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

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

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

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

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 $S \to aABe$

$A \to Abc \mid b$

$B \to d$

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 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

输入串： $abbcde$

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　$S \to aABe$

$A \to Abc \mid b$

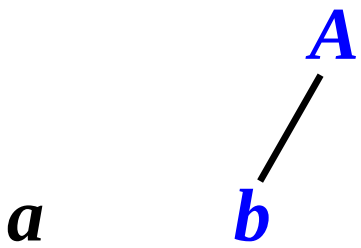
$B \to d$

输入串： $abbcde$

ab

aA（归约）

用产生式 $A \to b$
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 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

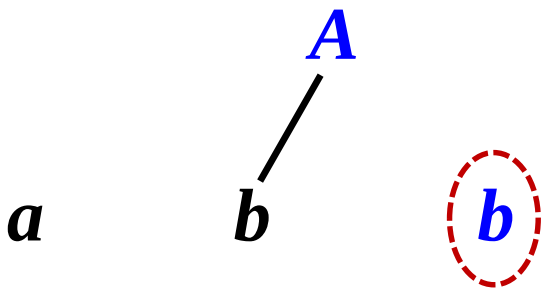
输入串： $abbcde$

ab

aAb（再读入b）

$A \rightarrow b$

b可以归约成A吗?



归约（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 $S \to aABe$

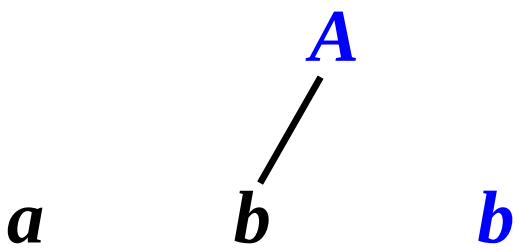
$A \to Abc \mid b$

$B \to d$

输入串： $abbcde$

ab

aAb（再读入b）



$A \to b$

b可以归约成A吗?

归约后能否形成右句型?

aAAcde 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 $S \rightarrow aABe$

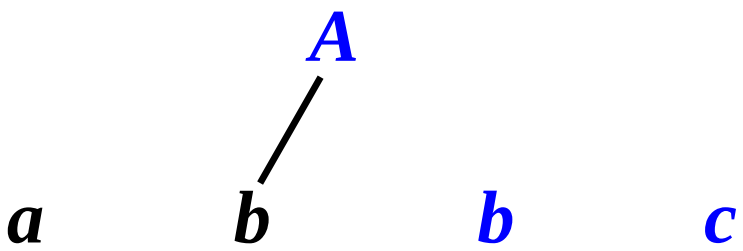
$A \rightarrow Abc \mid b$

$B \rightarrow d$

输入串： $abbcde$

ab

$aAbc$（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 $S \to aABe$

$A \to Abc \mid b$

$B \to d$

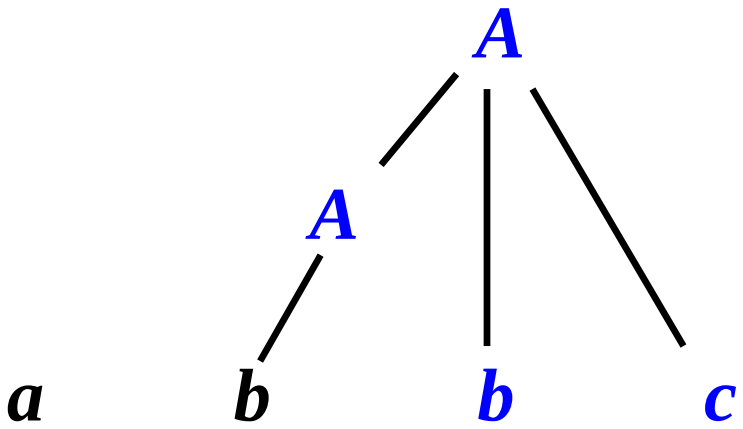
输入串： $abbcde$

ab

$aAbc$

aA（归约）

$A \to Abc$



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

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　$S \to aABe$

$A \to Abc \mid b$

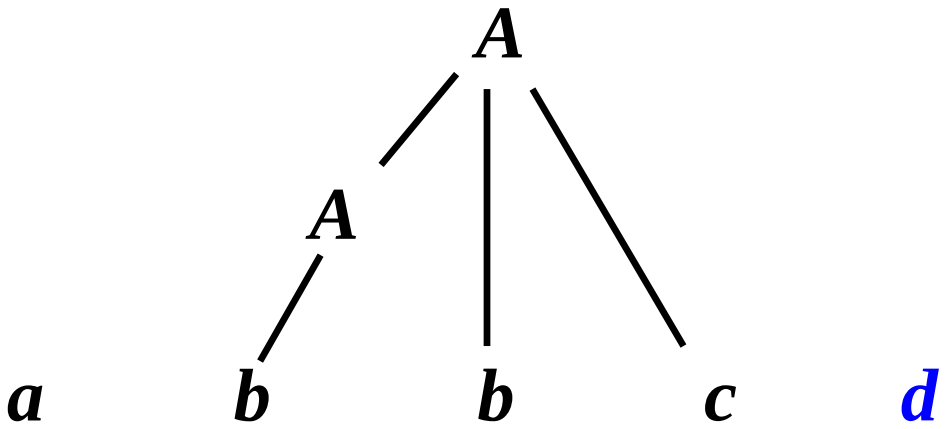
$B \to d$

输入串：$abbcde$

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 $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

$B \rightarrow d$

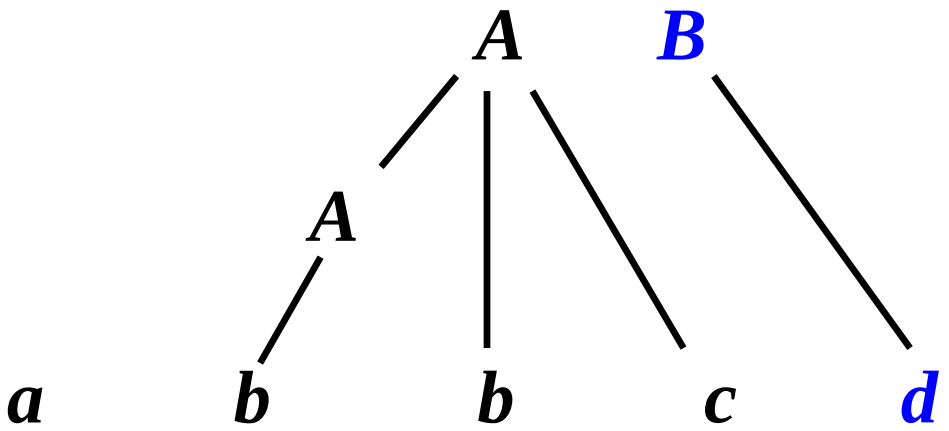
输入串： $abbcde$

ab

$aAbc$

aAd

aAB **(归约)**



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

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 $S \to aABe$

$A \to Abc \mid b$

$B \to d$

输入串： *abbcde*

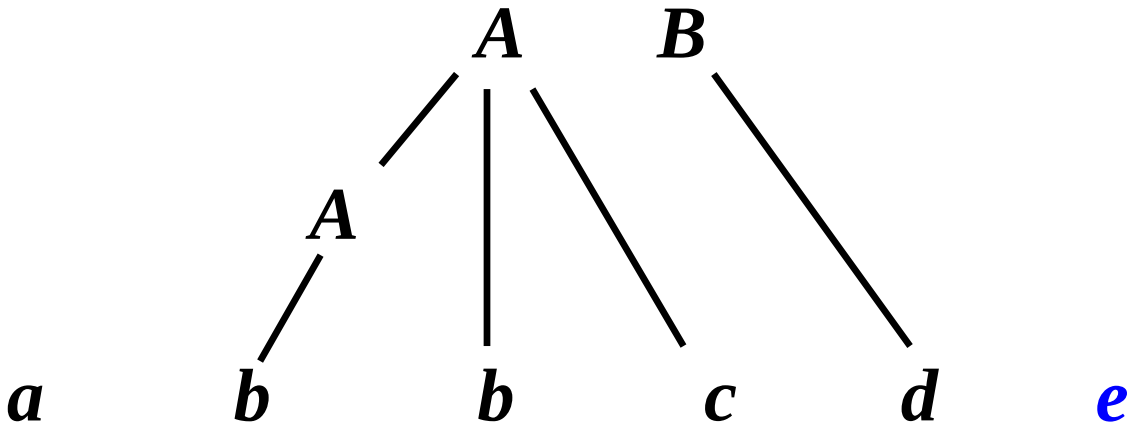
ab

aAbc

aAd

aAB

aABe(再读入*e*)



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

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 $S \to aABe$

$A \to Abc \mid b$

$B \to d$

输入串： *abbcde*

ab

aAbc

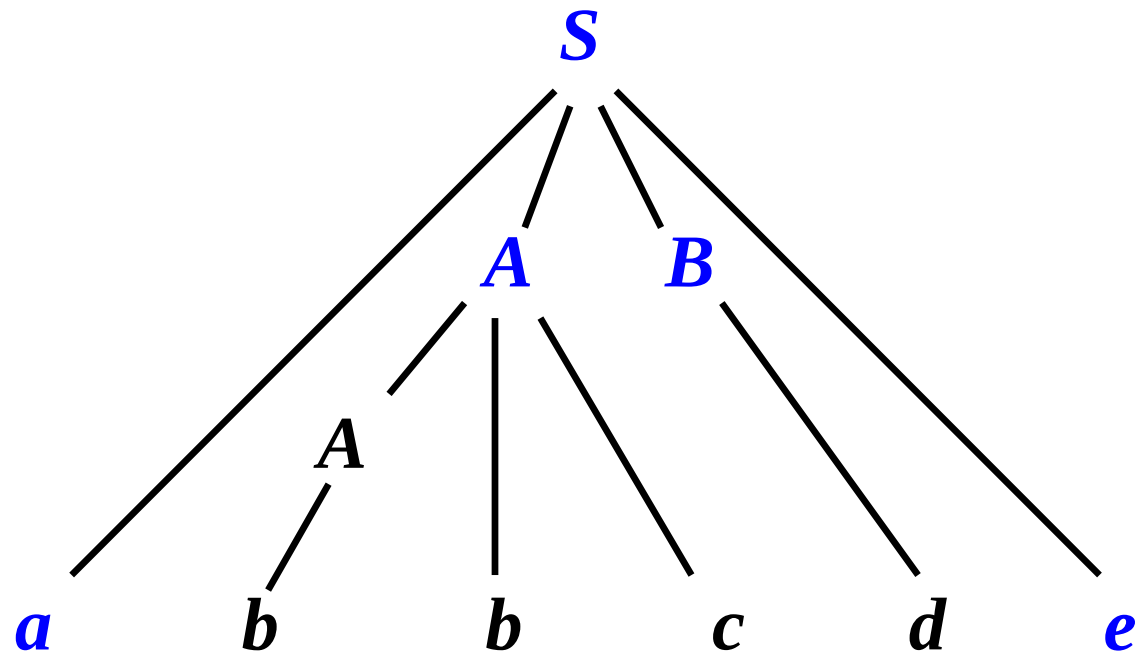
aAd

aAB

aABe

S（归约）

$S \to aABe$



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

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

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 $B \to \delta$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

  $S \to aABe$

  $A \to Abc \mid b$

  $B \to d$

  * 右句型：最右推导可得的句型

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

- 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

$$E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$$

$$\begin{aligned}
E &\Rightarrow_{rm} E * E \\
&\Rightarrow_{rm} E * E + E \\
&\Rightarrow_{rm} E * E + \mathrm{id}_3 \\
&\Rightarrow_{rm} E * \mathrm{id}_2 + \mathrm{id}_3 \\
&\Rightarrow_{rm} \mathrm{id}_1 * \mathrm{id}_2 + \mathrm{id}_3
\end{aligned}$$

例 句柄不唯一

$$E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$$

$$
\begin{aligned}
E &\Rightarrow_{rm} E * E \\
&\Rightarrow_{rm} E * E + E \\
&\Rightarrow_{rm} E * E + \mathrm{id}_3 \\
&\Rightarrow_{rm} E * \mathrm{id}_2 + \mathrm{id}_3 \\
&\Rightarrow_{rm} \mathrm{id}_1 * \mathrm{id}_2 + \mathrm{id}_3
\end{aligned}
\qquad
\begin{aligned}
E &\Rightarrow_{rm} E + E \\
&\Rightarrow_{rm} E + \mathrm{id}_3 \\
&\Rightarrow_{rm} E * E + \mathrm{id}_3 \\
&\Rightarrow_{rm} E * \mathrm{id}_2 + \mathrm{id}_3 \\
&\Rightarrow_{rm} \mathrm{id}_1 * \mathrm{id}_2 + \mathrm{id}_3
\end{aligned}
$$

在右句型$E * E + \mathrm{id}_3$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
$$E$	$* id_2 + id_3$$	

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

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？

$$E \Rightarrow_{rm} E*E \Rightarrow_{rm} E*E+E \Rightarrow_{rm} E*E+id_3$$ 移进

$$E \Rightarrow_{rm} E+E \Rightarrow_{rm} E+id_3 \Rightarrow_{rm} E*E+id_3$$ 归约

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 → id (*parameter_list*) | *expr* = *expr*　　id(…)是函数调用

  parameter_list → *parameter_list*, *parameter* | *parameter*

  parameter → id

  expr → id (*expr_list*) | id　　id(…)也表示数组元素的引用

  expr_list → *expr_list*, *expr* | *expr*

由*A*(*I*, *J*)开始的语句

归约成*expr*还是*parameter*？

栈	输入
… id (id	, 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 → **procid** (*parameter_list*) | *expr* = *expr*

  parameter_list → *parameter_list*, *parameter* | *parameter*

  parameter → **id**

  expr → **id** (*expr_list*) | **id**　　**id(…)也表示数组元素的引用**

  expr_list → *expr_list*, *expr* | *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	输入
… procid(id	, id)…

  - 需要修改上面的文法

方法2
改写文法，区分id是否是procid

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

3.5 LR分析器

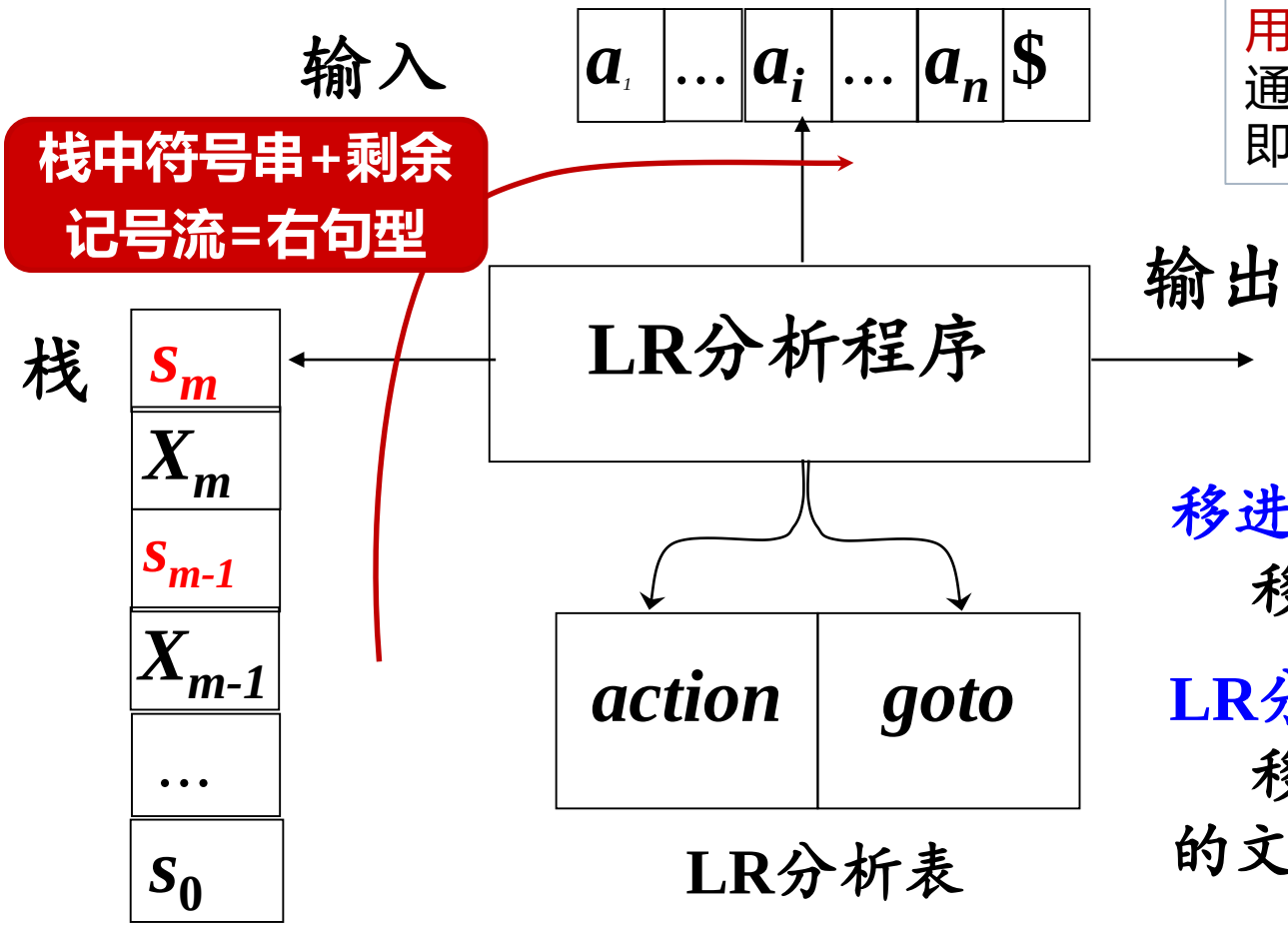
(L-scanning from left to right; R- rightmost derivation in reverse)

- 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

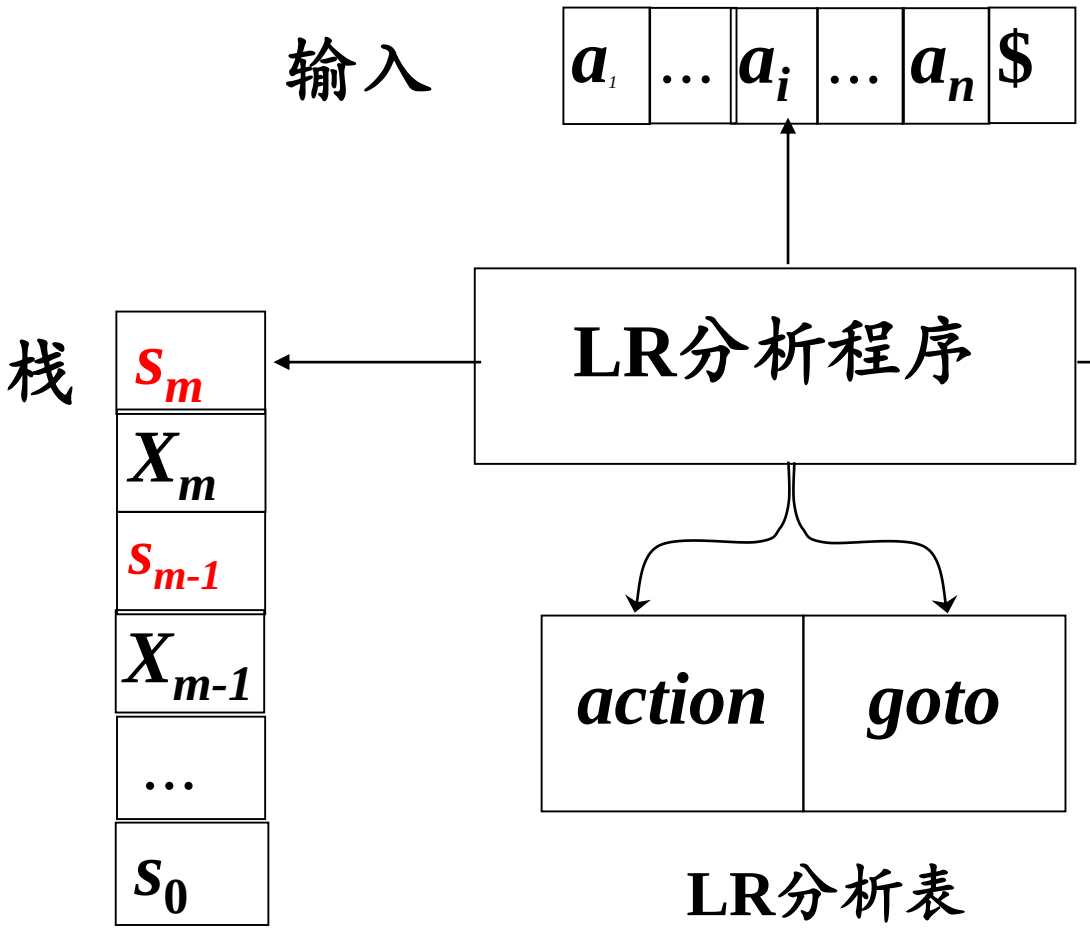


用**状态**刻画栈中的内容
通过查看栈顶状态，
即可判断是否形成句柄

移进-归约分析器
移进符号

LR分析器
移进**状态**和其相关的文法符号(s_0除外)

LR分析算法: 分析器的模型



s_j: 是栈中该状态到栈底信息的浓缩

action[s_m, a_i]:

移进|归约|接受|出错

goto[s_{m-r}, A]=s_j: 移进A和s_j (归约后使用)

LR分析算法：举例

实际可不用移进

例 P69

$E \to E + T \mid E \to T$

$T \to T * F \mid T \to E$

$F \to (E) \mid F \to \mathrm{id}$

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			

LR分析算法：举例

栈	输　入	动　作
0	id * 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 (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 $	

1. 查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goto[0, F] =>3
   - 将(F, 3)入栈

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 $	

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* → id归约
0 *T* 2 * 7 *F* 10	+ id $	按*T* → *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* → id归约
0 *T* 2 * 7 *F* 10	+ id $	按*T* → *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

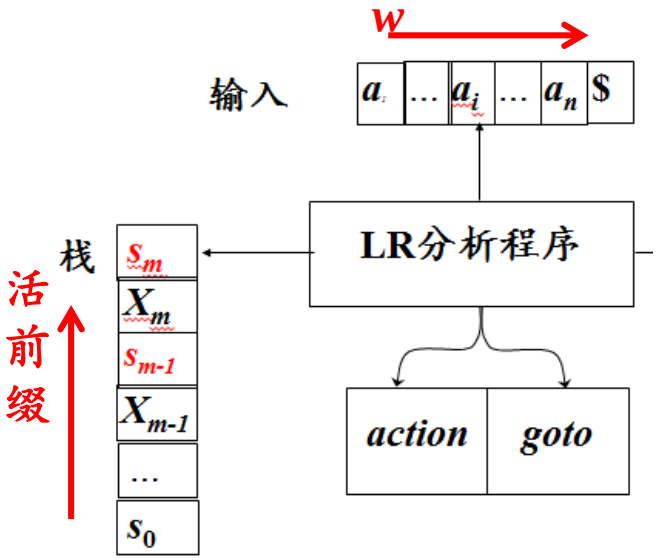
活前缀（viable prefix）

- 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	+	*	(	)	$	*E*	*T*	*F*
0	*s*5			*s*4			1	2	3
1		*s*6				*acc*			
2		*r*2	*s*7		*r*2	*r*2			
3		*r*4	*r*4		*r*4	*r*4			
4	*s*5			*s*4			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		

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

$A \rightarrow l\beta$

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		

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

$A \rightarrow l\beta$

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符后，便要确定用哪个产生式推导

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

$A \rightarrow l\beta$

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

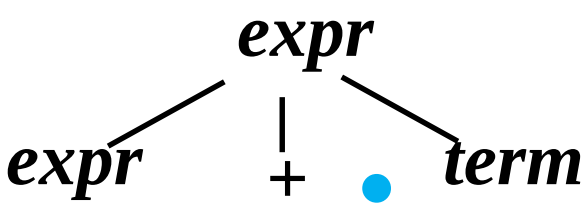
- LR(0)项目与LR(1)项目

- 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

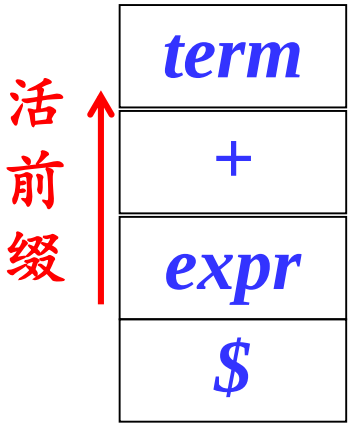
- **LR(0) 项目**
  - 在右部的某个地方加点的产生式
  - 加点的目的是用来表示分析过程所处的位置

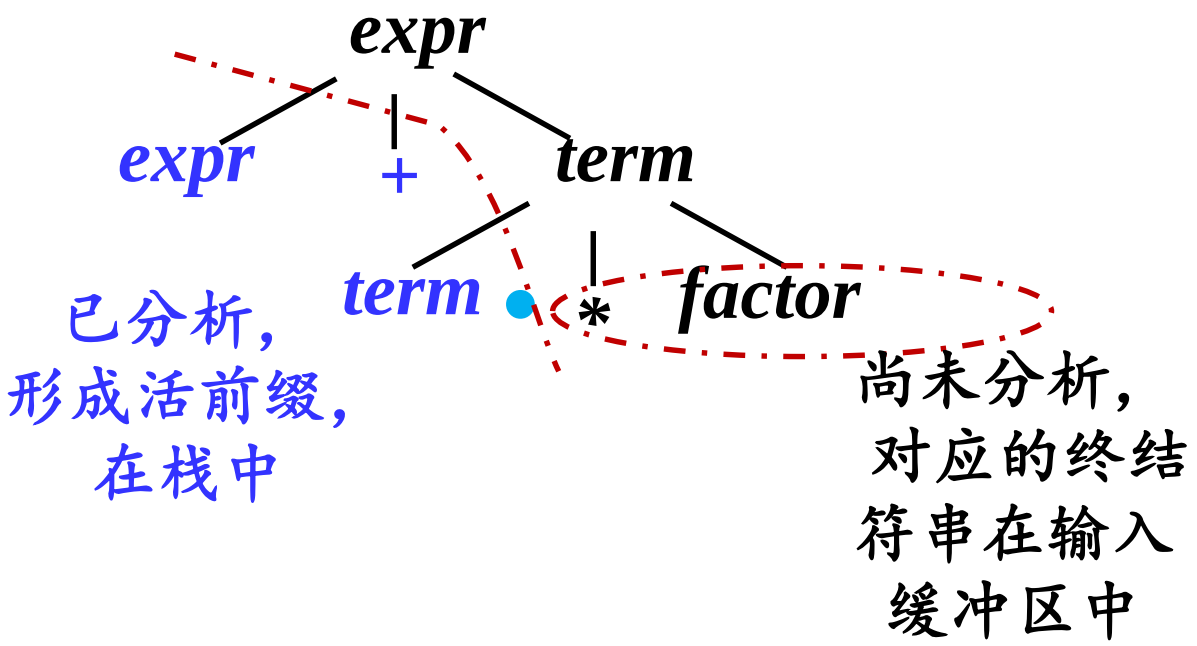
$expr \rightarrow expr + \cdot term$



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 $A \rightarrow XYZ$ 对应有四个项目

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

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

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 $[A\to\alpha\cdot\beta]$， 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

$E'\to E$ 旨在指示分析器何时开始分析、何时完成分析

$E\to E+T\mid T$

$T\to T*F\mid F$

$F\to(E)\mid \mathrm{id}$

$[E'\to\cdot E]$ $[E'\to E\cdot]$

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)

初始状态

I_0:

$E'\rightarrow\cdot E$

LR分析表的构造方法

- LR(0) 项目 $[A\to\alpha\cdot\beta]$， 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:

$E'\to\cdot E$

$E\to\cdot E+T$

$E\to\cdot T$

求项目集的闭包closure(I) P75

求LR(0)项目集的闭包closure(I) P75

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

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

LR分析表的构造方法

- LR(0) 项目 $[A\to\alpha\ \cdot\beta]$， 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:

$E'\to\cdot E$

$E\to\cdot E+T$

$E\to\cdot T$

$T\to\cdot T*F$

$T\to\cdot F$

求项目集的闭包closure(*I*) P75

求LR(0)项目集的闭包closure(*I*) P75

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

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

LR分析表的构造方法

- LR(0) 项目 $[A\to\alpha\cdot\beta]$，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:

$E'\to\cdot E$

$E\to\cdot E+T$

$E\to\cdot T$

$T\to\cdot T*F$

$T\to\cdot F$

$F\to\cdot(E)$

$F\to\cdot id$

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$S \Rightarrow^*_{rm} \delta Aw \Rightarrow_{rm} \delta\alpha\beta w$，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 => 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

$$S' \to S$$
$$S \to BB$$
$$B \to bB \mid a$$

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:

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot bB$

$B \rightarrow \cdot a$

- LR(1)项目集规范族

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

FIRST(B) ={a, b}

求LR(0)项目集的闭包closure(I)

P75

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

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

求LR(1)项目集的闭包closure(I)

P82

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

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

$b \in \mathbf{FIRST}(\beta a)$

构造识别活前缀的DFA (以LR(1)项目集为例)

$S' \to S$
$S \to BB$
$B \to bB \mid a$

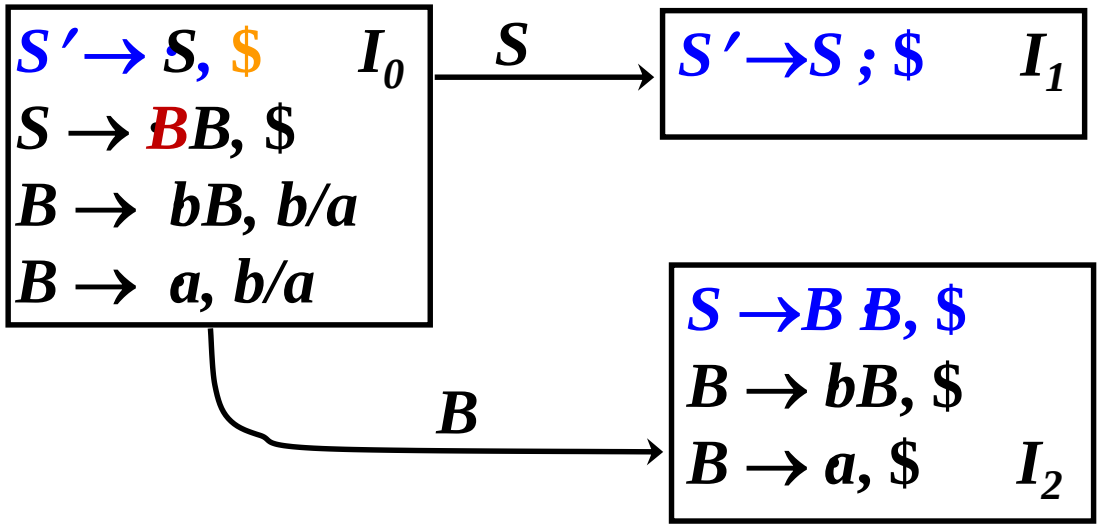
$S' \to S,\ \$ \quad I_0$
$S \to BB,\ \$$
$B \to bB,\ b/a$
$B \to a,\ b/a$

S

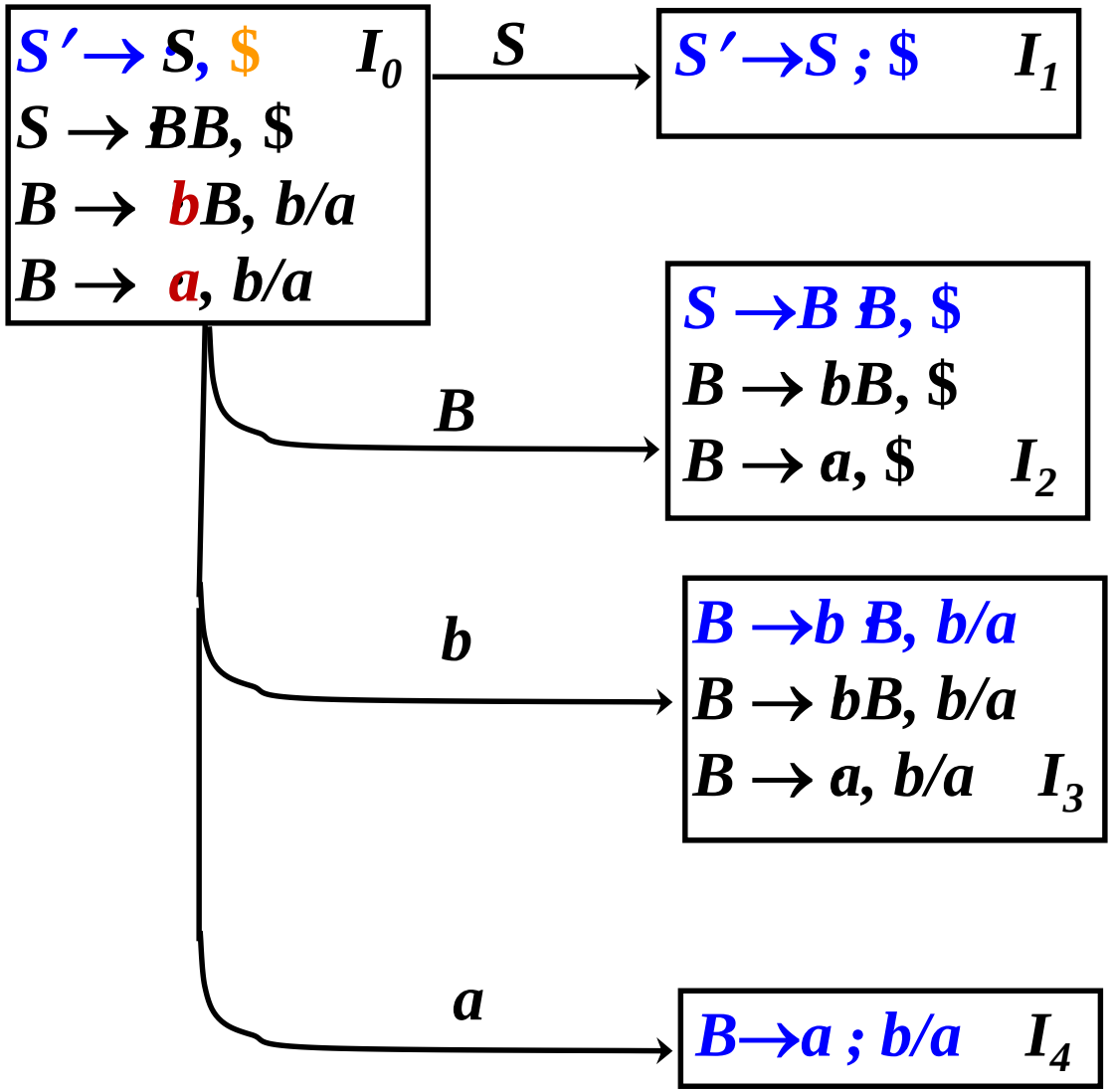
$S' \to S\ ;\ \$ \quad I_1$

$I_1 := goto\ (\ I_0,\ S\ )$

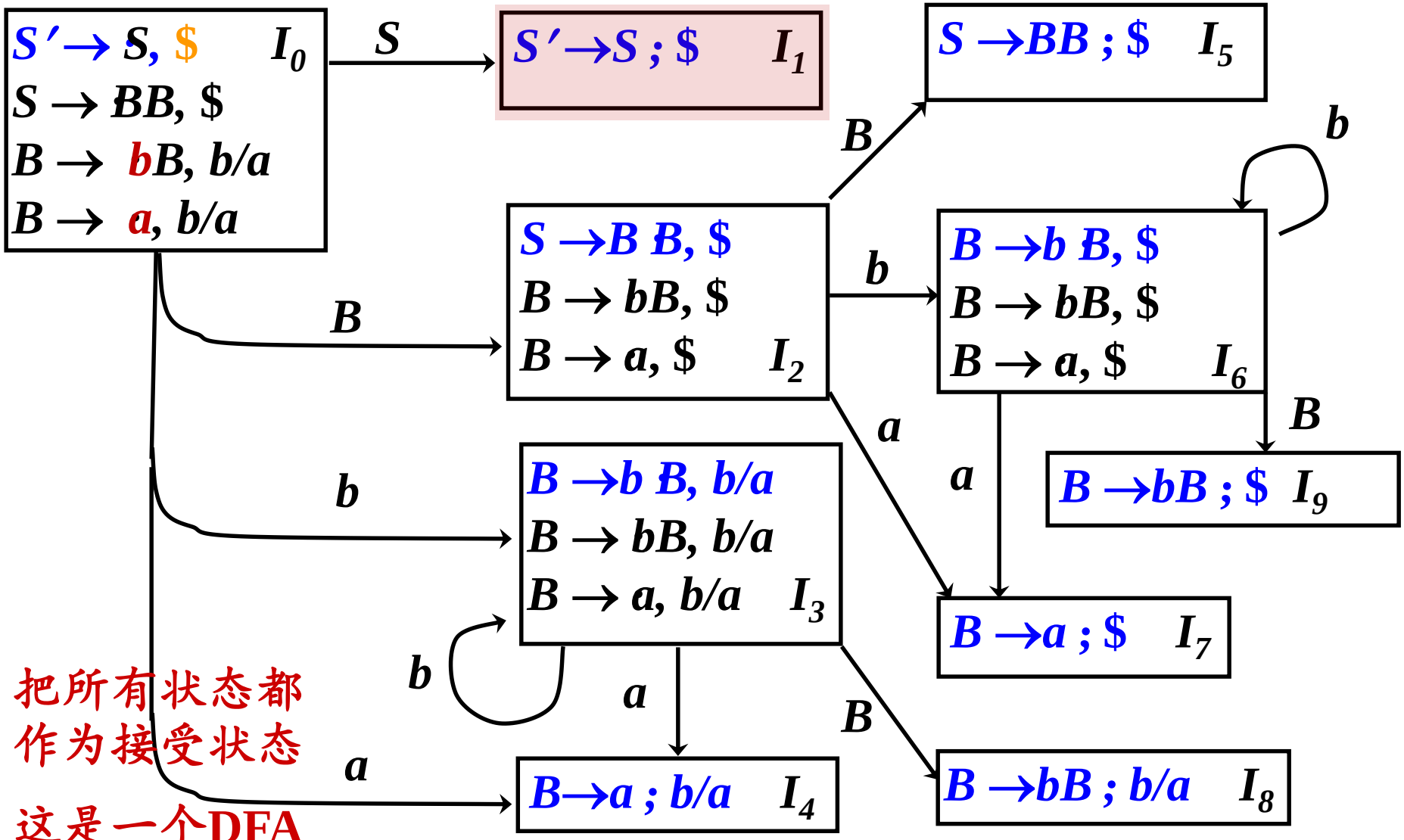
构造识别活前缀的DFA (以LR(1)项目集为例)



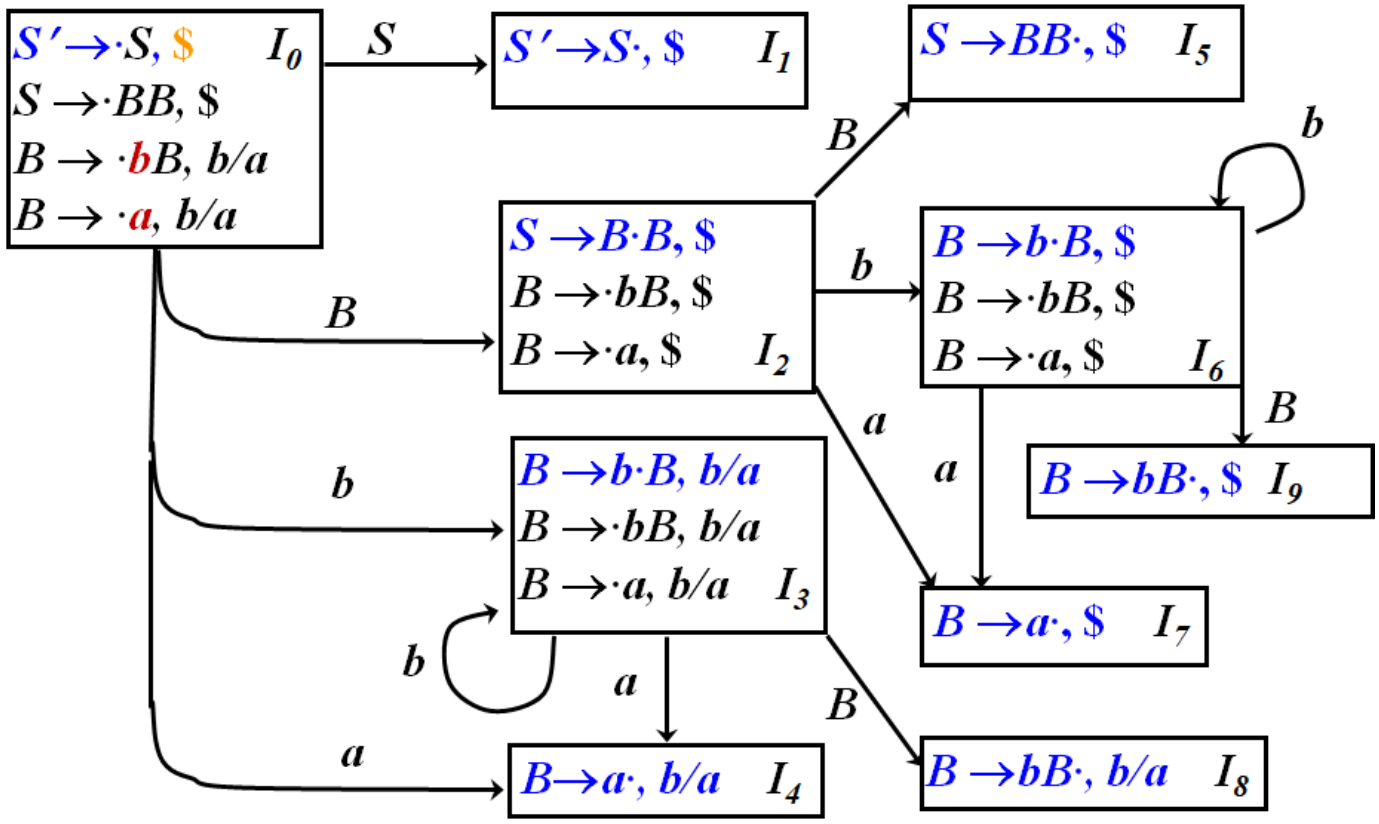
构造识别活前缀的DFA (以LR(1)项目集为例)



构造识别活前缀的DFA (以LR(1)项目集为例)



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:

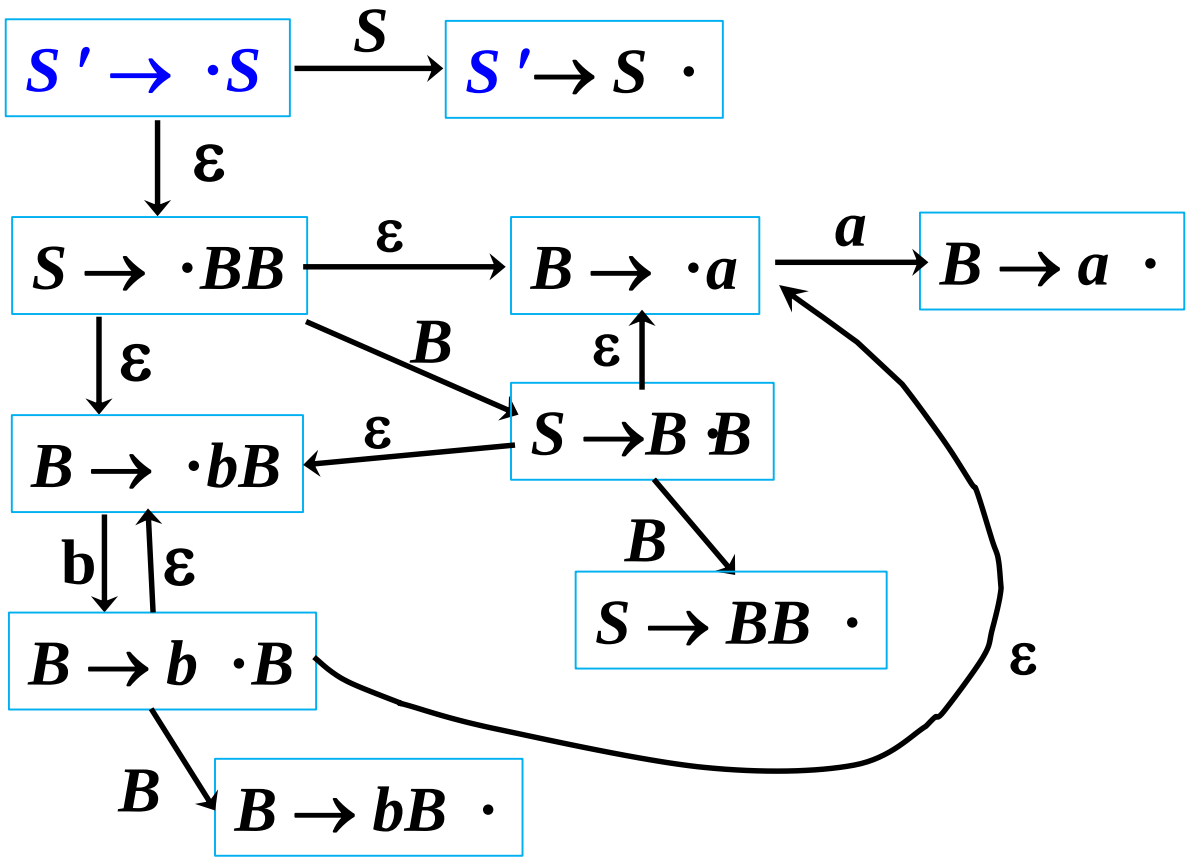
$S' \to \cdot S$

$S \to \cdot BB$

$B \to \cdot bB$

$B \to \cdot a$

每个项目一个状态

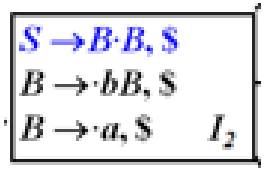


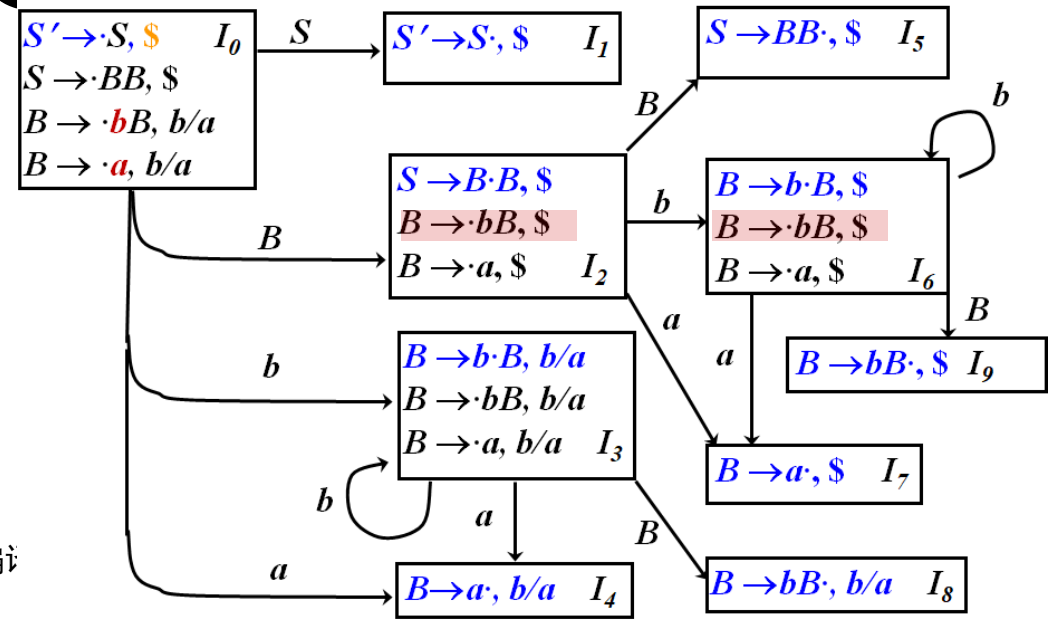
有效项目

- LR(0)项目[$A\to\alpha\cdot\beta$]对活前缀 $\gamma=\delta\alpha$有效：
  如果存在着推导$S'\Rightarrow^*_{rm}\delta Aw\Rightarrow_{rm}\delta\alpha\beta w$
- LR(1)项目[$A\to\alpha\cdot\beta, a$]对活前缀 $\gamma=\delta\alpha$ 有效：
  如果存在着推导 $S'\Rightarrow^*_{rm}\delta Aw\Rightarrow_{rm}\delta\alpha\beta w$，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

- [$B\to\cdot bB$,$]对活前缀B、Bb、Bbb都有效
- 活前缀B有多个有效项目

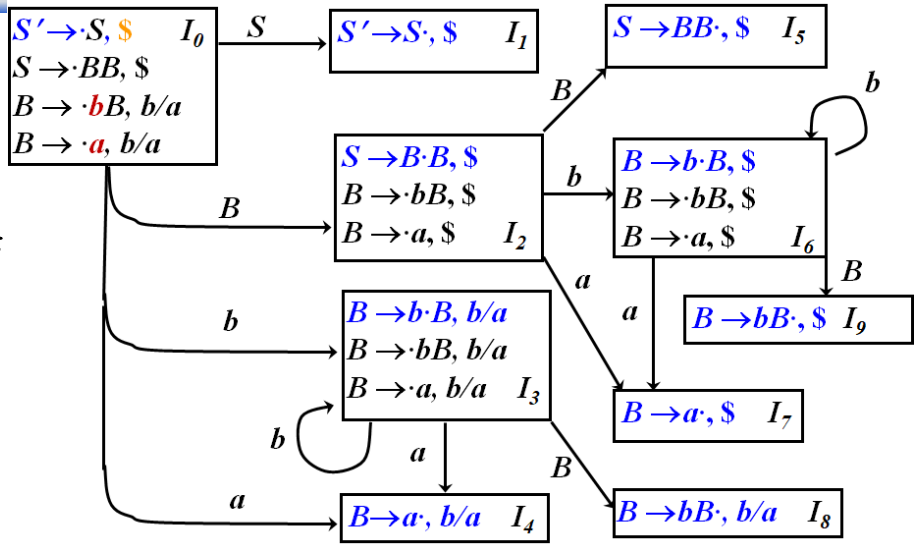




有效项目

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$\beta \neq \varepsilon$，应该移进
  - ✓ 如果$\beta = \varepsilon$， 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$[A\to\alpha\ a\beta]$在 I_i中，并且 $goto(I_i, a) = I_j$，那么置$action[i, a]$为sj
  - 归约：如果$[A\to\alpha\ \cdot]$在I_i中，那么对$FOLLOW(A)$中所有的a, 置$action[i, a]$为rj，j是产生式 $A\to\alpha$的编号
  - 接受：如果$[S'\to S\ \cdot]$在I_i中，那么置$action[\ i, \$\ ]$为$acc$

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$[A\to\alpha\ a\beta]$在 I_i中，并且 $goto(I_i, a) = I_j$，那么置$action[i, a]$为sj
  - 归约：如果$[A\to\alpha\ \cdot]$在I_i中，那么对$FOLLOW(A)$中所有的a, 置$action[i, a]$为rj，j是产生式 $A\to\alpha$的编号
  - 接受：如果$[S'\to S\ \cdot]$在I_i中，那么置$action[\ i, \$\ ]$为$acc$
- 构造状态i 的*goto*函数
  - 对所有的非终结符A, 如果$goto(I_i, A) = I_j$, 则$goto[i, A] = j$
- 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$[S'\to \cdot S]$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$[A \to \alpha \cdot a\beta, b]$在$I_i$中，且$goto(I_i, a) = I_j$，那么置$action[i, a]$为*sj*

② 如果$[A \to \alpha \cdot, a]$在I_i中，且$A \neq S'$，那么置$action[i, a]$为*rj*

③ 如果$[S' \to S \cdot, \$]$在$I_i$中，那么置$action[i, \$] = acc$

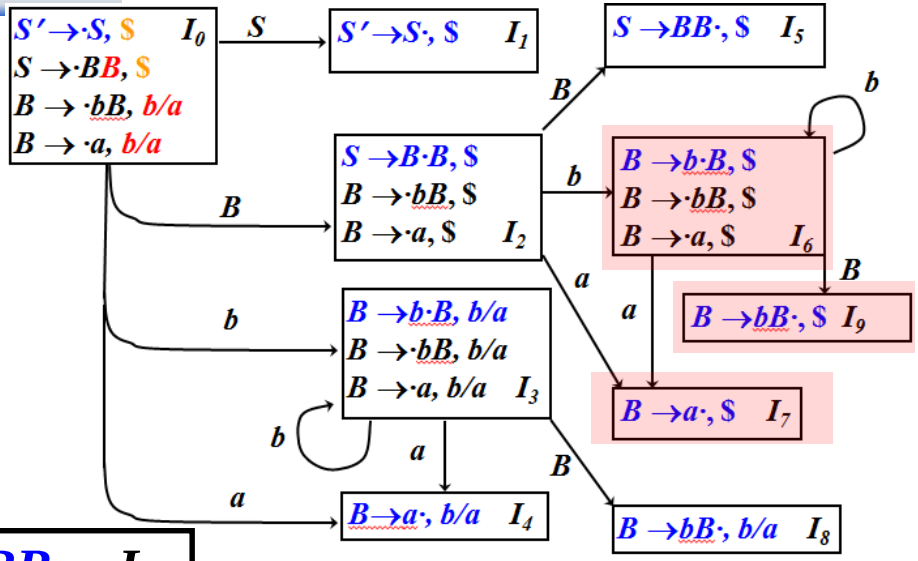
如果用上面规则构造，出现了冲突，则文法就不是LR(1)的

- *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$[S' \to \cdot S, \$]$的项目集对应的状态

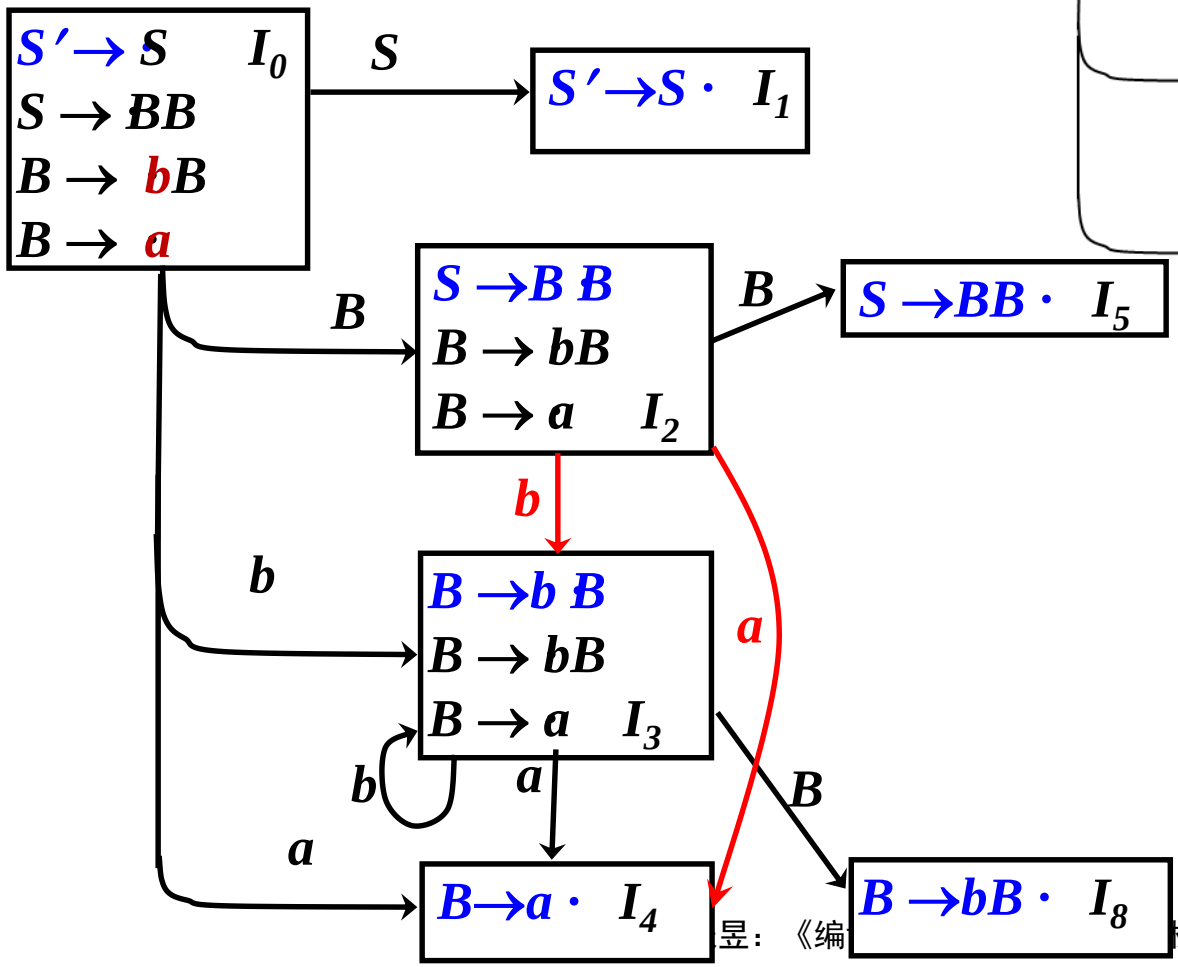
SLR是根据Follow(*A*)来确定归约动作
这里是根据搜索符（上下文信息）来确定

规范的LR vs. SLR 分析

规范的LR



SLR



规范的LR分析的状态数偏多

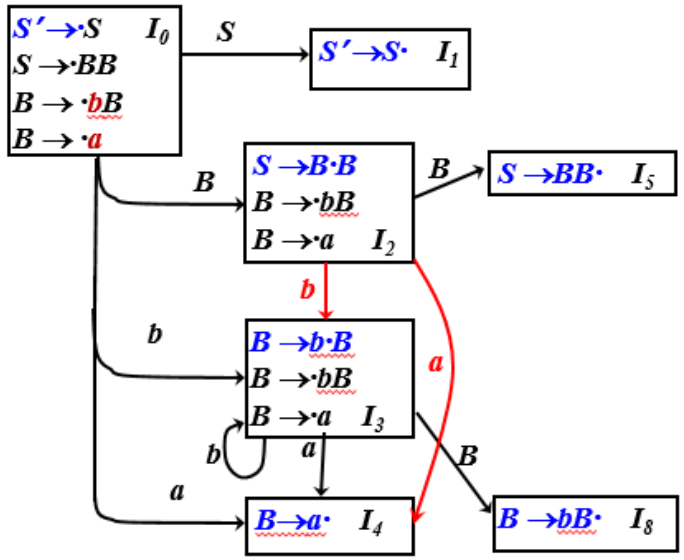
LALR分析表

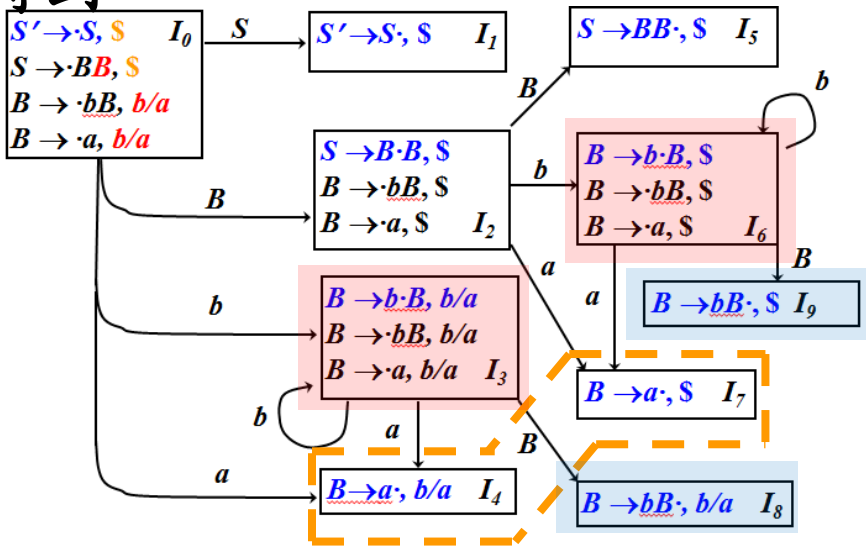
- LALR特点
  - 与SLR分析表有同样多状态
  - 能力介于SLR和规范LR之间
  - 其能力在很多情况下已够用
- LALR分析表构造方法
  - 通过合并规范LR(1)项目集来得到
- 同心的LR(1)项目集

两个项目集在略去搜索符后是相同的集合

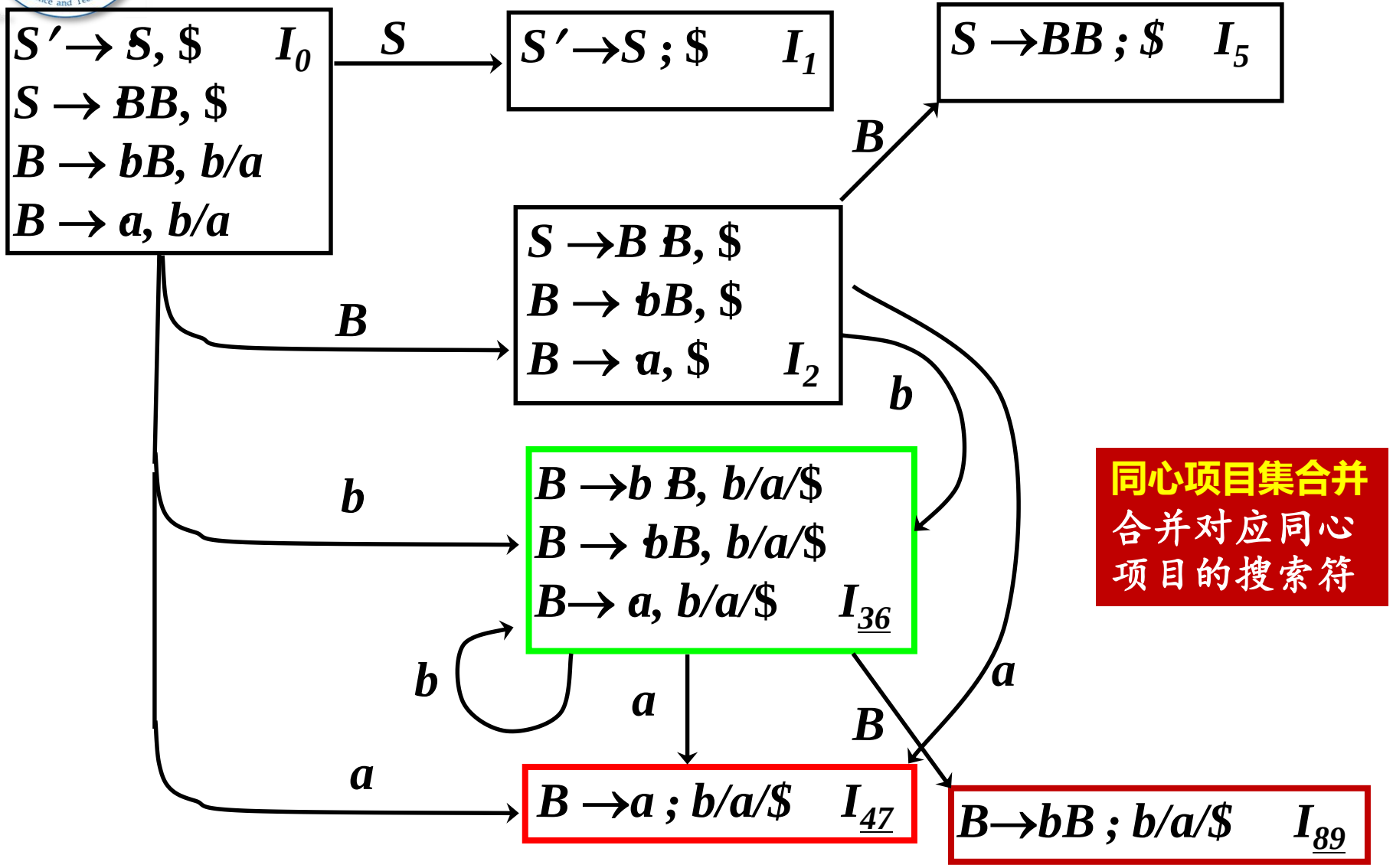
右图有 3 对同心项目集

(I3和I6、I4和I7、I8和I9)

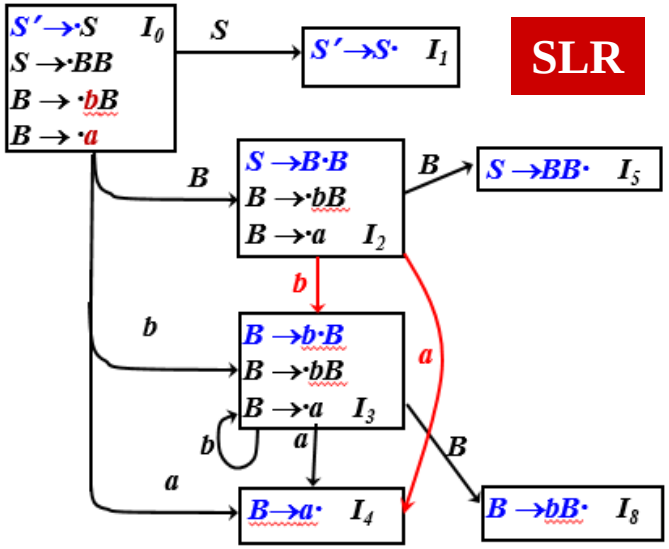


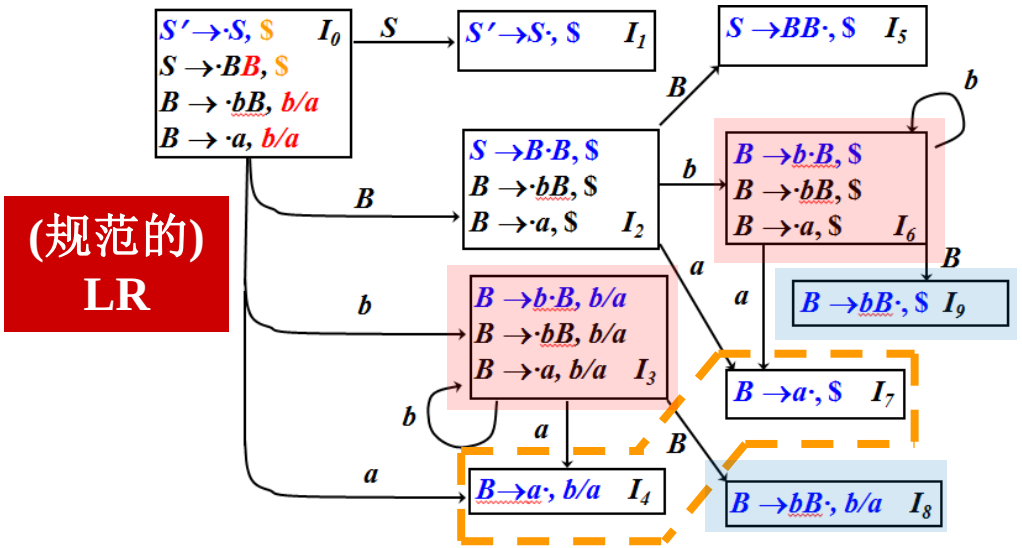


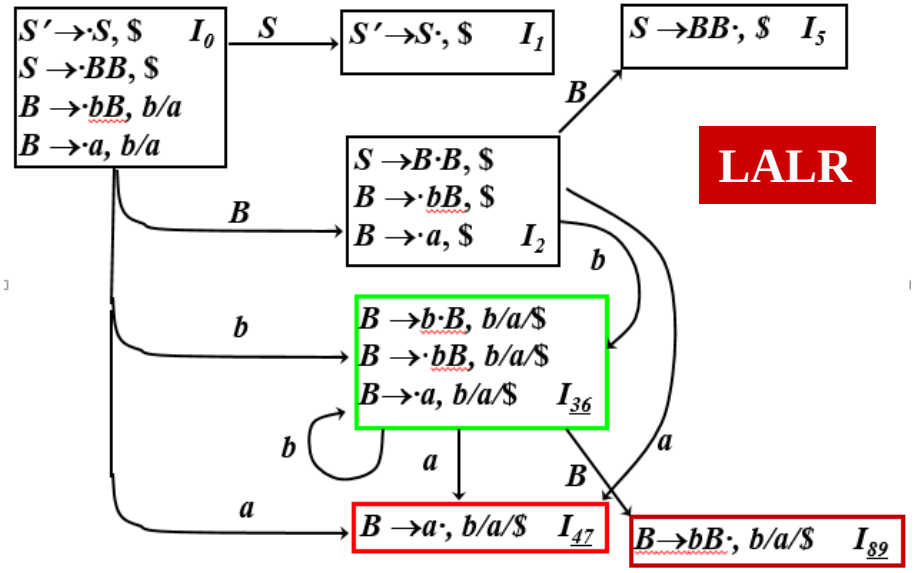
LALR分析


S′→S, $ I0
S→BB, $
B→bB, b/a
B→a, b/a
S
S′→S ; $ I1
S→BB ; $ I5
B
S→B B, $
B→bB, $
B→a, $ I2
B
b
B→b B, b/a/$
B→bB, b/a/$
B→a, b/a/$ I36
b
b
a
B
a
B→a ; b/a/$ I47
B→bB ; b/a/$ I89
a
同心项目集合并
合并对应同心项目的搜索符

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

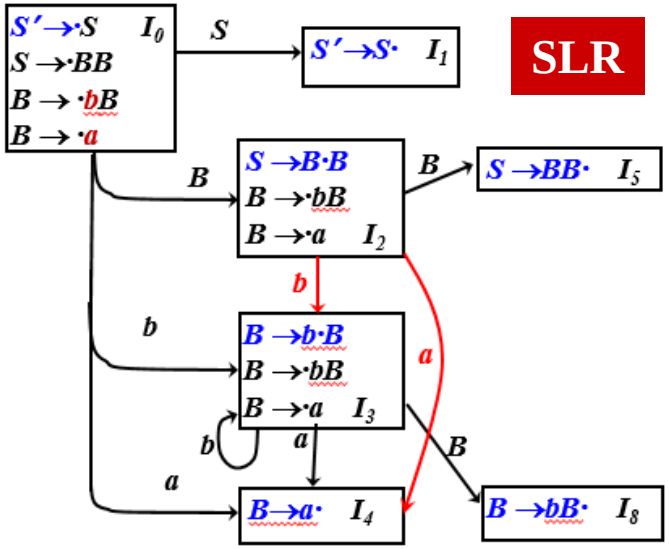
项目集1　　项目集2

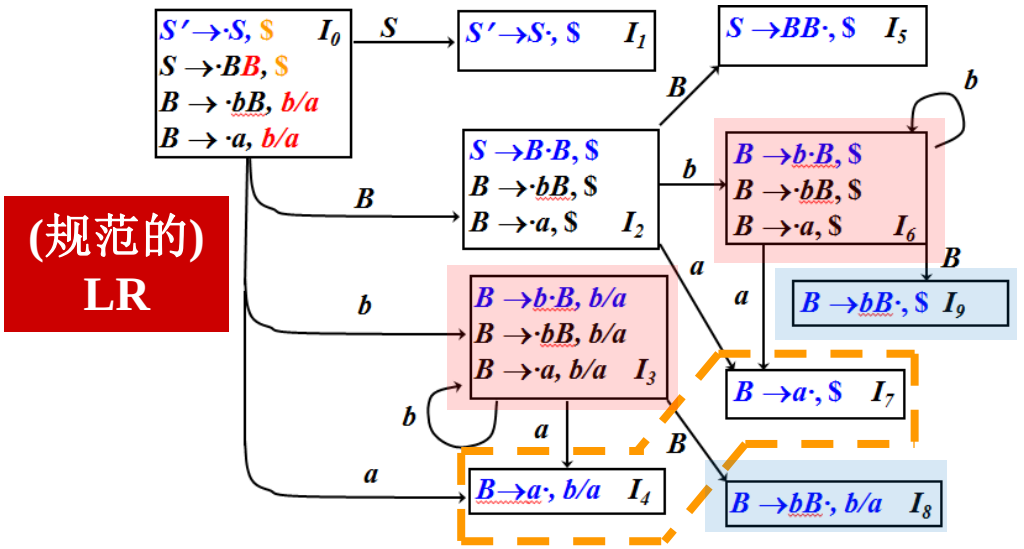
$[A\to\alpha\ ;a]$　　$[B\to\beta a\gamma, b]$

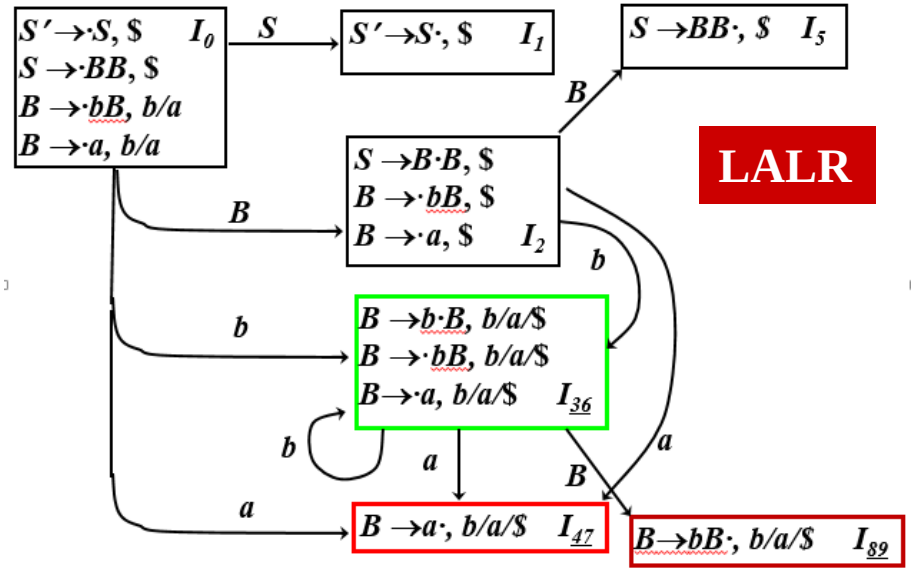
…　　…

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　　项目集2

$[A\to\alpha\ ;\ a]$　　$[B\to\beta\,a\gamma,\ b]$

$[B\to\beta\,a\gamma,\ c]$　　$[A\to\alpha\ ;\ d]$

…　　…

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

$S' \to S$
$S \to aAd \mid bBd \mid aBe \mid bAe$
$A \to c$
$B \to c$

对ac有效的项目集

$$A \to c \cdot, d$$
$$B \to c \cdot, e$$

对bc有效的项目集

$$A \to c \cdot, e$$
$$B \to c \cdot, d$$

合并同心集之后

$$A \to c \cdot ; d/e$$
$$B \to c \cdot ; d/e$$

该文法是LR(1)的，但不是LALR(1)的

SLR(1)文法的描述能力有限

$S \to V = E$

$S \to E$

$V \to * E$

$V \to \text{id}$

$E \to V$

I_0:

$S' \to \cdot S$

$S \to \cdot V = E$

$S \to \cdot E$

$V \to \cdot * E$

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

$E \to \cdot V$

V

I_2:

$S \to V \cdot = E$

$E \to V \cdot$

$=$

I_6:

$S \to V = \cdot E$

$E \to \cdot V$

$V \to \cdot * E$

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

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$S \to V \cdot = E$使得

action[2,=]=s6

项目$E \to V \cdot$使得

action[2,=]为按$E \to V$归约，

因为Follow(E) = {=, \$}

产生移进-归约冲突

不是SLR(1)但是LR(1)的文法

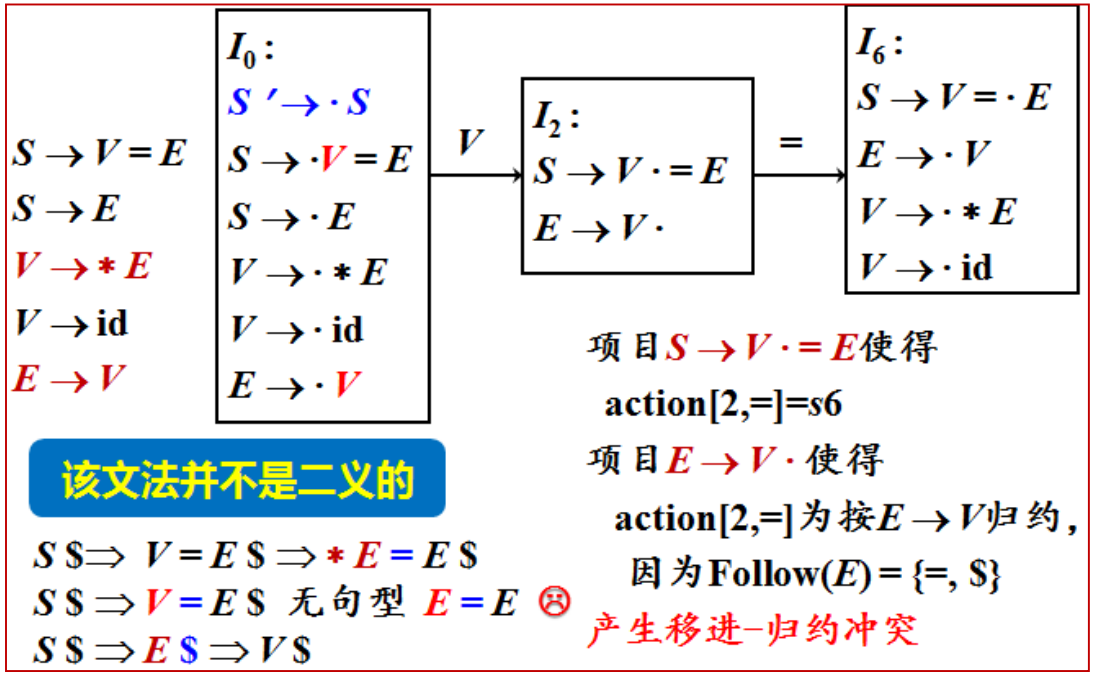
$S \to V = E$

$S \to E$

$V \to * E$

$V \to \text{id}$

$E \to V$



I_0:

$S' \to \cdot S,\quad \$$

$S \to \cdot V = E, \$$

$S \to \cdot E, \quad \$$

$V \to \cdot * E, \quad =$

$V \to \cdot \text{id}, \quad =$

$E \to \cdot V, \quad \$$

V

I_2:

$S \to V \cdot = E, \$$

$E \to V \cdot, \quad \$$

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　　*ababbbbaba*

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

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

是LR的　　*ababbcbbaba*

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

$ababbbbaba$

I_0:

$S' \rightarrow \cdot S$

$S \rightarrow a \cdot Sa$ ⟹ 面临a，*移进*

$S \rightarrow \cdot$ ⟹ $\because a \in Follow(S)$，$\therefore$面临a，*归约*

$S \rightarrow \cdot bSb$

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

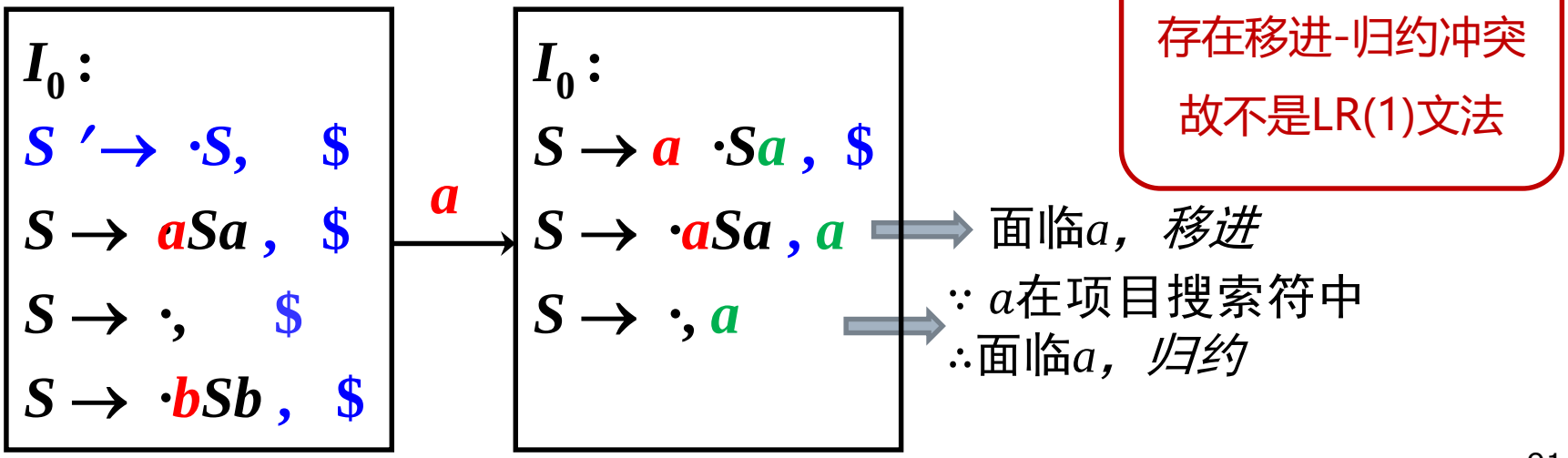
若自左向右扫描的移进–归约分析器能及时识别出现在栈顶的句柄，那么相应的文法就是LR（指规范的LR）的。

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

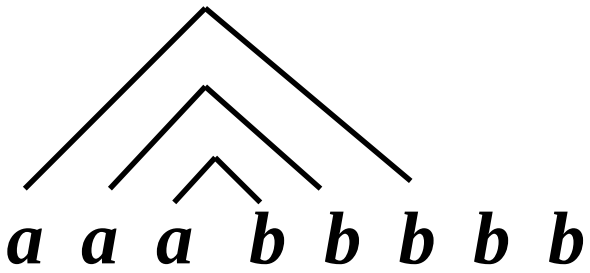
ababbbbaba



例题 写不同的文法

为语言$L=\{a^mb^n \mid n>m\geq 0\}$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。

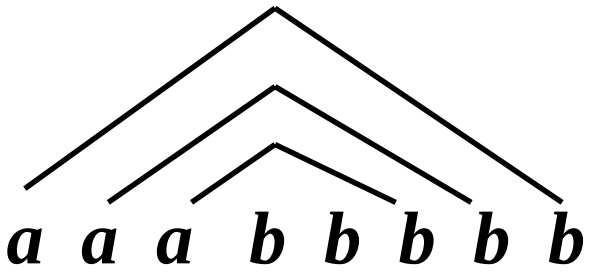
- LR(1)文法：$S \to AB$　　$A \to aAb \mid \varepsilon$　　$B \to Bb \mid b$



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。

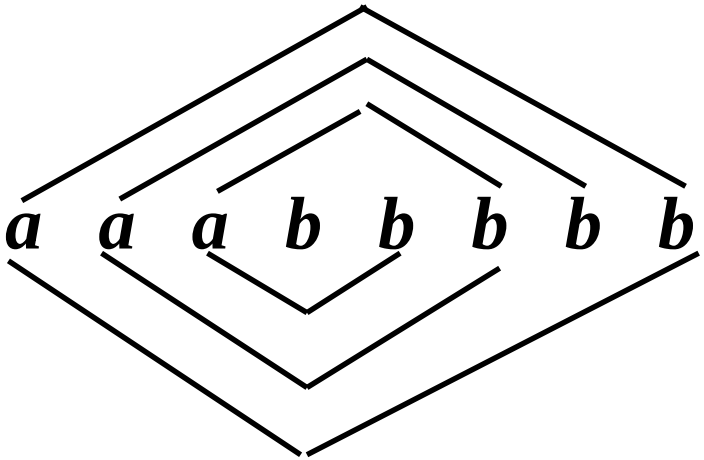
- LR(1)文法：$S \to AB \quad A \to aAb \mid \varepsilon \quad B \to Bb \mid b$
- 非二义且非LR(1)的文法：$S \to aSb \mid B \quad B \to Bb \mid b$



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：$S \to AB \quad A \to aAb \mid \varepsilon \quad B \to Bb \mid b$
- 非二义且非LR(1)的文法：$S \to aSb \mid B \quad B \to Bb \mid b$
- 二义的文法：$S \to aSb \mid Sb \mid b$

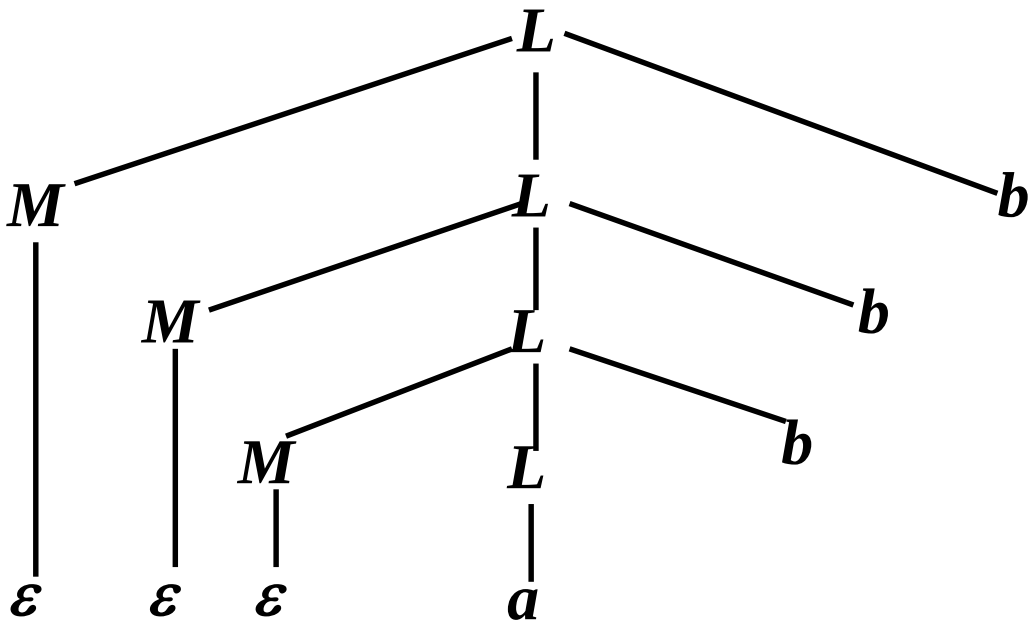


例题4

试说明下面文法不是LR(1)的：

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

$M \rightarrow \varepsilon$



句子*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 → begin *declist* ; *statement* end

declist → d ; *declist* | d

statement → 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 → begin *declist* *statement* end

declist → d ; *declist* | d ;

statement → 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).

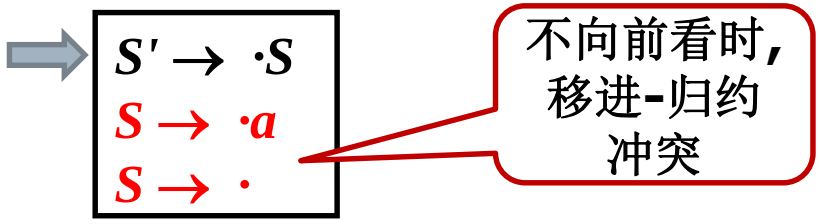
是否违反了LR分析的活前缀性质？

LR项目与LR文法小结

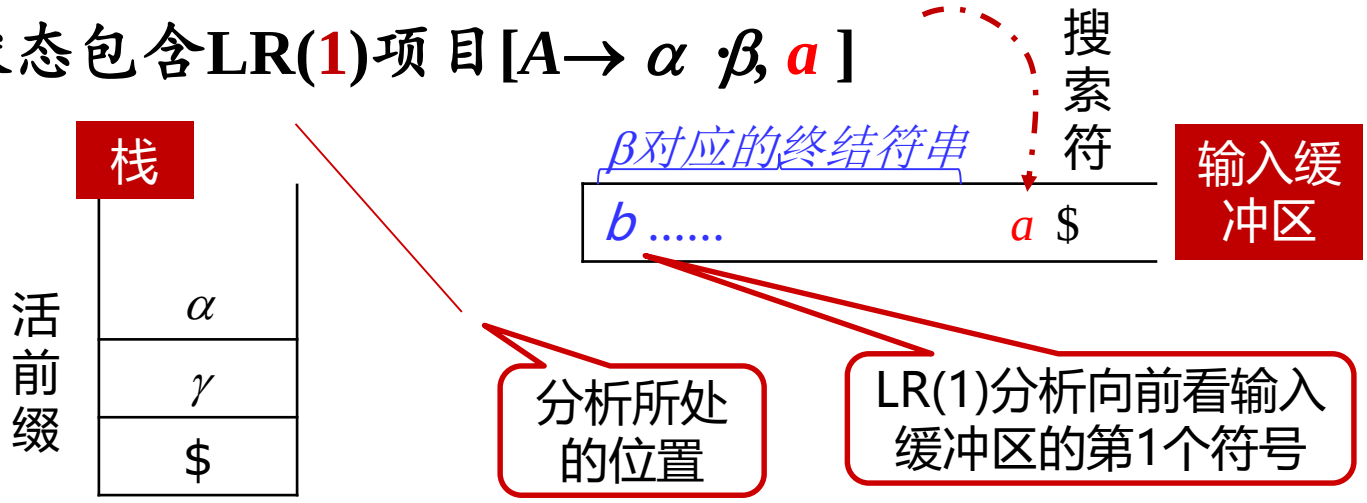
- LR(0)项目[$A \rightarrow \alpha \cdot \beta$]、LR(1)项目[$A \rightarrow \alpha \cdot \beta, a$]
  - 数字表示向前搜索的符号个数，0表示不向前搜索符号
- SLR(k)分析技术与SLR(k)文法
  - SLR(1)分析的状态：LR(0)项目集
  - k是指向前看输入缓冲区的k个符号
- [规范的]LR(k)分析技术与LR(k)文法
  - LR(1)分析的状态：LR(1)项目集
- LALR(k)分析技术与LALR(k)文法
  - LR(1)分析的状态：LR(1)项目集+同心项目集合并

LR项目与LR文法小结

- 不是SLR(0)文法，但是SLR(1)文法
  - 例：S → a | ε



- SLR(0)文法
  - S → a　S → a | 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→ α ·β, 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