编译原理

第五章　语法分析——自下而上分析

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法
  - LR 分析器的工作原理
  - LR(0) 项目集规范族的构造

5.3.2 LR(0) 项目集族和 LR(0) 分析表的构造

- 假定α是文法 G 的一个句子，我们称序列

  α_n ， α_{n-1} ， … ， α_0

  是的一个规范归约，如果此序列满足:

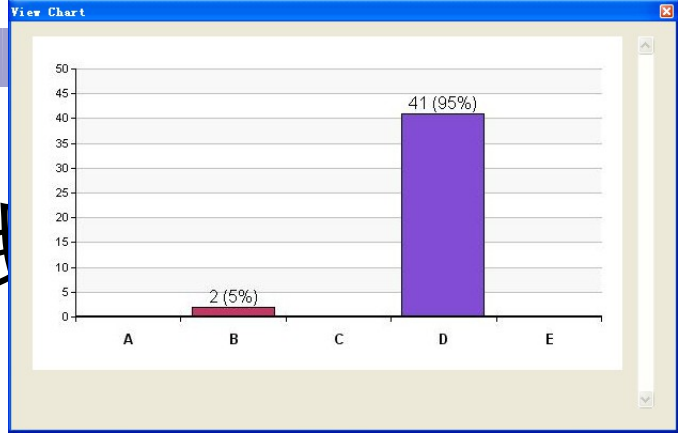
  1 $\alpha_n = \alpha$

  2 α_0 为文法的开始符号，即$\alpha_0 = S$

  3 对任何 i ， $0 \le i \le n$ ， α_{i-1} 是从α_i 经把句柄替换成为相应产生式左部符号而得到的。

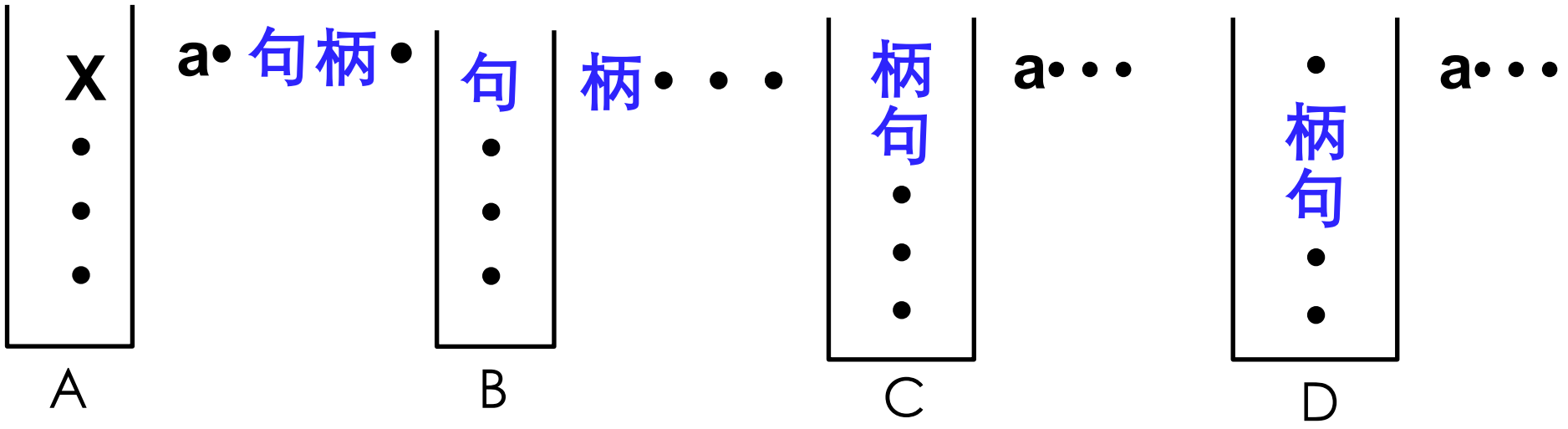
规范归约过程中栈内符号串

- 规范归约过程中
  - 栈内的符号串和扫描剩下的输入符号串构成了一个规范句型



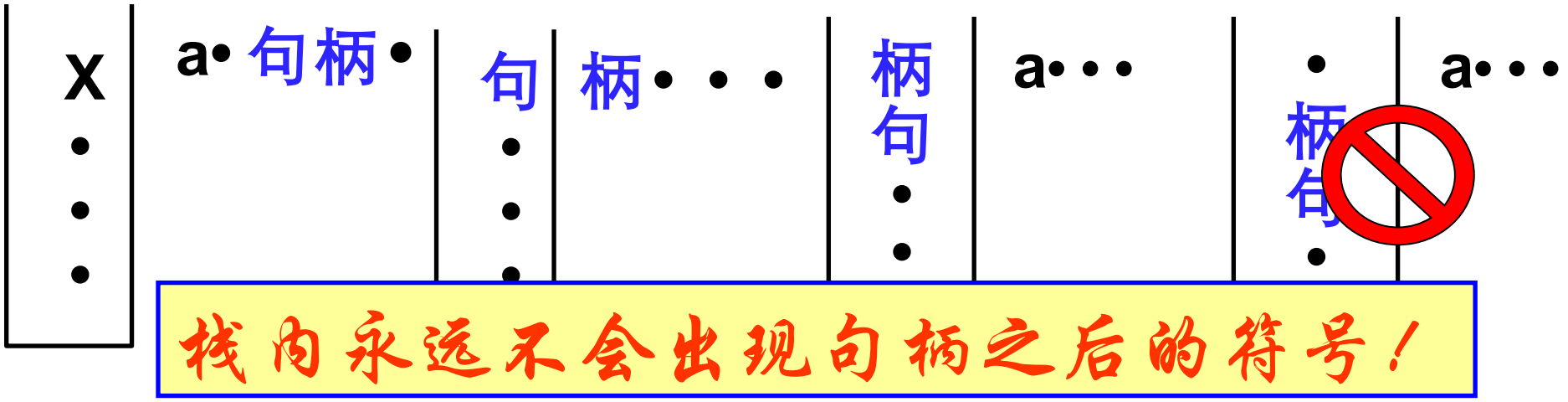
讨论：规范归约过程中栈

- 对于句子，在规范归约过程中，栈内的符号串和扫描剩下的输入符号串构成了一个规范句型，下面哪种格局不会出现：



规范归约过程中栈内符号串

- 规范归约过程中
  - 栈内的符号串和扫描剩下的输入符号串构成了一个规范句型
  - 栈内的如果出现句柄，句柄一定在栈的顶部



字的前缀、活前缀

- 字的前缀：是指字的任意首部，如字 *abc* 的前缀有ε， *a* ， *ab* ， *abc*
- 活前缀：是指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。即，对于规范句型αβδ，β为句柄，如果αβ $=u_1u_2\cdots u_r$ ，则符号串 $u_1u_2\cdots u_i(1\le i\le r)$ 是αβδ的活前缀。（δ必为终结符串）

指导思想——目标驱动

- 踢足球

“如果你不知道怎样踢球，就往球门方向踢”

——施拉普纳

- LR分析

“如果你不知道怎样分析，就保证栈中总是活前缀”

- 哪些字符串是活前缀？
- 能不能构造一个DFA来识别活前缀？

回答：对于一个文法G，可以构造一个DFA，它能识别G的所有活前缀。

字的前缀、活前缀

- 字的前缀：是指字的任意首部，如字 *abc* 的前缀有ε， *a* ， *ab* ， *abc*
- 活前缀：是指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。即，对于规范句型αβδ，β为句柄，如果αβ $=u_1u_2\cdots u_r$，则符号串 $u_1u_2\cdots u_i(1\le i\le r)$ 是αβδ的活前缀。（δ必为终结符串）
- 规范归约过程中，保证分析栈中总是活前缀，就说明分析采取的移进 / 归约动作是正确的

- 文法 G 的每个产生式的右部添加一个圆点称为 G 的 LR(0) 项目
- 如 :$A \rightarrow XYZ$ 有四个项目 :

$A \rightarrow \bullet XYZ$　$A \rightarrow X \bullet YZ$　$A \rightarrow XY \bullet Z$　$A \rightarrow XYZ \bullet$

  - $A \rightarrow \alpha \bullet$ 称为 " 归约项目 "
  - 归约项目 $S' \rightarrow \alpha \bullet$ 称为 " 接受项目 "
  - $A \rightarrow \alpha \bullet a\beta$ $(a \in V_T)$ 称为 " 移进项目 "
  - $A \rightarrow \alpha \bullet B\beta$ $(B \in V_N)$ 称为 " 待约项目 ".
- 项目表示我们在分析过程中看到了产生式多大部分

- 文法 G(S′)

  S′→E
  E→aA|bB
  A→cA|d
  B→cB|d

- 该文法的项目有:

1. S′→ · E 2. S′→E · 3. E→ · aA
4. E→a · A 5. E→aA · 6. A→ · cA
7. A→c · A 8. A→cA · 9. A→ · d
10. A→d · 11. E→ · bB 12. E→b · B
13. E→bB · 14. B→ · cB 15. B→c · B
16. B→cB · 17. B→ · d

- 构造识别文法所有活前缀的 NFA 方法

1. 若状态 i 为 $X \to X_1 \cdots X_{i-1} \bullet X_i \cdots X_n$ ，

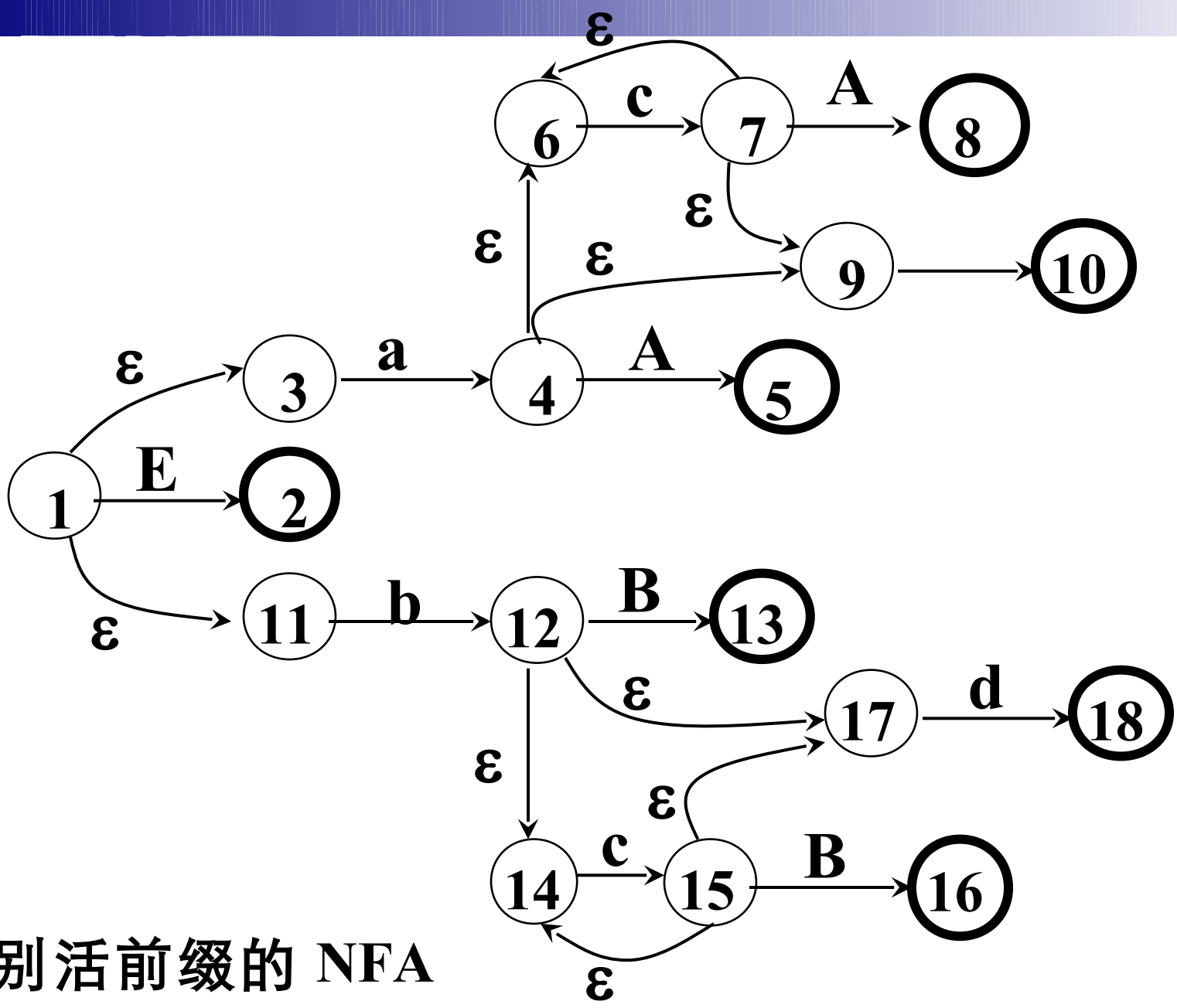
   状态 j 为 $X \to X_1 \cdots X_{i-1} X_i \bullet X_{i+1} \cdots X_n$ ，

则从状态 i 画一条标志为 X_i 的有向边到状态 j ；

2. 若状态 i 为 $X \to \alpha \bullet A\beta$ ， A 为非终结符，

   则从状态 i 画一条ε边到所有状态 $A \to \bullet\gamma$

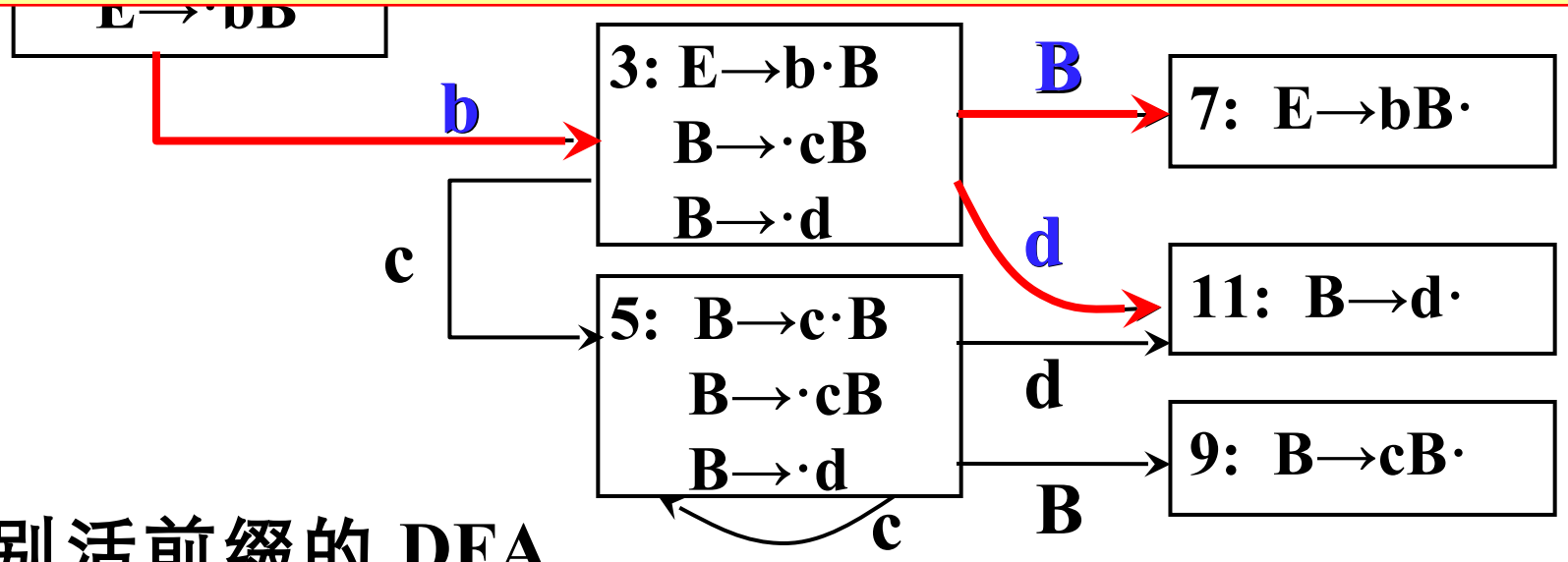
- 把识别文法所有活前缀的 NFA 确定化



识别活前缀的 NFA

对比 DFA ：

```
curState = 初态
GetChar();
while( stateTrans[curState][ch] 有定义 ){
    //存在后继状态，读入、拼接
    Concat();
    //进入下一状态，读入下一字符
   curState= stateTrans[curState][ch];
    if cur_state 是终态 then 返回 strToken 中的单词
    GetChar( );
}
```



识别活前缀的 DFA

LR(0) 项目集规范族

- 构成识别一个文法活前缀的 DFA 的项目集 (状态) 的全体称为文法的 LR(0) 项目集规范族。

有效项目

- 我们

  效的

  - 在任

X_m 的

的初

项目

- 结论：若项目 A→α•Bβ 对活前缀η =δα 是有效的且 B →γ 是一个产生式，则项目 B → •γ 对η =δα 也是有效的。

$$S' \underset{R}{\overset{*}{\Longrightarrow}} \delta A\omega \underset{R}{\Rightarrow} \delta\alpha B\beta\omega$$

设 $\beta\omega \underset{R}{\overset{*}{\Rightarrow}} \varphi\omega$ ，那么

$$S \underset{R}{\overset{*}{\Longrightarrow}} \delta A\omega \underset{R}{\Rightarrow} \delta\alpha B\beta\omega \underset{R}{\overset{*}{\Rightarrow}} \delta\alpha B\varphi\omega \underset{R}{\Rightarrow} \delta\alpha\gamma\varphi\omega$$

所以， B → •γ 对 η =δα 也是有效的。

项目 $A\rightarrow \beta_1 \bullet \beta_2$ 对活前缀$\alpha\beta_1$是有效的，其条件是存在规范推

- 文法 G(S′)

  S′→E

- 考虑

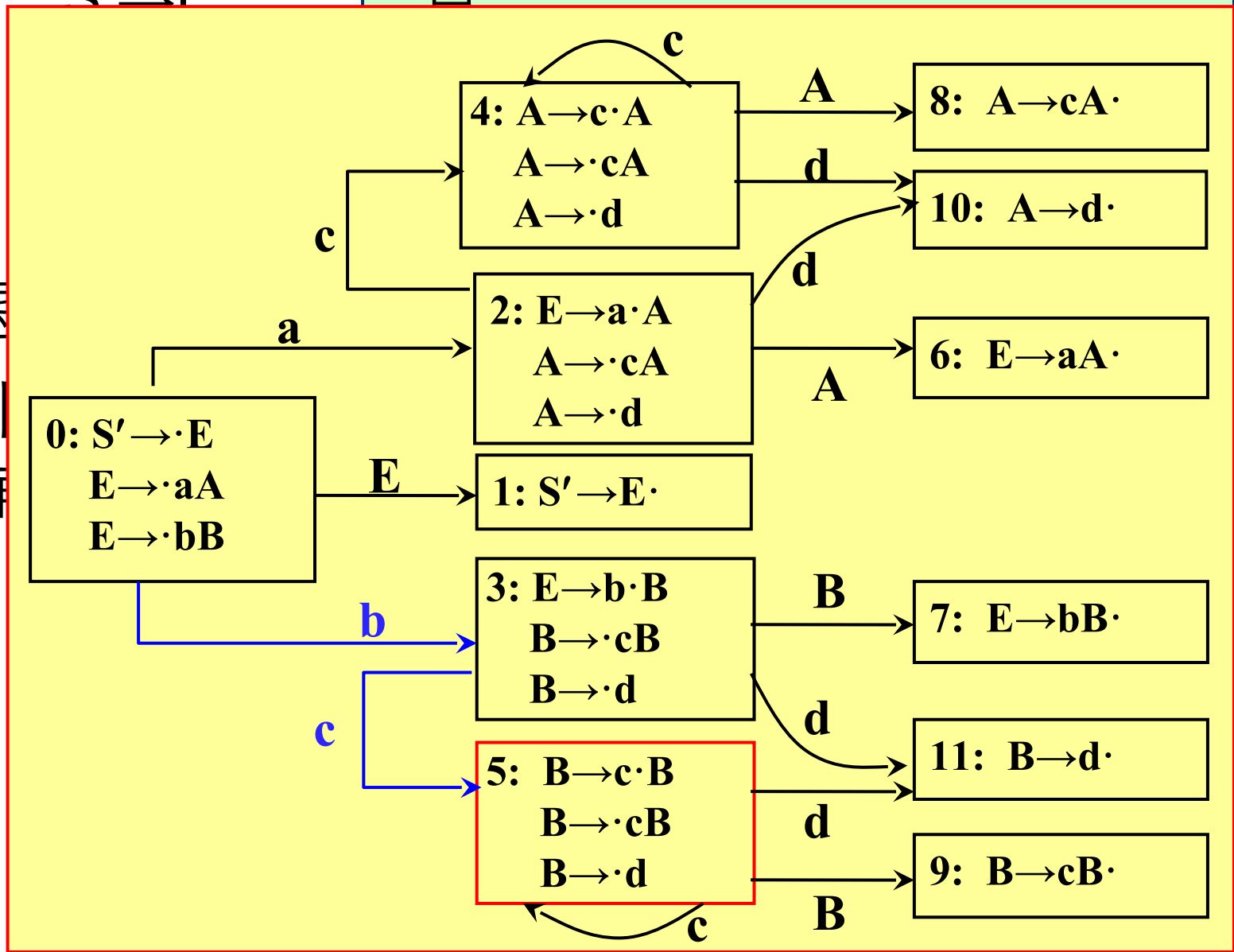
  项

  活

  S’

  S’

  S’



LR(0) 项目集规范族的构造

- 假定文法 G，以 S 为开始符号
- 构造一个 G′，它包含了整个 G，但它引进了一个不出现在 G 中的非终结符 S′，并加进一个新产生式 S′→S，S′ 是 G′ 的开始符号
- 称 G′ 是 G 的拓广文法
- G′ 唯一的“接受”态：仅含项目 S′→S• 的状态

闭包 CLOSURE

- 假定 I 是文法 G′ 的任一项目集，定义和构造 I 的闭包 CLOSURE(I) 如下：

2. 若状态 i 为 X→α•Aβ ， A 为非终结符，
则从状态 i 画一条ε边到所有状态 A→•γ 。

2 若 A→α·Bβ 属于 CLOSURE(I) ，那么

P50： NFA 确定化

设 I 是的状态集的一个子集，定义 I 的ε - 闭包ε -closure(I) 为：

i) 若 s∈I，则 s∈ε-closure(I)；

ii) 若 s∈I，则从 s 出发经过任意条ε弧而能到达的任何状态 s’ 都属于ε -closure(I)

ε-closure(I)=I∪{s’| 从某个 s∈I 出发经过任意条ε弧能到达

- 为了识别活前缀，我们定义一个状态转换函数 GO 是一个状态转换函数。I 是一个项目集，X 是一个文法符号。函数值 GO(I，X) 定义为：

$$GO(I, X) = CLOSURE(J)$$

其中

J = { 任何形如 $A \rightarrow \alpha X \cdot \beta$ 的项目 | $A \rightarrow \alpha \cdot X\beta$ 属于 I} 。

- **P50：设 a 是Σ中的一个字符，定义**

$$\mathbf{I_a = \varepsilon\text{-closure}(J)}$$

其中，J 为 I 中的某个状态出发经过一条 a 弧而到达的状态集合。

- 文法 G(S′)

  S′→E
  E→aA|bB
  A→cA|d
  B→cB|d

- I_0={S′→ · E, E→ · aA, E→ · bB}

GO(I_0, E)= closure(J)=closure({S’→E · })
= {S’→E · } = I_1

GO(I_0, a)= closure(J)=closure({E→a · A})
={ E→a · A, A→ · cA, A→ · d})=I_2

GO(I_0, b)= closure(J)=closure ({E →b · B})
={E →b · B, B→ · cB, B→ · d}= I_3

- 构造文法 G 的拓广文法 G′ 的 LR(0) 项目集规范族算法:

```
PROCEDURE  ITEMSETS(G′) ;
BEGIN
  C:={CLOSURE({S′→ · S})} ;
  REPEAT
     FOR  C 中每个项目集 I 和 G′ 的每个符号 X
  DO
          IF  GO(I ,  X) 非空且不属于 C   THEN
             把 GO(I ,  X) 放入 C 族中 ;
  UNTIL C   不再增大
END
```

- 转换函数 GO 把项目集连接成一个 DFA 转换图 .

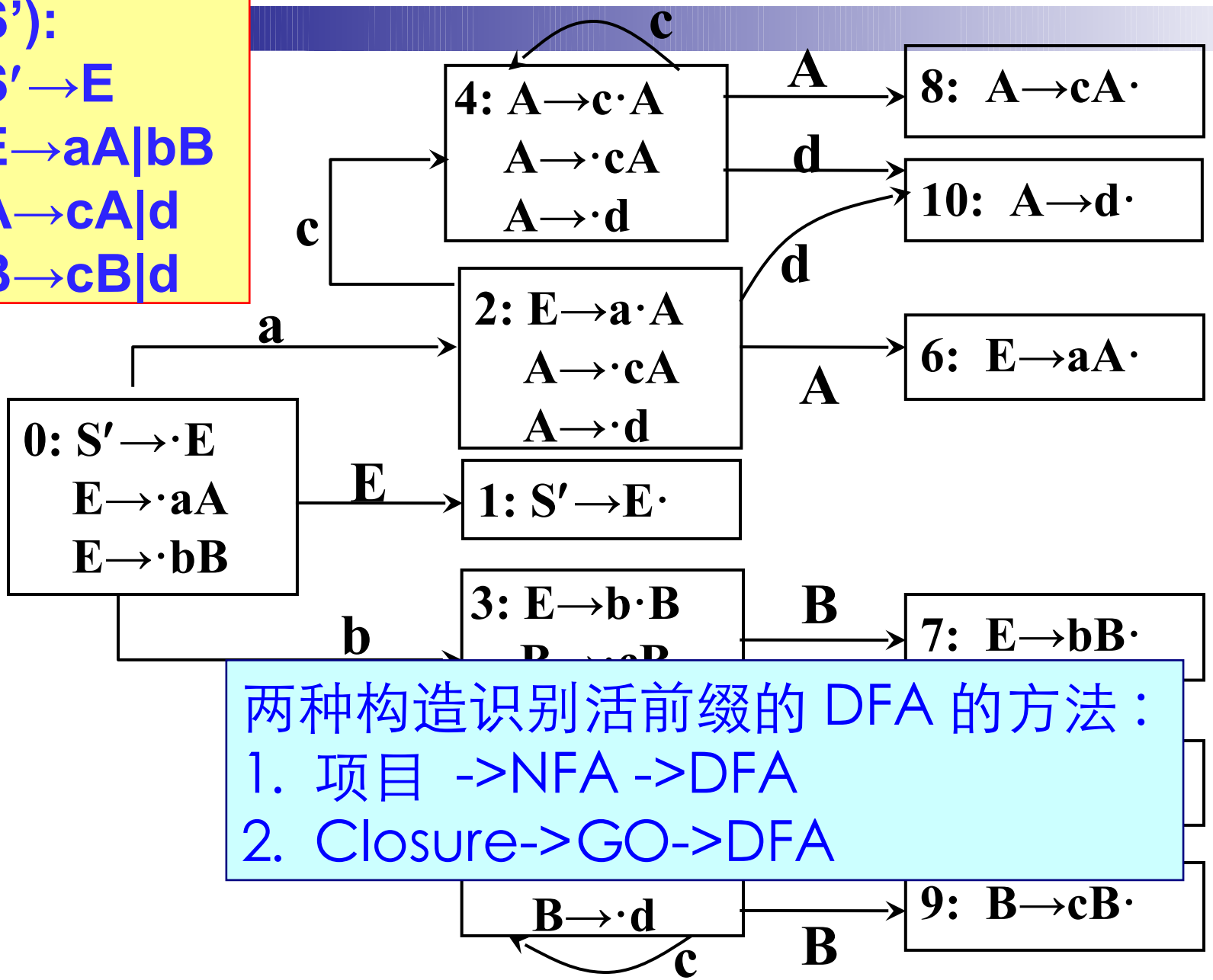
G'(S'):

$S' \rightarrow E$

$E \rightarrow aA|bB$

$A \rightarrow cA|d$

$B \rightarrow cB|d$



两种构造识别活前缀的 DFA 的方法：

1. 项目 ->NFA ->DFA
2. Closure->GO->DFA

小结

- 规范归约过程中，只要保证分析栈中总是活前缀，就说明分析采取的移进 / 归约动作是正确的
- 哪些字符串是活前缀？能不能构造一个 DFA 来识别活前缀？
- 两种构造识别活前缀的 DFA 的方法
  - 项目 ->NFA ->DFA
  - Closure->GO->DFA