



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



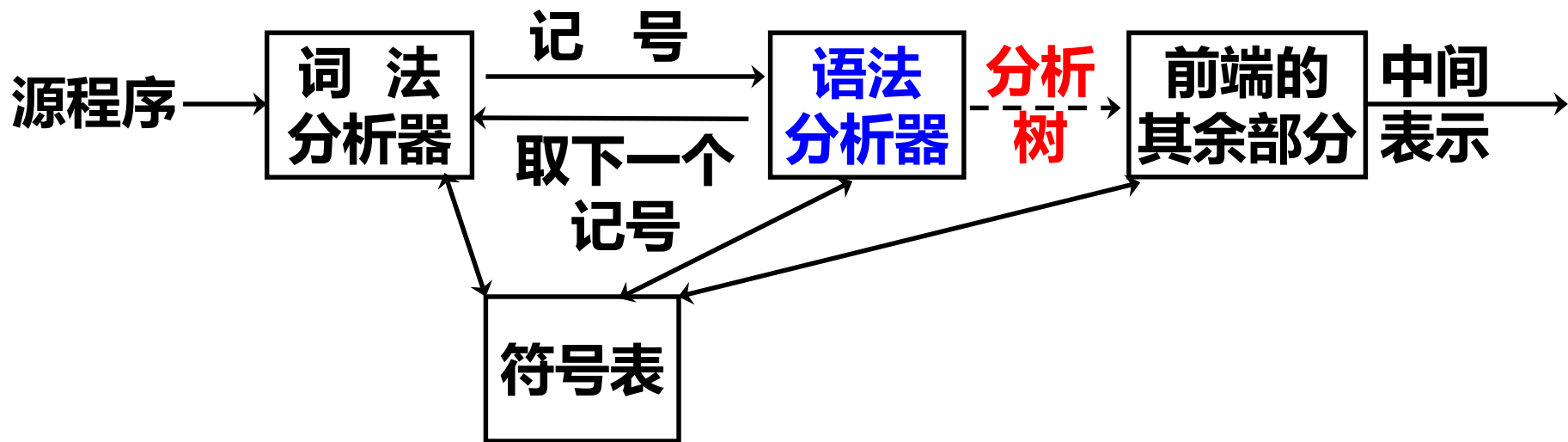
《编译原理与技术》

语法分析 V

计算机科学与技术学院

李诚

11/10/2018



□LR(k)分析技术

- ❖ 规范的LR方法
- ❖ 向前看的LR方法（简称LALR）
- ❖ LR错误恢复及处理
- ❖ 分析器的生成器



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

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

$=$

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



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

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

V

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

$=$

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

项目 $S \rightarrow V \cdot = E$ 使得
 $\text{action}[2, =] = \text{s6}$

项目 $E \rightarrow V \cdot$ 使得
 $\text{action}[2, =] = \text{r5}$
 $\text{Follow}(E) = \{=, \$\}$

产生移进-归约冲突，
但该文法不是二义的。

□与SLR(1)分析的区别

- ❖ 项目集的定义发生了改变
- ❖ $\text{closure}(I)$ 和 GOTO 函数需要修改

□添加了前向搜索符

- ❖ 一个项目 $A \rightarrow \alpha \cdot \beta$ ，如果最终用这个产生式进行归约之后，期望看见的符号是 a ，则这个加点项的前向搜索符是 a 。
- ❖ 上述项目可以写成： $A \rightarrow \alpha \cdot \beta, a$

□项目集改变的目的是增强描述能力

□LR(1)项目:

❖重新定义项目，让它带上搜索符，成为如下形式

$$[A \rightarrow \alpha \cdot \beta, a]$$

□LR(1)项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对活前缀 γ 有效:

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

➤ $\gamma = \delta \alpha$;

➤ a 是 w 的第一个符号，或者 w 是 ϵ 且 a 是 $\$$



□例 $S \rightarrow BB$

$B \rightarrow bB \mid a$

从最右推导 $S \Rightarrow_{rm}^* bbBba \Rightarrow_{rm} bbbBba$ 看出：

$[B \rightarrow b \cdot B, b]$ 对活前缀 $\gamma = bbb$ 是有效的

对于项目 $[A \rightarrow \alpha \cdot \beta, a]$ ，当 β 为空时，是**根据搜索符 a 来填表**（归约项目），而不是根据 A 的后继符来填表

□构造LR(1)项目集规范族

❖ 初始项目集 I_0 :

$[S' \rightarrow \cdot S, \$]$ 将\$作为向前的搜索符

□计算闭包CLOSURE(I)

❖ I中的任何项目都属于CLOSURE(I)

❖ 若有项目 $[A \rightarrow \alpha \cdot B \beta, a]$ 在CLOSURE(I)中，而 $B \rightarrow \gamma$ 是文法中的产生式， b 是FIRST(βa)中的元素，则 $[B \rightarrow \cdot \gamma, b]$ 也属于CLOSURE(I)

保证在用 $B \rightarrow \gamma$ 进行归约后，

- 出现的输入字符 b 是句柄 $\alpha B \beta$ 中 B 的后继符号
- 或者是 $\alpha B \beta$ 归约为 A 后可能出现的终结符。



规范的LR分析：举例



$S' \rightarrow \cdot S, \$$ I_0

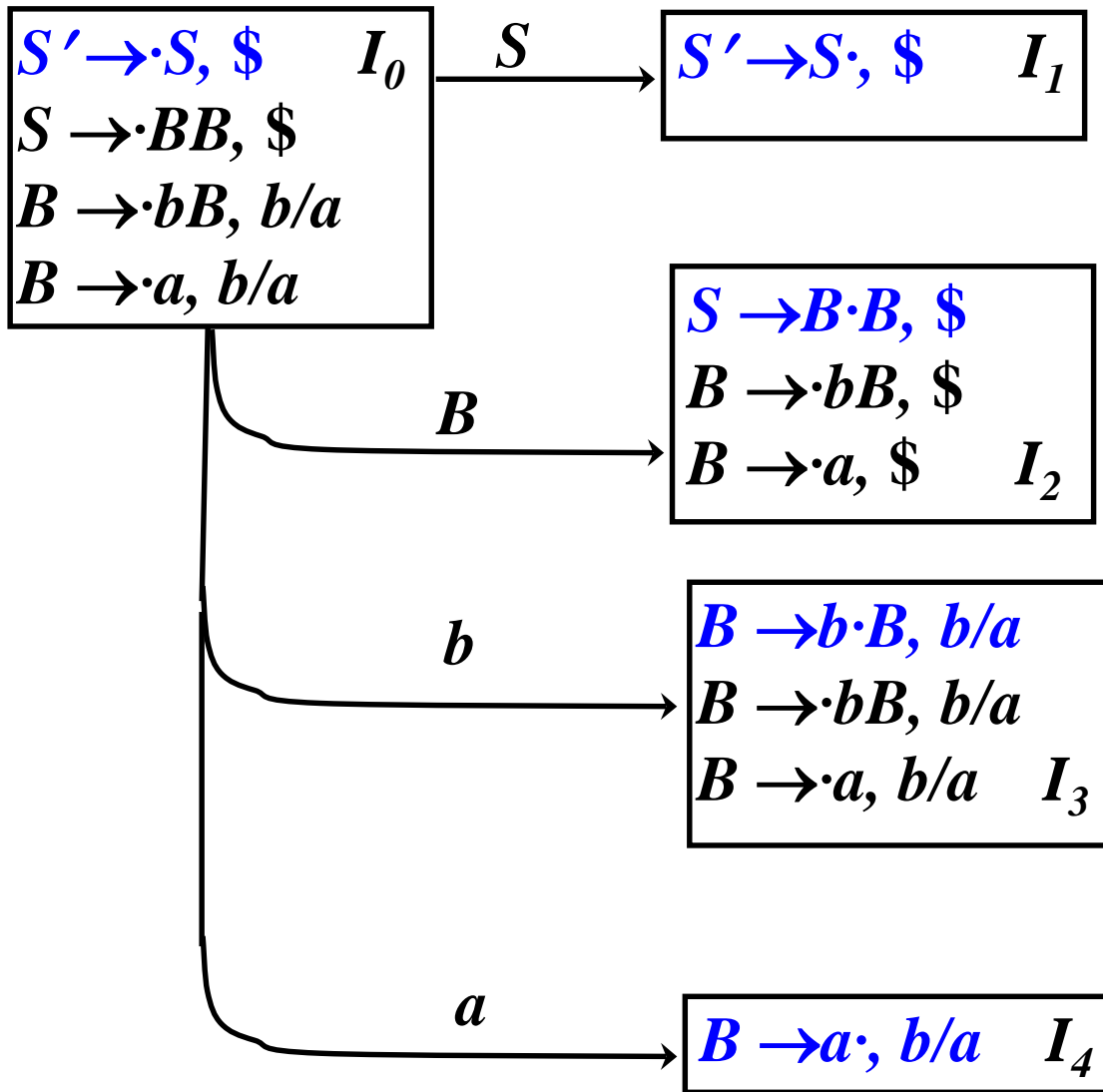
$S \rightarrow \cdot BB, \$$

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

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

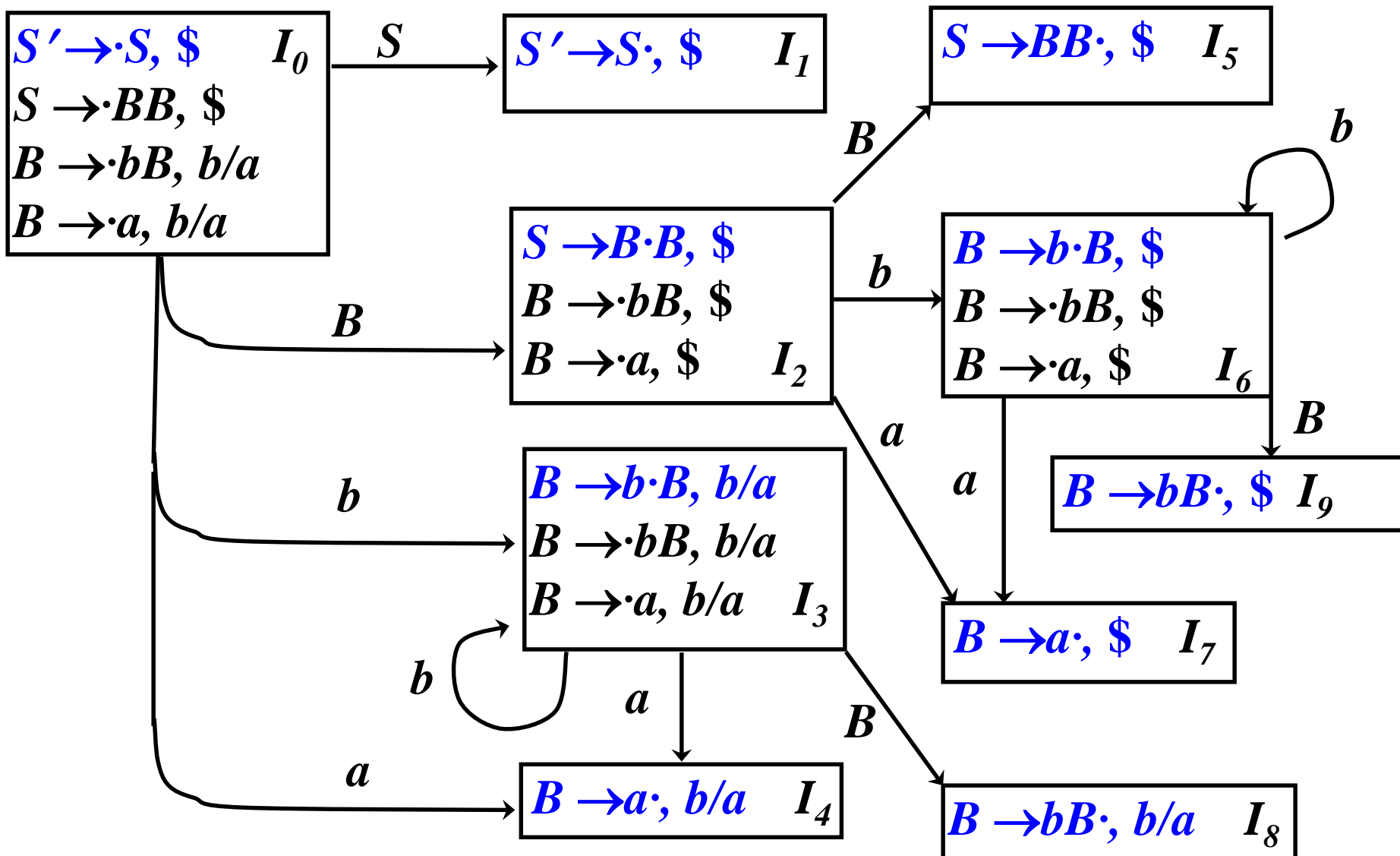


规范的LR分析：举例





规范的LR分析：举例





□构造识别拓广文法 G' 活前缀的DFA

❖基于LR(1)项目族来构造

□状态 i 的 $action$ 函数如下确定:

❖如果 $[A \rightarrow \alpha \cdot a \beta, b]$ 在 I_i 中, 且 $goto(I_i, a) = I_j$, 那么置 $action[i, a]$ 为 sj

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

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

如果上述构造出现了冲突, 那么文法就不是LR(1)的



□构造识别拓广文法 G' 活前缀的DFA

□状态 i 的 $action$ 函数如下确定:

❖参见上页ppt

□状态 i 的 $goto$ 函数如下确定:

❖如果 $goto(I_i, A) = I_j$, 那么 $goto[i, A] = j$



□构造识别拓广文法 G' 活前缀的DFA

□状态 i 的 $action$ 函数如下确定:

❖参见上页ppt

□状态 i 的 $goto$ 函数如下确定:

❖如果 $goto(I_i, A) = I_j$, 那么 $goto[i, A] = j$

□分析器的初始状态是包含 $[S' \rightarrow \cdot S, \$]$ 的项目集对应的状态

用上面规则未能定义的所有条目都置为**error**

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

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

V

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

$=$

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

项目 $S \rightarrow V \cdot = E$ 使得
 $\text{action}[2, =] = \text{s6}$

项目 $E \rightarrow V \cdot$ 使得
 $\text{action}[2, =] = \text{r5}$
 $\text{Follow}(E) = \{=, \$\}$

产生移进-归约冲突，
但该文法不是二义的。



非SLR(1)但是LR(1)文法



$S \rightarrow V = E$
 $S \rightarrow E$
 $V \rightarrow * E$
 $V \rightarrow id$
 $E \rightarrow V$

$I_0:$
 $S' \rightarrow \cdot S, \$$
 $S \rightarrow \cdot V = E, \$$
 $S \rightarrow \cdot E, \$$
 $V \rightarrow \cdot * E, =/\$$
 $V \rightarrow \cdot id, =/\$$
 $E \rightarrow \cdot V, \$$

V →

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

无移进归约冲突

计算闭包:

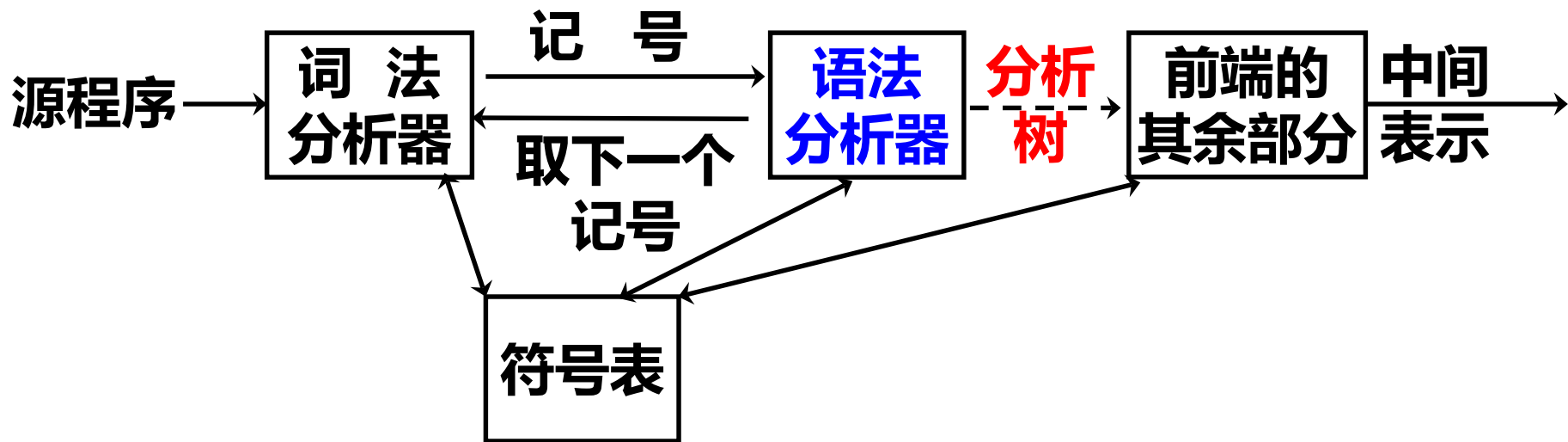
定义里: $[A \rightarrow \alpha \cdot B \beta, a]$

这里: $[S \rightarrow \cdot V = E, \$]$

FIRST(βa)



FIRST($=E\$$) = {=}



□LR(k)分析技术

- ❖ 规范的LR方法
- ❖ 向前看的LR方法（简称LALR）
- ❖ LR错误恢复及处理
- ❖ 分析器的生成器

□研究LALR的原因

规范LR分析表的状态数偏多

□LALR特点

- ❖ LALR和SLR的分析表有同样多的状态，比规范LR分析表要小得多
- ❖ LALR的能力介于SLR和规范LR之间
- ❖ LALR的能力在很多情况下已经够用

□LALR分析表构造方法

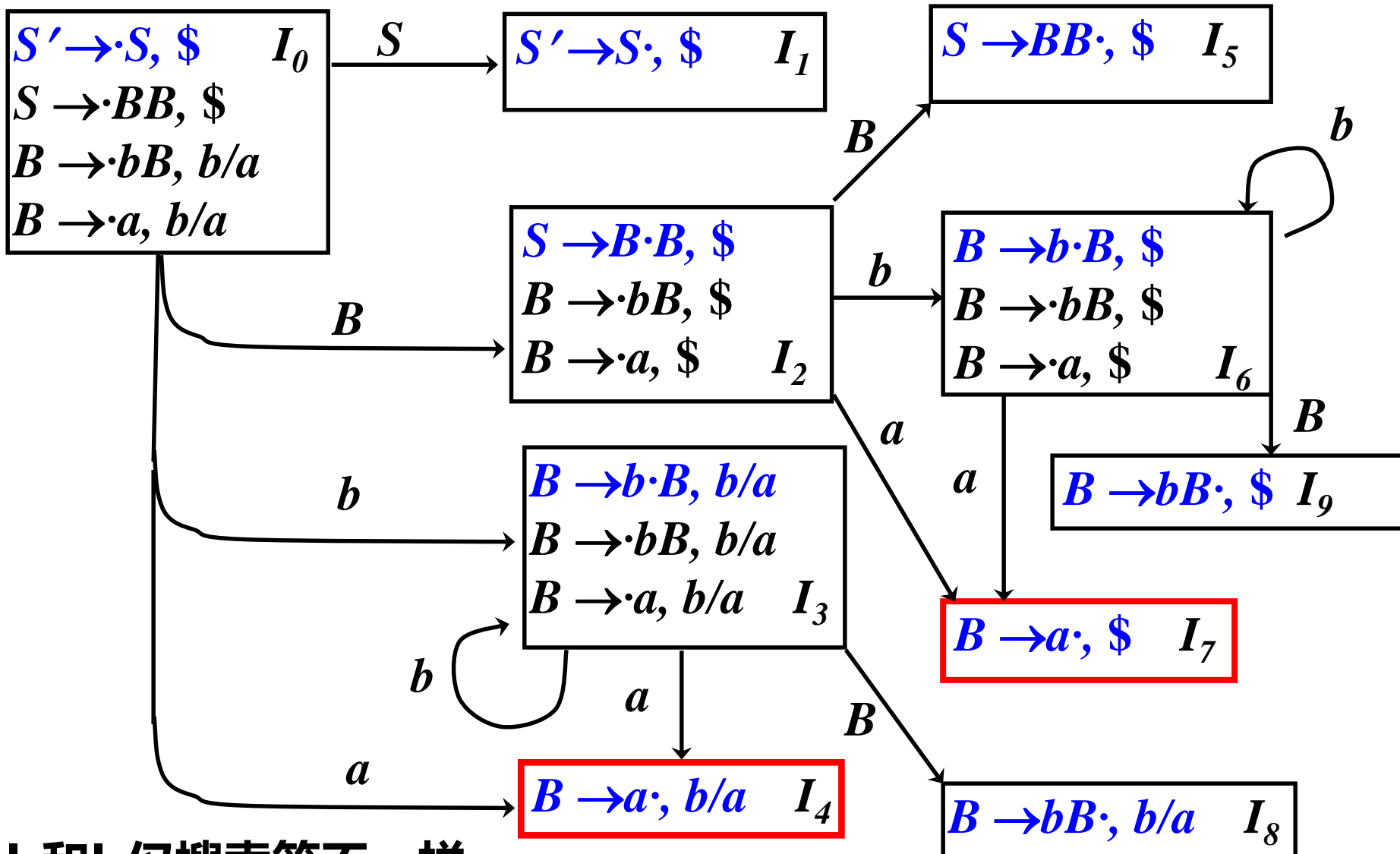
- ❖ 通过合并规范LR(1)项目集来得到



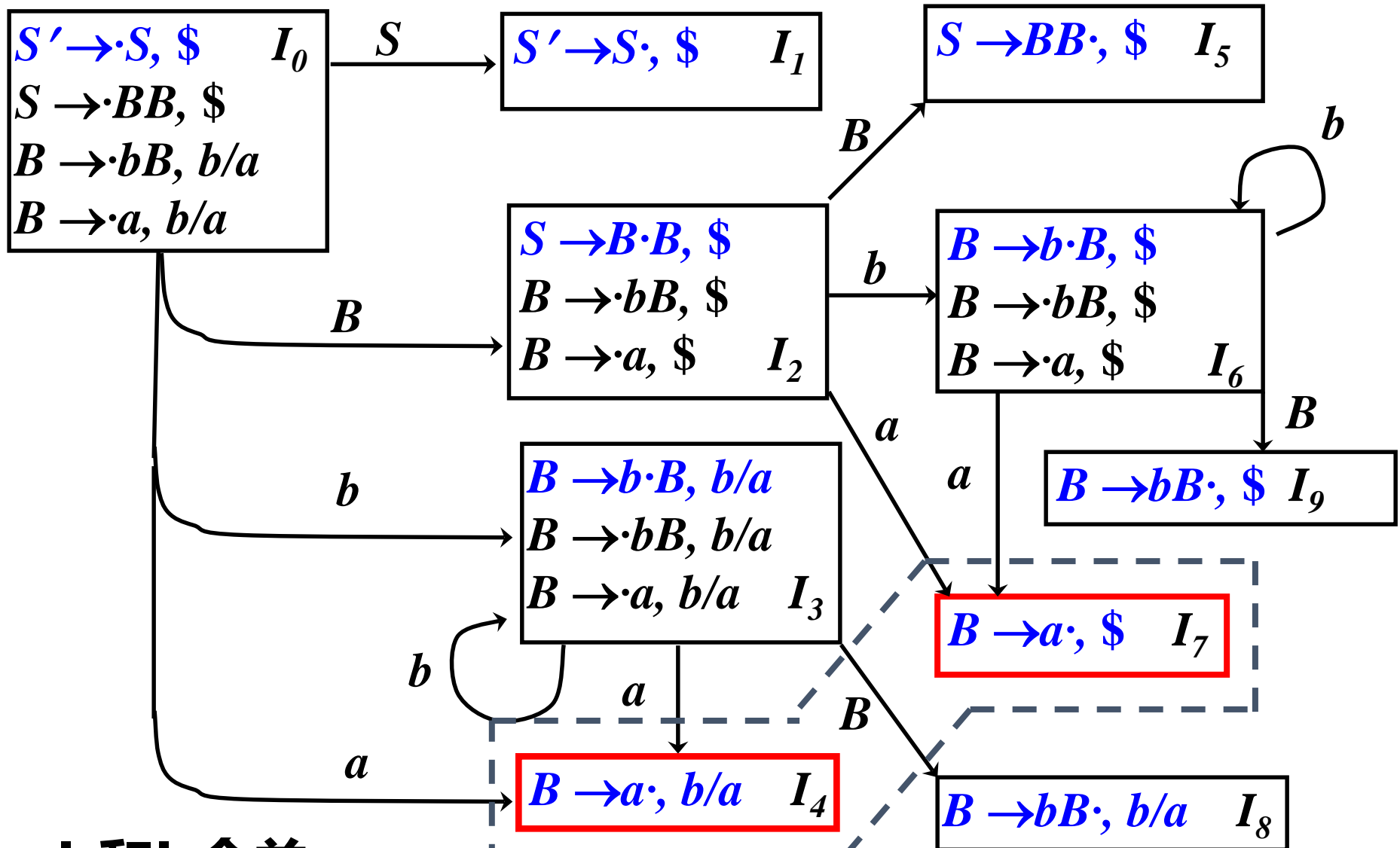
- 合并识别 LR(1)文法的活前缀的DFA中的同心项目集。
- 同心的LR(1)项目集
 - ❖ 略去搜索符后它们是相同的集合
 - ❖ 例： $B \rightarrow \cdot bB, \$$ 与 $B \rightarrow \cdot bB, b/a$



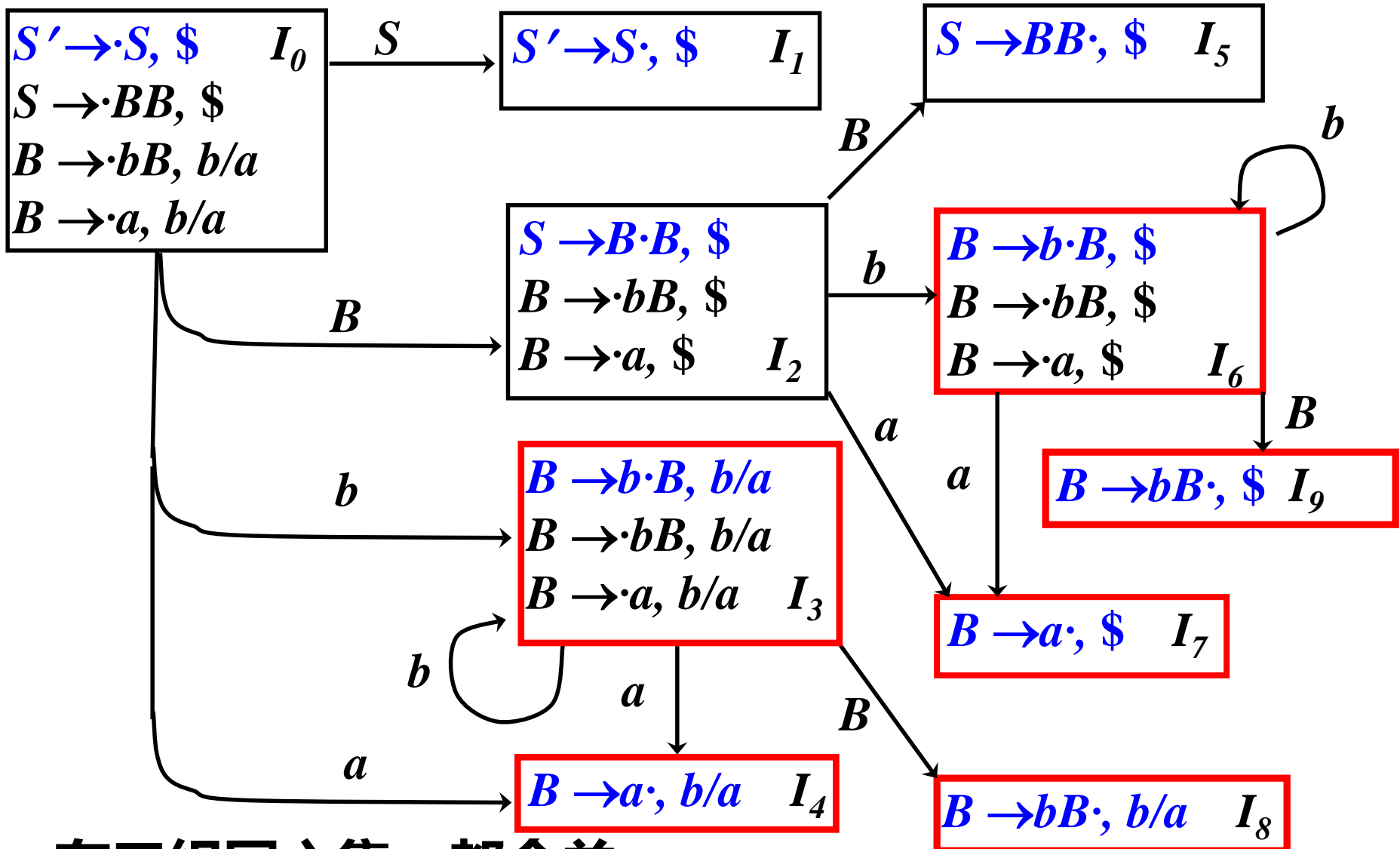
识别活前缀的DFA



I_4 和 I_7 仅搜索符不一样



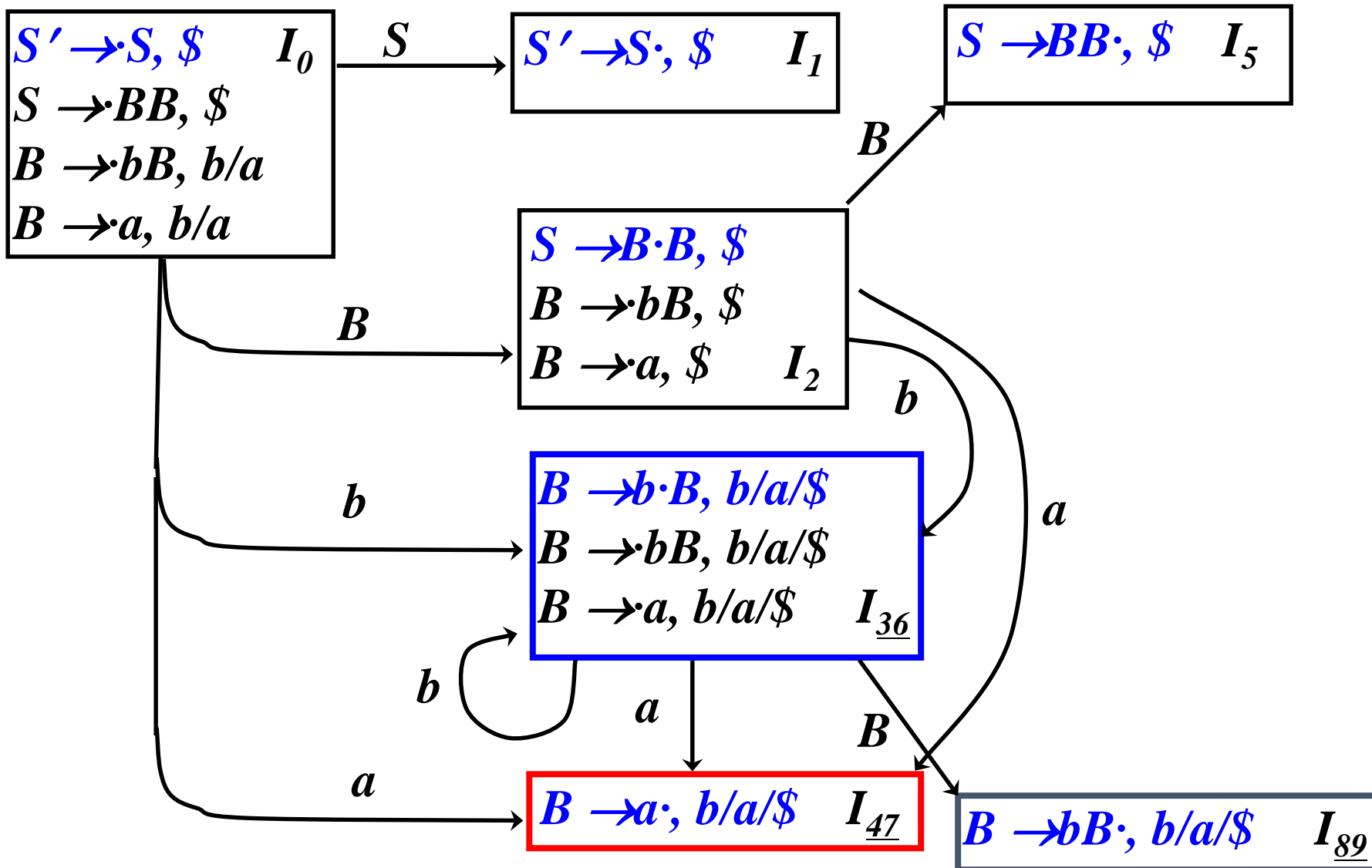
I_4 和 I_7 合并



有三组同心集，都合并

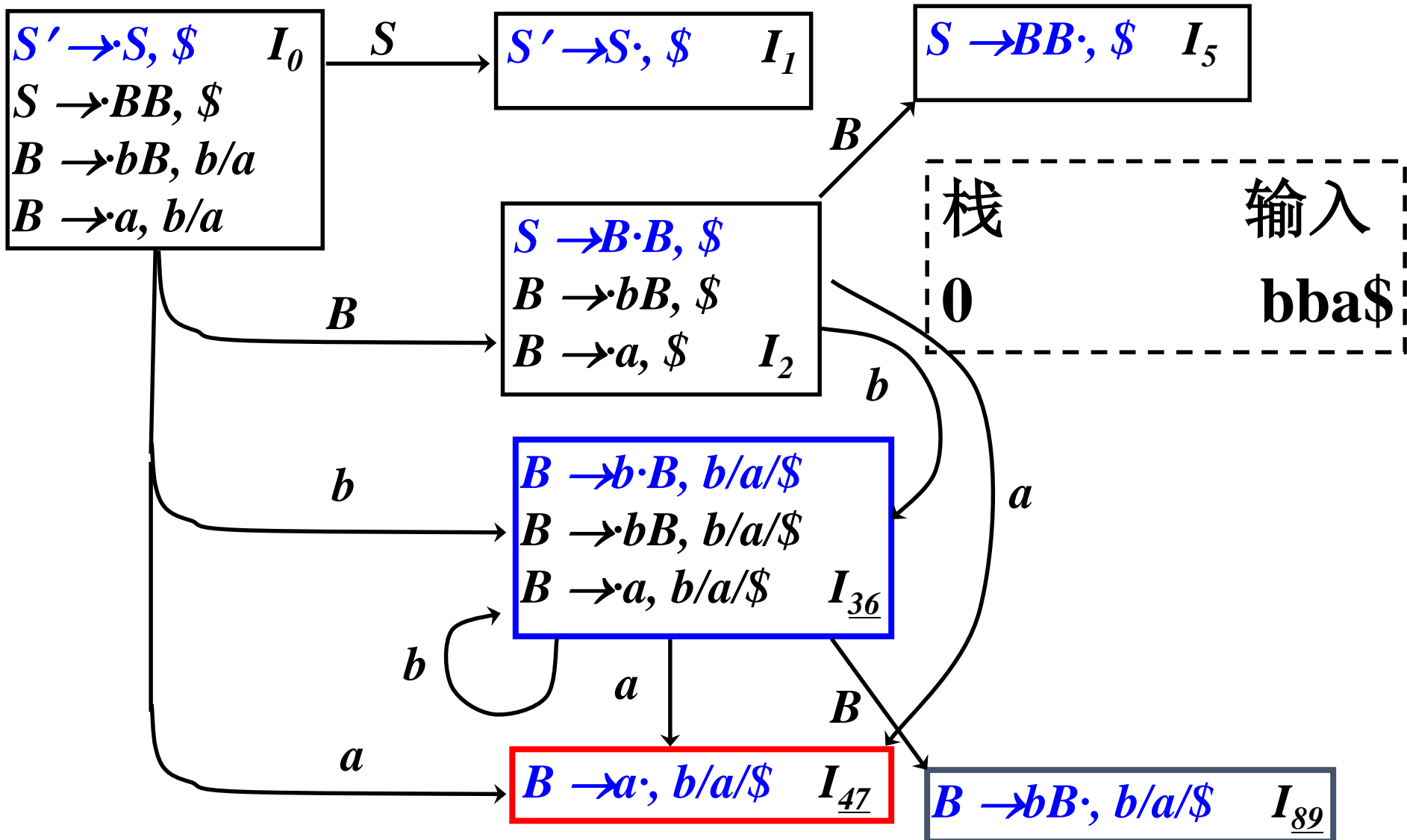


合并同心项目集



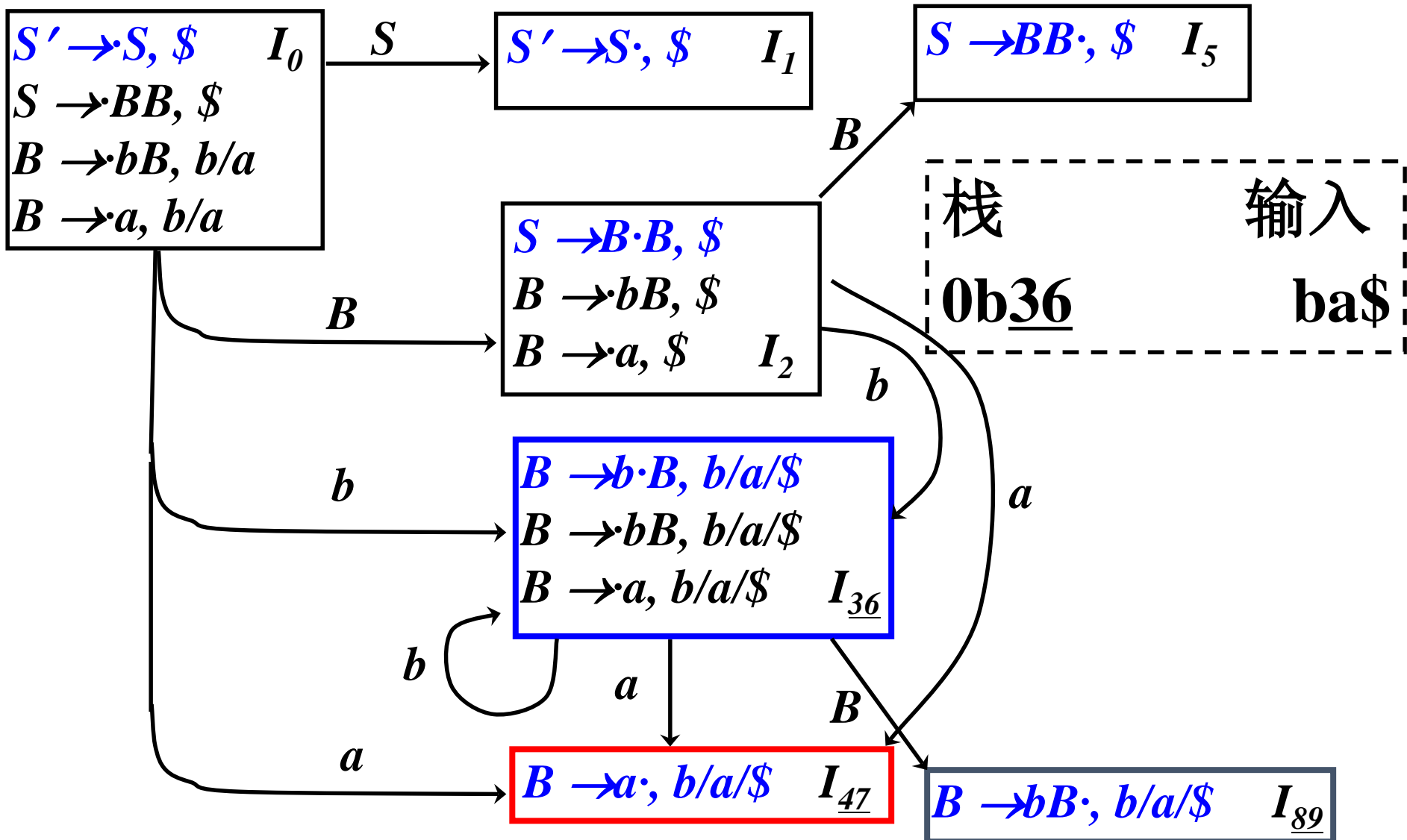


LALR分析: 举例



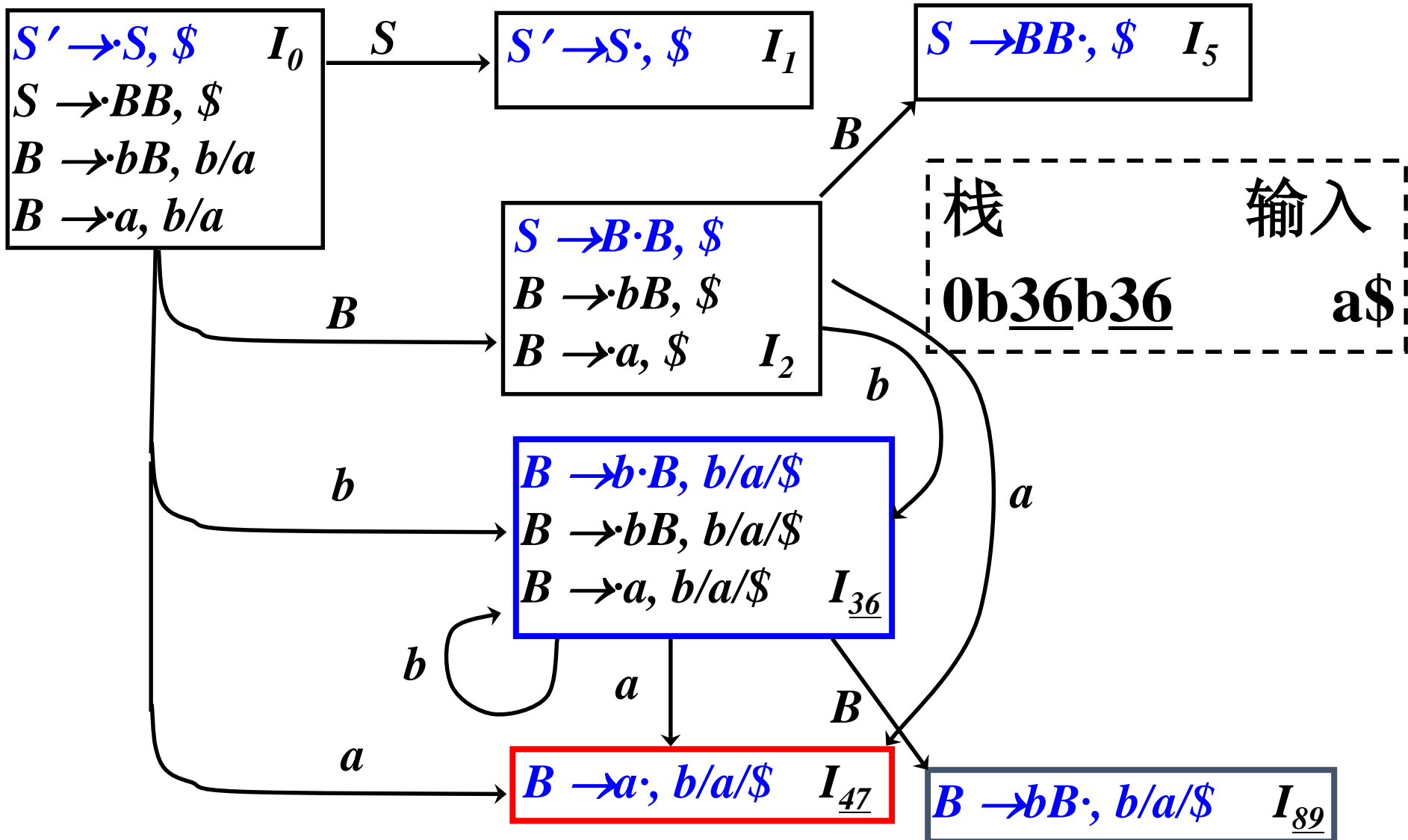


LALR分析: 举例



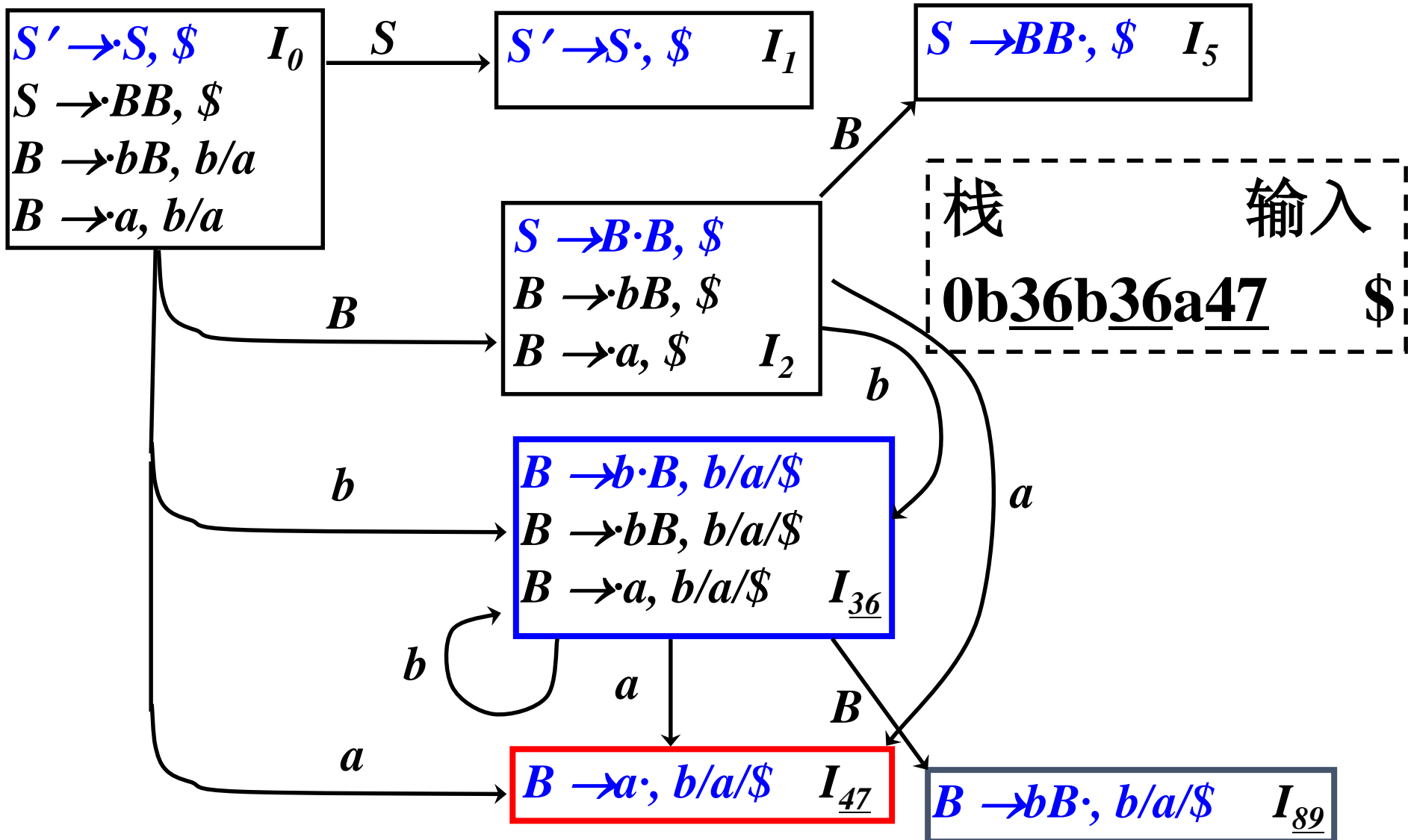


LALR分析: 举例



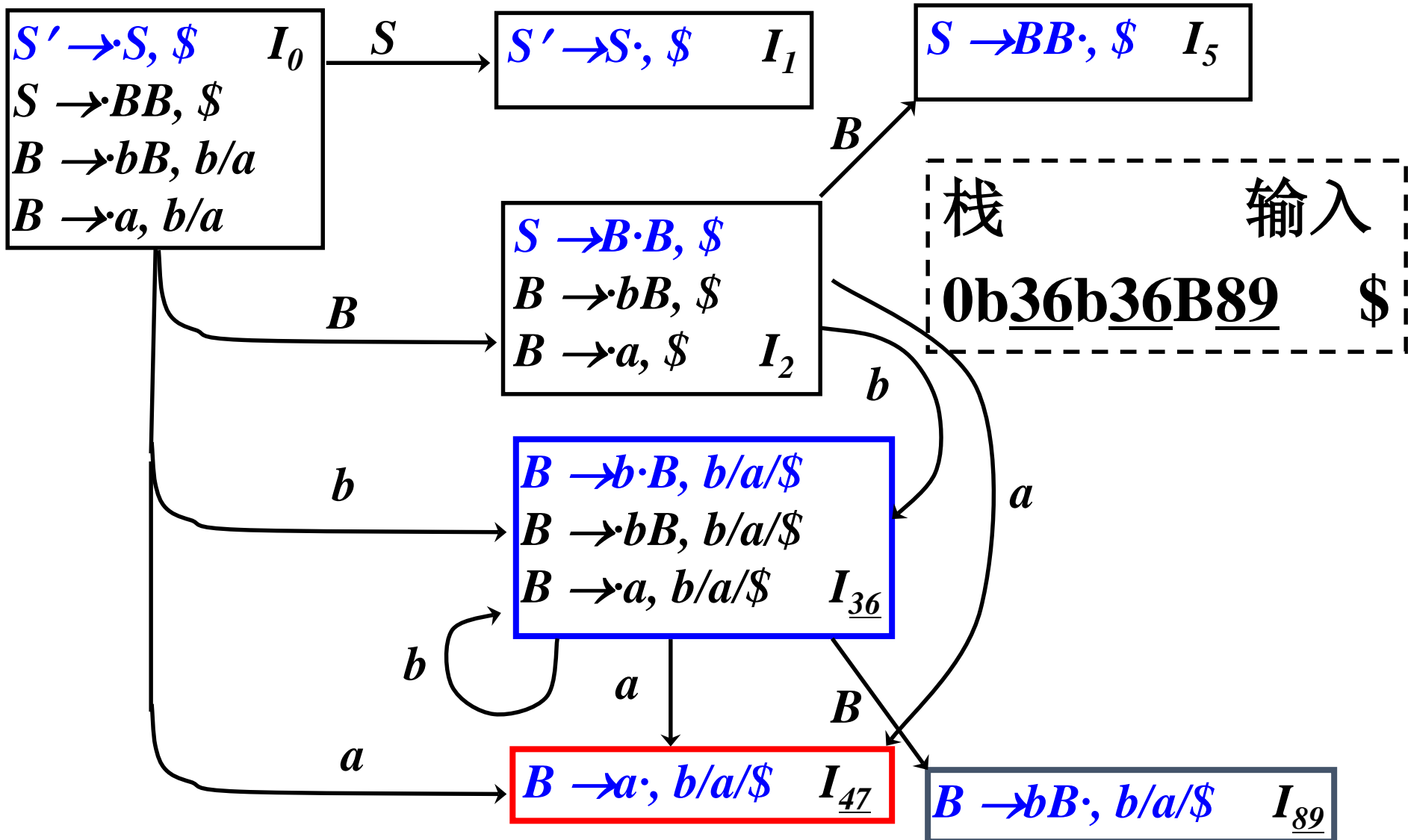


LALR分析: 举例



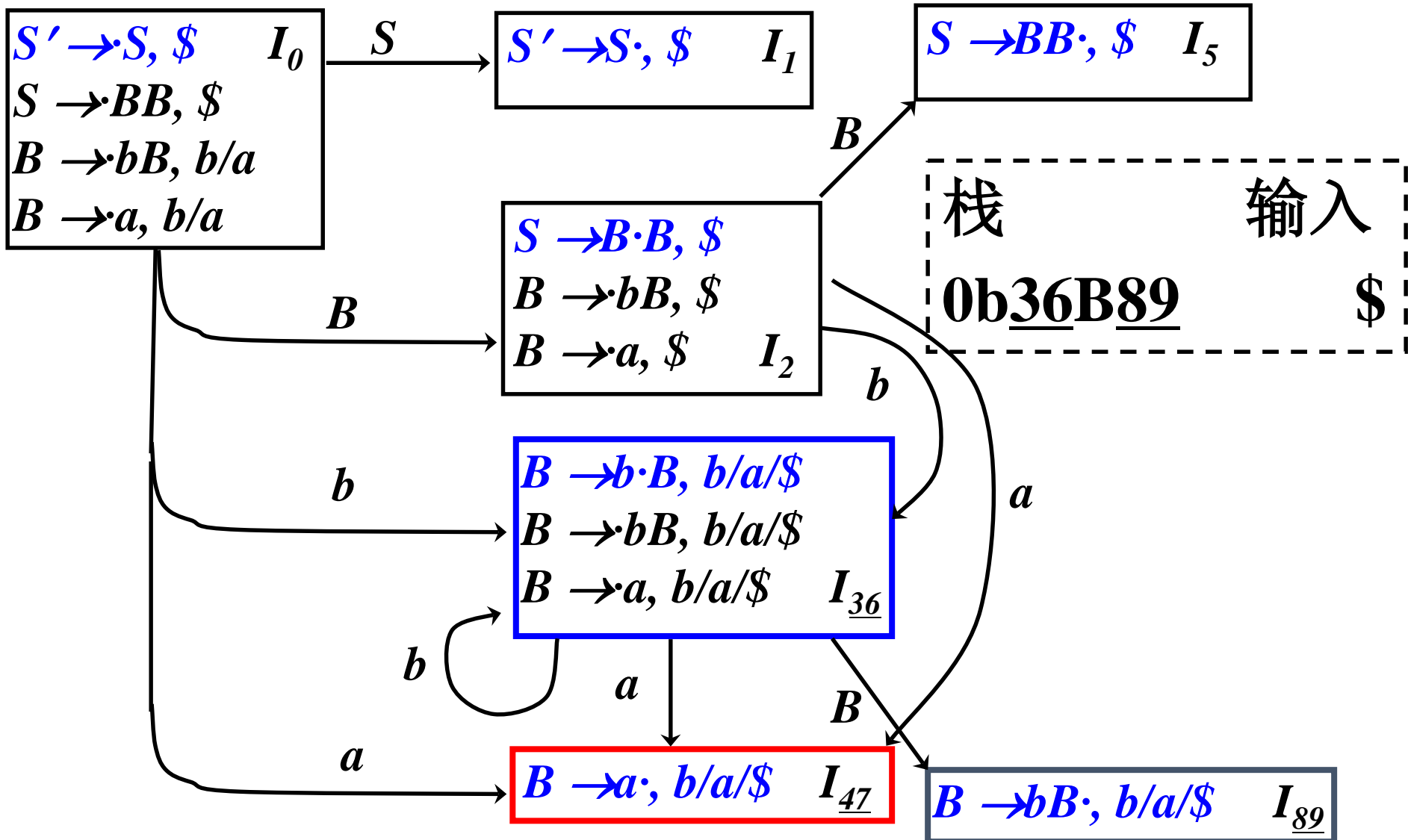


LALR分析: 举例



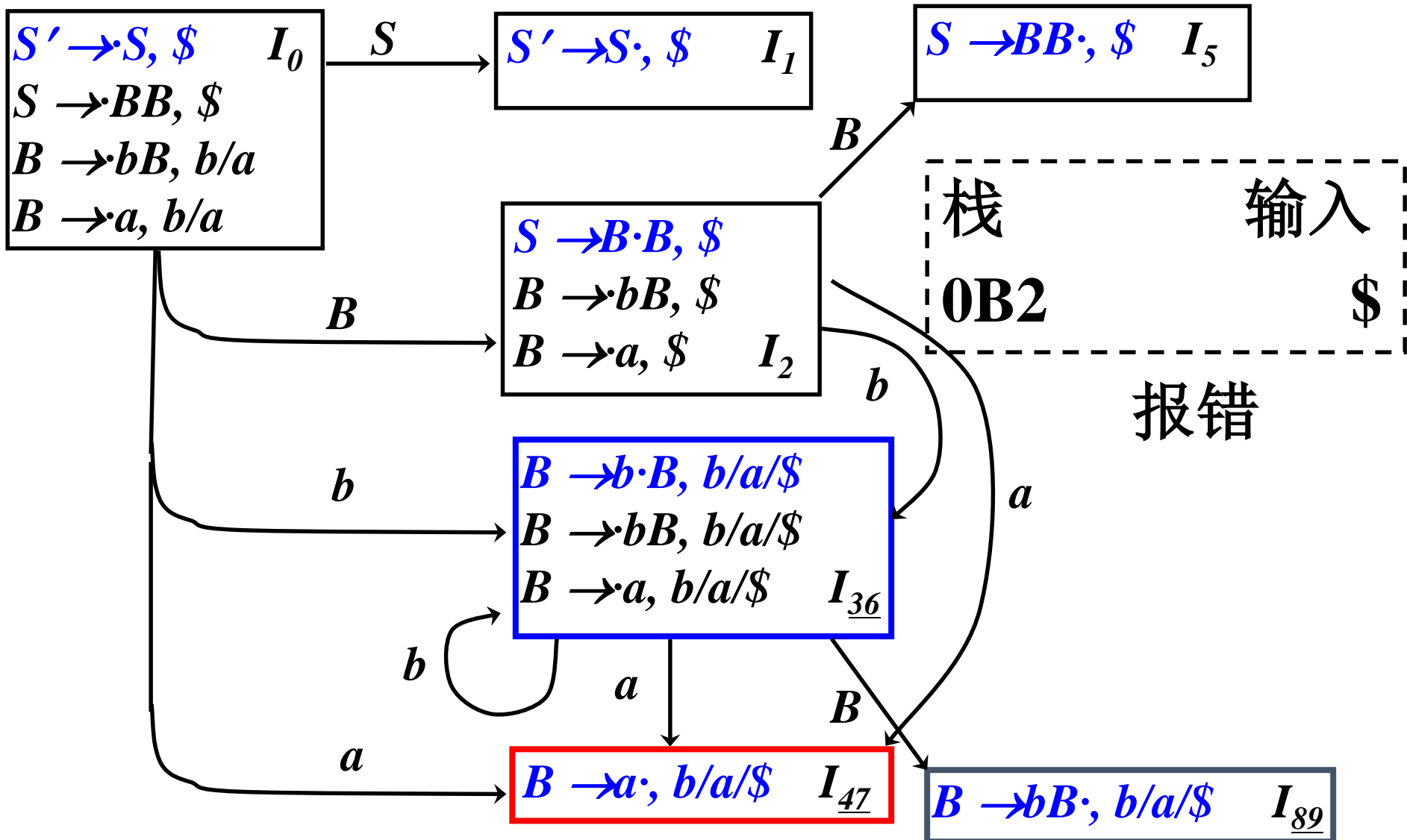


LALR分析: 举例



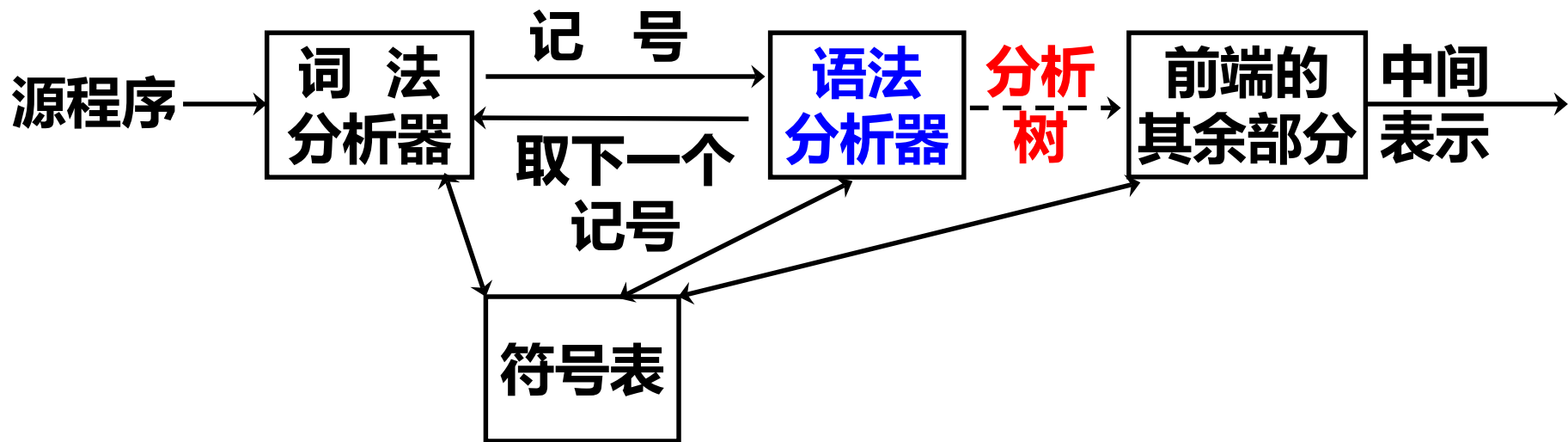


LALR分析: 举例



2、构造LALR(1)分析表

- (1) 构造LR(1)项目集规范族 $C = \{I_0, I_1, \dots, I_n\}$
- (2) 寻找LR(1)项目集规范族中同心的项目集，
用它们的并集代替它们
- (3) 按构造规范LR(1)分析表的方式构造分析表



□LR(k)分析技术

- ❖ 规范的LR方法
- ❖ 向前看的LR方法（简称LALR）
- ❖ LR错误恢复及处理
- ❖ 分析器的生成器

1、合并同心项目集可能会引起冲突

□同心集的合并不会引起新的移进-归约冲突

项目集1

$[A \rightarrow \alpha \cdot, a]$

...

项目集2

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

...

若合并后有冲突

1、合并同心项目集可能会引起冲突

□同心集的合并不会引起新的移进-归约冲突

项目集1

$[A \rightarrow \alpha \cdot, a]$

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

...

项目集2

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

$[A \rightarrow \alpha \cdot, d]$

...

则合并前就有冲突

1、合并同心项目集可能会引起冲突

- 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

$A \rightarrow c \cdot, d$ $B \rightarrow c \cdot, e$
--

对bc有效的项目集

$A \rightarrow c \cdot, e$ $B \rightarrow c \cdot, d$
--

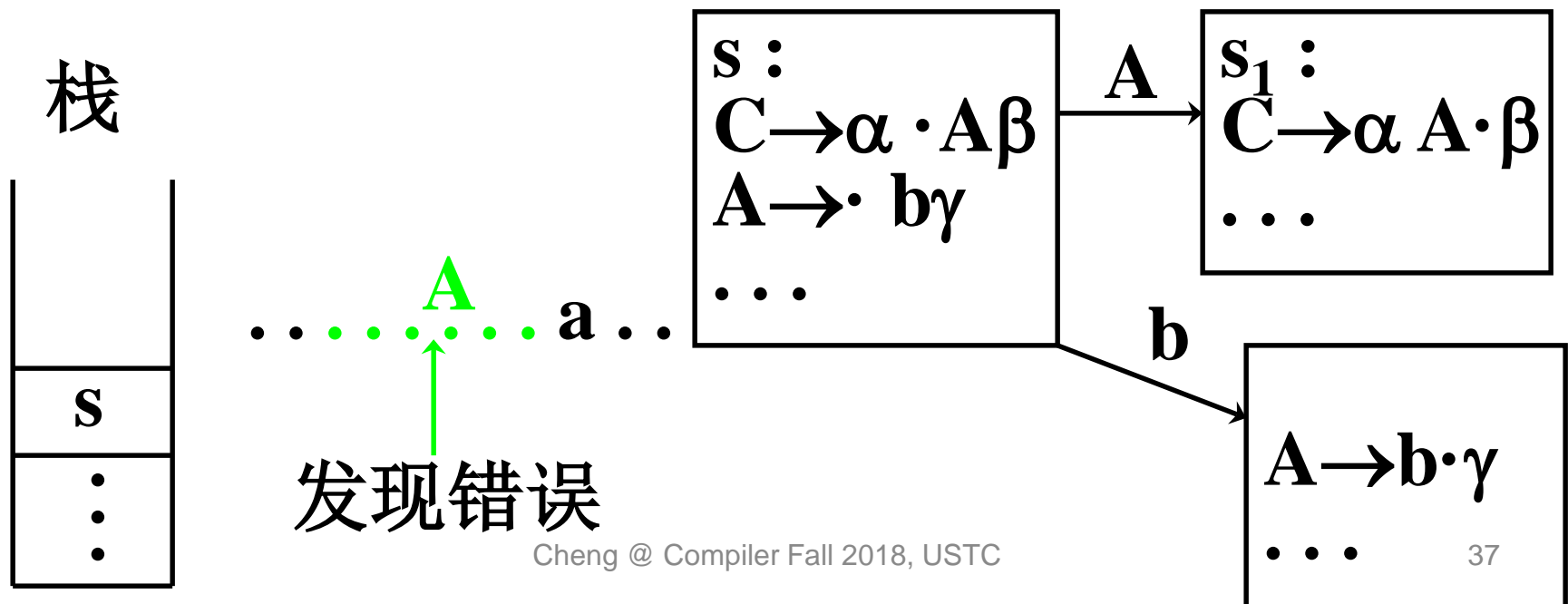
合并同心集后

$A \rightarrow c \cdot, d/e$ $B \rightarrow c \cdot, d/e$
--

该文法是LR(1)的,
但不是LALR(1)的

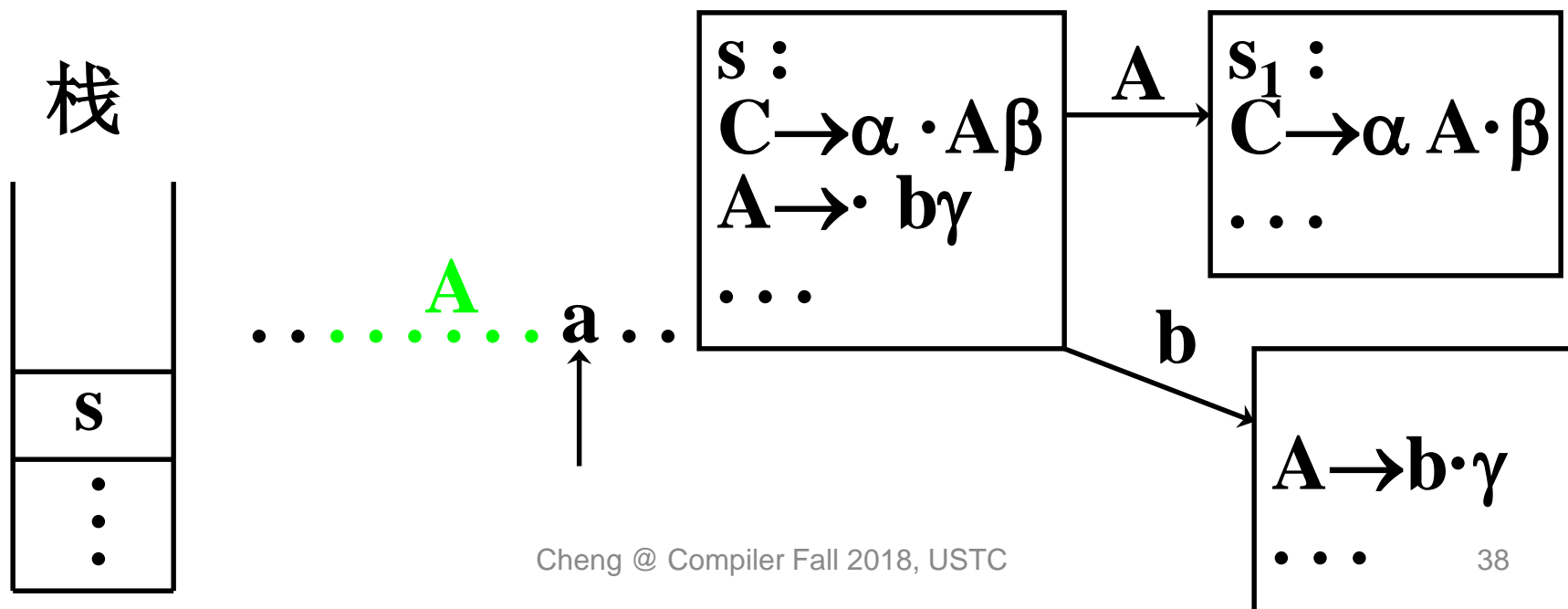
2、紧急方式错误恢复

(1) 退栈，直至出现状态 s ，它有预先确定的 A 的转移



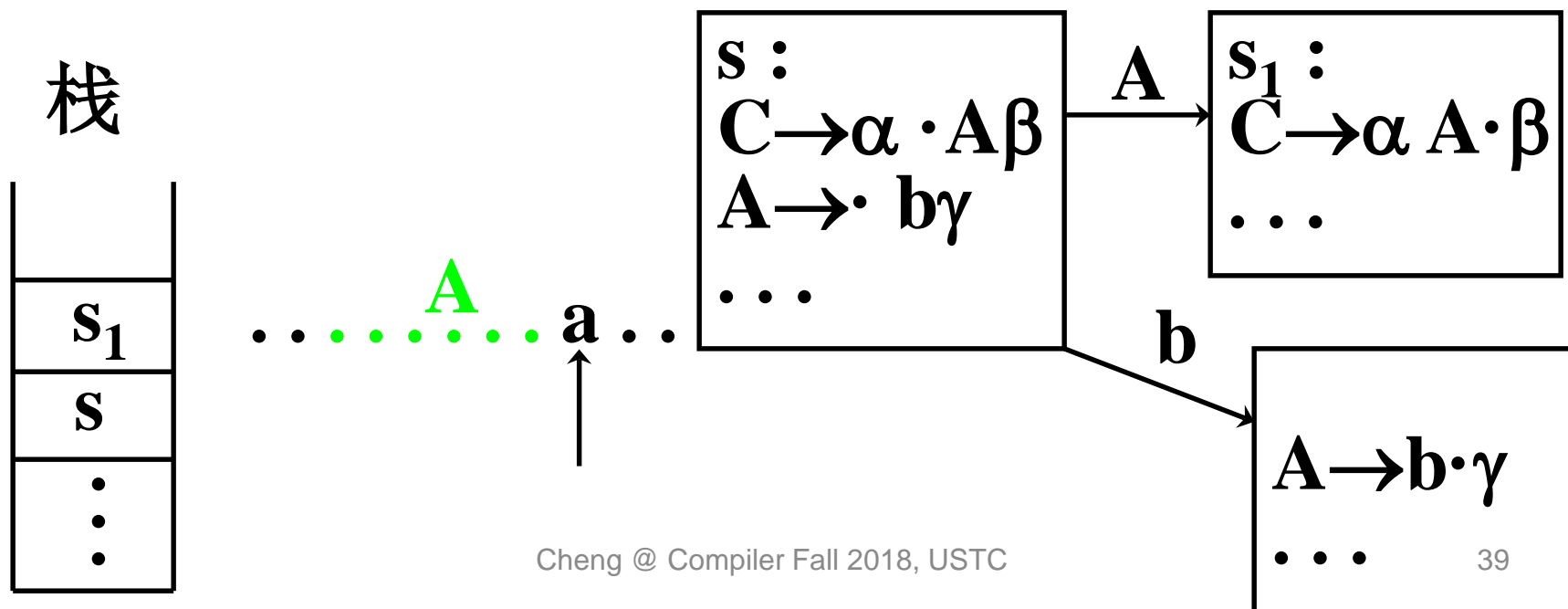
2、紧急方式错误恢复

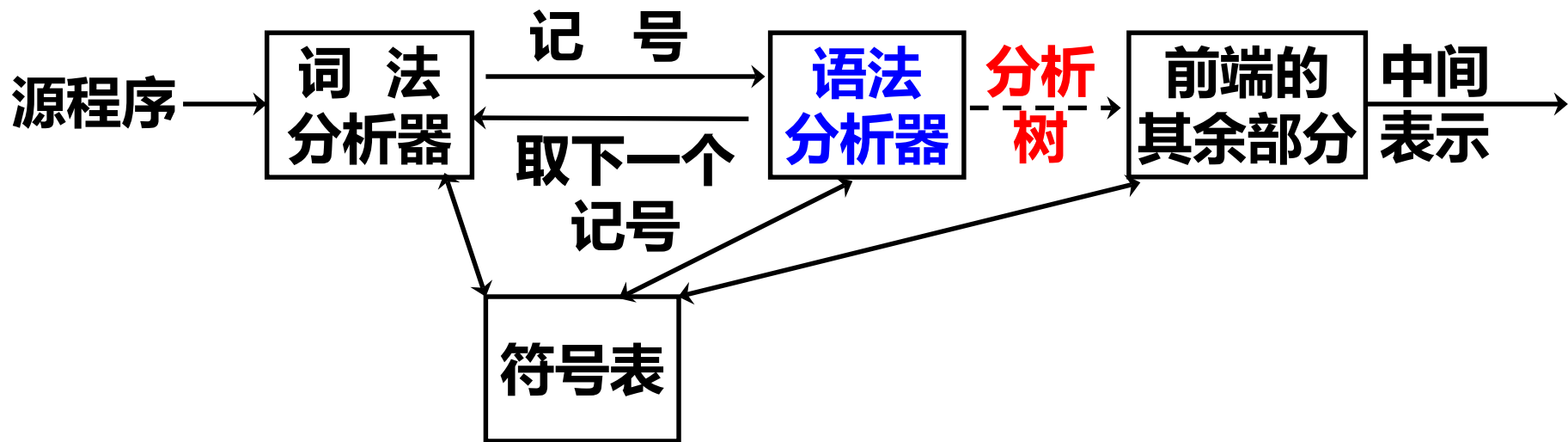
- (1) 退栈，直至出现状态 s ，它有预先确定的 A 的转移
- (2) 抛弃若干输入符号，直至找到 a ，它是 A 的合法后继



2、紧急方式错误恢复

- (1) 退栈，直至出现状态 s ，它有预先确定的 A 的转移
- (2) 抛弃若干输入符号，直至找到 a ，它是 A 的合法后继
- (3) 再把 A 和状态 $goto[s, A]$ 压进栈，恢复正常分析

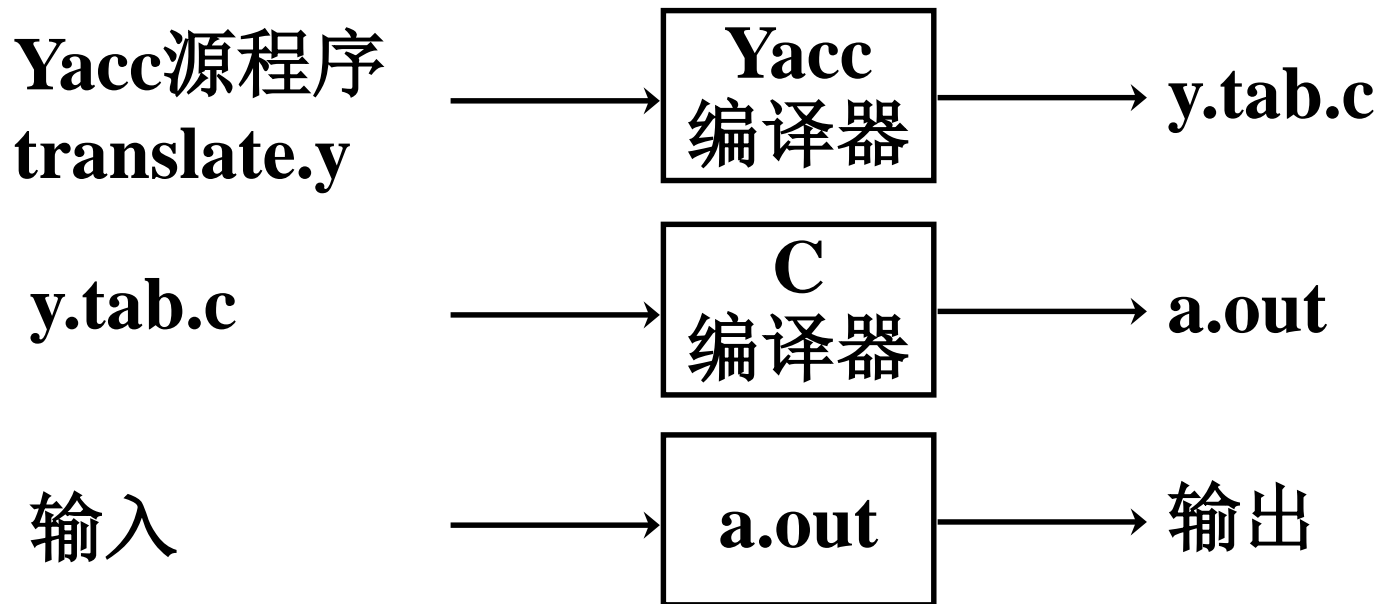




□LR(k)分析技术

- ❖ 规范的LR方法
- ❖ 向前看的LR方法（简称LALR）
- ❖ LR错误恢复及处理
- ❖ 分析器的生成器

分析器的生成器 Yacc



用Yacc处理二义文法

□例 简单计算器

- ❖ 输入一个表达式并回车，显示计算结果
- ❖ 也可以输入一个空白行

```
%{  
# include <ctype .h>  
# include <stdio.h >  
# define YYSTYPE double /*将栈定义为double类型 */  
%}
```

```
%token NUMBER  
%left '+' '-'  
%left '*' '/'  
%right UMINUS  
%%
```

```
lines      : lines expr '\n'  {printf ( "%g \n", $2 ) }
           | lines '\n'
           | /* ε */
           ;

expr       : expr '+' expr   { $$ = $1 + $3; }
           | expr '-' expr   { $$ = $1 - $3; }
           | expr '*' expr   { $$ = $1 * $3; }
           | expr '/' expr   { $$ = $1 / $3; }
           | '(' expr ')'    { $$ = $2; }
           | '-' expr %prec UMINUS { $$ = -$2; }
           | NUMBER
           ;

%%
```

```
lines      : lines expr '\n'  {printf ( "%g \n", $2 ) }
           | lines '\n'
           | /* ε */
           ;

expr       : expr '+' expr   { $$ = $1 + $3; }
           | expr '-' expr   { $$ = $1 - $3; }
           | expr '*' expr   { $$ = $1 * $3; }
           | expr '/' expr   { $$ = $1 / $3; }
           | '(' expr ')'    { $$ = $2; }
           | '-' expr %prec UMINUS { $$ = -$2; }
           | NUMBER
           ;
```

% %

-5+10看成是-(5+10), 还是(-5)+10? 取后者

```
yylex () {  
    int c;  
    while ( ( c = getchar ( ) ) == ' ');  
    if ( ( c == '.' ) || (isdigit ( c ) ) ) {  
        ungetc ( c, stdin);  
        scanf ( "%lf", &yylval);  
        return NUMBER;  
    }  
    return c;  
}
```

为了C编译器能准确报告yylex函数中错误的位置，需要在生成的程序y.tab.c中使用编译命令#line

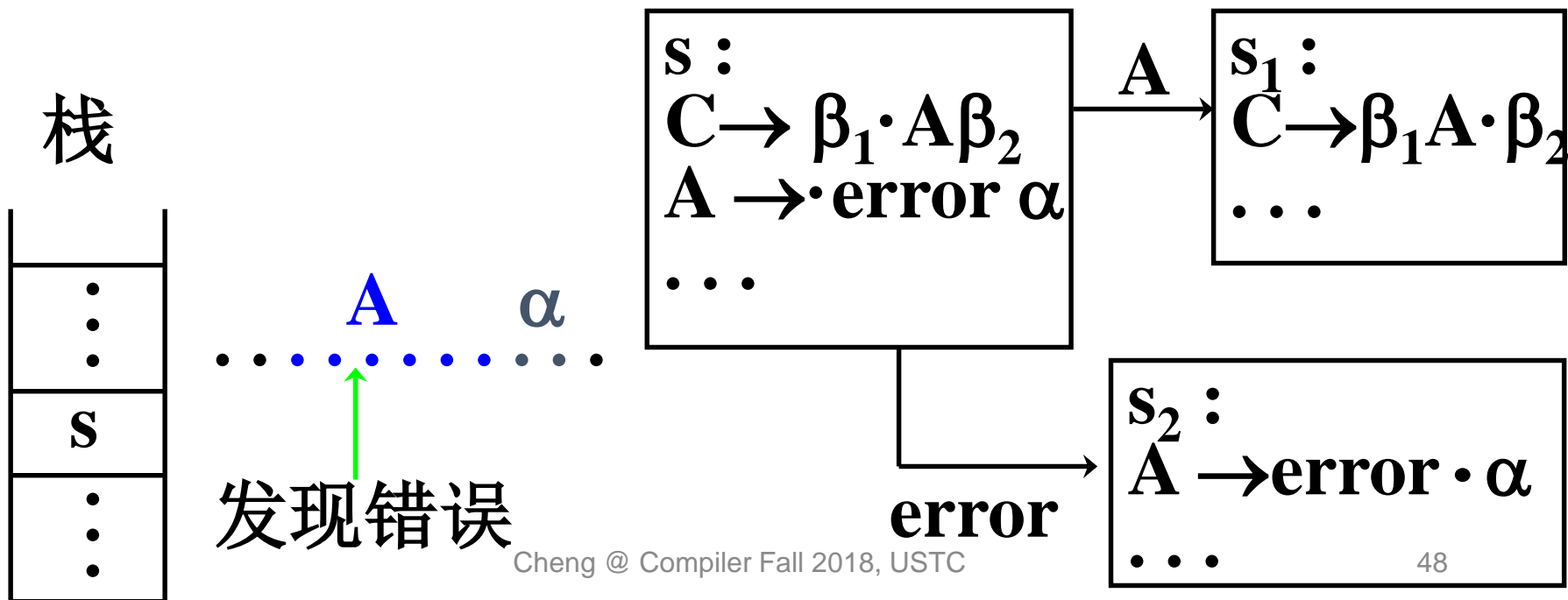
Yacc的错误恢复

□ 编译器设计者的工作

- ❖ 决定哪些“主要的”非终结符将有错误恢复与它们相关联
- ❖ 为各主要非终结符 A 加入形式为 $A \rightarrow \text{error } \alpha$ 的错误产生式，其中 α 是文法符号串
- ❖ 为这样的产生式配上语义动作

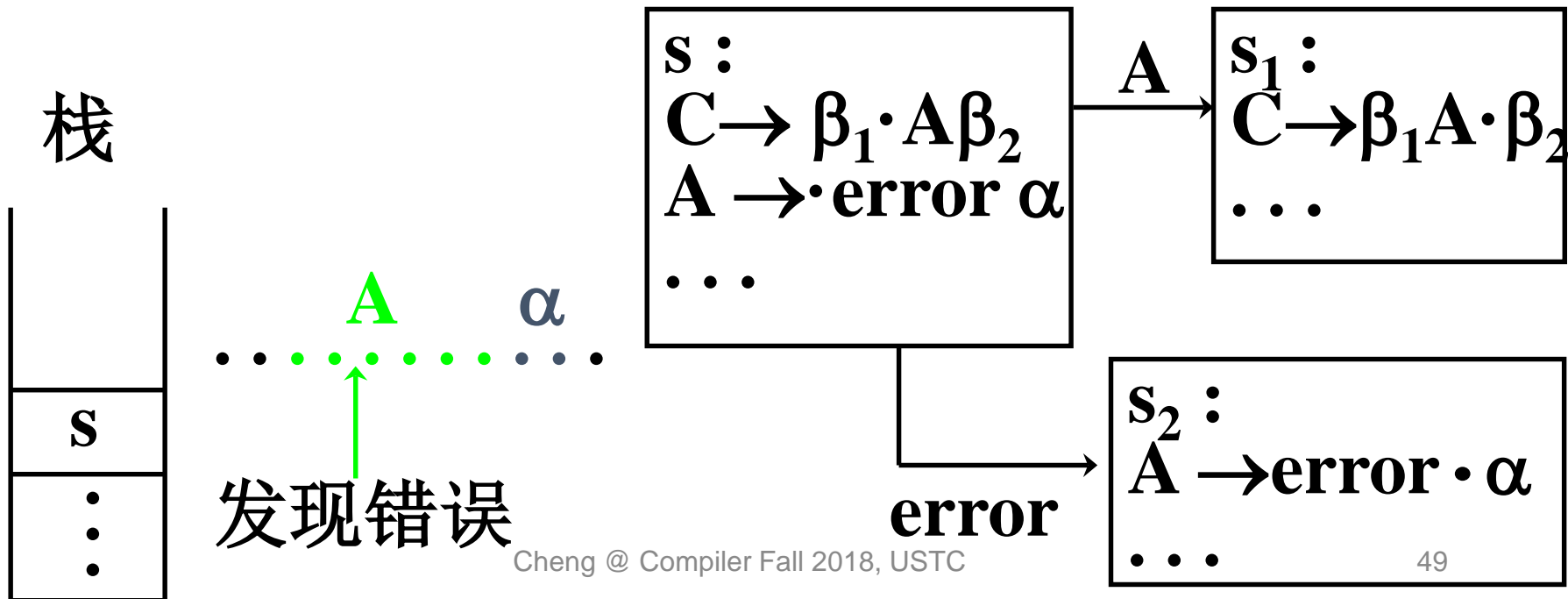
□ Yacc把错误产生式当作普通产生式处理

遇到语法错误时



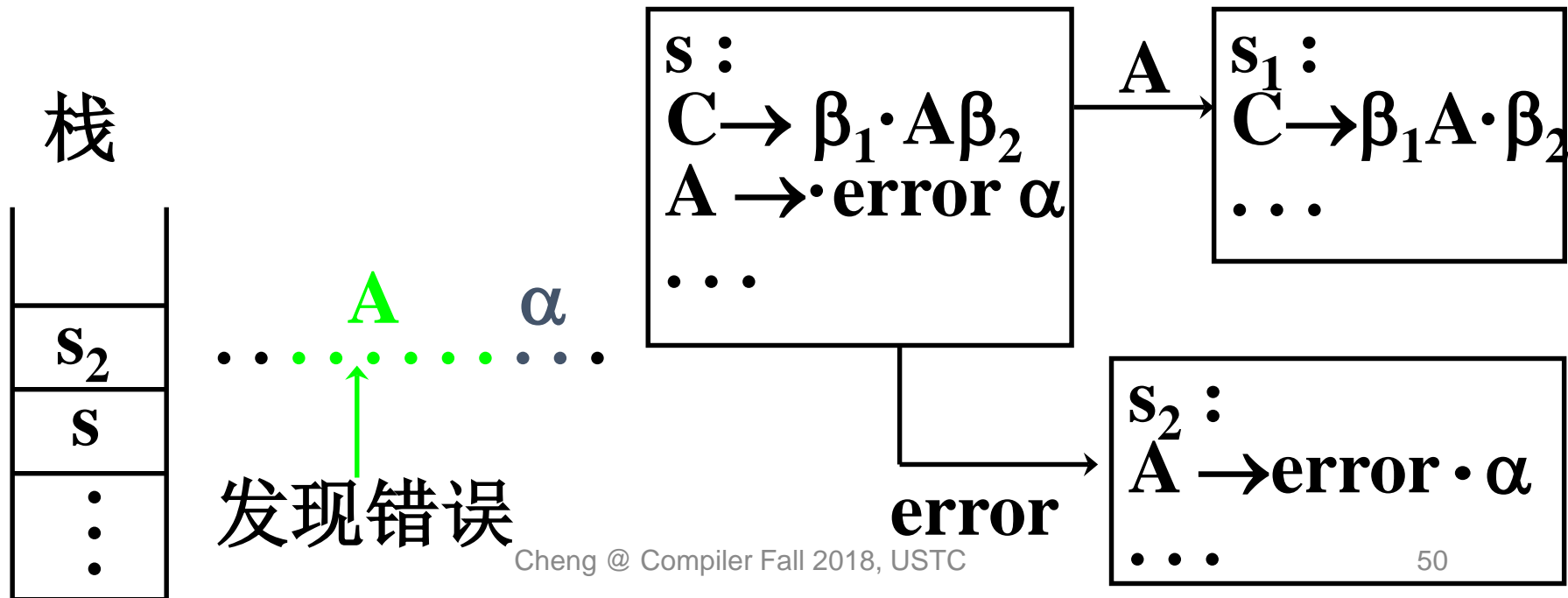
遇到语法错误时

❖ 从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \cdot \text{error } \alpha$ 的项目为止



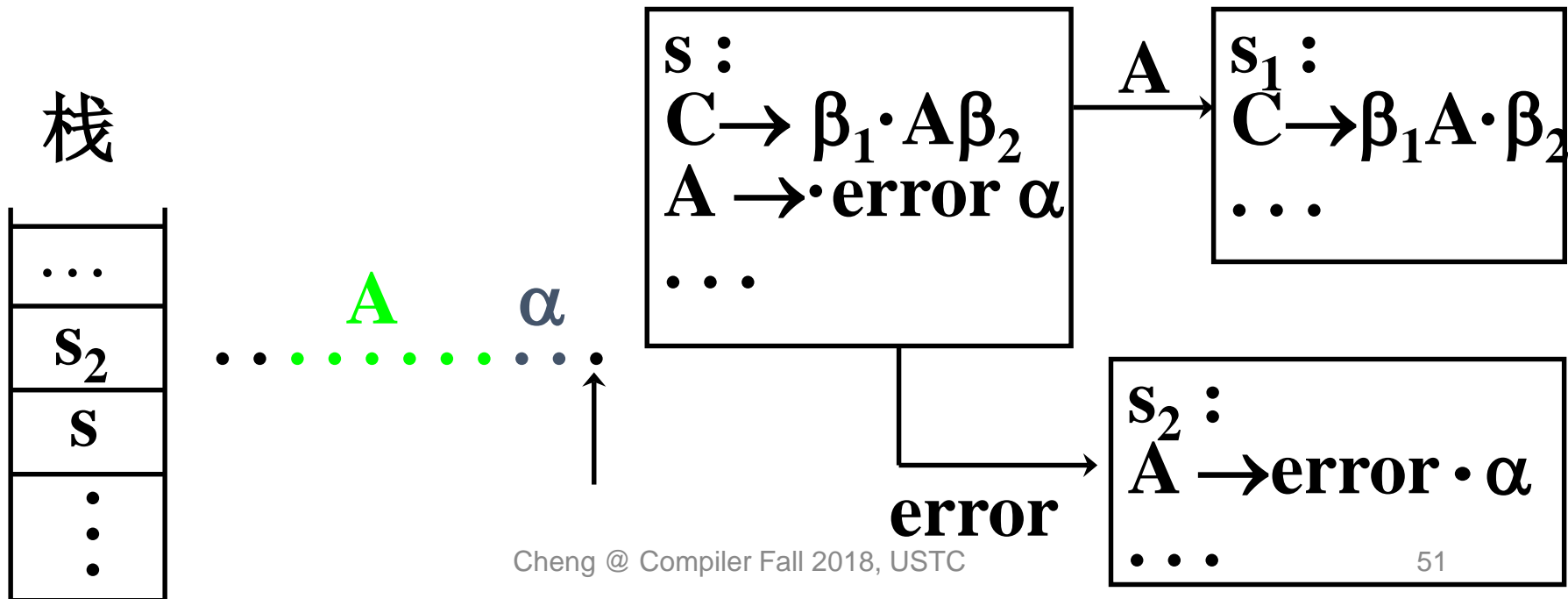
□遇到语法错误时

- ❖从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \cdot \text{error } \alpha$ 的项目为止
- ❖把虚构的终结符 error “移进”栈



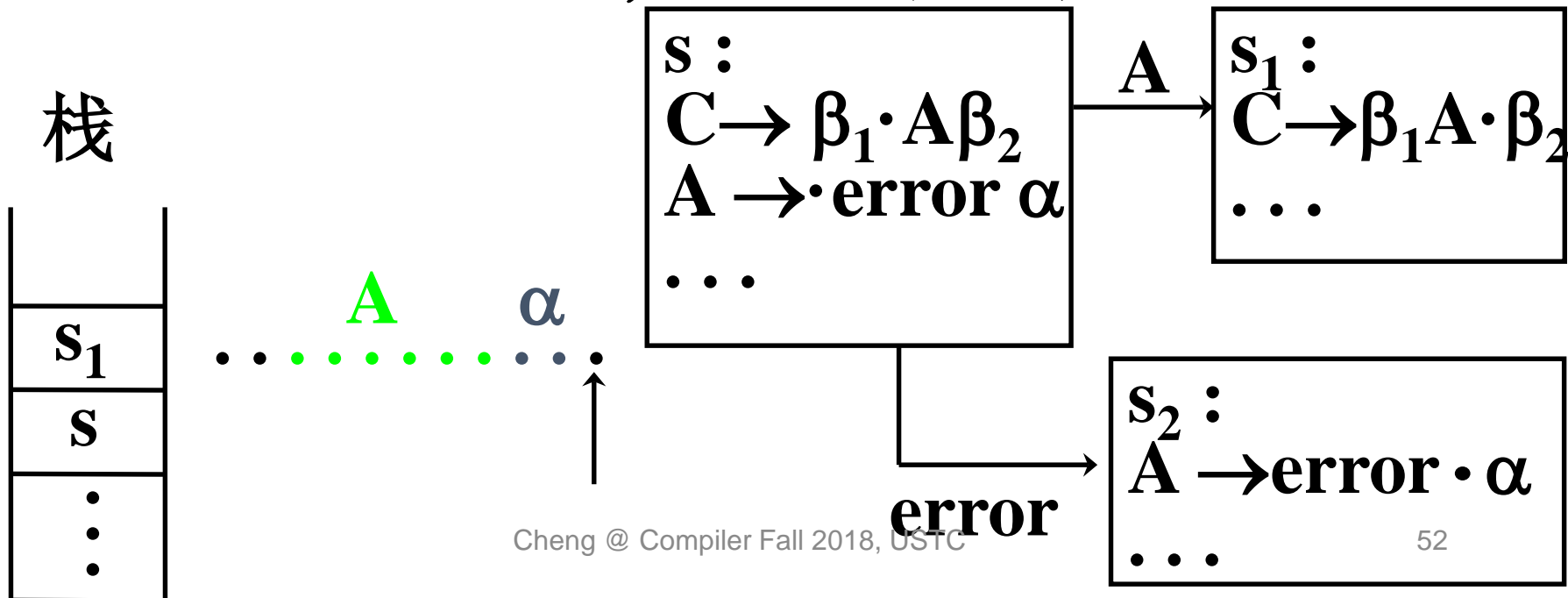
遇到语法错误时

- ❖ 从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \cdot \text{error } \alpha$ 的项目为止
- ❖ 把虚构的终结符 error “移进” 栈
- ❖ 忽略若干输入符号，直至找到 α ，把 α 移进栈



遇到语法错误时

- ❖ 从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \cdot \text{error } \alpha$ 的项目为止
- ❖ 把虚构的终结符 error “移进” 栈
- ❖ 忽略若干输入符号，直至找到 α ，把 α 移进栈
- ❖ 把 $\text{error } \alpha$ 归约为 A ，恢复正常分析



□增加错误恢复的简单计算器

```
lines : lines expr '\n'    {printf ( “%g \n”, $2 ) }  
      | lines '\n'  
      | /* ε */  
      | error '\n' {yyerror ( “重新输入上一行” );  
                  yyerrok;}  
      ;
```



《编译原理与技术》

语法分析 V

At least for the people who send me mail about a new language that they're designing, the general advice is: do it to learn about how to write a compiler.

—— *Dennis Ritchie*