

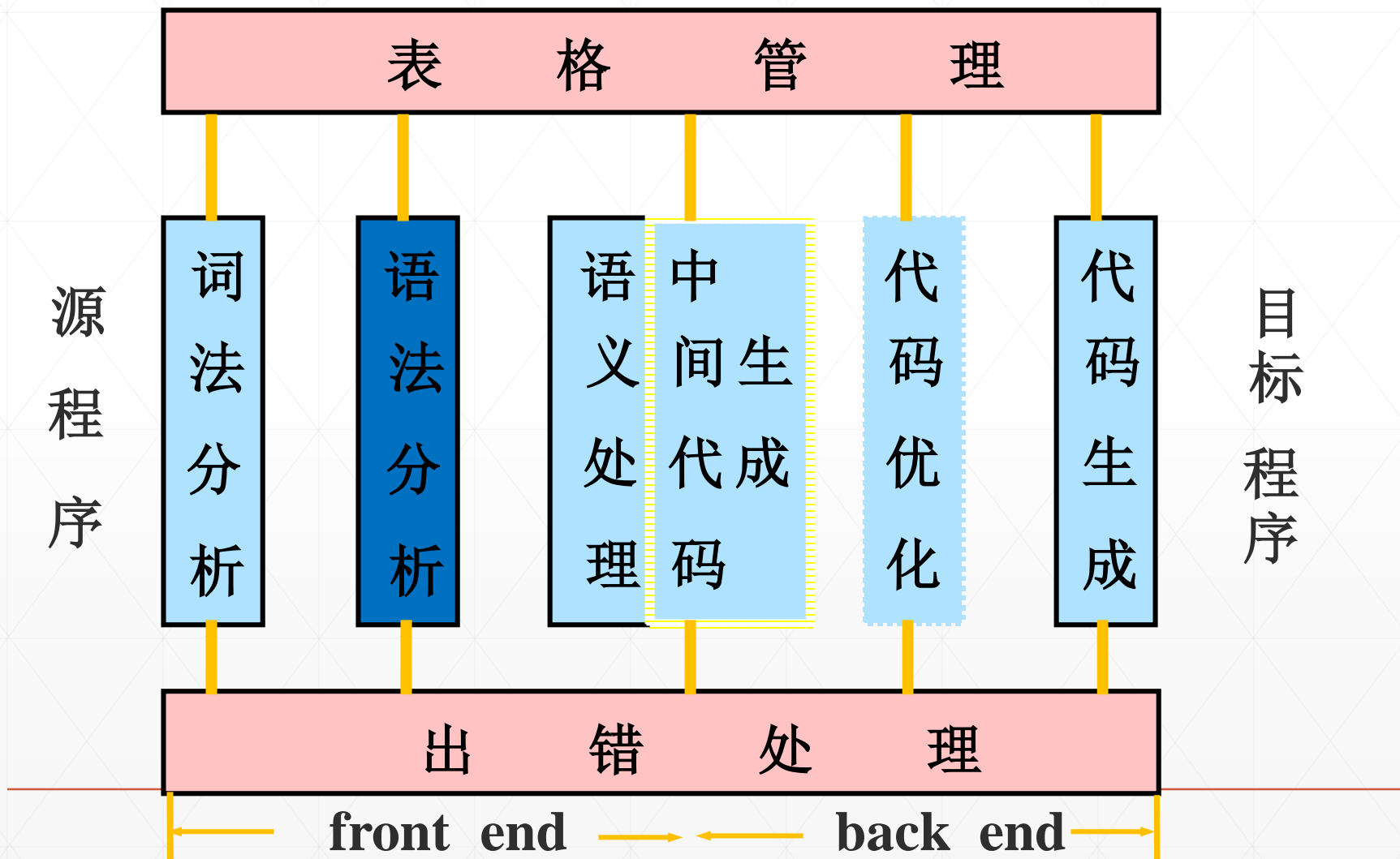


编译原理与设计

北京理工大学 计算机学院



语法分析：自下而上分析





自下而上分析：概览

从给定的输入串 S 开始，不断寻找子串与文法 G 中某个产生式 P 的候选式进行匹配，并用 P 的左部代替(归约)之，逐步归约到 S 。

归约条件

确定可归
约串

如何归约

归约原则



自下而上分析：移进-规约分析示例

- 设有文法 G 和输入字符串 $abbcd$

(1) $S \rightarrow aABe$

(2) $A \rightarrow Abc$

(3) $A \rightarrow b$

(4) $B \rightarrow d$

$S \xRightarrow[R]{\quad} aABe \xRightarrow[R]{\quad} aAde \xRightarrow[R]{\quad} aAbcde \xRightarrow[R]{\quad} abbcde$



自下而上分析：移进-规约分析示例

step	stack	\$	action
初始化	#	abbcde #	
(1)	# a	bbcd e #	shift
(2)	# ab	bcde #	shift
(3)	# aA	bcde #	A→b归约
(4)	# aAb	cde #	shift
(5)	# aAbc	de #	shift
(6)	# aA	de #	A→Abc归约
(7)	# aAd	e #	shift
(8)	# aAB	e #	B→d归约
(9)	# aABe	#	shift
(10)	# S	#	S→aABe归约
(11)	# S	#	接受(分析成功)

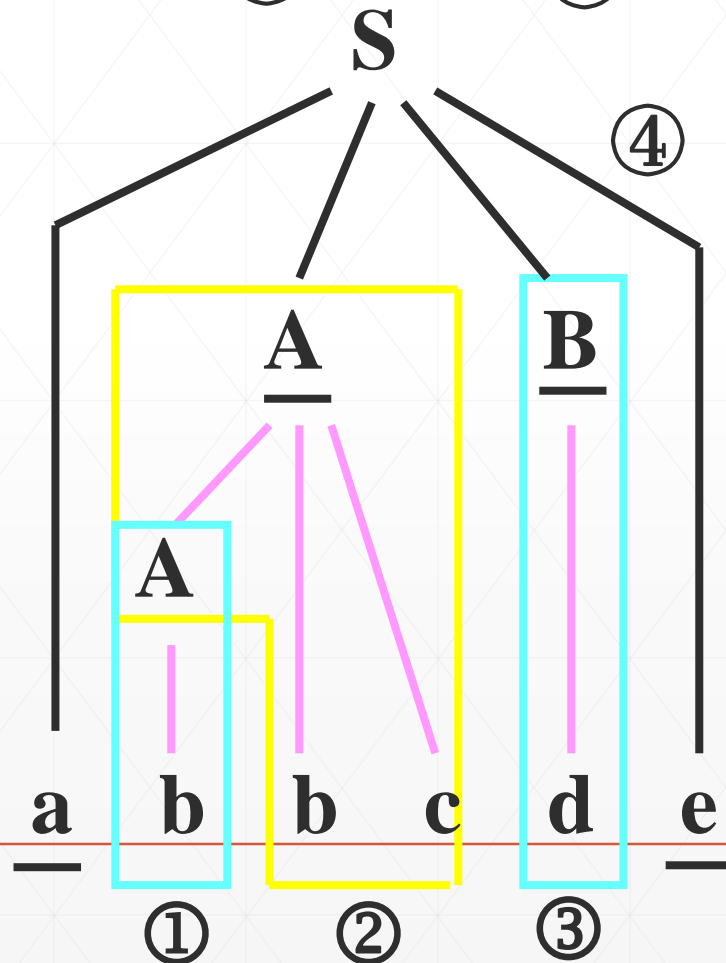


自下而上分析：移进-规约分析示例

$S \Rightarrow \underline{aA}Be \Rightarrow a\underline{A}de \Rightarrow a\underline{A}bcde \Rightarrow \underline{a}bbbcde$

④ ③ ② ①

- (1) $S \rightarrow aABe$
- (2) $A \rightarrow Abc$
- (3) $A \rightarrow b$
- (4) $B \rightarrow d$

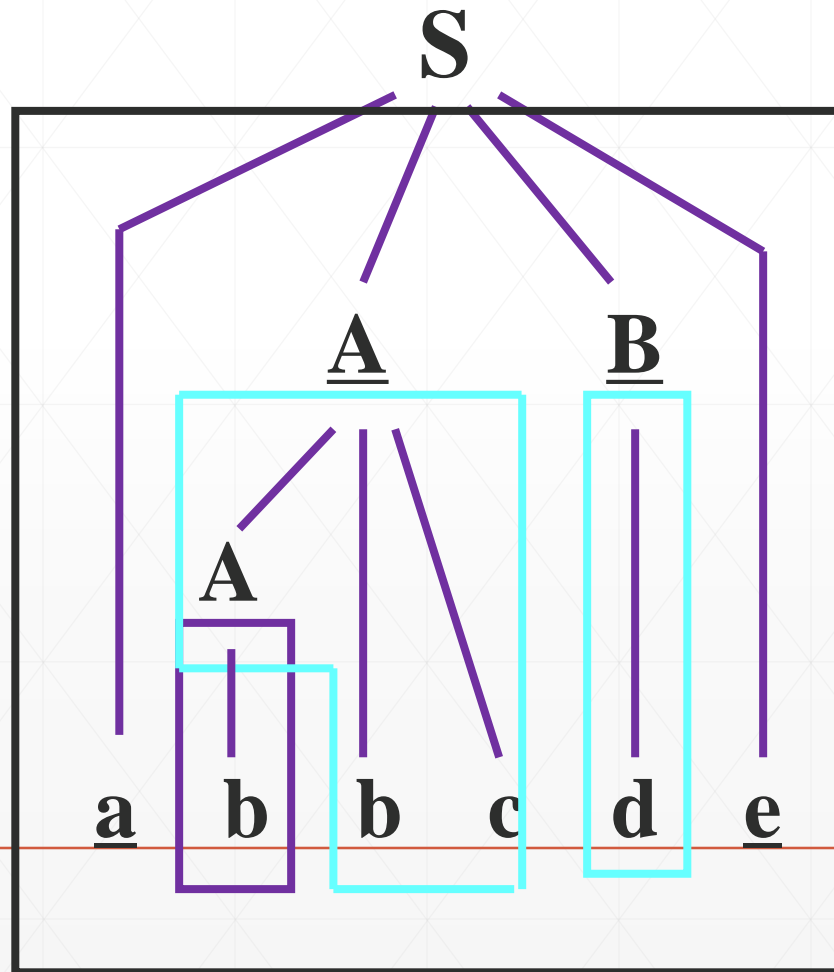




自下而上分析：移进-规约分析示例

$S \Rightarrow \underline{aAB} \underline{e} \Rightarrow aA\underline{d}e \Rightarrow a\underline{Abc}de \Rightarrow a\underline{b}bcde$

R **R** **R** **R**





自下而上分析：相关概念

■ 短语

令 G 是一部文法， S 是 G 的开始符号， $\alpha\beta\delta$ 是 G 的一个句型，若有 $S \xRightarrow{*} \alpha A \delta$ 且 $A \xRightarrow{+} \beta$ ，则 β 是句型 $\alpha\beta\delta$ 相对于 A 的短语。

■ 直接短语

令 G 是一部文法， S 是 G 的开始符号， $\alpha\beta\delta$ 是 G 的一个句型，若有 $S \xRightarrow{*} \alpha A \delta$ 且 $A \Rightarrow \beta$ ，则 β 是句型 $\alpha\beta\delta$ 相对于 A 的直接短语。



自下而上分析：相关概念

■ **句柄**：一个句型的最左直接短语

💧 **注意：**

- *直接短语一定是短语；
- *句柄一定是直接短语且具有最左性；
- *句子的句柄是语法树中最左子树的所有叶节点从左到右的排列或在句子的规范推导序列中，最后使用的产生式的右部；



自下而上分析：相关概念

例：设有文法G和输入串\$

$G: S \rightarrow aAcB \quad A \rightarrow P \quad P \rightarrow ab \quad B \rightarrow d$

$\$: aabcd$

对\$存在推导 $S \Rightarrow \underline{a} \underline{A} \underline{c} B \Rightarrow \underline{a} \underline{P} \underline{c} B \Rightarrow \underline{a} \underline{ab} \underline{c} B \Rightarrow \underline{a} \underline{abcd}$

则P是句型aPcB相对于A的短语，也是相对于A的直接短语，也是句型aPcB的句柄。



自下而上分析：相关概念

例：设有文法G和输入串\$

$G: S \rightarrow aAcB \quad A \rightarrow P \quad P \rightarrow ab \quad B \rightarrow d$

$\$: aabcd$

对\$存在推导 $S \Rightarrow \underline{a} \underline{A} \underline{c} B \Rightarrow a \underline{P} c B \Rightarrow a \underline{ab} c B \Rightarrow a a b c \underline{d}$

ab是句型aabcB相对于A的短语；
相对于P的直接短语和最左直接
短语即是句型aabcB的句柄。



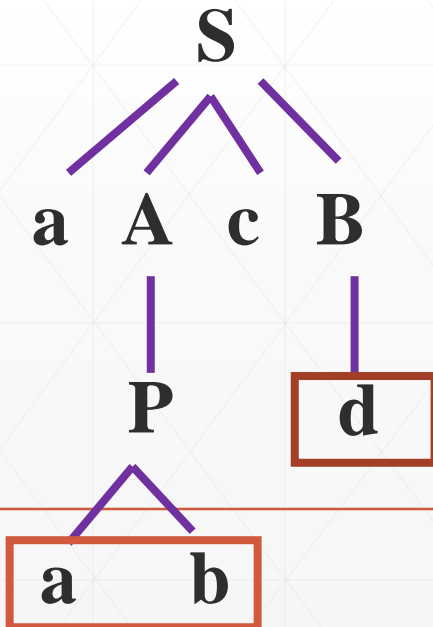
自下而上分析：相关概念

例：设有文法G和输入串\$

$G: S \rightarrow aAcB \quad A \rightarrow P \quad P \rightarrow ab \quad B \rightarrow d$

$\$: aabcd$

对\$存在推导 $S \Rightarrow aAcB \Rightarrow aAcd \Rightarrow aPcd \Rightarrow a\underline{abcd}$



d是句子abcd相对于B的短语；
相对于B的直接短语；ab是句
子abcd相对于A的短语；相对
于P的直接短语；也是句子
abcd的句柄。



自下而上分析：LR分析与LR分析器

- 理论上比较完善；
- 适用性强，对G限定少；
- 便于自动生成。

LR分析技术是编译系统中语法分析器实现最常用、最有效的一种分析方法。



自下而上分析：LR分析与LR分析器

- LR分析：一类对源程序串进行自左向右扫描并进行规范归约的语法分析方法。

LR(k)

在LR分析的每一步, 仅据分析栈当前已经移进和归约的全部语法符号并对\$最多再向前看k个输入字符, 就能确定适合于文法规则的句柄是否已在栈顶形成, 从而立即确定当前的分析动作。

分析模式：最右推导逆序(规范归约)

扫描模式：自左向右



自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构

输入字符串\$

a_1	a_2	a_i	a_n	#
-------	-------	-------	-------	-------	-------	---

语法分
析结果

总控程序

LR分析表

组成 { 总控程序
分析栈
LR分析表

stack

S_m	X_m
S_{m-1}	X_{m-1}
\vdots	\vdots
S_1	X_1
S_0	#



自下而上分析：LR分析与LR分析器

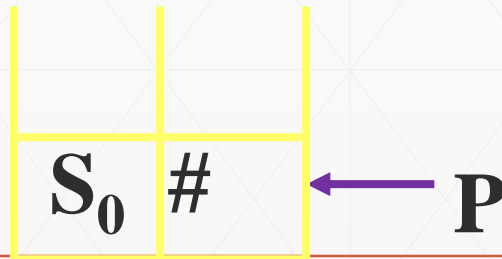
- LR分析器逻辑结构

- 分析栈：辅助完成LR分析的数据结构

stack { 状态 S_i : 记录分析过程中每一步的“历史”或“展望”信息;

文法符号 X_i : 放分析过程中移进 (V_T) 和归约 (V_N) 的符号;

stack初始化:





自下而上分析：LR分析与LR分析器

例如，设有文法 $G(L)$ 和
 $G(L)$ 的LR分析表

- ① $L \rightarrow E, L$
- ② $L \rightarrow E$
- ③ $E \rightarrow a$
- ④ $E \rightarrow b$

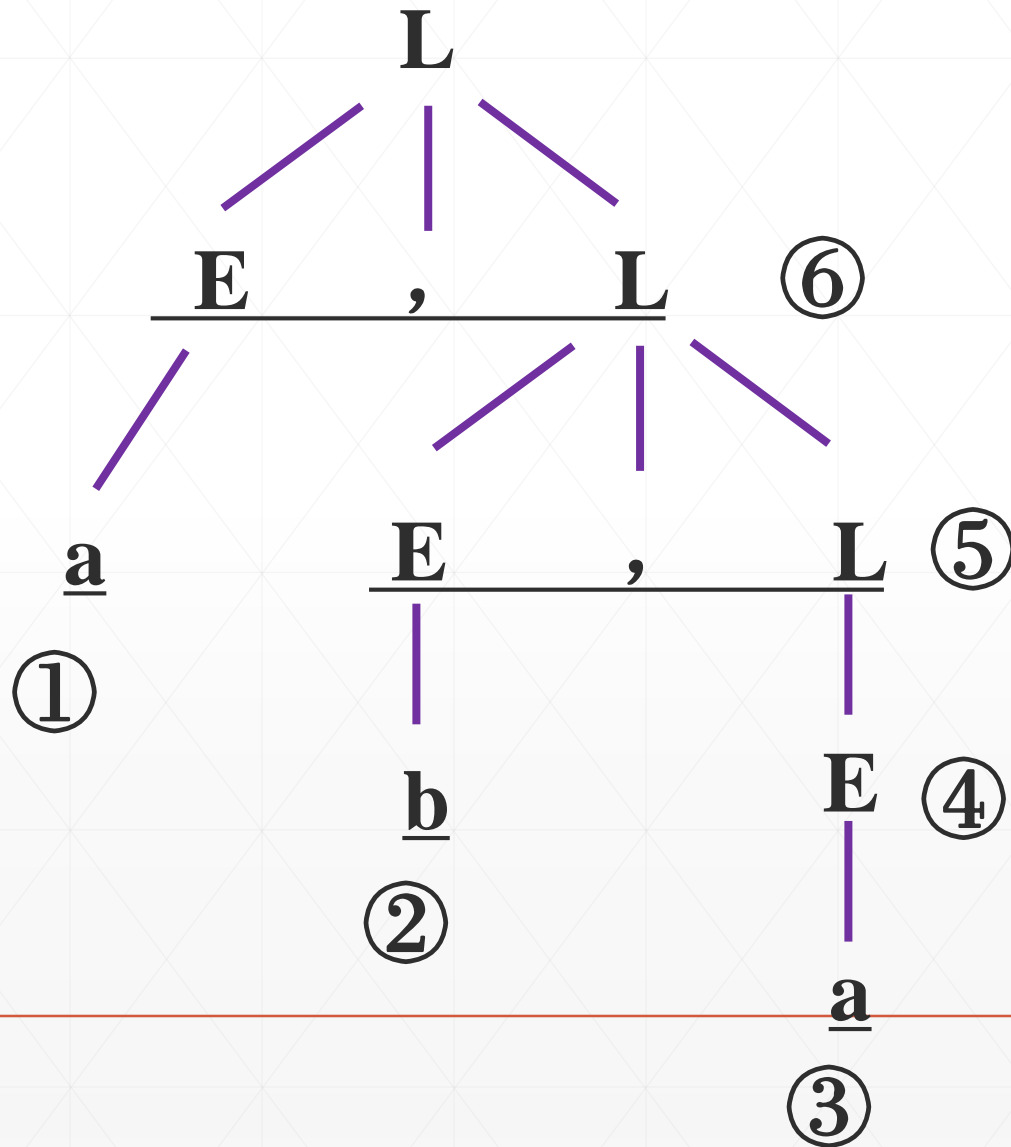
输入字符串 $a, b, a \#$

F
↓

$b \Rightarrow E$ (归约)	2	E	← P
, (移进栈)	5	,	
$a \Rightarrow E$ (归约)	2	E	
	0	#	



自下而上分析：LR分析与LR分析器





自下而上分析：LR分析与LR分析器

- LR分析器逻辑结构
 - 分析表：LR分析器的核心





自下而上分析：LR分析与LR分析器

- LR分析器逻辑结构：分析表

table state \ V	Action (V_T)				Goto (V_N)			
	V_{T1}	V_{T2}	...	V_{Tn}	V_{N1}	V_{N2}	...	V_{Nm}
S_0			
S_1			
...			
S_n			



自下而上分析：LR分析与LR分析器

- LR分析器逻辑结构：分析表：状态转换表

state V_N	X_1	X_2	...	X_n
S_0	goto(S_0, X_1)	goto(S_0, X_2)	...	goto(S_0, X_n)
S_1	goto(S_1, X_1)	goto(S_1, X_2)	...	goto(S_1, X_n)
...
S_n	goto(S_n, X_1)	goto(S_n, X_2)	...	goto(S_n, X_n)

$S_i \in$ 状态; $X_i \in$ 非终结符;



自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构：分析表

$\text{goto}(S_i, X_i) =$

j (移进：将第 j 个状态压入栈)

error (出错：语法错，调出错处理程序)

🔥 注意：

$\text{goto}(S_i, X_i)$ 的 S_i 、 X_i 意指当前栈顶的 X_i 和 **次栈顶** 的 S_i 元素。



自下而上分析：LR分析与LR分析器

- LR分析器逻辑结构：分析表

$\text{state} \backslash V_T$	a_1	a_2	...	a_n
S_0	$\text{action}(S_0, a_1)$	$\text{action}(S_0, a_2)$...	$\text{action}(S_0, a_n)$
S_1	$\text{action}(S_1, a_1)$	$\text{action}(S_1, a_2)$...	$\text{action}(S_1, a_n)$
...
S_n	$\text{action}(S_n, a_1)$	$\text{action}(S_n, a_2)$...	$\text{action}(S_n, a_n)$

$S_i \in \text{状态}; a_i \in \text{非终结符};$



自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构：分析表

$\text{action}(S_i, a_i) =$

S_j (移进：将 a_i 和第 j 个状态压入栈)

r_j (归约：用第 j 个产生式归约)

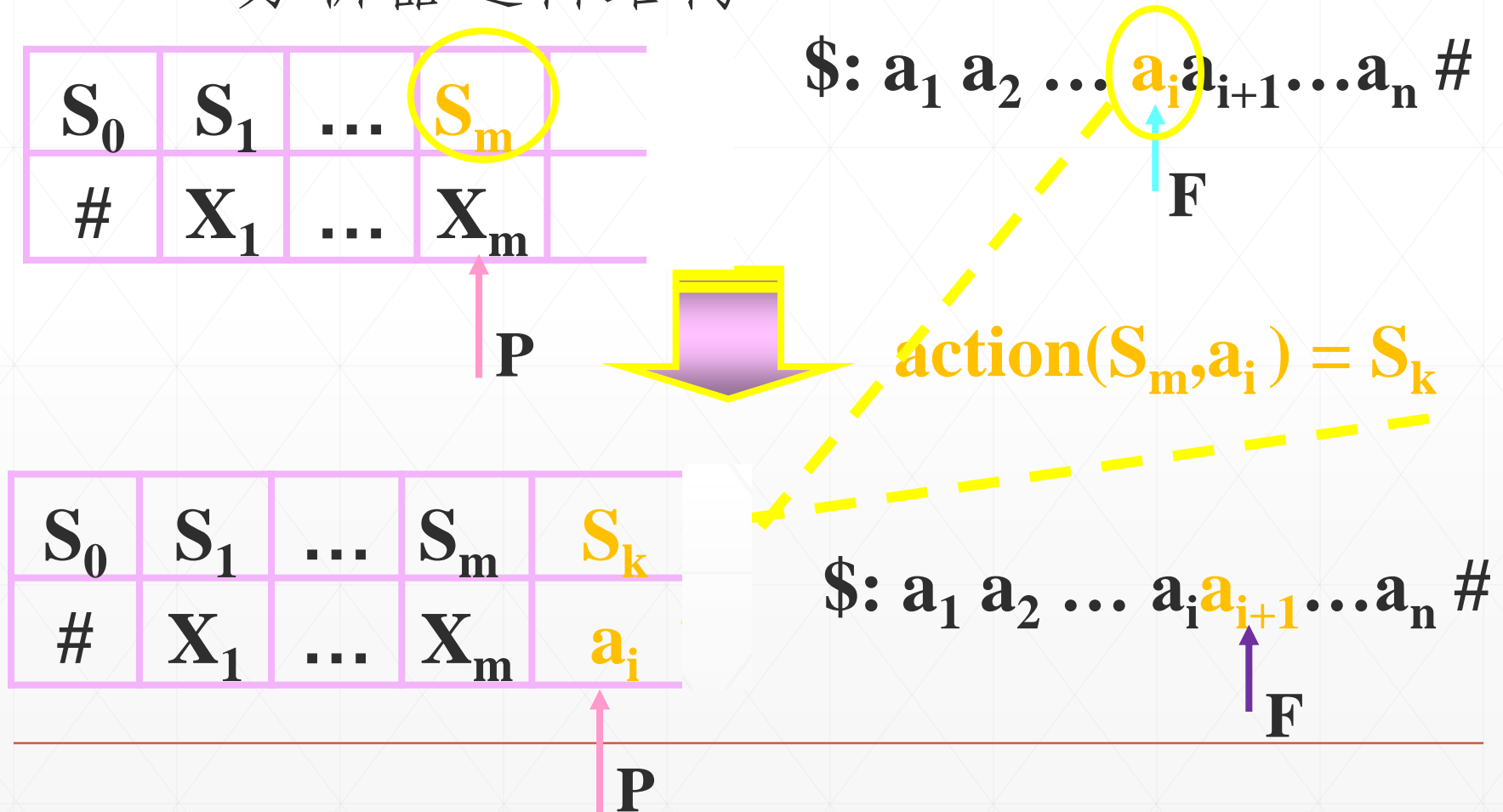
acc (接受：分析成功)

error (出错：语法错，调出错处理程序)



自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构





自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构 $\$: a_1 a_2 \dots a_i a_{i+1} \dots a_n \#$

S_0	S_1	\dots	S_{m-r}	S_{m-r+1}	\dots	S_m	
$\#$	X_1	\dots	X_{m-r}	X_{m-r+1}	\dots	X_m	



$$\text{action}(S_m, a_i) = r_j$$

① 归约。设G第j个产生式为： $A \rightarrow X_{m-r+1} X_{m-r+2} \dots X_m$

S_0	S_1	\dots	S_{m-r}		
$\#$	X_1	\dots	X_{m-1}	A	

P



自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构

② 查goto表。 $\text{goto}(\mathbf{S_{m-r}}, \mathbf{A}) = j$

S_0	S_1	\dots	S_{m-r}	S_j	
$\#$	X_1	\dots	X_{m-r}	A	

P

$\$: a_1 a_2 \dots a_i a_{i+1} \dots a_n \#$

F

(扫描指针不变)



自下而上分析：LR分析与LR分析器

■ LR分析器逻辑结构: LR分析总控程序

- ① 分析开始，将初始状态 S_0 及输入字符串左界符“#”推入分析栈；
 - ② 对分析的每一步，据当前分析栈栈顶 S_m ，当前输入符号 a_i 查**action**表：
 - i) 若 $\text{action}(S_m, a_i) = S_j$ ，完成**移进**动作；
 - ii) 若 $\text{action}(S_m, a_i) = r_j$ ，完成**归约**动作，并查goto表；
 - iii) 若 $\text{action}(S_m, a_i) = \text{acc}$ ，分析**成功**；
 - iv) 若 $\text{action}(S_m, a_i) = \text{error}$ ，**出错**处理。
 - ③ 转②。
-



自下而上分析：LR分析与LR分析器

■ LR分析实例

例 设有文法 $G(L)$ 和 $G(L)$ 的LR分析表

① $L \rightarrow E, L$

② $L \rightarrow E$

③ $E \rightarrow a$

④ $E \rightarrow b$

输入字符串 $a, b, a \#$

状 态	action 表				goto 表	
	a	b	,	#	E	L
0	S_3	S_4			2	1
1				acc		
2			S_5	r_2		
3			r_3	r_3		
4			r_4	r_4		
5	S_3	S_4			2	6
6				r_1		



自下而上分析：LR分析与LR分析器

■ LR分析实例

状态	action 表				goto 表	
	a	b	,	#	E	L
0	S ₃	S ₄			2	1
1				acc		
2			S ₅	r ₂		
3			r ₃	r ₃		
4			r ₄	r ₄		
5	S ₃	S ₄			2	6
6				r ₁		

步 骤	栈中状态	栈中符号	输入符号串	分 析 动 作
1	0	#	a, b, a #	S ₃
2	03	# a	, b, a #	r ₃ (用规则③归约)
3	02	# E	, b, a #	S ₅
4	025	# E,	b, a #	S ₄
5	0254	# E, b	, a #	r ₄ (用规则④归约)
6	0252	# E, E	, a #	S ₅
7	02525	# E, E,	a #	S ₃
8	025253	# E, E, a	#	r ₃ (用规则③归约)
9	025252	# E, E, E	#	r ₂ (用规则②归约)
10	025256	# E, E, L	#	r ₁ (用规则①归约)
11	0256	# E, L	#	r ₁ (用规则①归约)
12	01	# L	#	acc

① L → E, L

② L → E

③ E → a

④ E → b



自下而上分析：LR分析与LR分析器

■ LR分析实例

🔥 综述：

- (1) 处理直观简单；
- (2) 基本实现思想：引入状态，状态埋伏了分析的“历史”和“展望”信息；
- (3) 应用范围广，对G限定少；
- (4) LR分析的关键——LR分析表：集成了全部分析信息。



自下而上分析：LR(0)分析：实现思想

- **前缀：** 一个句型的任意首部，称为该句型的一个前缀。
- **活前缀：** 规范句型的一个不含句柄之后任何符号的前缀，称为该句型的一个活前缀。

注意：

- (1) $\exists S \xRightarrow{*}_R \alpha \mathbf{A} \omega \Rightarrow_R \alpha \mathbf{B} \omega$ ，若串 γ 是 $\alpha\beta$ 的前缀， γ 是活前缀；当 $\gamma=\alpha\beta$ ， γ 是可归前缀；
- (2) **活前缀特点：** 不含句柄之后的任何符号；
- (3) **LR分析中必需使栈中符号始终是活前缀**，这样再从输入串读入几个符号后，构成刚好包含句柄的活前缀，进而实施归约。



自下而上分析：LR(0)分析：实现思想

■ LR(0) 项目

在文法G的每个产生式的右部(候选式)的任何位置上添加一个圆点，所构成的每个产生式称为LR(0)项目。

约定：若产生式形为 $A \rightarrow \epsilon$ 则其LR(0)项目为： $A \rightarrow \cdot$ 。



自下而上分析：LR(0)分析：实现思想

■ LR(0) 项目

例： 设文法G(S)

$S \rightarrow A \mid B$ $A \rightarrow aA \mid b \mid \varepsilon$ $B \rightarrow c$

则G(S)的LR(0)项目有：

$S \rightarrow \cdot A$ $S \rightarrow A \cdot$

$S \rightarrow \cdot B$ $S \rightarrow B \cdot$

$A \rightarrow \cdot aA$ $A \rightarrow a \cdot A$ $A \rightarrow aA \cdot$

$A \rightarrow \cdot b$ $A \rightarrow b \cdot$

$A \rightarrow \cdot$

$B \rightarrow \cdot c$ $B \rightarrow c \cdot$



自下而上分析：LR(0)分析：实现思想

■ LR(0) 项目分类

(1) 归约项目： $A \rightarrow \alpha \cdot$

这类LR(0)项目表示句柄 α 恰好包含在栈中，即当前栈顶的部分内容构成了所期望的刚好含句柄的活前缀，应按 $A \rightarrow \alpha$ 进行归约。

(2) 接受项目： $S' \rightarrow \alpha \cdot$ （ S' 是开始符号）

其中 S' 是文法惟一的开始符号。这类LR(0)项目实际是特殊的归约项目，表示分析栈中内容恰好为 α ，用 $S' \rightarrow \alpha$ 进行归约，则整个分析成功。



自下而上分析：LR(0)分析：实现思想

■ LR(0) 项目分类

(3) 移进项目： $A \rightarrow \alpha \cdot a\beta$ ($a \in V_T$)

这类LR(0)项目表示分析栈中是不完全包含句柄的活前缀，为构成恰好含有句柄的活前缀，还需将 a 移进分析栈。

(4) 待约项目： $A \rightarrow \alpha \cdot B\beta$ ($B \in V_N$)

LR(0)项目表示分析栈中是不完全包含句柄的活前缀，为构成恰好有句柄的活前缀，应先把当前输入字符串中的相应内容先归约到 B 。



自下而上分析：LR(0)分析：实现思想

■ LR(0) 项目分类

例：设文法G(S)

$S \rightarrow A|B$ $A \rightarrow aA|b|\epsilon$ $B \rightarrow c$

则G(S)的LR(0)项目有：

$S \rightarrow \cdot A$

$S \rightarrow \cdot B$

$A \rightarrow \cdot aA$

$A \rightarrow \cdot b$

$A \rightarrow \cdot$

$B \rightarrow \cdot c$

$S \rightarrow A \cdot$

$S \rightarrow B \cdot$

$A \rightarrow a \cdot A$

$A \rightarrow b \cdot$

$B \rightarrow c \cdot$

$A \rightarrow aA \cdot$



自下而上分析：LR(0)分析：实现思想

■ 构造识别文法G的所有活前缀的非确定有限自动机

- ① 规定含有文法开始符号的产生式(设 $S' \rightarrow A$)的第一个LR(0)项目(即 $S' \rightarrow \cdot A$)为NFA的惟一初态;
- ② 令所有LR(0)项目分别对应NFA的一个状态且 LR(0)项目为归约项目的对应状态为终态。
- ③ 若状态i和状态j出自同一文法G的产生式且两个状态LR(0)项目的圆点只相差一个位置, 则从状态i引一条标记为 X_i 的弧到状态j

■ 若 i 为: $X \rightarrow X_1 X_2 \dots X_{i-1} \cdot X_i \dots X_n$

j 为: $X \rightarrow X_1 X_2 \dots X_i \cdot X_{i+1} \dots X_n$

- ④ 若状态i为待约项目(设 $X \rightarrow \alpha \cdot A \beta$), 则从状态i引 ϵ 弧到所有 $A \rightarrow \cdot \gamma$ 的状态。



自下而上分析：LR(0)分析：实现思想

例：设文法 $G(S')$

$S' \rightarrow A$ $A \rightarrow aA \mid b$

构造识别文法 $G(S')$ 的所有活前缀的NFA。

文法 $G(S')$ 的LR(0)项目：

1. $S' \rightarrow \cdot A$

2. $A \rightarrow \cdot aA$

3. $A \rightarrow a \cdot A$

4. $A \rightarrow \cdot b$

5. $A \rightarrow b \cdot$

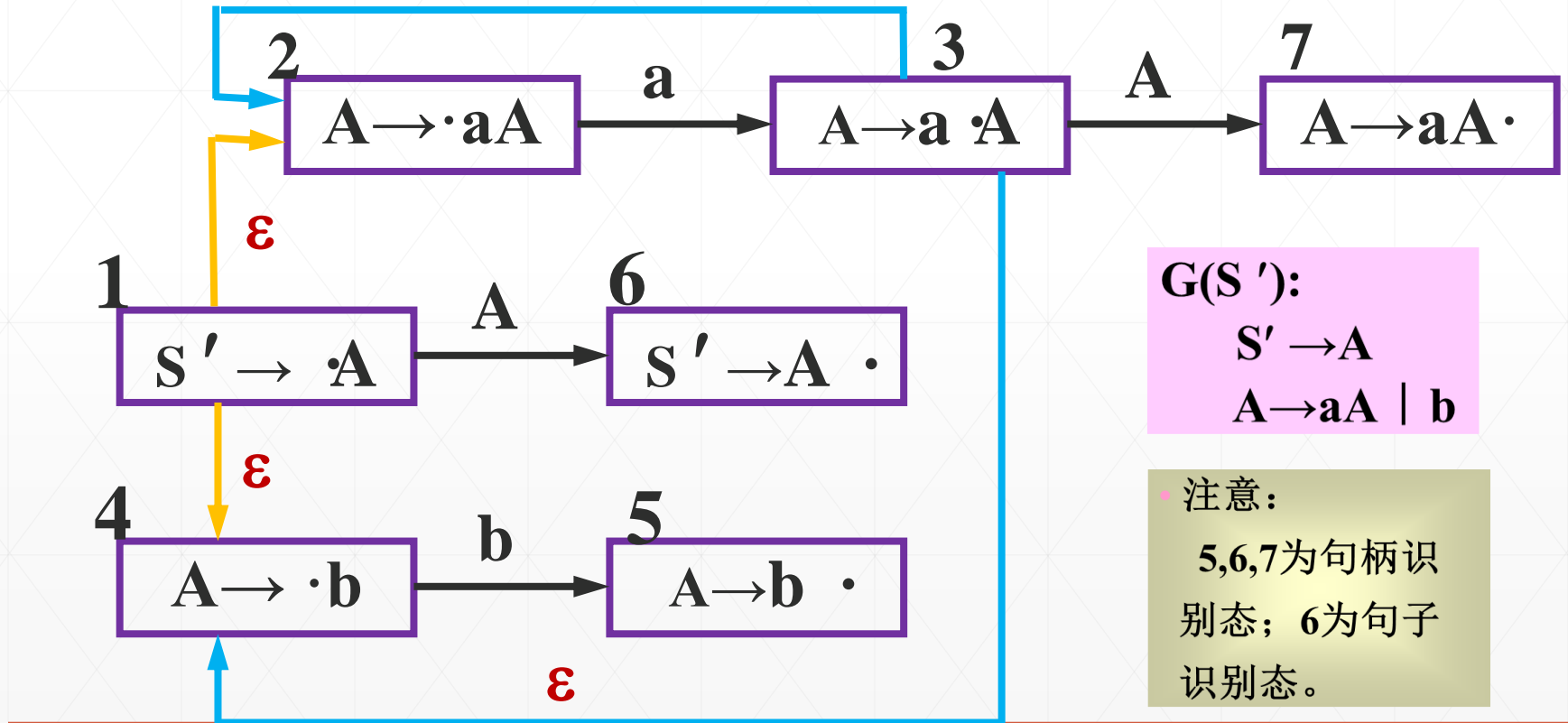
6. $S' \rightarrow A \cdot$

7. $A \rightarrow aA \cdot$



自下而上分析：LR(0)分析：实现思想

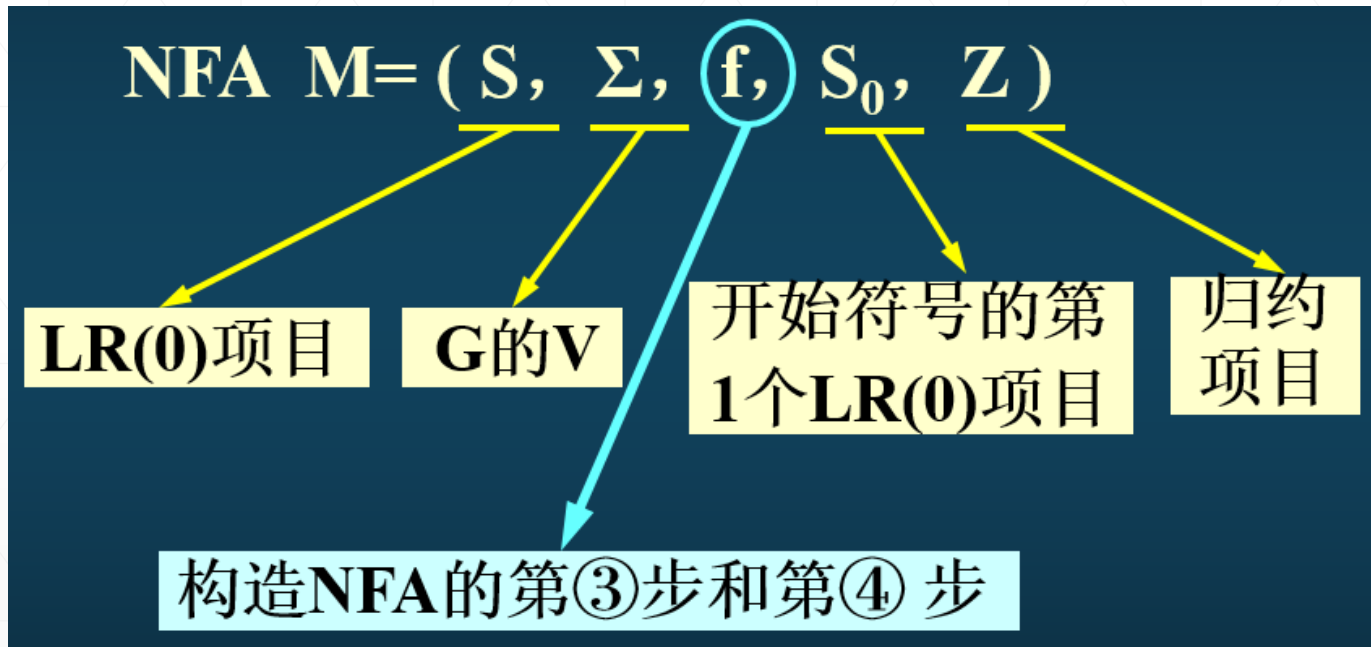
- 识别文法 $G(S')$ 的所有活前缀的NFA





自下而上分析：LR(0)分析：实现思想

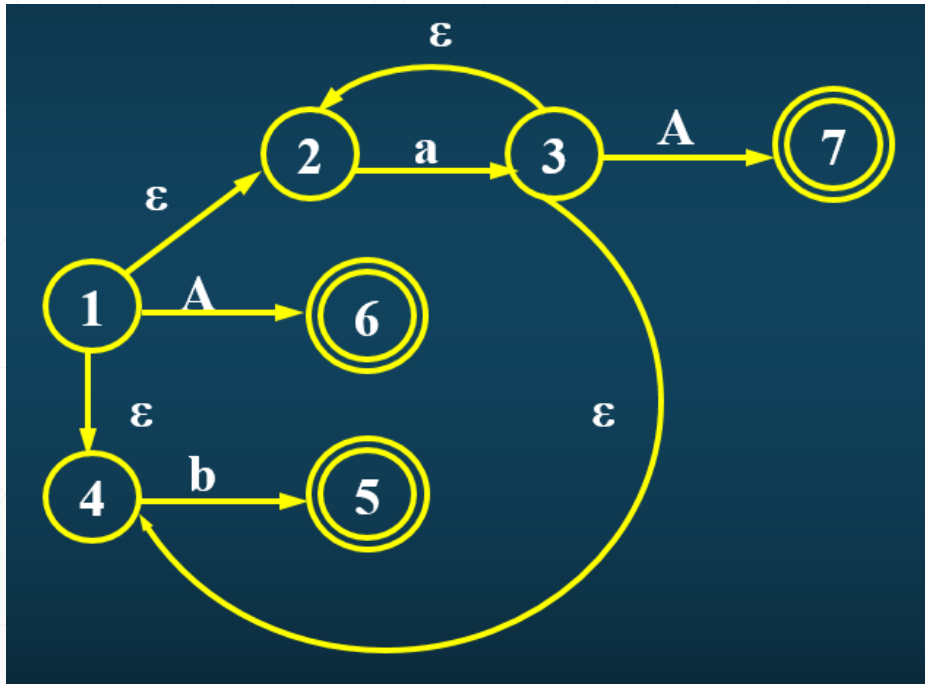
- 识别文法G的所有活前缀的NFA





自下而上分析：LR(0)分析：实现思想

- 识别文法G的所有活前缀的NFA



1. $S' \rightarrow \cdot A$

2. $A \rightarrow \cdot aA$

3. $A \rightarrow a \cdot A$

4. $A \rightarrow \cdot b$

5. $A \rightarrow b \cdot$

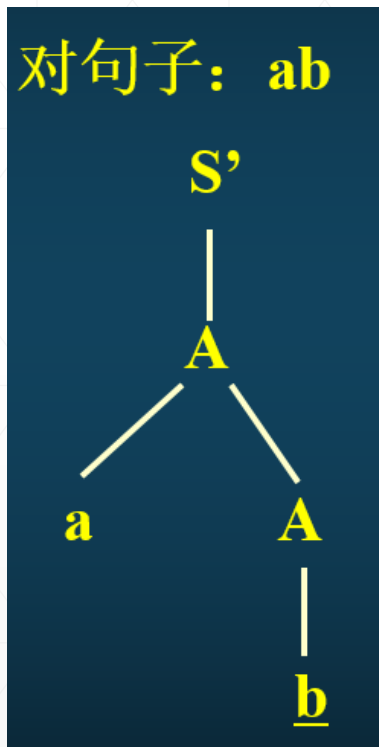
6. $S' \rightarrow A \cdot$

7. $A \rightarrow aA \cdot$



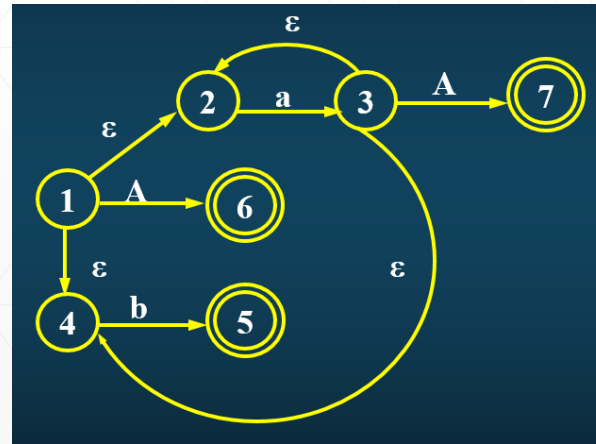
自下而上分析：LR(0)分析：实现思想

- 识别文法G的所有活前缀的NFA



句子ab归约第1步句柄为b，活前缀是 ϵ ， a， ab

\therefore 从1出发经 ϵ ， a， ϵ ， b到达5，此时活前缀ab含有句柄b.

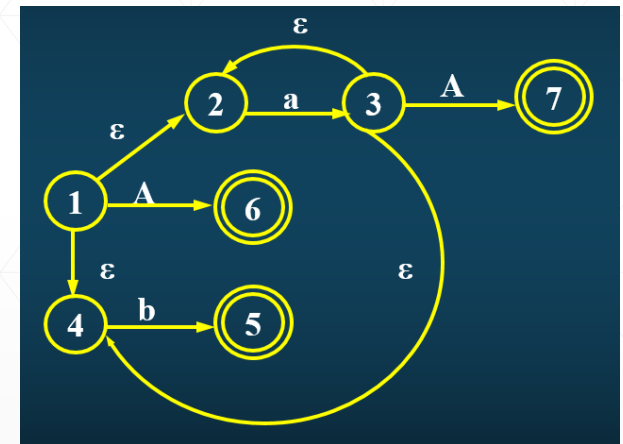




自下而上分析：LR(0)分析：实现思想

- 识别文法G的所有活前缀的NFA

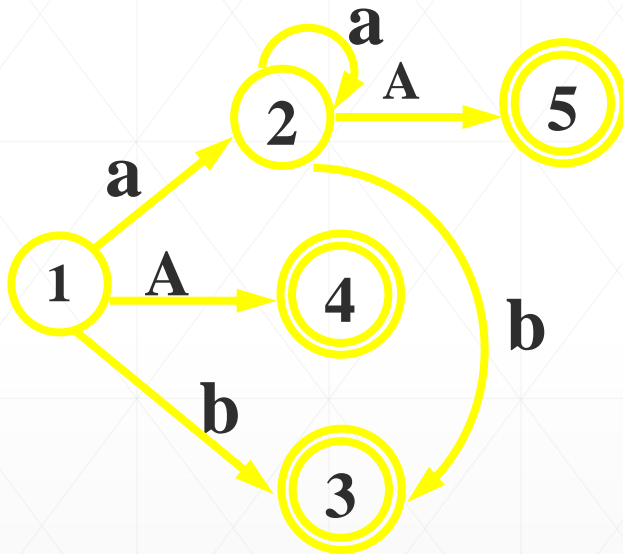
I	a	A	b
① {1,2,4}	{3,,2,4}	{6}	{5}
② {3,2,4}	{3,2,4}	{7}	{5}
③ {6} *	\varnothing	\varnothing	\varnothing
④ {5} *	\varnothing	\varnothing	\varnothing
⑤ {7} *	\varnothing	\varnothing	\varnothing





自下而上分析：LR(0)分析：实现思想

- 识别文法G的所有活前缀的NFA

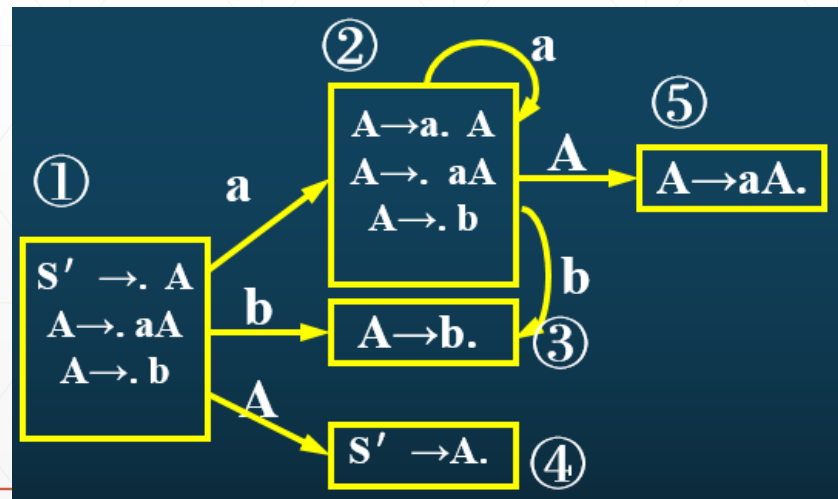


句子ab的归约：

$1 \rightarrow 2 \rightarrow 4$ ab (b归约到A)

$1 \rightarrow 2 \rightarrow 5$ aA (aA归约到A)

$1 \rightarrow 3$ A (A归约到S')





自下而上分析：LR(0)分析：实现思想

- 识别文法G的所有活前缀的NFA

识别文法G活前缀的DFA项目集的全体称为文法G的LR(0)项目集规范族。

$$C = (\{ S' \rightarrow \cdot A, A \rightarrow \cdot aA, A \rightarrow \cdot b \} \\ \{ A \rightarrow a \cdot A, A \rightarrow \cdot aA, A \rightarrow \cdot b \} \\ \{ A \rightarrow b \cdot \} \{ S' \rightarrow A \cdot \} \{ A \rightarrow aA \cdot \})$$



自下而上分析：LR(0)分析：实现思想

- 识别文法G的所有活前缀的NFA
 - **第一步**：构造文法G的LR(0)项目；
 - **第二步**：基于LR(0)项目，构造识别文法G所有活前缀的NFA；
 - **第三步**：NFA确定化为DFA，该DFA的各个状态所包含的项目的集合，即构成了G的LR(0)项目集规范族；
-




自下而上分析：LR(0)分析：实现思想

- 构造LR(0)项目集规范族的方法（之二）
 - 第一步：拓广文法（使文法开始符号的候选式惟一）

为使接受状态易于识别，对于不仅在一个产生式左端出现的文法的开始符号，要对文法进行拓广。设G是一文法，S是它的开始符号，则将产生式 $S' \rightarrow S$ 加入到G中构成新的文法G'，S'为G'的开始符号，G'称为G的拓广文法。

例如，设文法G(S)

$S \rightarrow A \mid B$	$A \rightarrow aA \mid b \mid \epsilon$	$B \rightarrow c$	
$S' \rightarrow S$	$S \rightarrow A \mid B$	$A \rightarrow aA \mid b \mid \epsilon$	



自下而上分析：LR(0)分析：实现思想

- 构造LR(0)项目集规范族的方法（之二）
 - 第二步：构造文法项目集的闭包。

假定 I 是文法 G' 的任一项目集，则构造 I 的闭包 $\text{closure}(I)$ 的方法如下：

- ① I 中的每一个项目皆属于 $\text{closure}(I)$;
 - ② 若形如 $A \rightarrow \alpha \cdot B \beta$ ($B \in V_N$) 的项目属于 I ，则对 G' 中的任何产生式 $B \rightarrow \gamma$ 的项目 $B \rightarrow \cdot \gamma$ 也属于 $\text{closure}(I)$;
 - ③ 重复上述步骤，直至不再有新的项目加入 $\text{closure}(I)$ 为止；
-



自下而上分析：LR(0)分析：实现思想

■ 构造LR(0)项目集规范族的方法（之二）

■ 第三步：求状态转换函数 $GO(I, X)$ 。

若 I 是文法 G 的一个项目集， X 为 G' 的符号，则

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

其中 $J = \{ \text{形如 } A \rightarrow \alpha X \beta \text{ 的项目} \mid A \rightarrow \alpha \cdot X \beta \in I \}$ 。

■ 第四步：构造文法 G 的LR(0)项目集规范族

```
itemsets( $G'$ ) {  
     $C = \text{closure}(\{ [S' \rightarrow \cdot S] \})$ ;  
    do {  
        if ( 对  $C$  的每个项目集  $I$  和每个文法符号  $X$ , 若  $GO(I, X)$  非空且不在  $C$  中)  
            把  $GO(I, X)$  加入  $C$  中;  
    } while (没有更多的项目可以加入  $C$ );  
}
```



自下而上分析：LR(0)分析：实现思想

若一个文法 G 的识别活前缀的 DFA 的每一个项目不存在

- ① 即含移进项目又含归约项目；
- 或 ② 含有多个归约项目；

则每个项目集的项目相容，称 G 是一个 LR(0) 文法。



自下而上分析：LR(0)分析：实现思想

■ 从DFA构造LR(0)分析表的算法

- ① 对应分析表中 $\text{action}(M, a) = S_N$ ($a \in V_T$), 在DFA中为从状态M出发, 经过一条a弧到达状态 N;
 - ② 对应分析表中 $\text{action}(M, a) = r_n$ ($a \in V_T$), 在DFA中, 应对应于归约状态, 该状态中的文法产生式编号为 n ;
 - ③ 对分析表中 $\text{GOTO}(M, B) = N$ ($B \in V_N$), 在DFA中为从状态M出发, 经过一条B弧到达状态 N;
 - ④ 对action表中的 “acc”即对应 DFA 中的惟一终态。
-



自下而上分析：LR(0)分析：实现思想

算法:构造LR(0)分析表

输入：文法G和文法G的LR(0)项目集规范族C和GO函数

输出：文法G的LR(0)分析表

设 $C = \{I_0, I_1, \dots, I_n\}$ ，每个项目集 I_k 的下标 k 作为分析器的状态。

① 若 $GO(I_k, a) = I_j$ 且项目 $A \rightarrow \alpha \cdot a \beta \in I_k$ ，则置 $action(K, a) = S_j$ ；

② 若 $GO(I_k, A) = I_j (A \in V_N)$ ，则置 $GOTO(K, A) = j$ ；

③ 若 $A \rightarrow \alpha \cdot \in I_k$ ，则对所有终结符 a 或结束符 “#”，置 $action(K, a) = r_j$ 或 $action(K, \#) = r_j$ 。（其中假设产生式 $A \rightarrow \alpha$ 是文法第 j 个产生式）

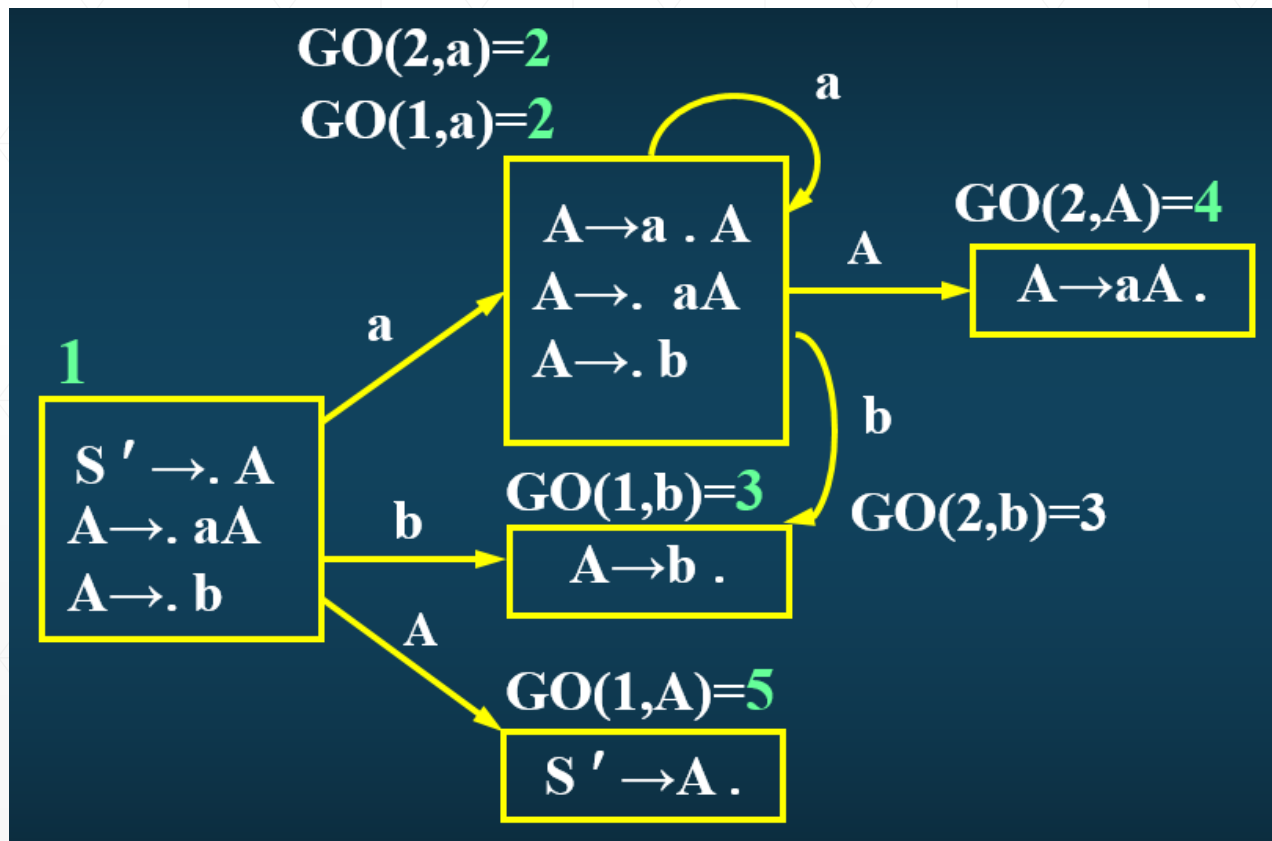
④ 若 $S' \rightarrow S \in I_k$ ，则置 $action(K, \#) = acc$ ；

⑤ 表中空白置出错标志。



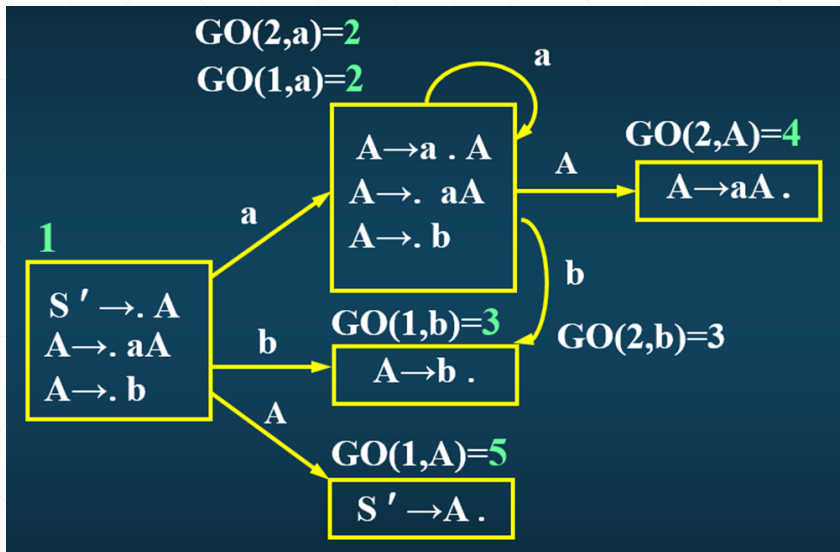
自下而上分析：LR(0)分析：实现思想

$S' \rightarrow A$ $A \rightarrow aA \mid b$





自下而上分析：LR(0)分析：实现思想



state	ACTION表			GOTO表
	a	b	#	A
1	S_2	S_3		5
2	S_2	S_3		4
3	r_3	r_3	r_3	
4	r_2	r_2	r_2	
5			$acc(r_1)$	



自下而上分析：LR(0)分析：实现思想

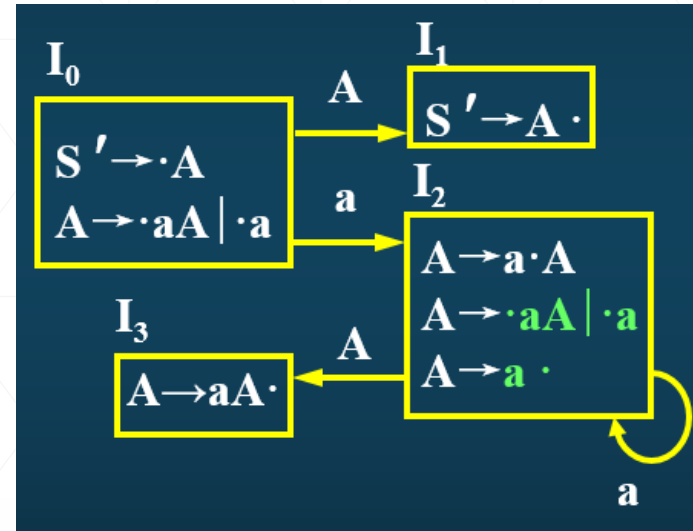
G : $A \rightarrow aA \mid a$

文法G拓广为

G' : $S' \rightarrow A$ ①

$A \rightarrow aA$ ②

$A \rightarrow a$ ③



state	action		goto
	a	#	
0	S ₂		1
1		acc	
2	S ₂ r ₃	r ₃	3
3	r ₂	r ₂	



自下而上分析：LR(0)分析：实现思想

在识别活前缀的DFA某一状态中，若既含有圆点不在最后的移进项目，又含有圆点在最后的归约项目，则称该项目集存在**移进 - 归约**冲突。若含有两个或两个以上圆点在最后的归约项目，则称该项目集存在**归约 - 归约**冲突。

💧 注意：

冲突情况都与**归约**动作相关。



自下而上分析：LR(0)分析：实现思想

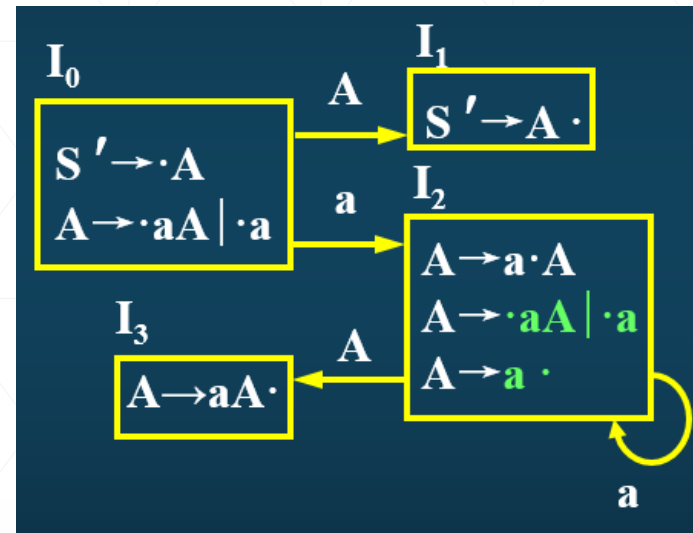
G : $A \rightarrow aA \mid a$

文法G拓广为

G' : $S' \rightarrow A$ ①

$A \rightarrow aA$ ②

$A \rightarrow a$ ③



state	action		goto
	a	#	
0	S_2		1
1		acc	
2	S_2 r_3	r_3	3
3	r_2	r_2	



自下而上分析：LR(0)分析：实现思想

在识别活前缀的DFA某一状态中，若既含有圆点不在最后的移进项目，又含有圆点在最后的归约项目，则称该项目集存在**移进 - 归约**冲突。若含有两个或两个以上圆点在最后的归约项目，则称该项目集存在**归约 - 归约**冲突。

💧 注意：

冲突情况都与**归约**动作相关。



自下而上分析：SLR(1)分析

LR(0)分析表构造算法对含有归约项目 $A \rightarrow a \cdot$ 的项目集 I_i ，不管当前输入符号为何，皆把action子表相应于状态 I_i 的那一行的诸元素都指定为 r_j （其中 j 为产生式 $A \rightarrow a \cdot$ 的编号）。

	a_1	a_2	...	a_n
...				
S_i	r_j	r_j	...	r_j
...				



自下而上分析：SLR(1)分析

设上下文无关文法G，S是文法的开始符号，对于文法G的任何非终结符A

$$\text{FOLLOW}(A) = \{a \mid S \Rightarrow \dots Aa \dots, a \in V_T\}$$

若 $S \Rightarrow \dots A$ ，则令 $\# \in \text{FOLLOW}(A)$ 。

对含有冲突的项目集 I_i ：

$$I_i = \{X \rightarrow \alpha \cdot b \beta, A \rightarrow \alpha \cdot, B \rightarrow \alpha \cdot\}$$

考察 $\text{FOLLOW}(A)$ ， $\text{FOLLOW}(B)$ 及 $\{b\}$ ，若 $\text{FOLLOW}(A) \cap \{b\} = \Phi$ ，

$$\text{FOLLOW}(B) \cap \{b\} = \Phi, \text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \Phi$$

则对任何输入符号a：

- (1) 当 $a = b$ 时，置 $\text{action}(I_i, b) = \text{“移进”}$ ；
 - (2) 当 $a \in \text{FOLLOW}(A)$ 时，置 $\text{action}(I_i, a) = \{\text{按产生式 } A \rightarrow \alpha \text{ 归约}\}$ ；
 - (3) 当 $a \in \text{FOLLOW}(B)$ 时，置 $\text{action}(I_i, a) = \{\text{按产生式 } B \rightarrow \alpha \text{ 归约}\}$ ；
 - (4) 当a不属于上述三种情况时，置 $\text{action}(I_i, a) = \text{“error”}$ 。
-



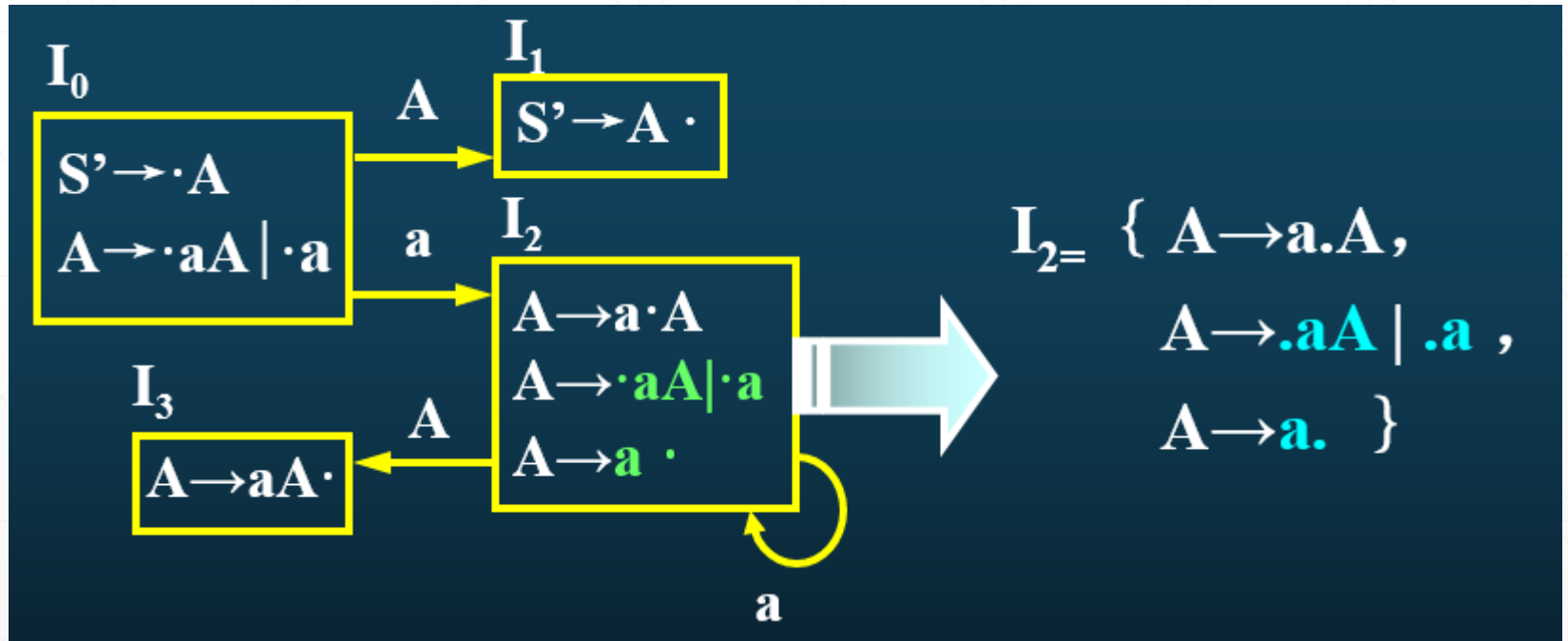
自下而上分析：SLR(1)分析

例：设拓广文法 G' 为

$$S' \rightarrow A \text{ ①}$$

$$A \rightarrow aA \text{ ②}$$

$$A \rightarrow a \text{ ③}$$





自下而上分析：SLR(1)分析

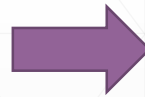
对 $I_2 = \{ A \rightarrow a.A, A \rightarrow .aA \mid .a, A \rightarrow a. \}$ 使用SLR(1)方法，有：

$$\text{FOLLOW}(A) \cap \{a\} = \{\#\} \cap \{a\} = \Phi$$

冲突可用SLR(1)方法得到解决，文法G是SLR(1)文法。

G' 的SLR(1)分析表 ?

state	action		goto
	a	#	
0	S ₂		1
1		acc	
2	S ₂	r ₃	3
3	r ₂	r ₂	



G' 的SLR(1)分析表

state	action		goto
	a	#	
0	S ₂		1
1		acc	
2	S ₂	r ₃	3
3		r ₂	



FOLLOW(A) = { # }



自下而上分析：SLR(1)分析

按照SLR(1)方法构造的文法G的LR分析表，如果每个入口不含多重定义，则称它为G的SLR(1)分析表。具有SLR(1)分析表的文法G称为SLR(1)文法。使用SLR(1)分析表的语法分析器称作SLR(1)分析器。

SLR(1)分析表构造 = SLR(1)方法 + LR(0)分析表构造

SLR(1)方法 = LR(0)方法 + “冲突”解决方法