



中国科学技术大学
University of Science and Technology of China

语法分析 V

《编译原理和技术(H)》、《编译原理(H)》

张昱

0551-63603804, yuzhang@ustc.edu.cn

中国科学技术大学
计算机科学与技术学院



3.6 LR解析器

(**L**-scanning from left to right; **R**- rightmost derivation in reverse)

- LR解析算法：效率高

- LR分析表的构造技术

简单的LR(SLR)、规范的LR、向前看LR(LALR)



LR(k)解析技术

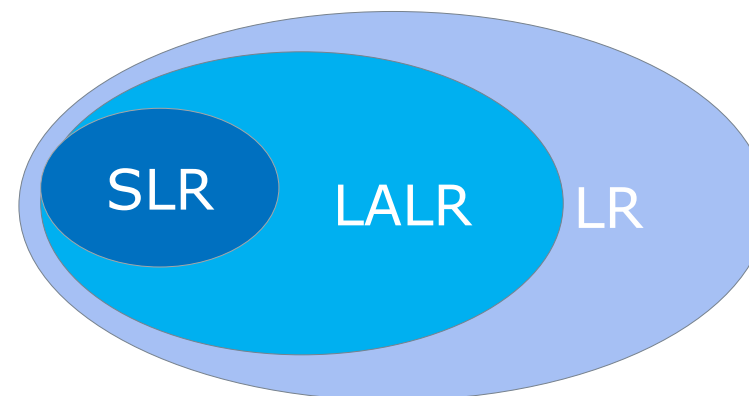
□ LR(k)

- **L**是指从左向右扫描输入,
- **R**是指构造最右推导的逆,
- **k**是指在决定分析动作时向前查看的符号个数,
(*k*)省略时, 表示*k*是1

□ LR解析算法基于LR分析表, 后者有三种构造技术

- SLR: 简单的LR方法
- LALR: 向前搜索的LR方法
- LR: 规范的LR方法

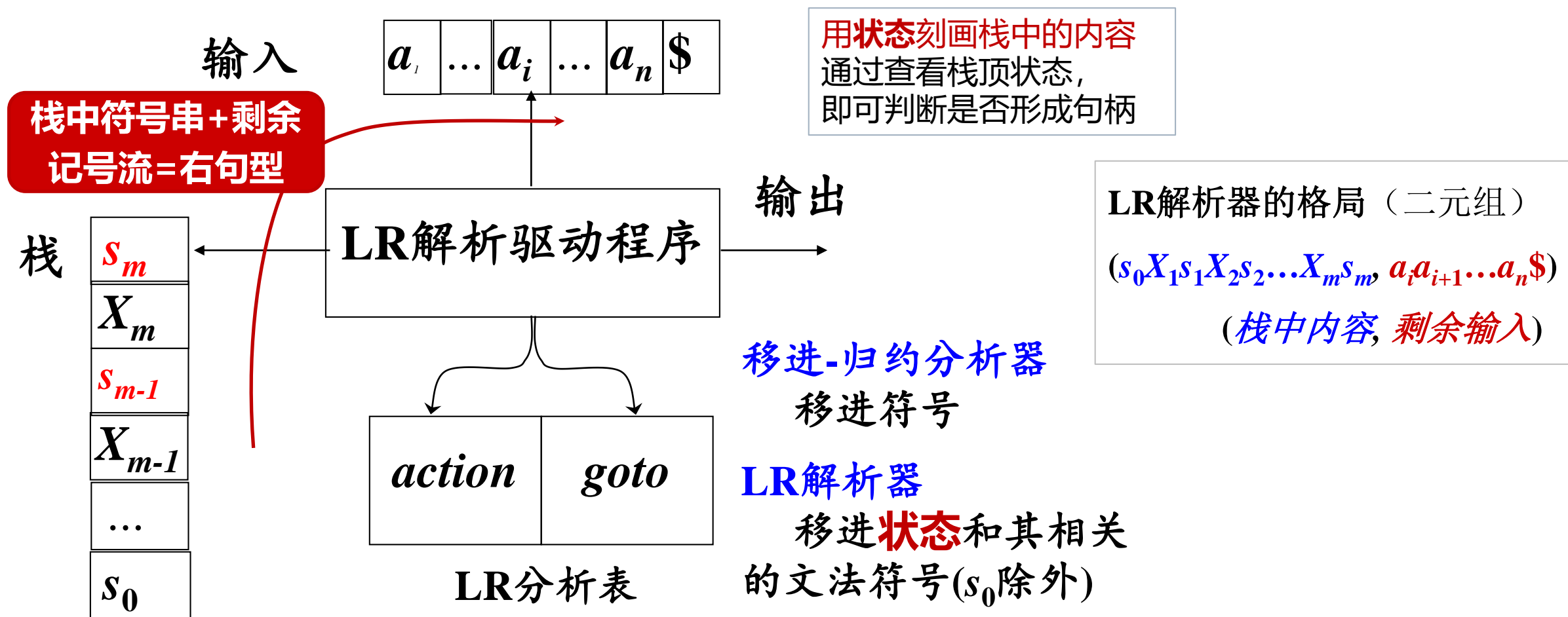
三种方法能表达的文法范围如右图所示





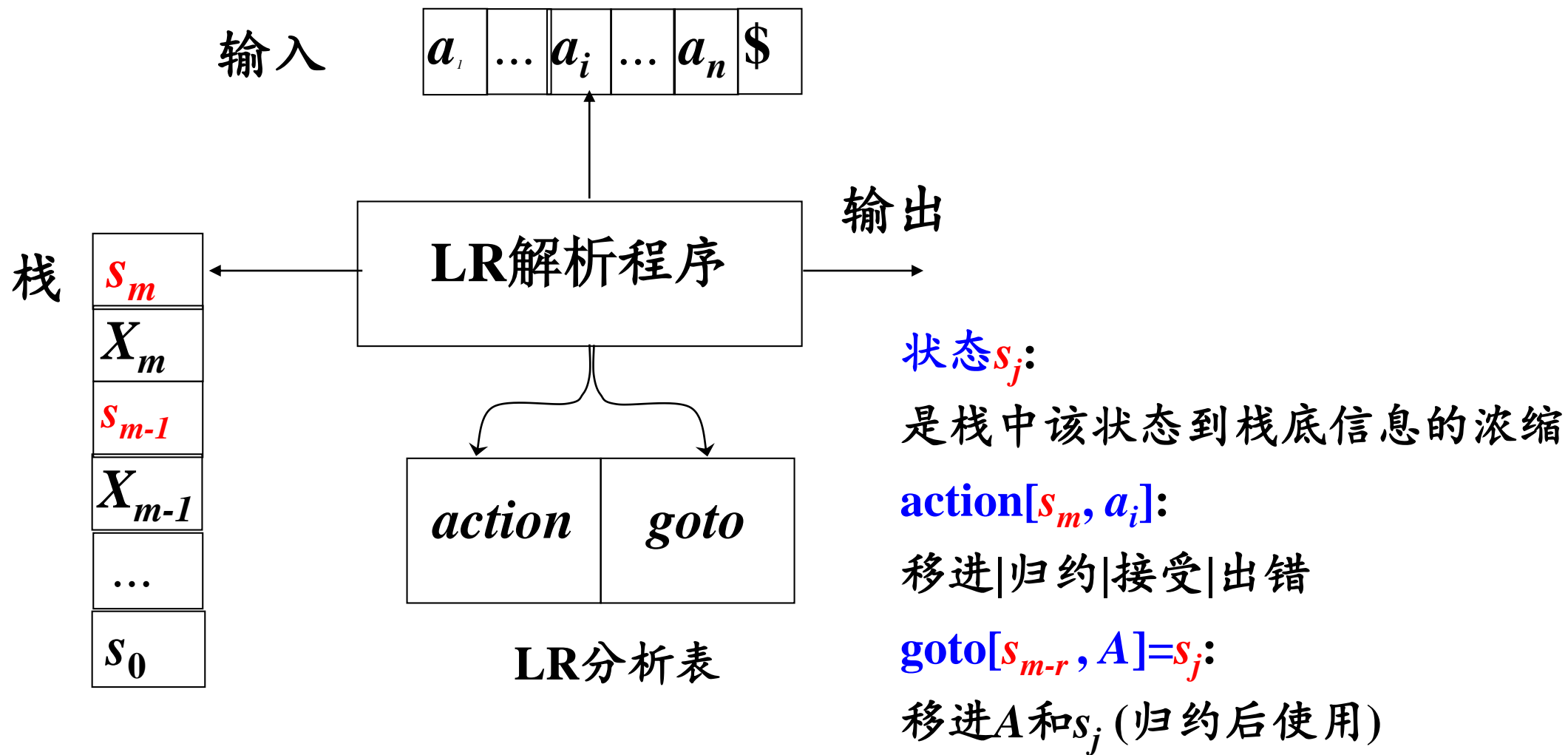
LR解析算法: LR系列解析器的模型

□ 如何快速识别栈的顶部是否形成句柄? → 引入抽象的**状态**





LR解析算法: LR系列解析器的模型





LR解析: 基本概念

□ 活前缀 (viable prefix)

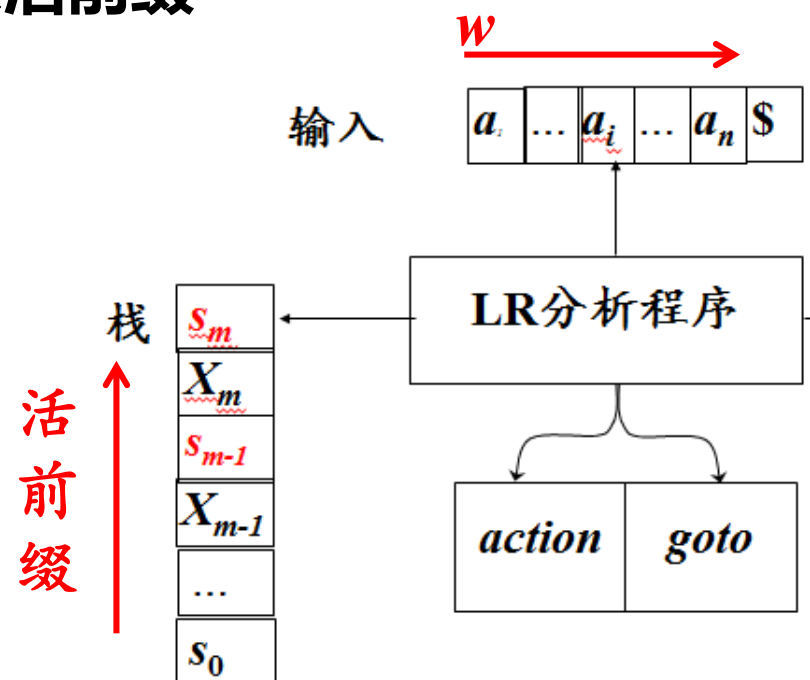
- 右句型的前缀 $\gamma\beta$, 该前缀不超过最右句柄的右端

$$S \Rightarrow_{rm}^* \gamma A w \Rightarrow_{rm} \gamma\beta w$$

- $\gamma\beta$ 的任何前缀 (包括 ϵ 和 $\gamma\beta$ 本身) 都是活前缀
- w 仅包含终结符

- 对应到LR解析模型上的特点

- 活前缀: 是LR解析栈中从栈底到栈顶的语法符号连接形成的串
- w : 输入缓冲区中剩余的记号串





LR解析算法：举例

实际可不用移进

例 $E \rightarrow E + T / E \rightarrow T$

3.24 $T \rightarrow T * F / T \rightarrow E$

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

LR分析表

si 移进当前输入符号和状态 i

rj 按第 j 个产生式进行归约

acc 接受

状态	动作						转移		
	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			



状态	动作						转移		
	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			



LR解析算法：举例

栈	输 入	动 作
0	id * id + id \$	移进 (查动作表)
0 id 5	* id + id \$	

状态	动作						转移		
	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			



LR解析算法：举例

栈	输 入	动 作
0	id * id + id \$	移进
0 id 5	* id + id \$	按 $F \rightarrow id$ 归约
0 F 3	* id + id \$	

状态	动作					转移		
	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			

1. 查 $\text{action}[5, *] = r6$ 归约
2. 按 $r6$ 执行归约($F \rightarrow id$):
 - 从栈中弹出|id|个状态-符号对
 - 查 $\text{goto}[0, F] = 3$
 - 将($F, 3$)入栈



LR解析算法：举例

栈	输 入	动 作
0	id * id + id \$	移进
0 id 5	* id + id \$	按 $F \rightarrow id$ 归约
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	

状态	动作						转移		
	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			



LR解析算法：举例

栈	输 入	动 作
0	id * id + id \$	移进
0 id 5	* id + id \$	按 $F \rightarrow id$ 归约
0 F 3	* id + id \$	按 $T \rightarrow F$ 归约
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	

状态	动作						转移		
	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			



LR解析算法： 举例

栈	输 入	
0	id * id + id	
0 id 5	* id + id	
0 F 3	* id + id	
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	

状态	动作						转移		
	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			



LR解析算法：举例

栈	输入	
0	id * id + id	
0 id 5	* id + id	
0 F 3	* id + id	
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按F → id归约
0 T 2 * 7 F 10	+ id \$	

状态	动作						转移		
	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			



LR解析算法：举例

栈	输入	
0	id * id + id	
0 id 5	* id + id	
0 F 3	* id + id	
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约
...

状态	动作					转移			
	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			



LR解析算法：举例

栈	输入	
0	id * id + id	
0 id 5	* id + id	
0 F 3	* id + id	
0 T 2	* id + id \$	移进
0 T 2 * 7	id + id \$	移进
0 T 2 * 7 id 5	+ id \$	按 $F \rightarrow id$ 归约
0 T 2 * 7 F 10	+ id \$	按 $T \rightarrow T * F$ 归约
...
0 E 1	\$	接受

状态	动作					转移			
	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			

归约为 开始符号

完成合法输入串的分析



LR解析: 基本概念

□ LR文法(LR grammar)

- 能为之构造出所有条目（若存在）**都唯一**的LR分析表

□ LR分析表

- 移进+ goto（转移函数）：**本质上是识别活前缀的DFA**

状态	动 作	转 移
	id + * () \$	<i>E</i> <i>T</i> <i>F</i>
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



LR方法与LL方法的比较

	LR(1)方法	LL(1)方法
建立分析树的方式	自下而上	自上而下
归约还是推导	规范归约	最左推导
决定使用产生式的时机		

$S \Rightarrow \dots \Rightarrow \delta A b w \Rightarrow \delta l \beta b w$

$A \rightarrow l\beta$

LL(1)决定用该
产生式的位置



LR方法与LL方法的比较

	LR(1)方法	LL(1)方法
建立分析树的方式	自下而上	自上而下
归约还是推导	规范归约	最左推导
决定使用产生式的时机		

$S \Rightarrow \dots \Rightarrow \delta A b w \Rightarrow \delta l \beta b w$

$A \rightarrow l\beta$

LL(1)决定用该
产生式的位置

LR(1)决定用该
产生式的位置



LR方法与LL方法的比较

	LR(1)方法	LL(1)方法
建立分析树的方式	自下而上	自上而下
归约还是推导	规范归约	最左推导
决定使用产生式的时机	看见产生式右部推出的 整个 终结字符串后，才确定用哪个产生式归约	看见产生式右部推出的 第一个 终结字符后，便要确定用哪个产生式推导

$S \Rightarrow \dots \Rightarrow \delta A b w \Rightarrow \delta l \beta b w$

$A \rightarrow l\beta$

LL(1)决定用该
产生式的位置

LR(1)决定用该
产生式的位置



3.5 LR解析器

(**L**-scanning from left to right; **R**- rightmost derivation in reverse)

□ LR解析算法：效率高

□ LR分析表的构造技术

简单的LR(SLR)、规范的LR、向前看LR(LALR)



LR(1)分析表的构造

□ 文法的LR(0)项目与LR(1)项目

- 产生式右部加点以刻画分析所处的位置 $E \rightarrow E + \cdot T$
- 1: 是引入长度为1的搜索符, 指示产生式右部之后的符号 $[E \rightarrow E + \cdot T, \$ / +]$

□ SLR方法: 简单的LR

- 构造LR(0)项目集规范族 \rightarrow 形成DFA状态 \rightarrow SLR分析表

□ LR方法: 规范的LR

- 构造LR(1)项目集规范族 \rightarrow 形成DFA状态 \rightarrow LR分析表

□ LALR方法: 向前搜索的LR

- 构造LR(1)项目集规范族 \rightarrow 形成DFA状态 \rightarrow 合并同心项目集 \rightarrow LALR分析表

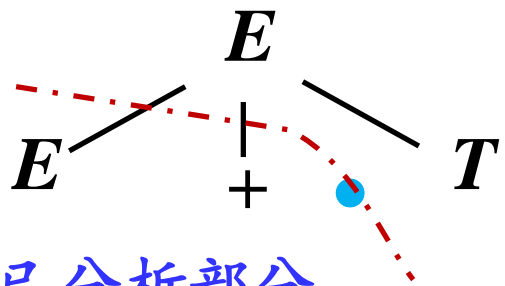


LR(0)项目和LR(1)项目

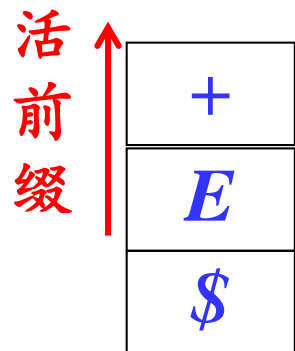
□ LR(0) 项目

- 在右部的某个地方**加点**的产生式
- 加点的目的是用来表示分析过程所处的位置 - 刻画分析的状态

$$E \rightarrow E + \cdot T$$



已分析部分
形成活前缀
在栈中



例 $A \rightarrow XYZ$ 对应四个LR(0)项目

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

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

$A \rightarrow \epsilon$ 只有一个项目和它对应

$$A \rightarrow \cdot$$

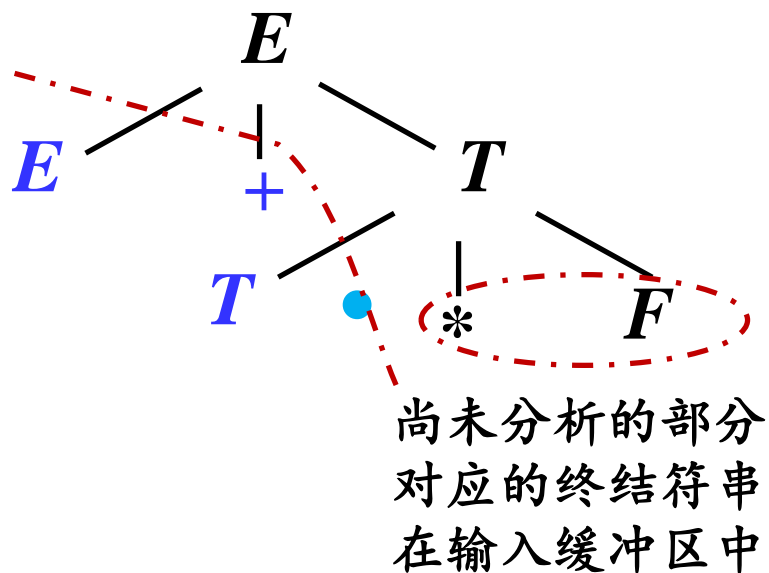
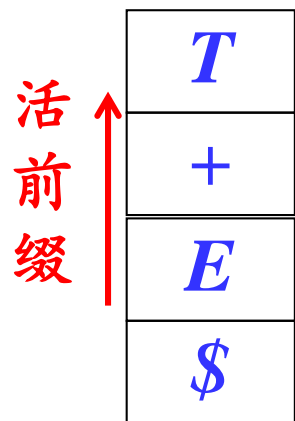


LR(0)项目和LR(1)项目

□ LR(0) 项目

- 在右部的某个地方**加点**的产生式
- 加点的目的是用来表示分析过程所处的位置 - 刻画分析的状态

$$T \rightarrow T \cdot * F$$





LR(0)项目和LR(1)项目

□ LR(0) 项目

新增产生式和新的开始符号

$S' \rightarrow S$
 $S \rightarrow BB$
 $B \rightarrow bB / a$

满足该语言的句子有:

aa
 baa
 $baba$
.....

■ 文法拓广: 旨在指示解析器

何时开始分析、何时完成分析

$[S' \rightarrow \cdot S]$
初始项目

$[S' \rightarrow S \cdot]$
结束项目

I_0 :

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot bB$

$B \rightarrow \cdot a$

核心项目

- 1) 初始项目;
- 2) 点不在最左端的项目

非核心项目

非初始项目且点在最左端的项目

可通过对核心项目求闭包来获得
为节省存储空间, 可省去

求LR(0)项目集的闭包closure(I) 图3.24

- I 中的每个项目都加入到closure(I)
- $[A \rightarrow \alpha \cdot B \beta] \in I$

$\forall B \rightarrow \gamma : [B \rightarrow \cdot \gamma] \in \text{closure}(I)$



LR(0)项目和LR(1)项目

□ LR(0)项目集规范族

初始项目: $[S' \rightarrow \cdot S]$

结束项目: $[S' \rightarrow S \cdot]$

I_0 :

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot bB$

$B \rightarrow \cdot a$

$S' \rightarrow S$

$S \rightarrow BB$

$B \rightarrow bB / a$

LR(1) 项目: $[A \rightarrow \alpha \cdot \beta, a]$
表示A之后紧跟a。
如果 $S \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$,
则a是w的第一个符号,
或者 w 是ε 且a是\$

求LR(0)项目集的闭包closure(I)

$[A \rightarrow \alpha \cdot B \beta] \in I$

$\forall B \rightarrow \gamma : [B \rightarrow \cdot \gamma] \in I$

□ LR(1)项目集规范族

新增搜索符

初始项目: $[S' \rightarrow \cdot S, \$]$

结束项目: $[S' \rightarrow S \cdot, \$]$

I_0 :

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

$S \rightarrow \cdot BB, \$$

$B \rightarrow \cdot bB, a/b$

$B \rightarrow \cdot a, a/b$

首终结符

aa\$

baa\$

baba\$

$\text{FIRST}(B) = \{a, b\}$

求LR(1)项目集的闭包closure(I)

$[A \rightarrow \alpha \cdot B \beta, a] \in I$

$\forall B \rightarrow \gamma : [B \rightarrow \cdot \gamma, b] \in I, b \in \text{FIRST}(\beta a)$

问题: LR(1)项目数量庞大 => 状态数偏多



LR分析表的构造

1. 拓广文法

$$S' \rightarrow S$$

$$S \rightarrow BB$$

$$B \rightarrow bB / a$$

2. 构造LR(0)项目集规范族或LR(1)项目集规范族

=>构造识别活前缀的DFA

活前缀: 某个右句型的一个前缀, 该前缀不超过该右句型的最右句柄的右端

右句型: 通过最右推导得到的句型

■ LALR解析 (LookAhead LR)

□ 通过合并同心的LR(1)项目集, 得到和SLR同样数量的状态

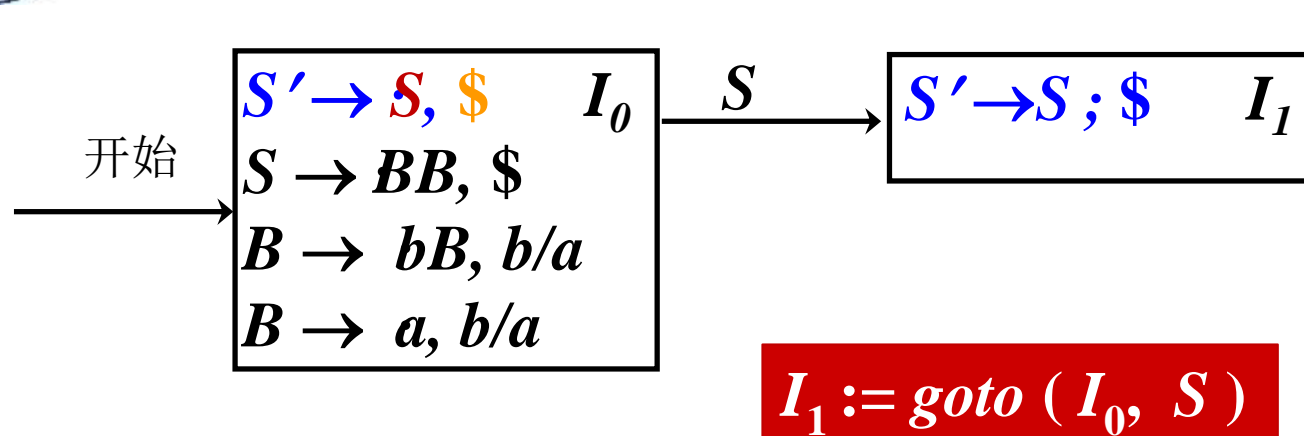
3. 从上述DFA构造LR分析表

注: LR(0)项目集规范族 => SLR分析表
LR(1)项目集规范族 => 规范的LR分析表
合并同心的LR(1)项目集规范族 => LALR分析表



构造识别活前缀的DFA

(以LR(1)项目集为例)



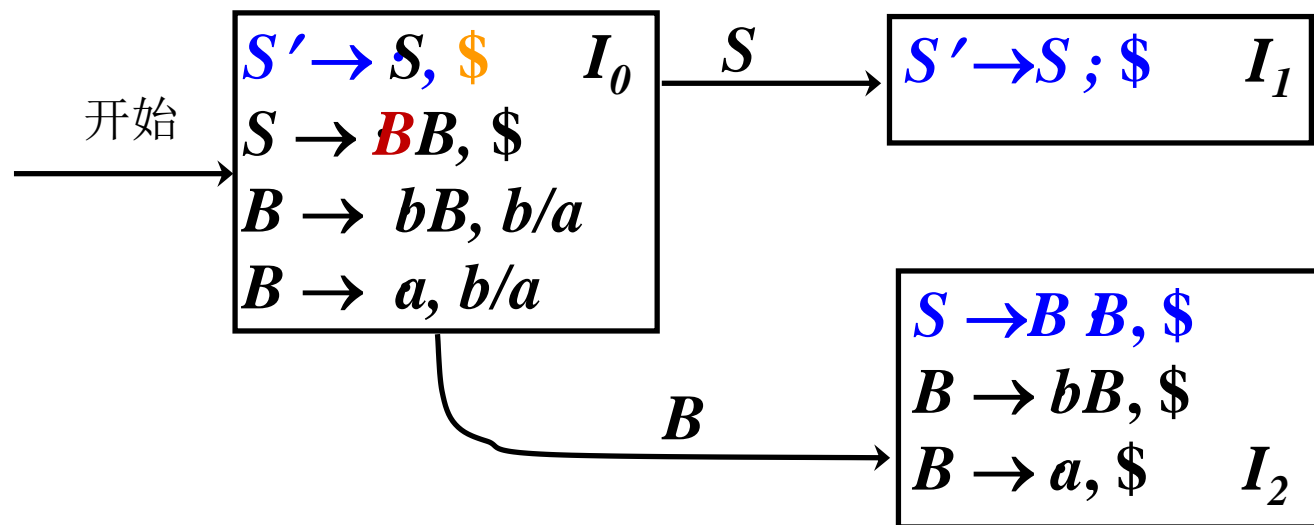
$S' \rightarrow S$

$S \rightarrow BB$

$B \rightarrow bB / a$



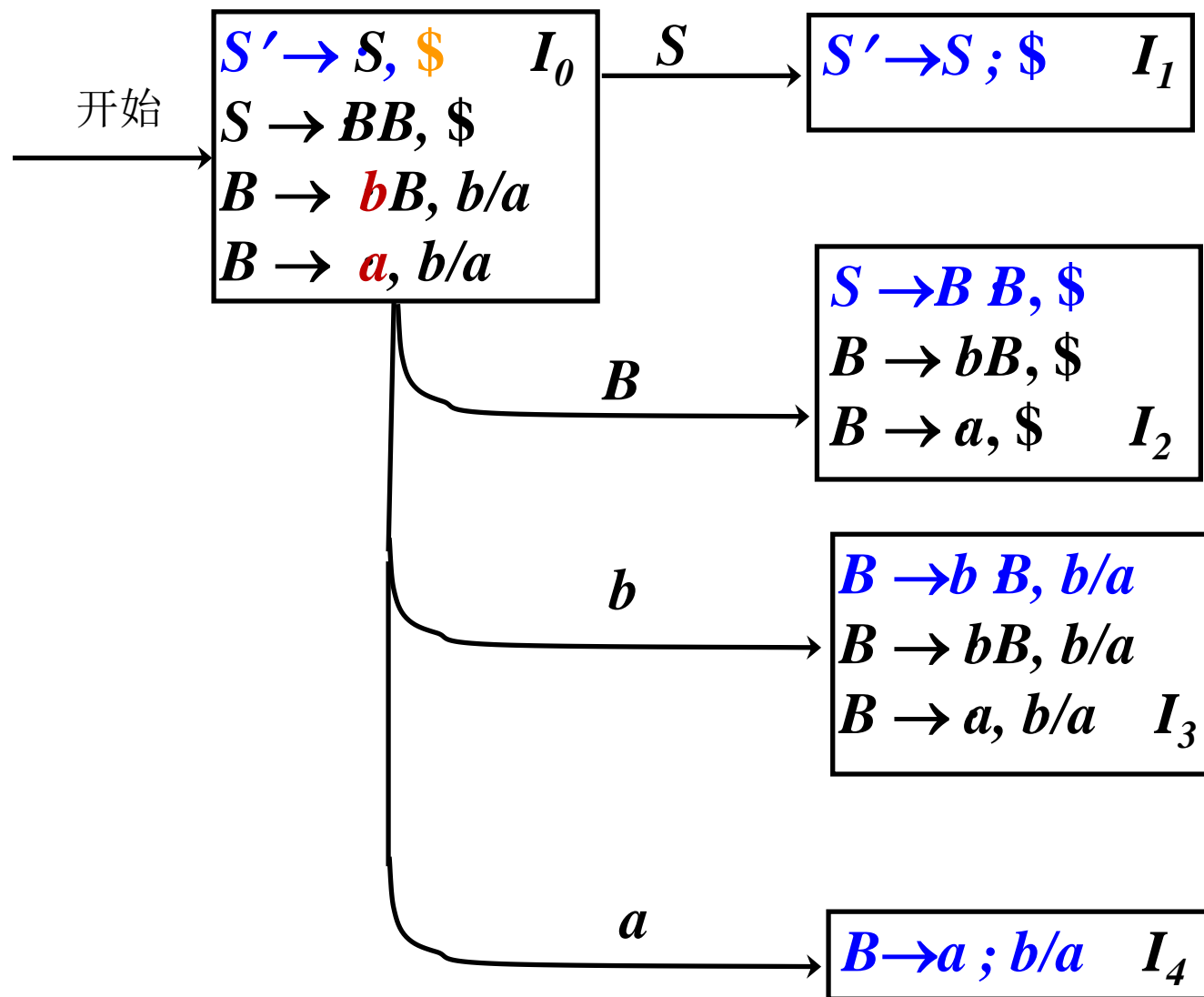
构造识别活前缀的DFA (以LR(1)项目集为例)





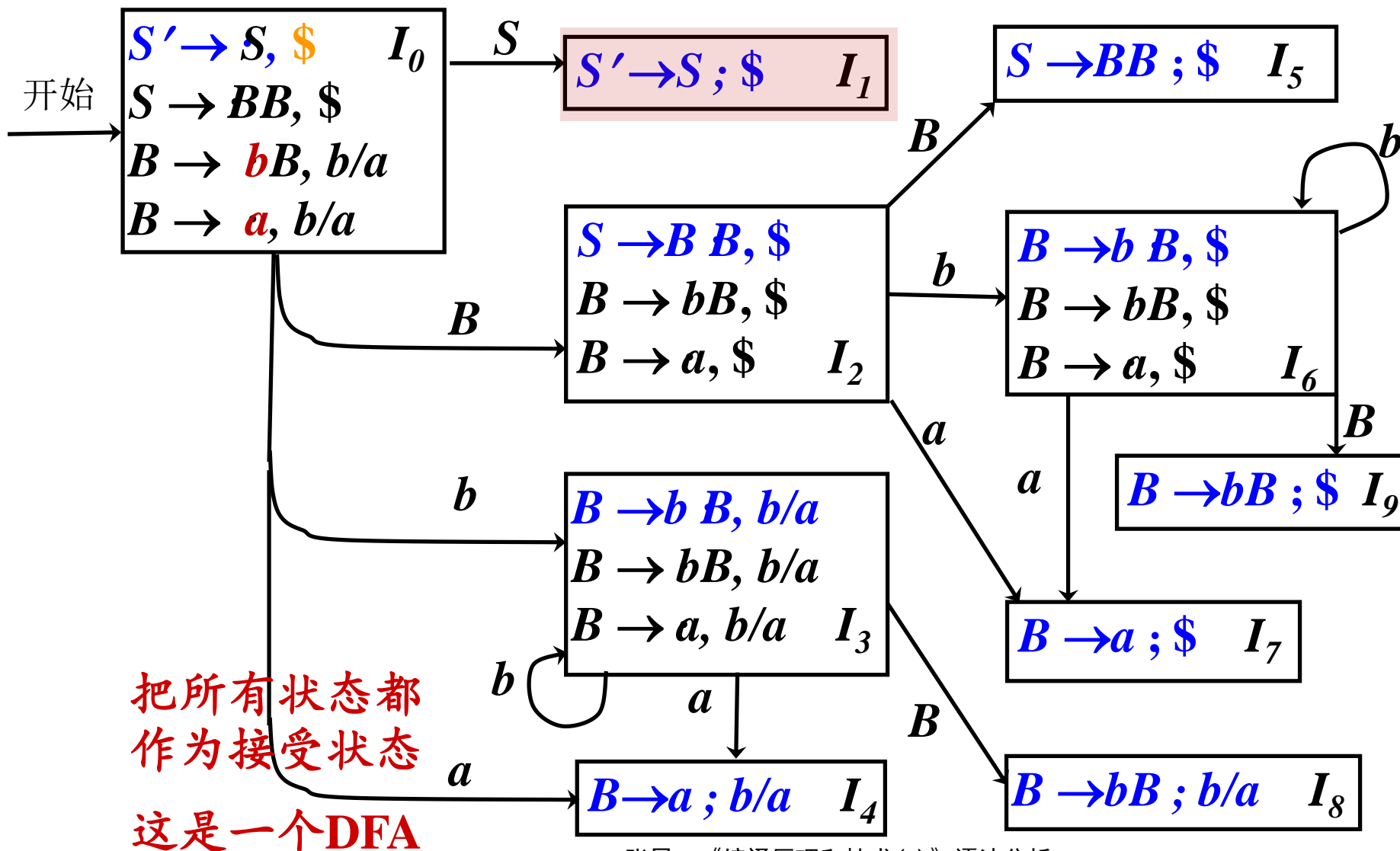
构造识别活前缀的DFA

(以LR(1)项目集为例)





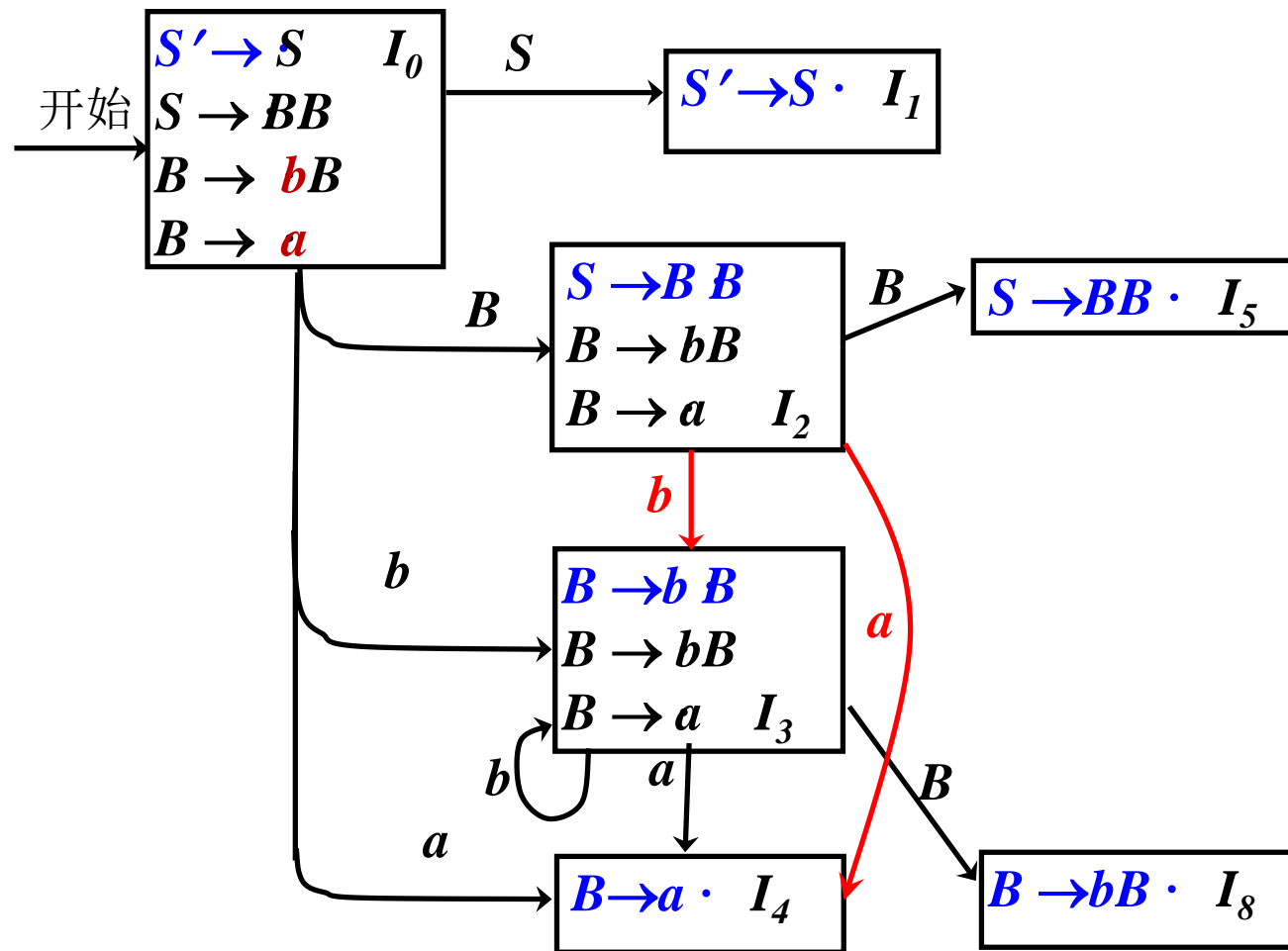
构造识别活前缀的DFA (以LR(1)项目集为例)



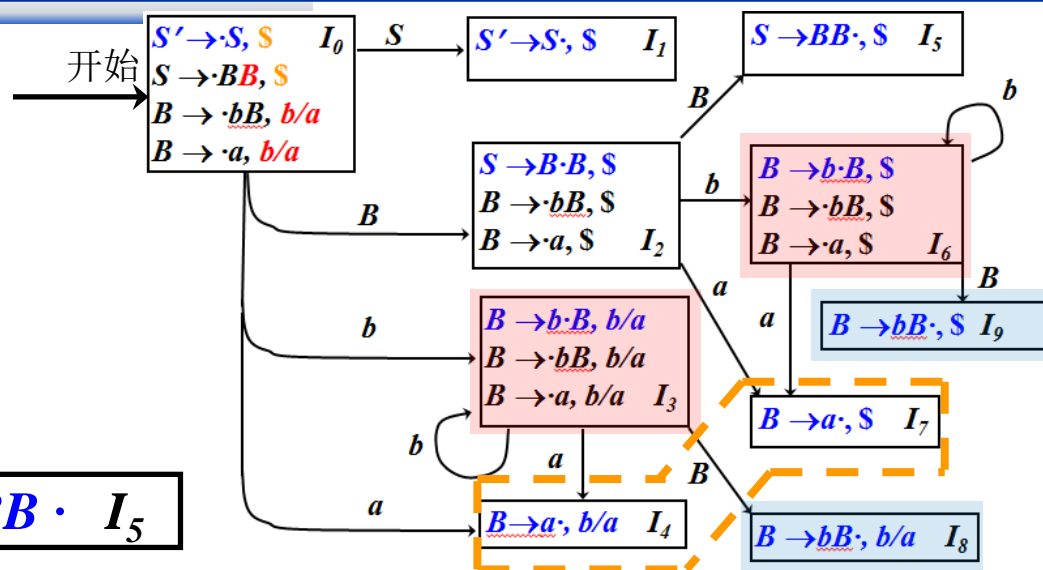


规范的LR vs. SLR 分析

SLR



规范的LR



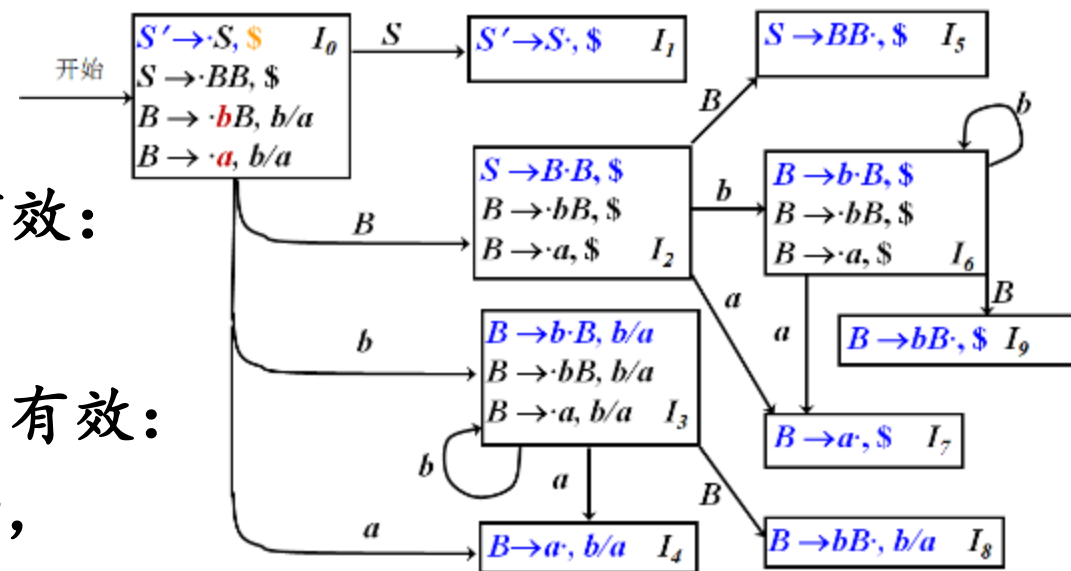
规范的LR分析
的状态数偏多



有效项目

□ 项目对活前缀的有效性

- LR(0)项目 $[A \rightarrow \alpha \cdot \beta]$ 对活前缀 $\gamma = \delta \alpha$ 有效:
如果存在推导 $S' \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$
- LR(1)项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对活前缀 $\gamma = \delta \alpha$ 有效:
如果存在推导 $S' \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$,
□ a 是 w 的第一个符号, 或者 w 是 ϵ 且 a 是 $\$$



□ 项目与活前缀之间的关系

- **活前缀** 是 DFA 中从初始状态到项目所在状态路径上的文法符号串联形成的串
- 一个活前缀 γ 的 **有效项目集** 是从这个 DFA 的初态出发, 沿着标记为 γ 的路径到达的那个项目集 (状态)



有效项目

- LR(1)项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对活前缀 $\gamma = \delta \alpha$ 有效:

如果存在着推导 $S' \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$, 其中:

- a 是 w 的第一个符号, 或者 w 是 ϵ 且 a 是 $\$$

项目与活前缀之间的关系

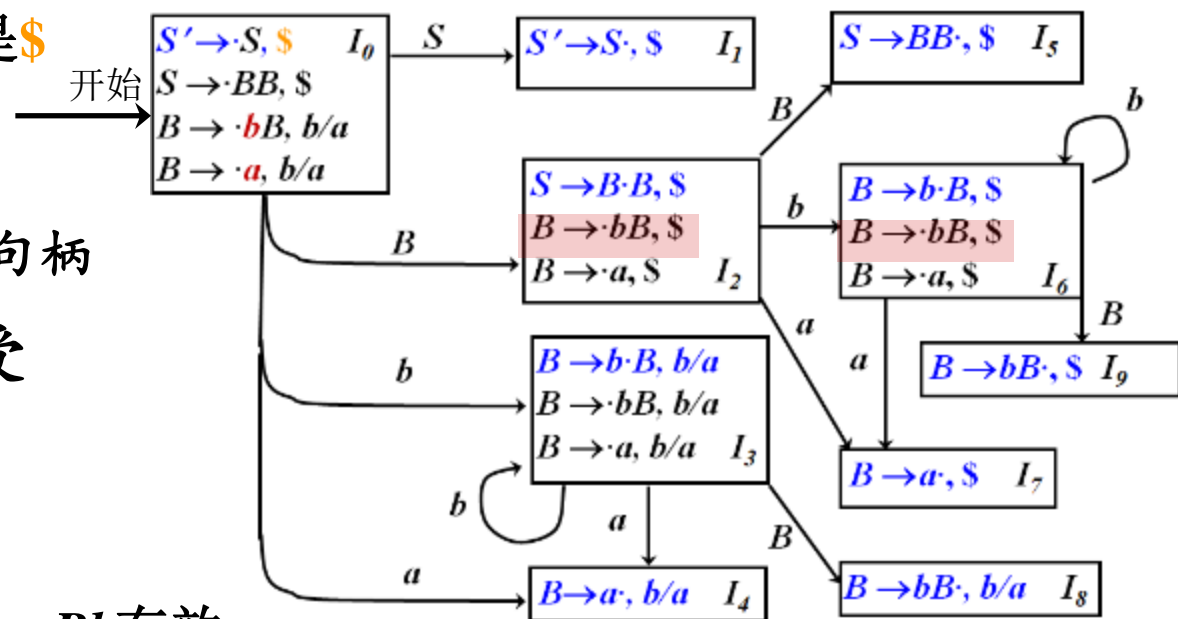
$S' \Rightarrow S \Rightarrow BB \Rightarrow BbB \Rightarrow BbbB$ bB 是最右句柄

- $BbbB$ 的所有前缀(活前缀)都可接受

- $I_9[B \rightarrow bB; \$]$ 对活前缀 $BbbB$ 是有效的
- $I_6[B \rightarrow bB, \$]$ 对活前缀 Bbb 是有效的
- I_2 和 I_6 中的 $[B \rightarrow \cdot bB, \$]$ 分别对活前缀 B 、 Bb 有效

- 活前缀 B 有多个有效项目

$S \rightarrow B \cdot B, \$$
$B \rightarrow \cdot bB, \$$
$B \rightarrow \cdot a, \$$
I_2





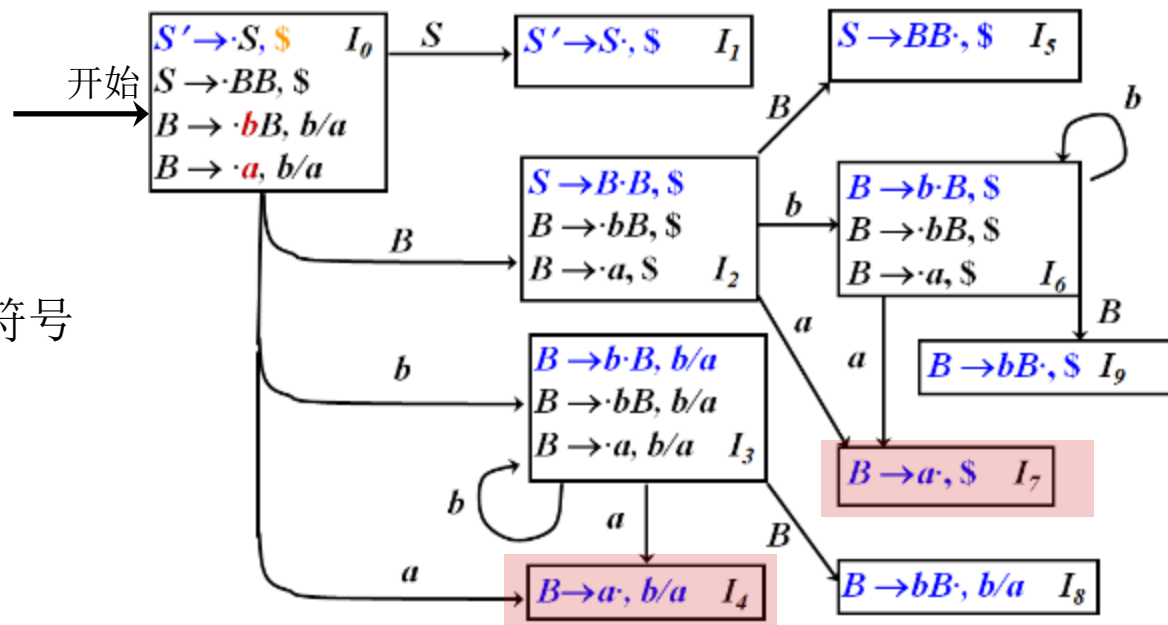
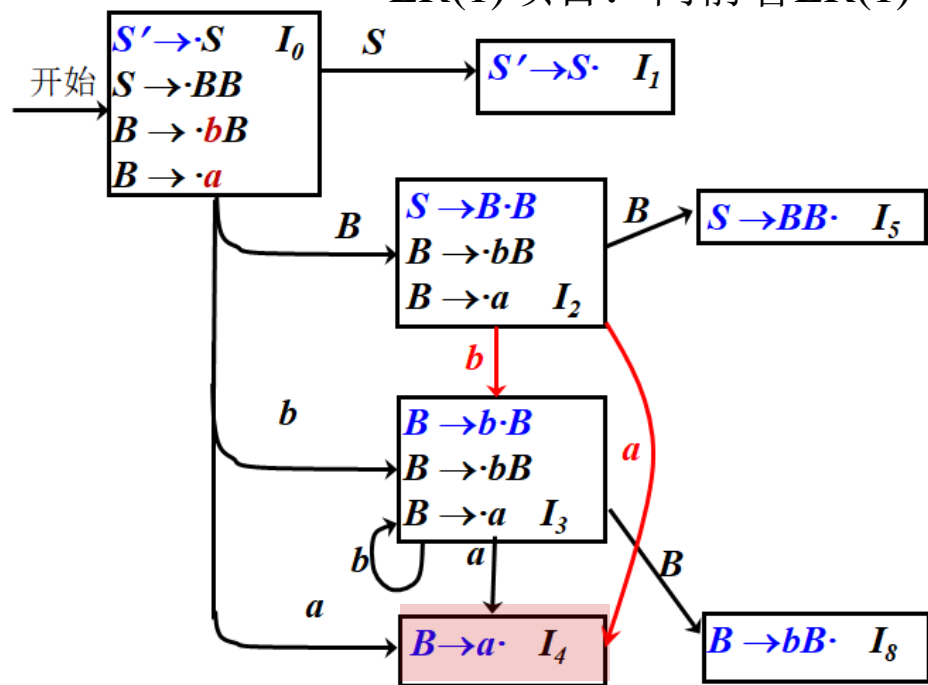
有效项目

■ 从LR(0)项目 $A \rightarrow \alpha \cdot \beta$ 或LR(1)项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对活前缀 $\delta \alpha$ 有效可以知道

- ✓ 如果 $\beta \neq \epsilon$, 应该移进
- ✓ 如果 $\beta = \epsilon$, 应该用产生式 $A \rightarrow \alpha$ 归约

LR(0)项目: 向前看Follow(A)中的符号

LR(1)项目: 向前看LR(1)项目中第2元的符号



Follow(B)={a, b, \$}

对于输入a\$:

LR(0)项目 $B \rightarrow a$: 面临a/b/\$归约

LR(1)项目 $[B \rightarrow a ; b/a]$: 面临a/b归约



从DFA构造SLR分析表

□ 状态 i 从LR(0)项目集 I_i 构造, 按如下方法确定 *action* 函数:

- ① 移进: 如果 $[A \rightarrow \alpha \textcolor{red}{a}\beta]$ 在 I_i 中, 并且 $\text{goto}(I_i, \textcolor{red}{a}) = I_j$, 那么置 $\text{action}[i, a]$ 为 $\textcolor{blue}{sj}$
- ② 归约: 如果 $[\textcolor{red}{A} \rightarrow \alpha \cdot]$ 在 I_i 中, 那么对 $\text{FOLLOW}(A)$ 中所有的 a , 置 $\text{action}[i, a]$ 为 $\textcolor{blue}{rj}$, j 是产生式 $A \rightarrow \alpha$ 的编号
- ③ 接受: 如果 $[\textcolor{red}{S'} \rightarrow \textcolor{red}{S} \cdot]$ 在 I_i 中, 那么置 $\text{action}[i, \$]$ 为 $\textcolor{blue}{acc}$

如果出现动作冲突, 那么该文法就不是SLR(1)的

■ 构造状态 i 的 *goto* 函数

□ 对所有的非终结符 A , 如果 $\text{goto}(I_i, A) = I_j$, 则 $\text{goto}[i, A] = j$

■ 不能由上面两步定义的条目都置为 **error**

■ 解析器的初始状态: 包含 $[\textcolor{red}{S'} \rightarrow \textcolor{red}{S}]$ 的项目集对应的状态



构造规范的LR分析表

□ 构造LR分析表，状态 i 的 $action$ 函数按如下确定

- ① 如果 $[A \rightarrow \alpha \textcolor{red}{a}\beta, b]$ 在 I_i 中, 且 $goto(I_i, a) = I_j$, 那么置 $action[i, a]$ 为 $\textcolor{blue}{sj}$
- ② 如果 $[A \rightarrow \alpha \cdot, \textcolor{red}{a}]$ 在 I_i 中, 且 $A \neq S'$, 那么置 $action[i, \textcolor{red}{a}]$ 为 $\textcolor{blue}{rj}$
- ③ 如果 $[S' \rightarrow S ; \$]$ 在 I_i 中, 那么置 $action[i, \$] = \textcolor{blue}{acc}$

如果用上面规则构造, 出现了冲突, 则文法就不是LR(1)的

- $goto$ 函数的确定: 如果 $goto(I_i, A) = I_j$, 那么 $goto[i, A] = j$
- 用上面规则未能定义的所有条目都置为 **error**
- 初始状态是包含 $[S' \rightarrow \textcolor{blue}{S}, \$]$ 的项目集对应的状态

SLR是根据Follow(A)来确定归约动作
这里是根据搜索符(上下文信息)来确定



LALR分析表

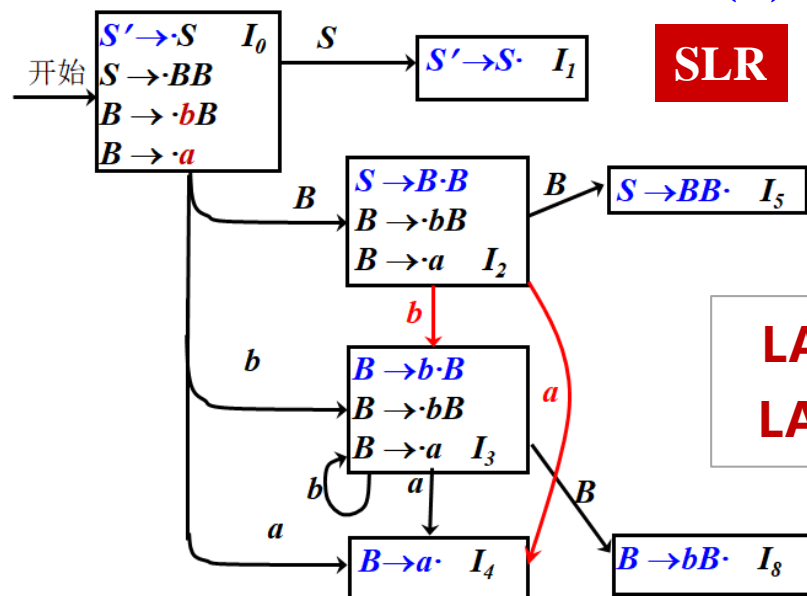
□ 同心的LR(1)项目集

两个项目集在略去搜索符后是相同的集合
右图有 3 对同心项目集

(I3和I6、I4和I7、I8和I9)

□ LALR分析表构造方法

■ 通过合并同心的LR(1)项目集来得到

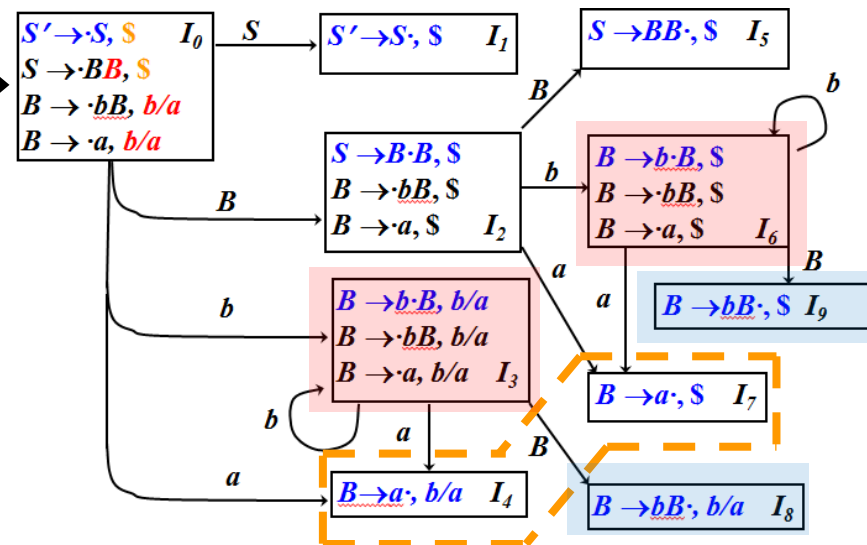


LALR与SLR有相同的状态数
LALR的归约：指定的搜索符

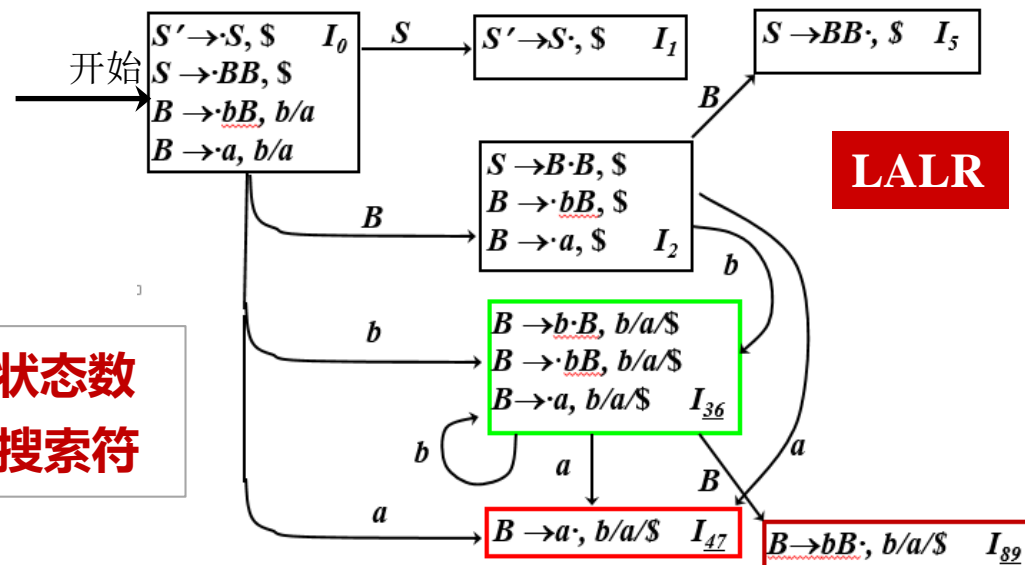
张昱：《编译原理和技术(H)》语法分析

(规范的)LR

开始

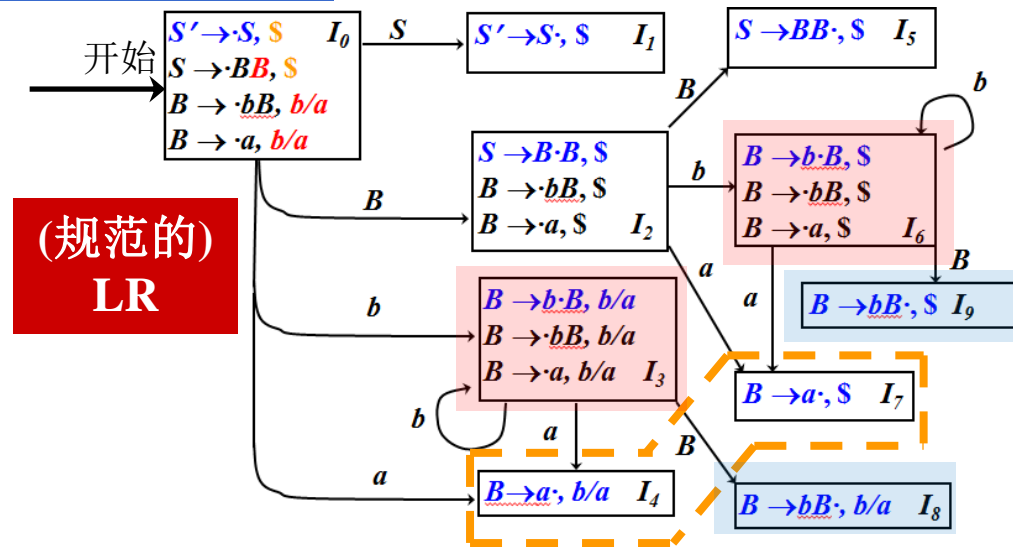
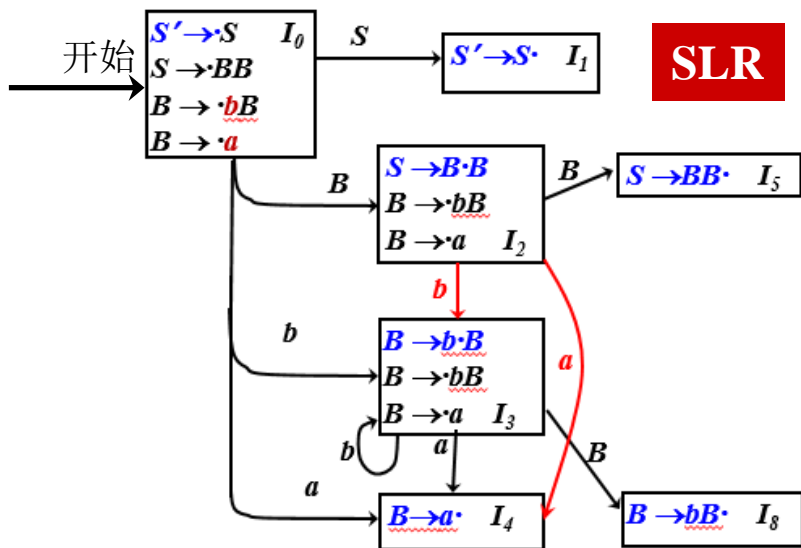


开始





SLR vs. LR vs. LALR解析



□ 同心集的合并不会引起新的移进-归约冲突

项目集1

项目集2

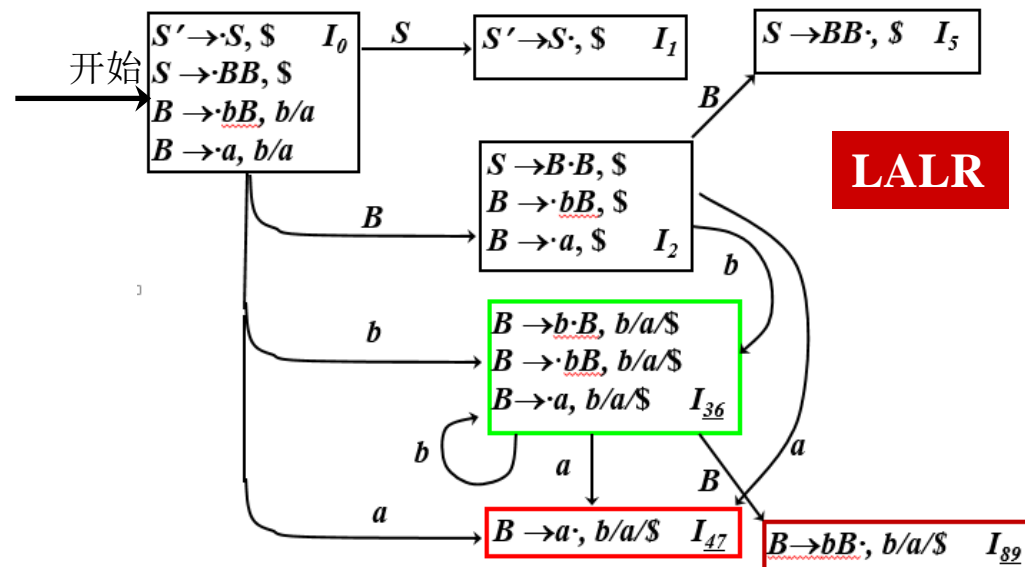
$[A \rightarrow \alpha ; a]$

$[B \rightarrow \beta a \gamma, b]$

...

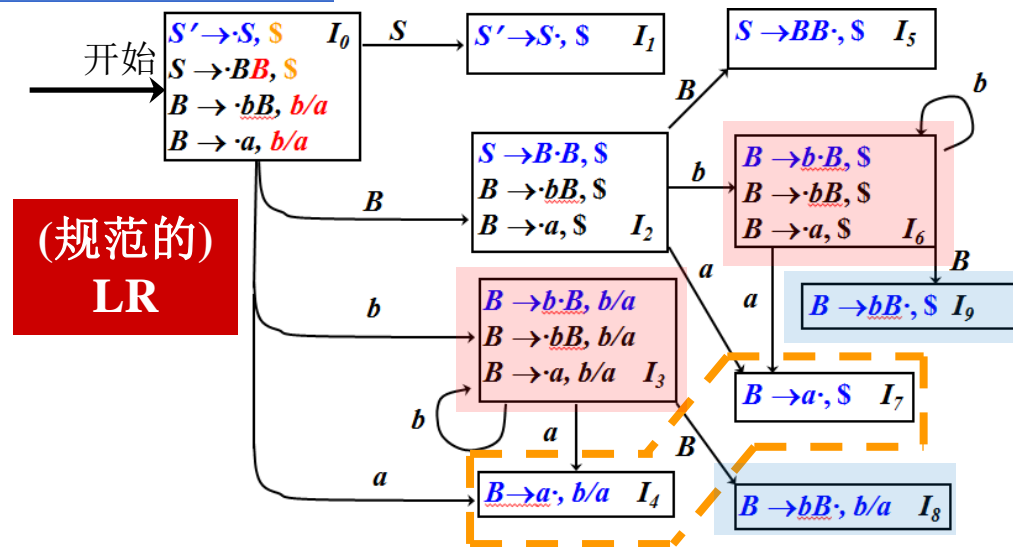
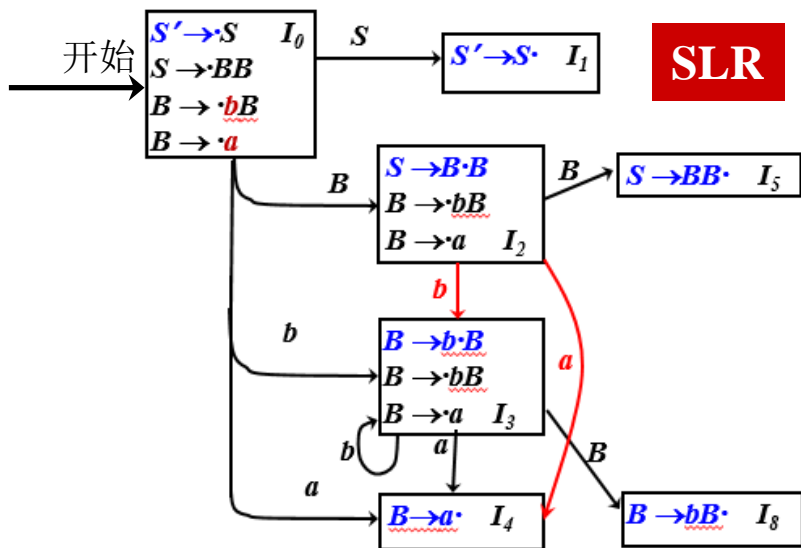
...

若合并后有冲突





SLR vs. LR vs. LALR解析



□ 同心集的合并不会引起新的移进-归约冲突

项目集1

$[A \rightarrow \alpha ; a]$

$[B \rightarrow \beta a \gamma, c]$

...

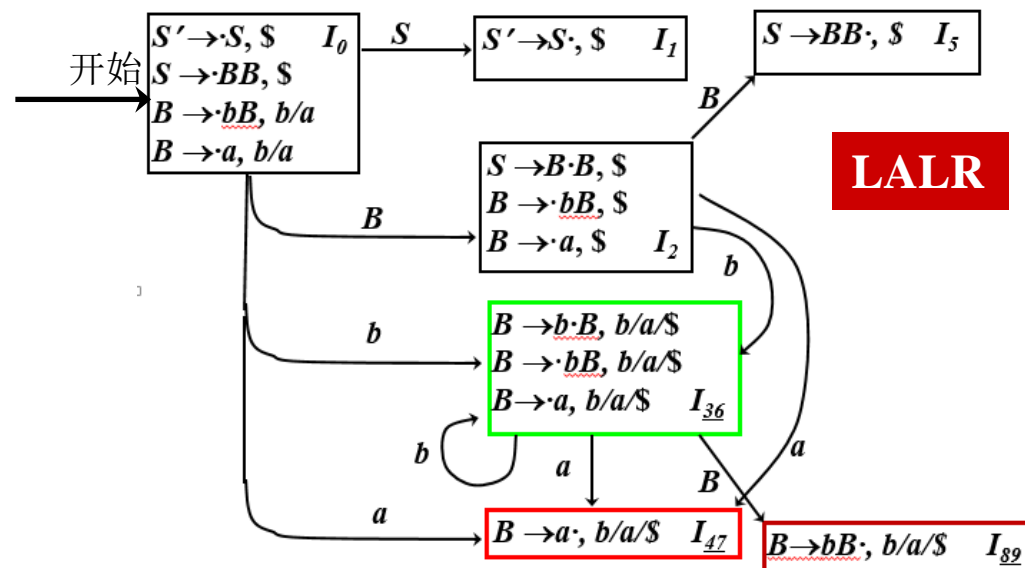
则合并前就有冲突

项目集2

$[B \rightarrow \beta a \gamma, b]$

$[A \rightarrow \alpha ; d]$

...





LALR vs. LR 分析

□ 同心的LR(1)项目集

- 两个项目集在略去搜索符后是相同的集合

□ 同心集的合并不会引起新的移进-归约冲突

□ 同心集的合并有可能产生新的归约-归约冲突

$S' \rightarrow S$

$S \rightarrow aAd \mid bBd \mid$

$aBe \mid bAe$

$A \rightarrow c$

$B \rightarrow c$

对 ac 有效的项目集

$A \rightarrow c ; d$

$B \rightarrow c ; e$

对 bc 有效的项目集

$A \rightarrow c ; e$

$B \rightarrow c ; d$

合并同心集之后

$A \rightarrow c ; d/e$

$B \rightarrow c ; d/e$

该文法是LR(1)的,
但不是LALR(1)的



SLR(1)文法的描述能力有限

$S \rightarrow V = E$

$S \rightarrow E$

$V \rightarrow * E$

$V \rightarrow \text{id}$

$E \rightarrow V$

$I_0:$

$S' \rightarrow \cdot S$

$S \rightarrow \cdot V = E$

$S \rightarrow \cdot E$

$V \rightarrow \cdot * E$

$V \rightarrow \cdot \text{id}$

$E \rightarrow \cdot V$

V

$I_2:$

$S \rightarrow V \cdot = E$

$E \rightarrow V \cdot$

$=$

$I_6:$

$S \rightarrow V = \cdot E$

$E \rightarrow \cdot V$

$V \rightarrow \cdot * E$

$V \rightarrow \cdot \text{id}$

该文法并不是二义的

$S \$ \Rightarrow V = E \$ \Rightarrow * E = E \$$

$S \$ \Rightarrow V = E \$$ 无右句型 $E = E$ ☹️

$S \$ \Rightarrow E \$ \Rightarrow V \$$

$I_2:$

项目 $S \rightarrow V \cdot = E$ 使得

$\text{action}[2,=]=s6$

项目 $E \rightarrow V \cdot$ 使得

$\text{action}[2,=]$ 为按 $E \rightarrow V$ 归约,

因为 $\text{Follow}(E) = \{=, \$\}$

产生移进-归约冲突



不是SLR(1)但是LR(1)的文法

$S \rightarrow V = E$

$S \rightarrow E$

$V \rightarrow * E$

$V \rightarrow \text{id}$

$E \rightarrow V$

$I_0:$
 $S' \rightarrow \cdot S, \$$
 $S \rightarrow \cdot V = E, \$$
 $S \rightarrow \cdot E, \$$
 $V \rightarrow \cdot * E, =$
 $V \rightarrow \cdot \text{id}, =$
 $E \rightarrow \cdot V, \$$

V

$I_2:$
 $S \rightarrow V \cdot = E, \$$
 $E \rightarrow V \cdot, \$$

$I_0:$
 $S' \rightarrow \cdot S$
 $S \rightarrow \cdot V = E$
 $S \rightarrow \cdot E$
 $V \rightarrow \cdot * E$
 $V \rightarrow \cdot \text{id}$
 $E \rightarrow \cdot V$

该文法并不是二义的

$SS \Rightarrow V = E S \Rightarrow * E = E S$
 $SS \Rightarrow V = E S$ 无句型 $E = E$ ⊗
 $SS \Rightarrow E S \Rightarrow V S$

$I_2:$
 $S \rightarrow V \cdot = E$
 $E \rightarrow V \cdot$

$I_6:$
 $S \rightarrow V = \cdot E$
 $E \rightarrow \cdot V$
 $V \rightarrow \cdot * E$
 $V \rightarrow \cdot \text{id}$

项目 $S \rightarrow V \cdot = E$ 使得

$\text{action}[2, =] = s6$

项目 $E \rightarrow V \cdot$ 使得

$\text{action}[2, =]$ 为按 $E \rightarrow V$ 归约,

因为 $\text{Follow}(E) = \{=, \$\}$

产生移进-归约冲突

LR(1) 解析
无移进-归约冲突



非LR的上下文无关结构

若自左向右扫描的移进-归约解析器能及时识别出现在栈顶的句柄，那么相应的文法就是LR（指规范的LR）的。

语言 $L = \{ww^R \mid w \in (a \mid b)^*\}$ 的文法

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

不是LR的

*ababb**b**aba*

语言 $L = \{w\mathbf{c}w^R \mid w \in (a \mid b)^*\}$ 的文法

$$S \rightarrow aSa \mid bSb \mid c$$

是LR的

*ababb**c**bbaba*



非LR的上下文无关结构

若自左向右扫描的移进-归约解析器能及时识别出现在栈顶的句柄，那么相应的文法就是LR（指规范的LR）的。

语言 $L = \{ww^R \mid w \in (a \mid b)^*\}$ 的文法

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

不是LR的

$ababb**b**aba$

$I_0:$
 $S' \rightarrow \cdot S$
 $S \rightarrow aSa$
 $S \rightarrow \cdot$
 $S \rightarrow \cdot bSb$

→ 面临 a ，移进

→ $\because a \in Follow(S)$ ， \therefore 面临 a ，归约

存在移进-归约冲突
故不是SLR(1)文法



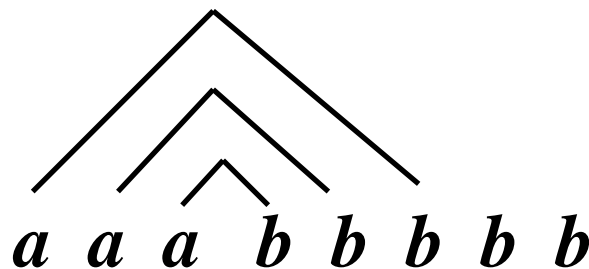
例题 写不同的文法

为语言 $L = \{ a^m b^n \mid n > m \geq 0 \}$ 写三个文法, 它们分别是LR(1)的、二义的和非二义且非LR(1)的。

□ LR(1)文法: $S \rightarrow \textcolor{red}{A}\textcolor{blue}{B}$ $A \rightarrow aAb \mid \varepsilon$ $B \rightarrow Bb \mid b$

□ $\textcolor{red}{A}$ 是每个 a 有个唯一的匹配的 b , 形成中心对称

□ $\textcolor{blue}{B}$ 是剩余 $(n-m)$ 个 b





例题 写不同的文法

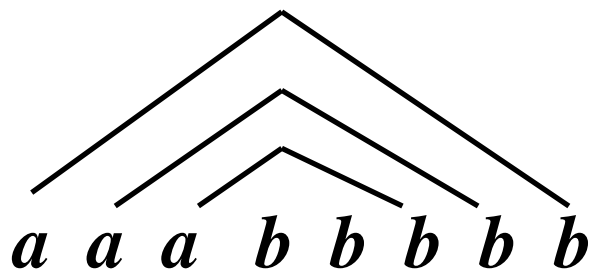
为语言 $L = \{ a^m b^n \mid n > m \geq 0 \}$ 写三个文法, 它们分别是LR(1)的、二义的和非二义且非LR(1)的。

□ LR(1)文法: $S \rightarrow \textcolor{red}{A}\textcolor{blue}{B} \quad A \rightarrow aAb \mid \varepsilon \quad B \rightarrow Bb \mid b$

□ 非二义且非LR(1)的文法: $S \rightarrow a\textcolor{red}{S}b \mid \textcolor{blue}{B} \quad B \rightarrow Bb \mid b$

□ 多出来的 $(n-m)$ 个 b 位于中间, 每个 a 有唯一配对的 b

□ 移进-归约冲突: 面临第2个 b 起



I_0 :

- $S' \rightarrow \cdot S, \$$
- $S \rightarrow \textcolor{red}{a}Sb, \$$
- $S \rightarrow \cdot \textcolor{blue}{B}, \$$
- $B \rightarrow \cdot \textcolor{red}{B}b, b/\$$
- $B \rightarrow \cdot \textcolor{red}{b}, b/\$$

I_1 :

- $S \rightarrow \textcolor{red}{a} \cdot Sb, \$$
- $S \rightarrow \textcolor{red}{a}S \cdot b, b$
- $S \rightarrow \cdot \textcolor{blue}{B}, b$
- $B \rightarrow \cdot \textcolor{red}{B}b, b$
- $B \rightarrow \cdot \textcolor{red}{b}, b$

I_2 :

- $S \rightarrow \textcolor{blue}{B} \cdot, b$
- $B \rightarrow B \cdot \textcolor{red}{b}, b$



例题 写不同的文法

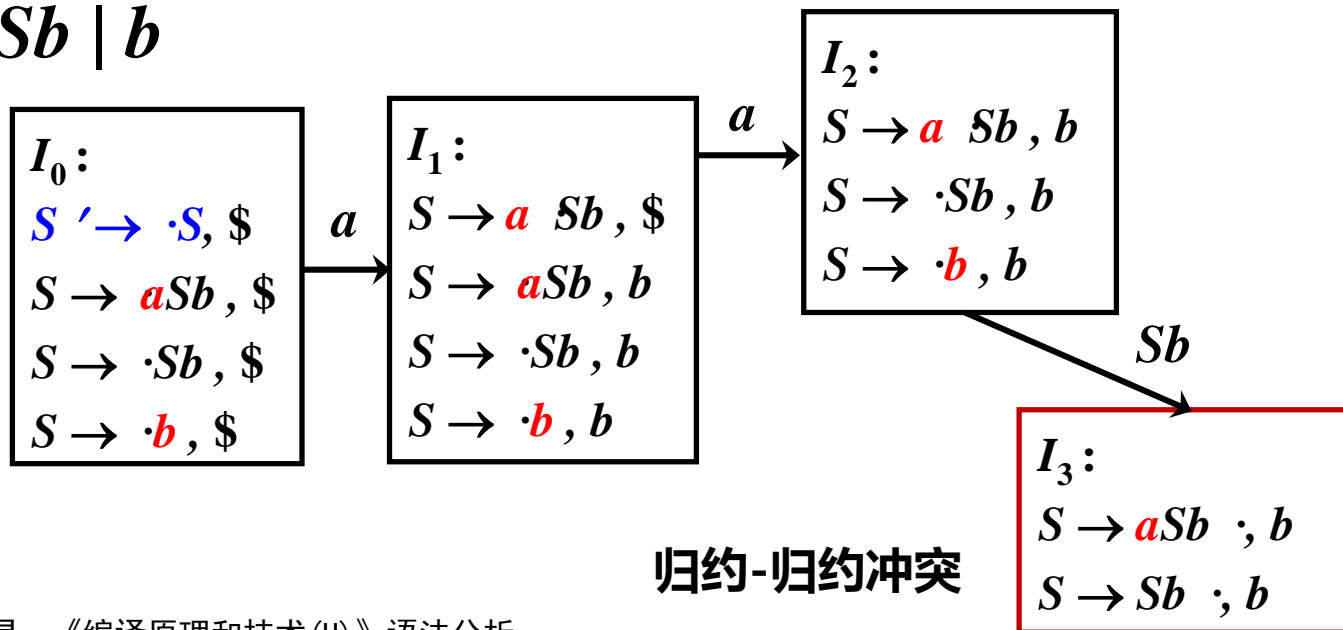
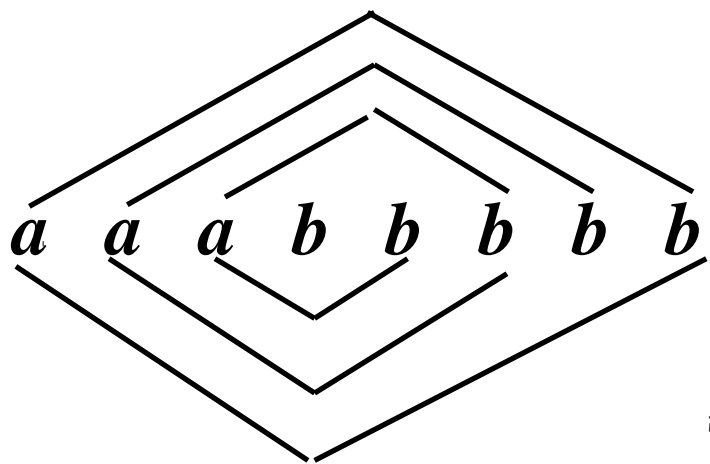
为语言 $L = \{ a^m b^n \mid n > m \geq 0 \}$ 写三个文法, 它们分别是LR(1)的、二义的和非二义且非LR(1)的。

□ LR(1)文法: $S \rightarrow \textcolor{red}{A}\textcolor{blue}{B} \quad A \rightarrow aAb \mid \varepsilon \quad B \rightarrow Bb \mid b$

□ 非二义且非LR(1)的文法: $S \rightarrow a\textcolor{red}{S}b \mid \textcolor{blue}{B} \quad B \rightarrow Bb \mid b$

□ 二义的文法: $S \rightarrow a\textcolor{blue}{S}b \mid Sb \mid b$

□ 每个 a 可有多多个配对的 b



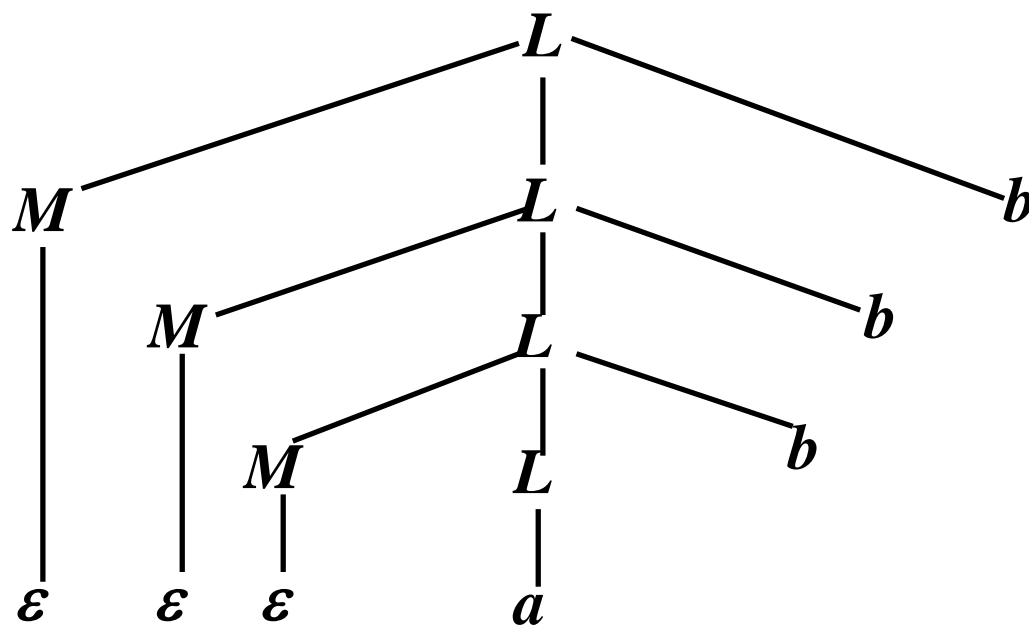


例题4

试说明下面文法不是LR(1)的：

$$L \rightarrow M L b \mid a$$

$$M \rightarrow \varepsilon$$



句子 $abbb$ 的分析树

面临 a 时，不知道该
做多少次空归约 $M \rightarrow \varepsilon$



例题5

下面的文法不是LR(1)的，对它略做修改，使之成为一个等价的SLR(1)文法

program \rightarrow *begin declist ; statement end*

declist \rightarrow *d ; declist* | *d*

statement \rightarrow *s ; statement* | *s*

该文法产生的句子的形式是

begin d ; d ; ... ; d ; s ; s ; ... ; s end

修改后的文法如下：

program \rightarrow *begin declist statement end*

declist \rightarrow *d ; declist* | *d ;*

statement \rightarrow *s ; statement* | *s*



例题6

一个C语言的文件如下，第四行的if误写成fi:

```
long gcd(p,q)
long p,q;
{
    fi (p%q == 0)
        return q;
    else
        return gcd(q, p%q);
}
```

基于LALR (1) 方法的一个编译器的报错情况如下:

parse error before 'return' (line 5).

是否违反了LR解析的活前缀性质?



LR项目与LR文法小结

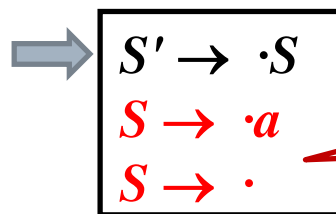
- LR(**0**)项目 $[A \rightarrow \alpha \cdot \beta]$ 、LR(**1**)项目 $[A \rightarrow \alpha \cdot \beta \text{ } a]$
 - 数字表示向前搜索的符号个数，**0**表示不向前搜索符号
- SLR(**k**)分析与SLR(**k**)文法
 - SLR(1)分析的状态：LR(0)项目集
 - **k**是指向前看输入缓冲区的k个符号
- [规范的]LR(k)分析与LR(k)文法
 - LR(1)分析的状态：LR(1)项目集
- LALR(k)分析与LALR(k)文法
 - LR(1)分析的状态：LR(1)项目集+同心项目集合并



LR项目与LR文法小结

□ 不是SLR(0)文法，但是SLR(1)文法

■ 例： $S \rightarrow a \mid \varepsilon$



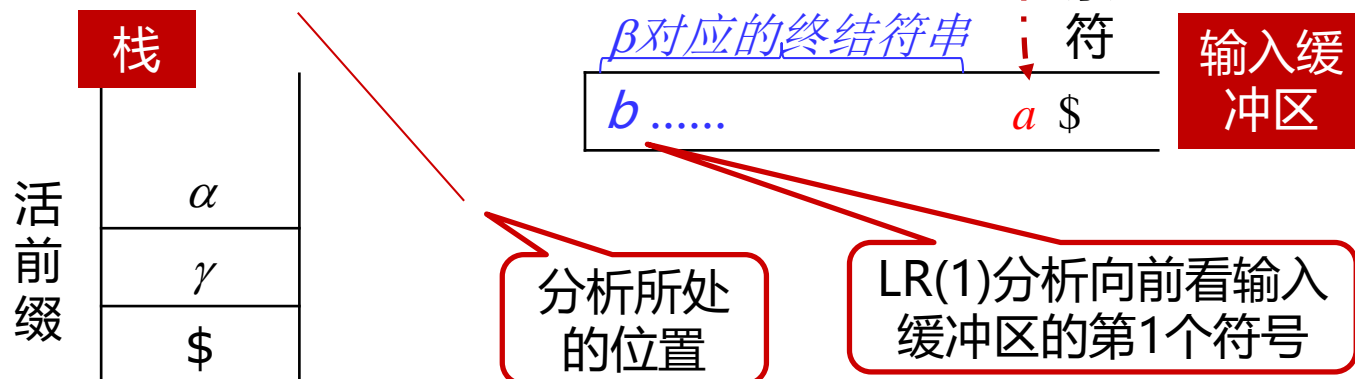
不向前看时，
移进-归约
冲突

□ SLR(0)文法

■ $S \rightarrow a$ $S \rightarrow a \mid b$

□ 理解LR(1)项目与LR(1)文法中的1

■ 若栈顶状态包含LR(1)项目 $[A \rightarrow \alpha \cdot \beta, a^-]$





中国科学技术大学
University of Science and Technology of China

下期预告：二义文法的应用

至此，本课程最抽象且
难以理解的部分已学完