

Lecture 2.5

更多文法分析理论

徐 辉

xuh@fudan.edu.cn



大纲

- 一、语言解析问题
- 二、自底向上分析
- 三、不同文法的关系

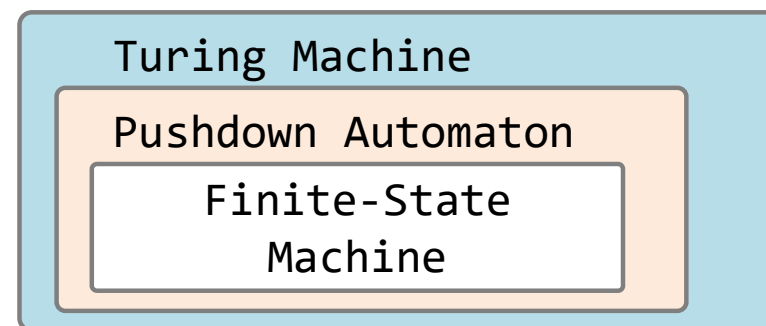
概念回顾

- 一门语言 (language) 是多个句子 (sentences) 的集合。
- 句子 (sentence) 是由终结符 (terminal symbols) 组成的序列 (sequence)。
- 字符串 (string) 是包含终结符和非终结符的序列。
 - 非终结符: X 、 Y 、 Z
 - 终结符 (标签): $\langle \text{BINOP} \rangle$ 、 $\langle \text{NUM} \rangle$
 - 字符串符号: α 、 β 、 γ
- 语法 (grammar) 包括一个开始符号 S 和多条推导规则 (productions)
 - $S \rightarrow \beta$
 - \dots

语言分析问题难度

Chomsky Hierarchy

类型	文法名称	自动机模型	生成式形式	语言示例
0 型	递归枚举	图灵机	无限制	
1 型	上下文敏感	Linear bounded TM	左侧可以多个符号 $\alpha S \rightarrow \beta$	$a^n b^n c^n$
2 型	上下文无关	下推自动机	左侧仅一个符号 $S \rightarrow \beta$	$a^n b^n$
3 型	正则	有穷自动机	右侧全部为终结符 $S \rightarrow \langle a \rangle \langle b \rangle$	a^n



类型判定

- 一般意义上，判断一个句子是否属于某个语言 $w \in L(G)$ 是不能(通过图灵机)计算的
- 根据产生式判断语言类型
- 根据句子集合判定语言类型

正则语言 VS 上下文无关语言

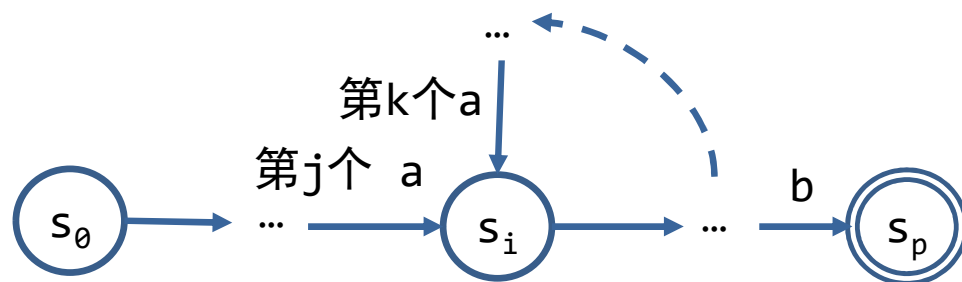
- 正则语言也可以用CFG规则形式表示：
 - 规则：
 - $X \rightarrow \gamma$
 - $\gamma \rightarrow \gamma_1$
 - ...
 - X 是非终结符
 - γ 是可能包含终结符和非终结符的字符串
- 特点：右侧的非终结符均可替换为终结符

[1]	$S \rightarrow A B$
[2]	$A \rightarrow (0?1)^*$
[3]	$B \rightarrow (1?0)^*$

 $\Longrightarrow S \rightarrow (0?1)^*|(1?0)^*$

非正则语言

- 不能用正则表达式或有穷自动机表示的语言
- $L = \{a^n b^n, n > 0\}$ 不是正则语言
 - 证明：
 - 假设DFA可识别该语言，其包含 p 个状态；
 - 假设某词素为 $a^q b^q, q > p$ 。
 - 识别该词素需要经过某状态 s_i 至少两次，分别对应第 j 和第 k 个 a ；
 - 该DFA可同时接受 $a^q b^q$ 和 $a^{q-k+j} b^q$ ，推出矛盾。

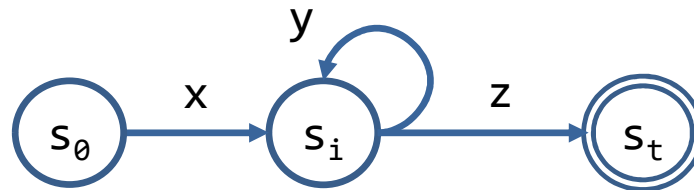


- $L = \{a^n b^n, n > 0\}$ 是上下文无关语言

$$S \rightarrow aSb$$

正则语言的泵引理 (Pumping Lemma)

- 词素数量有限的语言一定是正则语言。
- 词素数量无穷多的语言是否为正则语言？
- 某语言 $L(r)$ 是正则语言的必要条件：
 - 任意长度超过 p （泵长）的句子都可以被分解为 xyz 的形式
 - 其中 x 和 z 可为空，
 - 子句 y 被重复任意次（如 $xyyz$ ）后得到的句子仍属于该语言。



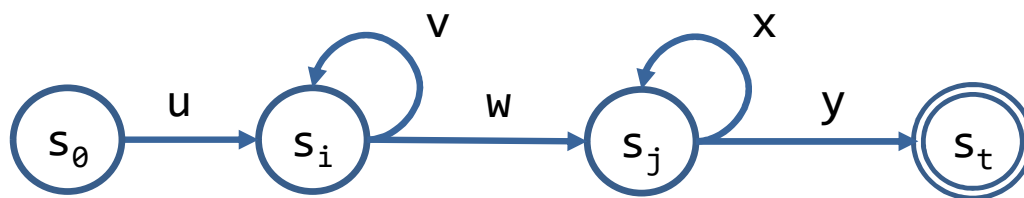
非CFG语言：上下文敏感语法

- $L = \{a^n b^n c^n, n > 0\}$ 不是CFG语言
- 如何定义？
- 上下文敏感语法定义
 - $\alpha A \beta \rightarrow \alpha \gamma \beta$
 - α 和 β 不变， A 展开

[1]	$S \rightarrow aBC$
[2]	$\quad \mid aSBC$
[3]	$CB \rightarrow BC$
[4]	$aB \rightarrow ab$
[5]	$bB \rightarrow bb$
[6]	$bC \rightarrow bc$
[7]	$cC \rightarrow cc$

非CFG语言的泵引理

- CFG语言的泵引理（必要条件）：
 - 任意长度超过 p （泵长）的句子可以被拆分为 $uvwxy$,
 - 子句 v 和 x 被重复任意次后得到的新句子（如 $uvvwxxy$ ）仍属于该语言。
- 正则属于CFG: $uv^n w \epsilon^n \epsilon$



练习：下列语言是否为正则语言？

- 集合表示

1) $L = \{a^n b^n | n \leq 100\}$

2) $L = \{a^n | n \geq 1\}$

3) $L = \{a^{2^n} | n \geq 1\}$

4) $L = \{a^p | p \text{ is prime}\}$

- Regex/CFG语法表示

1) $S \rightarrow (0? 1)^*$

2) $S \rightarrow aT | \epsilon, T \rightarrow Sb$

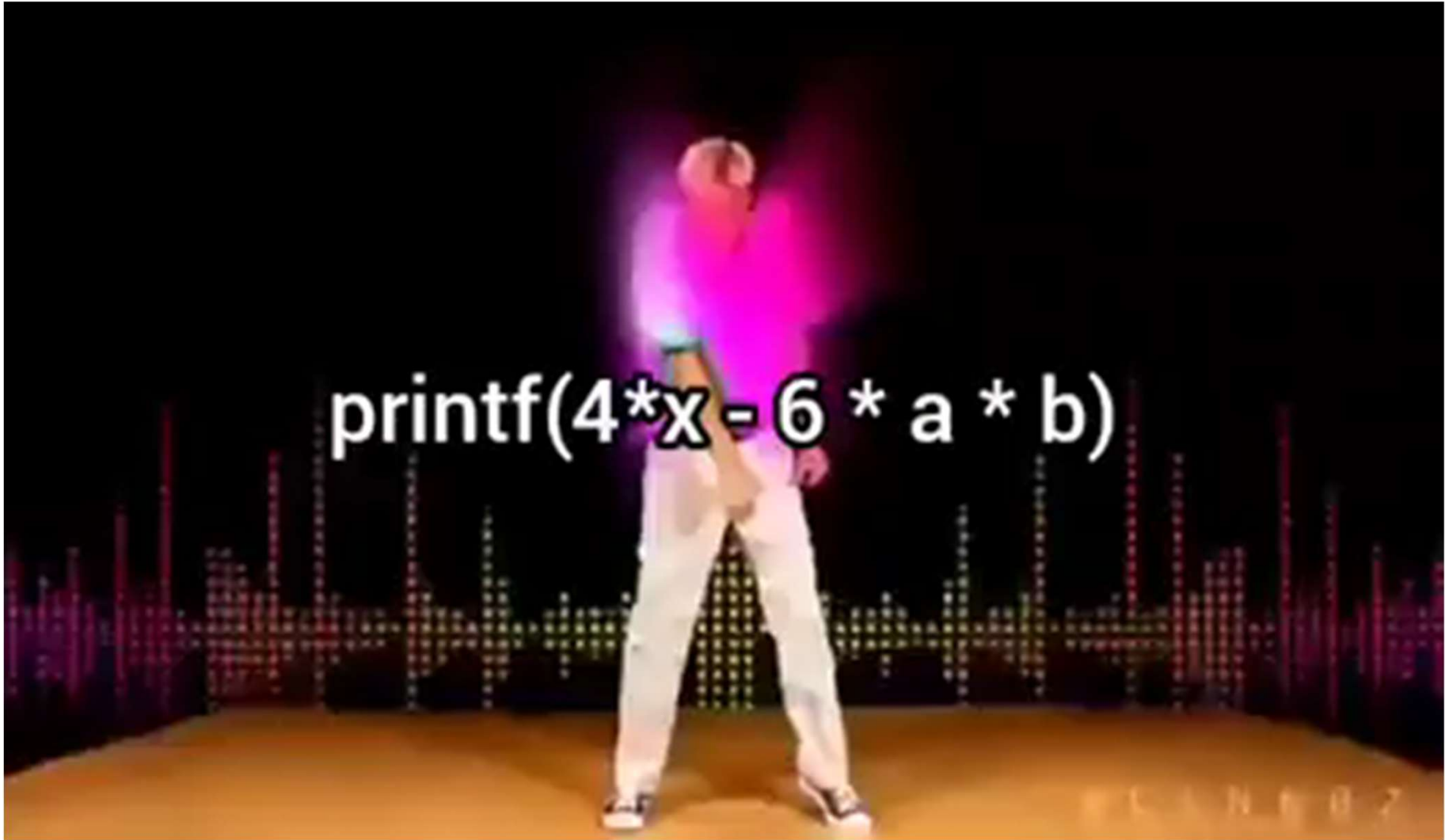
3) $S \rightarrow 0S1S | 1S0S | \epsilon$

思考

- 1) 用正则表达式可以定义所有的正则语言吗?
- 2) 有穷自动机可以解析任意正则表达式吗?
- 3) 用CFG可以定义任意正则语言吗?
- 4) 用CFG可以定义任意上下文无关语言吗?
- 5) 用下推自动机可以解析任意正则表达式吗?
- 6) 用下推自动机可以解析任意CFG吗?
- 7) 用通用图灵机可以解析任意CFG吗?
- 8) 用通用图灵机可以解析任意程序吗?

大纲

- 一、语言解析问题
- 二、自底向上分析
- 三、不同文法的关系



`printf(4*x-6*a*b)`



基本思路：基于规约的方法

- 如果在句法分析栈的上边缘找到 β 且 $A \rightarrow \beta$ ，则将其规约为 A ；
- 否则移进

[1]	$S \rightarrow \epsilon$
[2]	[S]S

	当前栈	待输入
第1步:		[]
第2步:	[]
第3步:	[S]
第4步:	[S]	
第5步:	[S]S	
	S	

形式化表示

移进: $\frac{w: \beta}{wa: \beta a} \text{shift}$

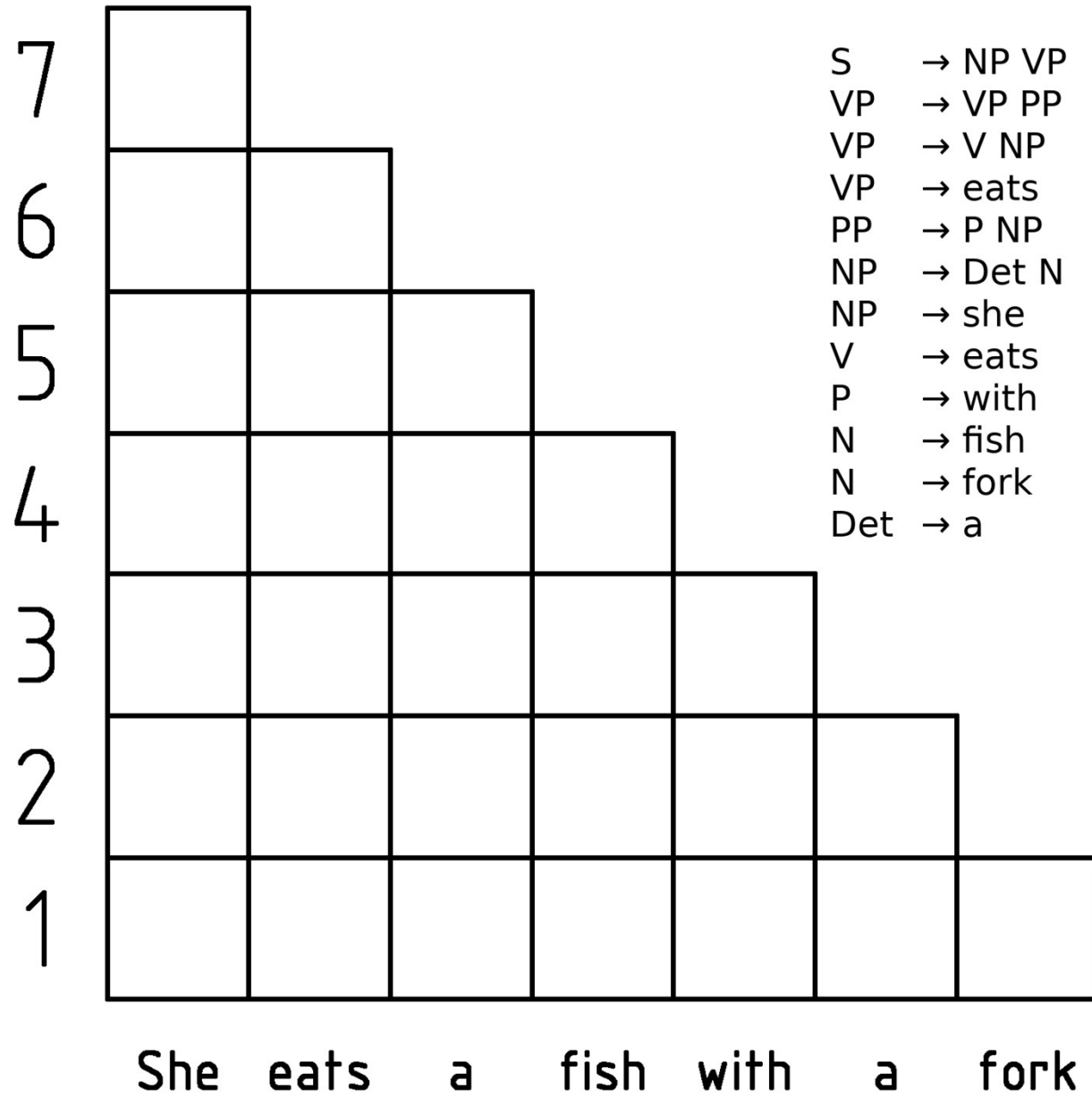
规约: $\frac{[r] X \rightarrow \alpha \quad w: \beta \alpha}{w: \beta X} \text{reduce}[r]$

$\begin{array}{c} [\\ [S \\ [S] \\ [S]S \\ S \end{array} \quad \begin{array}{c} [\\] \\] \end{array}$

$\frac{\overline{\epsilon: \epsilon}}{[: [\quad \text{(shift)}} \quad \frac{[: [S \quad \text{(reduce[2])}}{[S]: [S] \quad \text{(shift)}} \quad \frac{[S]: [S] \quad \text{(reduce[2])}}{[S]: [S]S \quad \text{(reduce[1])}} \quad [S]S: S$

[1]	$S \rightarrow [S]S$
[2]	$ \epsilon$

通用自底向上CFG分析： CYK算法



计算器语法分析

句柄状态分析(规范项)

```
[1] E → E OP1 E1
[2]   | E1
[3] E1 → E1 OP2 E2
[4]   | E2
[5] E2 → E3 OP3 E2
[6]   | E3
[7] E3 → NUM
[8]   | <LPAR> E <RPAR>
[9] NUM → <UNUM>
[10]   | <SUB> <UNUM>
[11] OP1 → <ADD>
[12]   | <SUB>
[13] OP2 → <MUL>
[14]   | <DIV>
[15] OP3 → <EXP>
```

```
[1] E → ◦ E OP1 E1
[1] E → E ◦ OP1 E1
[1] E → E OP1 ◦ E1
[1] E → E OP1 E1 ◦
[2] E → ◦ E1
[2] E → E1 ◦
[3] E1 → ◦ E1 OP2 E2
[3] E1 → E1 ◦ OP2 E2
[3] E1 → E1 OP2 ◦ E2
[3] E1 → E1 OP2 E2 ◦
...
```

语法增强

句柄状态分析(规范项)

[0] $G \rightarrow E$
[1] $E \rightarrow E \text{ OP1 } E1$
[2] | $E1$
[3] $E1 \rightarrow E1 \text{ OP2 } E2$
[4] | $E2$
[5] $E2 \rightarrow E3 \text{ OP3 } E2$
[6] | $E3$
[7] $E3 \rightarrow \text{NUM}$
[8] | $\langle \text{LPAR} \rangle E \langle \text{RPAR} \rangle$
[9] $\text{NUM} \rightarrow \langle \text{UNUM} \rangle$
[10] | $\langle \text{SUB} \rangle \langle \text{UNUM} \rangle$
[11] $\text{OP1} \rightarrow \langle \text{ADD} \rangle$
[12] | $\langle \text{SUB} \rangle$
[13] $\text{OP2} \rightarrow \langle \text{MUL} \rangle$
[14] | $\langle \text{DIV} \rangle$
[15] $\text{OP3} \rightarrow \langle \text{EXP} \rangle$

[0] $G \rightarrow \circ E$
[0] $G \rightarrow E \circ$
[1] $E \rightarrow \circ E \text{ OP1 } E1$
[1] $E \rightarrow E \circ \text{OP1 } E1$
[1] $E \rightarrow E \text{ OP1 } \circ E1$
[1] $E \rightarrow E \text{ OP1 } E1 \circ$
[2] $E \rightarrow \circ E1$
[2] $E \rightarrow E1 \circ$
[3] $E1 \rightarrow \circ E1 \text{ OP2 } E2$
[3] $E1 \rightarrow E1 \circ \text{OP2 } E2$
[3] $E1 \rightarrow E1 \text{ OP2 } \circ E2$
[3] $E1 \rightarrow E1 \text{ OP2 } E2 \circ$
...

构建LR(0)自动机：规范族

```
[0] G → E
[1] E → E OP1 E1
[2]   | E1
[3] E1 → E1 OP2 E2
[4]   | E2
[5] E2 → E3 OP3 E2
[6]   | E3
[7] E3 → NUM
[8]   | <LPAR> E <RPAR>
[9] NUM → <UNUM>
[10]   | <SUB> <UNUM>
[11] OP1 → <ADD>
[12]   | <SUB>
[13] OP2 → <MUL>
[14]   | <DIV>
[15] OP3 → <EXP>
```

S0

```
G → • E
E → • E OP1 E1
E → • E1
E1 → • E1 OP2 E2
E1 → • E2
E2 → • E3 OP3 E2
E2 → • E3
E3 → • NUM
E3 → • <LPAR> E <RPAR>
NUM → • <UNUM>
NUM → • <SUB> <UNUM>
```

Kernel items

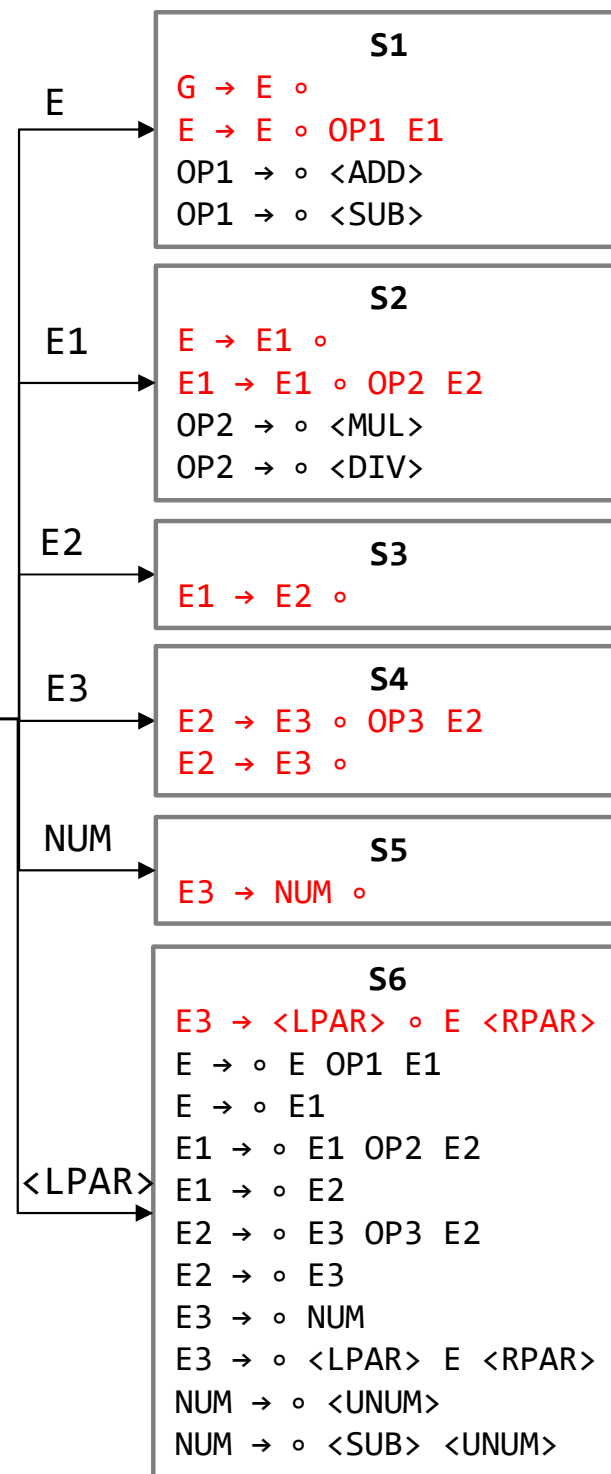
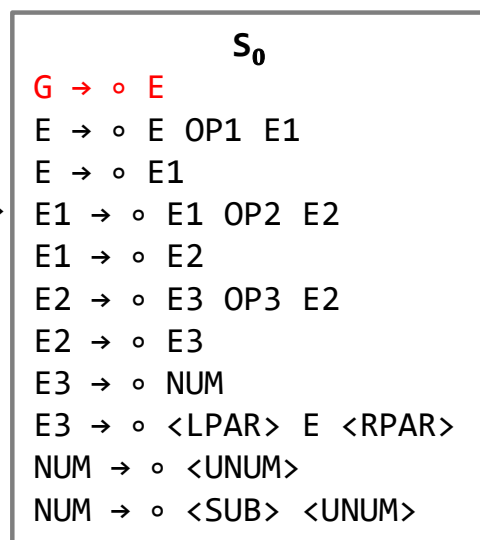
Nonkernel items

```
While (S has changed)
  for each item  $[A \rightarrow \beta \bullet C \delta, a] \in S$ 
    for each production  $[C \rightarrow \lambda] \in G$ 
      if  $[C \rightarrow \bullet \lambda] \notin S$ 
         $S \leftarrow S \cup [C \rightarrow \bullet \lambda]$ 
```

构建LR(0)自动机

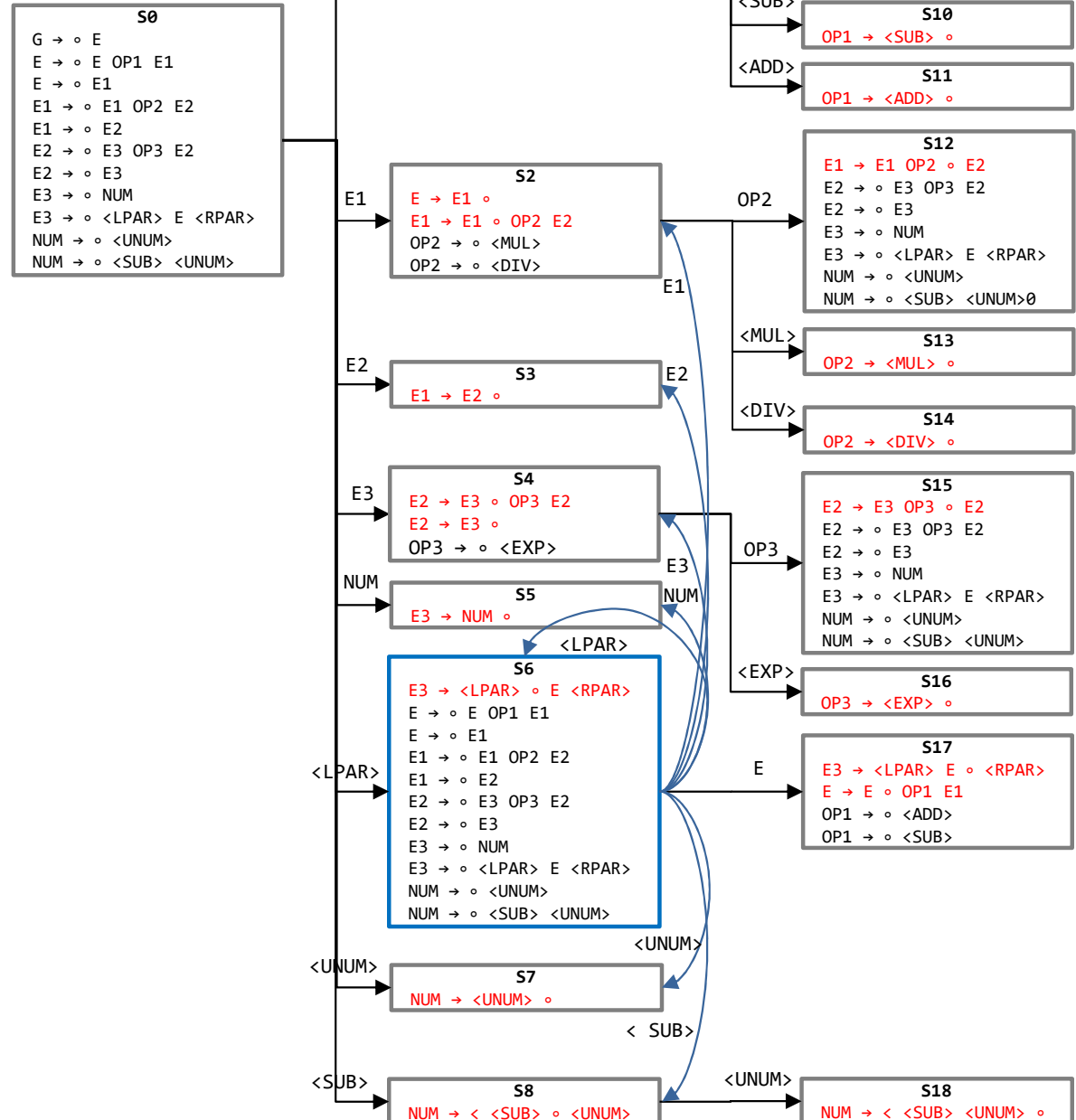
```

[0] G → E
[1] E → E OP1 E1
[2]   | E1
[3] E1 → E1 OP2 E2
[4]   | E2
[5] E2 → E3 OP3 E2
[6]   | E3
[7] E3 → NUM
[8]   | <LPAR> E <RPAR>
[9] NUM → <UNUM>
[10]   | <SUB> <UNUM>
[11] OP1 → <ADD>
[12]   | <SUB>
[13] OP2 → <MUL>
[14]   | <DIV>
[15] OP3 → <EXP>
    
```



构建LR(0)自动机

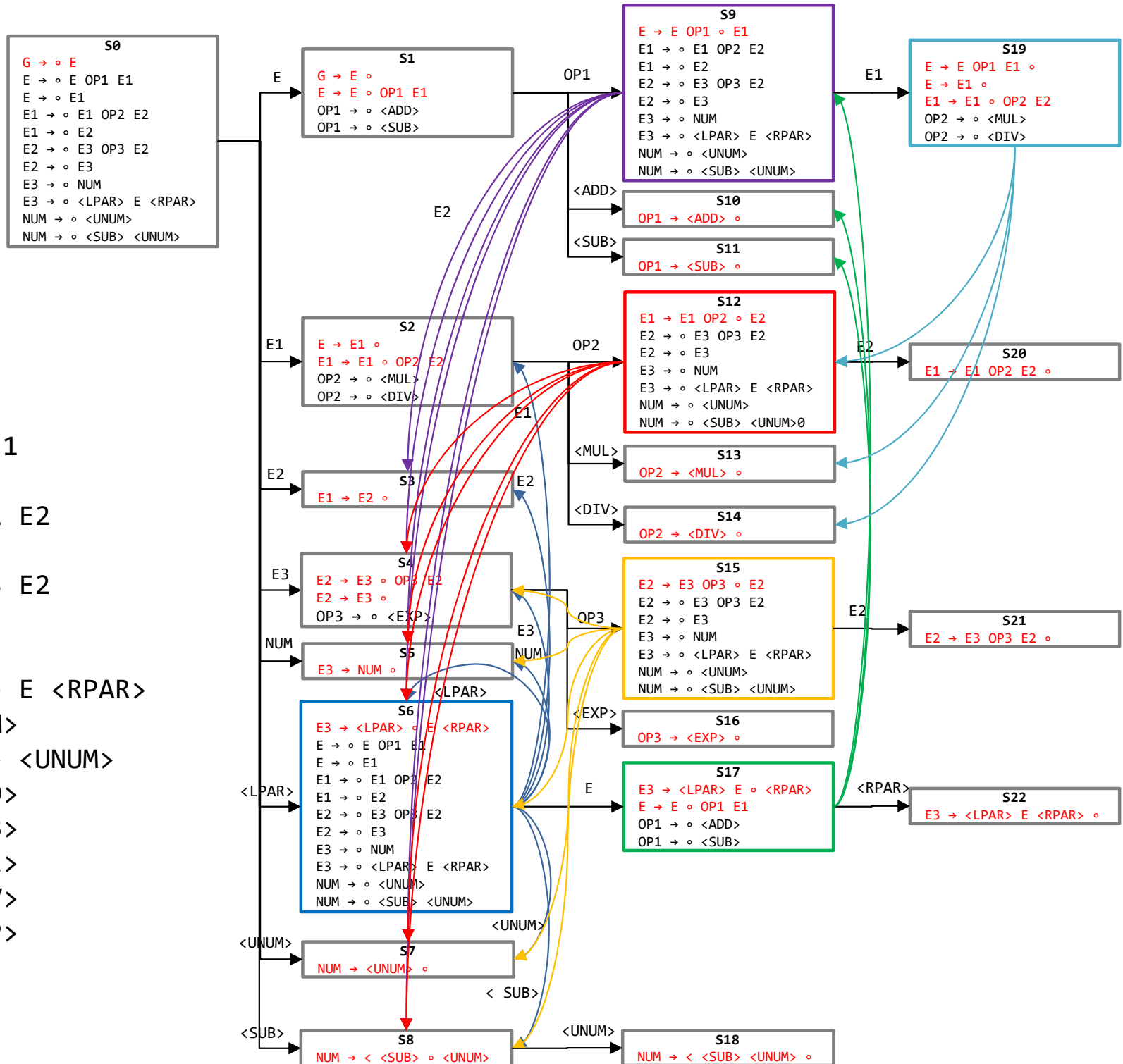
[0] $G \rightarrow E$
 [1] $E \rightarrow E \text{ OP1 } E1$
 [2] | $E1$
 [3] $E1 \rightarrow E1 \text{ OP2 } E2$
 [4] | $E2$
 [5] $E2 \rightarrow E3 \text{ OP3 } E2$
 [6] | $E3$
 [7] $E3 \rightarrow \text{NUM}$
 [8] | $\langle \text{LPAR} \rangle E \langle \text{RPAR} \rangle$
 [9] $\text{NUM} \rightarrow \langle \text{UNUM} \rangle$
 [10] | $\langle \text{SUB} \rangle \langle \text{UNUM} \rangle$
 [11] $\text{OP1} \rightarrow \langle \text{ADD} \rangle$
 [12] | $\langle \text{SUB} \rangle$
 [13] $\text{OP2} \rightarrow \langle \text{MUL} \rangle$
 [14] | $\langle \text{DIV} \rangle$
 [15] $\text{OP3} \rightarrow \langle \text{EXP} \rangle$



```

[0] G → E
[1] E → E OP1 E1
[2]   | E1
[3] E1 → E1 OP2 E2
[4]   | E2
[5] E2 → E3 OP3 E2
[6]   | E3
[7] E3 → NUM
[8]   | <LPAR> E <RPAR>
[9] NUM → <UNUM>
[10]   | <SUB> <UNUM>
[11] OP1 → <ADD>
[12]   | <SUB>
[13] OP2 → <MUL>
[14]   | <DIV>
[15] OP3 → <EXP>

```

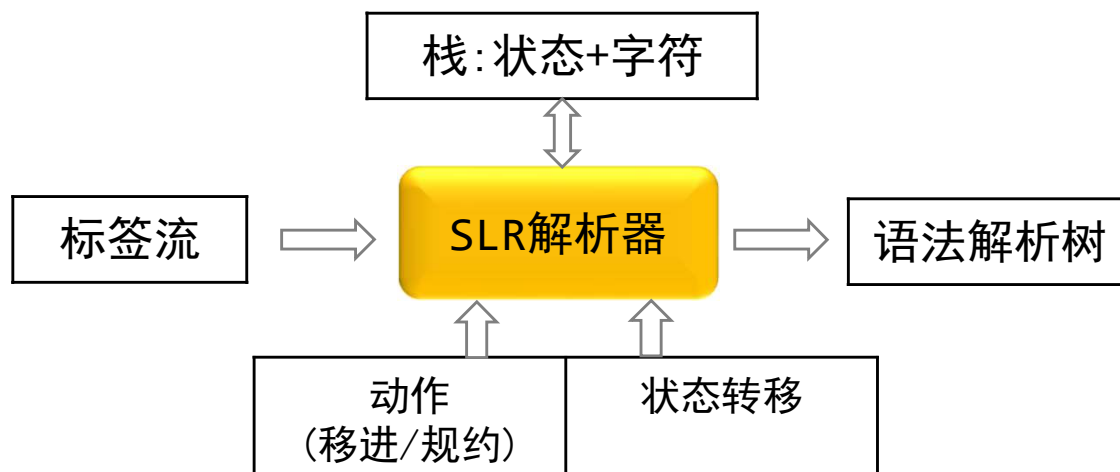


LR(0)自动机的状态转移关系表

[illegible]

构建SLR解析器

- 移进条件：如果 $A \rightarrow \alpha \circ a\beta \in S_i$ ，并且 $Goto(S_i, a) = S_j$ ，设置 $Action(S_i, a) = \text{"shift } j\text{"}$
- 规约条件：如果 $A \rightarrow \alpha \circ \in S_i$ ， $\forall a \in Follow(A)$ ，设置 $Action(S_i, a) = \text{"reduce } A \rightarrow \alpha\text{"}$



构建SLR解析表

- 移进条件：如果 $A \rightarrow \alpha \circ a\beta \in S_i$ ，并且 $Goto(S_i, a) = S_j$ ，设置 $Action(S_i, a) = \text{"shift } j\text{"}$
- 规约条件：如果 $A \rightarrow \alpha \circ \in S_i$ ， $\forall a \in Follow(A)$ ，设置 $Action(S_i, a) = \text{"reduce } A \rightarrow \alpha\text{"}$

规范族	GOTO								Action (Shift-Reduce)								
	E	E1	E2	E3	OP1	OP2	OP3	NUM	<UNUM>	<ADD>	<SUB>	<MUL>	<DIV>	<EXP>	<LP>	<RP>	<eof>
S0	S1	S2	S3	S4	∅	∅	∅	S5	S7	∅	S8	∅	∅	∅	S6	∅	∅
S1	∅	∅	∅	∅	S9	∅	∅	∅	∅	S10	S11		∅	∅	∅	∅	成功
S2	∅	∅	∅	∅	∅	S12	∅	∅	∅	R[2]	R[2]	S13	S14	∅	∅	R[2]	R[2]
S3	∅	∅	∅	∅	∅	∅	∅	∅	∅	R[4]	R[4]	R[4]	R[4]	∅	∅	R[4]	R[4]
S4	∅	∅	∅	∅	∅	∅	S15	∅	∅	R[6]	R[6]	R[6]	R[6]	S16	∅	R[6]	R[6]
S5	∅	∅	∅	∅	∅	∅	∅	∅	∅	R[7]	R[7]	R[7]	R[7]	R[7]	∅	R[7]	R[7]
S6	S17	S2	S3	S4	∅	∅	∅	S5	S7	∅	S8	∅	∅	∅	S6	∅	∅
S7	∅	∅	∅	∅	∅	∅	∅	∅	∅	R[9]	R[9]	R[9]	R[9]	R[9]	∅	R[9]	R[9]
S8	∅	∅	∅	∅	∅	∅	∅	∅	S18	∅	∅	∅	∅	∅	∅	∅	∅
S8																	
S9																	
S10																	
S11																	
S12																	
S13																	
S14																	
S15																	
...																	

SLR应用示例

Stack	Symbols	Input	Action
S0		<UNUM><MUL><UNUM><eof>	shift <UNUM>, goto S7
S0,S7	<UNUM>	<MUL><UNUM><eof>	Reduce [9], back to S0, goto S5
S0,S5	NUM	<MUL><UNUM><eof>	Reduce [7], back to S0, goto S4
S0,S4	E3	<MUL><UNUM><eof>	Reduce [6], back to S0, goto S3
S0,S3	E2	<MUL><UNUM><eof>	Reduce [4], back to S0, goto S2
S0,S2	E1	<MUL><UNUM><eof>	Shift <MUL>, goto S13
S0,S2,S13	E1 <MUL>	<UNUM><eof>	Reduce [13], back to S2, goto S12
S0,S2,S12	E1 OP2	<UNUM><eof>	Shift <UNUM>, goto S7
S0,S2,S12,S7	E1 OP2 <UNUM>	<eof>	Reduce [9], back to S12, goto S5
S0,S2,S12,S5	E1 OP2 NUM	<eof>	Reduce [7], back to S12, goto S4
S0,S2,S12,S4	E1 OP2 E3	<eof>	Reduce [6], back to S12, goto S20
S0,S2,S12,S20	E1 OP2 E2	<eof>	Reduce [4], back to S12, goto S2
S0	E1	<eof>	Reduce [3], back to s0, goto S2
S0,S2	E1	<eof>	Reduce [2], back to s0, goto S1
S0,S1	E	<eof>	成功

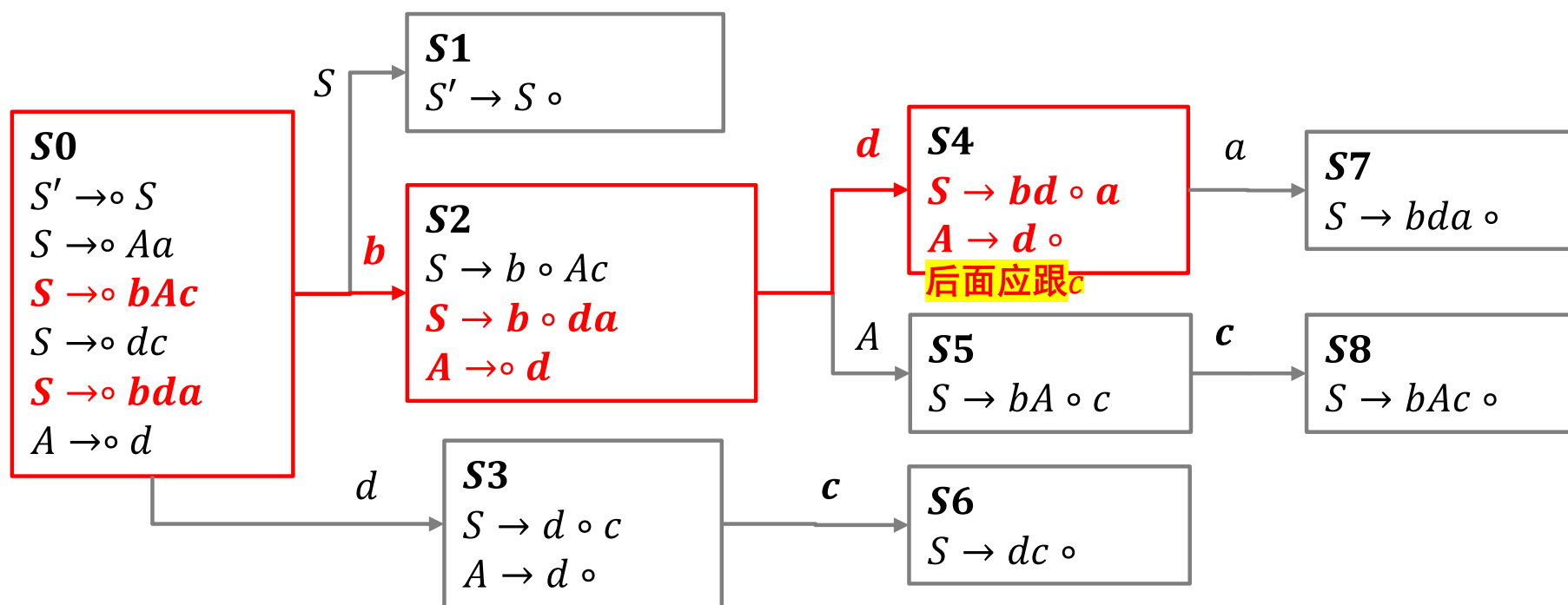
是否会存在冲突？

- 情形一：同时满足移进和规约
- 情形二：同时满足多条规约规则

二义性语法：移进-规约冲突

- 利用SLR解析表解析bda时存在移进-规约冲突
 - S_4 下一个字符为 a ，可移进
 - $a \in \text{Follow}(A)$ ，可规约

[1]	$S \rightarrow bAc$
[2]	$ bda$
[3]	$ Aa$
[4]	$A \rightarrow d$



什么情况下容易出现移进-规约冲突？

[1]	$S \rightarrow bAc$
[2]	bda
[3]	Aa
[4]	$A \rightarrow d$

[1]	$S \rightarrow \beta_1 X \beta_2$
[2]	$\beta_1 \beta_3 \beta_4$
[3]	$X \rightarrow \beta_3$

- 同一非终结符的两条规则：
 - 拥有共同的起始字符串 β_1 ；
 - β_1 后面分别为非终结符 $X\beta_2$ 和字符串 $\beta_3\beta_4$ ；
 - 存在规则 $X \rightarrow \beta_3$ 。
 - $Follow(X) \cap First(\beta_4) \neq \emptyset$
- 或存在两种推导满足上述条件，如：

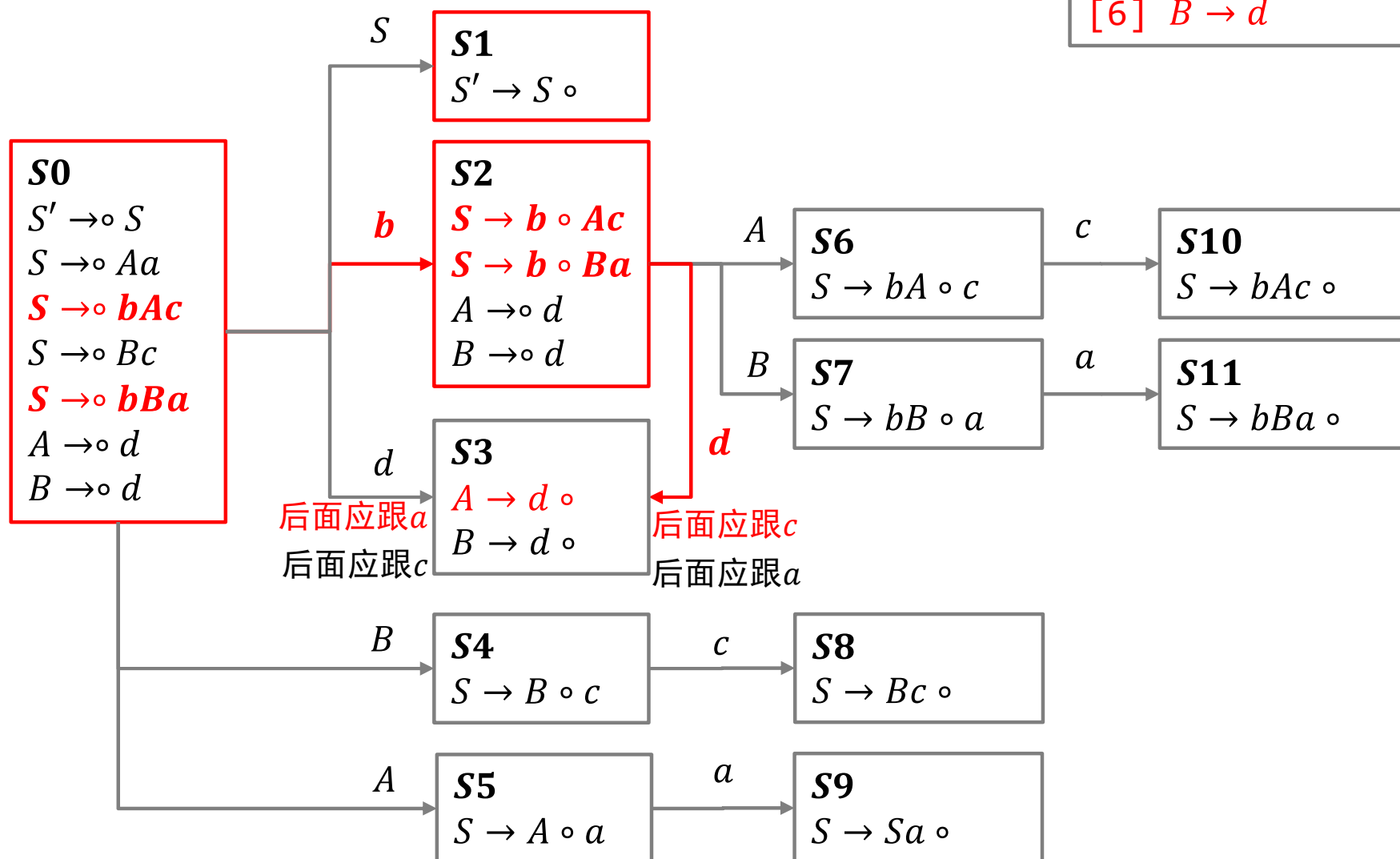
[1]	$S \rightarrow \beta_0 \beta_1 X \beta_3$
[2]	$S \rightarrow \beta_0 Y$
[3]	$Y \rightarrow \beta_1 \beta_2 \beta_3$
[4]	$X \rightarrow \beta_2$

二义性语法：规约-规约冲突

- 解析bda时存在规约($A \rightarrow d$)-规约($B \rightarrow d$)冲突
 - $a \in \text{Follow}(A)$ 且 $a \in \text{Follow}(B)$
- 解析da、dc等其它句子时存在同样的问题

```

[1]  $S \rightarrow Aa$ 
[2]    $|bAc$ 
[3]    $|Bc$ 
[4]    $|bBa$ 
[5]  $A \rightarrow d$ 
[6]  $B \rightarrow d$ 
    
```



什么情况下容易出现规约-规约冲突?

[1]	$S \rightarrow Aa$
[2]	$\quad bAc$
[3]	$\quad Bc$
[4]	$\quad bBa$
[5]	$A \rightarrow d$
[6]	$B \rightarrow d$

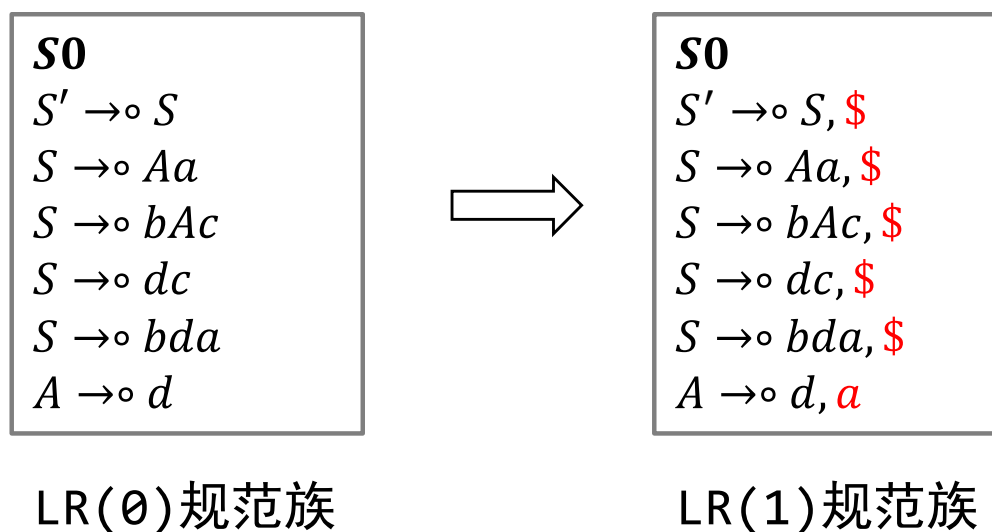
[1]	$S \rightarrow \beta_1 X \beta_2$
[2]	$\quad \beta_1 Y \beta_3$
[3]	$X \rightarrow \beta_4$
[4]	$Y \rightarrow \beta_4$

- 同一非终结符的两条规则：
 - 拥有共同的起始字符串 β_1 ；
 - β_1 后面分别为非终结符 $X\beta_2$ 和字符串 $Y\beta_3$ ；
 - 存在规则 $X \rightarrow \beta_4$ 和 $Y \rightarrow \beta_4$ 。
 - $Follow(X) \cap Follow(Y) \neq \emptyset$
- 或存在两种推导满足上述条件，如

[1]	$S \rightarrow \beta_0 X$
[2]	$\quad \beta_0 \beta_1 Y \beta_3$
[3]	$X \rightarrow \beta_1 Z \beta_3$
[4]	$Y \rightarrow \beta_2$
[5]	$Z \rightarrow \beta_2$

如果存在冲突怎么办？

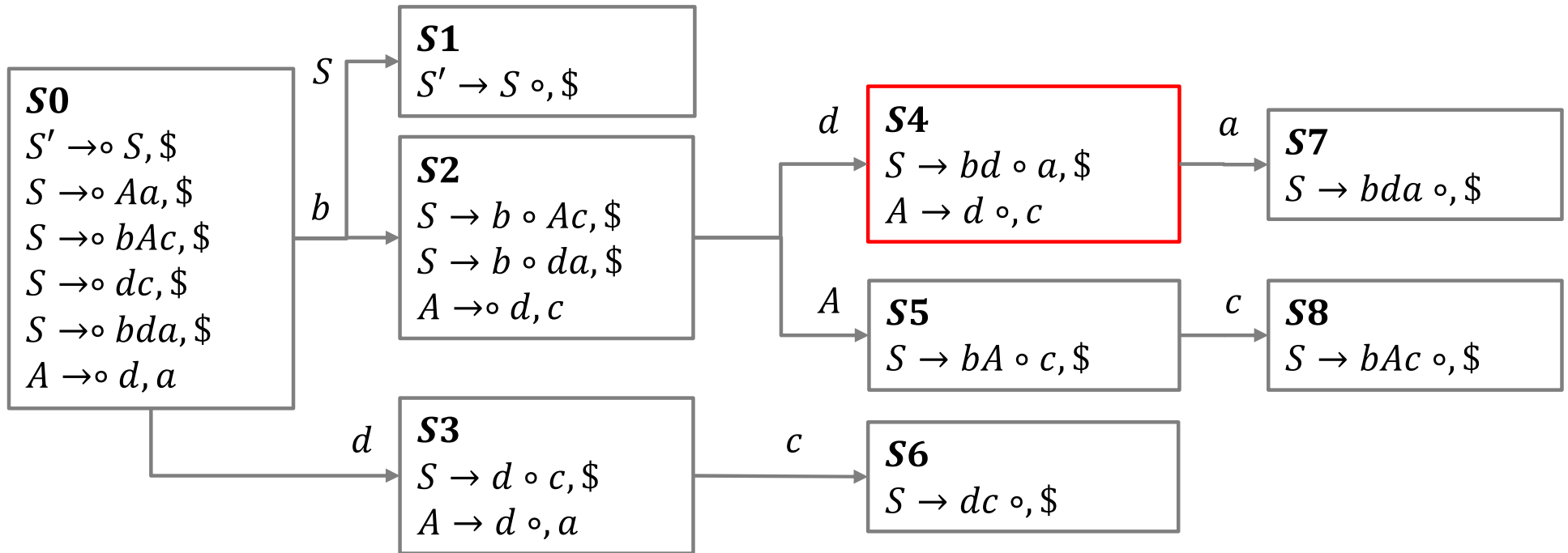
- 进一步细化SLR解析表
- LR(1)规范项/族：记录具体的Follow字符信息



LR(1)自动机构造

SLR存在移进-规约冲突的例子

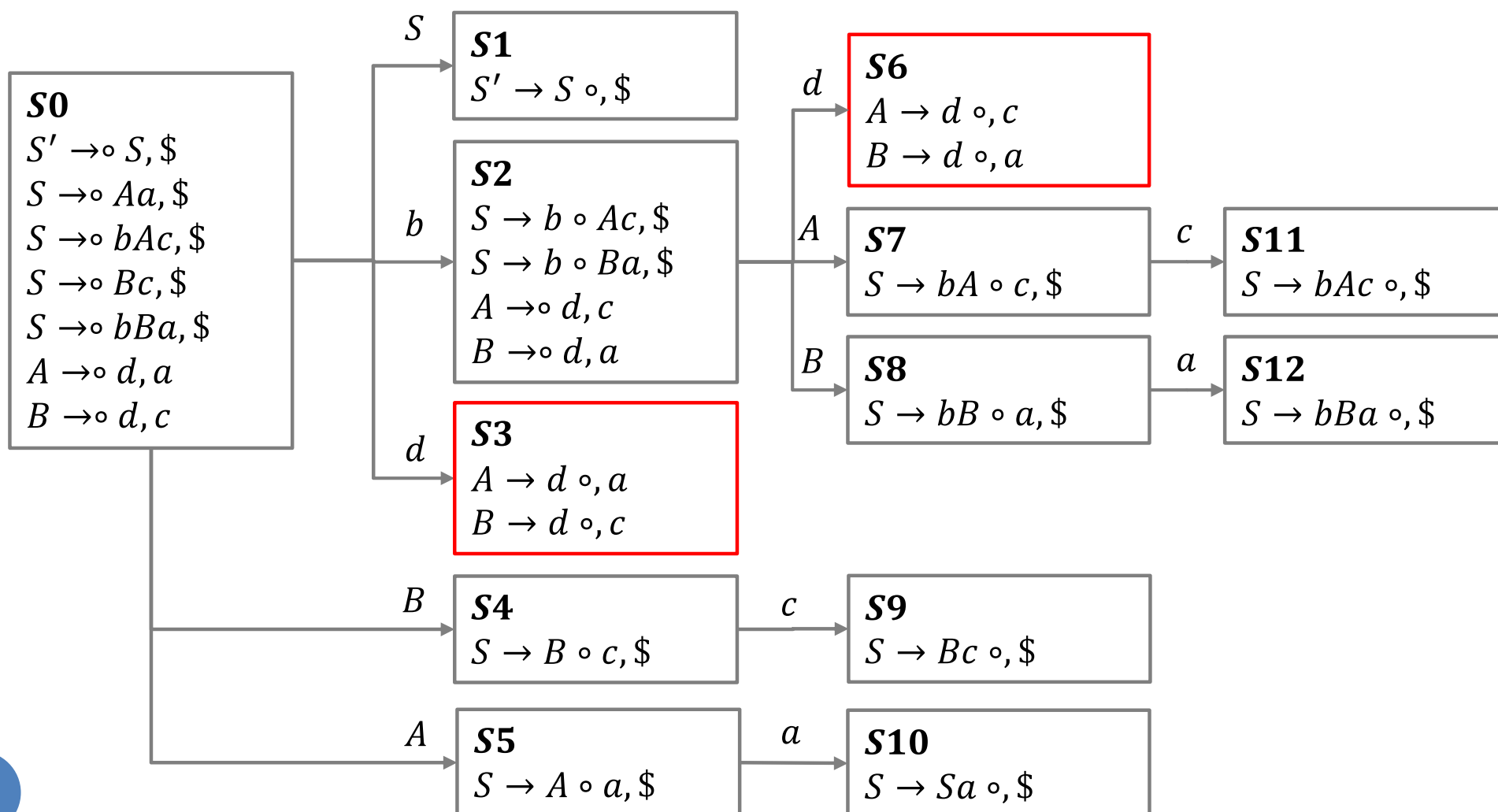
[0] $S' \rightarrow S$
[1] $S \rightarrow Aa$
[2] $\quad |bAc$
[3] $\quad |dc$
[4] $\quad |bda$
[5] $A \rightarrow d$



LR(1)自动机构造

SLR存在规约-规约冲突的例子

[1] $S \rightarrow Aa$
 [2] $\quad |bAc$
 [3] $\quad |Bc$
 [4] $\quad |bBa$
 [5] $A \rightarrow d$
 [6] $B \rightarrow d$

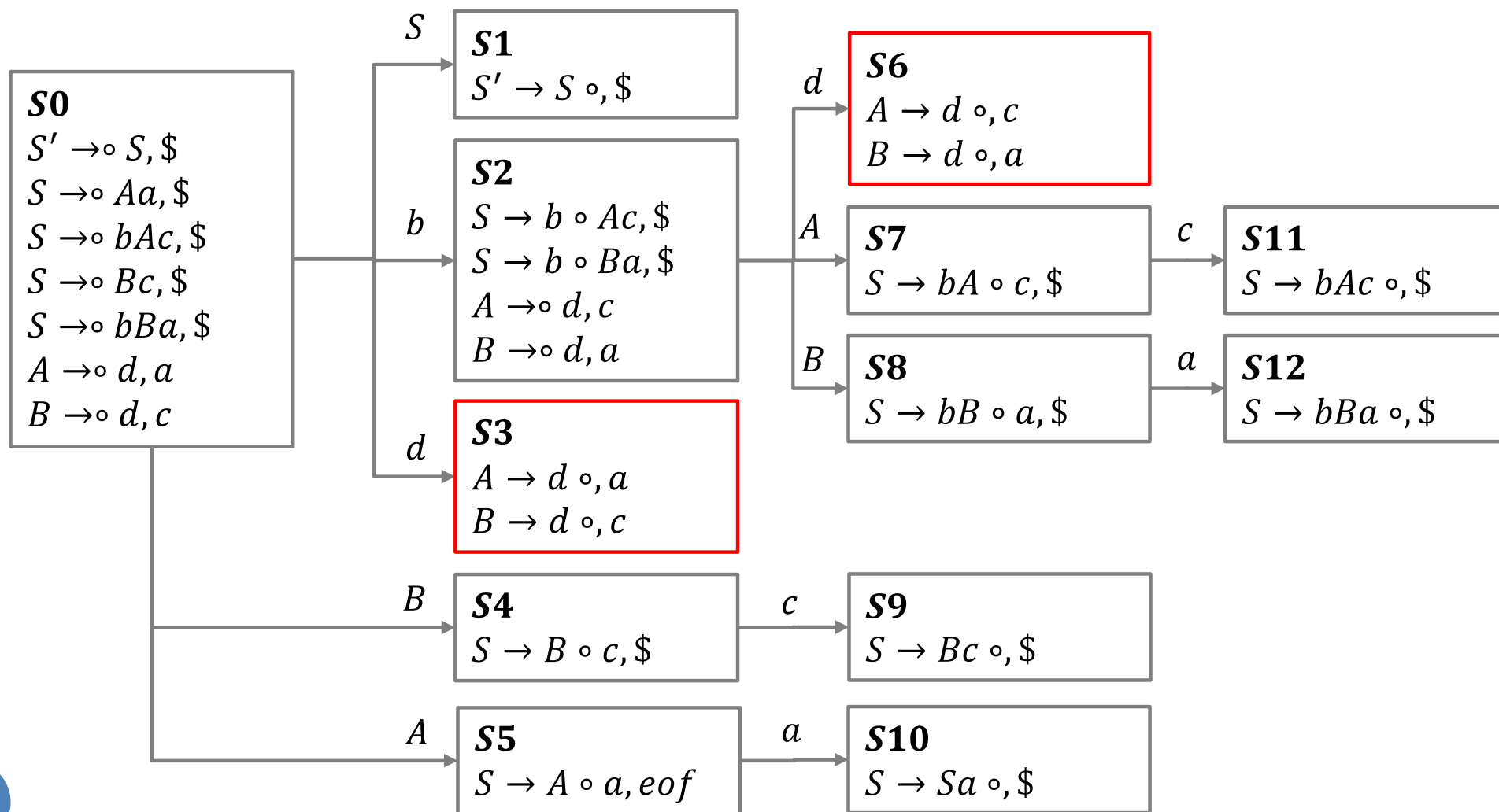


思考：SLR和LR如何选取移进、规约操作？

- SLR维护当前的栈顶句柄信息
 - 通过构造LR(0)自动机和下一个字符判断是否可以移进
 - 需要规约时根据Follow判断是否可行
- 经典LR(1)思路类似：
 - 自动机构造时考虑Follow信息
 - 但LR(1)的规范项和规范族数量很多
- 折中思路：LALR (Lookahead LR)
 - 自动机构造时考虑Follow信息
 - 同时精简规范族

LALR构造思路

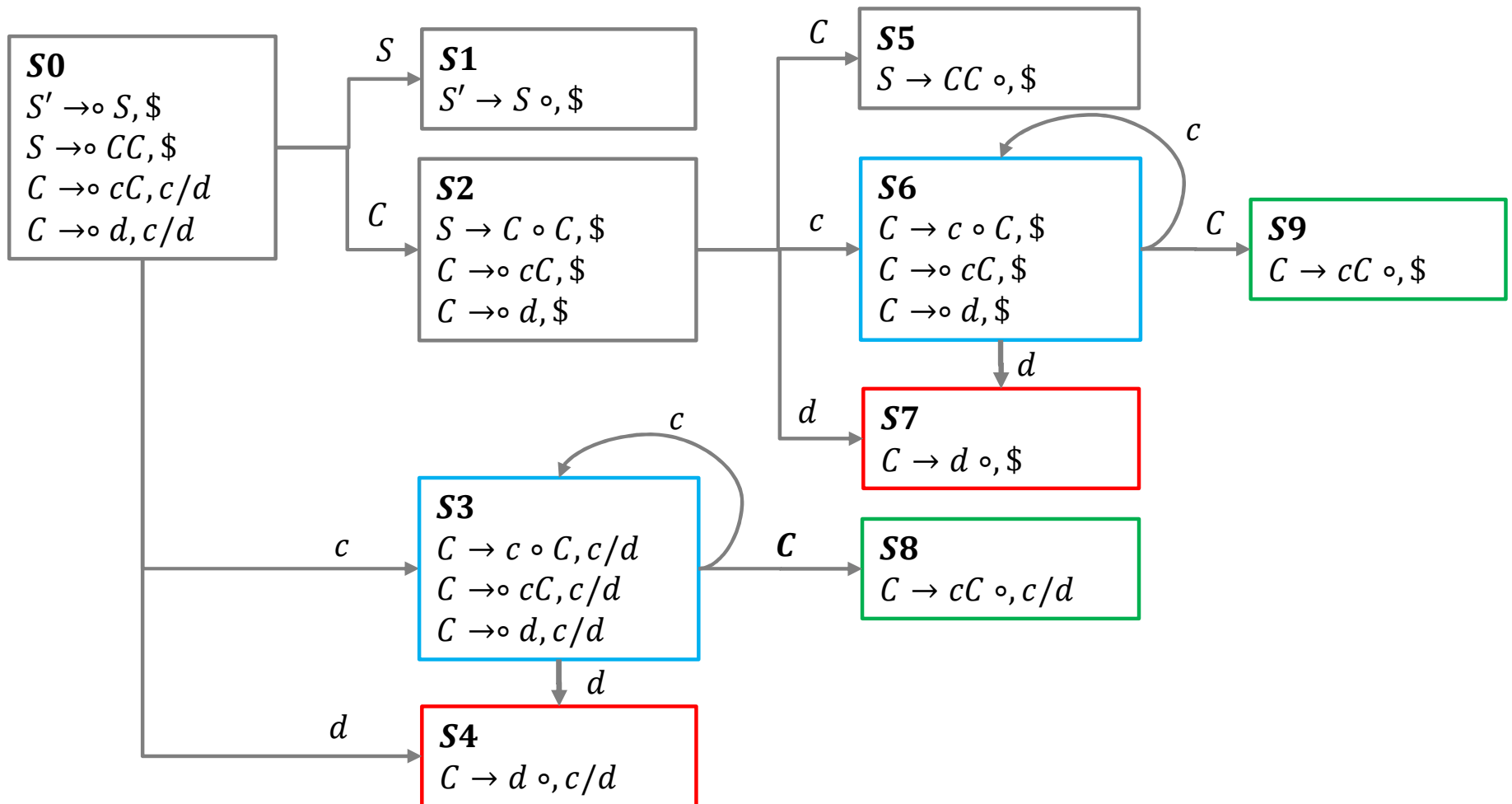
- 合并句柄状态完全相同的状态集
- 下面LR(1)自动机 S_2 和 S_5 可以合并，但合并后存在规约-规约冲突
 - 该语法不是LALR



LALR语法举例

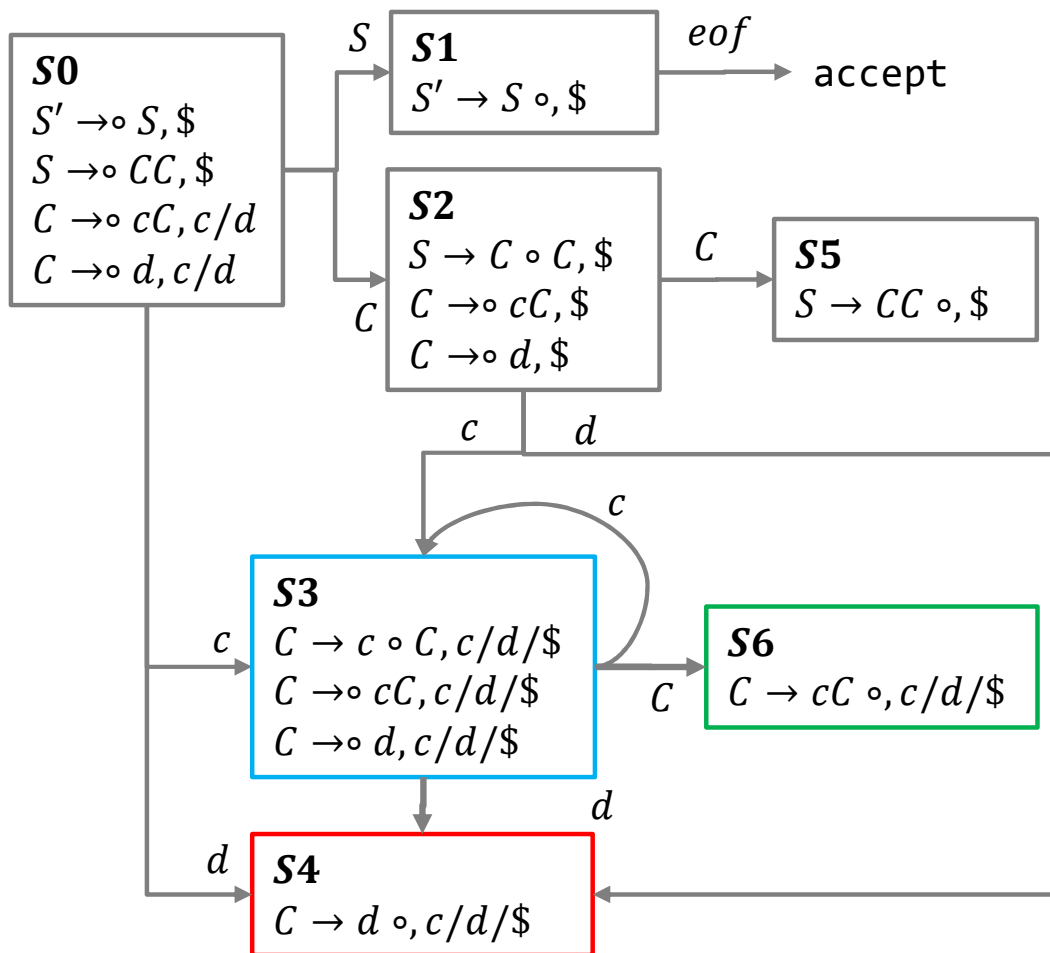
[1] $S' \rightarrow S$
[2] $S \rightarrow CC$
[3] $C \rightarrow cC$
[4] $\quad \quad | d$

- 可以合并的规范族
 - S3和S6、S4和S7、S8和S9;
 - Follow项取并集。



LALR解析表

- [1] $S' \rightarrow S$
- [2] $S \rightarrow CC$
- [3] $C \rightarrow cC$
- [4] $\quad \quad d$



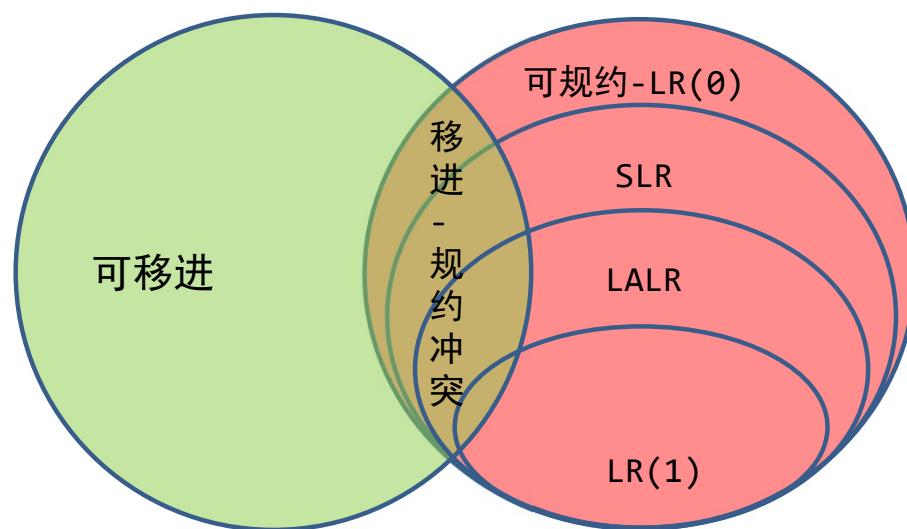
规范族				Goto	
	c	d	eof	S	C
S0	shift S ₃	shift S ₄		S ₁	S ₂
S1			accept		
S2	shift S ₃				S ₅
S3	shift S ₃	shift S ₄			S ₆
S4	reduce [4]	reduce [4]	reduce [4]		
S5			reduce [2]		
S6	reduce [3]	reduce [3]	reduce [3]		

大纲

- 一、语言解析问题
- 二、自底向上分析
- 三、不同文法的关系

几种语法的关系

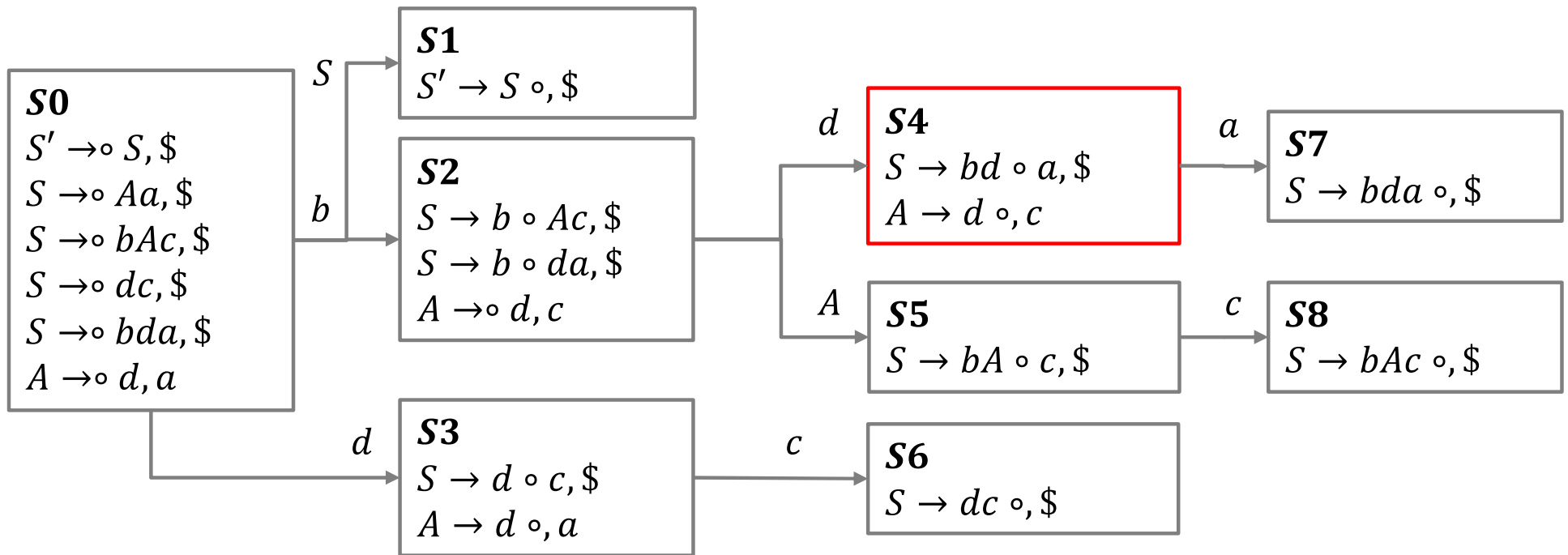
- 语法表达能力: $LR(1) > LALR(1) > SLR$
 - 规约条件严苛: $LR(1) > LALR(1) > SLR$
 - 移进条件同 $LR(0)$?



举例：LALR，非SLR语法

- 构造SLR解析表则解析bda时存在移进-规约冲突
- LALR解析方法可以避免冲突

[1] $S \rightarrow bAc$
[2] $|bda$
[3] $|Aa$
[4] $A \rightarrow d$



LL(1) vs LR(1)

- LL(1)语法一定是LR(1)吗？为什么？
- LL(1)不一定是SLR(1)，反例？
- LL(1)不一定是LALR(1)，反例？

举例说明：LL(1)非SLR

- [1] $S \rightarrow AaAb$
 [2] $|BbBa$
 [3] $A \rightarrow \epsilon$
 [4] $B \rightarrow \epsilon$

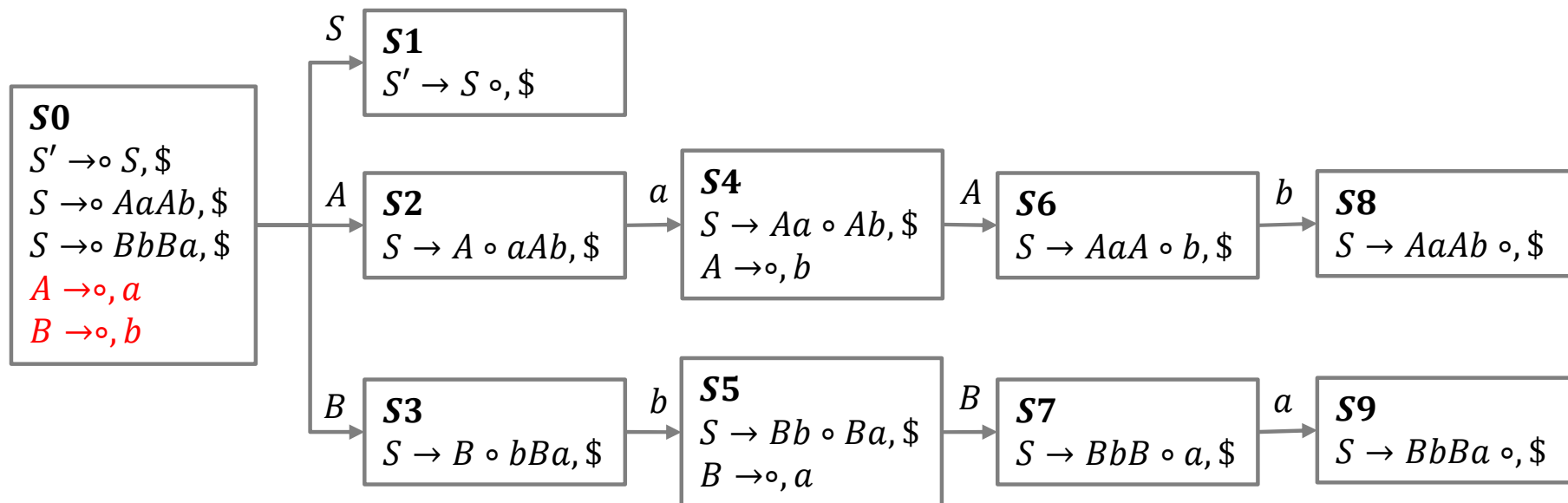
是LL(1)

- $First^+(S \rightarrow AaAb) = \{a\}$
- $First^+(S \rightarrow BbBa) = \{b\}$

不是SLR(1)

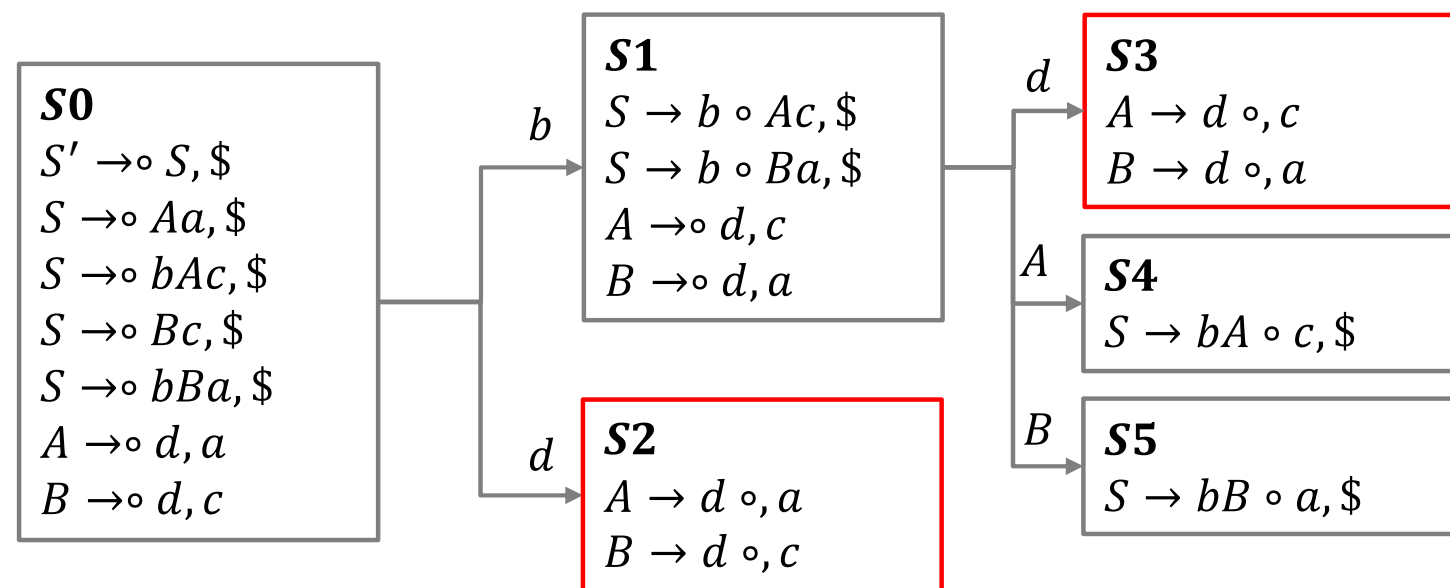
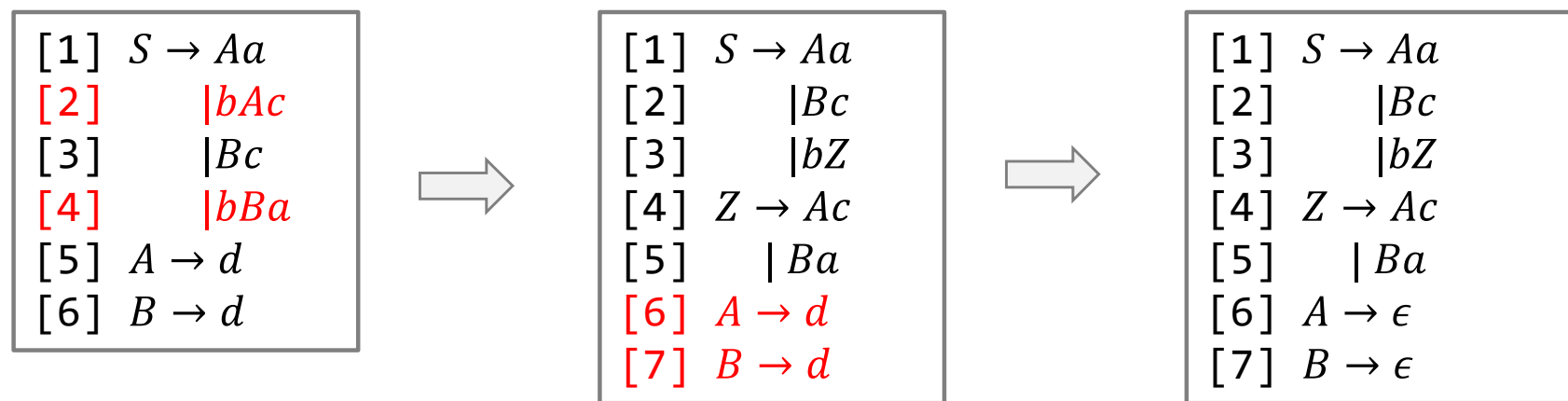
- $Follow(A) = Follow(B) = \{a, b\}$
- $Action(S_0, a) = reduce[3] \text{ 或 } reduce[4]$

是LR(1)



举例说明：LL(1)非LALR

- 基于前面非LL(1)、非LALR的例子改写。



练习

- 下面的语法是否是LL(1)? 是否是SLR(1)

[1]	$S \rightarrow SA$
[2]	$\quad A$
[3]	$A \rightarrow a$

练习

- 下列语法是否是LR(1) ?

```
[1] REGEX → UNION
[2]         | CONCAT
[3] UNION → REGEX <OR> CONCAT
[4] CONCAT → CONCAT CLOSURE
[5]         | CLOSURE
[6] CLOSURE → ITEM <STAR>
[7]         | ITEM
[8] ITEM → <LPAR>REGEX<RPAR>
[9]      | <CHAR>
```

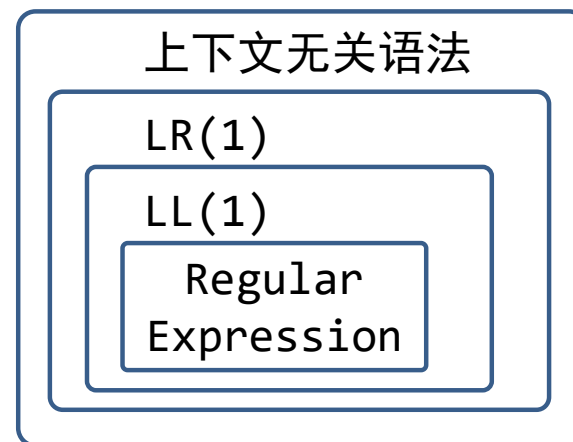
```
[1] REGEX → CONCAT REGEX1
[2] REGEX1 → <OR> CONCAT REGEX1
[3]         |  $\epsilon$ 
[4] CONCAT → CLOSURE CONCAT1
[5] CONCAT1 → CLOSURE CONCAT1
[6]         |  $\epsilon$ 
[7] CLOSURE → ITEM FOLLOW
[8] FOLLOW → <STAR>
[9]         |  $\epsilon$ 
[10] ITEM → <LPAR>REGEX<RPAR>
[11]      | <CHAR>
```


如果LR（1）不够用怎么办？

- LR(1)解析表存在冲突
- GLR (Generalized LR)
 - 遇到冲突时分别尝试两种解析指令
 - 复制栈状态，维护解析搜索树

小结

- 编译器的任务：找到语法树推导
 - 自顶向下 (top-down parser)
 - 自底向上 (bottom-up parser)
- 语法难度：CFG > LR(1) > LL(1) > RE
 - 任意CFG需要花费更多时间进行语法分析
 - CYK/Earley算法复杂度 $O(n^3)$
 - LL(1)是LR(1)的一个子集
 - Left-to-Right, Leftmost
 - 前瞻单词1个
 - 适合自顶向下分析
 - LR(1)是无歧义CFG的一个子集
 - Left-to-Right, Rightmost
 - 前瞻单词1个
 - 适合自底向上分析



总结

- 语言解析问题
 - Chomsky Hierarchy
- 自底向上分析
 - SLR、LALR、LR(1)语言
 - LR(1)语言解析器构造
 - 通用算法：GLR算法、CYK算法
- 不同语法之间的关系

