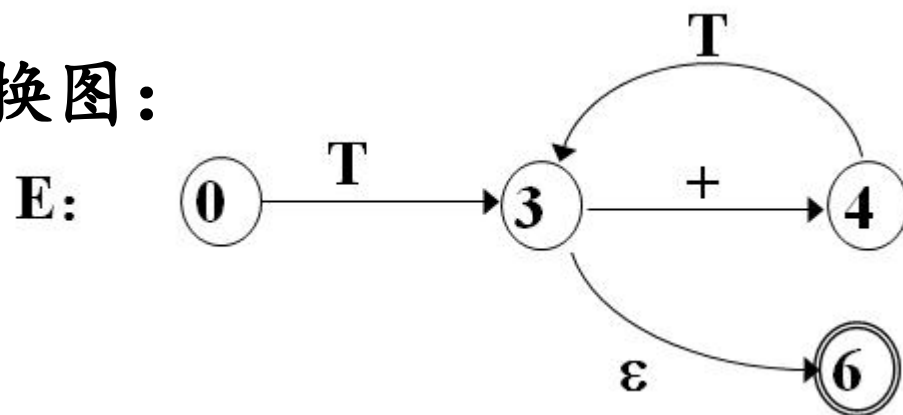


# 转换图的化简

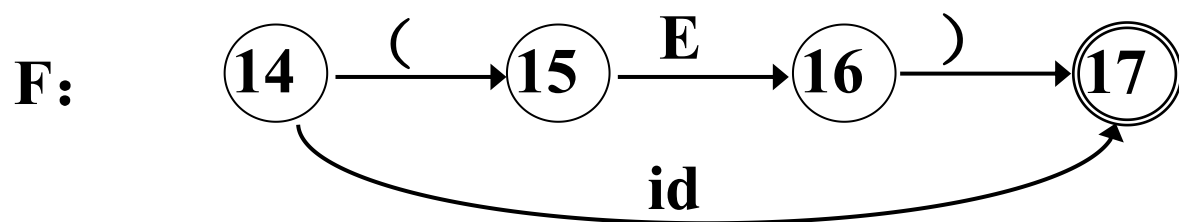
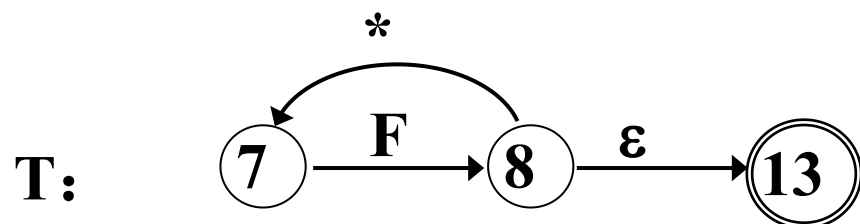
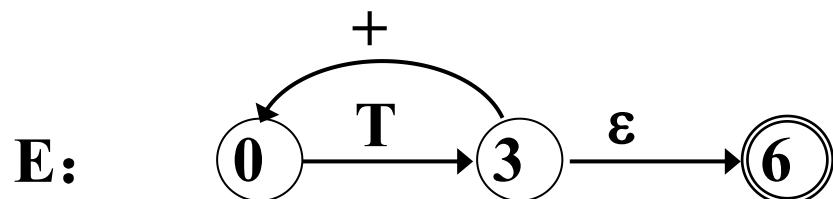
## ■ 用代入的方法进行化简



## ■ 把E'的转换图代入E的转换图:



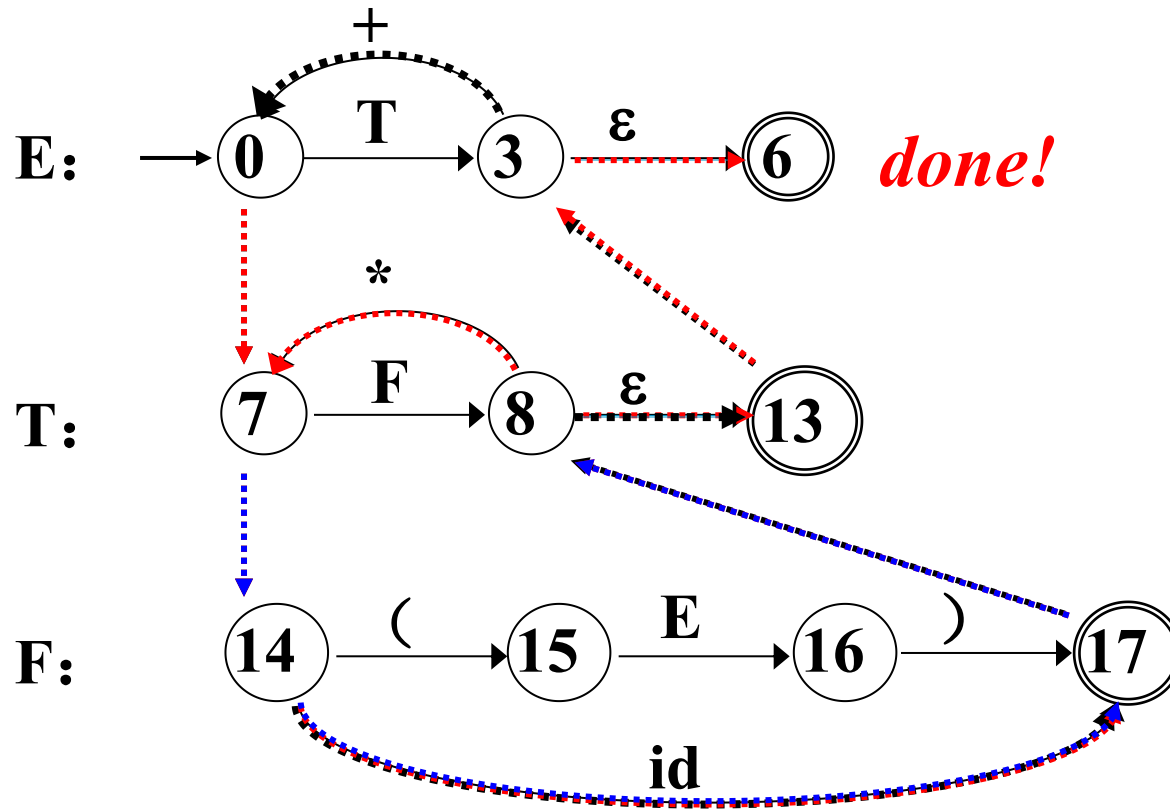
# 文法4.3的预测分析程序的状态转换图



练习:

利用状态转换图识别符号串: **id+id\*id**

# id+id\*id的识别过程



id + id \* id \$



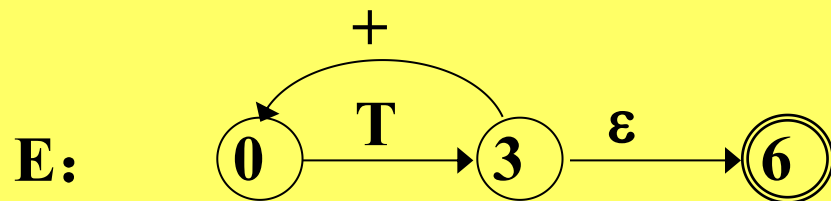


# 预测分析程序的实现

- 要求用来描述预测分析程序的语言允许递归调用

- E的过程:

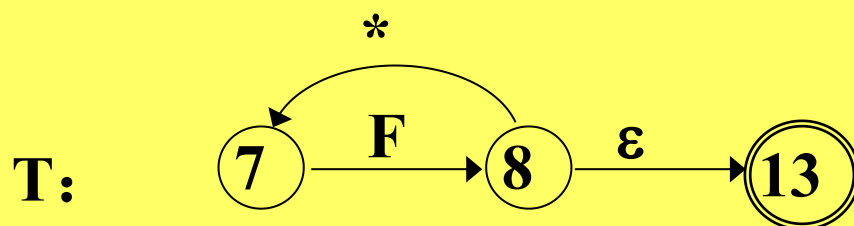
```
void procE(void)
{
    procT();
    if (char=='+') {
        forward pointer;
        procE();
    }
}
```



# 预测分析程序的实现

## ■ T的过程:

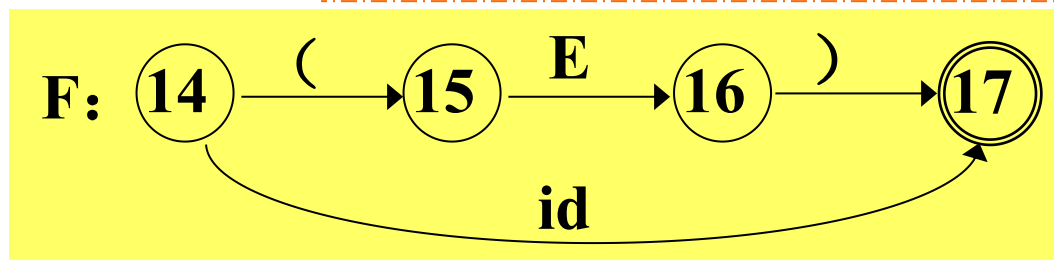
```
void procT(void)
{
    procF();
    if (char=='*') {
        forward pointer;
        procT();
    }
}
```



# 预测分析程序的实现

## ■ F的过程:

```
void procF(void) {  
    if (char=='(') {  
        forward pointer;  
        procE();  
        if (char==')') {  
            forward pointer;  
        };  
        else error();  
    };  
    else if (char=='id') {  
        forward pointer;  
        else error();  
    }  
}
```



## 练习4.1

有如下文法：

$$bexpr \rightarrow bexpr \text{ or } bterm \mid bterm$$
$$bterm \rightarrow bterm \text{ and } bfactor \mid bfactor$$
$$bfactor \rightarrow \text{not } bfactor \mid (bexpr) \mid \text{true} \mid \text{false}$$

请构造一个可以用来分析该文法所产生的句子的递归调用分析程序。

# Step 1: 消除左递归

$bexpr \rightarrow bterm \ E'$

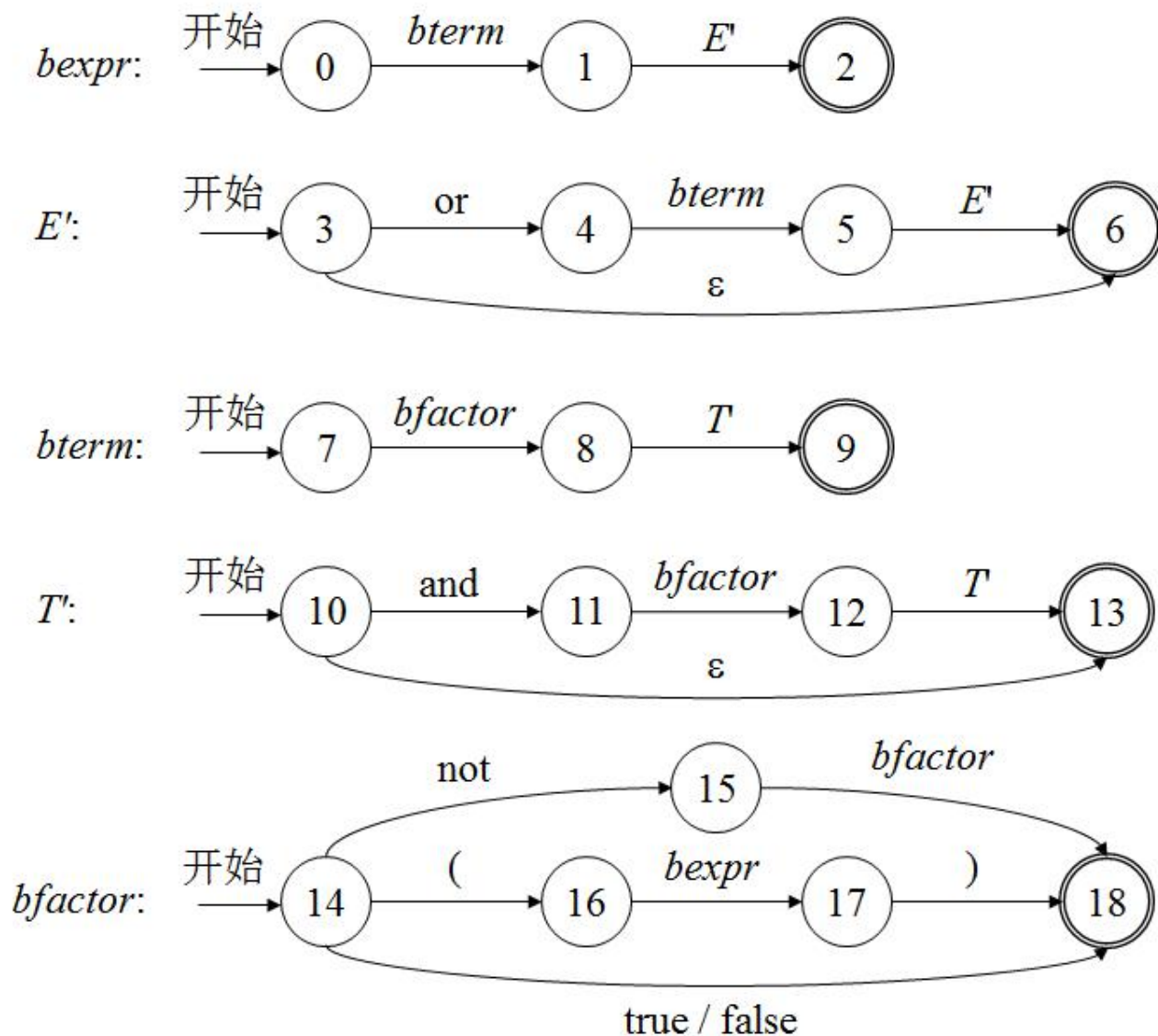
$E' \rightarrow \text{or } bterm \ E' \mid \varepsilon$

$bterm \rightarrow bfactor \ T'$

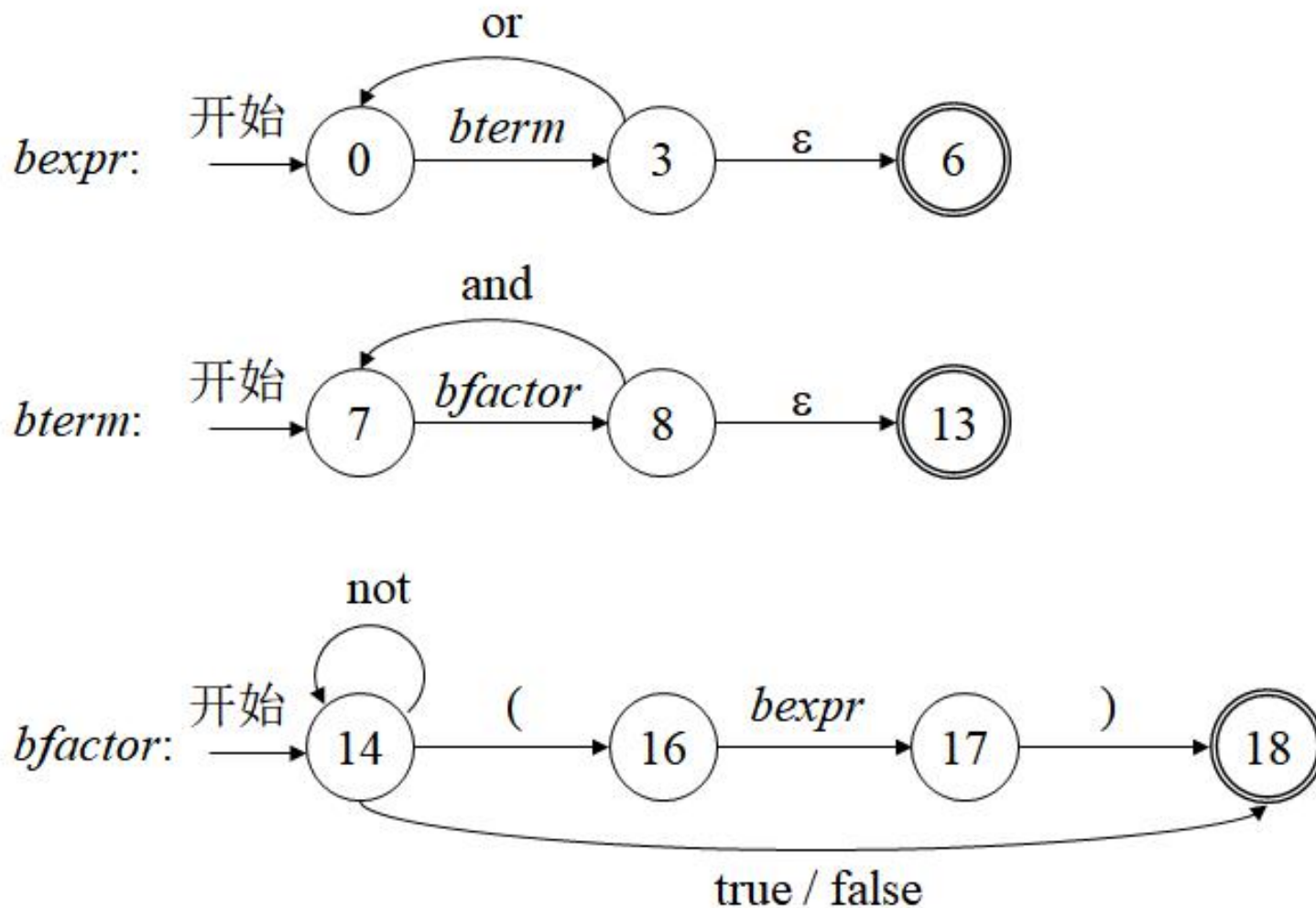
$T' \rightarrow \text{and } bfactor \ T' \mid \varepsilon$

$bfactor \rightarrow \text{not } bfactor \mid (bexpr) \mid \text{true} \mid \text{false}$

## Step 2: 文法 $G'$ 的预测分析程序状态转换图



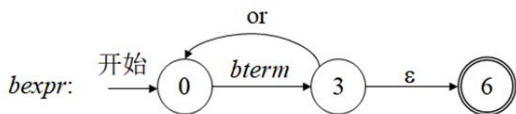
## Step 3: 化简后的预测分析程序状态转换图



# Step 4: 根据状态转换图进行程序设计

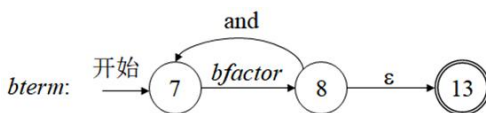
## ■ *bexpr*的函数

```
void proc_expr(void) {  
    proc_term();  
    if (char=='or') {  
        forward pointer;  
        proc_expr();  
    }  
}
```



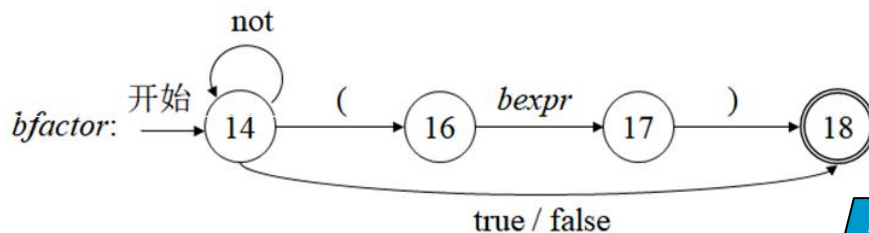
## ■ *bterm*的过程

```
void proc_term(void) {  
    proc_factor();  
    if (char=='and') {  
        forward pointer;  
        proc_term();  
    }  
}
```



## ■ *bfactor*的过程

```
void proc_factor(void) {  
    if (char=='not'){  
        forward pointer;  
        proc_factor();  
    }  
    else if (char=='(') {  
        forward pointer;  
        proc_expr();  
        if (char==')')  
            forward pointer;  
        else error();  
    }  
    else if (char=='true')||(char=='false')  
        forward pointer;  
    else error();  
}
```

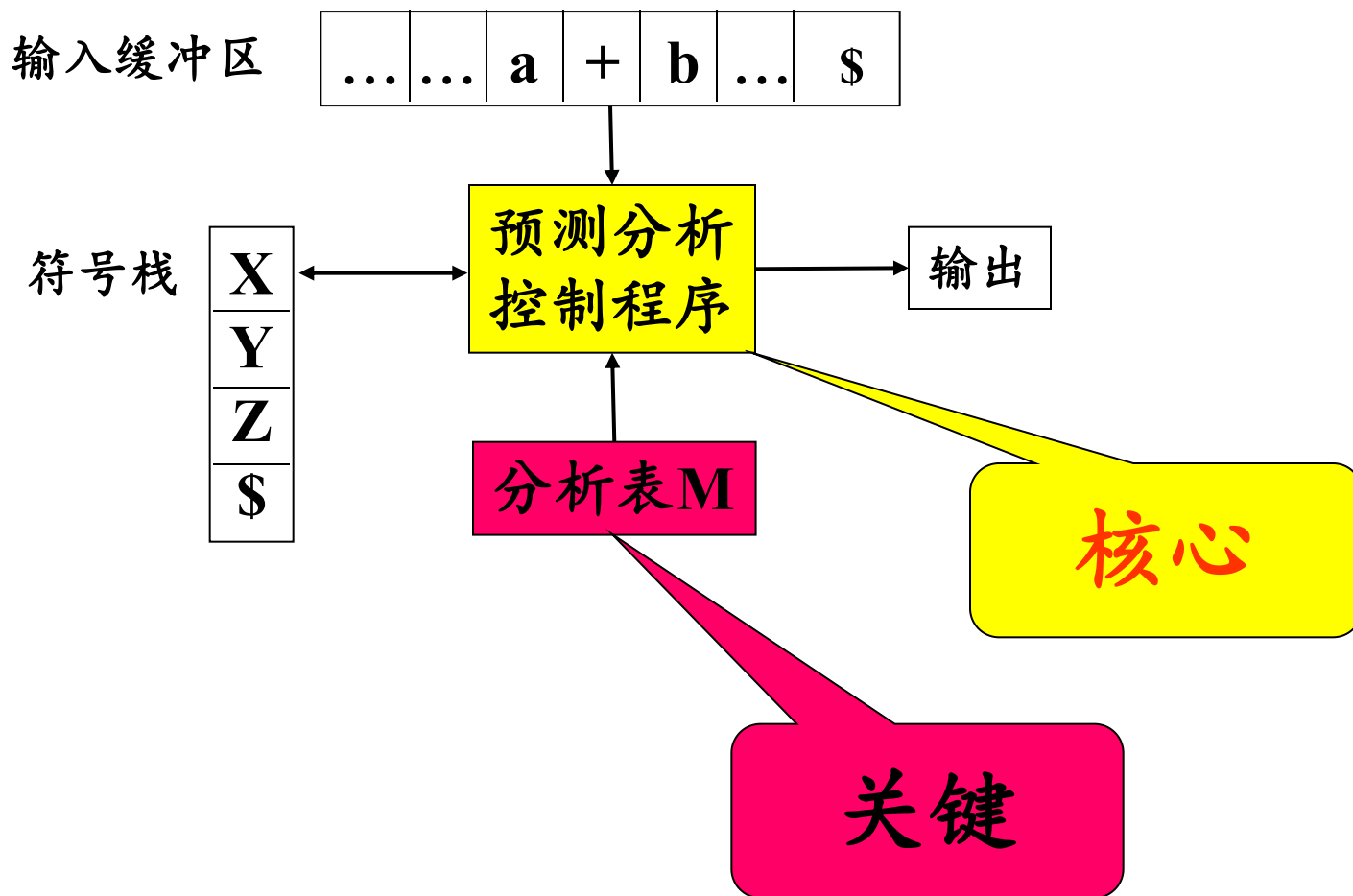




## 4.2.3 非递归预测分析

- 使用一张分析表和一个栈联合控制，实现对输入符号串的自顶向下分析。
- 预测分析程序的模型及工作过程
- 预测分析表的构造
- LL(1)文法
- 预测分析方法中的错误处理示例

# 预测分析程序的模型及工作过程



# 每部分的作用

## ■ 输入缓冲区：

存放被分析的输入符号串，串后随右尾标志符\$。

... .. \$

## ■ 符号栈：

存放一系列文法符号，\$存于栈底。分析开始时，先将\$入栈，以标识栈底，然后再将文法的开始符号入栈。

\$S

## ■ 分析表：

二维数组 $M[A, a]$ ， $A \in V_N$ ， $a \in V_T \cup \{\$ \}$ 。

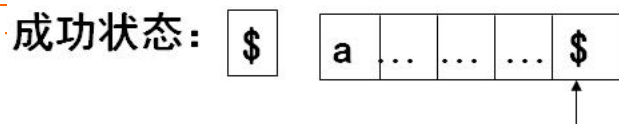
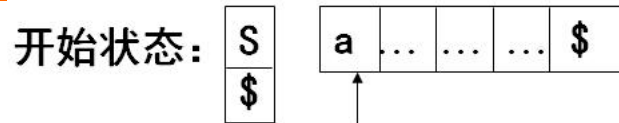
根据给定的 $A$ 和 $a$ ，在分析表 $M$ 中找到将被调用的产生式。

## ■ 输出流：

分析过程中不断产生的产生式序列。

... a ...  
A  $\left[ \begin{array}{c} A \rightarrow a \dots \end{array} \right]$

# 预测分析控制程序



■ 根据栈顶符号 $X$ 和当前输入符号 $a$ , 分析动作有4种可能:

- (1)  $X=a=\$$ , 宣告分析成功, 停止分析;
- (2)  $X=a\neq \$$ , 从栈顶弹出 $X$ , 输入指针前移一个位置;
- (3)  $X\in V_T$ , 但 $X\neq a$ , 报告发现错误, 调用错误处理程序, 报告错误及进行错误恢复;
- (4) 若 $X\in V_N$ , 访问分析表 $M[X, a]$

➤  $M[X, a]=X\rightarrow Y_1Y_2\ldots Y_n$

先将 $X$ 从栈顶弹出, 然后把产生式的右部符号串按反序推入栈中 (即按 $Y_n, \ldots, Y_{n-1}, Y_2, Y_1$ 的顺序);

➤  $M[X, a]=X\rightarrow\epsilon$  从栈顶弹出 $X$ ;

➤  $M[X, a]=\text{error}$  调用出错处理程序

# 算法4.1 非递归预测分析方法

输入：输入符号串 $\omega$ ，文法 $G$ 的一张预测分析表 $M$ 。

输出：若 $\omega$ 在 $L(G)$ 中，则输出 $\omega$ 的最左推导，否则报告错误。

方法：分析开始时， $\$$ 在栈底，文法开始符号 $S$ 在栈顶， $\omega\$$ 在输入缓冲区中置 $ip$ 指向 $\omega\$$ 的第一个符号；

```
do {  
    令 $X$ 是栈顶符号， $a$ 是 $ip$ 所指向的符号；  
    if ( $X$ 是终结符号或 $\$$ ) {  
        if ( $X==a$ ) {  
            从栈顶弹出 $X$ ； $ip$ 前移一个位置；  
        }  
        else error();  
    else /*  $X$ 是非终结符号 */  
        if ( $M[X, a]=X \rightarrow Y_1Y_2...Y_k$ ) {  
            从栈顶弹出 $X$ ；  
            把 $Y_k$ 、 $Y_{k-1}$ 、...、 $Y_2$ 、 $Y_1$ 压入栈， $Y_1$ 在栈顶；  
            输出产生式 $X \rightarrow Y_1Y_2...Y_k$ ；  
        }  
        else error();  
    } while( $X!=\$$ ) /* 栈不空，继续 */
```

示例：文法4.4的预测分析表M，试分析输入串  $id+id*id$

	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	$F \rightarrow id$			$F \rightarrow (E)$		

输出：

分析栈

\$	E'	T'	id	*	
----	----	----	----	---	--

输入串

id	+	id	*	id	\$
----	---	----	---	----	----

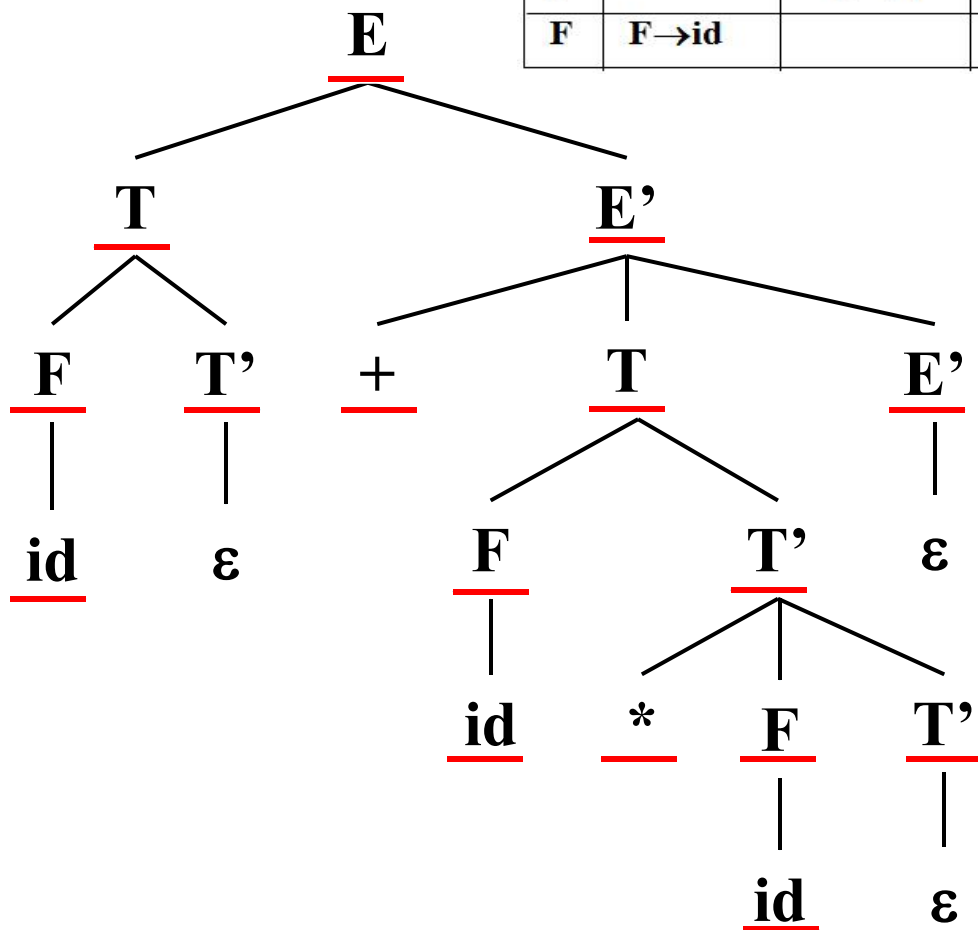


*done!*

$E \rightarrow TE'$   
 $T \rightarrow FT'$   
 $F \rightarrow id$   
 $T' \rightarrow \epsilon$   
 $E' \rightarrow +TE'$   
 $T \rightarrow FT'$   
 $F \rightarrow id$   
 $T' \rightarrow *FT'$   
 $F \rightarrow id$   
 $T' \rightarrow \epsilon$   
 $E' \rightarrow \epsilon$

# id + id \* id 分析树的构建过程:

	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	$F \rightarrow id$			$F \rightarrow (E)$		



id + id \* id \$



\$ E' T' id \*

*done !*

# 输入串 $id+id*id$ 的分析过程:

栈	输入	输出	左句型
\$ E	id + id * id \$		E
\$ E' T	id + id * id \$	$E \rightarrow TE'$	T E'
\$ E' T' F	id + id * id \$	$T \rightarrow FT'$	F T' E'
\$ E' T' id	id + id * id \$	$F \rightarrow id$	id T' E'
\$ E' T'	+ id * id \$		id T' E'
\$ E'	+ id * id \$	$T' \rightarrow \epsilon$	id E'
\$ E' T +	+ id * id \$	$E' \rightarrow +TE'$	id + T E'
\$ E' T	id * id \$		id + T E'
\$ E' T' F	id * id \$	$T \rightarrow FT'$	id + F T' E'
\$ E' T' id	id * id \$	$F \rightarrow id$	id + id T' E'
\$ E' T'	* id \$		id + id T' E'
\$ E' T' F *	* id \$	$T' \rightarrow *FT'$	id + id * F T' E'
\$ E' T' F	id \$		id + id * F T' E'
\$ E' T' id	id \$	$F \rightarrow id$	id + id * id T' E'
\$ E' T'	\$		id + id * id T' E'
\$ E'	\$	$T' \rightarrow \epsilon$	id + id * id E'
\$	\$	$E' \rightarrow \epsilon$	id + id * id



# 预测分析表的构造

- 改写文法
- FIRST集合及其构造
- FOLLOW集合及其构造
- 预测分析表的构造

# FIRST集合及其构造

## ■ FIRST集合

定义：对任何文法符号串 $\alpha \in (V_T \cup V_N)^*$ ,

$\text{FIRST}(\alpha)$ 是 $\alpha$ 可以推导出的开头终结符号集合

描述为： $\text{FIRST}(\alpha) = \{ a \mid \alpha \xRightarrow{*} a \cdots, a \in V_T \}$

若 $\alpha \xRightarrow{*} \varepsilon$ , 则 $\varepsilon \in \text{FIRST}(\alpha)$

# 构造每个文法符号 $X \in V_T \cup V_N$ 的 $\text{FIRST}(X)$

- 若  $X \in V_T$ , 则  $\text{FIRST}(X) = \{X\}$ ;
- 若  $X \in V_N$ , 且有产生式  $X \rightarrow a...$ , 其中  $a \in V_T$ , 则把  $a$  加入到  $\text{FIRST}(X)$  中;
- 若  $X \rightarrow \varepsilon$  也是产生式, 则  $\varepsilon$  也加入到  $\text{FIRST}(X)$  中。
- 若  $X \rightarrow Y...$  是产生式, 且  $Y \in V_N$ , 则把  $\text{FIRST}(Y)$  中的所有非  $\varepsilon$  元素加入到  $\text{FIRST}(X)$  中;

若  $X \rightarrow Y_1 Y_2 \dots Y_k$  是产生式, 如果对某个  $i$ ,  
 $\text{FIRST}(Y_1)$ 、 $\text{FIRST}(Y_2)$ 、...、 $\text{FIRST}(Y_{i-1})$  都含有  $\varepsilon$ ,  
即  $Y_1 Y_2 \dots Y_{i-1} \xRightarrow{*} \varepsilon$ ,  
则把  $\text{FIRST}(Y_i)$  中的所有非  $\varepsilon$  元素加入到  $\text{FIRST}(X)$  中;

若所有  $\text{FIRST}(Y_i)$  均含有  $\varepsilon$ , 其中  $i=1, 2, \dots, k$ ,  
则把  $\varepsilon$  加入到  $\text{FIRST}(X)$  中。

# FOLLOW集合及其构造

## ■ FOLLOW集合

定义：假定S是文法G的开始符号，对于G的任何非终结符号A，集合FOLLOW(A)是在所有句型中，紧跟A之后出现的终结符号或\$组成的集合。

描述为：  $\text{FOLLOW}(A) = \{ a \mid S \xRightarrow{*} \dots Aa \dots, a \in V_T \}$

特别地，若  $S \xRightarrow{*} \dots A$ ，则规定  $\$ \in \text{FOLLOW}(A)$

# 构造每个非终结符号A的集合FOLLOW(A)

- 对文法开始符号S，置\$于FOLLOW(S)中，\$为输入符号串的右尾标志。
- 若 $A \rightarrow \alpha B \beta$ 是产生式，则把FIRST( $\beta$ )中的所有非 $\epsilon$ 元素加入到FOLLOW(B)中。
- 若 $A \rightarrow \alpha B$ 是产生式，或 $A \rightarrow \alpha B \beta$ 是产生式并且 $\beta \xRightarrow{*} \epsilon$ ，则把FOLLOW(A)中的所有元素加入到FOLLOW(B)中。
- 重复此过程，直到所有集合不再变化为止。

# 示例

- 构造文法4.4中每个非终结符号的FIRST集合和FOLLOW集合

	FIRST	FOLLOW
E	(, id	\$, )
E'	+, $\epsilon$	\$, )
T	(, id	\$, +, )
T'	*, $\epsilon$	\$, +, )
F	(, id	\$, +, *, )

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid id$

## 算法4.2 预测分析表的构造方法

输入：文法G

输出：文法G的预测分析表M

方法：

```
for (文法G的每个产生式  $A \rightarrow \alpha$ ) {  
    for (每个终结符号  $a \in \text{FIRST}(\alpha)$ )  
        把  $A \rightarrow \alpha$  放入  $M[A, a]$  中;  
    if ( $\epsilon \in \text{FIRST}(\alpha)$ )  
        for (任何  $b \in \text{FOLLOW}(A)$ )  
            把  $A \rightarrow \alpha$  放入  $M[A, b]$  中;  
};  
for (所有无定义的  $M[A, a]$ ) 标上错误标志。
```

# 示例：为文法4.4构造预测分析表

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \varepsilon$   
 $F \rightarrow (E) \mid id$

	FIRST	FOLLOW
E	(, id	\$, )
E'	+, $\varepsilon$	\$, )
T	(, id	\$, +, )
T'	*, $\varepsilon$	\$, +, )
F	(, id	\$, +, *, )

	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	$F \rightarrow id$			$F \rightarrow (E)$		



# LL(1)文法

例：考虑如下映射程序设计语言中if语句的文法

$$S \rightarrow iEtSS' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$E \rightarrow b \quad (\text{文法4.5})$$

构造各非终结符号的FIRST和FOLLOW集合：

	S	S'	E
FIRST	i, a	e, $\varepsilon$	b
FOLLOW	\$, e	\$, e	t

应用算法4.2，构造该文法的分析表：

	a	b	e	i	t	\$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow eS$ $S' \rightarrow \varepsilon$			$S' \rightarrow \varepsilon$
E		$E \rightarrow b$				

# LL(1)文法

- 如果一个文法的预测分析表M不含多重定义的表项，则称该文法为LL(1)文法。
- LL(1)的含义：
  - 第一个L表示从左至右扫描输入符号串
  - 第二个L表示生成输入串的一个最左推导
  - 1表示在决定分析程序的每步动作时，向前看一个符号

# LL(1)文法的判断

## ■ 根据文法产生式判断:

一个文法是LL(1)文法, 当且仅当它的每一个产生式  $A \rightarrow \alpha \mid \beta$ , 满足:

□  $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \phi$  并且

□ 若 $\beta$ 推导出 $\varepsilon$ , 则 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \phi$

## ■ 根据分析表判断:

如果利用算法4.2构造出的分析表中不含多重定义的表项, 则文法是LL(1)文法。

## ■ 文法4.5不是LL(1)文法

□ 因为 $\text{FIRST}(es) \cap \text{FOLLOW}(S') = \{e\}$

□  $M[S', e]$ 中有两个产生式

# 练习

有文法G[S]:  $S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S$

(1) 判断该文法是否为LL(1)文法? 不是, 做(2); 是, 做(3)。

(2) 改写文法为LL(1)文法, 继续做(3)。

(3) 构造文法的FIRST和FOLLOW集合, 继续做(4)。

(4) 构造文法的LL(1)分析表。

解答:

■ 文法含有左递归, 故不是LL(1)文法

■ 改写文法: 消除左递归

$S \rightarrow (L) \mid a$

$L \rightarrow SL'$

$L' \rightarrow ,SL' \mid \epsilon$

$\text{FIRST}((L)) \cap \text{FIRST}(a) = \phi$

$\text{FIRST}(,SL') \cap \text{FOLLOW}(L') = \phi$

	FIRST	FOLLOW
S	( a	\$ , )
L	( a	)
L'	, ε	)

	a	(	)	,	\$
S	$S \rightarrow a$	$S \rightarrow (L)$			
L	$L \rightarrow SL'$	$L \rightarrow SL'$			
L'			$L' \rightarrow \epsilon$	$L' \rightarrow ,SL'$	

# 练习

有文法G[S]:

(1)  $S \rightarrow aAcBe$  (2)  $A \rightarrow b$  (3)  $A \rightarrow Ab$  (4)  $B \rightarrow d$   
 请构造其LL(1)分析表，并分析 abbcde。

解答:

(1) 改写文法:

$S \rightarrow aAcBe$   
 $A \rightarrow bA'$   
 $A' \rightarrow bA' \mid \varepsilon$   
 $B \rightarrow d$

(2) 构造FIRST和FOLLOW集合

	First	follow
S	a	\$
A	b	c
A'	b, $\varepsilon$	c
B	d	e

(3) 构造LL(1)分析表

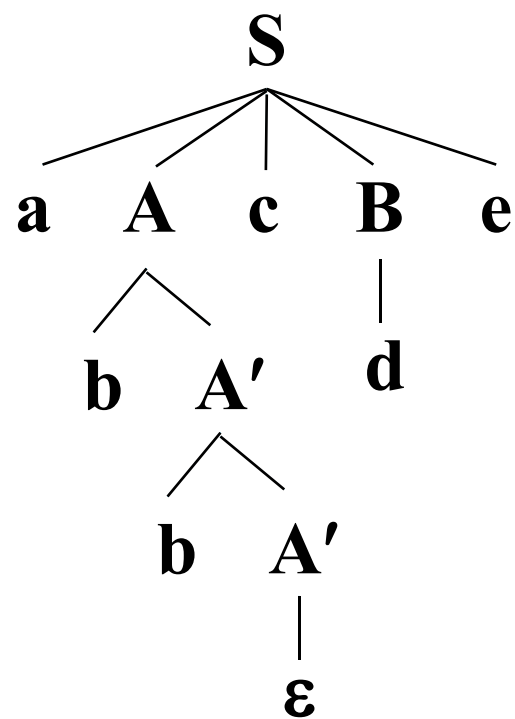
	a	b	c	d	e	\$
S	$S \rightarrow aAcBe$					
A		$A \rightarrow bA'$				
A'		$A' \rightarrow bA'$	$A' \rightarrow \varepsilon$			
B				$B \rightarrow d$		

# abbcde 的分析过程

	a	b	c	d	e	\$
S	$S \rightarrow aAcBe$					
A		$A \rightarrow bA'$				
A'		$A' \rightarrow bA'$	$A' \rightarrow \varepsilon$			
B				$B \rightarrow d$		

\$	e	d	c	A'	b
----	---	---	---	----	---

a b b c d e \$



*done !*

# 预测分析方法中的错误处理示例

## ■ 发现错误:

- (1)  $X \in V_T$ , 但  $X \neq a$ ;
- (2)  $X \in V_N$ , 但  $M[X, a]$  为空。

## ■ 处理方法:

情况(1): 弹出栈顶的终结符号;

情况(2): 跳过剩余输入符号串中的若干个符号, 直到可以继续进行分析为止。

## ■ 带有同步化信息的分析表的构造

- 对于  $A \in V_N$ ,  $b \in \text{FOLLOW}(A)$ , 若  $M[A, b]$  为空, 则加入“synch”

## ■ 带有同步化信息的分析表的使用

- 若  $M[A, b]$  为空, 则跳过  $a$ ;
- 若  $M[A, b]$  为 synch, 则弹出  $A$ 。

# 示例

## ■ 构造文法4.4的带有同步化信息的分析表

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

	FIRST	FOLLOW
E	(, id	\$, )
E'	+, $\varepsilon$	\$, )
T	(, id	\$(, +, )
T'	*, $\varepsilon$	\$(, +, )
F	(, id	\$(, +, *, )

	id	+	*	(	)	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch



# 示例

作业：  
4.1 4.3 4.5

栈	输入	输出
\$E	*id*+id\$	出错, $M[E,*]=$ 空白, 跳过*
\$E	id*+id\$	$E \rightarrow TE'$
\$E'T	id*+id\$	$T \rightarrow FT'$
\$E'T'F	id*+id\$	$F \rightarrow id$
\$E'T'id	id*+id\$	
\$E'T'	*+id\$	$T' \rightarrow *FT'$
\$E'T'F*	*+id\$	
\$E'T'F	+id\$	出错, $M[F,+]=synch$ , 弹出F
\$E'T'	+id\$	$T' \rightarrow \epsilon$
\$E'	+id\$	$E' \rightarrow +TE'$
\$E'T+	+id\$	
\$E'T	id\$	$T \rightarrow FT'$
\$E'T'F	id\$	$F \rightarrow id$
\$E'T'id	id\$	
\$E'T'	\$	$T' \rightarrow \epsilon$
\$E'	\$	$E' \rightarrow \epsilon$
\$	\$	

	id	+	*	(	)	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch

- 带有同步化信息的分析表的使用
  - 若 $M[A,b]$ 为空, 则跳过a;
  - 若 $M[A,b]$ 为synch, 则弹出A。
- $X \in V_T$ , 但 $X \neq a$  弹出栈顶的终结符号;

# 课堂练习1

有如下文法：

$$E \rightarrow E \vee T \mid T$$

$$T \rightarrow T \wedge F \mid F$$

$$F \rightarrow \neg F \mid (E) \mid t \mid f$$

(1) 该文法是LL(1)文法吗？说明理由。

若是，做(3)，若不是，做(2)

(2) 请改写该文法为LL(1)文法，继续做(3)。

(3) 构造每个非终结符号的FIRST和FOLLOW函数，继续做(4)。

(4) 构造LL(1)分析表。

# 参考答案

$E \rightarrow E \vee T \mid T$

$T \rightarrow T \wedge F \mid F$

$F \rightarrow \neg F \mid (E) \mid t \mid f$

(1) 由于该文法存在左递归，所以不是LL(1)文法。

(2) 改写文法。消除其中的左递归，得到文法G'：

$E \rightarrow TE'$

$E' \rightarrow \vee TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow \wedge FT' \mid \varepsilon$

$F \rightarrow \neg F \mid (E) \mid t \mid f$

(3) 每个非终结符号的FIRST和FOLLOW集合如右：

	FIRST	FOLLOW
E	$\neg, (, t, f$	$\$, )$
E'	$\vee, \varepsilon$	$\$, )$
T	$\neg, (, t, f$	$\vee, \$, )$
T'	$\wedge, \varepsilon$	$\vee, \$, )$
F	$\neg, (, t, f$	$\wedge, \vee, \$, )$

(4) 文法的LL(1)分析表如下：

	$\neg$	$\wedge$	$\vee$	t	f	(	)	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	$E \rightarrow TE'$	$E \rightarrow TE'$		
E'			$E' \rightarrow \vee TE'$				$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$	$T \rightarrow FT'$	$T \rightarrow FT'$		
T'		$T' \rightarrow \wedge FT'$	$T' \rightarrow \varepsilon$				$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \neg F$			$F \rightarrow t$	$F \rightarrow f$	$F \rightarrow (E)$		



## 4.3 自底向上分析方法

- 对输入串的扫描：自左向右
- 分析树的构造：自底向上
- 分析过程：
  - 从输入符号串开始分析
  - 查找当前句型的“可归约串”
  - 使用规则，把它归约成相应的非终结符号
  - 重复
- 关键：找出“可归约串”
- 常用方法：
  - 优先分析方法
  - LR分析方法

# 优先分析法简介 (\*)

- 分为：简单优先分析法和算符优先分析法。
- 简单优先分析法
  - 规范归约。
  - 按照文法符号（包括终结符号和非终结符号）之间的优先关系确定当前句型的“可归约串”。
  - 分析效率低，且只适用于简单优先文法。
  - 简单优先文法，满足以下两个条件：
    - (1) 任何两个文法符号之间最多存在一种优先关系。
    - (2) 不存在具有相同右部的产生式。

# 优先分析法简介

## ■ 算符优先分析法

- 只考虑终结符号之间的优先关系。
- 分析速度快，不是规范归约，且只适用于算符优先文法。
- 算符文法：  
没有形如 $A \rightarrow \dots BC \dots$ 的产生式，其中 $B, C \in V_N$
- 算符优先文法：
  - 算符文法、且不含有 $\varepsilon$ -产生式；
  - 任何两个构成序对的终结符号之间最多有 $>$ 、 $=$ 和 $<$ 三种优先关系中的一种成立。
- “可归约串”是句型的“最左素短语”。
  - 素短语：句型的一个短语，至少含有一个终结符号，并且除它自身之外不再含有其他更小的素短语。
  - 最左素短语：处于句型最左边的那个素短语。

# “移进-归约” 分析方法

- 符号栈：存放文法符号

- 分析过程：

- (1) 把输入符号一个个地移进栈中。

- (2) 当栈顶的符号串形成某个产生式的一个候选式时，  
在一定条件下，把该符号串替换(即归约)为该产生式的左部符号。

- (3) 重复(2)，直到栈顶符号串不再是“可归约串”为止。

- (4) 重复(1)~(3)，直到最终归约出文法开始符号S。

# 规范归约

例：分析符号串abbcde是否为如下文法的句子。

$$(1) S \rightarrow aAcBe$$

$$(2) A \rightarrow b$$

$$(3) A \rightarrow Ab$$

$$(4) B \rightarrow d \quad (\text{文法4.1})$$

## ■ 最右推导

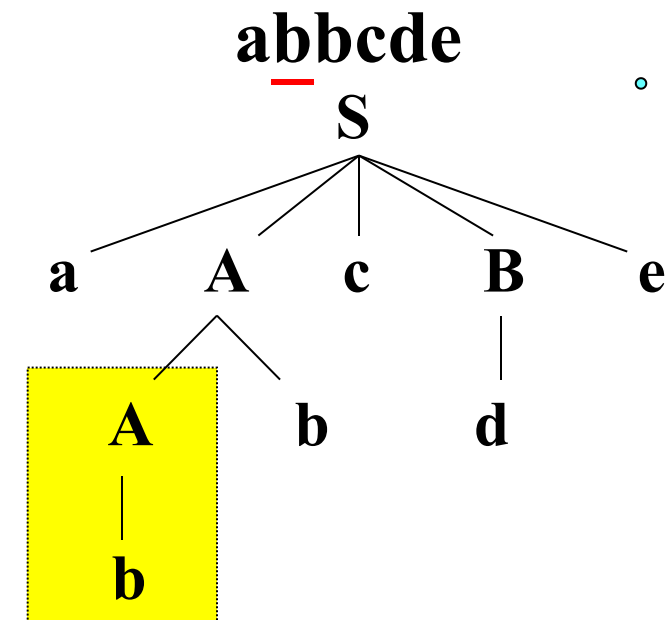
$$S \xRightarrow[rm]{(1)} aAcBe \xRightarrow[rm]{(4)} aAcde \xRightarrow[rm]{(3)} aAbcde \xRightarrow[rm]{(2)} abbcde$$



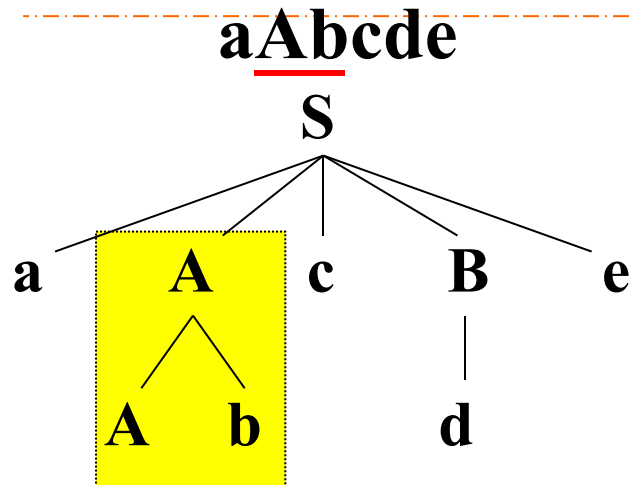
# 规范归约

句柄? 子树?

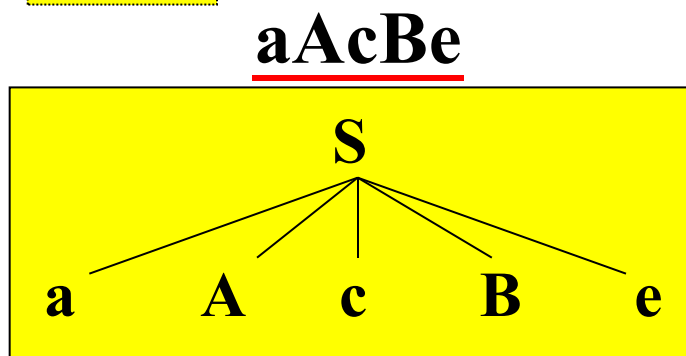
$\Rightarrow$



$\Rightarrow$   
(2)



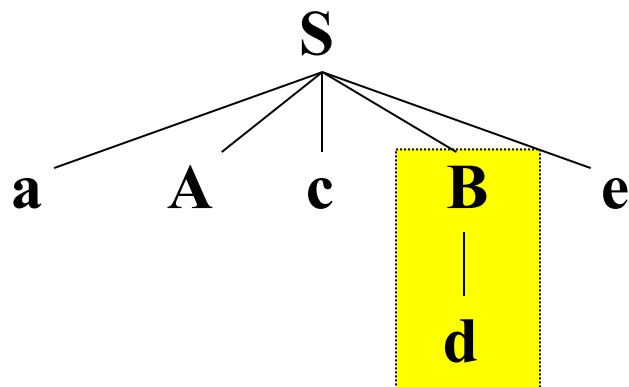
$\Downarrow$  (3)



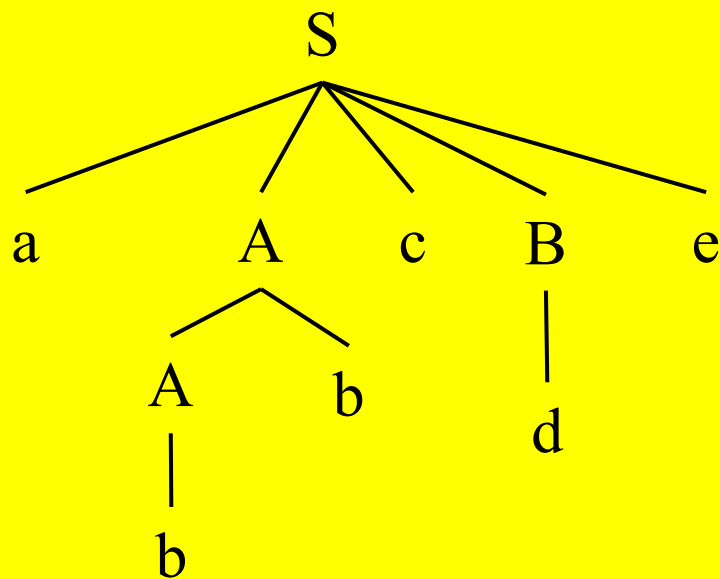
$\Downarrow$  (1)  
S

aAcde

$\Leftarrow$   
(4)



# 规范归约：最右推导的逆过程



a b b c d e

a A b c d e

a A c d e

a A c B e

S

S

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$

$S \rightarrow aAcBe$

最右推导：

$S \Rightarrow aAcBe$

$\Rightarrow aAcde$

$\Rightarrow aAbcde$

$\Rightarrow abbcde$

# 规范归约的定义

定义：假定 $\alpha$ 是文法 $G$ 的一个句子，我们称右句型序列 $\alpha_n, \alpha_{n-1}, \dots, \alpha_1, \alpha_0$ 是 $\alpha$ 的一个规范归约，如果序列满足：

(1)  $\alpha_n = \alpha, \alpha_0 = S$

(2) 对任何  $i$  ( $0 < i \leq n$ )， $\alpha_{i-1}$  是经过把  $\alpha_i$  的句柄替换为相应产生式的左部符号而得到的。

- 规范归约是关于 $\alpha$ 的一个最右推导的逆过程，因此规范归约也称为最左归约。
- $abbcde$ 的一个规范归约是如下的右句型序列：  
 $abbcde, aAbcde, aAcde, aAcBe, S$ 。

# 句柄的最左性

- 规范句型：最右推导得到的句型
- 规范句型的特点：句柄之后没有非终结符号
- 利用句柄的最左性：与符号栈的栈顶相关
- 不同的最右推导，其逆过程也是不同

例：考虑文法  $E \rightarrow E+E | E * E | (E) | id$  的句子  $id+id*id$

$E \Rightarrow E+E \Rightarrow E+E * E \Rightarrow \underline{E+E * id} \Rightarrow E+id * id \Rightarrow id+id * id$

$E \Rightarrow E * E \Rightarrow E * id \Rightarrow \underline{E+E * id} \Rightarrow E+id * id \Rightarrow id+id * id$

句柄：  $E+E$

产生式：  $E \rightarrow E+E$

句柄：  $id$

产生式：  $E \rightarrow id$

# “移进-归约”方法的实现

- 使用一个寄存文法符号的栈和一个存放输入字符串的缓冲区。
  - 分析开始时，先将符号 \$ 入栈，以示栈底；
  - 将 \$ 置入输入字符串之后，以示字符串的结束。

栈	输入
\$	$\omega$ \$

开始格局

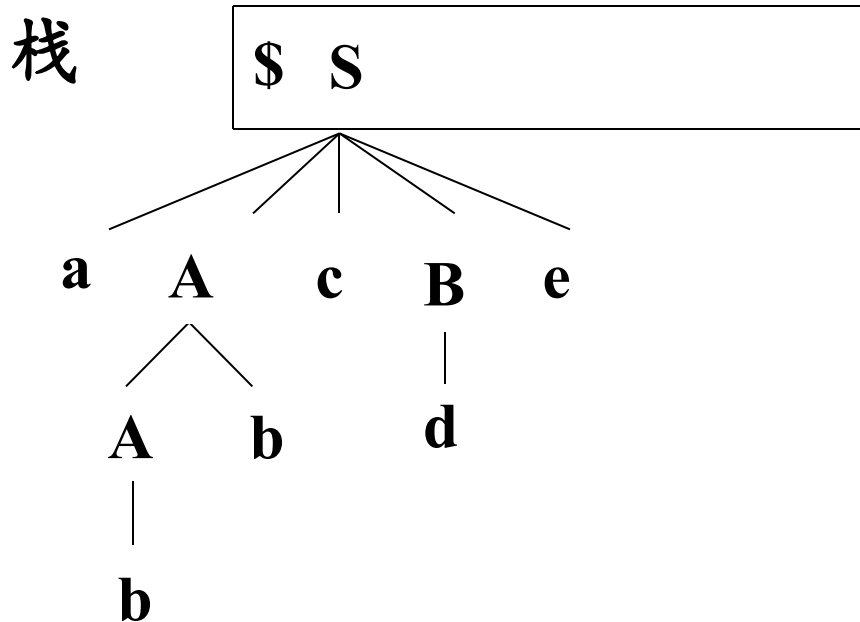
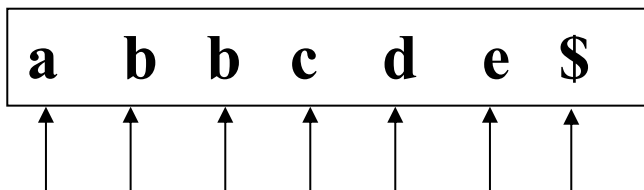
栈	输入
\$S	\$

结束格局

# 例：对文法4.6的句子 abbcde 的规范归约过程

栈	输入	分析动作
(1) \$	abbcde\$	shift
(2) \$a	bbcde\$	shift
(3) \$a <u>b</u>	bcde\$	reduce by $A \rightarrow b$
(4) \$aA	bcde\$	shift
(5) \$a <u>Ab</u>	cde\$	reduce by $A \rightarrow Ab$
(6) \$aA	cde\$	shift
(7) \$aAc	de\$	shift
(8) \$aAc <u>d</u>	e\$	reduce by $B \rightarrow d$
(9) \$aAcB	e\$	shift
(10) \$aAc <u>Be</u>	\$	reduce by $S \rightarrow aAcBe$
(11) \$\$	\$	accept

# 句子 abbcd e 的规范归约过程



a b b c d e

a A b c d e

a A c d e

a A c B e

S

# “移进-归约” 分析方法的分析动作

**移进**：把下一个输入符号移进到栈顶。

**归约**：用适当的归约符号去替换这个串。

**接受**：宣布分析成功，停止分析。

**错误处理**：调用错误处理程序进行诊断和恢复。

## 分析过程中的动作冲突：

“移进-归约” 冲突

“归约-归约” 冲突





## 4.4 LR分析方法

### LR分析技术概述

4.4.1 LR分析程序的模型及工作过程

4.4.2 SLR(1)分析表的构造

4.4.3 LR(1)分析表的构造

4.4.4 LALR(1)分析表的构造

4.4.5 LR分析方法对二义文法的应用

4.4.6 LR分析的错误处理与恢复

# LR分析技术概述

## ■ LR(k)的含义：

- L 表示自左至右扫描输入符号串
- R 表示为输入符号串构造一个最右推导的逆过程
- k 表示为作出分析决定而向前看的输入符号的个数

## ■ LR分析方法的基本思想

- “历史信息”：记住已经移进和归约出的整个符号串；
- “预测信息”：根据所用的产生式推测未来可能遇到的输入符号；
- 根据“历史信息”和“预测信息”，以及“现实”的输入符号，确定栈顶的符号串是否构成相对于某一产生式的句柄。

# LR分析技术概述

## ■ LR分析技术是一种比较完备的技术

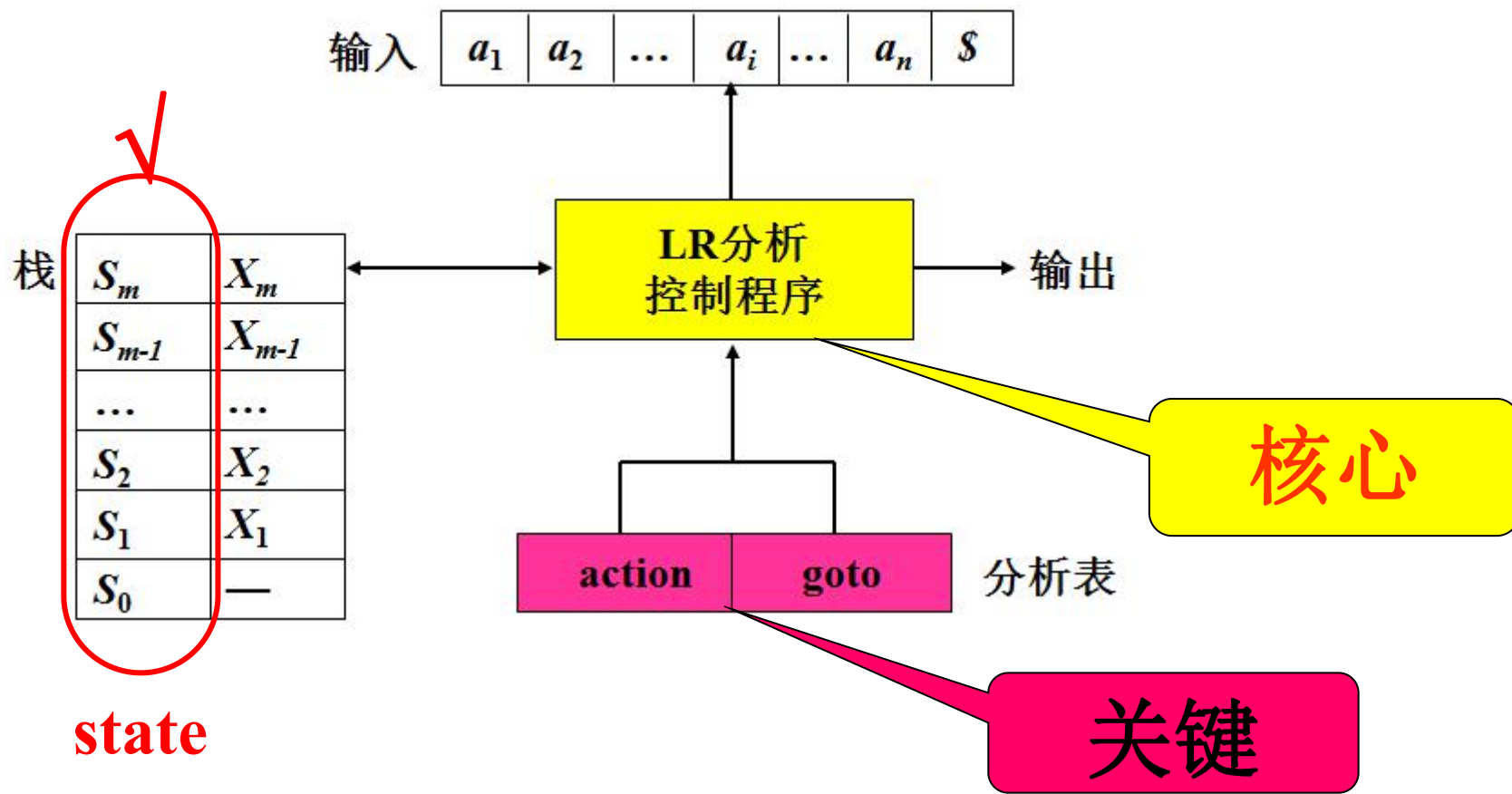
- 可以分析所有能用上下文无关文法书写的程序设计语言的结构；
- 最一般的无回溯的“移进-归约”方法；
- 能分析的文法类是预测分析方法能分析的文法类的真超集；
- 分析过程中，能及时发现错误，快到自左至右扫描输入的最大可能。

## ■ LR分析方法的不足之处

- 手工编写LR分析程序的工作量太大
- 需要专门的工具，即LR分析程序生成器（如YACC）

## 4.4.1 LR分析程序的模型及工作过程

### ■ LR分析程序的模型



# 分析表

- LR分析控制程序工作的依据
- $\text{goto}[S_m, X]$ : 状态  $S_m$  经  $X$  转移的后继状态
- $\text{action}[S_m, a_i]$ : 状态  $S_m$  面临输入符号  $a_i$  时应采取的分析动作
  - **shift S**: 把当前输入符号  $a_i$  及状态S推进栈, 向前扫描指针前移。这里,  $S = \text{goto}[S_m, a_i]$ 。
  - **reduce by  $A \rightarrow \beta$** : 若  $\beta$  的长度为  $r$ , 则从栈顶起向下弹出  $r$  项, 使  $S_{m-r}$  成为栈顶状态, 然后把文法符号  $A$  及状态S推进栈。这里,  $S = \text{goto}[S_{m-r}, A]$ 。
  - **accept**: 宣布分析成功, 停止分析。
  - **error**: 调用出错处理程序, 进行错误恢复。

# LR分析控制程序

- 核心部分，对所有的LR分析程序都是一样的。
- LR分析控制程序的工作过程：
  - 分析开始时，初始的二元式为： $(S_0, a_1 a_2 \dots a_n \$)$
  - 分析过程中每步的结果，均可表示为如下的二元式： $(S_0 S_1 S_2 \dots S_m, a_i a_{i+1} \dots a_n \$)$
  - 若  $\text{action}[S_m, a_i] = \text{shift } S, S = \text{goto}[S_m, a_i]$ ，  
则二元式变为： $(S_0 S_1 \dots S_m S, a_{i+1} \dots a_n \$)$
  - 若  $\text{action}[S_m, a_i] = \text{reduce by } A \rightarrow \beta$ ，  
则二元式变为： $(S_0 S_1 \dots S_{m-r} S, a_i a_{i+1} \dots a_n \$)$
  - 若  $\text{action}[S_m, a_i] = \text{accept}$ （接受），  
则分析成功，二元式变化过程终止。
  - 若  $\text{action}[S_m, a_i] = \text{error}$ （出错），  
则发现错误，调用错误处理程序。

## 算法4.3 LR分析控制程序

输入：文法  $G$  的一张分析表和一个输入符号串  $\omega$

输出：若  $\omega \in L(G)$ ，得到  $\omega$  的自底向上的分析，否则报错

方法：开始时，初始状态  $S_0$  在栈顶， $\omega \$$  在输入缓冲区中。

置  $ip$  指向  $\omega \$$  的第一个符号；

do {

    令  $S$  是栈顶状态， $a$  是  $ip$  所指向的符号

    if (action[ $S, a$ ] == shift  $S'$ ) {

        把  $a$  和  $S'$  分别压入符号栈和状态栈；

        推进  $ip$ ，使它指向下一个输入符号；

    };

    else if (action[ $S, a$ ] == reduce by  $A \rightarrow \beta$ ) {

        从栈顶弹出  $|\beta|$  个符号； // 令  $S'$  是现在的栈顶状态

        把  $A$  和 goto[ $S', A$ ] 分别压入符号栈和状态栈；

        输出产生式  $A \rightarrow \beta$ ；

    };

    else if (action[ $S, a$ ] == accept) return;

        else error();

} while(1).

移进

归约

# 对输入符号串 id+id\*id 的分析过程

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			

输入：id + id \* id \$  
↑

分析栈：

0	1
-	E

*Accept!*

- (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)  $F \rightarrow id$



# “活前缀”的概念

前缀? ==>

- 定义：一个规范句型的一个前缀，如果不含句柄之后的任何符号，则称它为该句型的一个活前缀。
- 例：右句型aAbcde的句柄是Ab  
该句型的活前缀有 $\varepsilon$ 、a、aA、aAb；  
句型aAcde的句柄是d  
它的活前缀有： $\varepsilon$ 、a、aA、aAc、aAcd。
- 之所以称它们为活前缀，是因为在其右边增加某些终结符号之后，就可以使之成为一个规范句型。
- 分析过程中  $(S_0X_1S_1X_2\ldots X_mS_m, a_ia_{i+1}\ldots a_n\$)$   
 $X_1X_2\ldots X_ma_ia_{i+1}\ldots a_n$  是一个右句型，  
 $X_1X_2\ldots X_m$  是它的一个活前缀。

# 示例：对输入符号串id+id\*id的分析过程

栈	输入	分析动作	
0	id+id*id\$	shift	5
0 5	+id*id\$	reduce by $F \rightarrow id$	goto[0,F]=3
<u>id</u>			
0 3	+id*id\$	reduce by $T \rightarrow F$	goto[0,T]=2
<u>F</u>			
0 2	+id*id\$	reduce by $E \rightarrow T$	goto[0,E]=1
<u>T</u>			
0 1	+id*id\$	shift	6
<u>E</u>			
0 1 6	id*id\$	shift	5
<u>E +</u>			
0 1 6 5	*id\$	reduce by $F \rightarrow id$	goto[6,F]=3
<u>E + id</u>			
0 1 6 3	*id\$	reduce by $T \rightarrow F$	goto[6,T]=9
<u>E + F</u>			

# 示例：对输入符号串id+id\*id的分析过程

栈	输入	分析动作
0 1 6 9 E + T	*id\$	shift 7
0 1 6 9 7 E + T *	id\$	shift 5
0 1 6 9 7 5 E + T * id	\$	reduce by $F \rightarrow id$ goto[7,F]=10
0 1 6 9 7 10 E + T * F	\$	reduce by $T \rightarrow T * F$ goto[6,T]=9
0 1 6 9 E + T	\$	reduce by $E \rightarrow E + T$ goto[0,E]=1
0 1 E	\$	ACC

## 4.4.2 SLR(1)分析表的构造

- 中心思想:

- 为给定的文法构造一个识别它所有活前缀的DFA
- 根据该DFA构造文法的分析表

- 构造识别给定文法的所有活前缀的DFA

- SLR(1)分析表的构造

# 构造识别给定文法所有活前缀的DFA

## ■ 活前缀与句柄之间的关系

- 活前缀不含有句柄的任何符号
- 活前缀只含有句柄的部分符号
- 活前缀已经含有句柄的全部符号

## ■ 分析过程中，分析栈中出现的活前缀

- 第一种情况，期望从剩余输入串中能够看到由某产生式  $A \rightarrow \alpha$  的右部  $\alpha$  所推导出的终结符号串；
- 第二种情况，某产生式  $A \rightarrow \alpha_1 \alpha_2$  的右部子串  $\alpha_1$  已经出现在栈顶，期待从剩余的输入串中能够看到  $\alpha_2$  推导出的符号串；
- 第三种情况，某一产生式  $A \rightarrow \alpha$  的右部符号串  $\alpha$  已经出现在栈顶，用该产生式进行归约。

# LR(0)项目

- 右部某个位置上标有圆点的产生式称为文法G的一个LR(0)项目
- 产生式 $A \rightarrow XYZ$ 对应应有4个LR(0)项目

$A \rightarrow \bullet XYZ$	}	移进-待约项目
$A \rightarrow X \bullet YZ$		
$A \rightarrow XY \bullet Z$		
$A \rightarrow XYZ \bullet$		归约项目

- 归约项目：圆点在产生式最右端的LR(0)项目
- 接受项目：对文法开始符号的归约项目
- 待约项目：圆点后第一个符号为非终结符号的LR(0)项目
- 移进项目：圆点后第一个符号为终结符号的LR(0)项目

产生式 $A \rightarrow \varepsilon$ ，只有一个LR(0)归约项目  $A \rightarrow \bullet$ 。

# 拓广文法

- 任何文法  $G=(V_T, V_N, S, \varphi)$ , 都有等价的文法:  
 $G'=(V_T, V_N \cup \{S'\}, S', \varphi \cup \{S' \rightarrow S\})$ ,  
称  $G'$  为  $G$  的拓广文法。
- 拓广文法  $G'$  的接受项目是唯一的(即  $S' \rightarrow S \bullet$ )

## 定义4.10: LR(0)有效项目

- 项目  $A \rightarrow \beta_1 \bullet \beta_2$  对活前缀  $\gamma = \alpha \beta_1$  是有效的, 如果存在一个规范推导:

$$S \xRightarrow{*} \alpha A \omega \Rightarrow \alpha \beta_1 \beta_2 \omega$$

- 推广: 若项目  $A \rightarrow \alpha \bullet B \beta$  对活前缀  $\gamma = \delta \alpha$  是有效的, 并且有产生式  $B \rightarrow \eta$ , 则项目  $B \rightarrow \bullet \eta$  对活前缀  $\gamma = \delta \alpha$  也是有效的。

$$S \xRightarrow{*} \delta A \omega \Rightarrow \delta \alpha B \beta \omega \xRightarrow{*} \delta \alpha B \lambda \Rightarrow \delta \alpha \eta \lambda$$

- LR(0)项目  $S' \rightarrow \bullet S$  是活前缀  $\varepsilon$  的有效项目**

(这里取  $\gamma = \varepsilon$ ,  $\beta_1 = \varepsilon$ ,  $\beta_2 = S$ ,  $A = S'$ )

$$S' \Rightarrow S$$



# LR(0)有效项目集和LR(0)项目集规范族

文法G的某个活前缀 $\gamma$ 的所有LR(0)有效项目组成的集合称为 $\gamma$ 的LR(0)有效项目集。

文法G的所有LR(0)有效项目集组成的集合称为G的LR(0)项目集规范族。

## 定义4.11：闭包(closure)

设 $I$ 是文法 $G$ 的一个LR(0)项目集合， $\text{closure}(I)$ 是从 $I$ 出发，用下面的方法构造的项目集：

- (1)  $I$ 中的每一个项目都属于 $\text{closure}(I)$ ;
- (2) 若项目  $A \rightarrow \alpha \bullet B \beta$  属于  $\text{closure}(I)$ ,  
且 $G$ 有产生式  $B \rightarrow \eta$ ,  
若  $B \rightarrow \bullet \eta$  不属于  $\text{closure}(I)$ ,  
则将  $B \rightarrow \bullet \eta$  加入  $\text{closure}(I)$ ;
- (3) 重复规则(2)，直到 $\text{closure}(I)$ 不再增大为止。

# 算法4.4 closure(I)的构造过程

输入：项目集合 $I$ 。

输出：集合 $J=\text{closure}(I)$ 。

方法：

$J=I$ ;

do {

$J_{\text{new}}=J$ ;

for (  $J_{\text{new}}$ 中的每一个项目  $A\rightarrow\alpha\bullet B\beta$   
和文法 $G$ 的每个产生式  $B\rightarrow\eta$ )

if ( $B\rightarrow\bullet\eta\notin J$ ) 把  $B\rightarrow\bullet\eta$  加入 $J$ ;

} while ( $J_{\text{new}}\neq J$ ).

## 定义4.12：转移函数go

若I是文法G的一个LR(0)项目集，X是一个文法符号，  
定义

$$\text{go}(I, X) = \text{closure}(J)$$

其中： $J = \{ A \rightarrow \alpha X \cdot \beta \mid \text{当 } A \rightarrow \alpha \cdot X \beta \text{ 属于 } I \text{ 时} \}$

$\text{go}(I, X)$ 称为转移函数

项目  $A \rightarrow \alpha X \cdot \beta$  称为  $A \rightarrow \alpha \cdot X \beta$  的后继

直观含义：

若I中的项目  $A \rightarrow \alpha \cdot X \beta$  是某个活前缀  $\gamma = \delta \alpha$  的有效项目，J中的项目  $A \rightarrow \alpha X \cdot \beta$  是活前缀  $\delta \alpha X$ （即  $\gamma X$ ）的有效项目。

## 算法4.5 构造文法G的LR(0)项目集规范族

输入：文法G

输出：G的LR(0)项目集规范族C

方法：

$C = \{\text{closure}(\{S' \rightarrow \bullet S\})\};$

do

for (对C中的每一个项目集I和每一个文法符号X)

if ( $\text{go}(I, X)$ 不为空, 且不在C中)

把 $\text{go}(I, X)$ 加入C中;

while (没有新项目集加入C中);

这里  $\text{closure}(\{S' \rightarrow \bullet S\})$  是活前缀  $\varepsilon$  的有效项目集

# 示例

- 构造如下文法G的LR(0)项目集规范族:

$S \rightarrow aA \mid bB \quad A \rightarrow cA \mid d \quad B \rightarrow cB \mid d$  (文法4.6)

- ① 拓广文法G':

$S' \rightarrow S \quad S \rightarrow aA \mid bB \quad A \rightarrow cA \mid d \quad B \rightarrow cB \mid d$

- ② 活前缀  $\varepsilon$  的有效项目集

$I_0 = \text{closure}(\{S' \rightarrow \bullet S\}) = \{S' \rightarrow \bullet S, S \rightarrow \bullet aA, S \rightarrow \bullet bB\}$

- ③ 从 $I_0$ 出发的转移有

活前缀

$I_1 = \text{go}(I_0, S) = \text{closure}(\{S' \rightarrow S \bullet\}) = \{S' \rightarrow S \bullet\}$

——S

$I_2 = \text{go}(I_0, a) = \text{closure}(\{S \rightarrow a \bullet A\})$

——a

$= \{S \rightarrow a \bullet A, A \rightarrow \bullet cA, A \rightarrow \bullet d\}$

$I_3 = \text{go}(I_0, b) = \text{closure}(\{S \rightarrow b \bullet B\})$

——b

$= \{S \rightarrow b \bullet B, B \rightarrow \bullet cB, B \rightarrow \bullet d\}$

# 示例

④  $I_2 = \{S \rightarrow a \bullet A, A \rightarrow \bullet cA, A \rightarrow \bullet d\}$

从 $I_2$ 出发的转移有

活前缀

$$I_4 = \text{go}(I_2, A) = \text{closure}(\{S \rightarrow aA \bullet\}) = \{S \rightarrow aA \bullet\}$$

— aA

$$I_5 = \text{go}(I_2, c) = \text{closure}(\{A \rightarrow c \bullet A\})$$

— ac

$$= \{A \rightarrow c \bullet A, A \rightarrow \bullet cA, A \rightarrow \bullet d\}$$

$$I_6 = \text{go}(I_2, d) = \text{closure}(\{A \rightarrow d \bullet\}) = \{A \rightarrow d \bullet\}$$

— ad

⑤  $I_3 = \{S \rightarrow b \bullet B, B \rightarrow \bullet cB, B \rightarrow \bullet d\}$

从 $I_3$ 出发的转移有

活前缀

$$I_7 = \text{go}(I_3, B) = \text{closure}(\{S \rightarrow bB \bullet\}) = \{S \rightarrow bB \bullet\}$$

— bB

$$I_8 = \text{go}(I_3, c) = \text{closure}(\{B \rightarrow c \bullet B\})$$

— bc

$$= \{B \rightarrow c \bullet B, B \rightarrow \bullet cB, B \rightarrow \bullet d\}$$

$$I_9 = \text{go}(I_3, d) = \text{closure}(\{B \rightarrow d \bullet\}) = \{B \rightarrow d \bullet\}$$

— bd

# 示例

⑥  $I_5 = \{ A \rightarrow c \bullet A, A \rightarrow \bullet cA, A \rightarrow \bullet d \}$

从 $I_5$ 出发的转移有

活前缀

$$I_{10} = \text{go}(I_5, A) = \text{closure}(\{A \rightarrow cA \bullet\}) = \{A \rightarrow cA \bullet\}$$

—acA

$$\text{go}(I_5, c) = \text{closure}(\{A \rightarrow c \bullet A\}) = I_5$$

—acc

$$\text{go}(I_5, d) = \text{closure}(\{A \rightarrow d \bullet\}) = I_6$$

—acd

⑦  $I_8 = \{ B \rightarrow c \bullet B, B \rightarrow \bullet cB, B \rightarrow \bullet d \}$

从 $I_8$ 出发的转移有

活前缀

$$I_{11} = \text{go}(I_8, B) = \text{closure}(\{B \rightarrow cB \bullet\}) = \{B \rightarrow cB \bullet\}$$

—bcB

$$\text{go}(I_8, c) = \text{closure}(\{B \rightarrow c \bullet B\}) = I_8$$

—bcc

$$\text{go}(I_8, d) = \text{closure}(\{B \rightarrow d \bullet\}) = I_9$$

—bcd

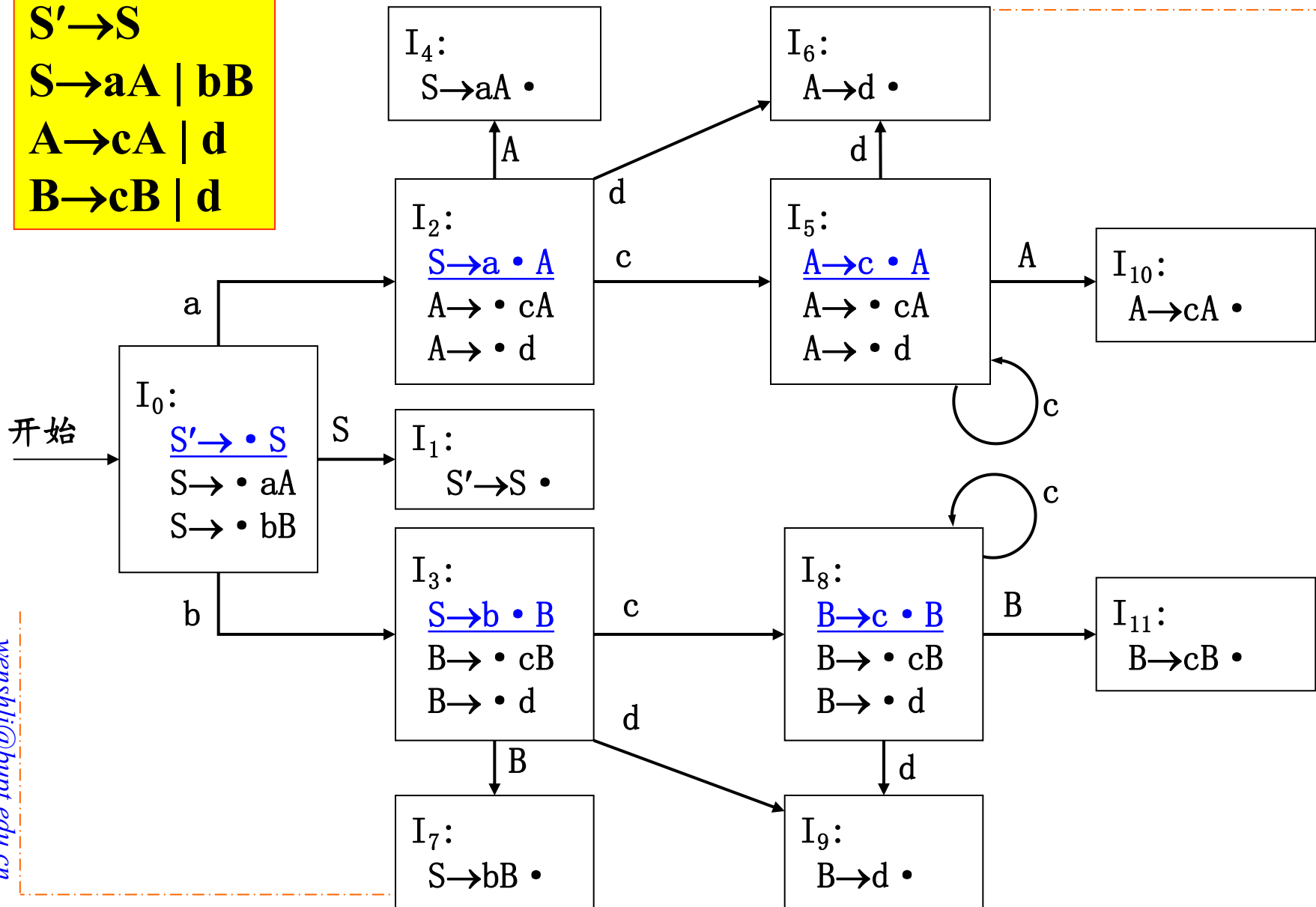
■ 文法 $G'$ 的LR(0)项目集规范族  $C = \{I_0, I_1, \dots, I_{11}\}$



# 识别文法G'的所有活前缀的DFA

练习：  
分析 accd

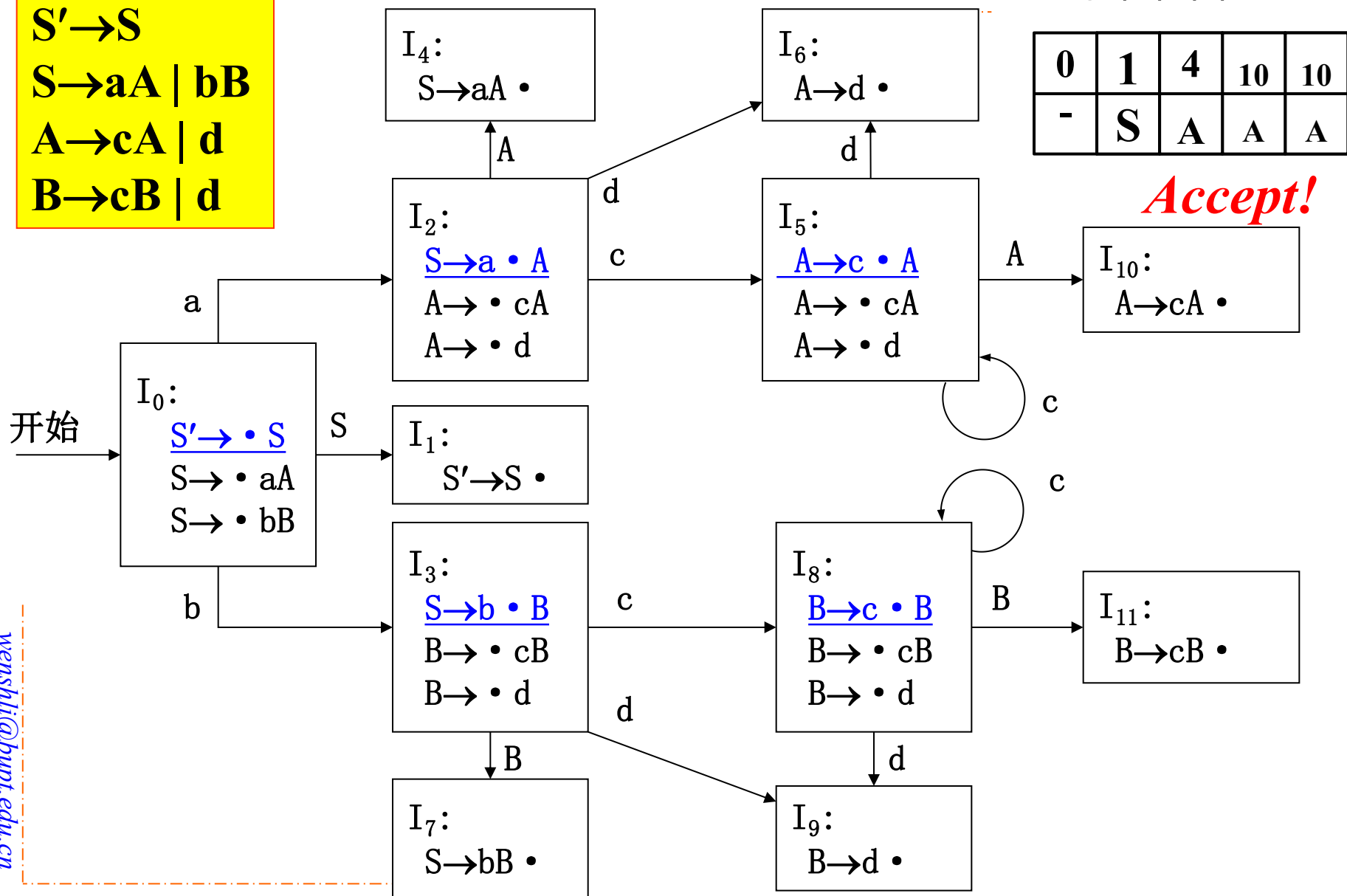
$S' \rightarrow S$   
 $S \rightarrow aA \mid bB$   
 $A \rightarrow cA \mid d$   
 $B \rightarrow cB \mid d$



## 示例： accd的分析过程

$$S' \rightarrow S$$
$$S \rightarrow aA \mid bB$$
$$\mathbf{A} \rightarrow \mathbf{cA} \mid \mathbf{d}$$

**B→cB | d**



# SLR(1)分析表的构造

## ■ SLR分析方法的一个特征

- 如果文法的有效项目集中有冲突动作，多数冲突可通过考察有关非终结符号的FOLLOW集合而得到解决

## ■ 如项目集： $I = \{X \rightarrow \alpha \cdot b \beta, A \rightarrow \alpha \cdot, B \rightarrow \beta \cdot\}$

- 存在移进-归约冲突
- 存在归约-归约冲突

## ■ 冲突的解决：查看FOLLOW(A)和FOLLOW(B)

- $\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \Phi$
- $b \notin \text{FOLLOW}(A)$  并且  $b \notin \text{FOLLOW}(B)$
- 决策：

- 当  $a=b$  时，把  $b$  移进栈里；
- 当  $a \in \text{FOLLOW}(A)$  时，用产生式  $A \rightarrow \alpha$  进行归约；
- 当  $a \in \text{FOLLOW}(B)$  时，用产生式  $B \rightarrow \beta$  进行归约。

## 算法4.6 构造SLR(1)分析表

输入：拓广文法 $G'$  输出： $G'$ 的SLR分析表 方法如下：

- 1.构造 $G'$ 的LR(0)项目集规范族 $C=\{I_0, I_1, \dots, I_n\}$ 。
- 2.对于状态 $i$ （对应于项目集 $I_i$ 的状态）的分析动作如下
  - a) 若 $A \rightarrow \alpha \bullet a \beta \in I_i$ ，且 $go(I_i, a)=I_j$ ，则置  $action[i, a]=S_j$
  - b) 若 $A \rightarrow \alpha \bullet \in I_i$ ，则对所有 $a \in FOLLOW(A)$ ，置  
 $action[i, a]=R A \rightarrow \alpha$
  - c) 若 $S' \rightarrow S \bullet \in I_i$ ，则置  $action[i, \$]=ACC$ ，表示分析成功
- 3.若 $go(I_i, A)=I_j$ ， $A$ 为非终结符号，则置  $goto[i, A]=j$
- 4.分析表中凡不能用规则(2)、(3)填入信息的空白表项，均置为出错标志error。
- 5.分析程序的初态是包含项目 $S' \rightarrow S$ 的有效项目集所对应的状态。

# 示例：构造文法4.6的SLR(1)分析表

- 构造出该文法的LR(0)项目集规范族、及识别文法活前缀的DFA
- 考察  $I_0 = \{ S' \rightarrow \bullet S, S \rightarrow \bullet aA, S \rightarrow \bullet bB \}$ 
  - 对项目  $S' \rightarrow \bullet S$ , 有  $go(I_0, S) = I_1$ , 所以  $goto[0, S] = 1$
  - 对项目  $S \rightarrow \bullet aA$ , 有  $go(I_0, a) = I_2$ , 所以  $action[0, a] = S2$
  - 对项目  $S \rightarrow \bullet bB$ , 有  $go(I_0, b) = I_3$ , 所以  $action[0, b] = S3$
- 考察  $I_1 = \{ S' \rightarrow S \bullet \}$ 
  - 项目  $S' \rightarrow S \bullet$  是接受项目, 所以  $action[1, \$] = ACC$
- 考察  $I_2 = \{ S \rightarrow a \bullet A, A \rightarrow \bullet cA, A \rightarrow \bullet d \}$ 
  - 对项目  $S \rightarrow a \bullet A$ , 有  $go(I_2, A) = I_4$ , 所以  $goto[2, A] = 4$
  - 对项目  $A \rightarrow \bullet cA$ , 有  $go(I_2, c) = I_5$ , 所以  $action[2, c] = S5$
  - 对项目  $A \rightarrow \bullet d$ , 有  $go(I_2, d) = I_6$ , 所以  $action[2, d] = S6$

# 示例：构造文法4.6的SLR(1)分析表

- 考察  $I_3 = \{S \rightarrow b \cdot B, B \rightarrow \cdot cB, B \rightarrow \cdot d\}$ 
  - 对项目  $S \rightarrow b \cdot B$ , 有  $go(I_3, B) = I_7$ , 所以  $goto[3, B] = 7$
  - 对项目  $B \rightarrow \cdot cB$ , 有  $go(I_3, c) = I_8$ , 所以  $action[3, c] = S8$
  - 对项目  $B \rightarrow \cdot d$ , 有  $go(I_3, d) = I_9$ , 所以  $action[3, d] = S9$
- 考察  $I_4 = \{S \rightarrow aA \cdot\}$ 
  - 项目  $S \rightarrow aA \cdot$  是归约项目, 且  $FOLLOW(S) = \{\$, \}$ , 所以  $action[4, \$] = R1$
- 考察  $I_5 = \{A \rightarrow c \cdot A, A \rightarrow \cdot cA, A \rightarrow \cdot d\}$ 
  - 对项目  $A \rightarrow c \cdot A$ , 有  $go(I_5, A) = I_{10}$ , 所以  $goto[5, A] = 10$
  - 对项目  $A \rightarrow \cdot cA$ , 有  $go(I_5, c) = I_5$ , 所以  $action[5, c] = S5$
  - 对项目  $A \rightarrow \cdot d$ , 有  $go(I_5, d) = I_6$ , 所以  $action[5, d] = S6$
- 考察  $I_6 = \{A \rightarrow d \cdot\}$ 
  - 项目  $A \rightarrow d \cdot$  是归约项目, 且  $FOLLOW(A) = \{\$, \}$ , 所以  $action[6, \$] = R4$

# 文法4.6的SLR(1)分析表

(0)  $S' \rightarrow S$

(1)  $S \rightarrow aA$

(2)  $S \rightarrow bB$

(3)  $A \rightarrow cA$

(4)  $A \rightarrow d$

(5)  $B \rightarrow cB$

(6)  $B \rightarrow d$

$I_0: S' \rightarrow \cdot S$

$S \rightarrow \cdot aA$

$S \rightarrow \cdot bB$

$I_1: S' \rightarrow S \cdot$

$I_3:$

$S \rightarrow b \cdot B$

$B \rightarrow \cdot cB$

$B \rightarrow \cdot d$

$I_2: S \rightarrow a \cdot A$

$A \rightarrow \cdot cA$

$A \rightarrow \cdot d$

$I_4: S \rightarrow aA \cdot$

$I_5: A \rightarrow c \cdot A$

$A \rightarrow \cdot cA$

$A \rightarrow \cdot d$

$I_6: A \rightarrow d \cdot$

$I_8: B \rightarrow c \cdot B$

$B \rightarrow \cdot cB$

$B \rightarrow \cdot d$

$I_7: S \rightarrow bB \cdot$

$I_9: B \rightarrow d \cdot$

$I_{10}: A \rightarrow cA \cdot$

$I_{11}: B \rightarrow cB \cdot$

状态	action					goto		
	a	b	c	d	\$	S	A	B
0	S2	S3				1		
1					acc			
2			S5	S6			4	
3			S8	S9				7
4					R1			
5			S5	S6			10	
6					R4			
7					R2			
8			S8	S9				11
9					R6			
10					R3			
11					R5			

# SLR(1)文法

- 若用算法4.5构造出来的分析表不含有冲突，则该分析表称为该文法的SLR(1)分析表
- 具有SLR(1)分析表的文法称为SLR(1)文法



## 课堂练习2

证明下面的文法是LL(1)的，但不是SLR(1)的。

$$S \rightarrow (X \mid E) \mid F$$
$$X \rightarrow E) \mid F]$$
$$E \rightarrow A$$
$$F \rightarrow A$$
$$A \rightarrow \varepsilon$$

# 解答：证明该文法是LL(1)文法

$S \rightarrow (X \mid E \mid F)$   
 $X \rightarrow E \mid F]$   
 $E \rightarrow A$   
 $F \rightarrow A$   
 $A \rightarrow \epsilon$

- 该文法每个非终结符号的FIRST集和FOLLOW集合如下：

	FIRST	FOLLOW
S	(, [, )	\$
X	[, )	\$
E	$\epsilon$	[, )
F	$\epsilon$	[, )
A	$\epsilon$	[, )

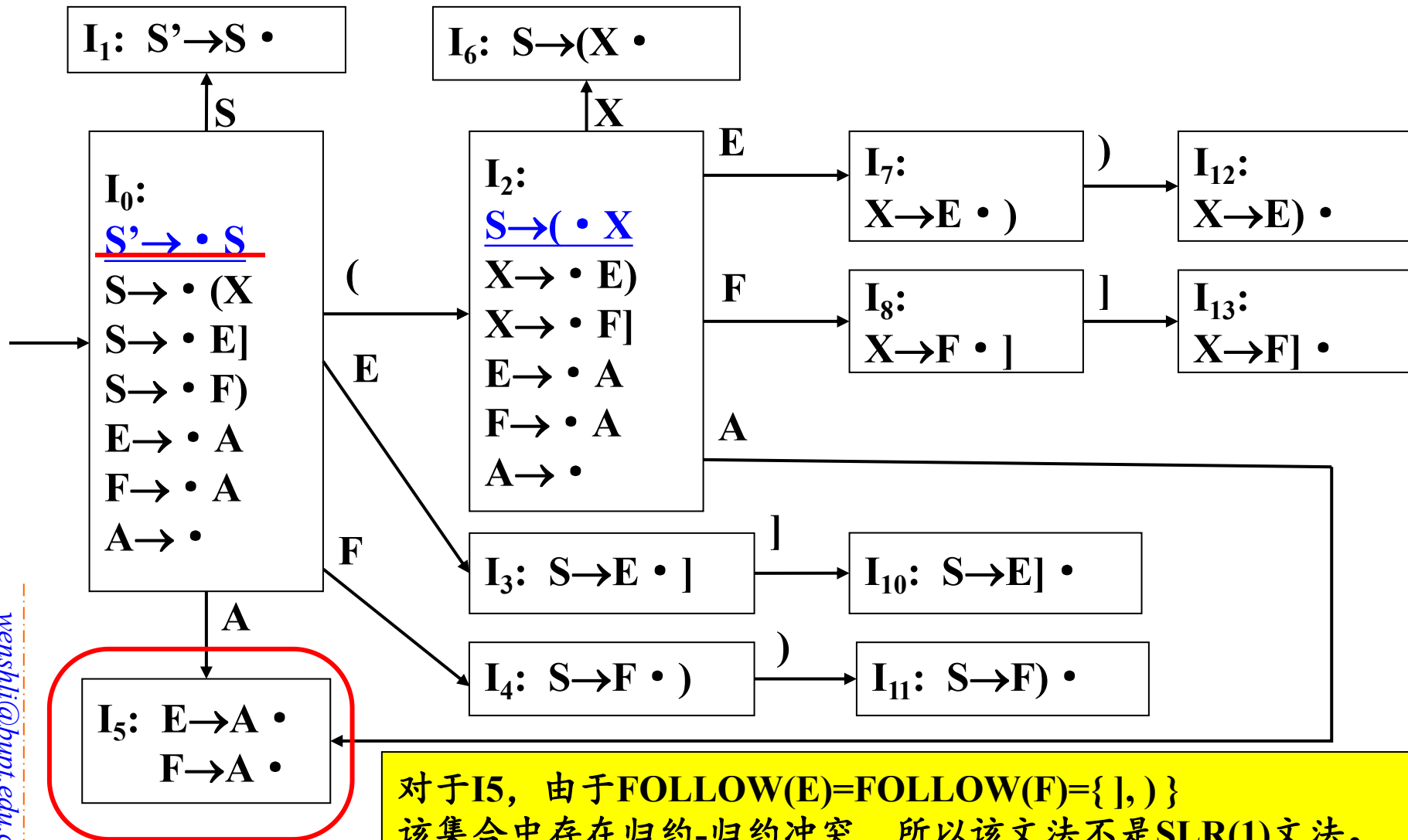
	(	)	[	\$
S	$S \rightarrow (X$	$S \rightarrow F)$	$S \rightarrow E]$	
X		$X \rightarrow E)$	$X \rightarrow F]$	
E		$E \rightarrow A$	$E \rightarrow A$	
F		$F \rightarrow A$	$F \rightarrow A$	
A		$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	

- 该文法的LL(1)分析表如右：
- 结论： LL(1)分析表中不含有多重定义的入口，所以该文法是LL(1)文法

# 证明该文法不是SLR(1)文法

$S \rightarrow (X \mid E \mid F) \quad X \rightarrow E \mid F$   
 $E \rightarrow A \quad F \rightarrow A \quad A \rightarrow \varepsilon$

- 构造该文法的LR(0)项目集规范族及识别所有活前缀的DFA



对于I5, 由于 $FOLLOW(E) = FOLLOW(F) = \{ [, ) \}$   
 该集合中存在归约-归约冲突, 所以该文法不是SLR(1)文法。

# LR(0)分析表和LR(0)文法

- 如果在执行上述算法的过程中，始终没有向前看任何输入符号，则构造的SLR分析表称为LR(0)分析表
- 具有LR(0)分析表的文法称为LR(0)文法。
- 一个文法是LR(0)文法，当且仅当该文法的每个活前缀的有效项目集中：
  - 要么所有元素都是移进-待约项目
  - 要么只含有唯一的归约项目
- 具有如下产生式的文法是一个LR(0)文法。

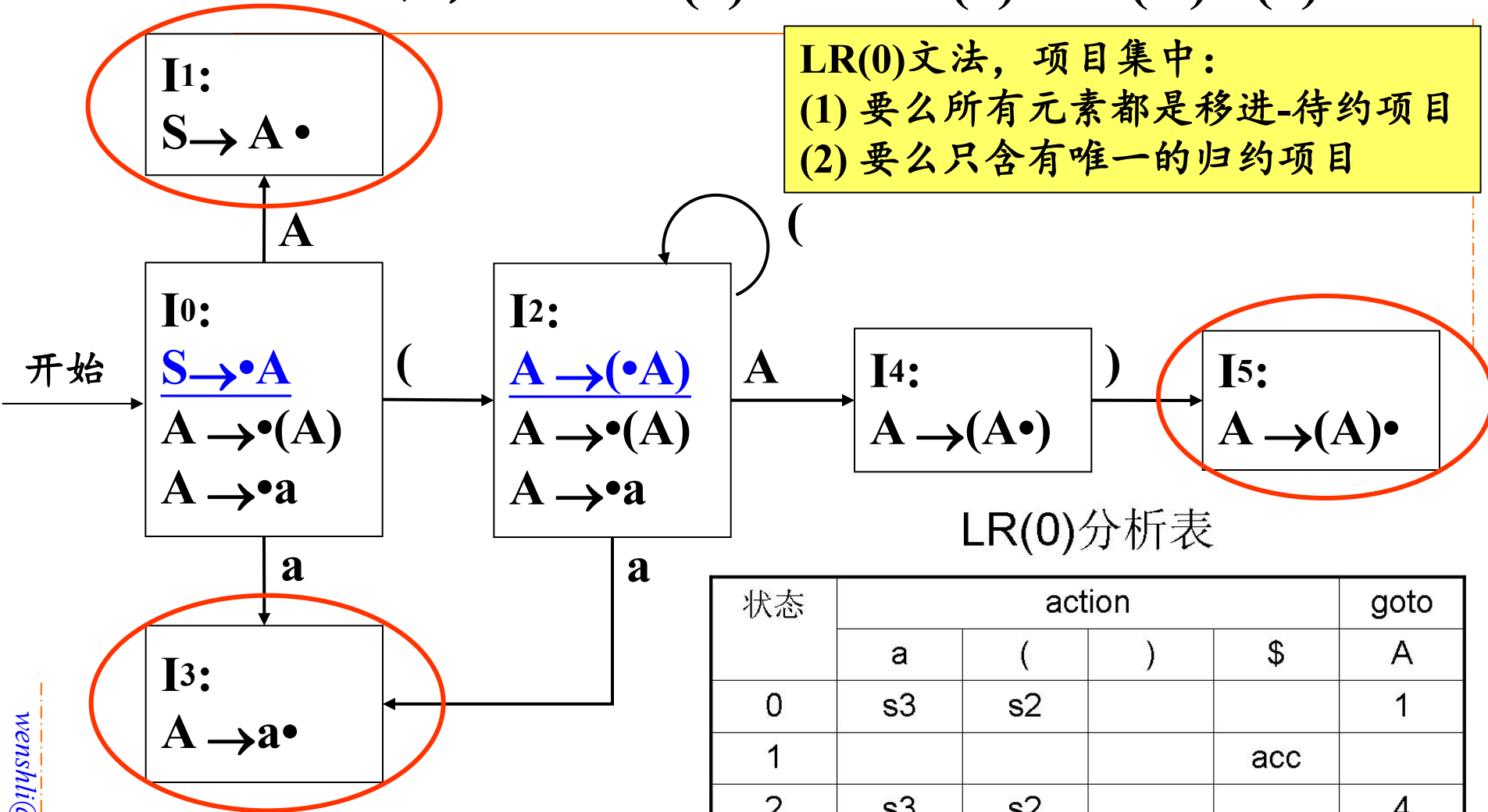
$A \rightarrow (A) | a$

练习：如何判断？

**判断过程：** 拓广文法：(0)  $S \rightarrow A$  (1)  $A \rightarrow (A)$  (2)  $A \rightarrow a$

LR(0)文法，项目集中：

- (1) 要么所有元素都是移进-待约项目
- (2) 要么只含有唯一的归约项目



LR(0)分析表

状态	action				goto
	a	(	)	\$	A
0	s3	s2			1
1				acc	
2	s3	s2			4
3	r2	r2	r2	r2	
4			s5		
5	r1	r1	r1	r1	

# 示例

- 文法4.3的拓广文法 $G'$ :

(0)  $E' \rightarrow E$       (1)  $E \rightarrow E+T$       (2)  $E \rightarrow T$       (3)  $T \rightarrow T * F$

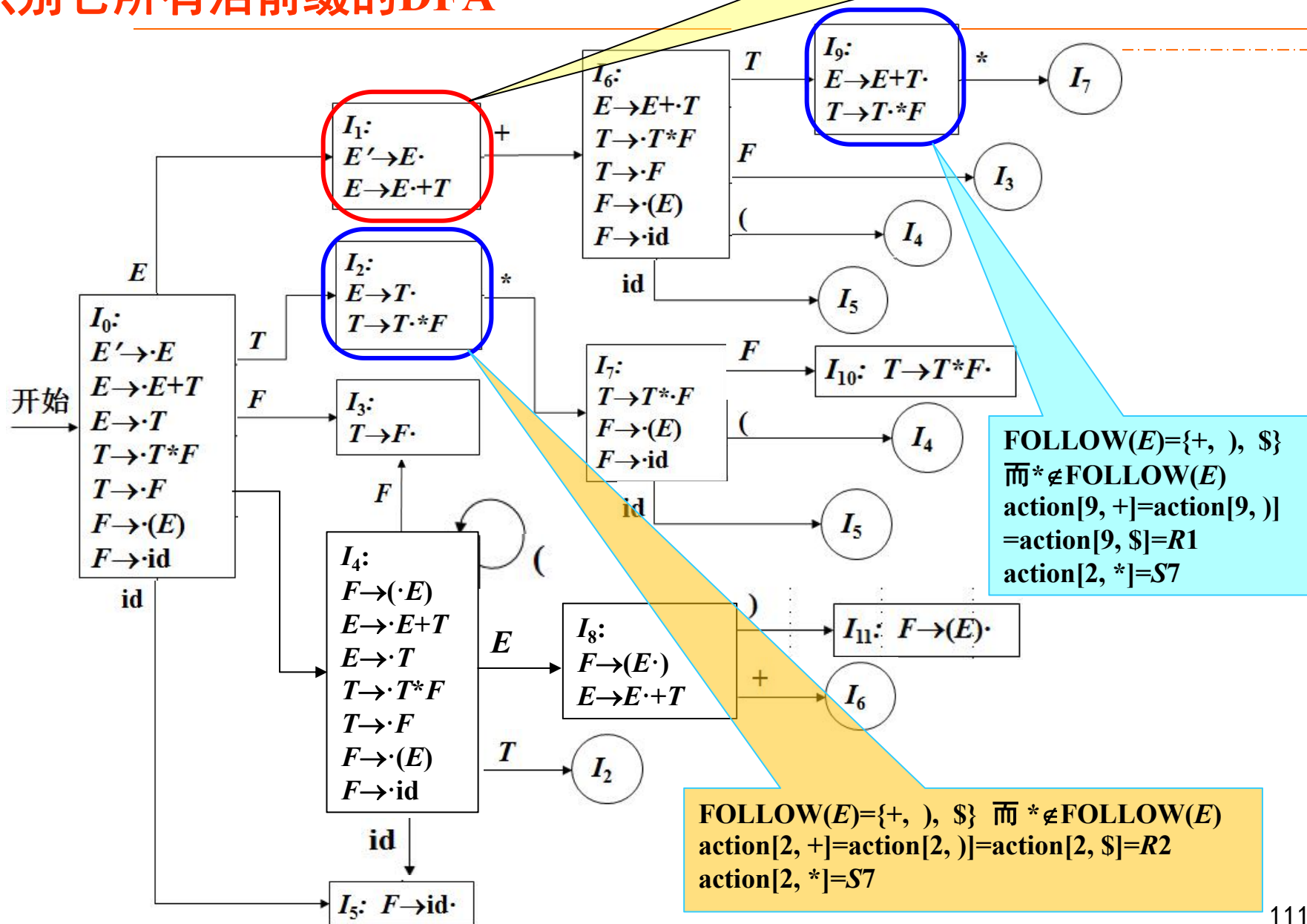
(4)  $T \rightarrow F$       (5)  $F \rightarrow (E)$       (6)  $F \rightarrow id$

- 判断文法4.3是LR(0)文法，还是SLR(1)文法？

- 构造 $G'$ 的LR(0)项目集规范族及识别它所有活前缀的DFA

# 文法4.3的 LR(0)项目集规范族及识别它所有活前缀的DFA

$\text{FOLLOW}(E')=\{\$, \text{而} + \notin \text{FOLLOW}(E')$   
 $\text{action}[1, \$]=\text{ACC}$ ,  $\text{action}[1, +]=S6$



# 定理

- 每一个SLR(1)文法都是无二义的文法，但并非无二义的文法都是SLR(1)文法。
- 例：有文法G具有如下产生式：

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow *R$

$L \rightarrow id$

$R \rightarrow L$  (文法4.7)

- 该文法无二义性，但不是SLR(1)文法。
- 拓广文法G'的产生式如下：

(0)  $S' \rightarrow S$       (1)  $S \rightarrow L = R$       (2)  $S \rightarrow R$

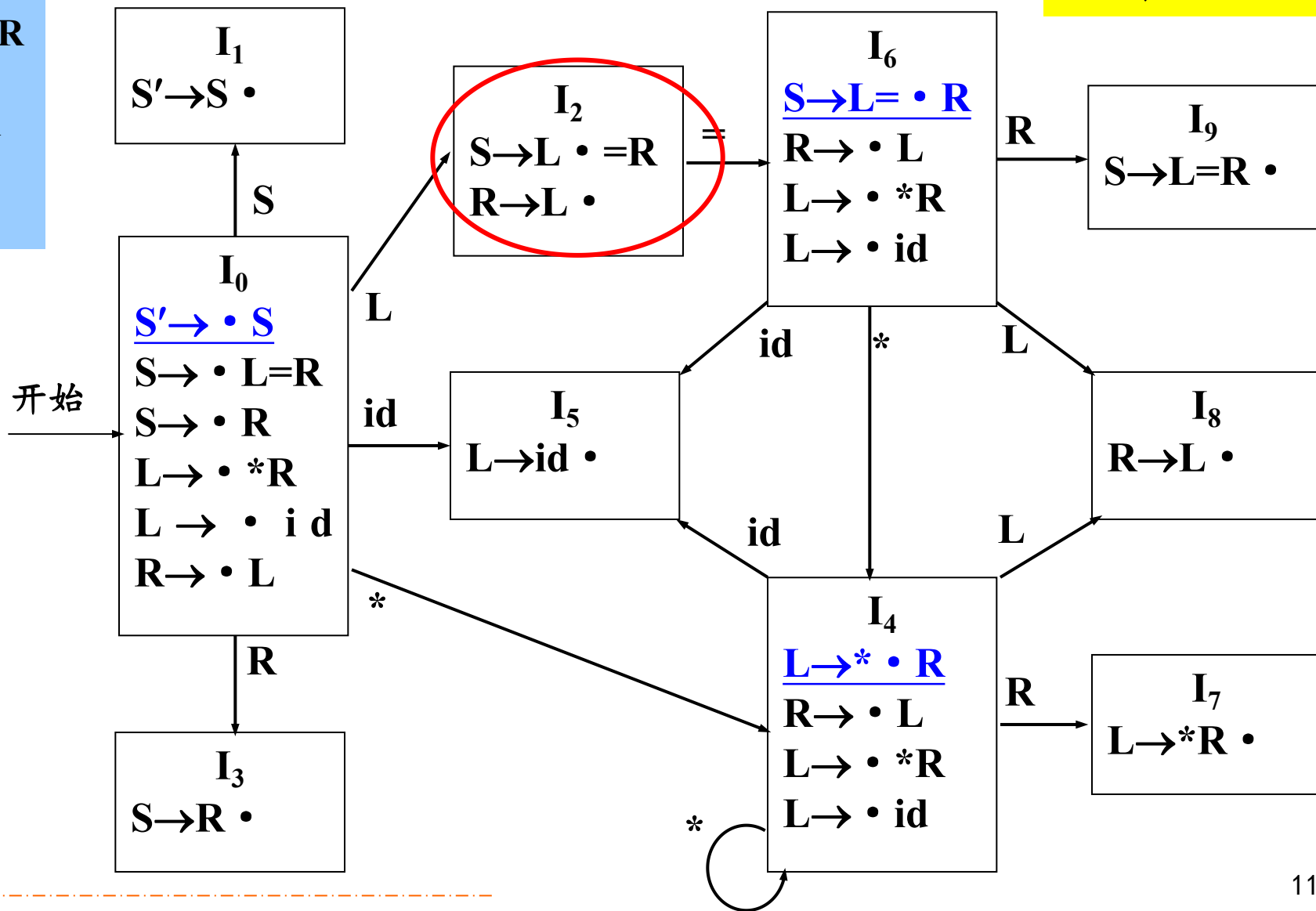
(3)  $L \rightarrow *R$       (4)  $L \rightarrow id$       (5)  $R \rightarrow L$



# 构造文法G'的LR(0)项目集规范族及识别活前缀的DFA

$S' \rightarrow S$   
 $S \rightarrow L=R$   
 $S \rightarrow R$   
 $L \rightarrow *R$   
 $L \rightarrow id$   
 $R \rightarrow L$

分析 \*id=id



# 构造文法G'的LR(0)项目集规范族及识别活前缀的DFA

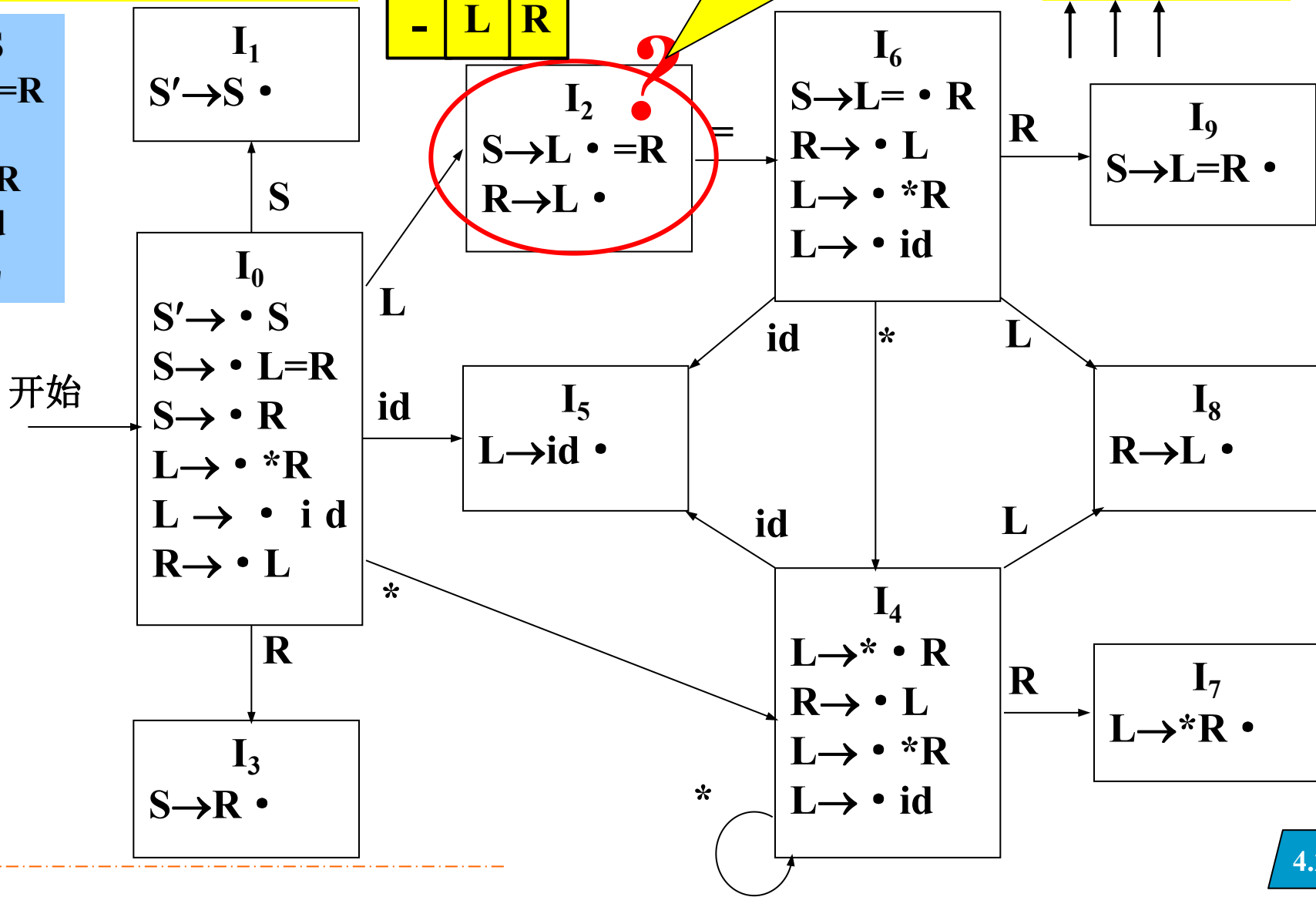
$FOLLOW(R) = \{=, \$\}$

$S' \rightarrow S$   
 $S \rightarrow L=R$   
 $S \rightarrow R$   
 $L \rightarrow *R$   
 $L \rightarrow id$   
 $R \rightarrow L$

0	2	7
-	L	R

移进-归约冲突

$* id = id \$$



## 4.4.3 LR(1)分析表的构造

- LR(k)项目： $[A \rightarrow \alpha \bullet \beta, a_1 a_2 \dots a_k]$ 
  - $A \rightarrow \alpha \bullet \beta$  是 LR(0) 项目
  - $a_i$  ( $i=1, 2, \dots, k$ ) 是终结符号
  - $a_1 a_2 \dots a_k$  称为该项目的向前看符号串
- 向前看符号串仅对归约项目  $[A \rightarrow \alpha \bullet, a_1 a_2 \dots a_k]$  起作用
- 归约项目  $[A \rightarrow \alpha \bullet, a_1 a_2 \dots a_k]$  意味着，当它所属项目集对应的状态在栈顶，且后续的  $k$  个输入符号为  $a_1 a_2 \dots a_k$  时，才允许把栈顶的语法符号串  $\alpha$  归约为  $A$ 。

## 定义4.13: LR(1)有效项目

- 称一个LR(1)项目  $[A \rightarrow \alpha \bullet \beta, a]$  对活前缀  $\gamma = \delta \alpha$  是有效的, 如果存在一个规范推导:

$$S \xRightarrow{*} \delta A \omega \Rightarrow \delta \alpha \beta \omega$$

其中 $\omega$ 的第一个符号为  $a$ , 或者 $\omega = \varepsilon$ , 且 $a = \$$ 。

- 推广: 若项目  $[A \rightarrow \alpha \bullet B \beta, a]$  对活前缀  $\gamma = \delta \alpha$  是有效的, 并且有产生式  $B \rightarrow \eta$ , 则对任何  $b \in \text{FIRST}(\beta a)$ , 项目  $[B \rightarrow \bullet \eta, b]$  对活前缀  $\gamma = \delta \alpha$  也是有效的。

- $b$ 或是从 $\beta$ 推出的开头终结符号, 或者 $\beta \Rightarrow \varepsilon$ , 而 $b = a$ 。

$$S \xRightarrow{*} \delta A \omega \Rightarrow \delta \alpha B \beta \omega \xRightarrow{*} \delta \alpha B \lambda \Rightarrow \delta \alpha \eta \lambda$$

- LR(1)项目  $[S' \rightarrow \bullet S, \$]$  对活前缀  $\varepsilon$  是有效的。

# LR(1)有效项目集和 LR(1)项目集规范族

- 文法G的某个活前缀 $\gamma$ 的所有LR(1)有效项目组成的集合称为 $\gamma$ 的LR(1)有效项目集。
- 文法G的所有LR(1)有效项目集组成的集合称为G的LR(1)项目集规范族。

## 定义4.14：闭包 (closure)

设 $I$ 是文法 $G$ 的一个LR(1)项目集， $\text{closure}(I)$ 是从 $I$ 出发，用下面的方法构造的项目集。

- (1)  $I$ 中的每一个项目都属于 $\text{closure}(I)$ ;
- (2) 若项目 $[A \rightarrow \alpha \bullet B \beta, a]$ 属于 $\text{closure}(I)$ ，且 $G$ 有产生式 $B \rightarrow \eta$ ，则  
对任何终结符号  $b \in \text{FIRST}(\beta a)$ ，  
若项目 $[B \rightarrow \bullet \eta, b]$ 不属于集合 $\text{closure}(I)$ ，  
则将它加入 $\text{closure}(I)$ ;
- (3) 重复规则(2)，直到 $\text{closure}(I)$ 不再增大为止。

# 算法4.7 closure(I)的构造过程

输入：项目集合I

输出：集合 $J = \text{closure}(I)$

方法：

$J = I;$

do {

$J_{\text{new}} = J;$

for (  $J_{\text{new}}$  中的每一个项目  $[A \rightarrow \alpha \bullet B \beta, a]$  和  
文法G的每个产生式  $B \rightarrow \eta$  )

for (FIRST( $\beta a$ )中的每一个终结符号b)

if ( $[B \rightarrow \bullet \eta, b] \notin J$ ) 把  $[B \rightarrow \bullet \eta, b]$  加入J;

} while ( $J_{\text{new}} \neq J$ );

## 定义4.15：转移函数go

若 $I$ 是文法 $G$ 的一个LR(1)项目集， $X$ 是一个文法符号，定义：

$$\text{go}(I, X) = \text{closure}(J)$$

其中： $J = \{ [A \rightarrow \alpha X \bullet \beta, a] \mid \text{当 } [A \rightarrow \alpha \bullet X \beta, a] \text{ 属于 } I \text{ 时} \}$

项目 $[A \rightarrow \alpha X \bullet \beta, a]$ 称为 $[A \rightarrow \alpha \bullet X \beta, a]$ 的后继。

直观含义：

若 $I$ 是某个活前缀 $\gamma$ 的有效项目集，

则 $\text{go}(I, X)$ 便是对活前缀 $\gamma X$ 的有效项目集。



# 算法4.8 构造文法G的LR(1)项目集规范族

输入：拓广文法 $G'$

输出： $G'$ 的LR(1)项目集规范族

方法：

$C = \{ \text{closure}(\{ [S' \rightarrow \bullet S, \$] \}) \};$

do

for ( $C$ 中的每一个项目集 $I$ 和每一个文法符号 $X$ )

if ( $\text{go}(I, X)$ 不为空, 且不在 $C$ 中)

把  $\text{go}(I, X)$  加入 $C$ 中;

while (没有新项目集加入 $C$ 中).

# 示例

构造如下文法的LR(1)项目集规范族:

(1)  $S \rightarrow CC$  (2)  $C \rightarrow cC$  (3)  $C \rightarrow d$  (文法4.8)

## ■ 拓广文法:

(0)  $S' \rightarrow S$  (1)  $S \rightarrow CC$  (2)  $C \rightarrow cC$  (3)  $C \rightarrow d$

## ■ 根据算法4.6 构造其LR(1)项目集规范族。

$I_0 = \text{closure}(\{[S' \rightarrow \bullet S, \$]\})$

$= \{[S' \rightarrow \bullet S, \$] \ [S \rightarrow \bullet CC, \$] \ [C \rightarrow \bullet cC, c/d] \ [C \rightarrow \bullet d, c/d]\}$

$I_1 = \text{go}(I_0, S) = \text{closure}(\{[S' \rightarrow S \bullet, \$]\}) = \{[S' \rightarrow S \bullet, \$]\}$

$I_2 = \text{go}(I_0, C) = \text{closure}(\{[S \rightarrow C \bullet C, \$]\})$

$= \{[S \rightarrow C \bullet C, \$] \ [C \rightarrow \bullet cC, \$] \ [C \rightarrow \bullet d, \$]\}$

$I_3 = \text{go}(I_0, c) = \text{closure}(\{[C \rightarrow c \bullet C, c/d]\})$

$= \{[C \rightarrow c \bullet C, c/d] \ [C \rightarrow \bullet cC, c/d] \ [C \rightarrow \bullet d, c/d]\}$

$I_4 = \text{go}(I_0, d) = \text{closure}(\{[C \rightarrow d \bullet, c/d]\}) = \{[C \rightarrow d \bullet, c/d]\}$

# 示例

$$I_5 = \text{go}(I_2, C) = \text{closure}(\{[S \rightarrow CC\bullet, \$]\}) = \{[S \rightarrow CC\bullet, \$]\}$$

$$I_6 = \text{go}(I_2, c) = \text{closure}(\{[C \rightarrow c\bullet C, \$]\}) \\ = \{[C \rightarrow c\bullet C, \$], [C \rightarrow \bullet cC, \$], [C \rightarrow \bullet d, \$]\}$$

$$I_7 = \text{go}(I_2, d) = \text{closure}(\{[C \rightarrow d\bullet, \$]\}) = \{[C \rightarrow d\bullet, \$]\}$$

$$I_8 = \text{go}(I_3, C) = \text{closure}(\{[C \rightarrow cC\bullet, c/d]\}) = \{[C \rightarrow cC\bullet, c/d]\}$$

$$\text{go}(I_3, c) = \text{closure}(\{[C \rightarrow c\bullet C, c/d]\}) = I_3$$

$$\text{go}(I_3, d) = \text{closure}(\{[C \rightarrow d\bullet, c/d]\}) = I_4$$

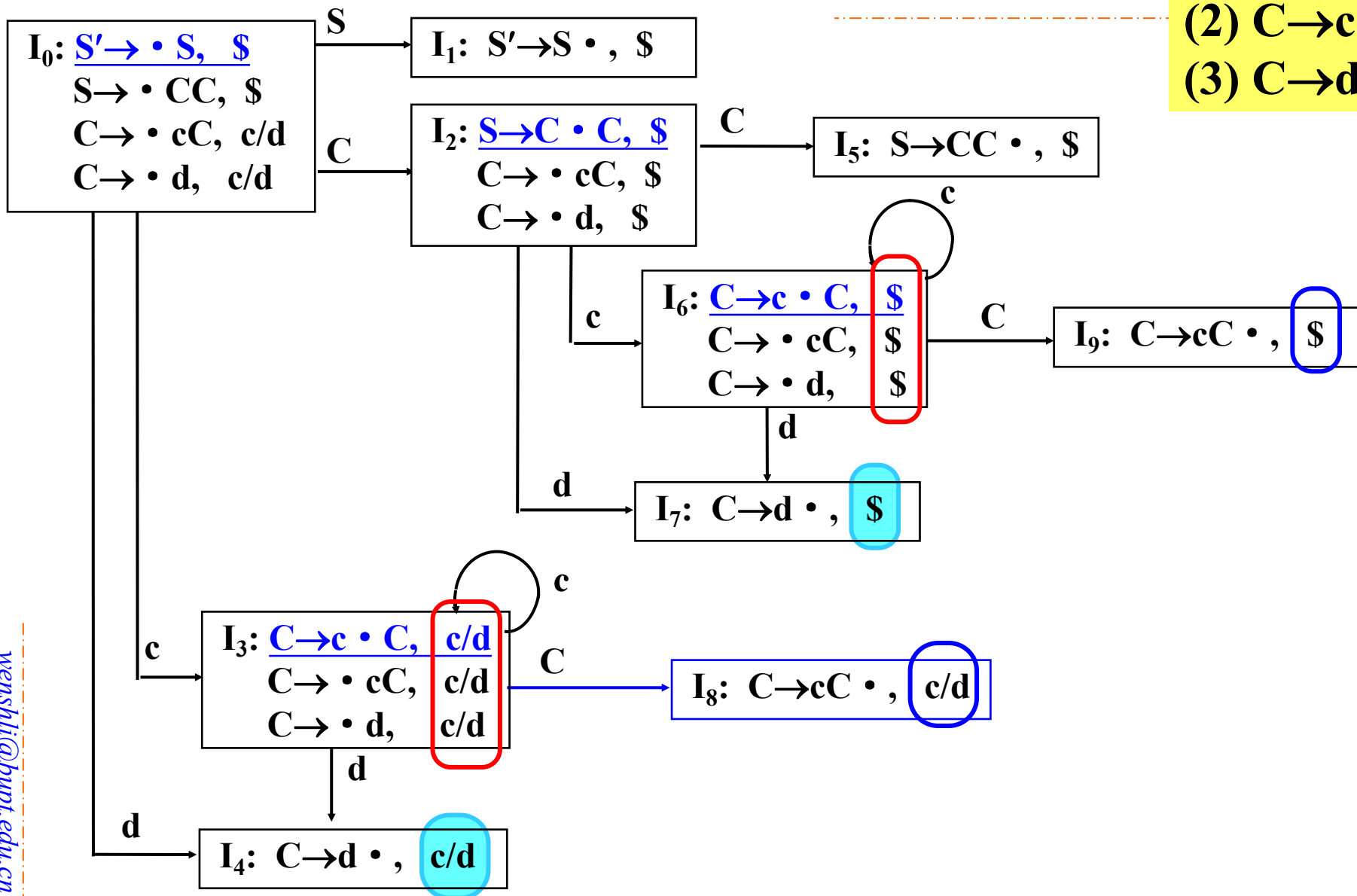
$$I_9 = \text{go}(I_6, C) = \text{closure}(\{[C \rightarrow cC\bullet, \$]\}) = \{[C \rightarrow cC\bullet, \$]\}$$

$$\text{go}(I_6, c) = \text{closure}(\{[C \rightarrow c\bullet C, \$]\}) = I_6$$

$$\text{go}(I_6, d) = \text{closure}(\{[C \rightarrow d\bullet, \$]\}) = I_7$$

# 识别文法4.8所有活前缀的DFA

- (0)  $S' \rightarrow S$
- (1)  $S \rightarrow CC$
- (2)  $C \rightarrow cC$
- (3)  $C \rightarrow d$



# 算法4.9 构造LR(1)分析表

输入：拓广文法 $G'$       输出：文法 $G'$ 的分析表

方法如下：

1.构造文法 $G'$ 的LR(1)项目集规范族 $C=\{I_0, I_1, \dots, I_n\}$

2.对于状态 $i$ （代表项目集 $I_i$ ），分析动作如下：

a) 若 $[A \rightarrow \alpha \bullet a \beta, b] \in I_i$ ，且 $go(I_i, a) = I_j$ ，则置  $action[i, a] = Sj$

b) 若 $[A \rightarrow \alpha \bullet, a] \in I_i$ ，且 $A \neq S'$ ，则置  $action[i, a] = Rj$

c) 若 $[S' \rightarrow S \bullet, \$] \in I_i$ ，则置  $action[i, \$] = ACC$

3.若对非终结符号 $A$ ，有 $go(I_i, A) = I_j$ ，则置  $goto[i, A] = j$

4.凡是不能用上述规则填入信息的空白表项，均置上出错标志  
**error**。

5.分析程序的初态是包括 $[S' \rightarrow \bullet S, \$]$ 的有效项目集所对应的状态。

# 例：构造文法4.8的LR(1)分析表

## ■ 考察 $I_0$ :

- $[S' \rightarrow \bullet S, \$]$  且  $go(I_0, S)=I_1$ , 故:  $goto[0, S]=1$
- $[S \rightarrow \bullet CC, \$]$   $go(I_0, C)=I_2$   $goto[0, C]=2$
- $[C \rightarrow \bullet cC, c/d]$   $go(I_0, c)=I_3$   $action[0, c]=S3$
- $[C \rightarrow \bullet d, c/d]$   $go(I_0, d)=I_4$   $action[0, d]=S4$

## ■ 考察 $I_1$ : 由于 $[S' \rightarrow S\bullet, \$]$ 故: $action[1, \$]=ACC$

## ■ 考察 $I_2$ :

- $[S \rightarrow C\bullet C, \$]$  且  $go(I_2, C)=I_5$  故:  $goto[2, C]=5$
- $[C \rightarrow \bullet cC, \$]$   $go(I_2, c)=I_6$   $action[2, c]=S6$
- $[C \rightarrow \bullet d, \$]$   $go(I_2, d)=I_7$   $action[2, d]=S7$

## ■ 考察 $I_4$ :

- $[C \rightarrow d\bullet, c/d]$  故  $action[4, c]=action[4, d]=R3$

# 例：构造文法4.8的LR(1)分析表

(0)  $S' \rightarrow S$

(1)  $S \rightarrow CC$

(2)  $C \rightarrow cC$

(3)  $C \rightarrow d$

$I_3: \underline{C \rightarrow c \cdot C}, \text{ c/d}$   
 $C \rightarrow \cdot cC, \text{ c/d}$   
 $C \rightarrow \cdot d, \text{ c/d}$

$I_6: \underline{C \rightarrow c \cdot C}, \$$   
 $C \rightarrow \cdot cC, \$$   
 $C \rightarrow \cdot d, \$$

$I_0: \underline{S' \rightarrow \cdot S}, \$$   
 $S \rightarrow \cdot CC, \$$   
 $C \rightarrow \cdot cC, \text{ c/d}$   
 $C \rightarrow \cdot d, \text{ c/d}$

$I_1: S' \rightarrow S \cdot, \$$

$I_2: \underline{S \rightarrow C \cdot C}, \$$   
 $C \rightarrow \cdot cC, \$$   
 $C \rightarrow \cdot d, \$$

$I_4: C \rightarrow d \cdot, \text{ c/d}$

$I_5: S \rightarrow CC \cdot, \$$

$I_7: C \rightarrow d \cdot, \$$

$I_8: C \rightarrow cC \cdot, \text{ c/d}$

$I_9: C \rightarrow cC \cdot, \$$

状态	action			goto	
	c	d	\$	S	C
0	S3	S4		1	2
1			ACC		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

# 示例

## ■ 试构造文法4.7的LR(1)分析表

(1)  $S \rightarrow L=R$    (2)  $S \rightarrow R$    (3)  $L \rightarrow *R$   
(4)  $L \rightarrow id$    (5)  $R \rightarrow L$

## ■ 拓广文法 $G'$

(0)  $S' \rightarrow S$    (1)  $S \rightarrow L=R$    (2)  $S \rightarrow R$   
(3)  $L \rightarrow *R$    (4)  $L \rightarrow id$    (5)  $R \rightarrow L$

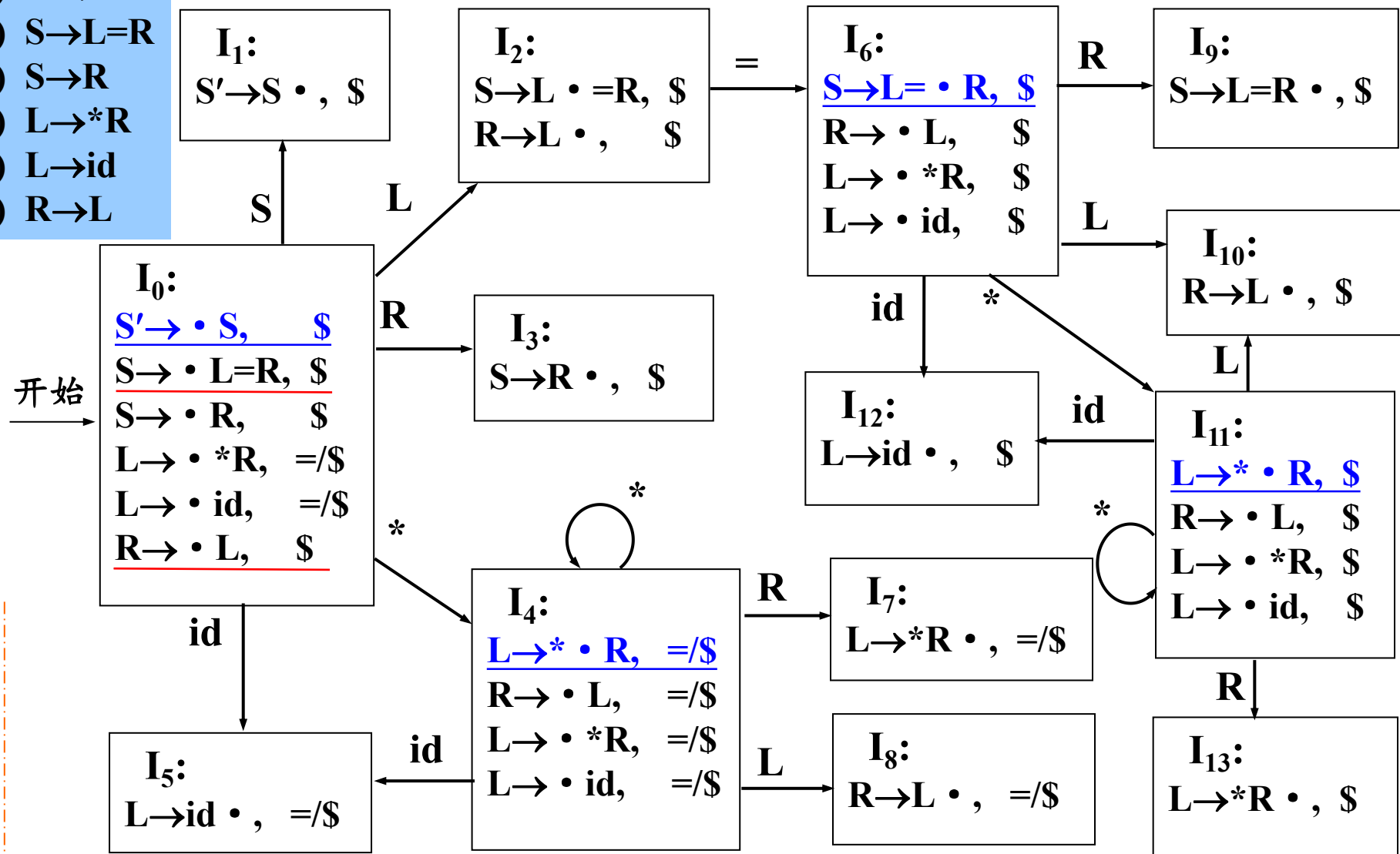
## ■ 构造文法 $G'$ 的LR(1)项目集规范族及识别所有活前缀的DFA



# 示例

练习：  
分析 \*id=id

- 0)  $S' \rightarrow S$
- 1)  $S \rightarrow L=R$
- 2)  $S \rightarrow R$
- 3)  $L \rightarrow *R$
- 4)  $L \rightarrow id$
- 5)  $R \rightarrow L$



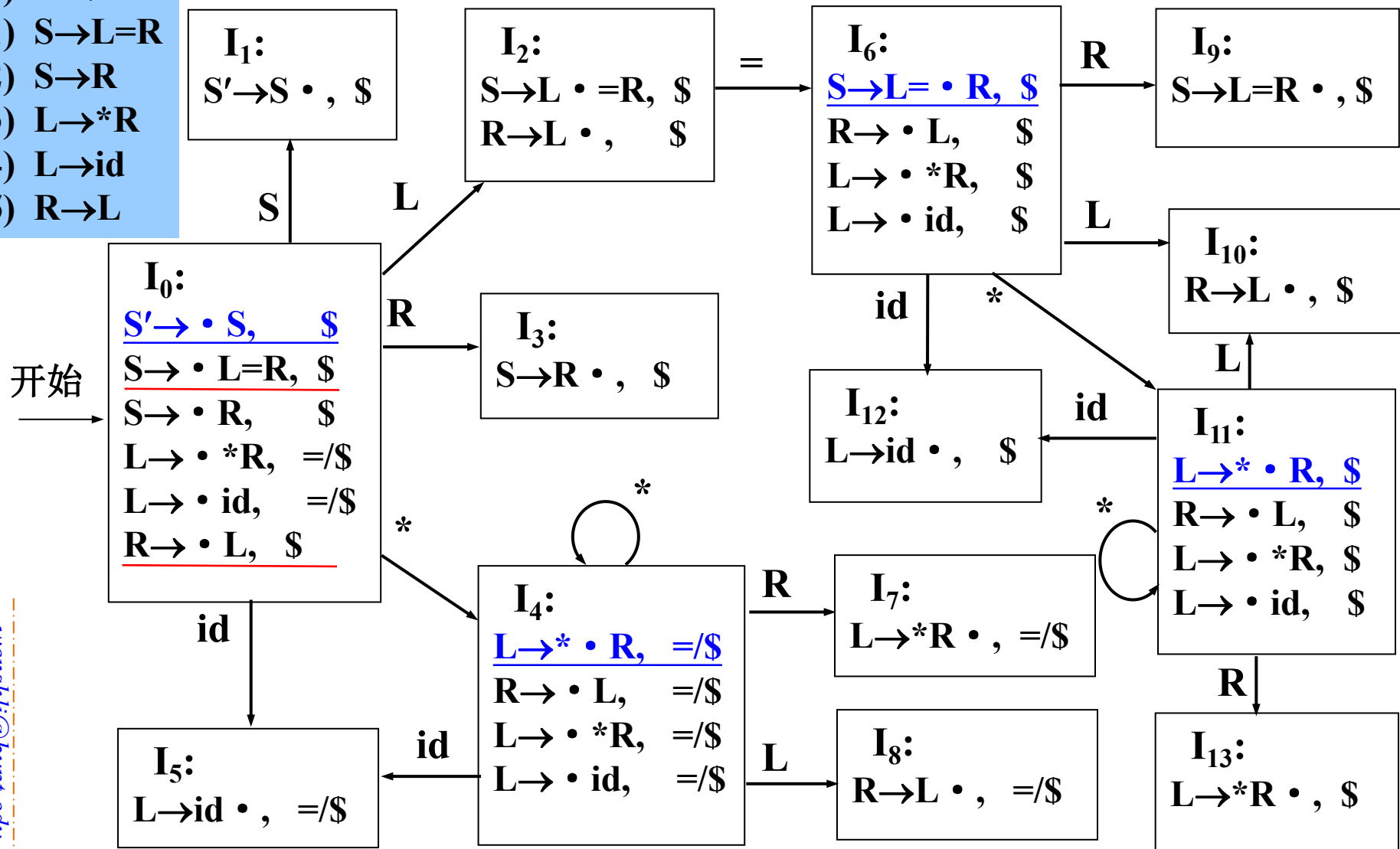
# 分析过程:

\* id = id \$

0	1	6	9
-	S	=	R

*Accept!*

- 0)  $S' \rightarrow S$
- 1)  $S \rightarrow L=R$
- 2)  $S \rightarrow R$
- 3)  $L \rightarrow *R$
- 4)  $L \rightarrow id$
- 5)  $R \rightarrow L$



# 文法4.7的LR(1)分析表

$I_0$ :

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

$S \rightarrow \cdot L=R, \$$

$S \rightarrow \cdot R, \$$

$L \rightarrow \cdot *R, =/\$$

$L \rightarrow \cdot id, =/\$$

$R \rightarrow \cdot L, \$$

$I_1$ :

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

$I_2$ :

$S \rightarrow L \cdot =R, \$$

$R \rightarrow L \cdot, \$$

$I_4$ :

$L \rightarrow * \cdot R, =/\$$

$R \rightarrow \cdot L, =/\$$

$L \rightarrow \cdot *R, =/\$$

$L \rightarrow \cdot id, =/\$$

状态	action				goto		
	=	*	id	\$	S	L	R
0		S4	S5		1	2	3
1				ACC			
2	S6			R5			
3				R2			
4		S4	S5			8	7
5	R4			R4			
6		S11	S12			10	9
7	R3			R3			
8	R5			R5			
9				R1			
10				R5			
11		S11	S12			10	13
12				R4			
13				R3			

## 课堂练习3

说明下面的文法是LR(1)文法，但不是SLR(1)文法。

$$X \rightarrow Ma \mid bMc \mid dc \mid bda$$
$$M \rightarrow d$$

# 解答:

首先, 拓广文法

(0)  $S \rightarrow X$

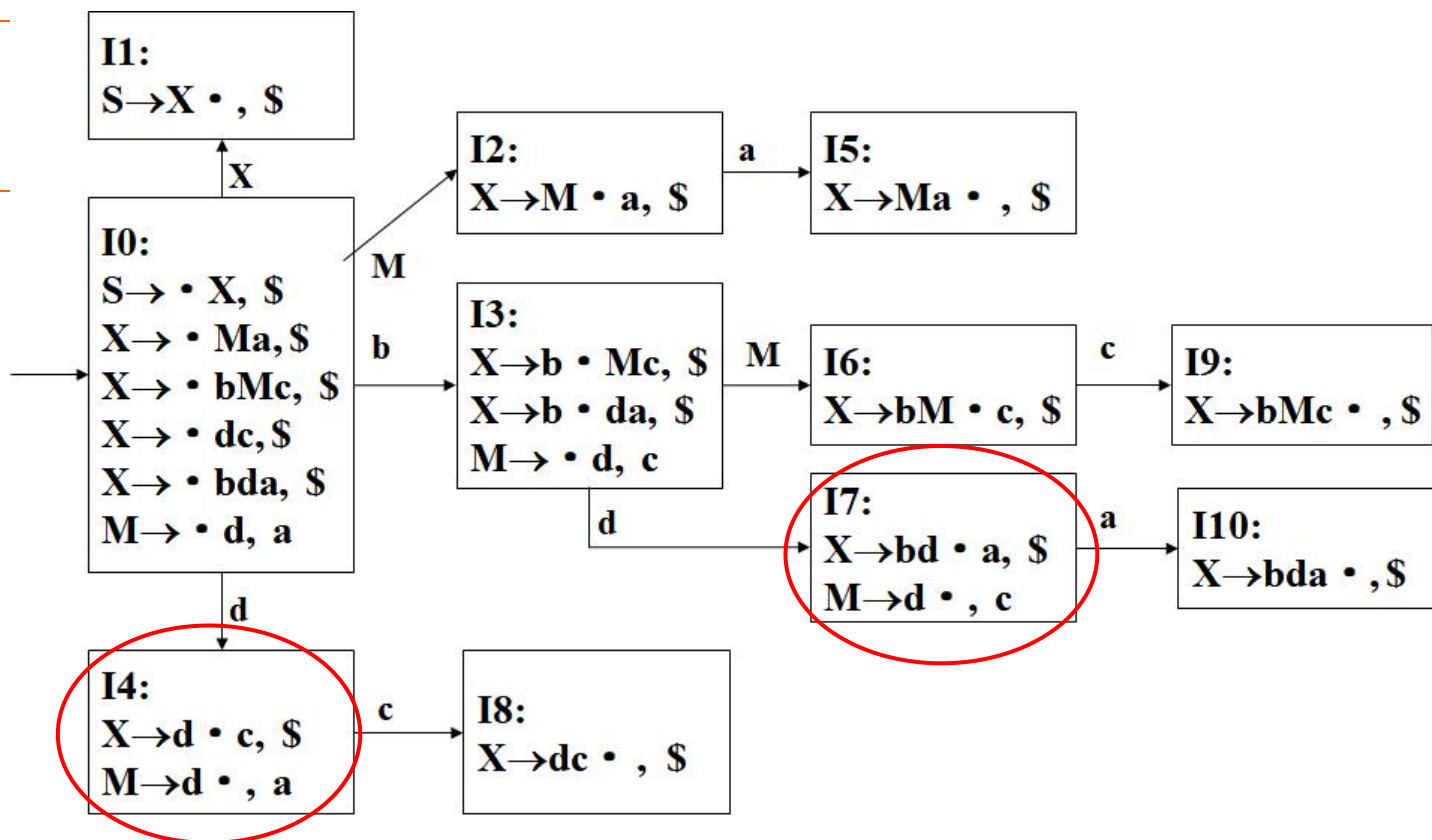
(1)  $X \rightarrow Ma$

(2)  $X \rightarrow bMc$

(3)  $X \rightarrow dc$

(4)  $X \rightarrow bda$

(5)  $M \rightarrow d$



其次, 构造文法的LR(1)项目集规范族及识别其所有活前缀的DFA。

判断该文法是LR(1)文法: (1)集合I0、I3中没有归约项目, 所以, 不存在冲突;

(2)集合I1、I2、I5、I6、I8、I9、I10 各只有一个归约项目, 所以这些集合中没有冲突;

(3)集合I4和I7中既有移进项目又有归约项目, 但是归约符号和移进符号不同, 所以也没有冲突。

结论: 是LR(1)

然后, 构造文法的LR(0)项目集规范族及识别其所有活前缀的DFA。

$FOLLOW(S) = \{ \$ \}$   
 $FOLLOW(X) = \{ \$ \}$   
 $FOLLOW(M) = \{ a, c \}$

I<sub>4</sub>、I<sub>7</sub>中存在移进-归约冲突,  $FOLLOW(M) = \{ a, c \}$   
这种冲突用SLR(1)方法无法解决,  
所以该文法不是SLR(1)文法。

## 4.4.4 LALR(1)分析表的构造

### ■描述LR(1)项目集特征的两个定义

#### ■ 定义4.16: 同心集

如果两个LR(1)项目集去掉搜索符号之后是相同的, 则称这两个项目集**具有相同的心 (core)**, 即这两个项目集是**同心集**。

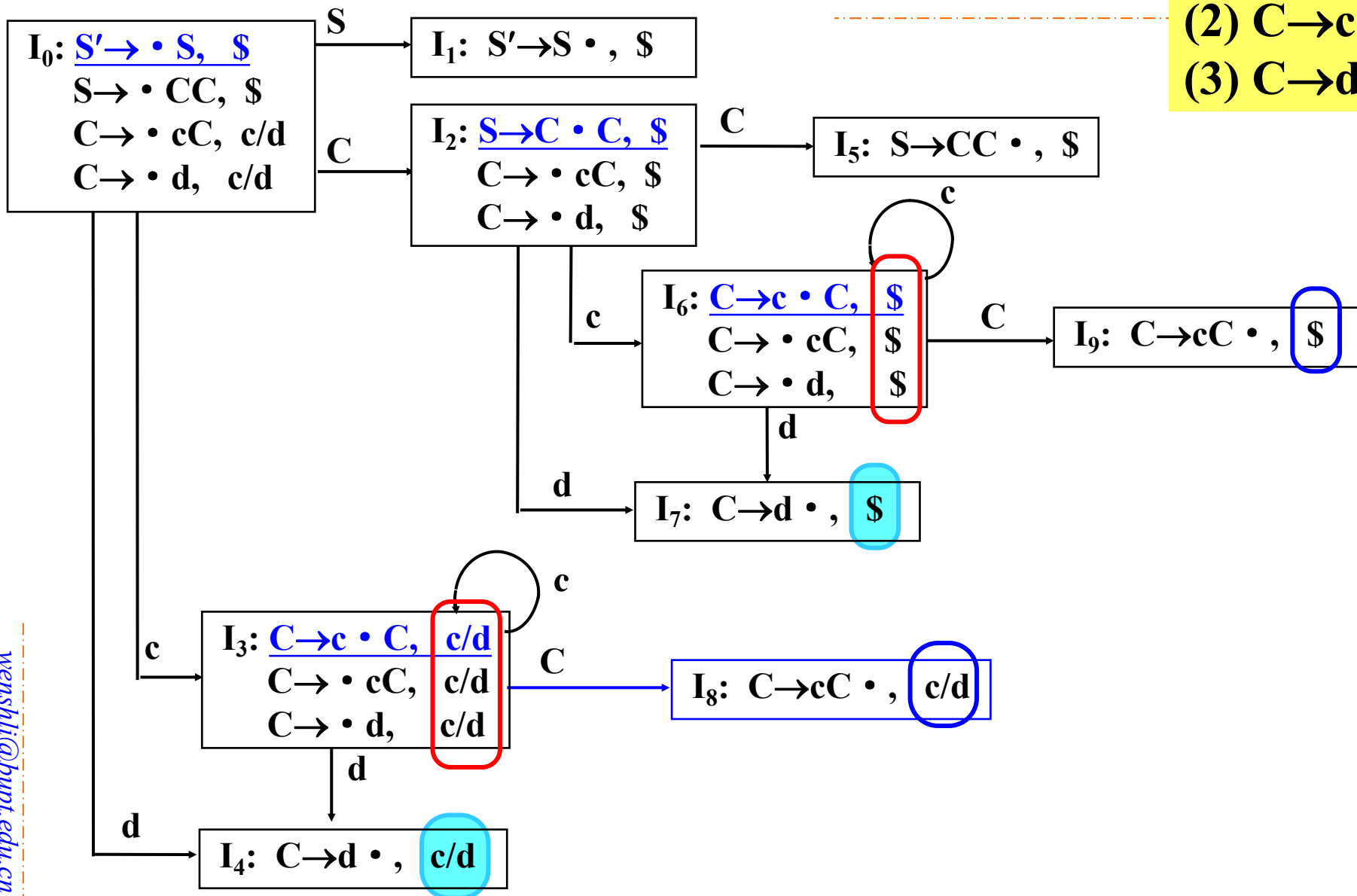
#### ■ 定义4.17: 项目集的核

除去初态项目集外, 一个**项目集的核 (kernel)**是由该项目集中那些圆点不在最左边的项目组成。

LR(1)初态项目集的核中有且只有项目  $[S' \rightarrow \bullet S, \$]$ 。

# 识别文法4.8所有活前缀的DFA

- (0)  $S' \rightarrow S$
- (1)  $S \rightarrow CC$
- (2)  $C \rightarrow cC$
- (3)  $C \rightarrow d$



# 构造LALR(1)分析表的基本思想

- 合并LR(1)项目集规范族中的同心集，以减少分析表的状态数。
- 用核代替项目集，以减少项目集所需的存储空间
- $go(I, X)$ 仅仅依赖于I的心，因此LR(1)项目集合并后的转移函数可以通过 $go(I, X)$ 自身的合并得到。
- 同心集的合并，可能导致归约-归约的冲突，但不会产生新的移进-归约冲突



# 同心集的合并不会引进新的移进-归约冲突

如果合并后的项目集中存在移进-归约冲突，则意味着：  
项目  $[A \rightarrow \alpha \bullet, a]$  和  $[B \rightarrow \beta \bullet a \gamma, b]$  处于合并后的同一项目集中。

合并前必存在某个  $c$ ，使得  $[A \rightarrow \alpha \bullet, a]$  和  $[B \rightarrow \beta \bullet a \gamma, c]$  同处于某个项目集中。

说明，原来的LR(1)项目集中已经存在移进-归约冲突。

# 同心集的合并可能导致归约-归约冲突

例：有文法G：

$$S \rightarrow aAd \mid bBd \mid aBe \mid bAe$$
$$A \rightarrow c$$
$$B \rightarrow c \quad (\text{文法4.9})$$

## ■ 拓广文法G'

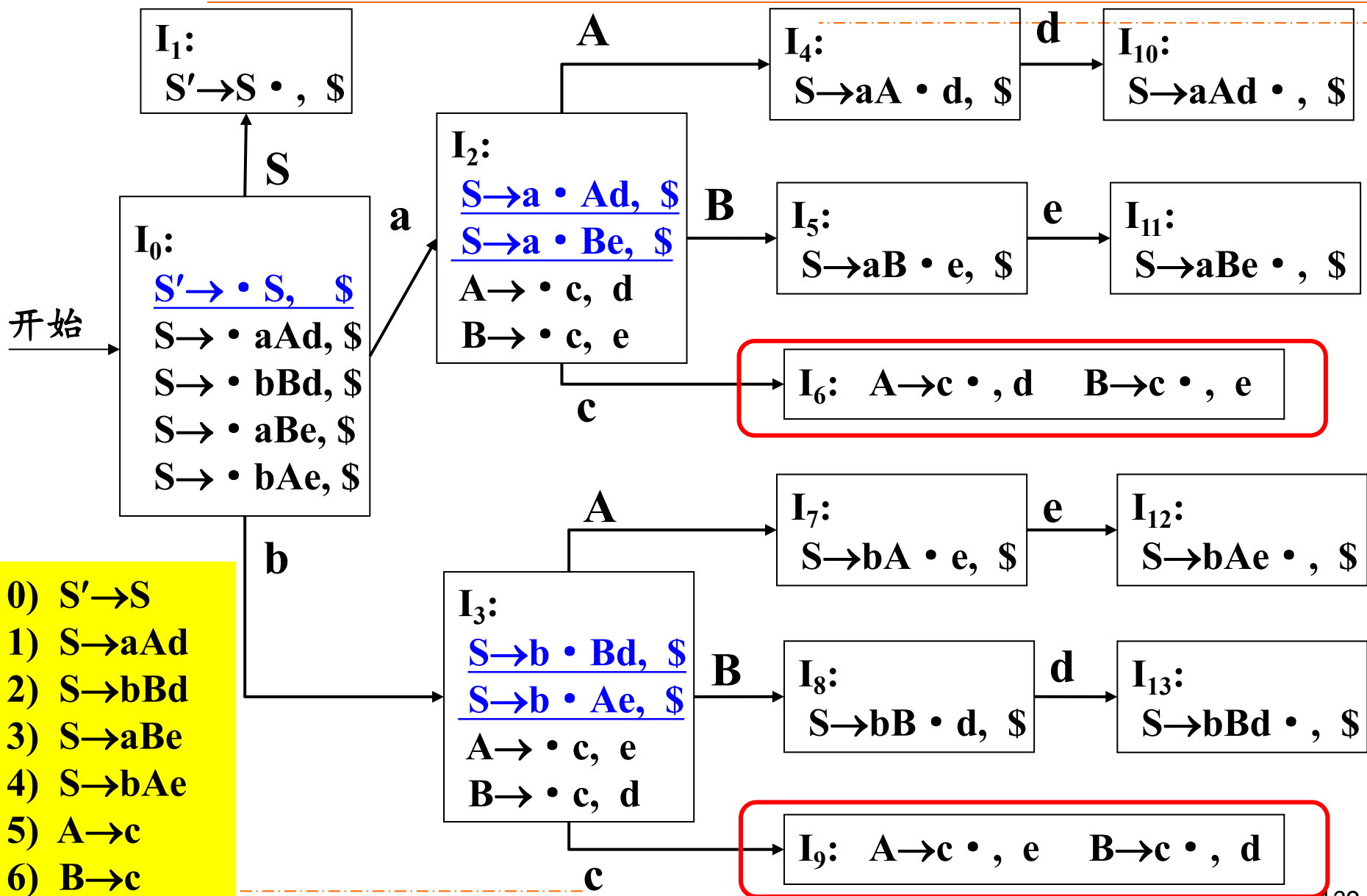
$$(0) S' \rightarrow S$$
$$(1) S \rightarrow aAd \quad (2) S \rightarrow bBd \quad (3) S \rightarrow aBe \quad (4) S \rightarrow bAe$$
$$(5) A \rightarrow c \quad (6) B \rightarrow c$$

## ■ 构造识别文法G'的所有活前缀的DFA

$$I_0 = \{ [S' \rightarrow \bullet S, \$]$$
$$[S \rightarrow \bullet aAd, \$] \quad [S \rightarrow \bullet bBd, \$] \quad [S \rightarrow \bullet aBe, \$] \quad [S \rightarrow \bullet bAe, \$] \}$$

# 示例

## 练习：分析 acd、bce



# 文法4.9的LR(1)分析表

状态	action						goto		
	a	b	c	d	e	\$	S	A	B
0	S2	S3					1		
1						ACC			
2			S6					4	5
3			S9					7	8
4				S10					
5					S11				
6				R5	R6				
7					S12				
8				S13					
9				R6	R5				
10						R1			
11						R3			
12						R4			
13						R2			

# 合并同心集

## ■ LR(1)项目集规范族中

- 活前缀ac的有效项目集是 $I_6$
- 活前缀bc的有效项目集是 $I_9$
- 这两个项目集都不含冲突项目，且是同心集。

## ■ 它们合并后得到的集合为：

$\{ [A \rightarrow c\bullet, d/e] \ [B \rightarrow c\bullet, d/e] \}$

## ■ 含有归约-归约冲突。

# LALR(1)分析表的构造

- 首先构造LR(1)项目集规范族
- 检查LR(1)项目集规范族
  - 含有冲突，则该文法不是LR(1)文法。
  - 不存在冲突，该文法是LR(1)文法，检查是否存在同心集
    - 不存在，则LR(1)项目集规范族即LALR(1)项目集规范族；
    - 存在，合并同心集；
    - 检查合并后的项目集是否存在冲突
      - 有冲突，则该文法不是LALR文法。
      - 无冲突，则该文法是LALR(1)文法，合并同心集得到的项目集规范族为LALR(1)项目集规范族。
- 根据LALR(1)项目集规范族构造分析表。

## 算法4.10 构造LALR(1)分析表

输入：一个拓广文法 $G'$       输出：文法 $G'$ 的LALR(1)分析表

方法：

- 1.构造文法 $G'$ 的LR(1)项目集规范族  $C=\{I_0, I_1, \dots, I_n\}$ 。
- 2.合并 $C$ 中的同心集，得到一个新的项目集规范族 $C'=\{J_0, J_1, \dots, J_m\}$ ，其中含有项目 $[S' \rightarrow \bullet S, \$]$ 的 $J_k$ 为分析表的初态。
- 3.从 $C'$ 出发，构造action子表
  - a) 若 $[A \rightarrow \alpha \bullet a \beta, b] \in J_i$ ，且 $go(J_i, a)=J_j$ ，则置  $action[i, a]=S_j$
  - b) 若 $[A \rightarrow \alpha \bullet, a] \in J_i$ ，则置  $action[i, a]=R \ A \rightarrow \alpha$
  - c) 若 $[S' \rightarrow S \bullet, \$] \in J_i$ ，则置  $action[i, \$]=ACC$
- 4.构造 goto 子表  
设 $J_k=\{I_{i1}, I_{i2}, \dots, I_{it}\}$ ，由于这些 $I_i$ 是同心集，因此 $go(I_{i1}, X)$ 、 $go(I_{i2}, X)$ 、...、 $go(I_{it}, X)$ 也是同心集。  
把所有这些项目集合并后得到的集合记作 $J_i$ ，则有： $go(J_k, X)=J_i$   
于是，若  $go(J_k, A)=J_i$ ，则置  $goto[k, A]=i$
- 5.分析表中凡不能用上述规则填入信息的空表项，均置上出错标志。

# 示例: 构造文法4.8的LALR(1)分析表

- 有三对同心集可以合并, 即

- $I_3$  和  $I_6$  合并, 得到项目集:

$$I_{36} = \{[C \rightarrow c \bullet C, c/d/\$] \ [C \rightarrow \bullet c C, c/d/\$] \ [C \rightarrow \bullet d, c/d/\$]\}$$

- $I_4$  和  $I_7$  合并, 得到项目集:

$$I_{47} = \{[C \rightarrow d \bullet, c/d/\$]\}$$

- $I_8$  和  $I_9$  合并, 得到项目集:

$$I_{89} = \{[C \rightarrow c C \bullet, c/d/\$]\}$$

- 同心集合并后得到的新的项目集规范族为:

$$C' = \{I_0, I_1, I_2, I_{36}, I_{47}, I_5, I_{89}\}$$

- 利用算法4.10, 可以为该文法构造LALR(1)分析表。



# 文法4.8的LALR(1)分析表

状态	action			goto	
	c	d	\$	S	C
0	S36	S47		1	2
1			ACC		
2	S36	S47			5
36	S36	S47			89
47	R3	R3	R3		
5			R1		
89	R2	R2	R2		

- 此文法是LALR(1)文法
- 转移函数go的计算只依赖于项目集的心

# LALR(1)分析程序和LR(1)分析程序的比较

## ■ LR(1)分析程序对符号串 ccd 的分析过程

步骤	栈	输入	分析动作
0	0	ccd\$	shift
1	0c3	cd\$	shift
2	0c3c3	d\$	shift
3	0c3c3d4	\$	error

## ■ LALR(1)分析程序对符号串 ccd 的分析过程

步骤	栈	输入	分析动作
0	0	ccd\$	shift
1	0c36	cd\$	shift
2	0c36c36	d\$	shift
3	0c36c36d47	\$	reduce by $C \rightarrow d$
4	0c36c36C89	\$	reduce by $C \rightarrow cC$
5	0c36C89	\$	reduce by $C \rightarrow cC$
6	0C2	\$	error

# LALR(1)分析程序和LR(1)分析程序的比较

## ■ LR(1)分析程序对符号串 cdc d 的分析过程

步骤	栈	输入	分析动作
0	0	cdcd\$	shift
1	0c3	dcd\$	shift
2	0c3d4	cd\$	reduce by $C \rightarrow d$
3	0c3C8	cd\$	reduce by $C \rightarrow cC$
4	0C2	cd\$	shift
5	0C2c6	d\$	shift
6	0C2c6d7	\$	reduce by $C \rightarrow d$
7	0C2c6C9	\$	reduce by $C \rightarrow cC$
8	0C2C5	\$	reduce by $S \rightarrow CC$
9	0S1	\$	accept

## ■ LALR(1)分析程序对符号串 cdc d 的分析过程

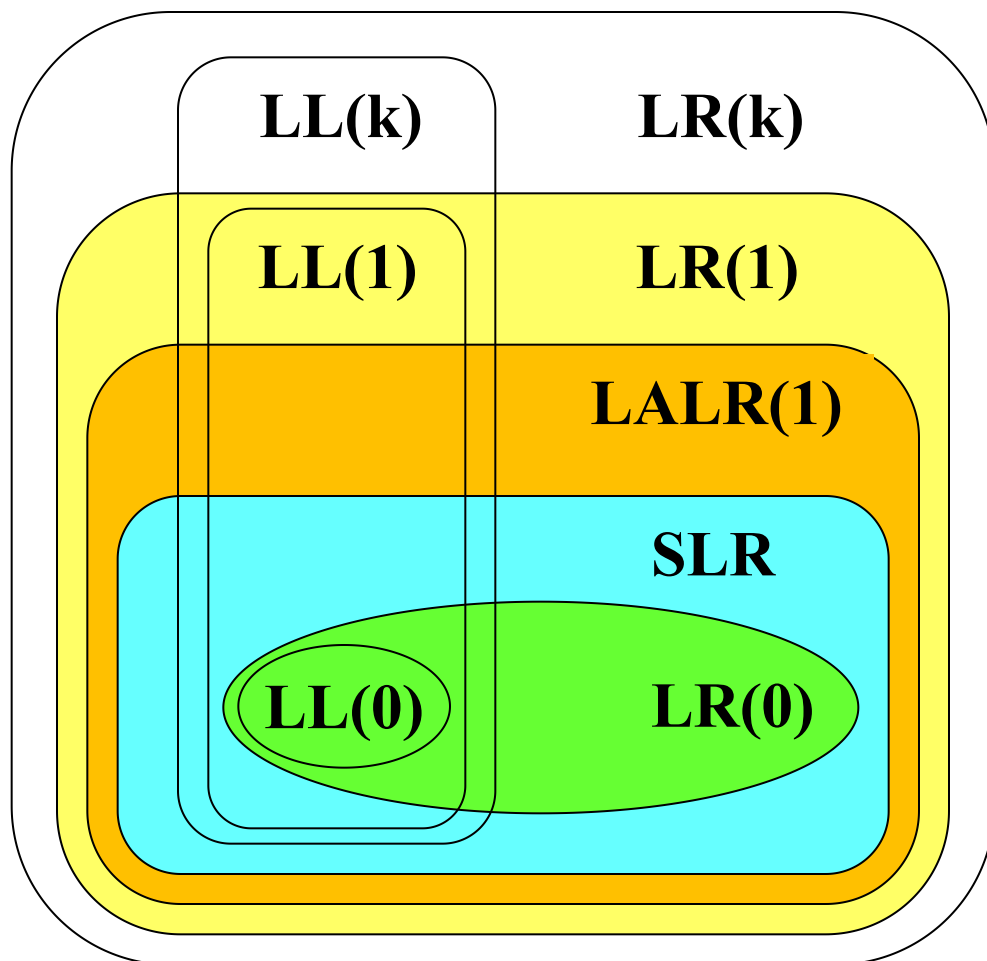
步骤	栈	输入	分析动作
0	0	cdcd\$	shift
1	0c36	dcd\$	shift
2	0c36d47	cd\$	reduce by $C \rightarrow d$
3	0c36C89	cd\$	reduce by $C \rightarrow cC$
4	0C2	cd\$	shift
5	0C2c36	d\$	shift
6	0C2c36d47	\$	reduce by $C \rightarrow d$
7	0C2c36C89	\$	reduce by $C \rightarrow cC$
8	0C2C5	\$	reduce by $S \rightarrow CC$
9	0S1	\$	accept

## ■ 结论：

- 对于正确的输入符号串：分析动作完全一样。
- 对于错误的输入符号串：LR(1)可以及时发现错误，LALR(1)可以在同样的位置发现错误，但要多做一些归约分析。

# 文法分类

非二义性文法



二义性文法

作业:

4.9

4.14

4.16

## 4.4.5 LR分析方法对二义文法的应用

skip

- 定理：任何二义性文法决不是LR文法，因而也不是SLR或LALR文法。

二义文法？

==>

- 程序设计语言的某些结构用二义性文法描述比较直观，使用方便。
  - 算术表达式
  - if语句
- 说明消除二义性的一些规则（即这类结构的使用限制）
  - 算术表达式：运算符优先级、运算结合规则
  - If语句：else的最近最后匹配原则

# 利用优先级和结合规则解决表达式冲突

- 描述算术表达式集合的二义性文法：

$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$       (文法4.10)

- 无二义性的文法：

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

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

- 前者具有两个明显的优点：

- 改变运算符优先级或运算结合规则时，文法本身无需改变，只需改变限制条件。
- LR分析表所包含的状态数比后者少

# 例：构造文法4.10的LR分析表。

## ■ 文法G (文法 4.10)

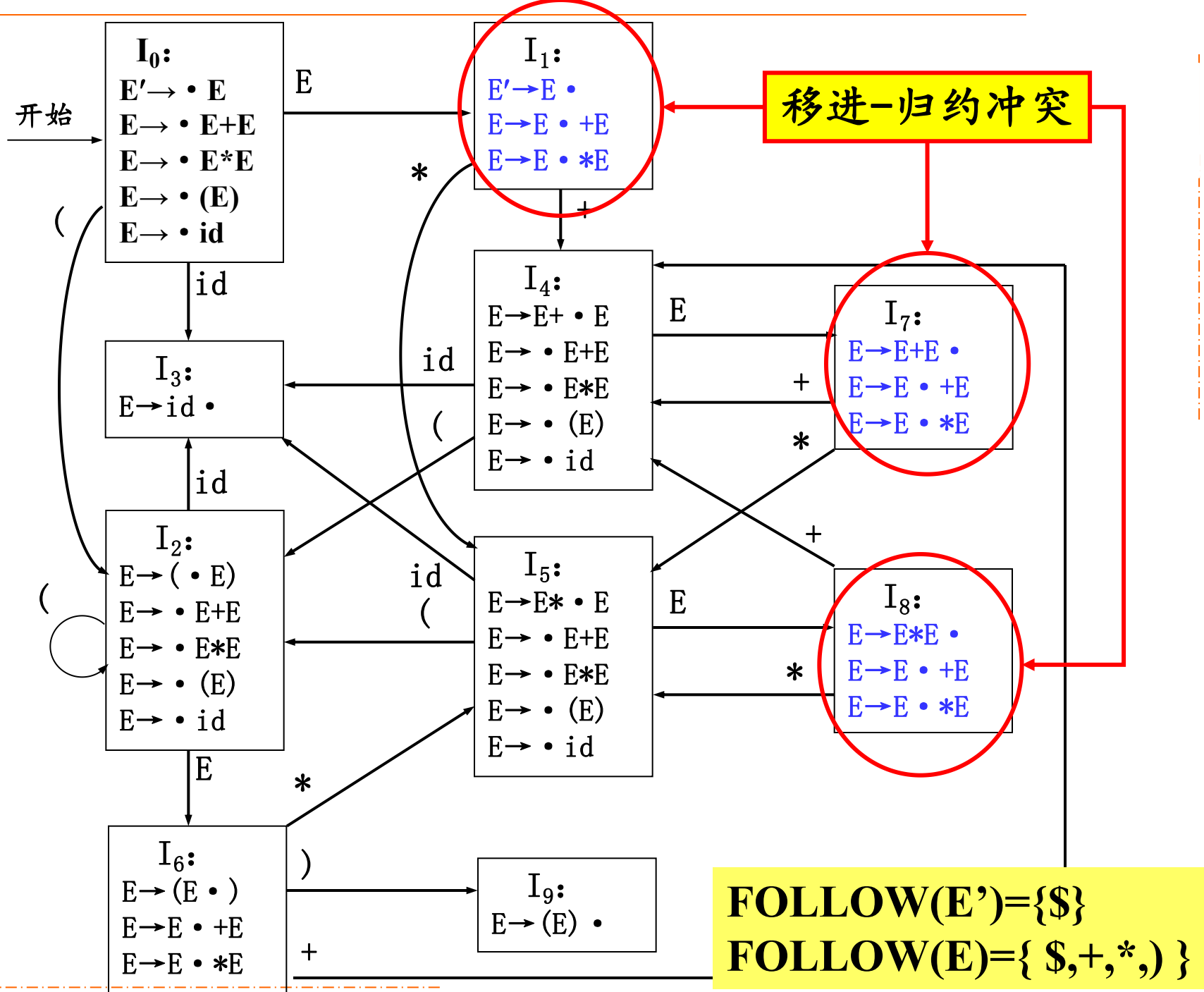
$E \rightarrow E + E \mid E * E \mid (E) \mid id$

## ■ 其拓广文法G'具有产生式：

(0)  $E' \rightarrow E$     (1)  $E \rightarrow E + E$     (2)  $E \rightarrow E * E$

(3)  $E \rightarrow (E)$     (4)  $E \rightarrow id$

## ■ 构造文法G'的LR(0)项目集规范族及识别其所有活前缀的DFA





# '+'和'\*'的优先级及结合规则共有4种情况

- (1) \* 优先于 +，遵从左结合规则
- (2) \* 优先于 +，遵从右结合规则
- (3) + 优先于 \*，遵从左结合规则
- (4) + 优先于 \*，遵从右结合规则

$I_7:$   
 $E \rightarrow E + E \cdot$   
 $E \rightarrow E \cdot + E$   
 $E \rightarrow E \cdot * E$

$I_8:$   
 $E \rightarrow E * E \cdot$   
 $E \rightarrow E \cdot + E$   
 $E \rightarrow E \cdot * E$

条件	状态	action					goto
		id	+	*	( )	\$	E
(1)	7		R1	S5	R1	R1	
	8		R2	R2	R2	R2	
(2)	7		S4	S5	R1	R1	
	8		R2	S5	R2	R2	
(3)	7		R1	R1	R1	R1	
	8		S4	R2	R2	R2	
(4)	7		S4	R1	R1	R1	
	8		S4	S5	R2	R2	

# 文法4.10的LR分析表

状态	action						goto
	id	+	*	(	)	\$	E
0	S3			S2			1
1		S4	S5			ACC	
2	S3			S2			6
3		R4	R4		R4	R4	
4	S3			S2			7
5	S3			S2			8
6		S4	S5		S9		
7		R1	S5		R1	R1	
8		R2	R2		R2	R2	
9		R3	R3		R3	R3	

# 解决if语句冲突：最近最后匹配原则

- 映射程序设计语言中if-then-else结构的文法：

$S \rightarrow \text{if } E \text{ then } S \text{ else } S$

$\quad | \text{if } E \text{ then } S$

$\quad | \text{others}$

(文法4.11)

- 对该文法进行抽象

- 用i表示“if E then”

- 用e表示“else”

- 用a表示“others”

- 得到文法：

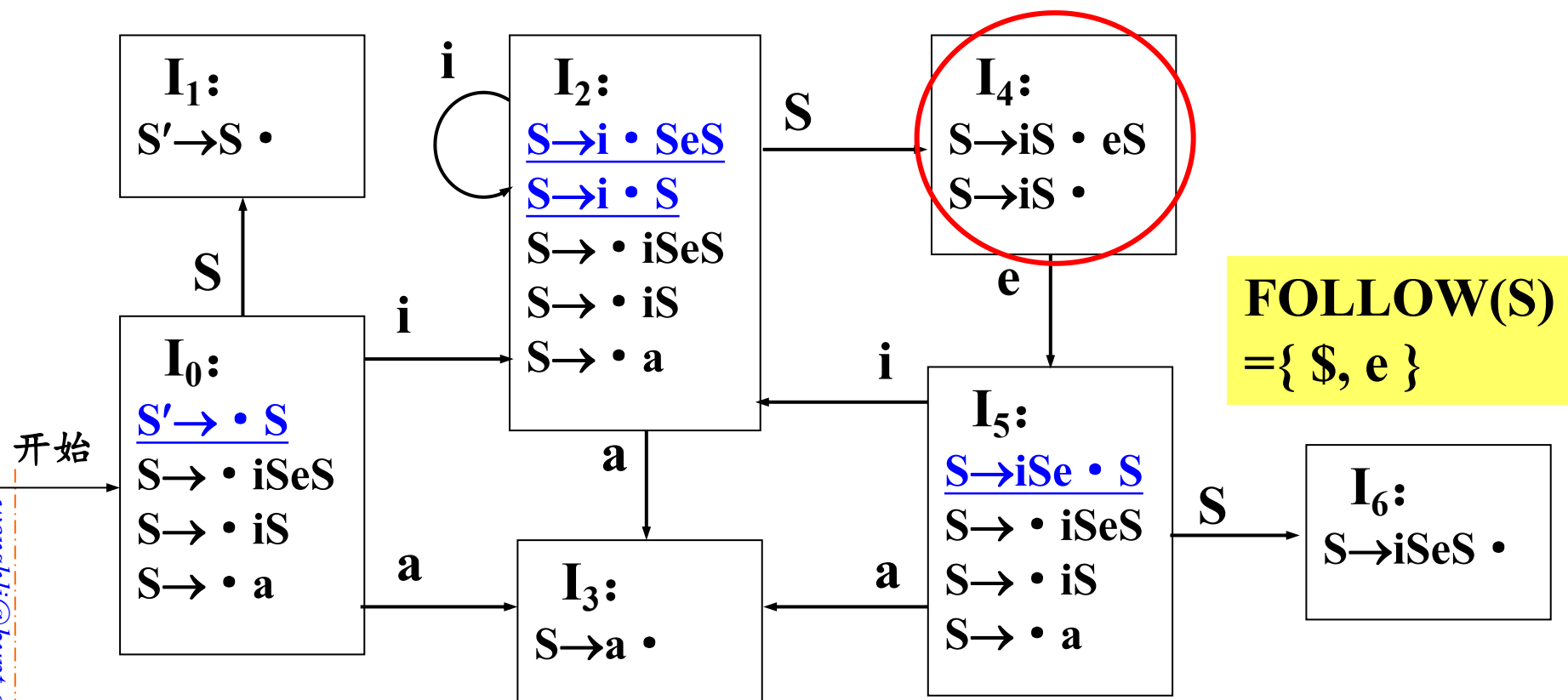
$S \rightarrow iS \mid iSeS \mid a$

(文法4.12)

# 文法4.12的LR(0)项目集规范族及识别其所有活前缀的DFA

其拓广文法 $G'$ 为：

(0)  $S' \rightarrow S$  (1)  $S \rightarrow iSeS$  (2)  $S \rightarrow iS$  (3)  $S \rightarrow a$



# 最近最后匹配原则

- else与离它最近的一个未匹配的then相匹配。
- 文法4.14 的LR分析表：

状态	action				goto
	i	e	a	\$	S
0	S2		S3		1
1				ACC	
2	S2		S3		4
3		R3		R3	
4		S5		R2	
5	S2		S3		6
6		R1		R1	

**I<sub>4</sub>:**  
 $S \rightarrow iS \cdot eS$   
 $S \rightarrow iS \cdot$

# 例：分析符号串iiaea

状态	action				goto
	i	e	a	\$	S
0	S2		S3		1
1				ACC	
2	S2		S3		4
3		R3		R3	
4		S5		R2	
5	S2		S3		6
6		R1		R1	

步骤	栈	输入	分析动作
1	0	iiaea\$	shift 2
	-		
2	0 2	iaea\$	shift 2
	- i		
3	0 2 2	aea\$	shift 2
	- i i		
4	0 2 2 3	ea\$	reduce by $S \rightarrow a$
	- i i a		
5	0 2 2 4	ea\$	shift
	- i i S		
6	0 2 2 4 5	a\$	shift
	- i i S e		
7	0 2 2 4 5 3	\$	reduce by $S \rightarrow a$
	- i i S e a		
8	0 2 2 4 5 6	\$	reduce by $S \rightarrow iSeS$
	0 i i S e S		
9	0 2 4	\$	reduce by $S \rightarrow iS$
	- i S		
10	0 1	\$	accept
	- S		

## 4.4.6 LR分析的错误处理与恢复

### ■ LR分析程序可采取以下恢复策略：

□ 首先，从栈顶开始退栈，可能弹出0个或若干个状态，直到出现状态S为止。

(1) 状态S有相对于当前输入符号的转移，或者

(2) goto表中有S相对于某非终结符号A的后继。

□ 针对(1)：移进，分析继续。

□ 针对(2)：跳过0个或若干个输入符号，直到出现符号a为止， $a \in \text{FOLLOW}(A)$ ；

然后，把状态goto[S, A]压入栈顶，继续分析。

■ A的选择：通常选择表示主要结构成分的非终结符号。

■ 跳过了包含错误的一段终结符号串。

# 例：考虑符号遵从

期待输入符号为运算符或右括号，而遇到的却是运算对象（id或左括号）。

诊断信息：“缺少运算符”

恢复策略：把运算符符号 ‘+’压入栈，转移到状态4。

即遇到

状态							goto
	id	(	)	\$	E		
0	S3	e1	e1	S2	e2	e1	1
1	e3	S4	S5	e3	e2	ACC	
2	S3	e1	e1	S2	e2	e1	6
3	R4	R4	R4	R4	R4	R4	
4	S3	e1	e1	S2	e2	e1	7
5	S3	e1	e1	S2	e2	e1	8
6	e3	S4	S5	e3	S9	e4	
7	R1	R1	S5	R1	R1		
8	R2	R2	R2	R2	R2		

期待输入符号为运算符或右括号，而遇到的却是输入串结束标志 ‘\$’  
诊断信息：“缺少右括号”

期待输入符号为运算符或右括号，而遇到的却是输入串结束标志 ‘\$’。

诊断信息：“缺少右括号”

恢复策略：把右括号压入栈，转移到状态9。



# 错误处理与恢复示例

- e1: 在状态0、2、4、5, 期待输入符号为运算对象的首字符, 即id或(, 而输入中出现的却是运算符‘+’或‘\*’, 或是输入串结束标志‘\$’。
  - 策略: 把一个假想的id压入栈, 并将状态3推入栈顶。
  - 诊断信息: “缺少运算对象”
- e2: 在状态0、1、2、4、5, 期待输入符号为运算对象的首字符或运算符, 但却遇到右括号。
  - 策略: 删掉输入的右括号
  - 诊断信息: “括号不匹配”
- e3: 在状态1、6, 期待输入符号为运算符或右括号, 而遇到的却是运算对象(id或左括号)。
  - 策略: 把运算符‘+’压入栈, 转移到状态4。
  - 诊断信息: “缺少运算符”
- e4: 在状态6, 期待输入符号为运算符或右括号, 而遇到的却是输入串结束标志‘\$’。
  - 策略: 把右括号压入栈, 转移到状态9。
  - 诊断信息: “缺少右括号”

# 示例：分析符号串 id+)

步骤	栈	输入	分析动作
(1)	0	id+)\$	shift
	—		
(2)	0 3	+) \$	reduce by $E \rightarrow id$
	— id		
(3)	0 1	+) \$	shift
	— E		
(4)	0 1 4	) \$	CALL e2 “括号不匹配”，删掉 ‘)’
	— E +		
(5)	0 1 4	\$	CALL e1 “缺少运算对象”，id压入栈
	— E +		
(6)	0 1 4 3	\$	reduce by $E \rightarrow id$
	— E + id		
(7)	0 1 4 7	\$	reduce by $E \rightarrow E + E$
	— E + E	\$	reduce by $E \rightarrow E + E$
(8)	0 1	\$	accept
	— E		



# 本章小结

## 一、自顶向下的分析方法

### ■ 递归下降分析方法

- 试探性、回溯
- 要求：文法不含左递归

### ■ 递归调用预测分析方法

- 不带回溯的递归分析方法
- 要求：
  - 文法不含左递归，并且
  - 对任何产生式： $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$   
 $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$
- 构造步骤：
  - 描述结构的上下文无关文法
  - 根据文法构造预测分析程序的状态转换图
  - 状态转换图化简
  - 根据状态转换图构造递归过程

# 本章小结

## ■ 非递归预测分析方法

- 不带回溯、不含递归

- 模型：

输入缓冲区：存放输入符号串  $a_1a_2\dots a_n\$$

符号栈：分析过程中存放文法符号

分析表：二维表，每个A有一行，每个a包括\$有一列

表项内容是产生式（关键）

控制程序：根据栈顶X和当前输入a决定分析动作（永恒的核心）

$X=a=\$$  分析成功

$X=a\neq\$$  弹出X，扫描指针前移

X是非终结符号，查分析表： $M[X, a]$

$M[X, a]=X\rightarrow Y_1Y_2\dots Y_K$ ，弹出X， $Y_K$ 、...、 $Y_2$ 、 $Y_1$ 入栈

$M[X, a]=X\rightarrow\epsilon$ ，弹出X

$M[X, a]=$ 空白，出错处理

输出：对输入符号串进行最左推导所用的产生式序列

# 本章小结

## ■ 预测分析表的构造

- 构造每个文法符号的**FIRST**集合
- 构造每个非终结符号的**FOLLOW**集合
- 检查每个产生式 $A \rightarrow \alpha$ 
  - 对任何 $a \in \text{FIRST}(\alpha)$ ,  $M[A, a] = A \rightarrow \alpha$
  - 若 $\alpha \Rightarrow \varepsilon$ , 对所有 $b \in \text{FOLLOW}(A)$ ,  $M[A, b] = A \rightarrow \alpha$

## ■ LL(1)文法

- LL(1)的含义
- 判断一个文法是否为LL(1) 文法
  - 构造分析表, 或者
  - 检查每个产生式:  $A \rightarrow \alpha | \beta$   
 $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \phi$   
若 $\beta \Rightarrow \varepsilon$ , 则 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \phi$

# 本章小结

## 二、自底向上分析方法

### ■ 移进-归约分析方法

- 分析栈、输入缓冲区
- 可归约串
- 规范归约：最右推导的逆过程

### ■ LR分析方法

#### □ 模型：

输入缓冲器：                      输出：分析动作序列

分析栈： $S_0X_1S_1...X_nS_n$

分析表：包括 action 和 goto 两部分（关键）

控制程序：根据栈顶状态  $S_n$  和当前输入符号  $a$  查分析表

$action[S_n, a]$ ，决定分析动作（永恒的核心）

$action[S_n, a]=S\ i$ ， $a$ 入符号栈， $i$ 入状态栈  $i=goto(S_n, a)$

$action[S_n, a]=R\ A\rightarrow\beta$ ，弹出  $|\beta|$ 个符号，

$A$ 入符号栈， $goto(S_{n-r}, A)$ 入状态栈

$action[S_n, a]=ACC$ ，分析成功

$action[S_n, a]=$ 空白，出错处理

# 本章小结

## ■ SLR(1)分析表的构造

- LR(0)项目集规范族
- 识别文法所有活前缀的DFA
- 构造分析表：检查每个状态集
  - 若  $A \rightarrow \alpha \cdot a\beta \in I_i$ ，且  $go(I_i, a) = I_j$ ，则置  $action[i, a] = Sj$ ，
  - 若  $A \rightarrow \alpha \cdot \in I_i$ ，则对所有  $a \in FOLLOW(A)$ ，置  $action[i, a] = R A \rightarrow \alpha$
  - 若  $S' \rightarrow S \cdot \in I_i$ ，则置  $action[i, \$] = ACC$ ，表示分析成功。
  - 若  $go(I_i, A) = I_j$ ， $A$  为非终结符号，则置  $goto[i, A] = j$

## ■ LR(1)分析表的构造

- LR(1)项目集规范族
- 构造分析表：检查每个状态集
  - 若  $[A \rightarrow \alpha \cdot a\beta, b] \in I_i$ ，且  $go(I_i, a) = I_j$ ，则置  $action[i, a] = Sj$ ，
  - 若  $[A \rightarrow \alpha \cdot, a] \in I_i$ ，则置  $action[i, a] = R A \rightarrow \alpha$ ，
  - 若  $[S' \rightarrow S \cdot, \$] \in I_i$ ，则置  $action[i, \$] = ACC$ ，表示分析成功。
  - 若  $go(I_i, A) = I_j$ ， $A$  为非终结符号，则置  $goto[i, A] = j$

# 本章小结

## ■ LALR(1)分析表的构造

- LR(1)项目集规范族，若没有冲突，继续
- 合并同心集，若没有冲突，则为LALR(1)项目集规范族
- 构造分析表，方法同LR(1)分析表的构造方法

## ■ 利用LR技术处理二义文法

- 利用运算符的优先级和结合性质，处理算术表达式文法
- 利用最近最后匹配原则，处理IF语句文法。



# 算法清单

- 算法4.1 非递归预测分析方法
- 算法4.2 预测分析表的构造方法
- 算法4.3 LR分析程序
- 算法4.4  $\text{closure}(I)$ 的构造过程 ( $I$ 为LR(0)项目集)
- 算法4.5 构造文法 $G$ 的LR(0)项目集规范族
- 算法4.6 构造文法 $G$ 的SLR(1)分析表
- 算法4.7  $\text{closure}(I)$ 的构造过程 ( $I$ 为LR(1)项目集)
- 算法4.8 构造文法 $G$ 的LR(1)项目集规范族
- 算法4.9 构造文法 $G$ 的LR(1)分析表
- 算法4.10 构造LALR(1)分析表

给定文法, 构造 候选式/非终结符号的 FIRST 集合  
构造 非终结符号的 FOLLOW 集合

# 作业

■ 4.1

■ 4.3

■ 4.5

■ 4.9

■ 4.14

■ 4.16

题目：语法分析程序的设计与实现

实验内容：编写语法分析程序，实现对算术表达式的语法分析。要求所分析算术表达式由如下的文法产生。

$$E \rightarrow E+T \mid E-T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow (E) \mid \text{num}$$

实验要求：在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

方法1：编写递归调用程序实现自顶向下的分析。

方法2：编写LL(1)语法分析程序，要求如下。（必做）

(1) 编程实现算法4.2，为给定文法自动构造预测分析表。

(2) 编程实现算法4.1，构造LL(1)预测分析程序。

方法3：编写语法分析程序实现自底向上的分析，要求如下。（必做）

(1) 构造识别该文法所有活前缀的DFA。

(2) 构造该文法的LR分析表。

(3) 编程实现算法4.3，构造LR分析程序。

方法4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

# 实验报告要求

- 实验题目、要求
- 程序设计说明
- 源程序
- 可执行程序
- 测试报告：包括输入、运行结果、及结果分析
- 提交：
  - 所有资料打包，命名规则： 班级-学号-姓名
- 在线提交。

# 课堂练习4

已知文法G[A]为：

$$A \rightarrow aABe|a$$

$$B \rightarrow Bb|d$$

$$A \rightarrow aA'$$

$$A' \rightarrow ABe \mid \varepsilon$$

$$B \rightarrow dB'$$

$$B' \rightarrow bB' \mid \varepsilon$$

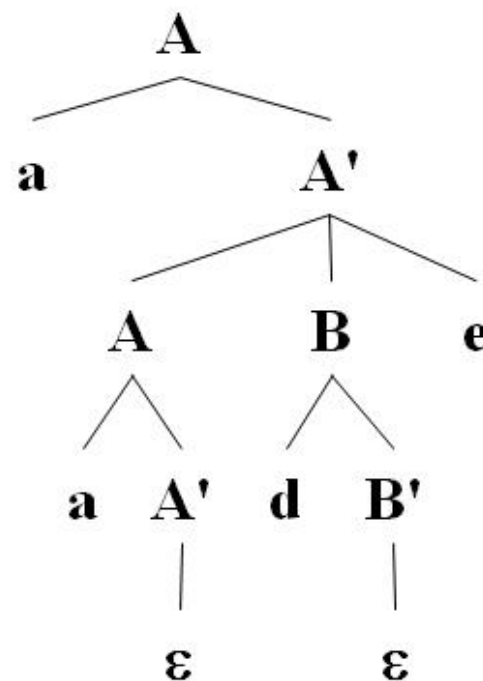
	First	Follow
A	a	\$, d
A'	a, $\varepsilon$	\$, d
B	d	e
B'	b, $\varepsilon$	e

(1) 试给出与G[A]等价的LL(1)文法G'[A]

(2) 构造G'[A]的预测分析表

(3) 给出输入串aade的分析过程。

	a	b	d	e	\$
A	$A \rightarrow aA'$				
A'	$A' \rightarrow ABe$		$A' \rightarrow \varepsilon$		$A' \rightarrow \varepsilon$
B			$B \rightarrow dB'$		
B'		$B' \rightarrow bB'$		$B' \rightarrow \varepsilon$	



# aade的分析过程

步骤	栈	输入	分析动作
(1)	\$A	aade\$	$A \rightarrow aA'$
(2)	\$A'a	aade\$	
(3)	\$A'	ade\$	$A' \rightarrow ABe$
(4)	\$eBA	ade\$	$A \rightarrow aA'$
(5)	\$eBA'a	ade\$	
(6)	\$eBA'	de\$	$A' \rightarrow \epsilon$
(7)	\$eB	de\$	$B \rightarrow dB'$
(8)	\$eB'd	de\$	
(9)	\$eB'	e\$	$B' \rightarrow \epsilon$
(10)	\$e	e\$	
(11)	\$	\$	分析成功

	a	b	d	e	\$
A	$A \rightarrow aA'$				
A'	$A' \rightarrow ABe$		$A' \rightarrow \epsilon$		$A' \rightarrow \epsilon$
B			$B \rightarrow dB'$		
B'		$B' \rightarrow bB'$		$B' \rightarrow \epsilon$	

# 课堂练习5

有如下文法G[A]:

$$A \rightarrow BA \mid a$$
$$B \rightarrow aB \mid b$$

(1) 判断该文法是以下哪些类型的文法，要求给出判断过程。

LL(1)、LR(0)、SLR(1)

(2) 构造该文法的LR(1)项目集规范族及识别其所有活前缀的DFA。

(3) 构造该文法的LR(1)分析表

(4) 给出对输入符号串abb的分析过程。

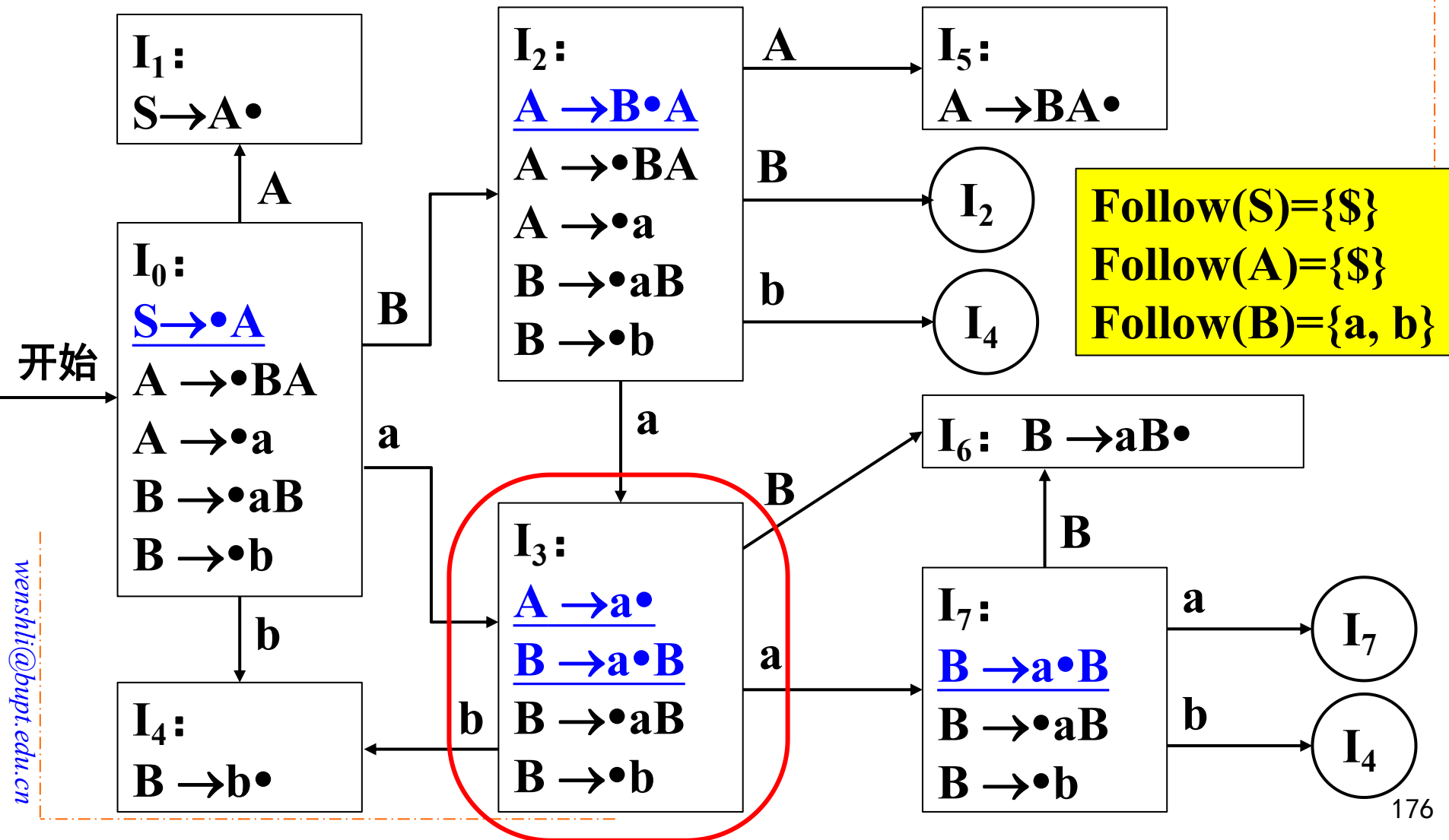
# 参考答案

LR(0)文法，项目集中：

- (1) 要么所有元素都是移进-待约项目
- (2) 要么只含有唯一的归约项目

- (0)  $S \rightarrow A$
- (1)  $A \rightarrow BA$
- (2)  $A \rightarrow a$
- (3)  $B \rightarrow aB$
- (4)  $B \rightarrow b$

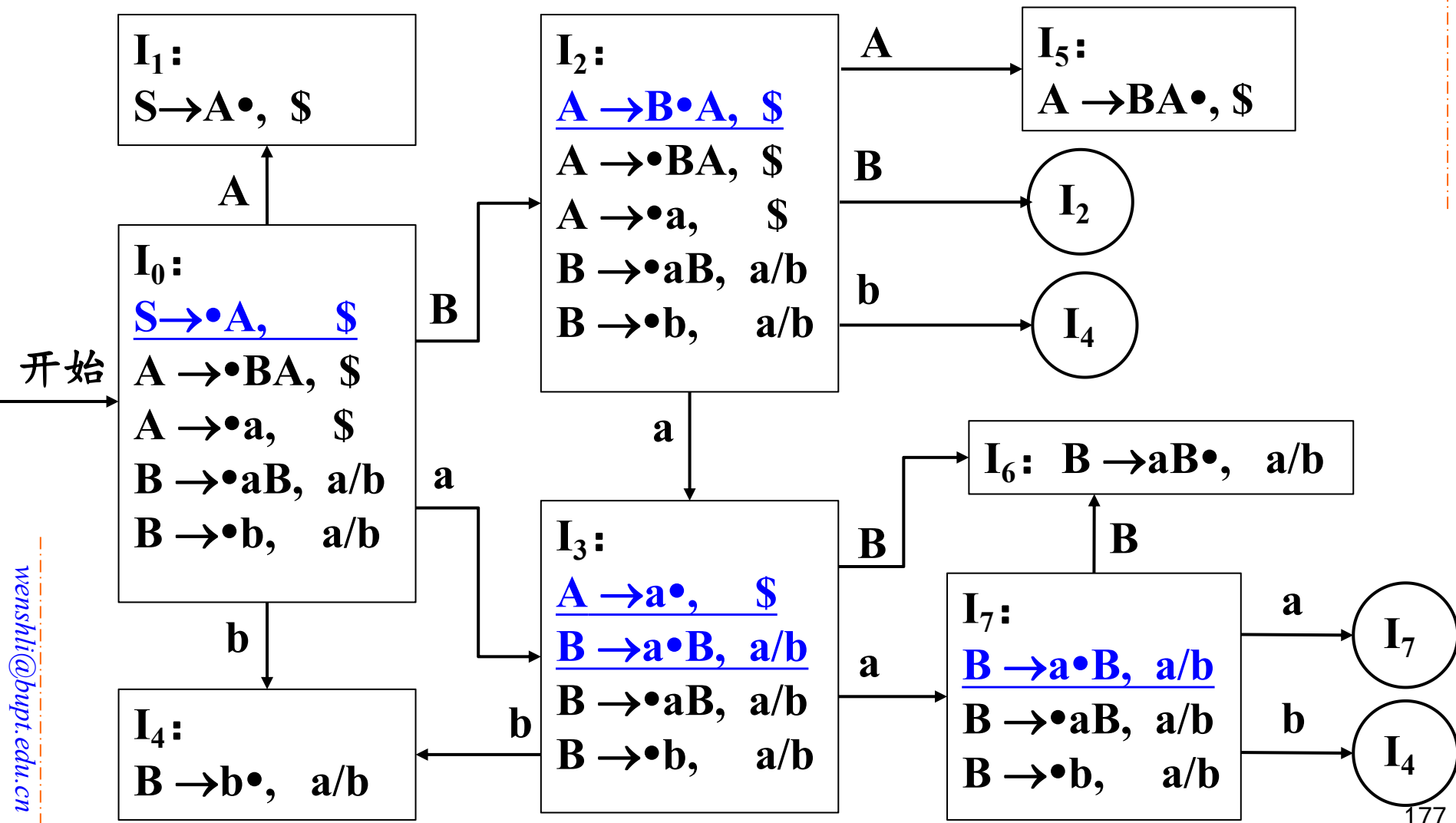
LR(0)项目集规范族及识别其所有活前缀的DFA:





# 参考答案

LR(1)项目集规范族及识别其所有活前缀的DFA:



# 文法的LR(1)分析表

状态	action			goto	
	a	b	\$	A	B
0	S3	S4		1	2
1			ACC		
2	S3	S4		5	2
3	S7	S4			6
4	R4	R4			
5			R1		
6	R3	R3			
7	S7	S4			6

# abb的分析过程

步骤	栈	输入	分析动作
(1)	0	abb\$	S3
(2)	0 3 - a	bb\$	S4
(3)	0 3 4 - a b	b\$	R4 B→b
(4)	0 3 6 - a B	b\$	R3 B→aB
(5)	0 2 - B	b\$	S4
(6)	0 2 4 - B b	\$	error 弹出栈顶状态4
(7)	0 2 - B	\$	goto(2, A)=5 将状态5压入栈顶
(8)	0 2 5 - B A	\$	R1 A→BA
(9)	0 1 - A	\$	accept



## \* 推导和短语

例：考虑简单算术表达式的文法G：

$$G = (\{+, *, (, ), i\}, \{E, T, F\}, E, \varphi)$$
$$\varphi: E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid i$$

### ■ 文法所产生的语言

- 从文法的开始符号出发，反复连续使用产生式对非终结符号进行替换和展开，得到的终结符号串是该文法定义的句子。
- 文法所有句子构成的集合就是相应文法的语言。

## \* 推导

假定  $A \rightarrow \gamma$  是一个产生式,  $\alpha$  和  $\beta$  是任意的文法符号串, 则有:  $\alpha A \beta \Rightarrow \alpha \gamma \beta$

■ “ $\Rightarrow$ ” 表示 “一步推导”

利用产生式对左边符号串中的一个非终结符号进行替换, 得到右边的符号串。

■ 称  $\alpha A \beta$  直接推导出  $\alpha \gamma \beta$

■ 也可以说  $\alpha \gamma \beta$  是  $\alpha A \beta$  的直接推导

■ 或说  $\alpha \gamma \beta$  直接归约到  $\alpha A \beta$

# \* 推导

如果有直接推导序列:  $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$

则说 $\alpha_1$ 推导出 $\alpha_n$ , 记作:  $\alpha_1 \xRightarrow{*} \alpha_n$

从 $\alpha_1$ 到 $\alpha_n$ 的长度为n的  
推导

0步或多步推导

从文法开始符号E推导出符号串i+i的详细过程

$\alpha A \beta$	$\alpha \gamma \beta$	$\alpha$	$\beta$	所用产生式	从E到 $\alpha \gamma \beta$ 的推导步数
E	E+T	$\epsilon$	$\epsilon$	$E \rightarrow E+T$	1
E+T	T+T	$\epsilon$	+T	$E \rightarrow T$	2
T+T	F+T	$\epsilon$	+T	$T \rightarrow F$	3
F+T	i+T	$\epsilon$	+T	$F \rightarrow i$	4
i+T	i+F	i+	$\epsilon$	$T \rightarrow F$	5
i+F	i+i	i+	$\epsilon$	$F \rightarrow i$	6

**$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$**

# \* 推导

## 最左推导

如果  $\alpha \xRightarrow{*} \beta$ , 并且在每“一步推导”中, 都替换 $\alpha$ 中**最左边**的非终结符号, 则称这样的推导为最左推导。记作:  $\alpha \xRightarrow{*}_{lm} \beta$

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$$

## 最右推导

如果  $\alpha \xRightarrow{*} \beta$ , 并且在每“一步推导”中, 都替换 $\alpha$ 中**最右边**的非终结符号, 则称这样的推导为最右推导。记作:  $\alpha \xRightarrow{*}_{rm} \beta$

最右推导也称为**规范推导**。

$$E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+i \Rightarrow T+i \Rightarrow F+i \Rightarrow i+i$$

# \* 句型、句子和语言

## 句型

对于文法  $G=(V_T, V_N, S, \varphi)$ , 如果  $S \xRightarrow{*} \alpha$ , 则称  $\alpha$  是当前文法的一个句型。

若  $S \xRightarrow[lm]{*} \alpha$ , 则  $\alpha$  是当前文法的一个左句型,

若  $S \xRightarrow[rm]{*} \alpha$ , 则  $\alpha$  是当前文法的一个右句型。

## 句子

仅含有终结符号的句型是文法的一个句子。

## 语言

文法  $G$  产生的所有句子组成的集合是文法  $G$  所定义的语言, 记作  $L(G)$ 。

$$L(G)=\{ \alpha | S \xRightarrow{+} \alpha, \text{ 并且 } \alpha \in V_T^* \}$$



## \* 短语

对于文法 $G=(V_T, V_N, S, \varphi)$ ，假定 $\alpha\beta\delta$ 是文法 $G$ 的一个句型，如果存在：

$$S \xRightarrow{*} \alpha A \delta, \text{ 并且 } A \xRightarrow{+} \beta$$

则称 $\beta$ 是句型 $\alpha\beta\delta$ 关于非终结符号 $A$ 的**短语**。

如果存在：

$$S \xRightarrow{*} \alpha A \delta, \text{ 并且 } A \Rightarrow \beta$$

则称 $\beta$ 是句型 $\alpha\beta\delta$ 关于非终结符号 $A$ 的**直接短语**。

一个句型的最左直接短语称为该句型的**句柄**。

例：

$$\begin{array}{cccccccccccc} \underline{E} \Rightarrow \underline{T} \Rightarrow \underline{T * F} \Rightarrow \underline{T * (E)} \Rightarrow \underline{F * (E)} \Rightarrow \underline{i * (E)} \Rightarrow \underline{i * (E + T)} \Rightarrow \underline{i * (T + T)} \Rightarrow \underline{i * (F + T)} \Rightarrow \underline{i * (i + T)} \\ \textcircled{1} \quad \textcircled{2} \quad \textcircled{3} \quad \quad \textcircled{4} \quad \quad \textcircled{5} \quad \quad \textcircled{6} \quad \quad \textcircled{7} \quad \quad \textcircled{8} \quad \quad \textcircled{9} \quad \quad \textcircled{10} \end{array}$$

# \* 改写文法

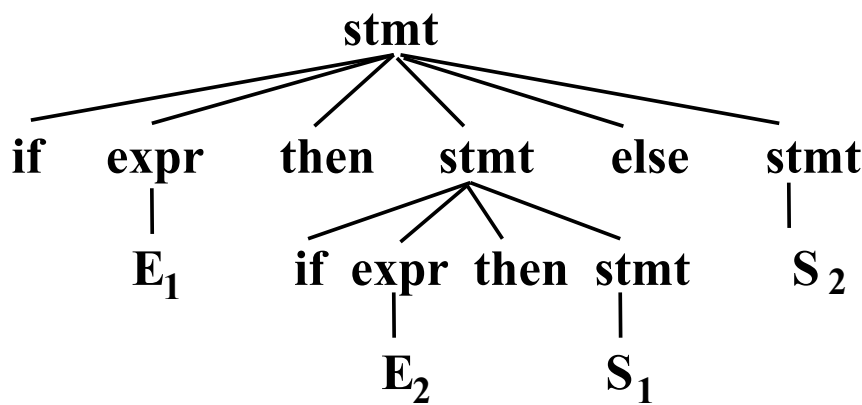
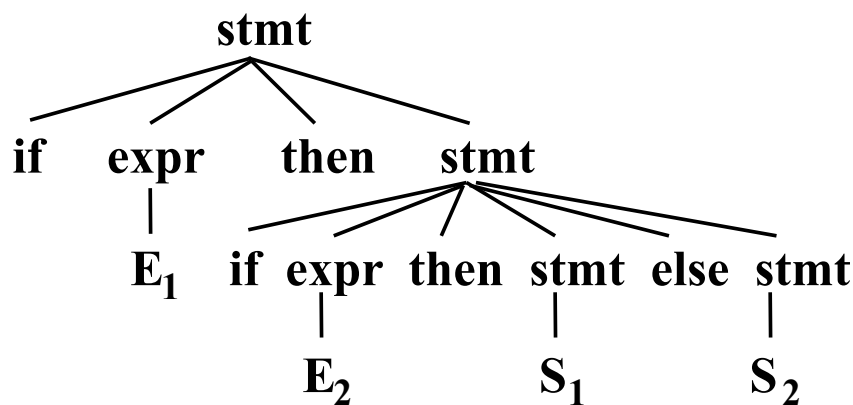
- 文法二义性的消除
- 左递归的消除
- 提取左公因子

# \* 文法二义性的消除

## ■ 映射程序设计语言中IF语句的文法:

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

## ■ 句子if $E_1$ then if $E_2$ then $S_1$ else $S_2$ 有两棵不同的分析树:



## \* 最近最后匹配原则

else必须匹配离它最近的那个未匹配的then

- 出现在then和else之间的语句必须是“匹配的”。
- 所谓匹配的语句是不包含不匹配语句的if-then-else语句，或是其他任何非条件语句。
- 改写后的文法：

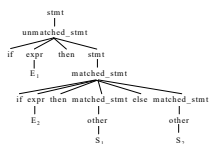
$stmt \rightarrow matched\_stmt \mid unmatched\_stmt$

$matched\_stmt \rightarrow \text{if } expr \text{ then } matched\_stmt \text{ else } matched\_stmt$   
 $\quad \mid \text{ other}$

$unmatched\_stmt \rightarrow \text{if } expr \text{ then } stmt$   
 $\quad \mid \text{if } expr \text{ then } matched\_stmt \text{ else } unmatched\_stmt$

# \* 改写文法

句子  $\text{if } E_1 \text{ then if } E_2 \text{ then } S_1 \text{ else } S_2$  的分析树



# \* 左递归的消除

- 一个文法是左递归的，如果它有非终结符号A，对某个文法符号串 $\alpha$ ，存在推导： $A \xRightarrow{+} A\alpha$
- 若存在某个 $\alpha=\varepsilon$ ，则称该文法是有环路的。
- 消除左递归的方法：

□ 简单情况：如果文法G有产生式： $A \rightarrow A\alpha \mid \beta$   
可以把A的这两个产生式改写为： $A \rightarrow \beta A'$   
 $A' \rightarrow \alpha A' \mid \varepsilon$

□ 一般情况：假定关于A的全部产生式是：

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$   
产生式可以改写为： $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$   
 $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$

## \* 左递归的消除

**例：**消除表达式文法中的左递归：

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

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

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

■ 简单情况，改写为：

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \varepsilon$$

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

间接左递归文法：

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

## \* 算法：消除左递归

输入：无环路、无 $\varepsilon$ -产生式的文法G

输出：不带有左递归的、与G等价的文法  $G'$

方法：

(1) 把文法G的所有非终结符号按某种顺序排列成 $A_1, A_2, \dots, A_n$

(2) for  $i:=1$  to  $n$  do

    for  $j:=1$  to  $i-1$  do

        begin

            把每个形如  $A_i \rightarrow A_j \gamma$  的产生式改写为： $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$ ;

            其中  $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$  是关于当前 $A_j$ 的所有产生式

            消除关于 $A_i$ 的产生式中的直接左递归

        end

(3) 化简第(2)步得到的文法，即去除无用的非终结符号和产生式。

这种方法得到的非递归文法可能含有 $\varepsilon$ -产生式。



## 为含有 $\varepsilon$ -产生式的文法 $G=(V_T, V_N, S, \varphi)$ 构造不含 $\varepsilon$ -产生式的文法 $G'=(V_T', V_N', S, \varphi')$ 的方法

- 若产生式 $A \rightarrow X_1 X_2 \dots X_n \in \varphi$

则把产生式 $A \rightarrow \alpha_1 \alpha_2 \dots \alpha_n$ 加入 $\varphi'$

其中： $X_i$ 、 $\alpha_i$ 为文法符号，即 $X_i$ 、 $\alpha_i \in (V_T \cup V_N)$

若 $X_i$ 不能产生 $\varepsilon$ ，则 $\alpha_i = X_i$

若 $X_i$ 能产生 $\varepsilon$ ，则 $\alpha_i = X_i$  或  $\alpha_i = \varepsilon$

不能所有的 $\alpha_i$ 都取 $\varepsilon$

- 文法 $G'$ 满足：
$$L(G') = \begin{cases} L(G) - \{\varepsilon\} & \text{如果 } L(G) \text{ 中含有 } \varepsilon \\ L(G) & \text{如果 } L(G) \text{ 中不含有 } \varepsilon \end{cases}$$

- 一个文法是 $\varepsilon$ -无关的

- 如果它没有 $\varepsilon$ -产生式（即形如 $A \rightarrow \varepsilon$ 的产生式），或者
- 只有一个 $\varepsilon$ -产生式，即 $S \rightarrow \varepsilon$ ，并且文法的开始符号 $S$ 不出现在任何产生式的右部

例:

消除下面文法中的左递归

$$S \rightarrow Aa \mid b$$
$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

■ 首先，必须保证此文法中无环路、无 $\varepsilon$ -产生式。

■ 改写为无 $\varepsilon$ -产生式的文法：

$$S \rightarrow Aa \mid a \mid b$$
$$A \rightarrow Ac \mid c \mid Sd$$

■ 消除其中的左递归：

step1: 把文法的非终结符号排列为S、A；

step2: 由于S不存在直接左递归，所以算法第2步在 $i=1$ 时不做工作；

在 $i=2$ 时，把产生式  $S \rightarrow Aa \mid a \mid b$  代入A的有关产生式中，得到：

$$A \rightarrow Ac \mid c \mid Aad \mid ad \mid bd$$

消除A产生式中的直接左递归，得到文法：

$$S \rightarrow Aa \mid a \mid b$$
$$A \rightarrow cA' \mid adA' \mid bdA'$$
$$A' \rightarrow cA' \mid adA' \mid \varepsilon$$

## \* 提取左公因子

- 如有产生式  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

提取左公因子 $\alpha$ ，则原产生式变为：

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 \mid \beta_2$$

- 若有产生式  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$

可用如下的产生式代替：

$$A \rightarrow \alpha A' \mid \gamma$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

## \* 提取左公因子

例：映射程序设计语言中IF语句的文法

$$\begin{aligned} stmt &\rightarrow \text{if } expr \text{ then } stmt \\ &\quad | \text{if } expr \text{ then } stmt \text{ else } stmt \\ &\quad | a \\ expr &\rightarrow b \end{aligned}$$

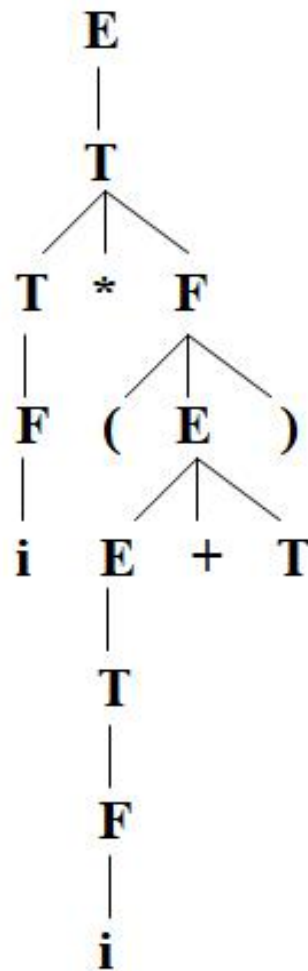
■ 左公因子  $\text{if } expr \text{ then } stmt$

■ 提取左公因子，得到文法：

$$\begin{aligned} stmt &\rightarrow \text{if } expr \text{ then } stmt S' | a \\ S' &\rightarrow \text{else } stmt | \varepsilon \\ expr &\rightarrow b \end{aligned}$$

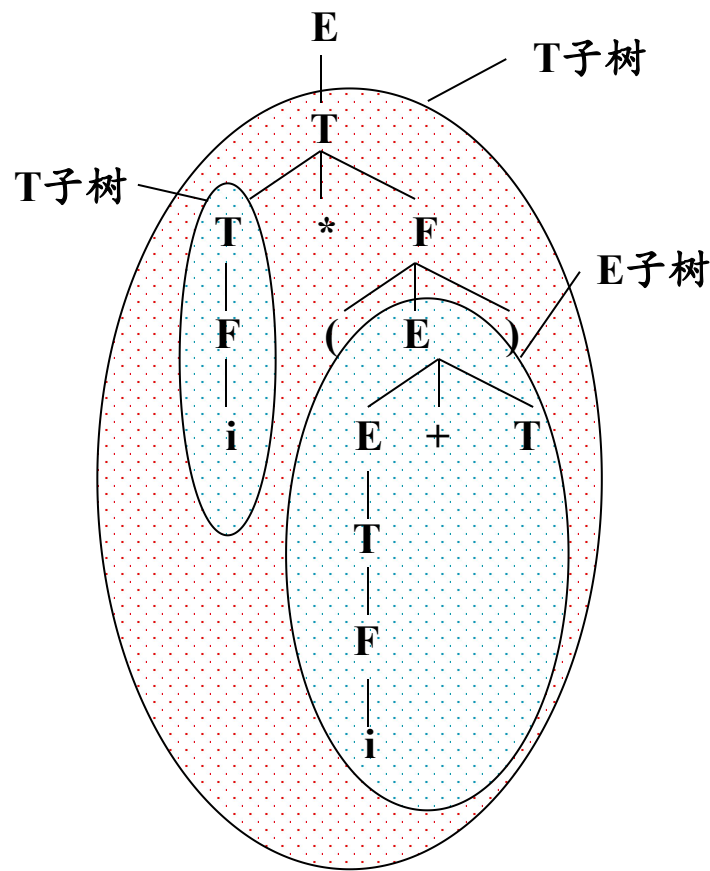
# \* 分析树

- 推导的图形表示，又称推导树。
- 一棵有序有向树，具有树的性质；
- 特点：每一个结点都有标记。
  - 根结点由文法的开始符号标记；
  - 内部结点由非终结符号标记，其子结点由这个非终结符号的这次推导所用产生式的右部各符号从左到右依次标记；
  - 叶结点由非终结符号或终结符号标记，它们从左到右排列起来，构成句型。
- $E \Rightarrow T \Rightarrow T * F \Rightarrow F * F \Rightarrow i * F \Rightarrow i * (E)$   
 $\Rightarrow i * (E + T) \Rightarrow i * (T + T) \Rightarrow i * (F + T) \Rightarrow i * (i + T)$
- $E \Rightarrow T \Rightarrow T * F \Rightarrow T * (E) \Rightarrow F * (E) \Rightarrow i * (E)$   
 $\Rightarrow i * (E + T) \Rightarrow i * (T + T) \Rightarrow i * (F + T) \Rightarrow i * (i + T)$



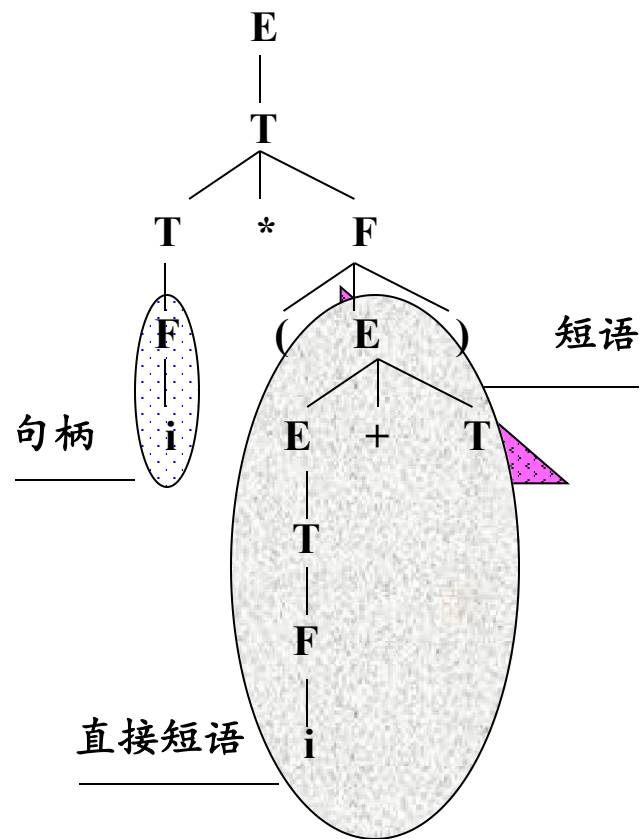
# \*子树

- 分析树中一个特有的**结点**、连同它的**全部后裔结点**、连接这些结点的**边**、以及这些结点的**标记**。
- 子树的根结点的标记可能不是文法的开始符号。
- 如果子树的根结点标记为非终结符号A，则可称该子树为**A-子树**。



# \* 子树与短语的关系

- 一棵子树的所有叶结点自左至右排列起来，形成此句型相对于该子树根的**短语**；
- 分析树中**只有父子两代**的子树的所有叶结点自左至右排列起来，形成此句型相对于该子树根的**直接短语**；
- 分析树中**最左边**的那棵只有父子两代的子树的所有叶结点自左至右排列起来，就是该句型的**句柄**。



# \* 符号串相关基本概念

## 字母表

- 符号的非空有限集合
- 典型的符号是字母、数字、各种标点和运算符等。

## 符号串

- 定义在某一字母表上
- 由该字母表中的符号组成的有限符号序列
- 同义词：句子、字

## 符号串有关的几个概念

### ■ 长度

- 符号串 $\alpha$ 的长度是指 $\alpha$ 中出现的符号的个数，记作 $|\alpha|$ 。
- 空串的长度为0，常用 $\varepsilon$ 表示。



# \* 符号串相关基本概念

## ■ 前缀

- 符号串 $\alpha$ 的前缀是指从符号串 $\alpha$ 的末尾删除0个或多个符号后得到的符号串。如：univ 是 university 的前缀。

## ■ 后缀

- 符号串 $\alpha$ 的后缀是指从符号串 $\alpha$ 的开头删除0个或多个符号后得到的符号串。如：sity 是 university 的后缀。

## ■ 子串

- 符号串 $\alpha$ 的子串是指删除了 $\alpha$ 的前缀和/或后缀后得到的符号串。如：ver 是 university 的子串。

## ■ 真前缀、真后缀、真子串

- 如果非空符号串 $\beta$ 是 $\alpha$ 的前缀、后缀或子串，并且 $\beta \neq \alpha$ ，则称 $\beta$ 是 $\alpha$ 的真前缀、真后缀、或真子串。

## ■ 子序列

- 符号串 $\alpha$ 的子序列是指从 $\alpha$ 中删除0个或多个符号(这些符号可以是不连续的)后得到的符号串。如：nvst

# \* 符号串相关基本概念

## ■ 连接

- 符号串 $\alpha$ 和符号串 $\beta$ 的连接 $\alpha\beta$ 是把符号串 $\beta$ 加在符号串 $\alpha$ 之后得到的符号串
- 若 $\alpha=ab$ ,  $\beta=cd$ , 则 $\alpha\beta=abcd$ ,  $\beta\alpha=cdba$ 。
- 对任何符号串 $\alpha$ 来说, 都有 $\varepsilon\alpha=\alpha\varepsilon=\alpha$

## ■ 幂

- 若 $\alpha$ 是符号串,  $\alpha$ 的 $n$ 次幂 $\alpha^n$ 定义为:  $\underbrace{\alpha\alpha \dots \alpha\alpha}_{n\text{个}}$
- 当 $n=0$ 时,  $\alpha^0$ 是空串,  $\alpha^0=\varepsilon$ 。

- 假如 $\alpha=ab$ , 则有:

$$\alpha^0=\varepsilon \quad \alpha^1=ab \quad \alpha^2=abab$$

.....

$$\alpha^n=\underbrace{abab\dots ab}_{n\text{个}ab}$$

# \* 二义性文法

- 如果一个文法的某个句子有不只一棵分析树，则这个句子是**二义性的**。
- 含有二义性句子的文法是**二义性的文法**。

例：考虑文法  $G = (\{+, *, (, ), i\}, \{E\}, E, \varphi)$

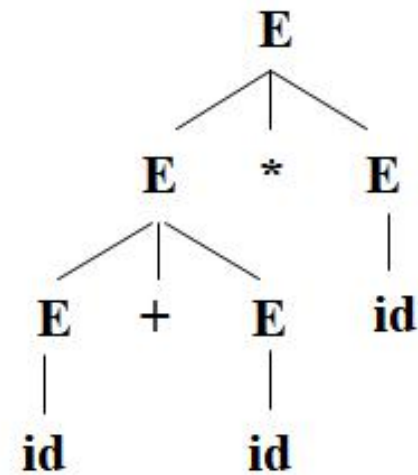
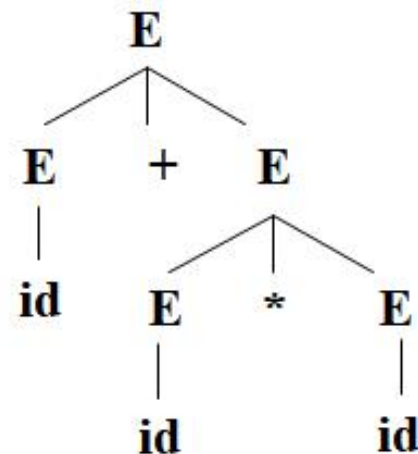
$\varphi: E \rightarrow E + E \mid E * E \mid (E) \mid id$

句子  $id + id * id$  存在两个不同的最左推导：

$E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + E * E \Rightarrow id + id * E \Rightarrow id + id * id$

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow id + E * E \Rightarrow id + id * E \Rightarrow id + id * id$

有两棵不同的分析树



# \* 文法的二义性和语言的二义性

- 如果两个文法产生的语言相同，即 $L(G)=L(G')$ ，则称这两个文法是等价的。
- 有时，一个二义性的文法可以变换为一个等价的、无二义性的文法。
- 有些语言，根本就不存在无二义性的文法，这样的语言称为二义性的语言。
- 二义性问题是不可判定的
  - 不存在一种算法，它能够在有限的步骤内确切地判定出一个文法是否是二义性的。
  - 可以找出一些充分条件（未必是必要条件），当文法满足这些条件时，就可以确信该文法是无二义性的。