



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

语法分析 IV

《编译原理和技术》

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



3.4 自下而上分析

(移进-归约分析)

- 归约(最右推导的逆过程)
- 句柄(可归约串),可能不唯一
- 冲突: 移进-归约、归约-归约



语法分析的主要方法

□ 自上而下 (top-down) 预测分析

- 从开始符号出发，为输入串寻找最左推导
是自上而下形成分析树的过程
- 即便消除左递归、提取左公因子，仍然存在一些程序语言，
它们对应的文法不是LL(1)

□ 自下而上 (bottom-up) 移进-归约

- 针对输入串，尝试根据产生式归约(reduce, 将与产生式右部匹配的串归约为左部符号)，直至归约到开始符号
是自下而上形成分析树的过程
- 比top-down方法更一般化



3.4 自下而上分析

(移进-归约分析)

- 归约(最右推导的逆过程)
- 句柄(可归约串),可能不唯一
- 冲突: 移进-归约、归约-归约



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例

$$S \rightarrow aABe$$

$$A \rightarrow Abc / b$$

$$B \rightarrow d$$



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

输入串: $abbcd e$

ab (读入 ab) 寻找能匹配某产生式右部的子串

a b



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

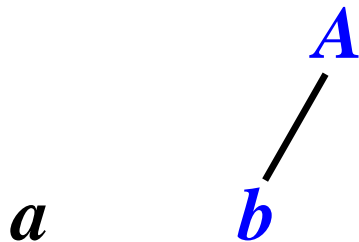
例 $S \rightarrow aABe$
 $A \rightarrow Abc / b$
 $B \rightarrow d$

输入串: $abbcd$

ab

aA (归约)

用产生式 $A \rightarrow b$
归约后仍能形成右句型
 $aAbcde$ 是右句型



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$

右句型: 按最右推导推出的句型, $aABe$ 、 $aAde$ 、 $aAbcde$ 、 $abbcde$ 都是右句型



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$

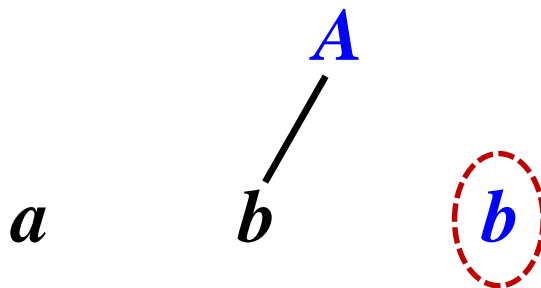
$A \rightarrow Abc / b$

$B \rightarrow d$

输入串: $abbcd e$

ab

aAb (再读入 b)



$A \rightarrow b$
 b 可以归约成 A 吗?



归约 (Reduce)

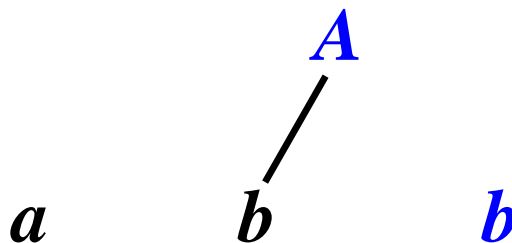
把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$
 $A \rightarrow Abc / b$
 $B \rightarrow d$

输入串: $abbcde$

ab

aAb (再读入 b)



$A \rightarrow b$
 b 可以归约成 A 吗?

归约后能否形成右句型?
 $aAAbcde$ 不是右句型
故不能将 b 归约成 A

$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$



归约 (Reduce)

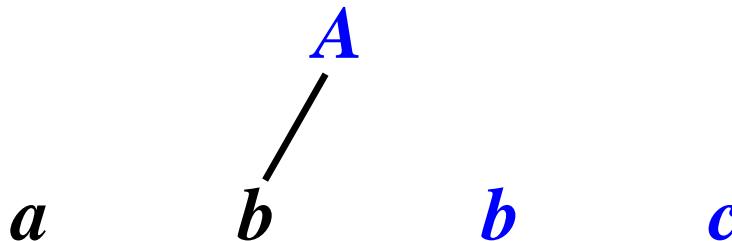
把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$
 $A \rightarrow \textcolor{blue}{Abc} / b$
 $B \rightarrow d$

输入串: $abbcde$

$\textcolor{blue}{ab}$

$\textcolor{blue}{aAbc}$ (再读入 c)





归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$

$A \rightarrow \textcolor{blue}{Abc} / b$

$B \rightarrow d$

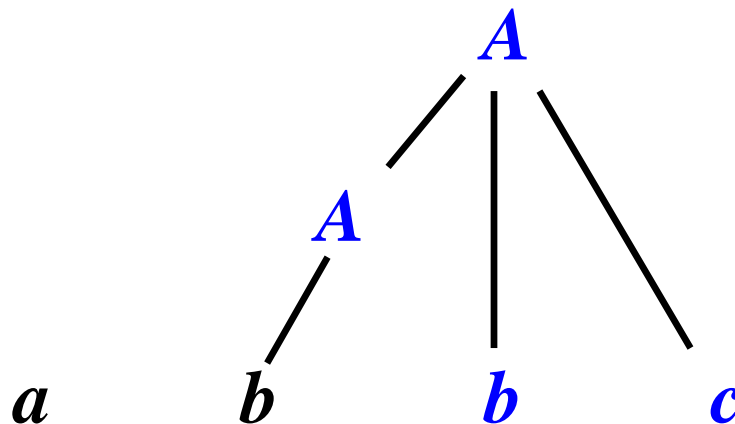
输入串: $abbcd\textcolor{blue}{e}$

$\textcolor{blue}{ab}$

$a\textcolor{blue}{Abc}$

$a\textcolor{blue}{A}$ (归约)

$A \rightarrow \textcolor{blue}{Abc}$



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} a\textcolor{red}{A}de \Rightarrow_{rm} a\textcolor{red}{A}\textcolor{blue}{bc}de \Rightarrow_{rm} abbcd\textcolor{blue}{e}$



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

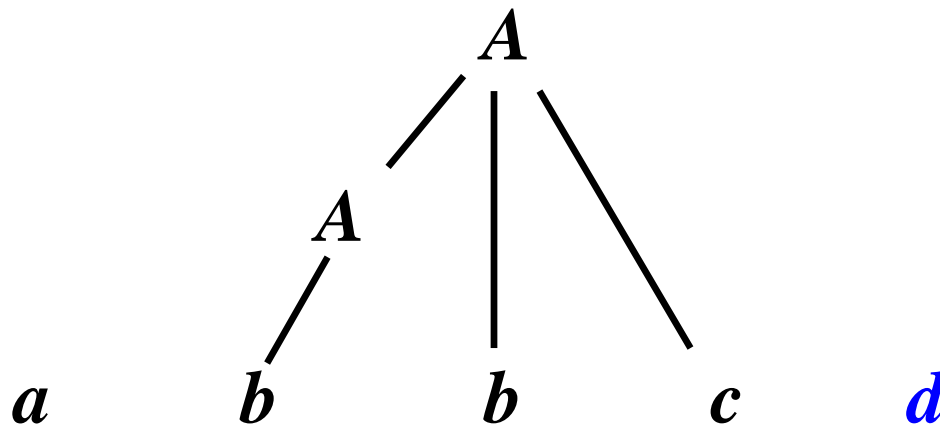
例 $S \rightarrow aABe$
 $A \rightarrow Abc / b$
 $B \rightarrow d$

输入串: $abbcd$

ab

$aAbc$

aAd (再读入 d)



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

$B \rightarrow d$

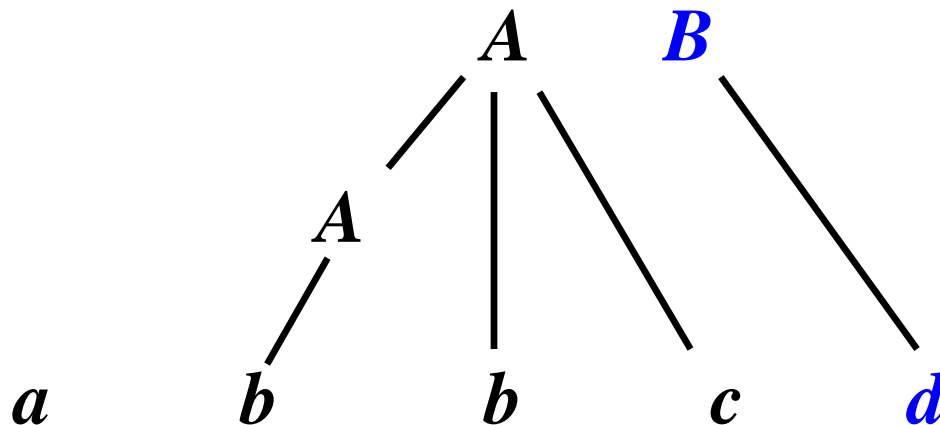
输入串: $abbcd e$

ab

$aAbc$

aAd

aAB (归约)



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} a\underline{Abc}de \Rightarrow_{rm} abbcde$



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

输入串: $abbcd e$

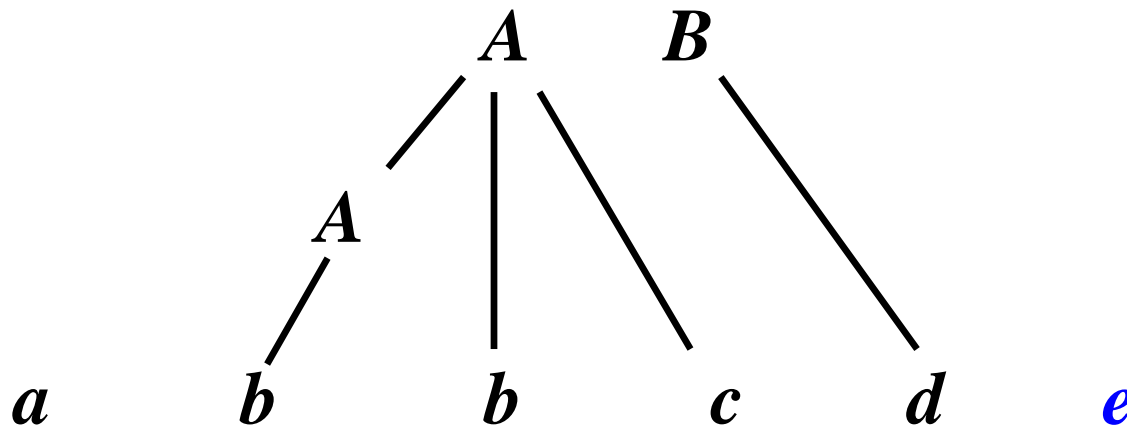
ab

$aAbc$

aAd

aAB

$aABe$ (再读入 e)



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} a\underline{Abc}de \Rightarrow_{rm} abbcde$



归约 (Reduce)

把输入串归约成文法的开始符号，是最右推导的逆过程

例 $S \rightarrow aABe$

$A \rightarrow Abc / b$

$B \rightarrow d$

输入串: $abbcd e$

ab

$aAbc$

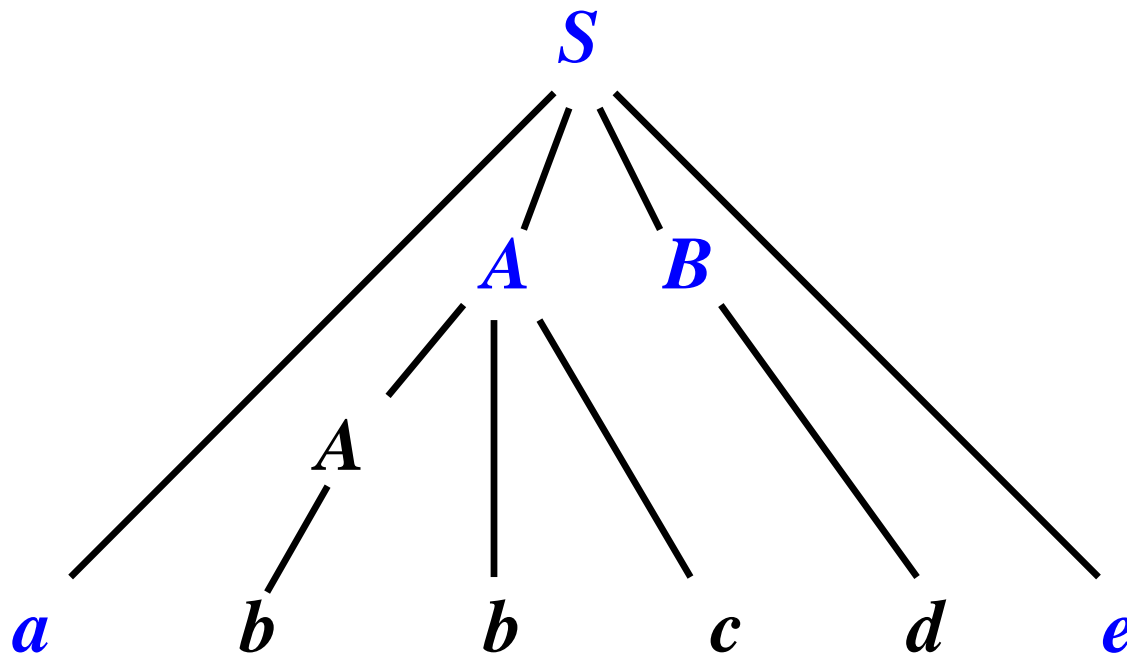
aAd

aAB

$aABe$

S (归约)

$S \rightarrow aABe$



$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$



3.4 自下而上分析

(移进-归约分析)

- 归约(最右推导的逆过程)
- 句柄(可归约串),可能不唯一
- 冲突: 移进-归约、归约-归约



句柄(handles)

□ 右句型的句柄（可归约串）

■ 是右句型 $\alpha \delta \beta$ 中和某产生式 $B \rightarrow \delta$ 右部匹配的子串 δ , 并且

■ 把 δ 归约成 B 代表了最右推导的逆过程的一步

右句型 $\alpha \delta \beta$ 中将子串 δ 归约成 B 后得到的串 $\alpha B \beta$ 仍是右句型

$S \rightarrow aABe$

* 右句型：最右推导可得的句型

$A \rightarrow Abc / b$

$B \rightarrow d$

$S \Rightarrow_{rm} aABe \Rightarrow_{rm} aAde \Rightarrow_{rm} aAbcde \Rightarrow_{rm} abbcde$

■ 句柄的右边仅含终结符（是尚未处理输入串）

■ 如果文法二义，那么句柄可能不唯一



例 句柄不唯一

$$E \rightarrow E + E / E * E / (E) / \text{id}$$

$$E \Rightarrow_{rm} E * E$$

$$\Rightarrow_{rm} E * E + E$$

$$\Rightarrow_{rm} E * E + \text{id}_3$$

$$\Rightarrow_{rm} E * \text{id}_2 + \text{id}_3$$

$$\Rightarrow_{rm} \text{id}_1 * \text{id}_2 + \text{id}_3$$



例 句柄不唯一

$$E \rightarrow E + E / E * E / (E) / \text{id}$$

$$E \Rightarrow_{rm} E * E$$

$$\Rightarrow_{rm} E * E + E$$

$$\Rightarrow_{rm} E * E + \text{id}_3$$

$$\Rightarrow_{rm} E * \text{id}_2 + \text{id}_3$$

$$\Rightarrow_{rm} \text{id}_1 * \text{id}_2 + \text{id}_3$$

$$E \Rightarrow_{rm} E + E$$

$$\Rightarrow_{rm} E + \text{id}_3$$

$$\Rightarrow_{rm} E * E + \text{id}_3$$

$$\Rightarrow_{rm} E * \text{id}_2 + \text{id}_3$$

$$\Rightarrow_{rm} \text{id}_1 * \text{id}_2 + \text{id}_3$$

在右句型 $E * E + \text{id}_3$ 中，句柄不唯一： id_3 、 $E * E$

* 右句型：最右推导可得的句型



用栈实现移进-归约分析

先通过“移进-归约分析器在分析输入串 $id_1 * id_2 + id_3$ 时的动作序列“来了解移进-归约分析的工作方式。

需要引入**栈**保存移进的文法符号

归约时需要从栈的顶部将形成句柄的文法符号串弹出，
再将归约成的非终结符入栈



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3$ \$	移进



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
\$ <i>E</i>	$* id_2 + id_3 \$$	



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
\$ E	$* id_2 + id_3 \$$	移进



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
$\$E$	$* id_2 + id_3 \$$	移进
$\$E*$	$id_2 + id_3 \$$	



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
$\$E$	$* id_2 + id_3 \$$	移进
$\$E*$	$id_2 + id_3 \$$	移进



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
$\$E$	$* id_2 + id_3 \$$	移进
$\$E*$	$id_2 + id_3 \$$	移进
$\$E*id_2$	$+ id_3 \$$	



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
$\$E$	$* id_2 + id_3 \$$	移进
$\$E*$	$id_2 + id_3 \$$	移进
$\$E*id_2$	$+ id_3 \$$	按 $E \rightarrow id$ 归约



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
$\$E$	$* id_2 + id_3 \$$	移进
$\$E*$	$id_2 + id_3 \$$	移进
$\$E*id_2$	$+ id_3 \$$	按 $E \rightarrow id$ 归约
$\$E*E$	$+ id_3 \$$	



移进-归约分析: $id_1 * id_2 + id_3$

栈	输 入	动 作
\$	$id_1 * id_2 + id_3 \$$	移进
\$ id_1	$* id_2 + id_3 \$$	按 $E \rightarrow id$ 归约
\$ E	$* id_2 + id_3 \$$	移进
\$ $E*$	$id_2 + id_3 \$$	移进
\$ $E*id_2$	$+ id_3 \$$	按 $E \rightarrow id$ 归约
\$ $E*E$	$+ id_3 \$$	移进? 归约?
<div> <div> $\begin{aligned} E &\Rightarrow_{rm} E * E \\ &\Rightarrow_{rm} E * E + E \\ &\Rightarrow_{rm} E * E + id_3 \end{aligned}$ </div> <div> $\begin{aligned} E &\Rightarrow_{rm} E + E \\ &\Rightarrow_{rm} E + id_3 \\ &\Rightarrow_{rm} E * E + id_3 \end{aligned}$ </div> <div> <div>移进</div> <div>归约</div> </div> </div>		



3.4 自下而上分析

(移进-归约分析)

- 归约(最右推导的逆过程)
- 句柄(可归约串),可能不唯一
- 冲突: 移进-归约、归约-归约



移进-归约分析需要解决的一些问题

- 如何决策是选择移进还是归约?
- 进行归约时, 怎么确定右句型中要归约的子串(即句柄)
 - 句柄总是出现在栈顶
- 进行归约时, 如何确定选择哪一个产生式?



移进-归约分析的冲突

□ 移进-归约冲突(shift/reduce conflict)

例

stmt → **if** *expr* **then** *stmt*

| **if** *expr* **then** *stmt* **else** *stmt*

| **other**

如果移进-归约分析器处于格局(configuration)

栈

输入

... **if** *expr* **then** *stmt*

else ... \$

一般地，优先移进
也满足else的
就近匹配原则



移进-归约分析的冲突

□ 归约-归约冲突(reduce/reduce conflict)

$stmt \rightarrow id (parameter_list) \mid expr = expr$ id(...)是函数调用

$parameter_list \rightarrow parameter_list, parameter \mid parameter$

$parameter \rightarrow id$

$expr \rightarrow id (expr_list) \mid id$ id(...)也表示数组元素的引用

$expr_list \rightarrow expr_list, expr \mid expr$

由A(I, J)开始的语句

归约成 $expr$ 还是 $parameter$?

栈

... id (id

输入

, id)...

方法1
一般用位于前面的产生式进行归约



移进-归约分析的冲突

□ 归约-归约冲突(reduce/reduce conflict)

$stmt \rightarrow \text{procid}(parameter_list) \mid expr = expr$

$parameter_list \rightarrow parameter_list, parameter \mid parameter$

$parameter \rightarrow id$

$expr \rightarrow id(expr_list) \mid id$

$id(...)$ 也表示数组元素的引用

$expr_list \rightarrow expr_list, expr \mid expr$

由 $A(I, J)$ 开始的语句(词法分析查符号表, 区分第一个 id)

栈

... **procid**(id

输入

, id)...

方法2
改写文法, 区分
 id 是否是 $procid$

■ 需要修改上面的文法



3.5 LR分析器

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

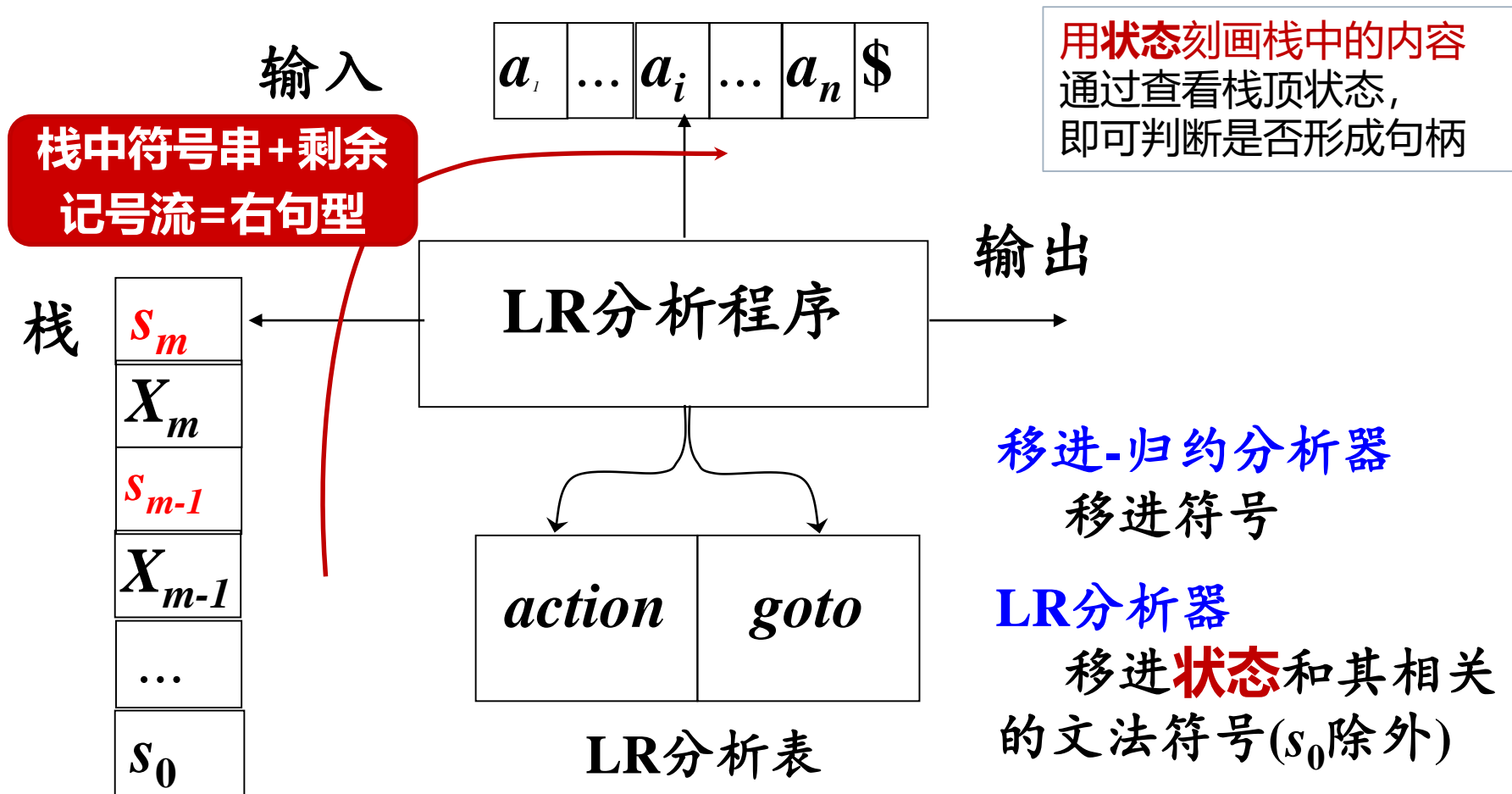
- LR分析算法：效率高
- LR分析表的构造技术

简单的LR(SLR)、规范的LR、向前看LR(LALR)



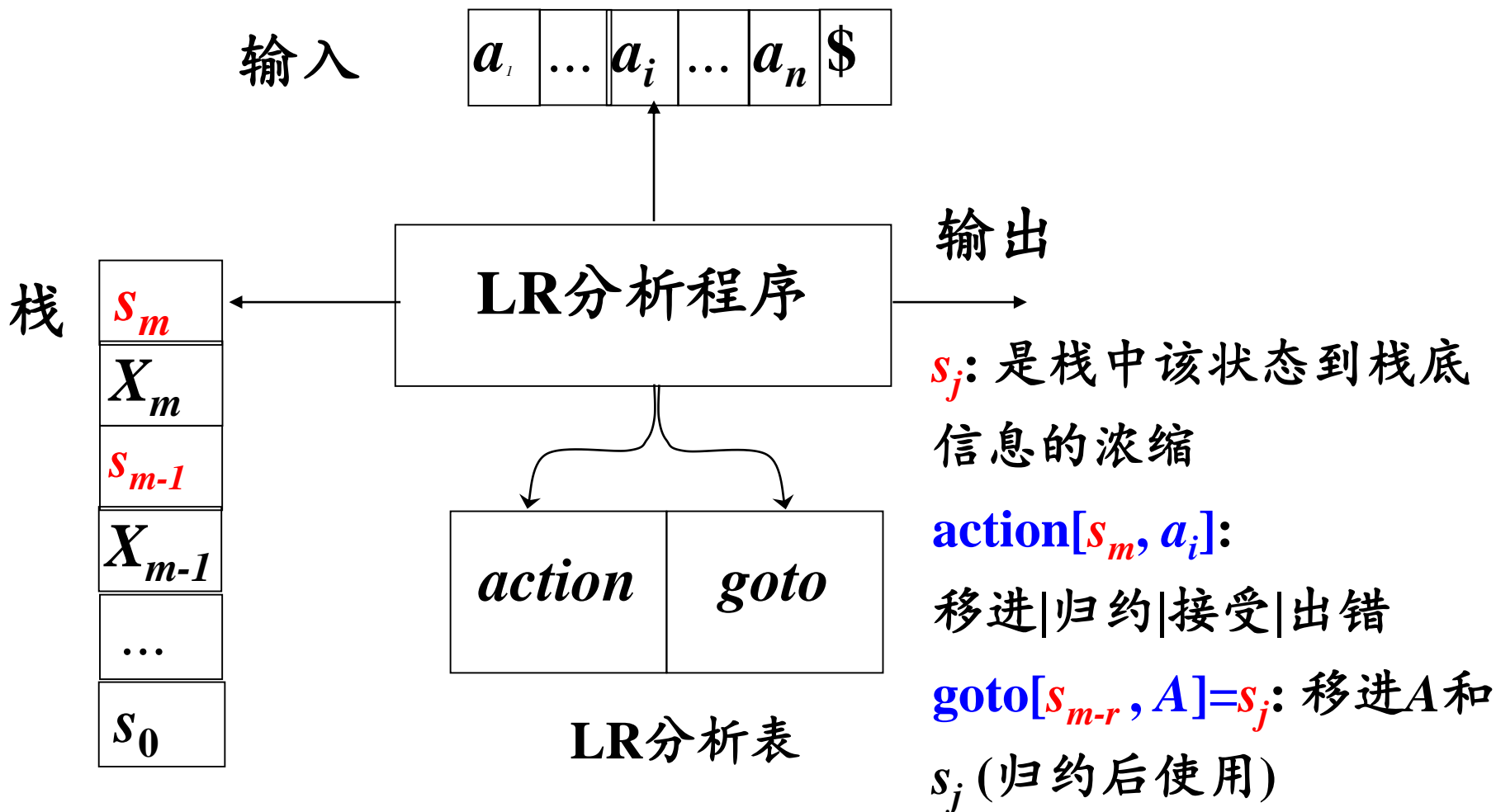
LR分析算法: 分析器的模型

- 如何快速识别栈的顶部是否形成句柄? → 引入 **状态**





LR分析算法: 分析器的模型





LR分析算法：举例

shift reduce

实际可不用移进

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

P69 $T \rightarrow T * F / T \rightarrow E$

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

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

LR分析表

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			



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 ($F \rightarrow a$):
 - 从栈中弹出 $|a|$ 个状态-符号对
 - 查 $\text{goto}[0, F] = 3$
 - 将 $(F, 3)$ 入栈



LR分析算法：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分析算法：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分析算法：举例

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			

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 \$	



LR分析算法：举例

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			

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 \$	



LR分析算法：举例

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			

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
...



LR分析算法：举例

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			

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

归约为 开始符号

完成合法输入串的分析



LR分析: 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

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

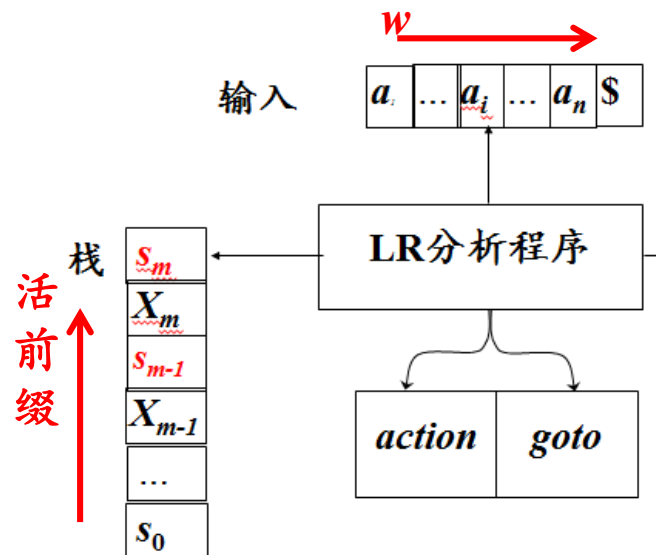
➤ $\gamma\beta$ 的任何前缀 (包括 ε 和 $\gamma\beta$ 本身) 都是活前缀

➤ w 仅包含终结符

■ 对应到LR分析模型上的特点

□ 活前缀: 是LR分析栈中从栈底到栈顶的**文法符号**连接形成的串

□ w : 输入缓冲区中剩余的记号串





LR分析: 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	<i>s5</i> <i>s4</i>	1 2 3
1	<i>s6</i> <i>acc</i>	
2	<i>r2</i> <i>s7</i> <i>r2</i> <i>r2</i>	
3	<i>r4</i> <i>r4</i> <i>r4</i> <i>r4</i>	
4	<i>s5</i> <i>s4</i>	8 2 3



LR分析方法的特点

- 栈中的文法符号总是形成一个**活前缀**
- 分析表的转移函数本质上是**识别活前缀的DFA**
- 栈顶的状态符号包含了确定句柄所需要的一切信息
- 是已知的**最一般的无回溯**的移进-归约方法
- 能分析的文法类是预测分析能分析的文法类的**真超集**
- 能及时发现语法错误
- 手工构造分析表的工作量太大



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		

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

$A \rightarrow l\beta$

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		

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

$A \rightarrow l\beta$

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

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

$A \rightarrow l\beta$

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分析器

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

□ LR分析算法：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分析表的构造

□ LR(0)项目与LR(1)项目

□ SLR：简单的LR

■ 构造LR(0) 项目集规范族→形成DFA状态→SLR分析表

□ LR：规范的LR

■ 构造LR(1) 项目集规范族→形成DFA状态→LR分析表

□ LALR：向前看的LR

■ 构造LR(1) 项目集规范族→形成DFA状态→合并同心项目集→LR分析表

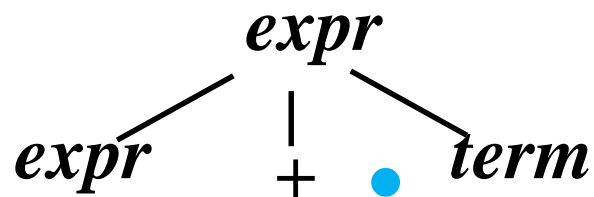


LR分析表的构造方法

■ 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

$expr \rightarrow expr + \cdot term$

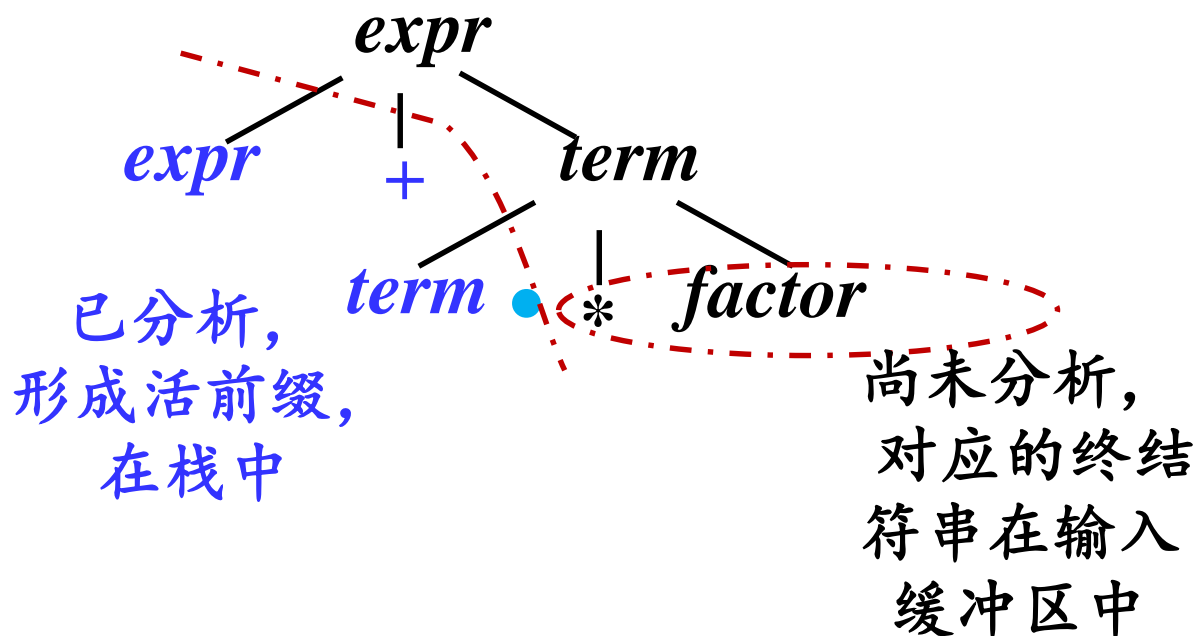
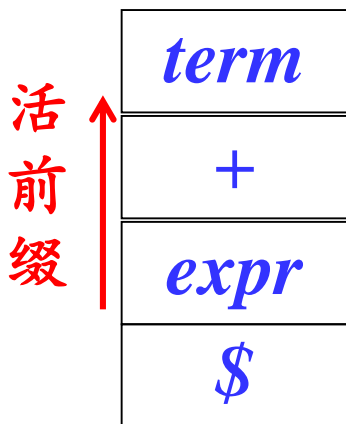




LR分析表的构造方法

■ 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





LR分析表的构造方法

■ 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

例 $A \rightarrow XYZ$ 对应四个项目

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

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

例 $A \rightarrow \varepsilon$ 只有一个项目和它对应

$A \rightarrow \cdot$



LR分析表的构造方法

- LR(0) 项目 $[A \rightarrow \alpha \cdot \beta]$, α 和 β 为文法符号串(可能为 ϵ)
- 状态: LR(0) 项目集规范族(canonical LR(0) collection)

1. 拓广文法 (augmented grammar)

新增产生式和新的开始符号

E 原来的开始符号

$E' \rightarrow E$ 旨在指示分析器何时开始分析、何时完成分析

$E \rightarrow E + T \mid T$

$[E' \rightarrow \cdot E]$ $[E' \rightarrow E \cdot]$

$T \rightarrow T * F \mid F$

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



LR分析表的构造方法

- LR(0) 项目 $[A \rightarrow \alpha \cdot \beta]$, α 和 β 为文法符号串(可能为 ϵ)
- 状态: LR(0) 项目集规范族(canonical LR(0) collection)

初始状态

I_0 :

$E' \rightarrow \cdot E$



LR分析表的构造方法

- LR(0) 项目 $[A \rightarrow \alpha \cdot \beta]$, α 和 β 为文法符号串(可能为 ϵ)
- 状态: LR(0) 项目集规范族(canonical LR(0) collection)

初始状态

I_0 :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

求项目集的闭包closure(I) P75

求LR(0)项目集的闭包closure(I)
P75

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

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



LR分析表的构造方法

- LR(0) 项目 $[A \rightarrow \alpha \cdot \beta]$, α 和 β 为文法符号串(可能为 ϵ)
- 状态: LR(0) 项目集规范族(canonical LR(0) collection)

初始状态

I_0 :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

求项目集的闭包closure(I) P75

求LR(0)项目集的闭包closure(I)
P75

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

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



LR分析表的构造方法

- LR(0) 项目 $[A \rightarrow \alpha \cdot \beta]$, α 和 β 为文法符号串(可能为 ϵ)
- 状态: LR(0) 项目集规范族(canonical LR(0) collection)

初始状态

I_0 :

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

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

求项目集的闭包closure(I) P75

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

可以通过对核心项目求闭包来获得
为节省存储空间, 可省去



LR分析表的构造方法

□ SLR (SimpleLR)

- LR(0) 项目 $[A \rightarrow \alpha \cdot \beta]$, α 和 β 为文法符号串(可能为 ϵ)
- 每个DFA状态: LR(0) 项目集规范族

□ 规范的LR分析

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

问题: LR(1)项目数量庞大 \Rightarrow LR(1)分析的状态数偏多

□ LALR分析 (LookAhead LR)

和SLR同样多的状态, 通过合并规范LR(1)项目集来得到



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

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



构造识别活前缀的DFA

□ LR(0)项目集规范族

$I_0:$

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot bB$

$B \rightarrow \cdot a$

求LR(0)项目集的闭包closure(I)

P75

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

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

□ LR(1)项目集规范族

$I_0:$

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

$S \rightarrow \cdot \mathbf{B}B, \$$ 首终结符

$B \rightarrow \cdot bB, \mathbf{a/b}$

$B \rightarrow \cdot a, \mathbf{a/b}$

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

求LR(1)项目集的闭包closure(I)

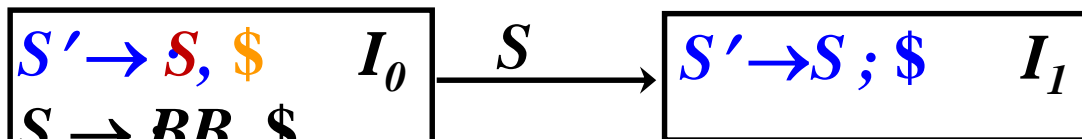
P82

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

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



构造识别活前缀的DFA (以LR(1)项目集为例)



$I_1 := goto (I_0, S)$

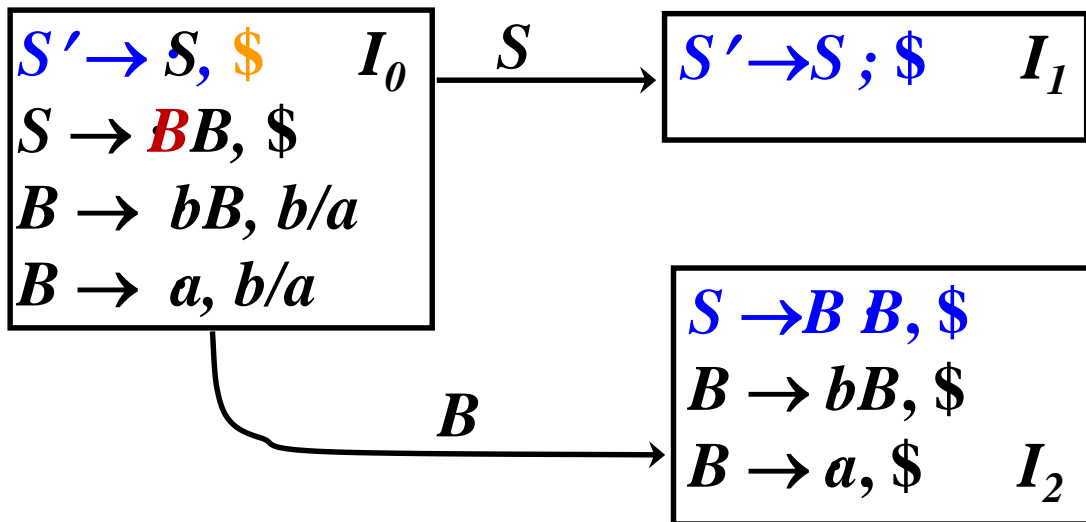
$S' \rightarrow S$

$S \rightarrow BB$

$B \rightarrow bB / a$

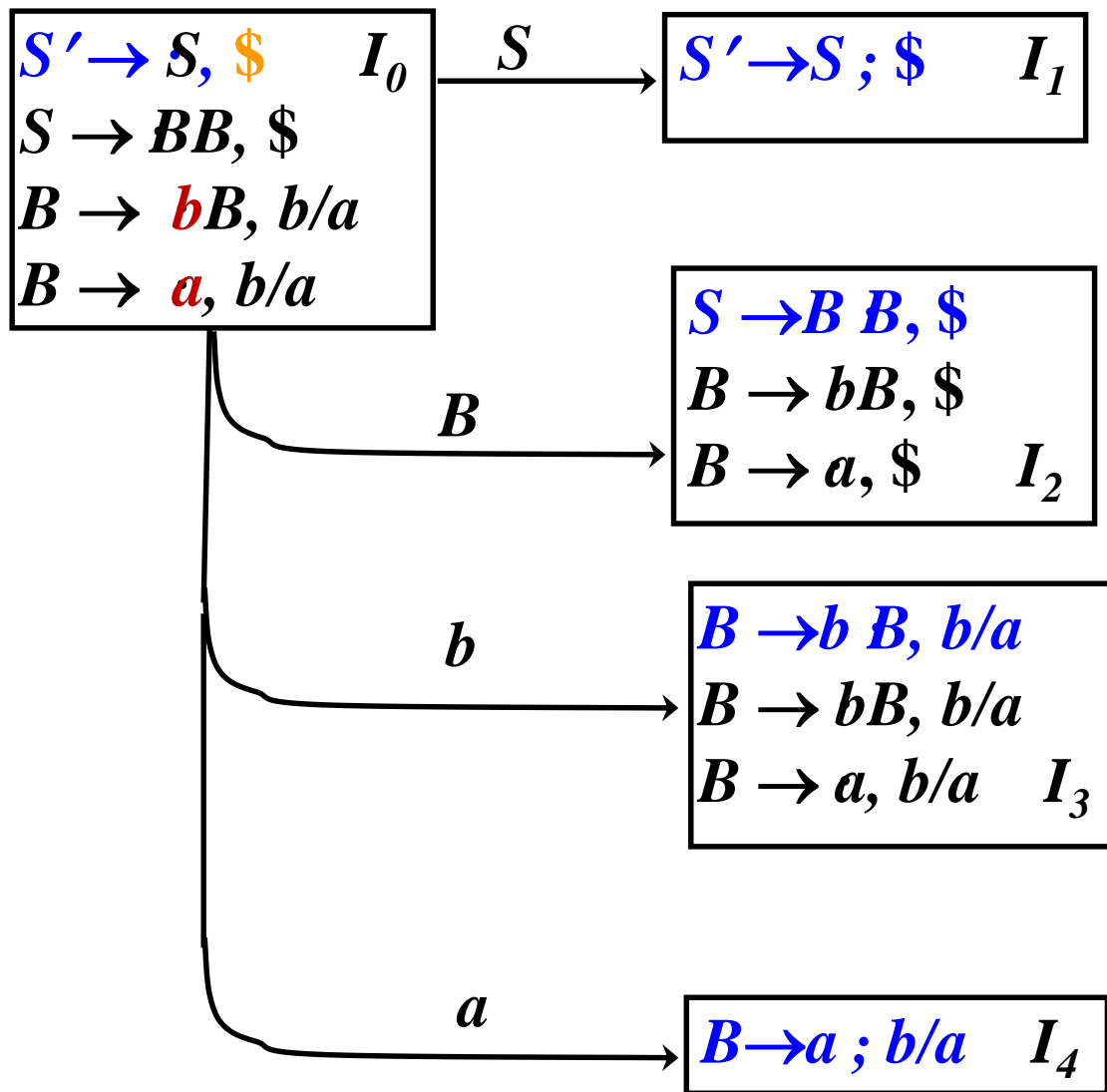


构造识别活前缀的DFA (以LR(1)项目集为例)



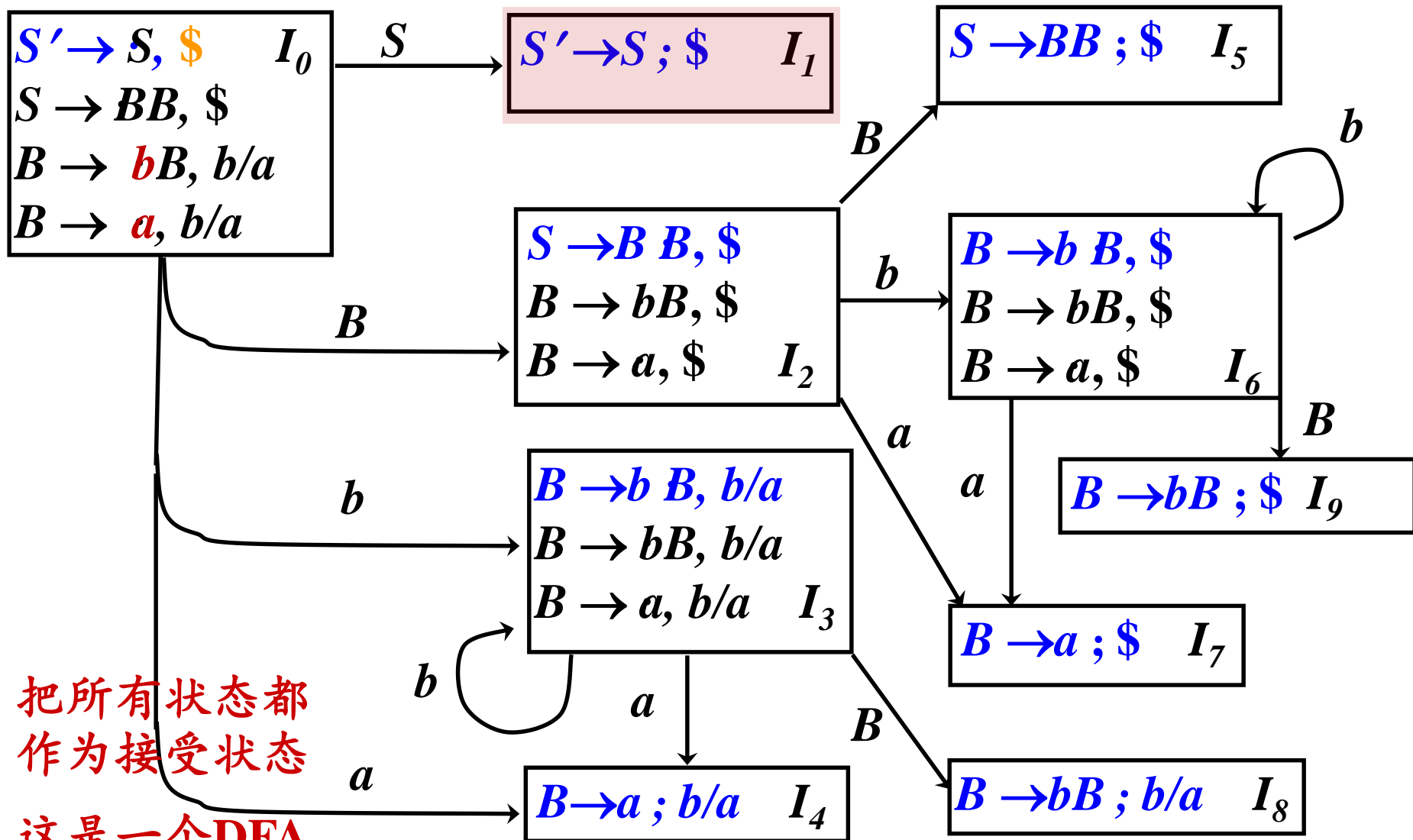


构造识别活前缀的DFA (以LR(1)项目集为例)





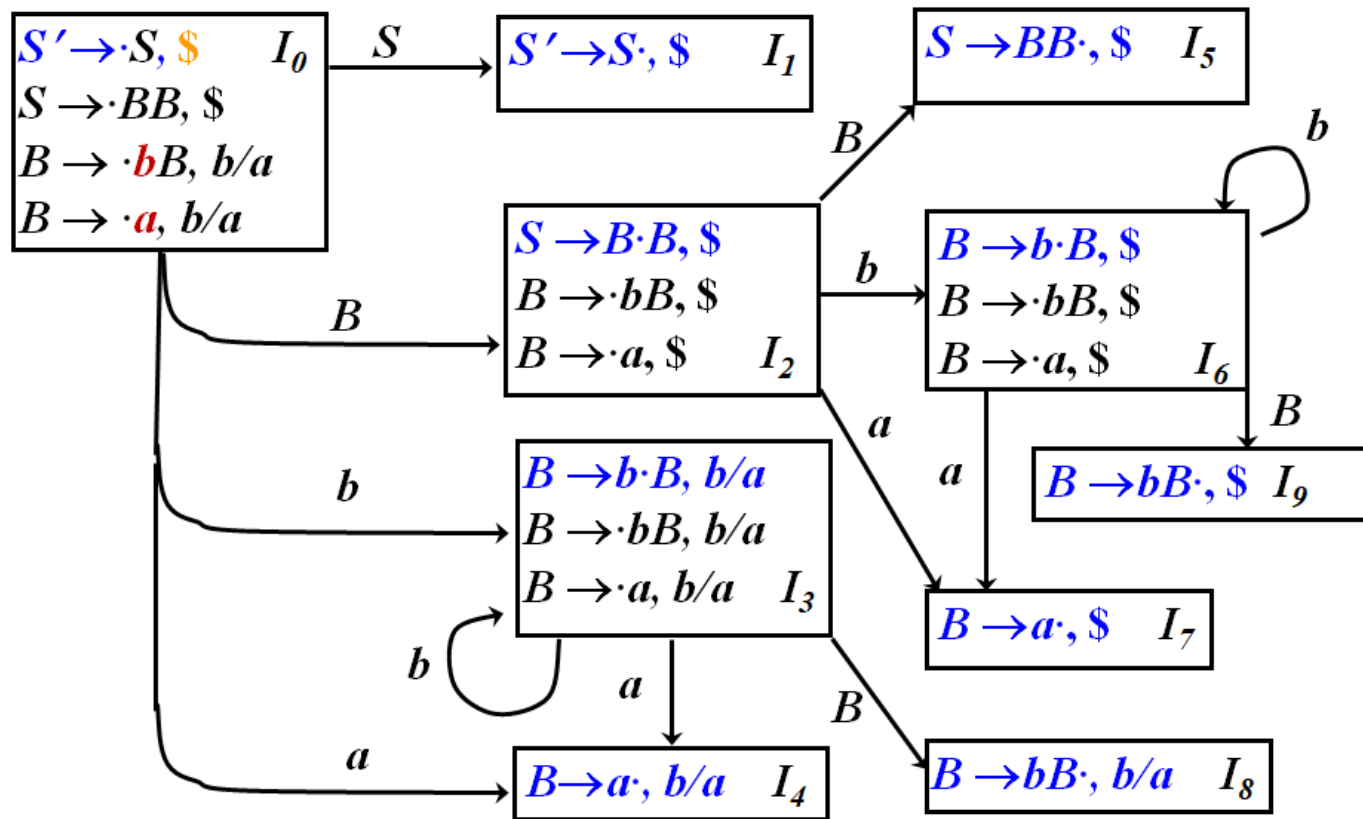
构造识别活前缀的DFA (以LR(1)项目集为例)



把所有状态都
作为接受状态
这是一个DFA



构造识别活前缀的DFA (以LR(1)项目集为例)



$S' \Rightarrow S$
 $\Rightarrow BB$
 $\Rightarrow BbB$
 $\Rightarrow BbbB$

把所有状态都
作为接受状态
这是一个DFA

bB 是最右句柄
 $BbbB$ 的所有前缀(活前缀)都可接受



构造识别活前缀的NFA

以LR(0)项目集为例说明

I_0 :

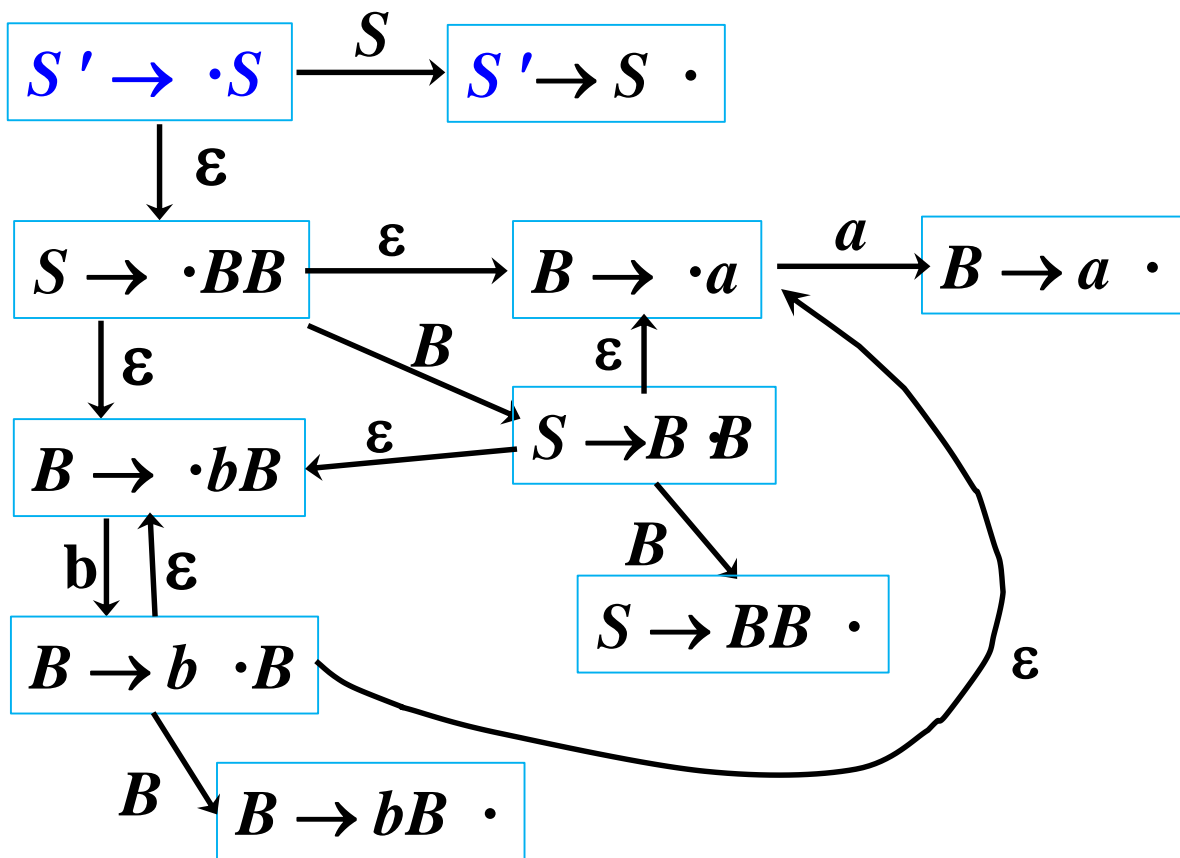
$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot bB$

$B \rightarrow \cdot a$

每个项目一个状态





有效项目

- LR(0)项目 $[A \rightarrow \alpha \cdot \beta]$ 对活前缀 $\gamma = \delta \alpha$ 有效:

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

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

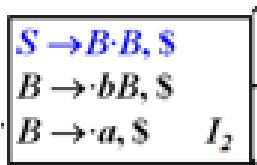
如果存在着推导 $S' \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm} \delta \alpha \beta w$, 其中:

□ a 是 w 的第一个符号, 或者 w 是 ϵ 且 a 是 $\$$

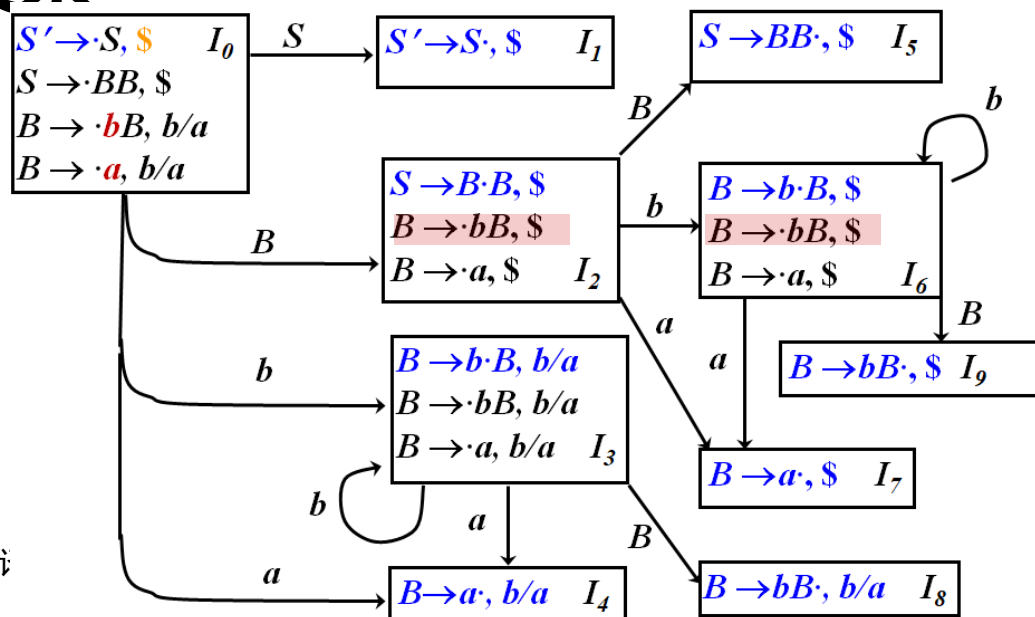
项目与活前缀之间的关系

- $[B \rightarrow \cdot bB, \$]$ 对活前缀 B 、 Bb 、 Bbb 都有效

- 活前缀 B 有多个有效项目



张昱: 《编译

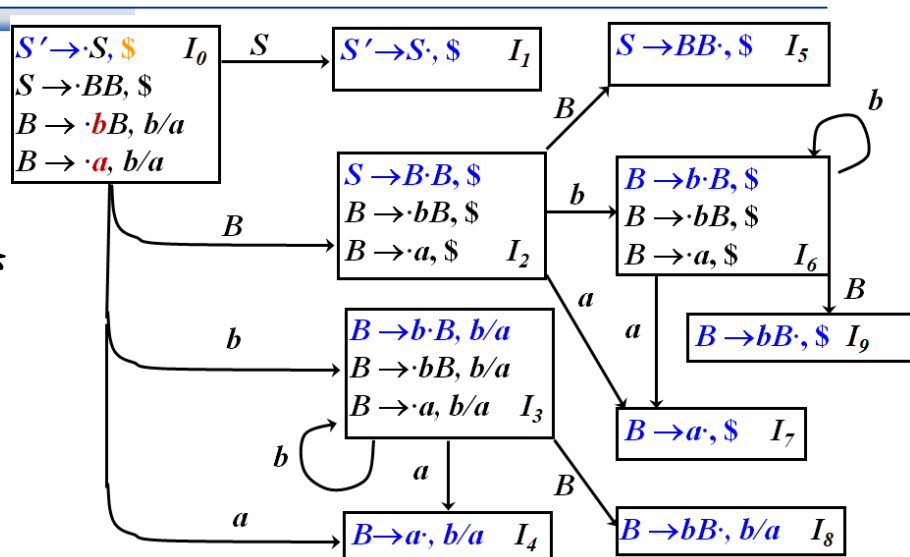




有效项目

□ 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



- 从项目 $A \rightarrow \alpha \cdot \beta$ 对活前缀 $\delta \alpha$ 有效这个事实可以知道

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

- 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)



从DFA构造SLR分析表

□ 状态 i 从 I_i 构造，按如下方法确定 $action$ 函数：

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

如果出现动作冲突，那么该文法就不是SLR(1)的



从DFA构造SLR分析表

- 状态 i 从 I_i 构造, 按如下方法确定 *action* 函数:
 - 移进: 如果 $[A \rightarrow \alpha a \beta]$ 在 I_i 中, 并且 $\text{goto}(I_i, a) = I_j$, 那么置 $\text{action}[i, a]$ 为 *sj*
 - 归约: 如果 $[A \rightarrow \alpha \cdot]$ 在 I_i 中, 那么对 **FOLLOW**(A) 中所有的 a , 置 $\text{action}[i, a]$ 为 *rj*, j 是产生式 $A \rightarrow \alpha$ 的编号
 - 接受: 如果 $[S' \rightarrow S \cdot]$ 在 I_i 中, 那么置 $\text{action}[i, \$]$ 为 *acc*
- 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
- 分析器的初始状态: 包含 $[S' \rightarrow \cdot S]$ 的项目集对应的状态



□ 构造LR分析表，状态 i 的 $action$ 函数按如下确定

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

如果用上面规则构造, 出现了冲突, 则文法就不是LR(1)的

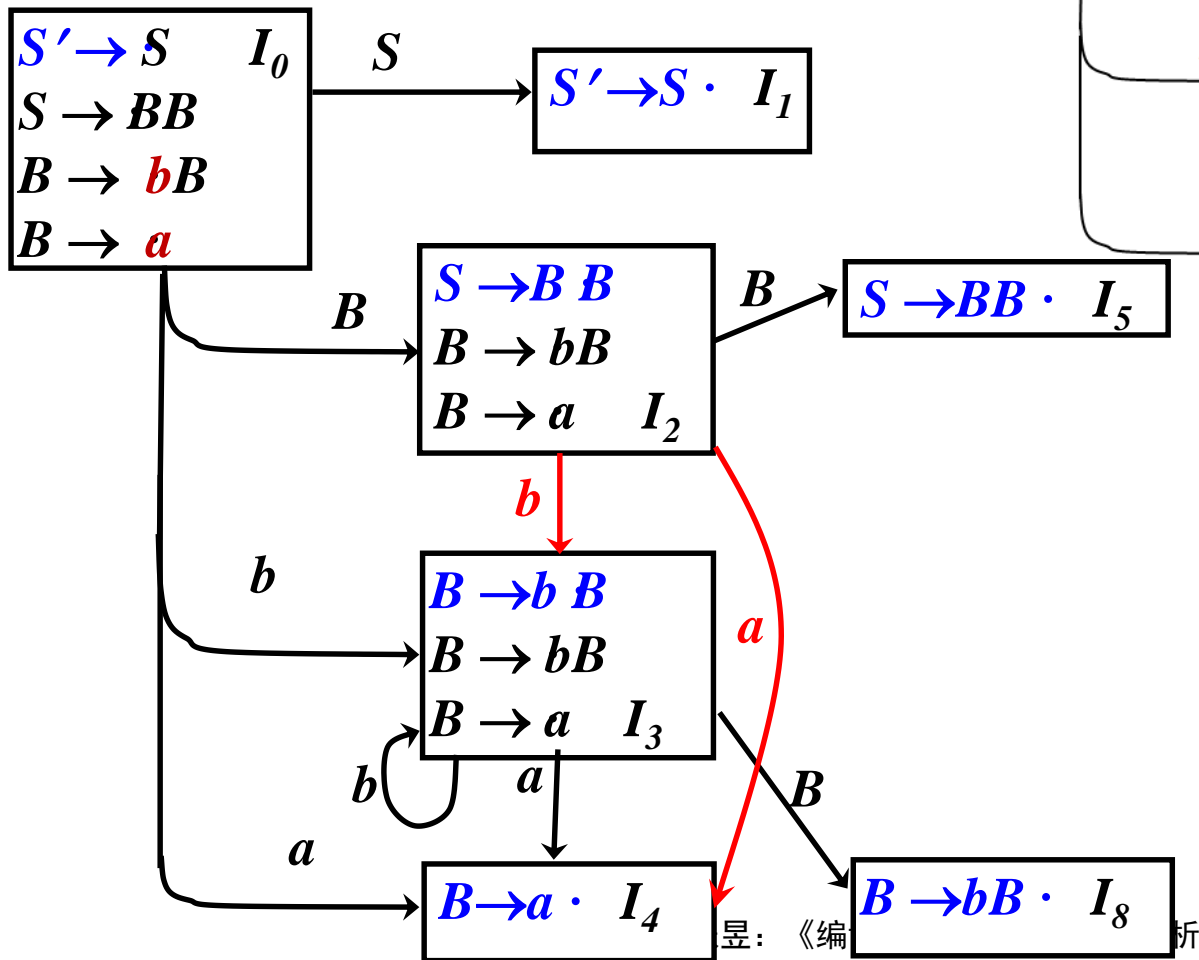
- $goto$ 函数的确定: 如果 $goto(I_i, A) = I_j$, 那么 $goto[i, A] = j$
- 用上面规则未能定义的所有条目都置为 **error**
- 初始状态是包含 $[S' \rightarrow \color{blue}{S}, \$]$ 的项目集对应的状态

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

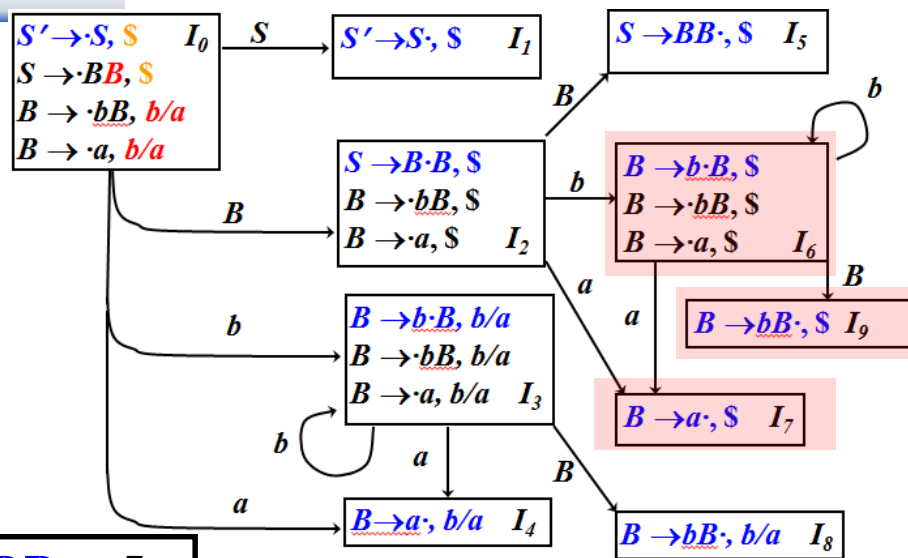


规范的LR vs. SLR 分析

SLR



规范的LR



规范的LR分析
的状态数偏多



LALR分析表 Look ahead LR

□ LALR特点

- 与SLR分析表有同样多状态
- 能力介于SLR和规范LR之间
- 其能力在很多情况下已够用

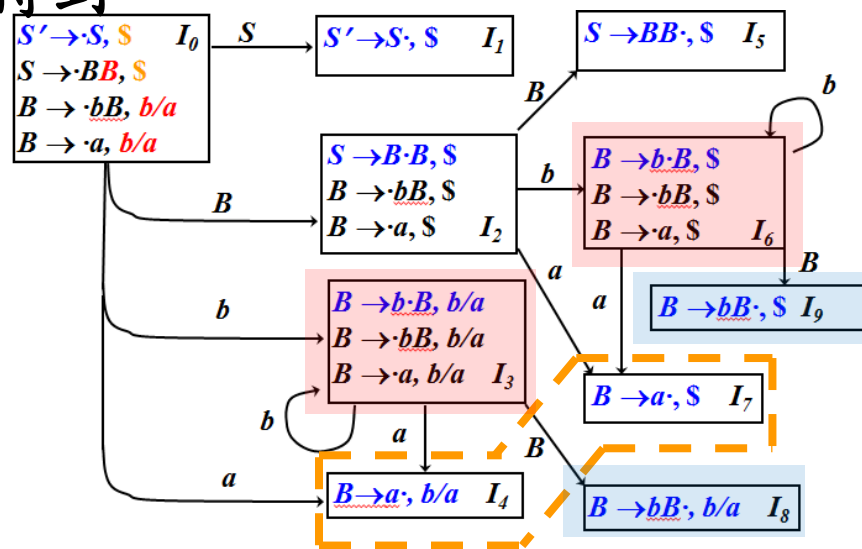
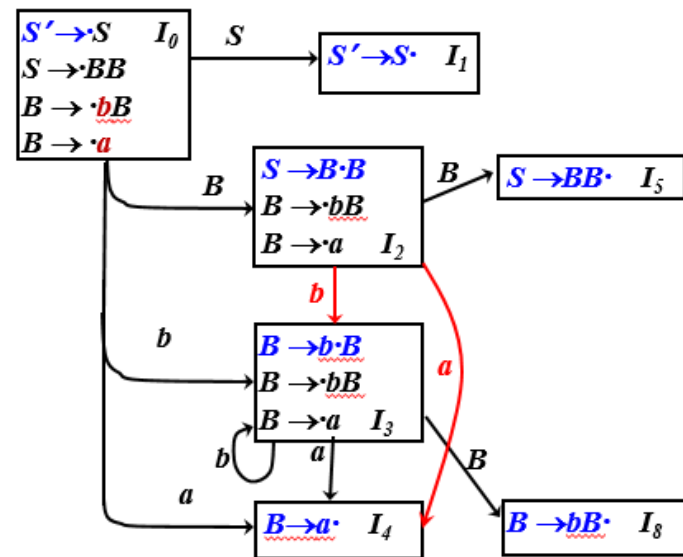
□ LALR分析表构造方法

- 通过合并规范LR(1)项目集来得到

□ 同心的LR(1)项目集

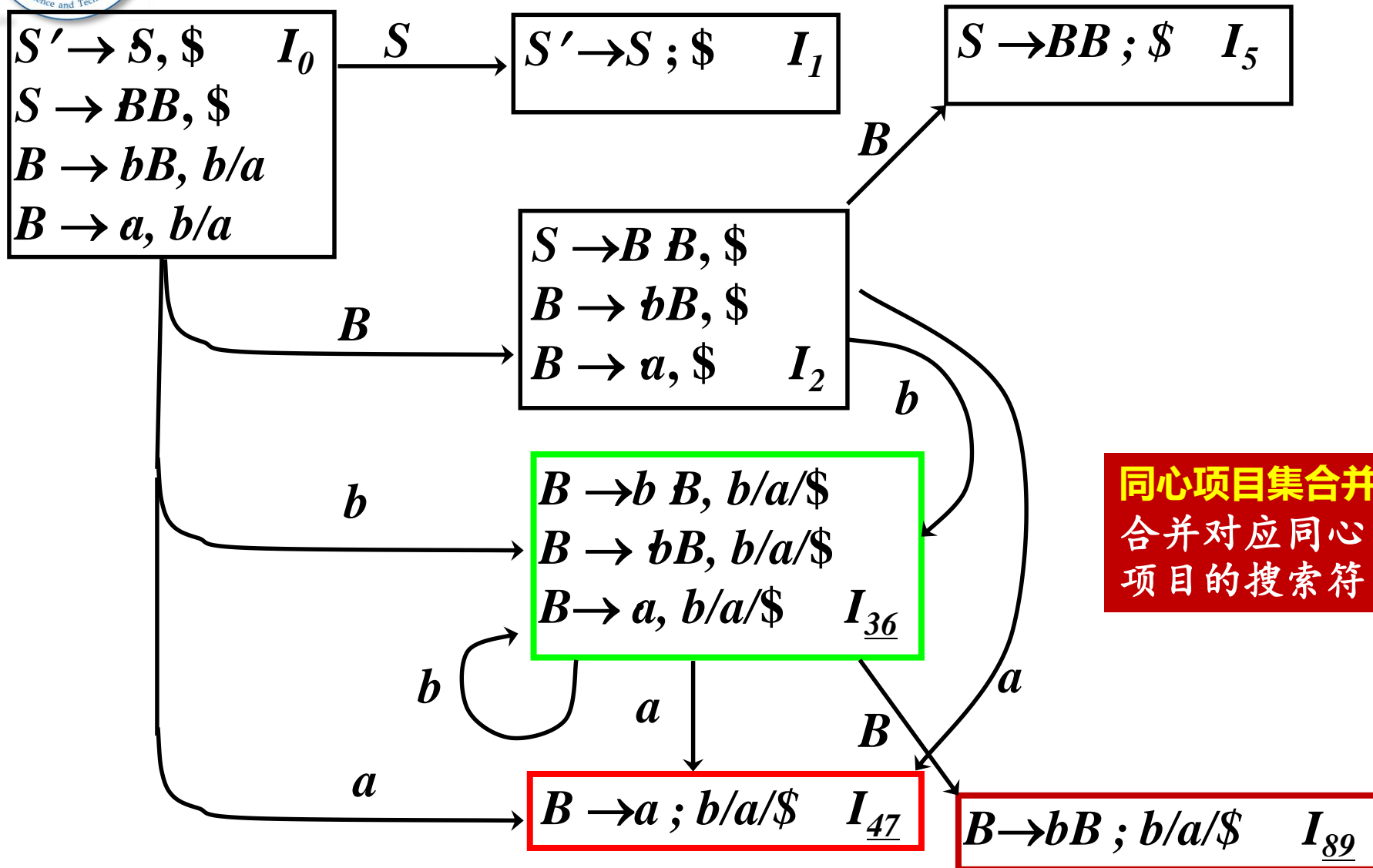
两个项目集在略去搜索符后
是相同的集合

右图有 3 对同心项目集
(I3和I6、I4和I7、I8和I9)



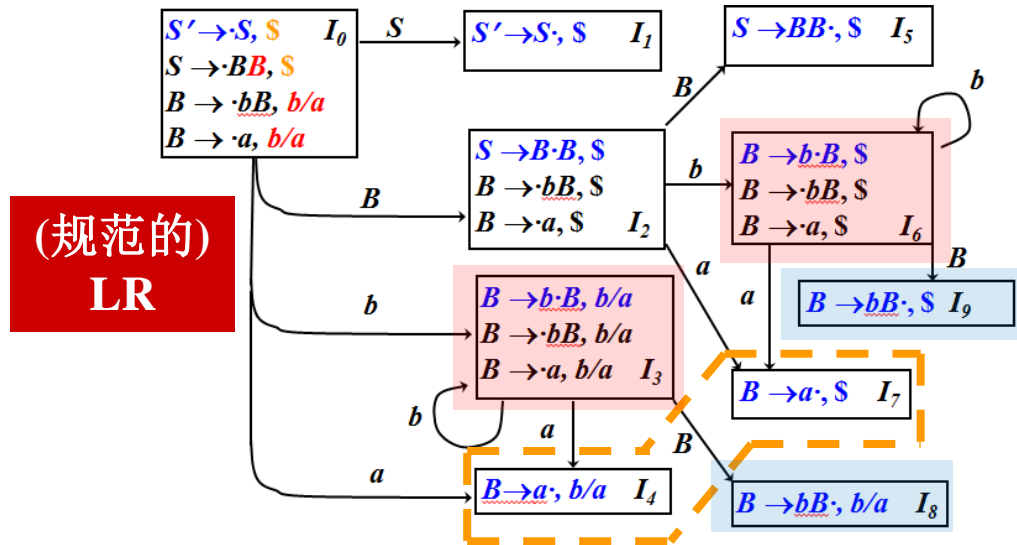
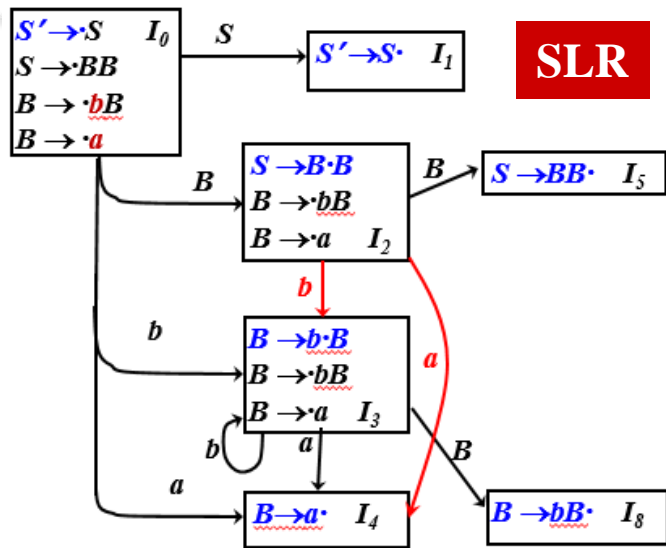


LALR分析





SLR vs. LR vs. LALR分析



□ 同心集的合并不会引起新的移进-归约冲突

项目集1

$[A \rightarrow \alpha ; a]$

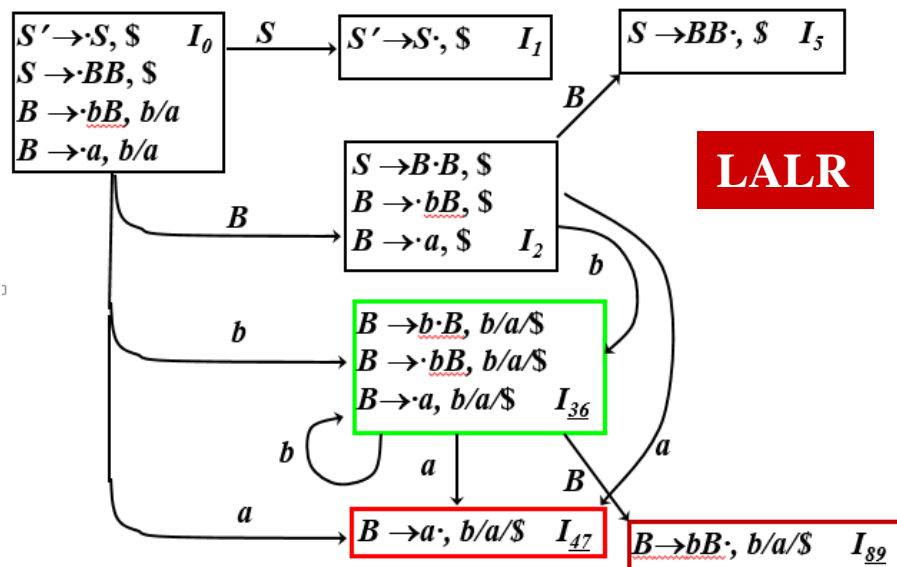
...

项目集2

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

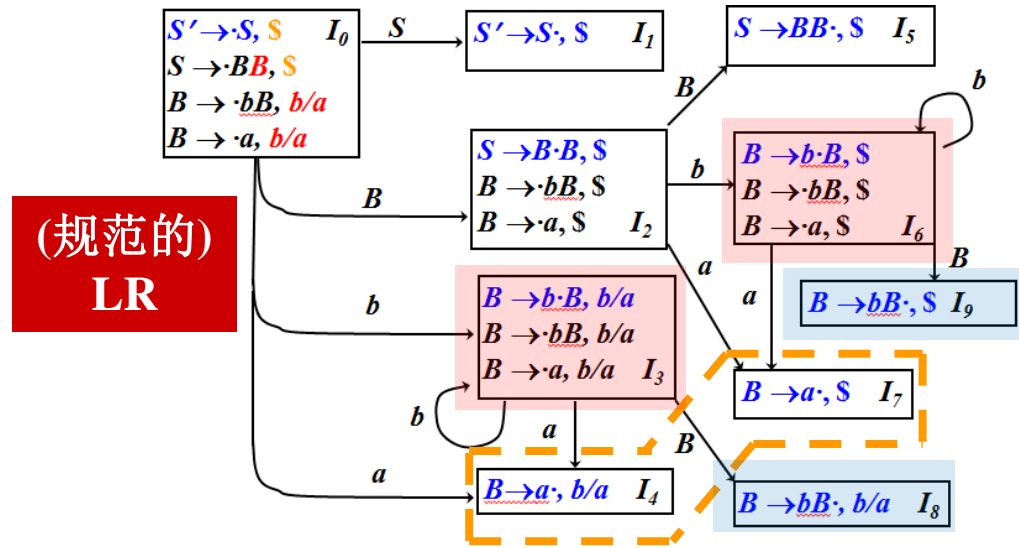
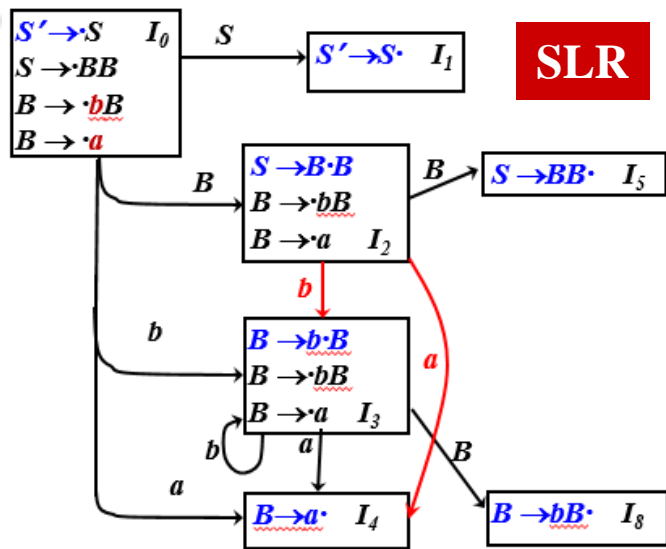
...

若合并后有冲突





SLR vs. LR vs. LALR分析



□ 同心集的合并不会引起新的移进-归约冲突

项目集1

$[A \rightarrow \alpha ; a]$

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

...

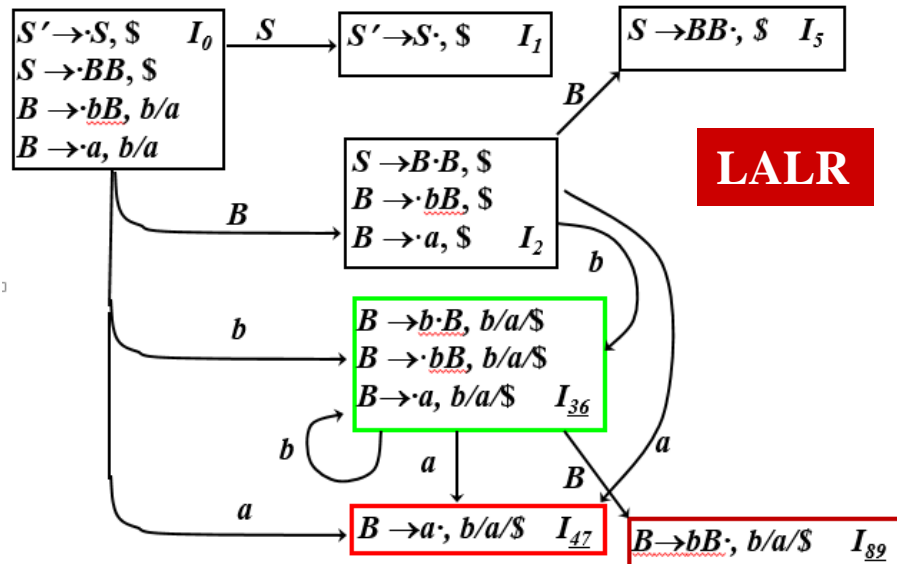
项目集2

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

$[A \rightarrow \alpha ; d]$?

...

则合并前就有冲突





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

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

$S \$ \Rightarrow V = E \$$ 无句型 $E = E$ ☹️

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

项目 $S \rightarrow V \cdot = E$ 使得

$\text{action}[2, =] = s_6$

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

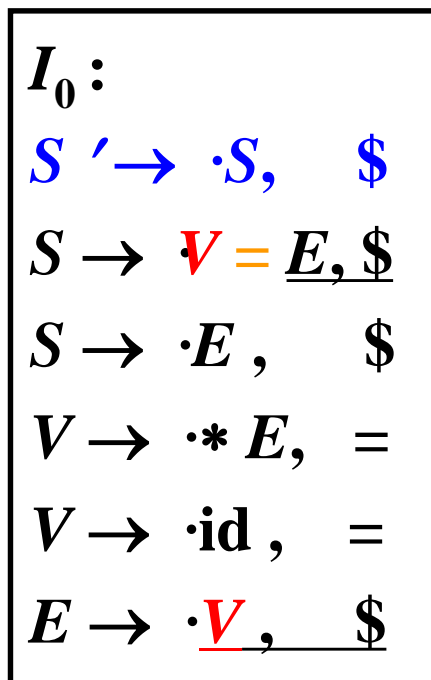
$S \rightarrow V = E$

$S \rightarrow E$

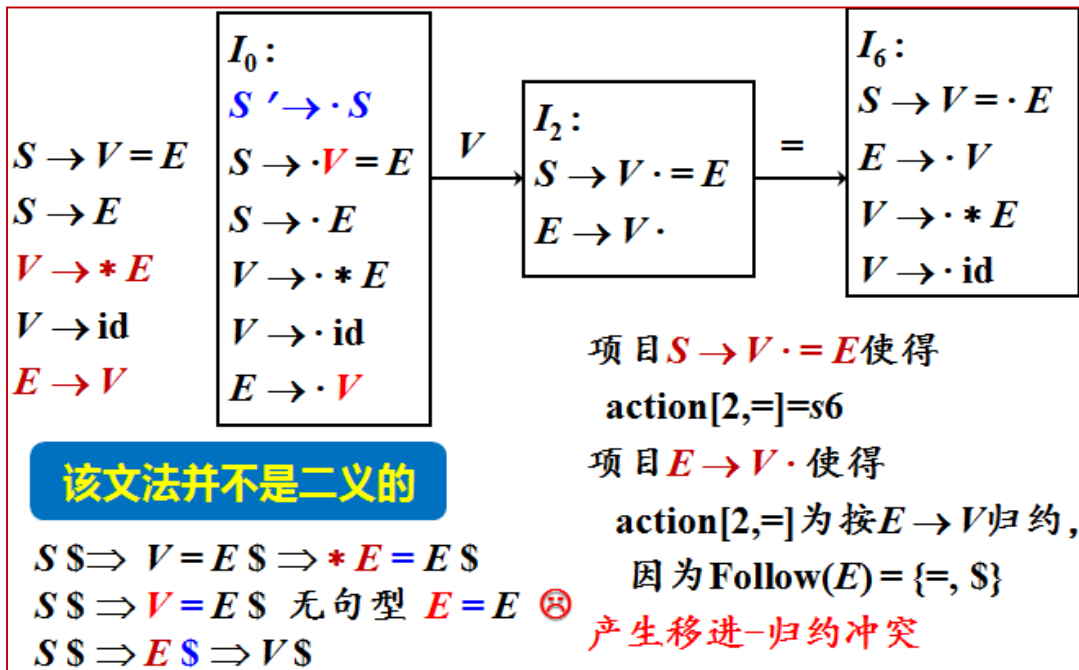
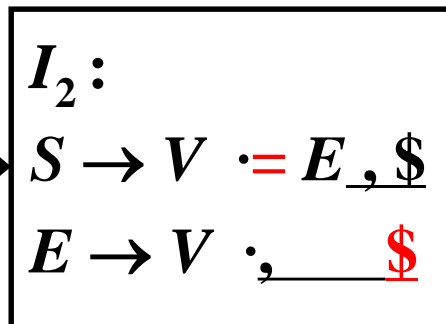
$V \rightarrow * E$

$V \rightarrow \text{id}$

$E \rightarrow V$



V



LR(1) 分析
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-归约分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**bb**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*

没读到c移进
读到c后规约



非LR的上下文无关结构

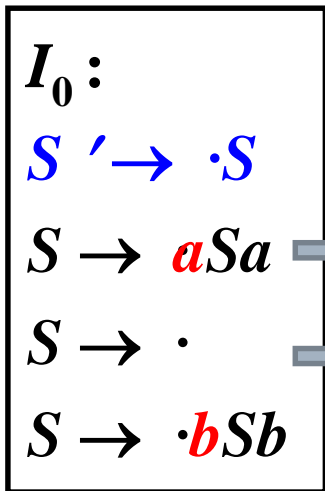
若自左向右扫描的移进-归约分析器能及时识别出现在栈顶的句柄，那么相应的文法就是LR（指规范的LR）的。

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**bb**aba*



→ 面临 a ，移进

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

存在移进-归约冲突
故不是SLR(1)文法



非LR的上下文无关结构

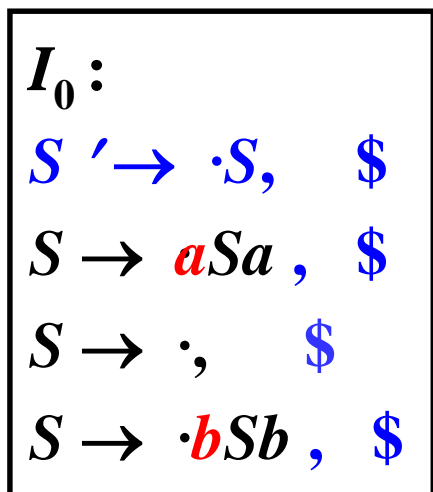
若自左向右扫描的移进-归约分析器能及时识别出现在栈顶的句柄，那么相应的文法就是LR（指规范的LR）的。

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

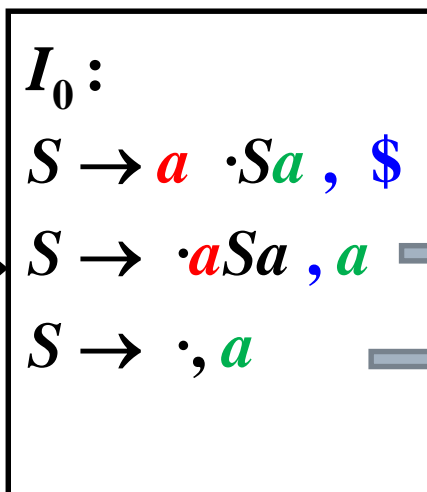
$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

不是LR的

*ababb**bb**aba*



a



面临*a*，移进

$\because a$ 在项目搜索符中

\therefore 面临*a*，归约

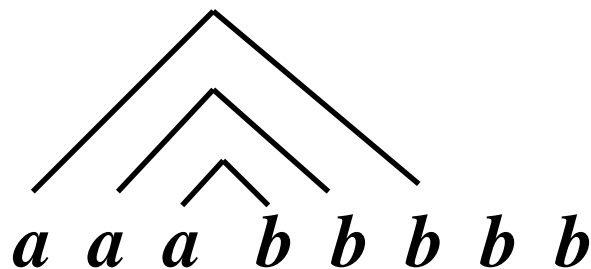
存在移进-归约冲突
故不是LR(1)文法



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: $S \rightarrow AB$ $A \rightarrow aAb \mid \varepsilon$ $B \rightarrow Bb \mid b$



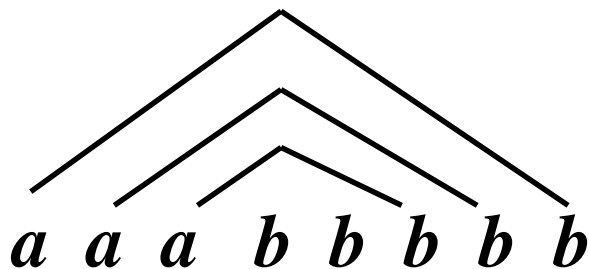


例题 写不同的文法

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: $S \rightarrow AB$ $A \rightarrow aAb \mid \varepsilon$ $B \rightarrow Bb \mid b$

□ 非二义且非 LR(1) 的文法: $S \rightarrow aSb \mid B$ $B \rightarrow Bb \mid b$

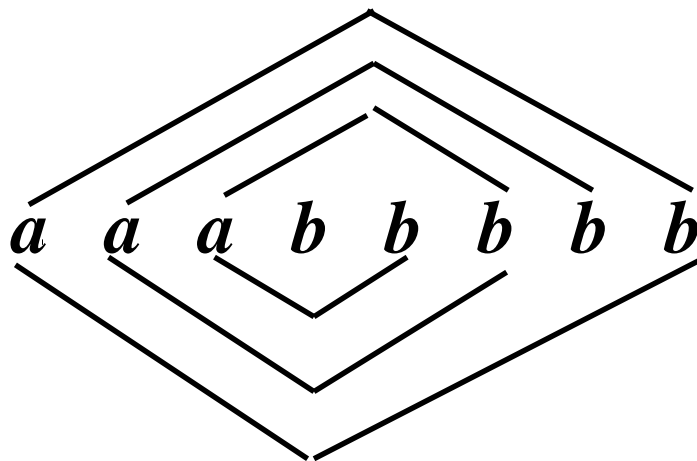




例题 写不同的文法

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: $S \rightarrow a\textcolor{blue}{S}b \mid Sb \mid b$



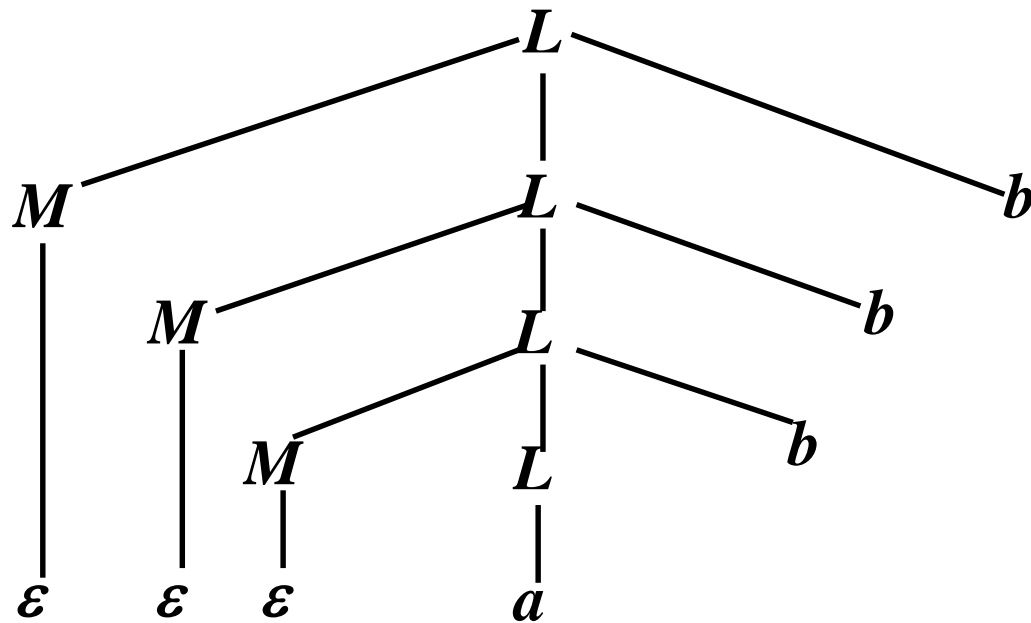


例题4

试说明下面文法不是LR(1)的：

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

$$M \rightarrow \varepsilon$$



句子 $abbbb$ 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分析的活前缀性质?



LR项目与LR文法小结

□ LR(**0**)项目 $[A \rightarrow \alpha \cdot \beta]$ 、LR(**1**)项目 $[A \rightarrow \alpha \cdot \beta, a]$

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

$S' \rightarrow \cdot S$
 $S \rightarrow \cdot a$
 $S \rightarrow \cdot$

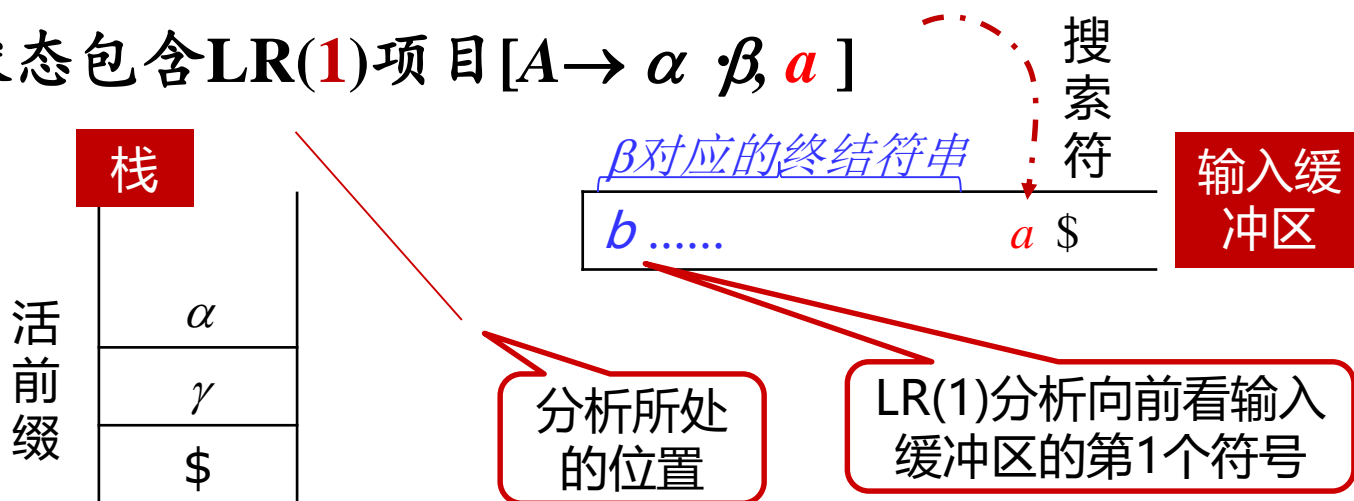
不向前看时,
移进-归约
冲突

□ SLR(0)文法

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

□ 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