



# 第4章 语法分析



重庆大学 葛亮

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

# § 4 语法分析

**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**

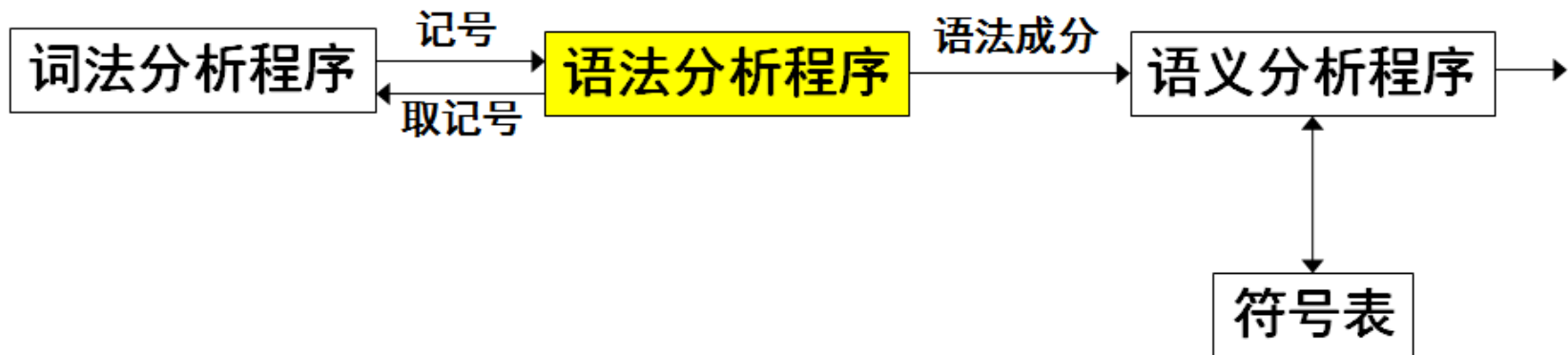
**4.4 LR分析方法**

**4.5 软件工具YACC**

**小结**

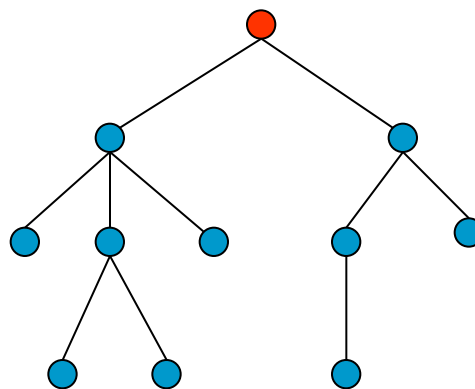
## 4.1 语法分析简介

- 语法分析是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：



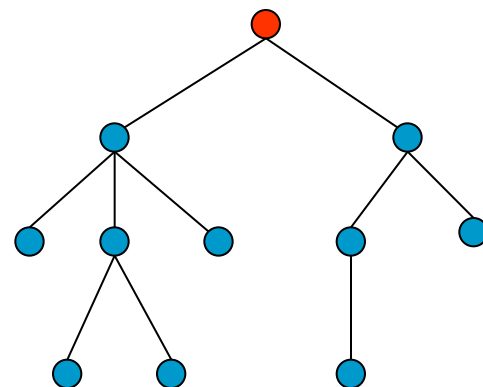
## ■ 语法分析程序

- 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、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

- 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、块结束标识END等。

### ◆ 短语级恢复

### ◆ 出错产生式

### ◆ 全局纠正

## 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，是反复使用不同产生式谋求匹配输入串的过程。
- 例：试分析输入串 $\omega = \text{abbcde}$ 是否为如下文法的一个句子

$S \rightarrow aAcBe$

$A \rightarrow b \mid Ab$

$B \rightarrow d$

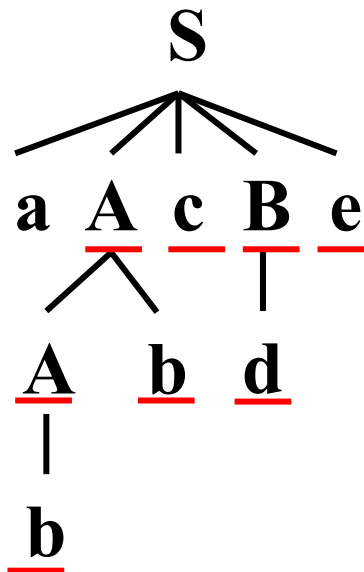
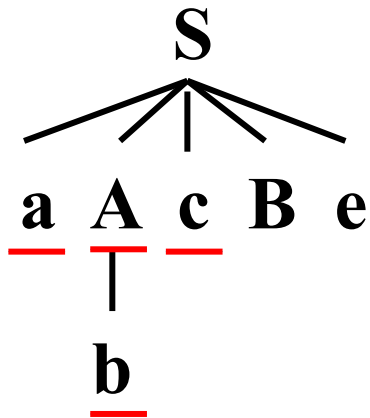
(文法4.1)

# 示例：abbcde的递归下降分析

a b b c d e \$  
          ↑

a b b c d e \$  
                  ↑

$S \rightarrow aAcBe$   
 $A \rightarrow b \mid Ab$   
 $B \rightarrow d$



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

试图为输入符号串建立一个最左推导序列的过程。



## ■ 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：

$A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_i | \dots | \alpha_n$

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

### 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:

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

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

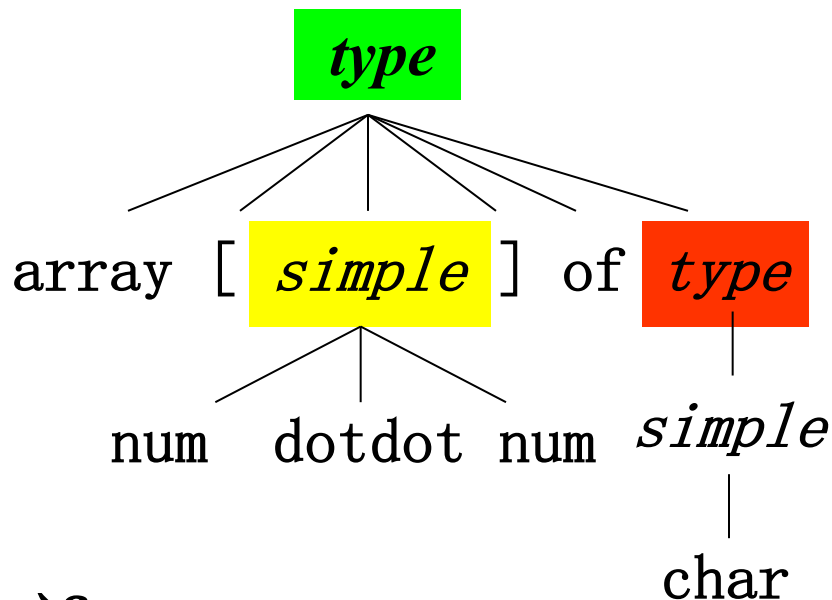
$FIRST(num \ dotdot \ num) = \{ num \}$

## 示例（续）

输入: array [num dotdot num] of char

↑                    ↑                    ↑

树:



产生式:  $A \rightarrow \varepsilon$

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  $\varepsilon$ -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

## ■ 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

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

(文法4.3)

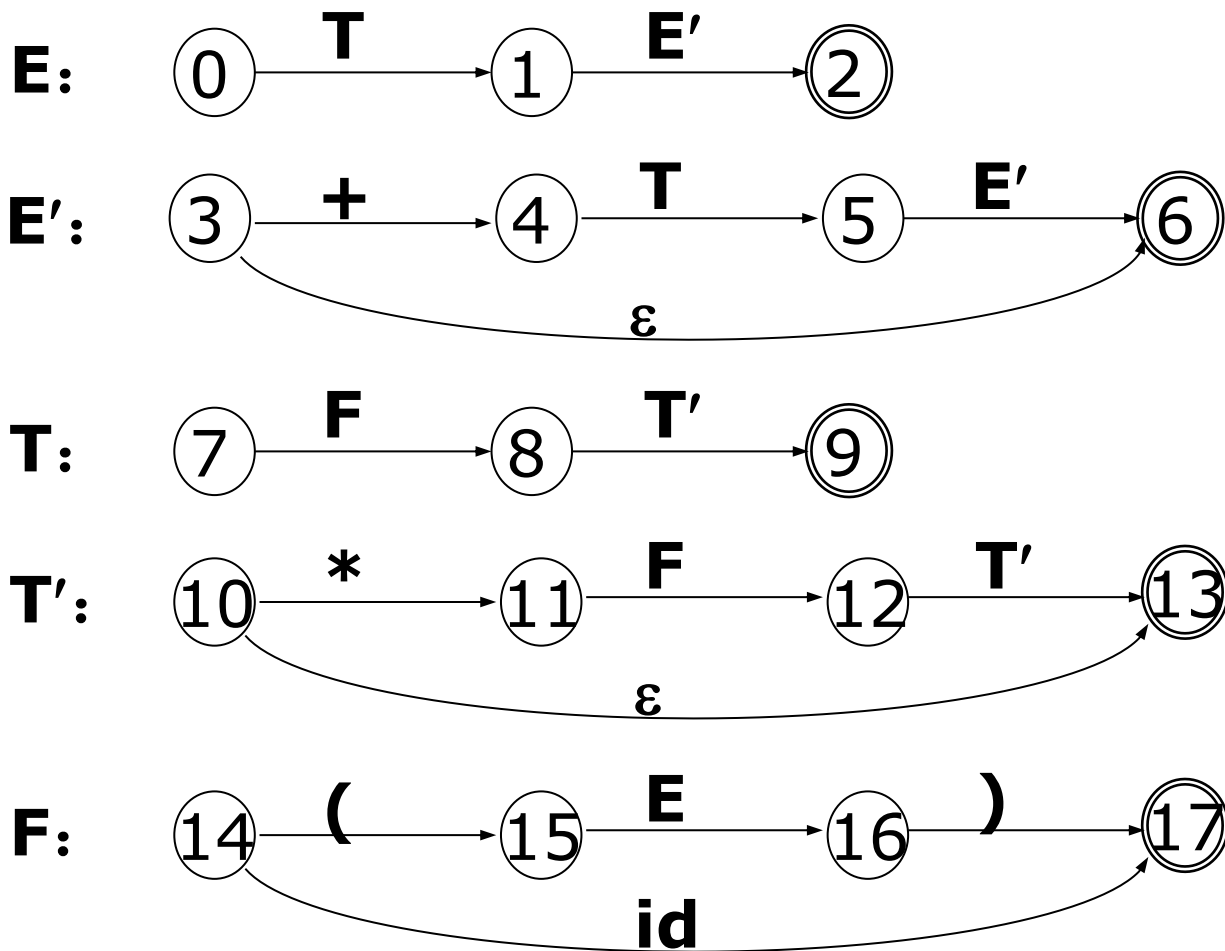
- 消除文法中存在的左递归, 得到

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

(文法4.4)

■ 为每个非终结符号构造转换图：

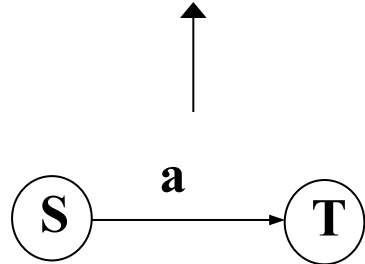
$E \rightarrow TE'$   
 $E' \rightarrow +TE' | \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' | \varepsilon$   
 $F \rightarrow (E) | id$



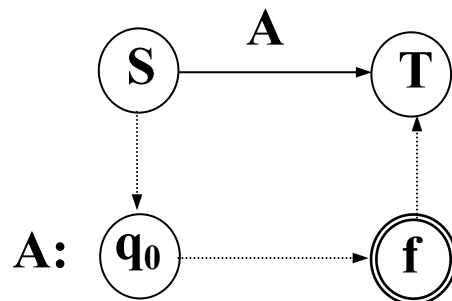
# 转换图的工作过程

- 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**

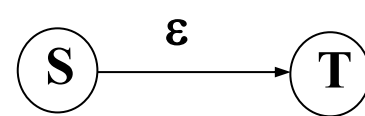
输入串: ... **a** ...



输入串: ... **a** ..... **b** ...

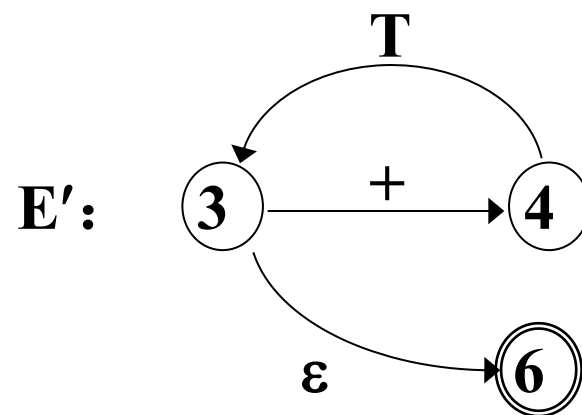
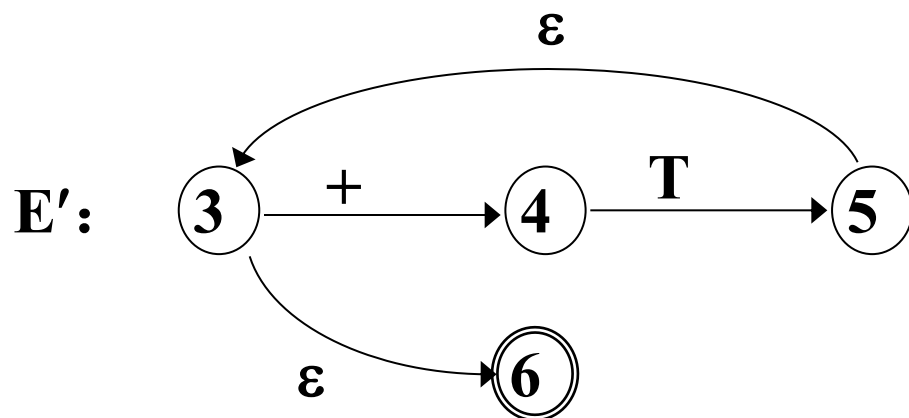
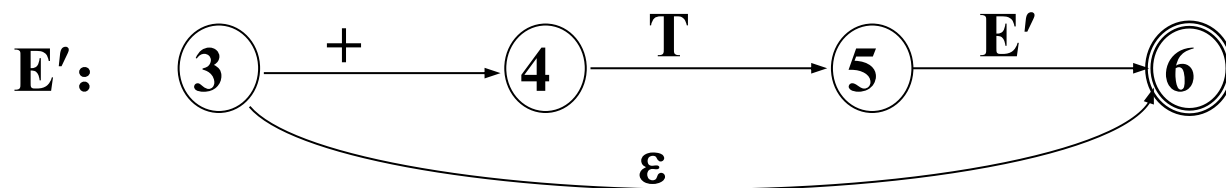


输入串: ... **a** ...

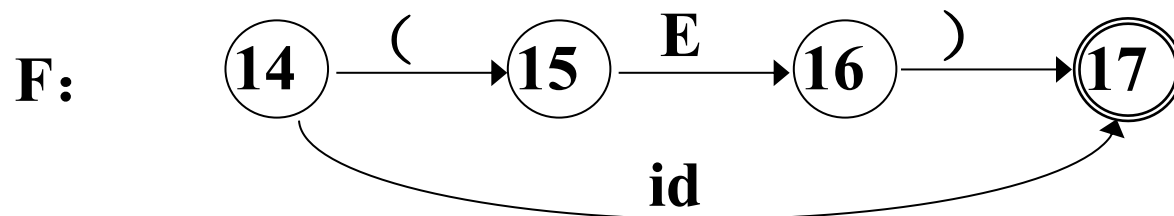
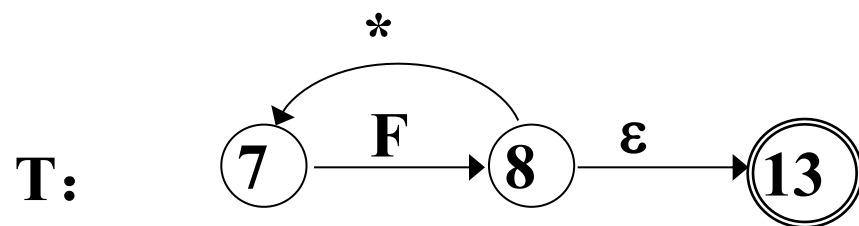
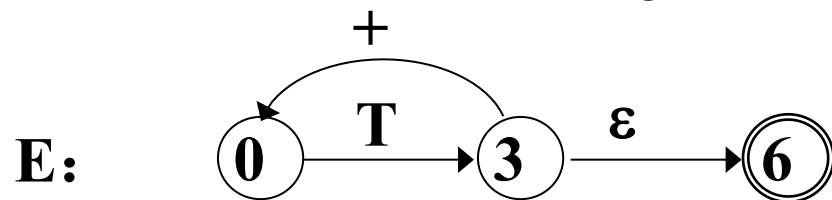
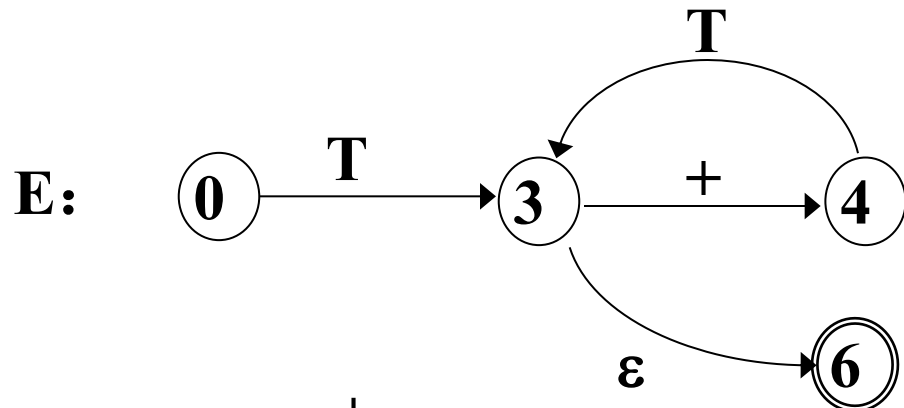


# 转换图的化简

## ■ 用代入的方法进行化简



## ■ 把E'的转换图代入E的转换图：

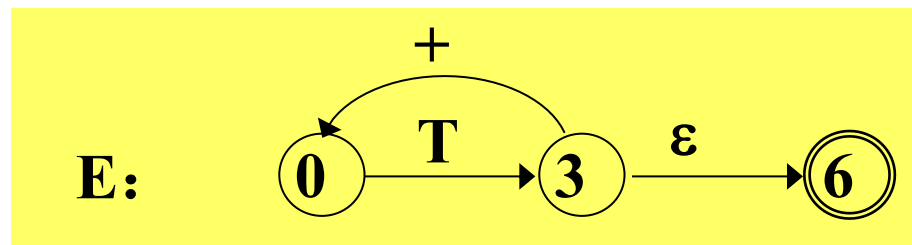


# 预测分析程序的实现

- 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:

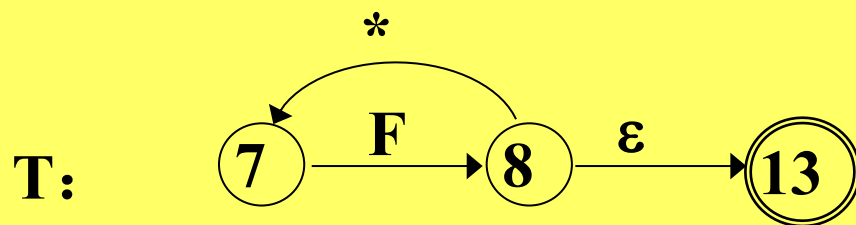
```
void procE(void)  
{  
    procT();  
    if (char=='+') {  
        forward pointer;  
        procE();  
    }  
}
```



# 预测分析程序的实现(续1)

## ■ T的过程:

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

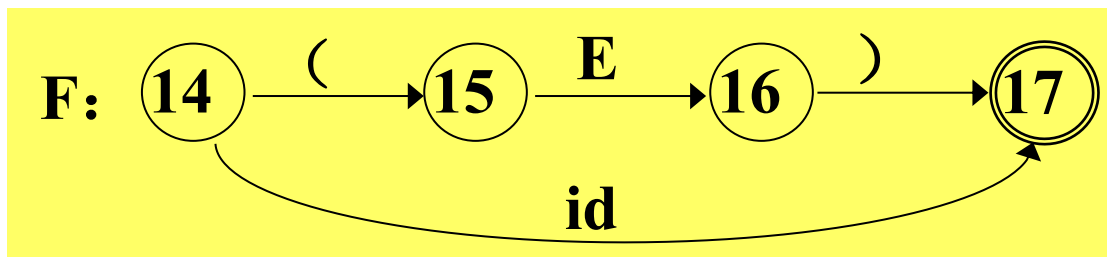




# 预测分析程序的实现（续2）

## ■ F的过程:

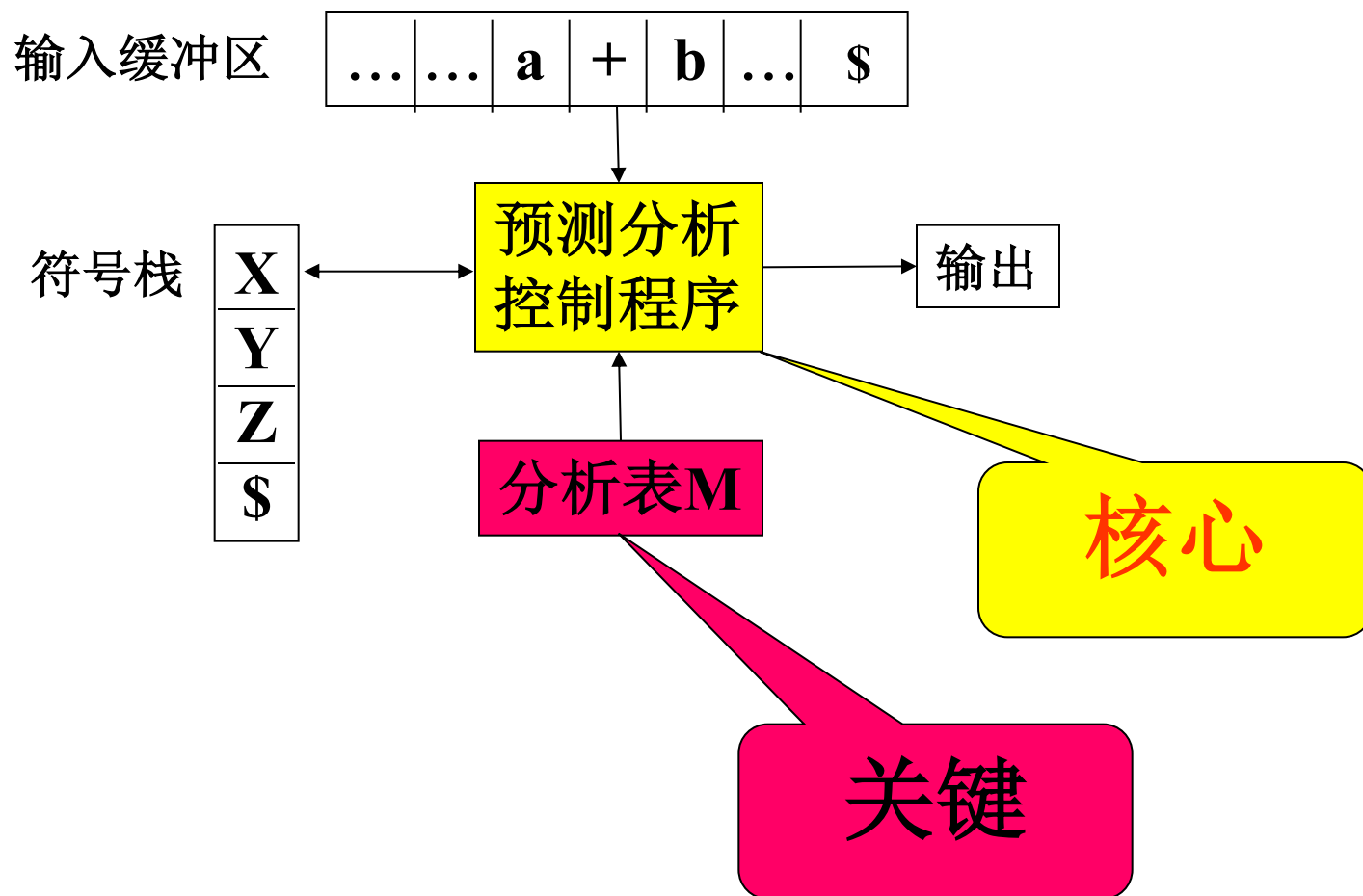
```
void procF(void)
{
    if (char=='(') {
        forward pointer;
        procE();
        if (char==')') {
            forward pointer;
        };
        else error();
    };
    else if (char=='id') {
        forward pointer;
        else error();
    }
}
```



## 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 $M[A, a]$ ， $A \in V_N$ ， $a \in V_T \cup \{\$ \}$ 。

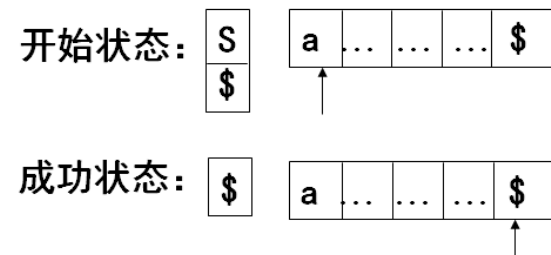
根据给定的 $A$ 和 $a$ ，在分析表 $M$ 中找到将被调用的产生式。

## ■ 输出流：

分析过程中不断产生的产生式序列。

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

Liang GE      CQU



- 29

## 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 ( $M[X,a]=X\rightarrow Y_1Y_2\cdots Y_k$ ) {  
            从栈顶弹出 $X$ ；  
            把 $Y_k$ 、 $Y_{k-1}$ 、 $\cdots$ 、 $Y_2$ 、 $Y_1$ 压入栈， $Y_1$ 在栈顶；  
            输出产生式 $X\rightarrow Y_1Y_2\cdots Y_k$ ；  
        };  
        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 $\alpha \in (V_T \cup V_N)^*$ ,

**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：**

$$\text{FOLLOW}(A) = \{ a \mid S \xRightarrow{*} \cdots Aa \cdots, a \in V_T \}$$

特别地，若 $S \xRightarrow{*} \cdots A$ ，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

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

	FIRST	FOLLOW
<b>E</b>	(, id	\$, )
<b>E'</b>	+, $\varepsilon$	\$, )
<b>T</b>	(, id	\$, ), +
<b>T'</b>	*, $\varepsilon$	\$, ), +
<b>F</b>	(, 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→ $\alpha$ ) {  
    for (每个终结符号a∈FIRST( $\alpha$ ))  
        把A→ $\alpha$ 放入M[A,a]中;  
    if ( $\epsilon$ ∈FIRST( $\alpha$ ))  
        for (任何b∈FOLLOW(A))  
            把A→ $\alpha$ 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

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

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



# 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 (续)

- 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

# 预测分析方法中的错误处理示例

- 分析过程中，有两种情况可以发现源程序中的语法错误：
  - (1)  $X \in V_T$ ，但 $X \neq a$ ；
  - (2)  $X \in V_N$ ，但 $M[X, a]$ 为空。
- 错误处理方法：
  - ◆ 第(1)种情况，弹出栈顶的终结符号；
  - ◆ 第(2)种情况，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

## ■ 带有同步化信息的分析表的构造

- ◆ 对于  $A \in V_N$ ,  $b \in FOLLOW(A)$ , 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

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

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

	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

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

- 帶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;

## 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

- 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（续）

## ■ 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 $A \rightarrow \cdots BC \cdots$ 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。

# 算符优先分析法

## ■ $G(E)$ :

$E \rightarrow EAE \mid (E) \mid id$

$A \rightarrow + \mid *$

## ■ $G'(E)$ :

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

## ■ $id+id*id+id$

	+	*	(	)	id
+	>	<	<		<
*	>	>	<		<
(				=	
)	>	>			
id	>	>			

# “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)  $A \rightarrow b$

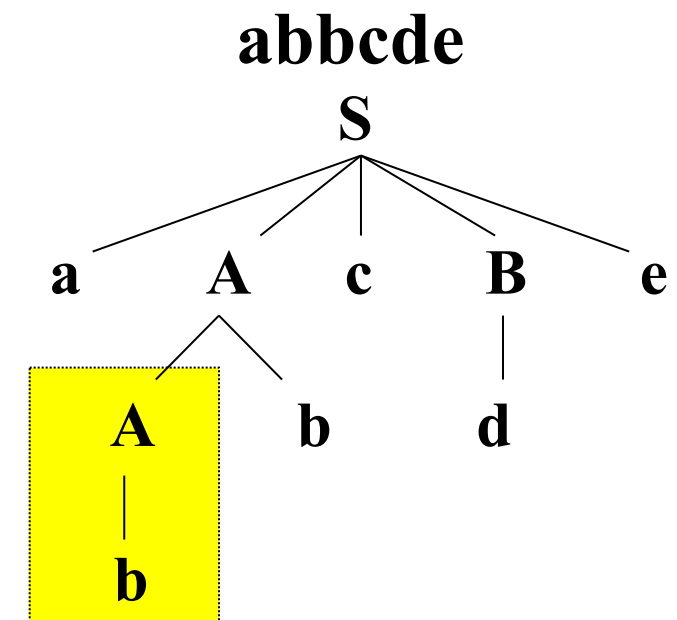
3)  $A \rightarrow Ab$

4)  $B \rightarrow d$  (文法4.1)

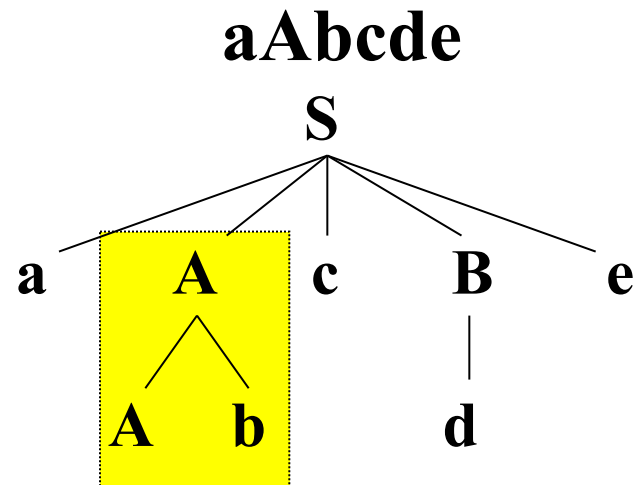
## ■ 最右推导

$$\begin{array}{ccccccc} & 1) & & 4) & & 3) & & 2) \\ S & \xRightarrow{\text{rm}} & aAcBe & \xRightarrow{\text{rm}} & aAcde & \xRightarrow{\text{rm}} & aAbcde & \xRightarrow{\text{rm}} & abbcde \end{array}$$

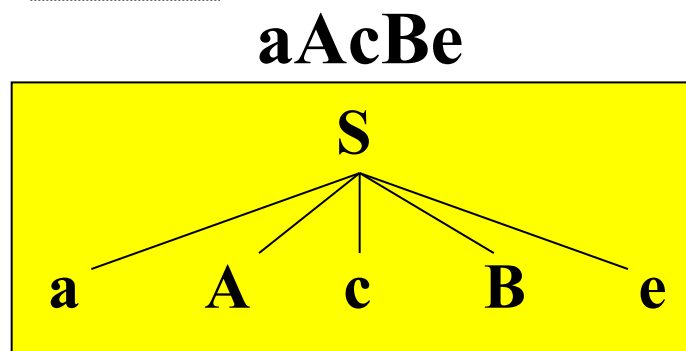
# 最右推导的逆过程



$\Rightarrow$   
(2)



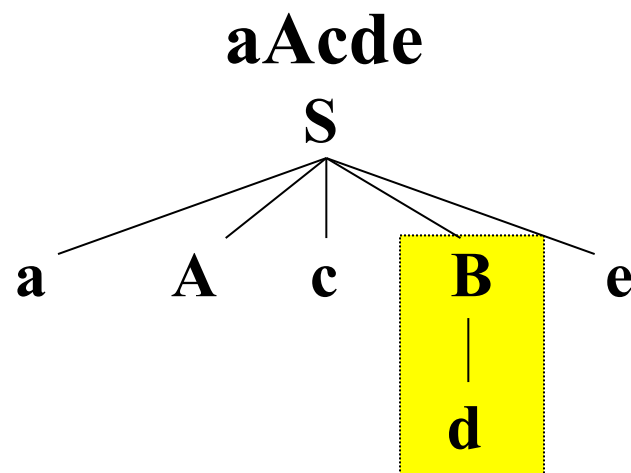
$\Downarrow$  (3)



$\Downarrow$  (1)

**S**

$\Leftarrow$   
(4)



# 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  $E \rightarrow E+E \mid E * E \mid (E) \mid id$  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**和一个存放输入符号串的**缓冲区**。
  - ◆ 分析开始时，先将符号\$入栈，以示栈底；
  - ◆ 将\$置入输入符号串之后，以示符号串的结束。



## 例：对文法4. 6的句子abbcede的规范归约过程

栈	输入	分析动作
1) \$	abbcede\$	shift
2) \$a	bbcede\$	shift
3) \$ab	bcde\$	reduce by $A \rightarrow b$
4) \$aA	bcde\$	shift
5) \$aAb	cde\$	reduce by $A \rightarrow Ab$
6) \$aA	cde\$	shift
7) \$aAc	de\$	shift
8) \$aAcd	e\$	reduce by $B \rightarrow d$
9) \$aAcB	e\$	shift
10) \$aAcBe	\$	reduce by $S \rightarrow aAcBe$
11) \$S	\$	accept

$S \rightarrow aAcBe$   
 $A \rightarrow b$   
 $A \rightarrow Ab$   
 $B \rightarrow d$

# “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(续)

## ■ LR分析技术是一种比较完备的技术

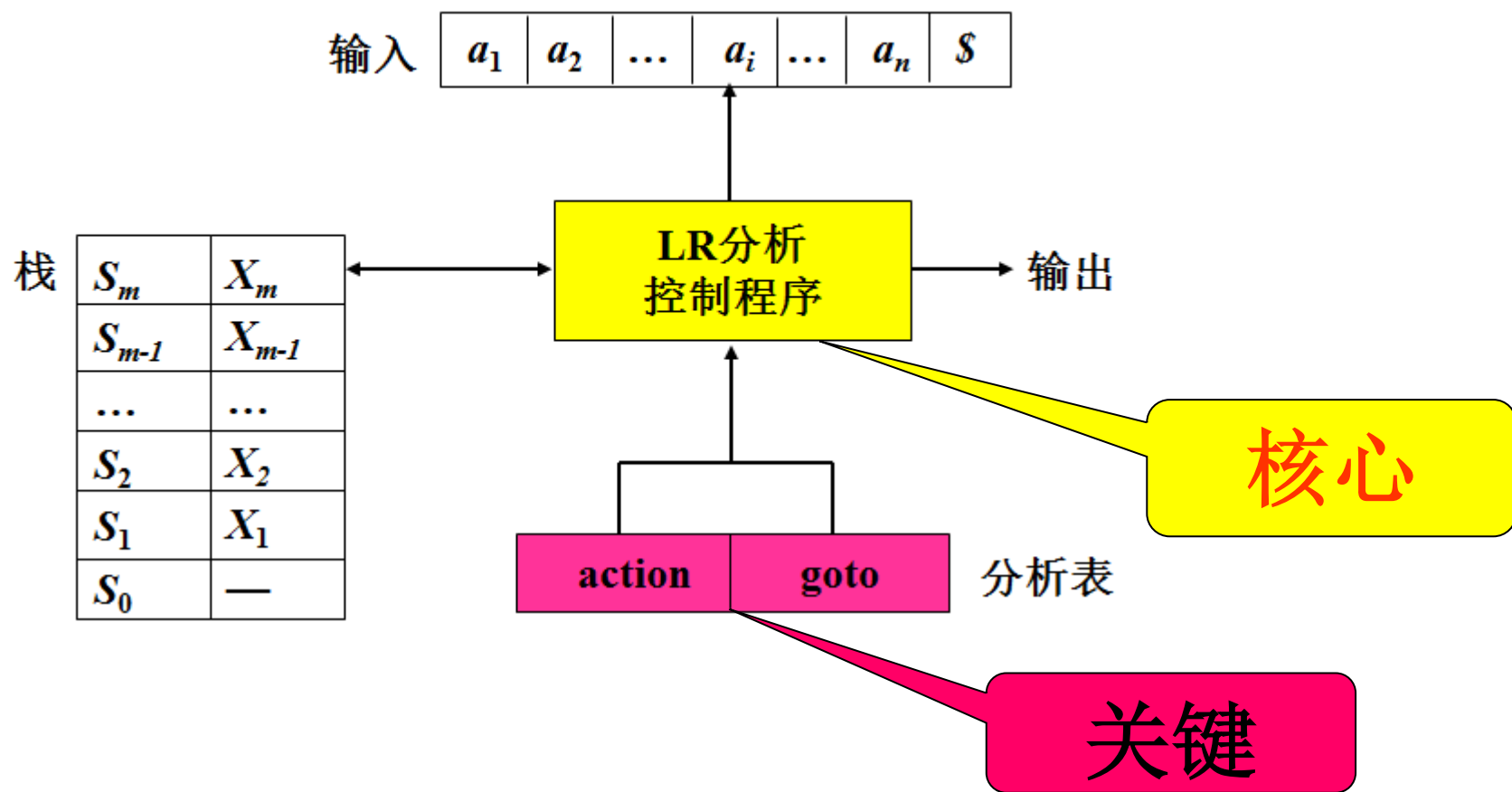
- ◆ 可以分析所有能用上下文无关文法书写的程序设计语言的结构；
- ◆ 最一般的无回溯的“移进-归约”方法；
- ◆ 分析过程中，能及时发现错误。

## ■ 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
- **goto[ $S_m$ ,  $X$ ]**: 状态 $S_m$ 经 $X$ 转移的后继状态
- **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
  - ◆ **移进**: 把当前输入符号 $a_i$ 及由 $S_m$ 和 $a_i$ 所决定的下一个状态 $S = \text{goto}[S_m, a_i]$ 推进栈, 向前扫描指针前移。
  - ◆ **归约**: 用某产生式 $A \rightarrow \beta$ 进行归约, 若 $\beta$ 的长度为 $r$ , 归约的动作从栈顶起向下弹出 $r$ 项, 使 $S_{m-r}$ 成为栈顶状态, 然后把文法符号 $A$ 及状态 $S = \text{goto}[S_{m-r}, A]$ 推进栈。
  - ◆ **接受**: 宣布分析成功, 停止分析。
  - ◆ **出错**: 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： $(S_0, a_1a_2\cdots a_n\$)$
  - ◆ 分析过程中每步的结果，均可表示为如下的二元式： $(S_0S_1S_2\cdots S_m, a_ia_{i+1}\cdots a_n\$)$
  - ◆ 若 $\text{action}[S_m, a_i] = \text{shift } S$ ，且 $S = \text{goto}[S_m, a_i]$ ，  
则二元式变为： $(S_0S_1\cdots S_mS, a_{i+1}\cdots a_n\$)$
  - ◆ 若 $\text{action}[S_m, a_i] = \text{reduce by } A \rightarrow \beta$ ，  
则二元式变为： $(S_0S_1\cdots S_{m-r}S, a_ia_{i+1}\cdots a_n\$)$
  - ◆ 若 $\text{action}[S_m, a_i] = \text{accept}$ （接受）  
分析成功，二元式变化过程终止。
  - ◆ 若 $\text{action}[S_m, a_i] = \text{error}$ （出错）  
发现错误，调用错误处理程序。

# “活前缀”的概念

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  $(S_0X_1S_1X_2\cdots X_mS_m, a_ia_{i+1}\cdots a_n\$)$

$X_1X_2\cdots X_ma_ia_{i+1}\cdots a_n$ 是一个右句型，

$X_1X_2\cdots X_m$ 是它的一个活前缀。

## 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 ( $\text{action}[S, a] == \text{shift } S'$ ) {

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

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

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

        输出产生式 $A \rightarrow \beta$ ；

    };

    else if ( $\text{action}[S, a] == \text{accept}$ ) return;

    else error();

} while(1).

# 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
id		
0 3	+id*id\$	reduce by $T \rightarrow F$ goto[0,T]=2
F		
0 2	+id*id\$	reduce by $E \rightarrow T$ goto[0,E]=1
T		
0 1	+id*id\$	shift 6
E		
0 1 6	id*id\$	shift 5
E +		
0 1 6 5	*id\$	reduce by $F \rightarrow id$ goto[6,F]=3
E + id		
0 1 6 3	*id\$	reduce by $T \rightarrow F$ goto[6,T]=9
E + F		

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

	id	+	*	(	)	\$	E	T	F
0	S5			S4			1	2	3
1		S6				AC C			
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			S1 1				
9		R1			R1	R1			
10		R3			R3	R3			
11		R5			R5	R5			

# 示例：对输入符号串 $id+id*id$ 的分析过程（续）

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

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

$A \rightarrow XYZ \cdot$

} 移进-待约项目

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  $A \rightarrow \cdot$ 。



# 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  $S' \rightarrow S \cdot$ )

## 定义4.10: LR(0)有效项目

项目 $A \rightarrow \beta_1 \cdot \beta_2$ 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 $A \rightarrow \alpha \cdot B \beta$ 对活前缀 $\gamma = \delta \alpha$ 是有效的, 并且 $B \rightarrow \eta$ 是一个产生式, 则项目 $B \rightarrow \cdot \eta$ 对活前缀 $\gamma = \delta \alpha$ 也是有效的。
- LR(0)项目 $S' \rightarrow \cdot S$ 是活前缀 $\epsilon$ 的有效项目  
(这里取 $\gamma = \epsilon$ ,  $\beta_1 = \epsilon$ ,  $\beta_2 = S$ ,  $A = S'$ )

# 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 $A \rightarrow \alpha \cdot B \beta$ 属于 $\text{closure}(I)$ ，  
且 $B \rightarrow \eta$ 是 $G$ 的一个产生式，  
若 $B \rightarrow \cdot \eta$ 不属于 $\text{closure}(I)$ ，  
则将 $B \rightarrow \cdot \eta$ 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 $\text{closure}(I)$ 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 $A\rightarrow\alpha\cdot B\beta$   
和文法 $G$ 的每个产生式 $B\rightarrow\eta$ )

if ( $B\rightarrow\cdot\eta\notin J$ ) 把 $B\rightarrow\cdot\eta$ 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

$$go(I, X) = closure(J)$$

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

$go(I, X)$ 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 = {closure({ $S' \rightarrow \cdot S$ })};**

**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 $\text{closure}(\{S' \rightarrow \cdot S\})$ 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:

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

- 拓广文法G':

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

- 活前缀 $\epsilon$ 的有效项目集

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

活前缀

- 从 $I_0$ 出发的转移有

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

**$I_2 = \text{go}(I_0, a) = \text{closure}(\{S \rightarrow a \cdot A\})$**   
 **$= \{S \rightarrow a \cdot A, A \rightarrow \cdot cA, A \rightarrow \cdot d\}$**    **——a**

**$I_3 = \text{go}(I_0, b) = \text{closure}(\{S \rightarrow b \cdot B\})$**   
 **$= \{S \rightarrow b \cdot B, B \rightarrow \cdot cB, B \rightarrow \cdot d\}$**    **——b**



## 示例（续1）：

### ■ 从 $I_2$ 出发的转移有

活前缀

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

$$\begin{aligned} I_5 &= \text{go}(I_2, c) = \text{closure}(\{A \rightarrow c \cdot A\}) \\ &= \{A \rightarrow c \cdot A, A \rightarrow \cdot cA, A \rightarrow \cdot d\} \quad - ac \end{aligned}$$

$$I_6 = \text{go}(I_2, d) = \text{closure}(\{A \rightarrow d \cdot\}) = \{A \rightarrow d \cdot\} \quad - ad$$

### ■ 从 $I_3$ 出发的转移有

活前缀

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

$$\begin{aligned} I_8 &= \text{go}(I_3, c) = \text{closure}(\{B \rightarrow c \cdot B\}) \\ &= \{B \rightarrow c \cdot B, B \rightarrow \cdot cB, B \rightarrow \cdot d\} \quad - bc \end{aligned}$$

$$I_9 = \text{go}(I_3, d) = \text{closure}(\{B \rightarrow d \cdot\}) = \{B \rightarrow d \cdot\} \quad - bd$$

## 示例（续2）：

### ■ 从 $I_5$ 出发的转移有

$I_{10} = \text{go}(I_5, A) = \text{closure}(\{A \rightarrow cA \cdot\}) = \{A \rightarrow cA \cdot\}$  活前缀 —acA

$\text{go}(I_5, c) = \text{closure}(\{A \rightarrow c \cdot A\}) = I_5$  —acc

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

### ■ 从 $I_8$ 出发的转移有

$I_{11} = \text{go}(I_8, B) = \text{closure}(\{B \rightarrow cB \cdot\}) = \{B \rightarrow cB \cdot\}$  活前缀 —bcB

$\text{go}(I_8, c) = \text{closure}(\{B \rightarrow c \cdot B\}) = I_8$  —bcc

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

### ■ 文法 $G'$ 的LR(0)项目集规范族

$$C = \{I_0, I_1, \dots, I_{11}\}$$

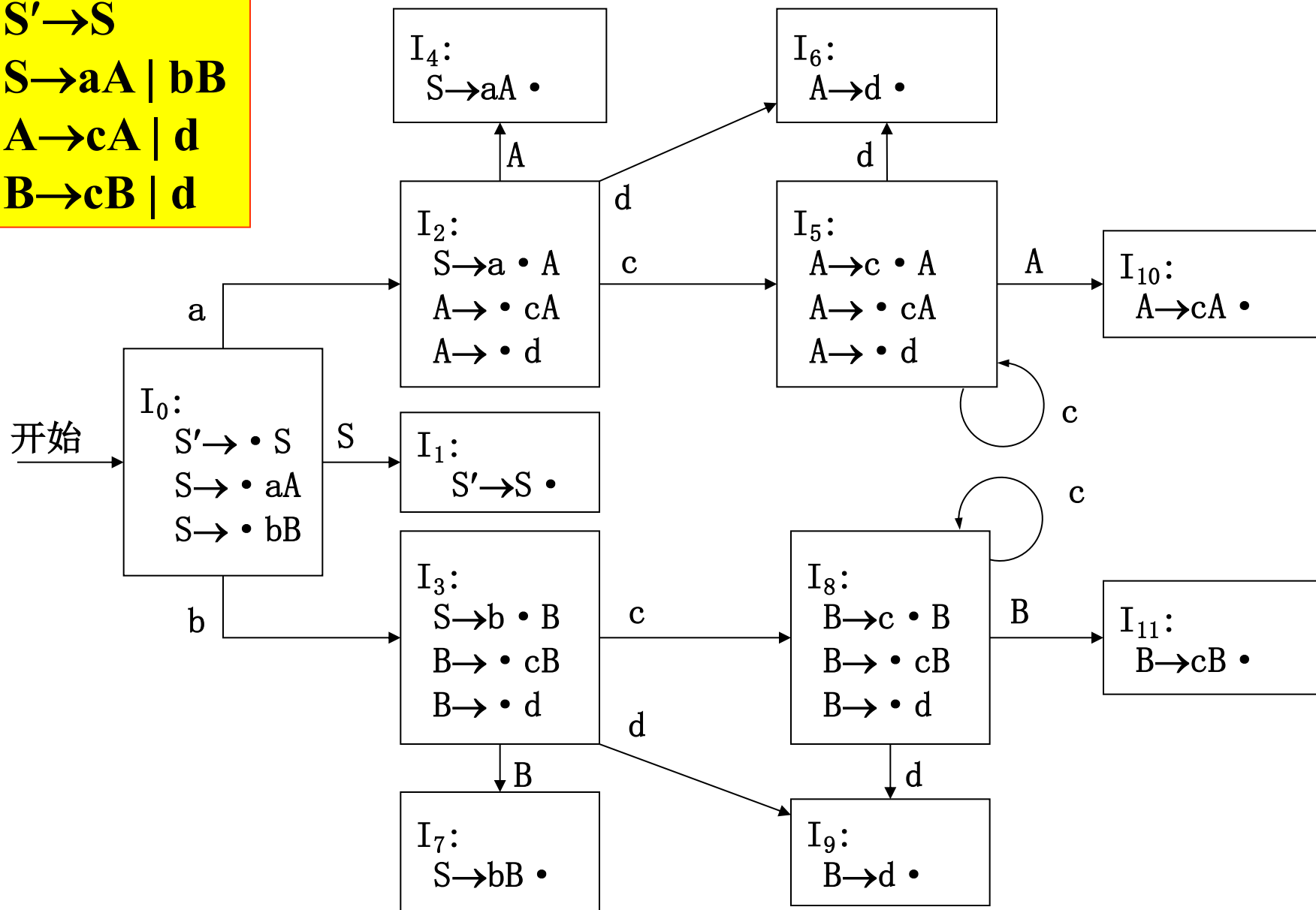
# 识别文法G'的所有活前缀的DFA

$S' \rightarrow S$

$S \rightarrow aA \mid bB$

$A \rightarrow cA \mid d$

$B \rightarrow cB \mid d$



# 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。

$$A \rightarrow (A) | a$$

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

- ◆ 存在移进-归约冲突
- ◆ 存在归约-归约冲突

## ■ 冲突的解决方法，查看FOLLOW(A)和FOLLOW(B)

- ◆  $FOLLOW(A) \cap FOLLOW(B) = \Phi$
- ◆  $b \notin FOLLOW(A)$  并且  $b \notin FOLLOW(B)$
- ◆ 决策：
  - 当 $a=b$ 时，把 $b$ 移进栈里；
  - 当 $a \in FOLLOW(A)$ 时，用产生式 $A \rightarrow \alpha$ 进行归约；
  - 当 $a \in FOLLOW(B)$ 时，用产生式 $B \rightarrow \alpha$ 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 $A \rightarrow \alpha \cdot a \beta \in I_i$ ，且 $go(I_i, a) = I_j$ ，则置  
 $action[i, a] = sj$
  - b) 若 $A \rightarrow \alpha \cdot \in I_i$ ，则对所有 $a \in FOLLOW(A)$ ，置  
 $action[i, a] = r A \rightarrow \alpha$
  - c) 若 $S' \rightarrow S \cdot \in I_i$ ，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  $I_0 = \{ S' \rightarrow \cdot S, S \rightarrow \cdot aA, S \rightarrow \cdot bB \}$

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

- ◆ 对项目  $S' \rightarrow \cdot S$ , 有  $go(I_0, S) = I_1$ , 所以  $goto[0, S] = 1$
- ◆ 对项目  $S \rightarrow \cdot aA$ , 有  $go(I_0, a) = I_2$ , 所以  $action[0, a] = s2$
- ◆ 对项目  $S \rightarrow \cdot bB$ , 有  $go(I_0, b) = I_3$ , 所以  $action[0, b] = s3$

- 考察  $I_1 = \{ S' \rightarrow S \cdot \}$

- ◆ 项目  $S' \rightarrow S \cdot$  是接受项目, 所以  $action[1, \$] = acc$

- 考察  $I_2 = \{ S \rightarrow a \cdot A, A \rightarrow \cdot cA, A \rightarrow \cdot d \}$

- ◆ 对项目  $S \rightarrow a \cdot A$ , 有  $go(I_2, A) = I_4$ , 所以  $goto[2, A] = 4$
- ◆ 对项目  $A \rightarrow \cdot cA$ , 有  $go(I_2, c) = I_5$ , 所以  $action[2, c] = s5$
- ◆ 对项目  $A \rightarrow \cdot d$ , 有  $go(I_2, d) = I_6$ , 所以  $action[2, d] = s6$

■ 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, 因为  $FOLLOW(S) = \{\$ \}$ , 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, 因为  $FOLLOW(A) = \{\$ \}$ , 所以  $action[6, \$] = r4$



# 文法4. 6的SLR(1) 分析表

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

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

示例：

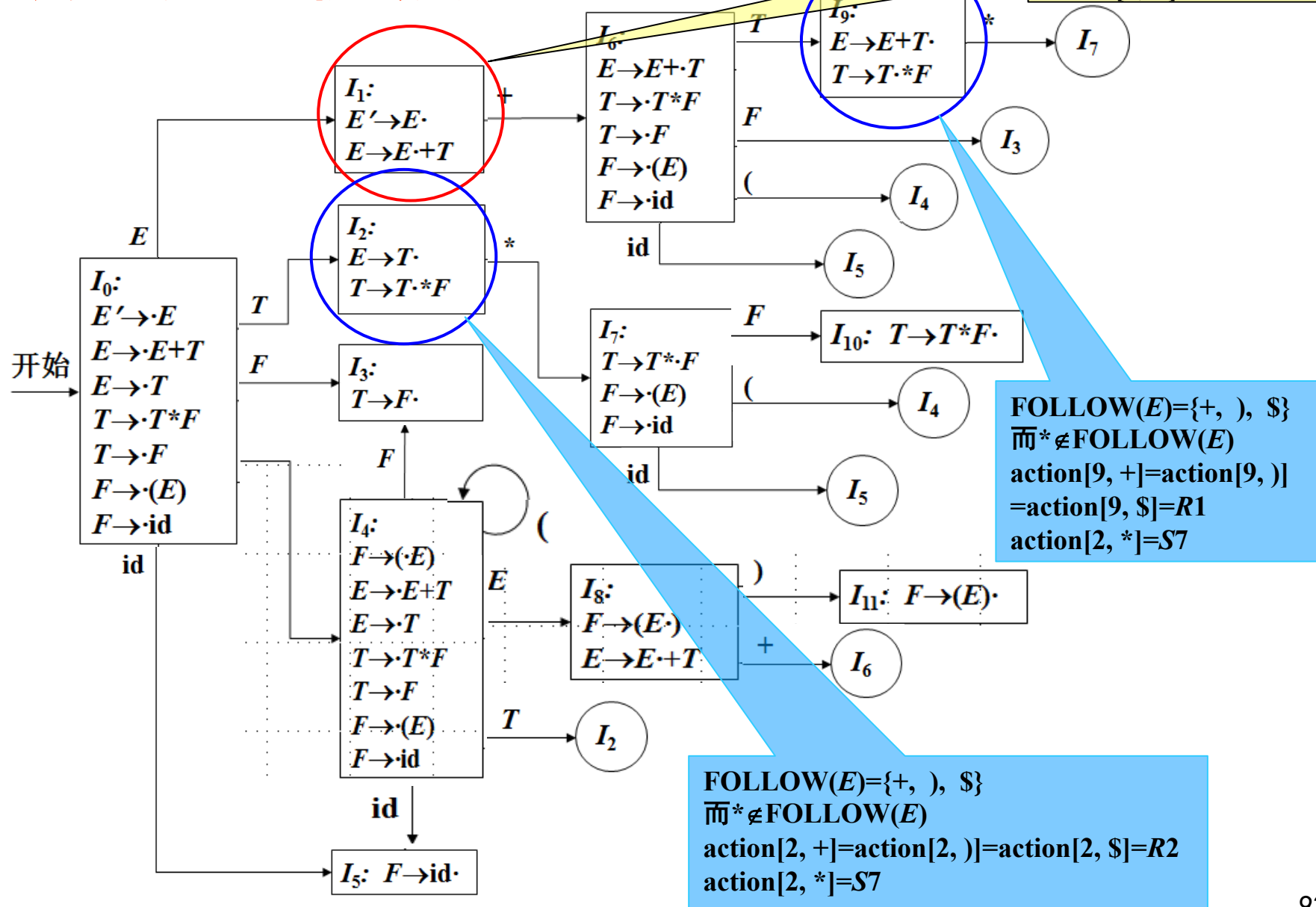
判断文法4.3是LR(0)文法，还是SLR(1)文法。

■ 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) $F \rightarrow id$ |                           |

■ 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



# 课堂练习

- 考虑如下文法G:

$$E \rightarrow (L) \mid a$$
$$L \rightarrow EL \mid E$$

(1) 构造该文法的SLR(1)分析表

(2) 给出对输入符号串(aa(a))的移进-归约分析动作

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)

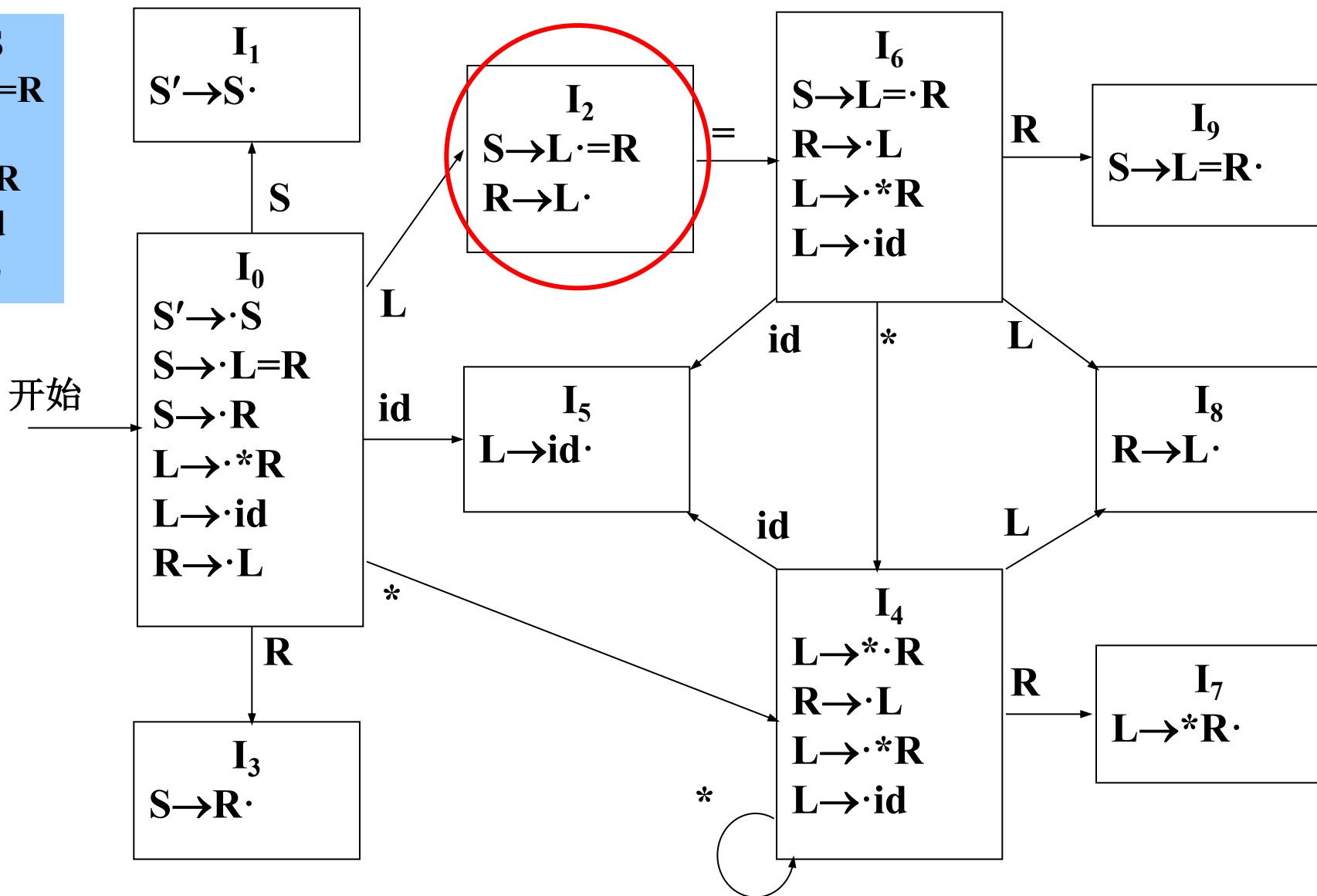
- 该文法无二义性，但不是**SLR(1)**文法。
- 拓广文法**G'**的产生式如下：

(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

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



# 应用算法4.5，为之构造SLR分析表

## ■ 考察 $I_2$ :

- ◆ 项目 $S \rightarrow L \cdot = R$ 与 $R \rightarrow L \cdot$ 存在移进-归约冲突,
- ◆ 根据第一个项目, 有:  $action[2, =] = s6$
- ◆ 根据第二个项目, 由于 $OLLOW(R) = \{=, \$\}$   
所以有:  $action[2, \$] = action[2, =] = r5$

## ■ 存在“移进-归约”冲突。

## ■ 原因是在SLR分析表中未包含足够多的预测信息。



## 4.4.3 LR(1)分析表的构造

- **LR(k)项目**:  $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \dots a_k]$ 
  - ◆  $A \rightarrow \alpha \cdot \beta$  是一个 **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  $[A \rightarrow \alpha \cdot, a_1 a_2 \dots a_k]$  起作用, 对任何移进或待约项目  $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \dots a_k], (\beta \neq \varepsilon)$  是没有意义的。
- 归约项目  $[A \rightarrow \alpha \cdot, a_1 a_2 \dots a_k]$  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, 才允许把栈顶的文法符号串  $\alpha$  归约为  $A$ 。

## 定义4.13: LR(1)有效项目

称一个LR(1)项目  $[A \rightarrow \alpha \cdot \beta, a]$  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  $[A \rightarrow \alpha \cdot B \beta, a]$  对活前缀  $\gamma = \delta \alpha$  是有效的, 并且  $B \rightarrow \eta$  是一个产生式, 则对任何  $b \in \text{FIRST}(\beta a)$ , 项目  $[B \rightarrow \cdot \eta, b]$  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  $b = a$ 。
- LR(1) 项目  $[S' \rightarrow \cdot S, \$]$  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 $[A \rightarrow \alpha \cdot B \beta, a]$ 属于 $\text{closure}(I)$ ，且 $B \rightarrow \eta$ 是 $G$ 的一个产生式，则对任何终结符号 $b \in \text{FIRST}(\beta a)$ ，若项目 $[B \rightarrow \cdot \eta, b]$ 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 $\text{closure}(I)$ 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 $[A \rightarrow \alpha \cdot B \beta, a]$  和  
文法 $G$ 的每个产生式 $B \rightarrow \eta$  )

for ( $\text{FIRST}(\beta a)$ 中的每一个终结符号 $b$ )

if ( $[B \rightarrow \cdot \eta, b] \notin J$ ) 把 $[B \rightarrow \cdot \eta, b]$ 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：

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

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

项目 $[A \rightarrow \alpha \mathbf{X} \cdot \beta, a]$ 称为 $[A \rightarrow \alpha \cdot \mathbf{X} \beta, a]$ 的后继。

直观含义：

若**I**是某个活前缀  $\gamma$  的有效项目集，

则  $\text{go}(\mathbf{I}, \mathbf{X})$  便是对活前缀  $\gamma \mathbf{X}$  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 = {closure({[S' → ·S, \$]})};**

**do**

**for (C中的每一个项目集I和每一个文法符号X)**

**if (go(I,X)不为空, 且不在C中)**

**把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。

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

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

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

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

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

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

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

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



## 示例（续）

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

$$\begin{aligned} I_6 &= \text{go}(I_2, c) = \text{closure}(\{[C \rightarrow c \cdot C, \$]\}) \\ &= \{[C \rightarrow c \cdot C, \$] \quad [C \rightarrow \cdot cC, \$] \quad [C \rightarrow \cdot d, \$]\} \end{aligned}$$

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

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

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

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

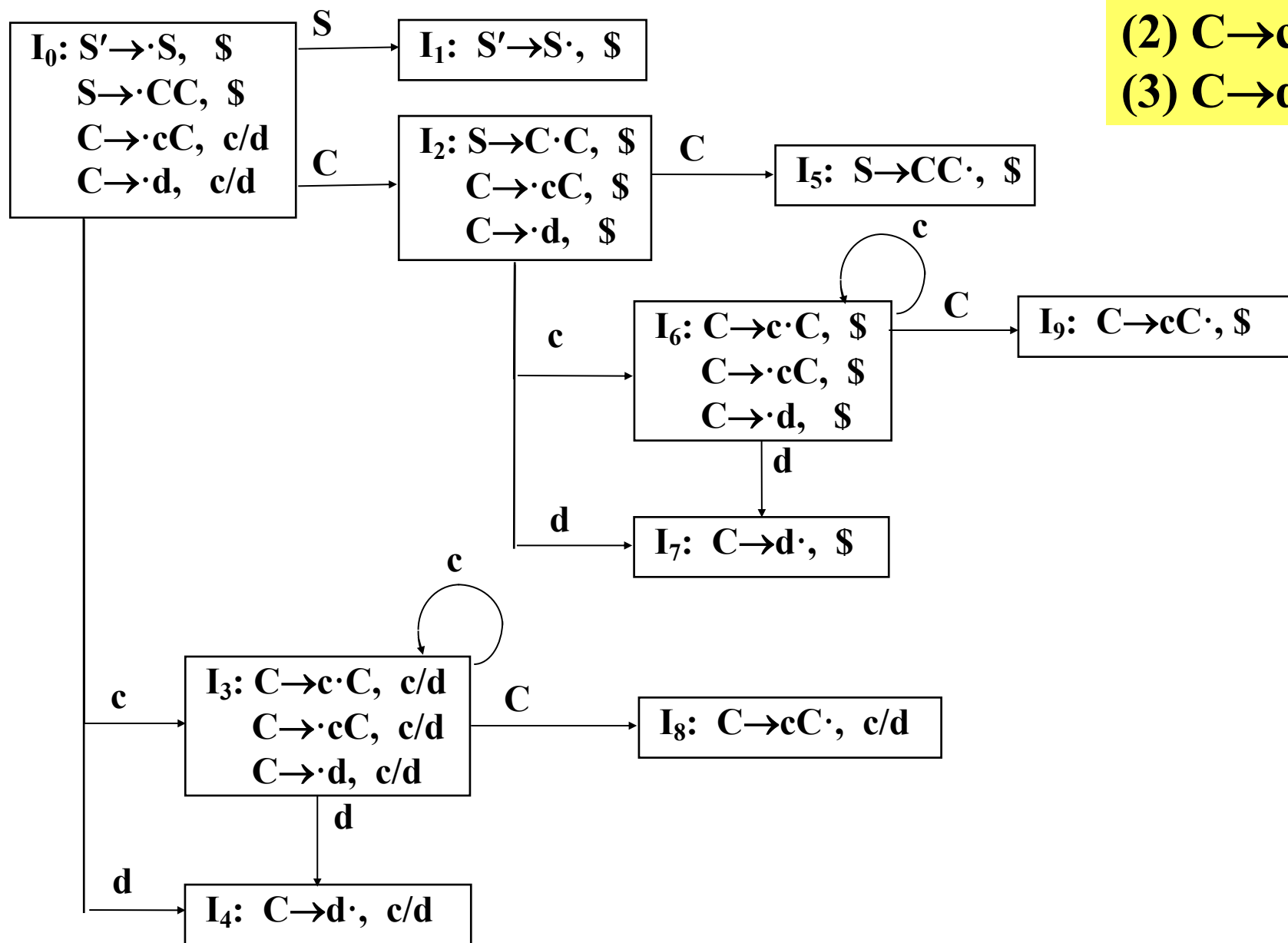
$$I_9 = \text{go}(I_6, C) = \text{closure}(\{[C \rightarrow cC \cdot, \$]\}) = \{[C \rightarrow cC \cdot, \$]\}$$

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

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

# 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 $[A \rightarrow \alpha \cdot a \beta, b] \in I_i$ ，且 $go(I_i, a) = I_j$ ，则置 $action[i, a] = sj$
  - b) 若 $[A \rightarrow \alpha \cdot, a] \in I_i$ ，且 $A \neq S'$ ，则置 $action[i, a] = rj$
  - c) 若 $[S' \rightarrow S \cdot, \$] \in I_i$ ，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 $[S' \rightarrow \cdot s, \$]$ 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 $I_0$ :

- ◆  $[S' \rightarrow \cdot S, \$]$  且  $go(I_0, S) = I_1$ , 故:  $goto[0, S] = 1$
- ◆  $[S \rightarrow \cdot CC, \$]$   $go(I_0, C) = I_2$   $goto[0, C] = 2$
- ◆  $[C \rightarrow \cdot cC, c/d]$   $go(I_0, c) = I_3$   $action[0, c] = s3$
- ◆  $[C \rightarrow \cdot d, c/d]$   $go(I_0, d) = I_4$   $action[0, d] = s4$

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

## ■ 考察 $I_2$ :

- ◆  $[S \rightarrow C \cdot C, \$]$  且  $go(I_2, C) = I_5$  故:  $goto[2, C] = 5$
- ◆  $[C \rightarrow \cdot cC, \$]$   $go(I_2, c) = I_6$   $action[2, c] = s6$
- ◆  $[C \rightarrow \cdot d, \$]$   $go(I_2, d) = I_7$   $action[2, d] = s7$

## ■ 考察 $I_4$ :

- ◆  $[C \rightarrow d \cdot, c/d]$  故  $action[4, c] = action[4, d] = r3$

# 文法4. 8的LR(1) 分析表

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 $G'$

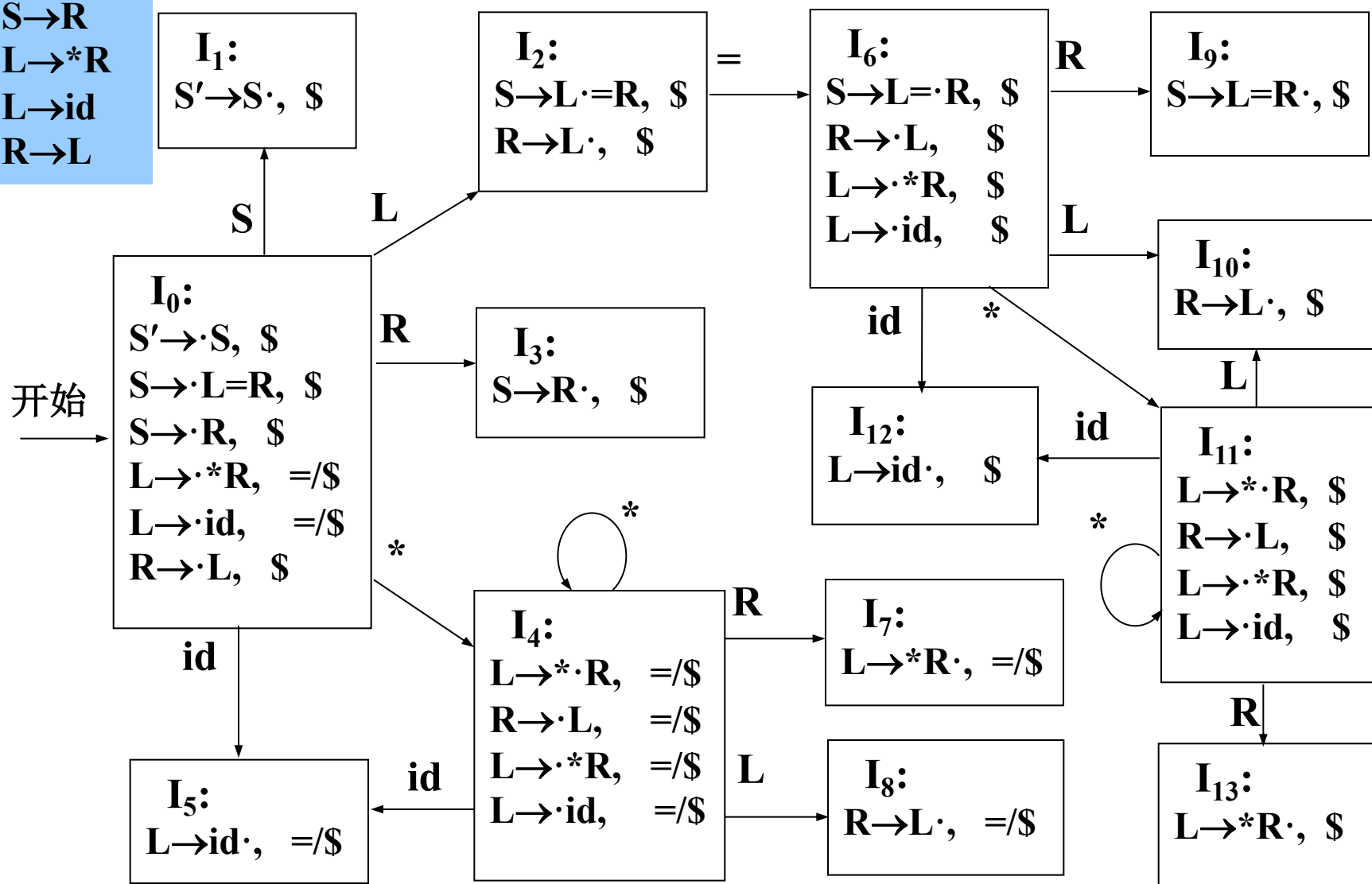
(0)  $S' \rightarrow S$     (1)  $S \rightarrow L = R$     (2)  $S \rightarrow R$

(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

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

## 示例 (续) :



# 文法4. 7的LR(1) 分析表

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	
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			



# 练习

■ 考虑如下文法G:

$$S \rightarrow A$$

$$A \rightarrow BA \mid \varepsilon$$

$$B \rightarrow aB \mid b$$

- (1) 证明该文法是LR(1)文法;
- (2) 构造该文法的LR(1)分析表;
- (3) 给出对于输入符号串abab的分析过程。

## 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** 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** 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  $[S' \rightarrow \cdot S, \$]$ 。

# 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  $[A \rightarrow \alpha \cdot, a]$  和  $[B \rightarrow \beta \cdot a \gamma, b]$  处于合并后的同一项目集中

合并前必存在某个  $c$ ，使得  $[A \rightarrow \alpha \cdot, a]$  和  $[B \rightarrow \beta \cdot a \gamma, c]$  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**：

**$S \rightarrow aAd \mid bBd \mid aBe \mid bAe$**

**$A \rightarrow c$**

**$B \rightarrow c$**             (文法4.9)

## ■ 拓广文法**G'**

(0)  **$S' \rightarrow S$**

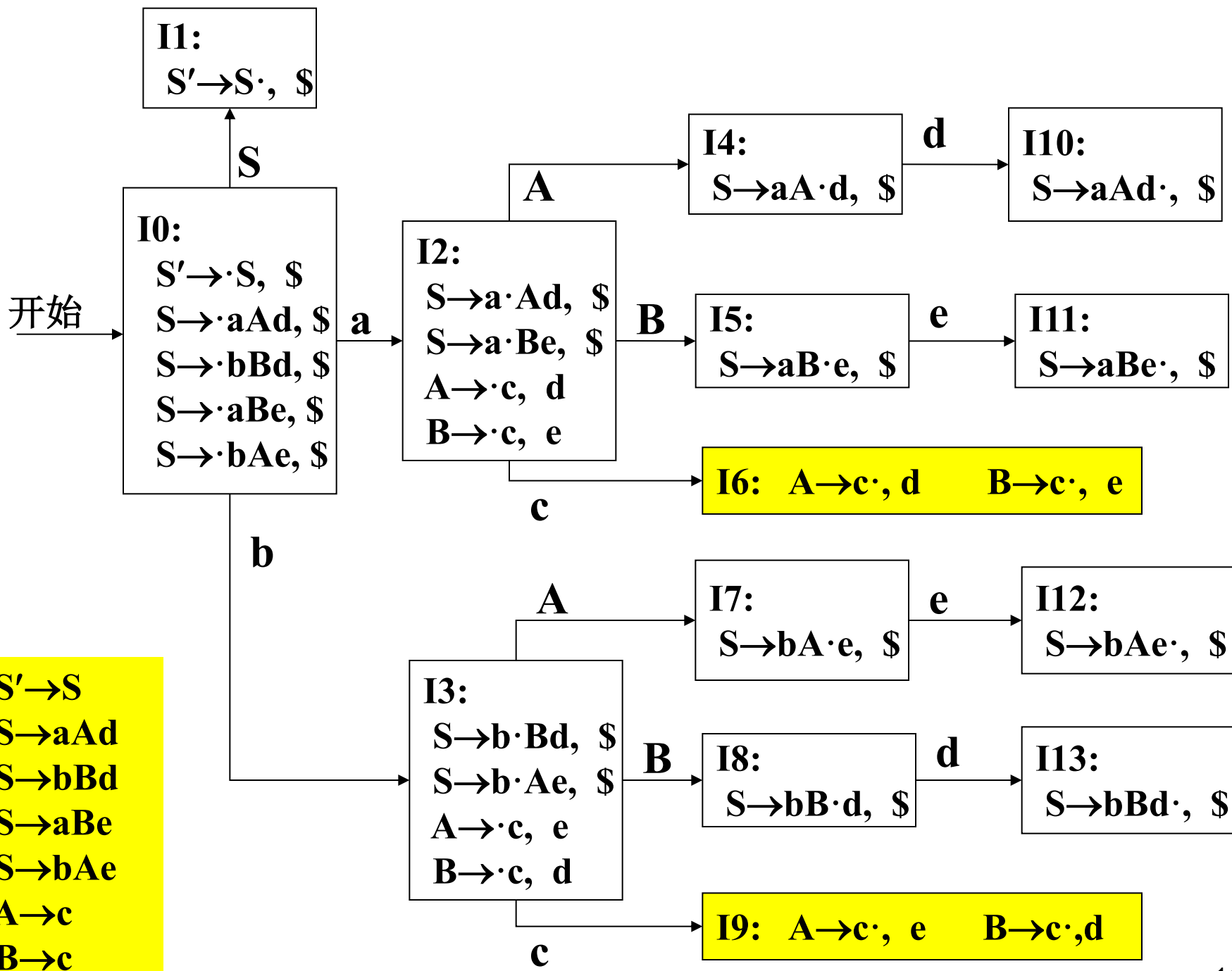
(1)  **$S \rightarrow aAd$**    (2)  **$S \rightarrow bBd$**    (3)  **$S \rightarrow aBe$**    (4)  **$S \rightarrow bAe$**

(5)  **$A \rightarrow c$**         (6)  **$B \rightarrow c$**

## ■ 构造识别文法**G'**的所有活前缀的DFA

$I_0 = \{ [S' \rightarrow \cdot S, \$]$

$[S \rightarrow \cdot aAd, \$] \quad [S \rightarrow \cdot bBd, \$] \quad [S \rightarrow \cdot aBe, \$] \quad [S \rightarrow \cdot bAe, \$] \}$



# 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：

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

## ■ 含有归约-归约冲突。



# LALR(1)分析表的构造

## ■ 基本思想是：

- ◆ 首先构造LR(1)项目集规范族；
- ◆ 如果它不存在冲突，就把同心集合并在一起；  
如果LR(1)项目集规范族中含有冲突，  
则该文法不是LR(1)文法，  
不能为它构造LALR分析程序。
- ◆ 若合并后的项目集规范族不存在归约-归约冲突，  
就按这个项目集规范族构造分析表。  
如果合并后得到的项目集规范族中含有冲突，  
则该文法是LR(1)文法，但不是LALR文法，  
因而，也不能为它构造LALR分析程序。

## 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 $[S' \rightarrow \cdot S, \$]$ 的 $J_k$ 为分析表的初态。

3.从 $C'$ 出发，构造action子表

a) 若 $[A \rightarrow \alpha \cdot a \beta, b] \in J_i$ ，且 $\text{go}(J_i, a) = J_j$ ，则  
置 $\text{action}[i, a] = sj$

b) 若 $[A \rightarrow \alpha \cdot, a] \in J_i$ ，则置 $\text{action}[i, a] = r \ A \rightarrow \alpha$

c) 若 $[S' \rightarrow S \cdot, \$] \in J_i$ ，则置 $\text{action}[i, \$] = \text{acc}$

## 算法4.10 构造LALR(1)分析表（续）

### 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

## ■ 文法4.8

(0)  $S' \rightarrow S$

(1)  $S \rightarrow CC$

(2)  $C \rightarrow cC$

(3)  $C \rightarrow d$

# 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：

$$I_{36} = \{ [C \rightarrow c \cdot C, c/d/\$] \quad [C \rightarrow \cdot cC, c/d/\$] \quad [C \rightarrow \cdot d, c/d/\$] \}$$

- ◆  $I_4$ 和 $I_7$ 合并，得到项目集：

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

- ◆  $I_8$ 和 $I_9$ 合并，得到项目集：

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

## ■ 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. 8，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

## ■ LALR(1) 分析程序对符号串cdcd的分析过程

步骤	栈	输入	分析动作
0	0	cdcd\$	shift
1	0c <sup>36</sup>	dcd\$	shift
2	0c <sup>36</sup> d <sup>47</sup>	cd\$	reduce by $C \rightarrow d$
3	0c <sup>36</sup> C <sup>89</sup>	cd\$	reduce by $C \rightarrow cC$
4	0C2	cd\$	shift
5	0C2c <sup>36</sup>	d\$	shift
6	0C2c <sup>36</sup> d <sup>47</sup>	\$	reduce by $C \rightarrow d$
7	0C2c <sup>36</sup> C <sup>89</sup>	\$	reduce by $C \rightarrow cC$
8	0C2C5	\$	reduce by $S \rightarrow CC$
9	0S1	\$	accept

## ■ LR(1) 分析程序对符号串cdcd的分析过程

步骤	栈	输入	分析动作
0	0	cdcd\$	shift
1	0c <sup>3</sup>	dcd\$	shift
2	0c <sup>3</sup> d <sup>4</sup>	cd\$	reduce by $C \rightarrow d$
3	0c <sup>3</sup> C <sup>8</sup>	cd\$	reduce by $C \rightarrow cC$
4	0C <sup>2</sup>	cd\$	shift
5	0C <sup>2</sup> c <sup>6</sup>	d\$	shift
6	0C <sup>2</sup> c <sup>6</sup> d <sup>7</sup>	\$	reduce by $C \rightarrow d$
7	0C <sup>2</sup> c <sup>6</sup> C <sup>9</sup>	\$	reduce by $C \rightarrow cC$
8	0C <sup>2</sup> C <sup>5</sup>	\$	reduce by $S \rightarrow CC$
9	0S <sup>1</sup>	\$	accept



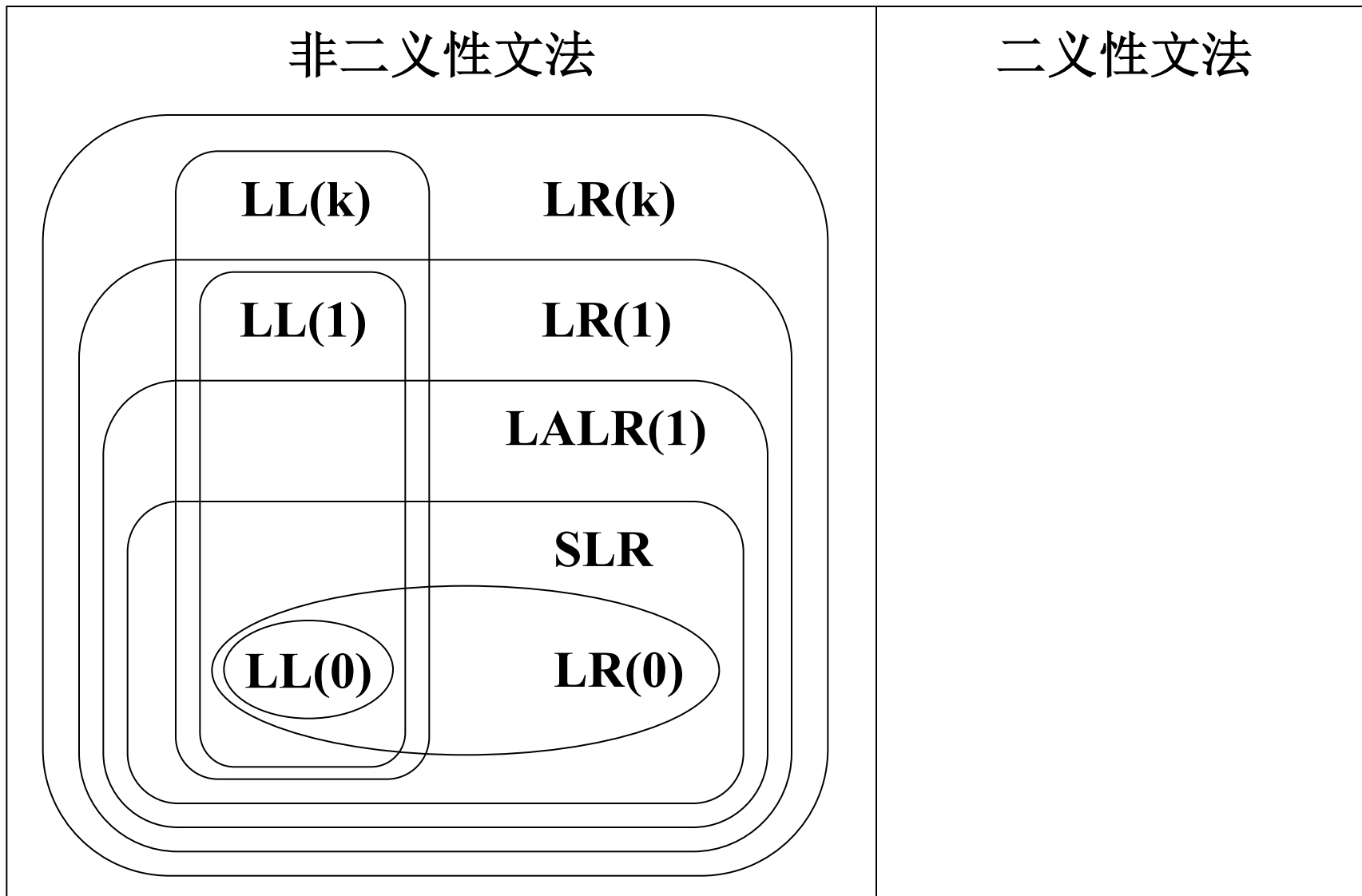
## ■ LR(1) 分析程序对符号串ccd的分析过程

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

# 文法分类



## 4.4.5 LR分析方法对二义文法的应用

- **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。
- 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。例如关于算术表达式的文法和if语句的文法。
- 在所有情况下，都说明了消除二义性的一些规则（即这类结构的使用限制）。

# 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：

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

- 无二义性的文法：

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

- 前者具有两个明显的优点：

- ◆ 改变运算符的优先级或结合规则时，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得多

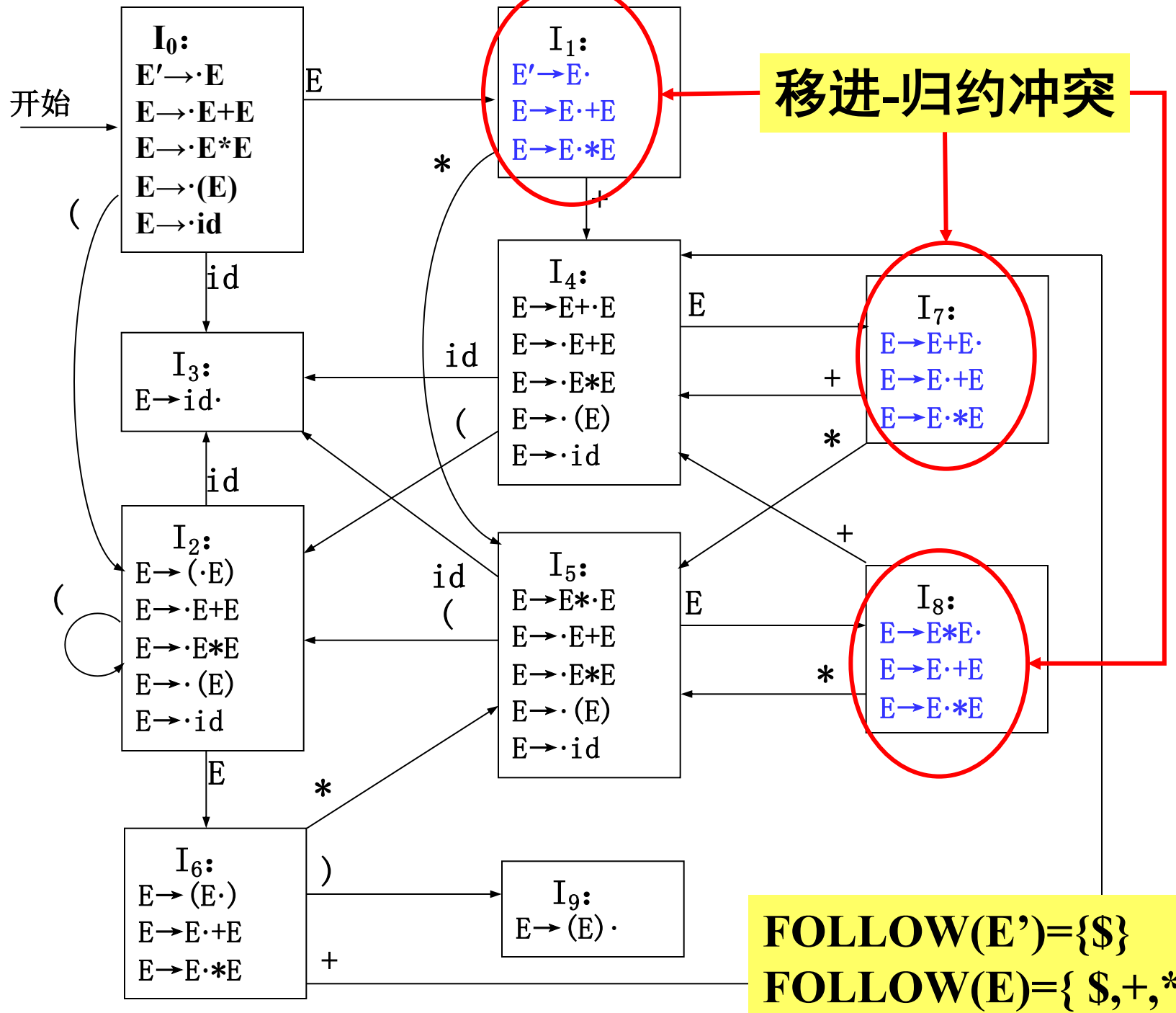
# 例：构造文法4.10的LR分析表。

- 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<sub>7</sub>:  
E→E+E·  
E→E·+E  
E→E·\*E

I<sub>8</sub>:  
E→E\*E·  
E→E·+E  
E→E·\*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	



# 利用最近最后匹配原则解决if语句冲突

## ■ 映射程序设计语言中if-then-else结构的文法:

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

(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:

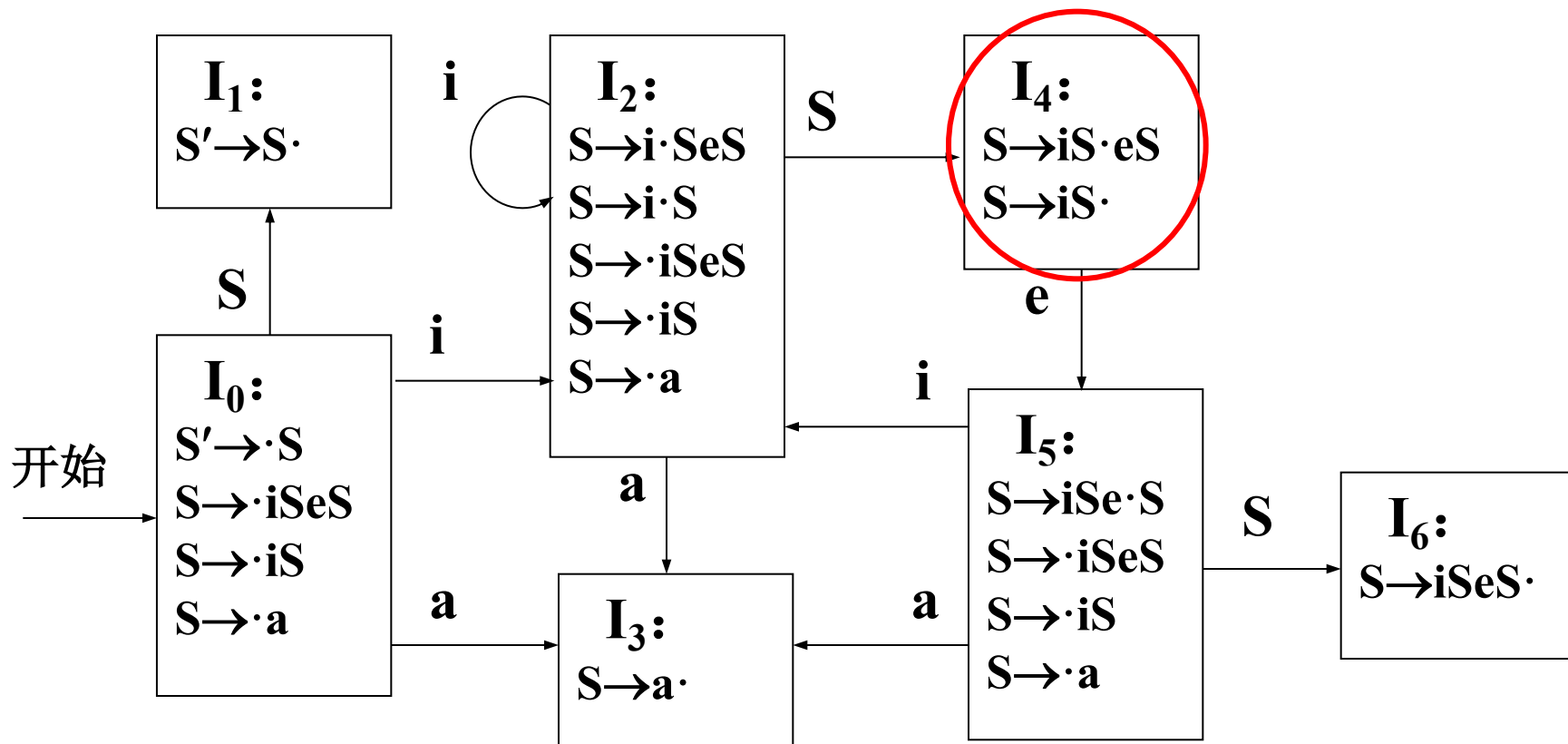
$$S \rightarrow iS \mid iSeS \mid a$$

(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)  $S \rightarrow a$



**FOLLOW(S) = { \$, e }**

## 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	

# 例：分析输入符号串iiaea

步骤	栈	输入	分析动作
1	0	iiaea\$	shift
	-		
2	0 2	iaea\$	shift
	- i		
3	0 2 2	aea\$	shift
	- 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，根据 $S$ 在 $goto$ 子表中可以找到一个非终结符号 $A$ ，即它有关于 $A$ 的转移；
- ◆ 然后，抛弃0个或若干个输入符号，直到找到符号 $a$ 为止， $a \in FOLLOW(A)$ ，即 $a$ 可以合法地跟在 $A$ 的后面；
- ◆ 然后，分析程序把状态 $goto[S, A]$ 压入栈，继续进行语法分析。

### ■ 在弹栈过程中出现的 $A$ 可能不止一个，通常选择表示主要结构成分的非终结符号。

### ■ 实际上是跳过包含错误的一部分终结符号串。

# 例：运算符

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 ‘+’ 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							
9							

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。

**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号‘+’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号，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号或右括号，而遇到的却是运算对象（**id**或左括号）。

- ◆ 策略：把运算符号‘+’压入栈，转移到状态**4**。

- ◆ 诊断信息：“缺少运算符号”

**e4:** 在状态**6**，期待输入符号为运算符号或右括号，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		



## 4.5 软件工具YACC

### ■ 语法分析程序自动生成工具

YACC: Yet Another Compiler-Compiler

一、**YACC**使用方式

二、**YACC**源程序结构

三、**YAC**处理二义文法的处理

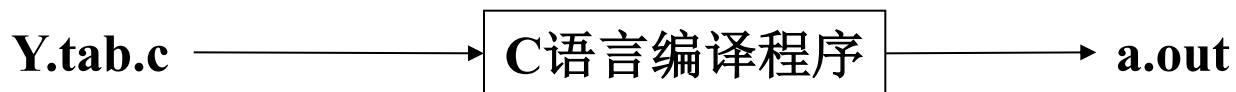
四、用**LEX**建立**YACC**的词法分析程序

五、**YACC**内部名称

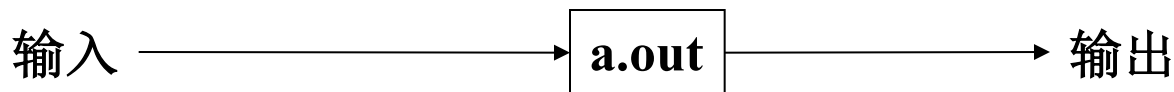
# 一、YACC 使用方式



`%yacc translate.y`



`%cc y.tab.c -ly -o a.out`



## 二、YACC源程序结构

- 由说明、翻译规则和辅助过程三部分组成，各部分之间用双百分号分隔。

### 说明部分：有任选的两节

- ◆ 第一节是处于`%{`和`%}`之间的部分，在这里是一些普通的C语言的声明。
- ◆ 第二节是文法记号的声明，一般
  - 以 `%start S` 的形式说明文法的开始符号。
  - 用 `%token IF、DO、ID、...` 的形式说明记号。
  - 记号被YACC赋予了不会与任何字符值冲突的数字值。

# 翻译规则部分

每条规则由一个产生式和有关的语义动作组成

- 产生式 $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ ，在YACC说明文件中写成

$A : \alpha_1 \{ \text{语义动作1} \}$   
 $| \alpha_2 \{ \text{语义动作2} \}$   
 $\dots$   
 $| \alpha_n \{ \text{语义动作n} \}$   
;

- 用单引号括起来的单个字符，如 ‘c’，是由终结符号c组成的记号
- 没有用引号括起来、也没有被说明成token类型的字母数字串是非终结符号
- 语义动作是用C语言描述的语句序列
  - ◆ ‘\$\$’ 表示和产生式左部非终结符号相关的属性值，‘\$i’ 表示和产生式右部第i个文法符号相关的属性值。

# 辅助过程部分

用C语言书写一些语义动作中用到的辅助程序

- 名字为**yylex()**的词法分析程序必须提供

- 函数**main**

```
main()
{
    return yyparse();
}
```

- YACC生成的**yyparse**过程调用一个词法分析程序**yylex**

- **yyparse** 每次调用 **yylex()** 时，得到一个二元式记号：  
<记号，属性值>。

- ◆ 返回的**记号**必须事先在YACC说明文件的第一部分中用**%token**说明
- ◆ **属性值**必须通过YACC定义的变量**yyval**传给分析程序

# 三、YACC对二义文法的处理

## ■ 处理冲突的两条缺省规则

- ◆ “归约-归约” 冲突，选择排在前面的产生式进行归约
- ◆ “移进-归约” 冲突，选择执行移进动作

## ■ 处理移进-归约冲突的机制

- ◆ 利用 **%left** ‘+’ ‘-’ 说明 ‘+’ 和 ‘-’ 具有同样的优先级，并且遵从左结合规则。
- ◆ 利用 **%right** ‘↑’ 声名算符 ‘↑’ 遵从右结合规则。
- ◆ 利用 **%nonassoc** ‘<’ 说明某些二元运算符不具有结合性。
- ◆ 先出现的记号的优先级低
- ◆ 同一声明中的记号具有相同的优先级

## -产生式的优先级

- ◆ 和它最右边的终结符号的优先级一致。
- ◆ 最右终结符号不能给产生式以适当的优先级
  - 通过给产生式附加标记 **%prec <terminal>** 来限制它的优先级
  - 它的优先级和结合性质同这个指定的终结符号的一样
  - 这个终结符号可以是一个占位符，不是由词法分析程序返回的记号，仅用来决定一个产生式的优先级。

- **YACC**不报告用这种优先级和结合性质能够解决的移进-归约冲突。

## 四、用LEX建立YACC的词法分析程序

- **LEX**编译器将提供词法分析程序**yylex()**
- 如果用**LEX**产生词法分析程序，则**YACC**说明文件中第三部分的函数**yylex()**应由语句  
**#include "lex.yy.c"** 代替。
- 使用这条语句，程序**yylex()**可以访问**YACC**中记号的名字，因为**LEX**的输出是**YACC**输出文件的一部分，所以每个**LEX**动作都返回**YACC**知道的终结符号。



## 五、YACC内部名称

YACC内部名称	说明
y.tab.c	YACC输出文件名
y.tab.h	YACC生成的头文件，包含有记号定义
yyparse	YACC分析程序
yylval	栈中当前记号的值
yyerror	由YACC使用的用户定义的错误信息打印程序
error	YACC错误伪记号
yyerrok	在错误处理之后，使分析程序回到正常操作方式的程序
yychar	变量，记录导致错误的先行记号
YYSTYPE	定义分析栈值类型的预处理器符号
yydebug	变量，当由用户设置为1时，生成有关分析动作的运行信息

# 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

对任何产生式： $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$

$$\mathbf{FIRST}(\alpha_i) \cap \mathbf{FIRST}(\alpha_j) = \phi$$

- ◆ 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

## ■ 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 $a_1a_2\ldots a_n\$$

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： $M[X,a]$

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

## ■ 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 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \phi$

## 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）

$action[S_n, a]=s$   $i$ ， $a$ 入栈， $i$ 入栈  $i=goto(S_n, a)$

$action[S_n, a]=r$   $A \rightarrow \beta$ ，弹出 $2*|\beta|$ 个符号，  
 $A$ 入栈， $goto(S_{n-r}, A)$ 入栈

$action[S_n, a]=acc$ ，分析成功

$action[S_n, a]=$ 空白，出错处理

## 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 $goto[i, A] = j$

# 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。

## YACC的使用

- ◆ YACC源程序结构
- ◆ 冲突的缺省处理原则
- ◆ 二义性处理机制

# 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