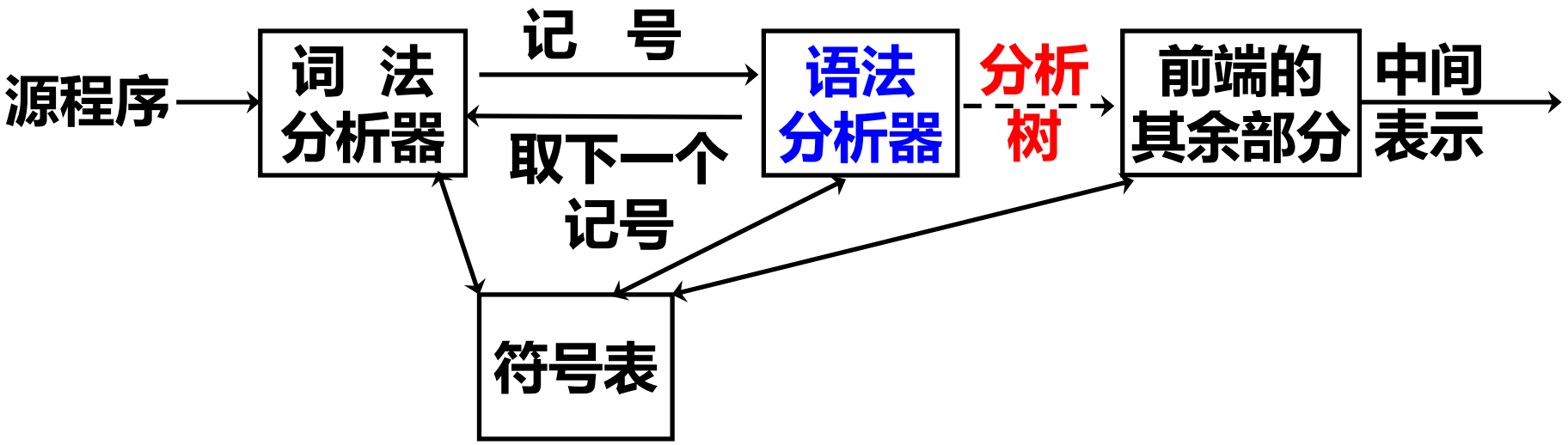
中国科学技术大学
University of Science and Technology of China
USTC, CHINA
ADSLAB
先进数据系统实验室

《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

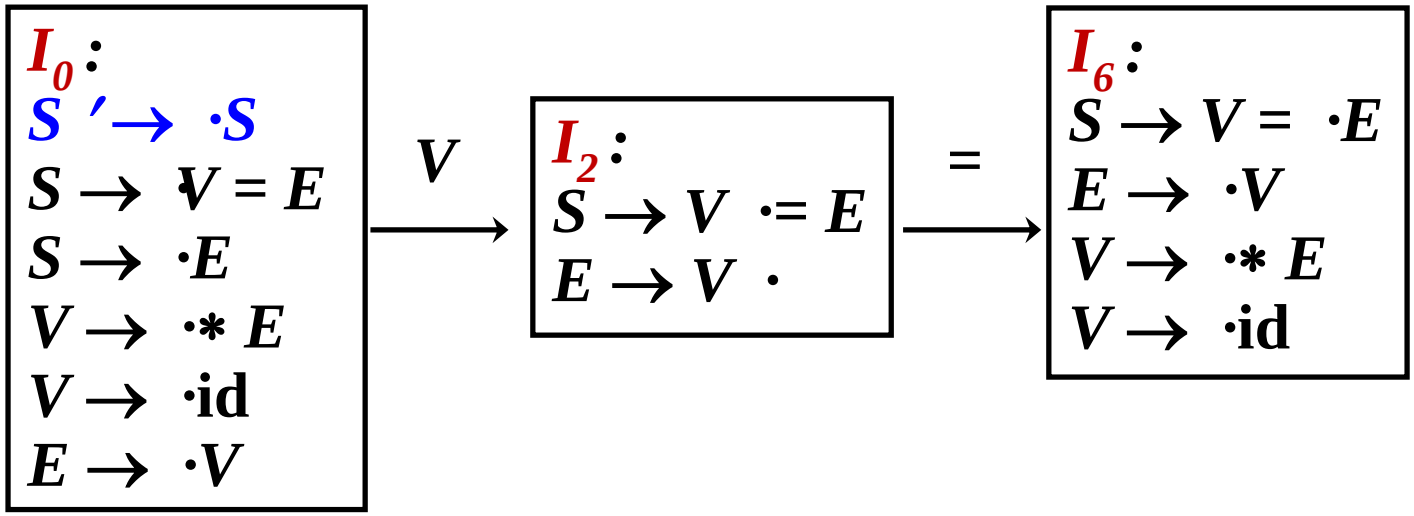
主要内容



- LR(k)分析技术
  - 规范的LR方法、向前看的LR方法
  - 二义性文法的使用、错误恢复
  - 分析器的生成器(Yacc)
- 语法分析技术总结

SLR(1)文法的描述能力有限

$S \to V = E$
$S \to E$
$V \to * E$
$V \to \mathbf{id}$
$E \to V$



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

项目$S \to V \cdot = E$使得
action[2, =] = s6

项目$E \to V \cdot$ 使得
action[2, =] = r5
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
  - closure(I) 和GOTO函数需要修改
- 添加了前向搜索符
  - 一个项目$A \rightarrow \alpha\beta$，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：$A \rightarrow \alpha\beta,\ a$
- 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$\gamma$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$

规范的LR分析：举例

❑例 $S \to BB$

$B \to bB \mid a$

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

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

对于项目$[A \to \alpha \cdot \beta, a]$，当$\beta$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

规范的LR分析

❑构造LR(1)项目集规范族

- ❖初始项目集I_0:

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

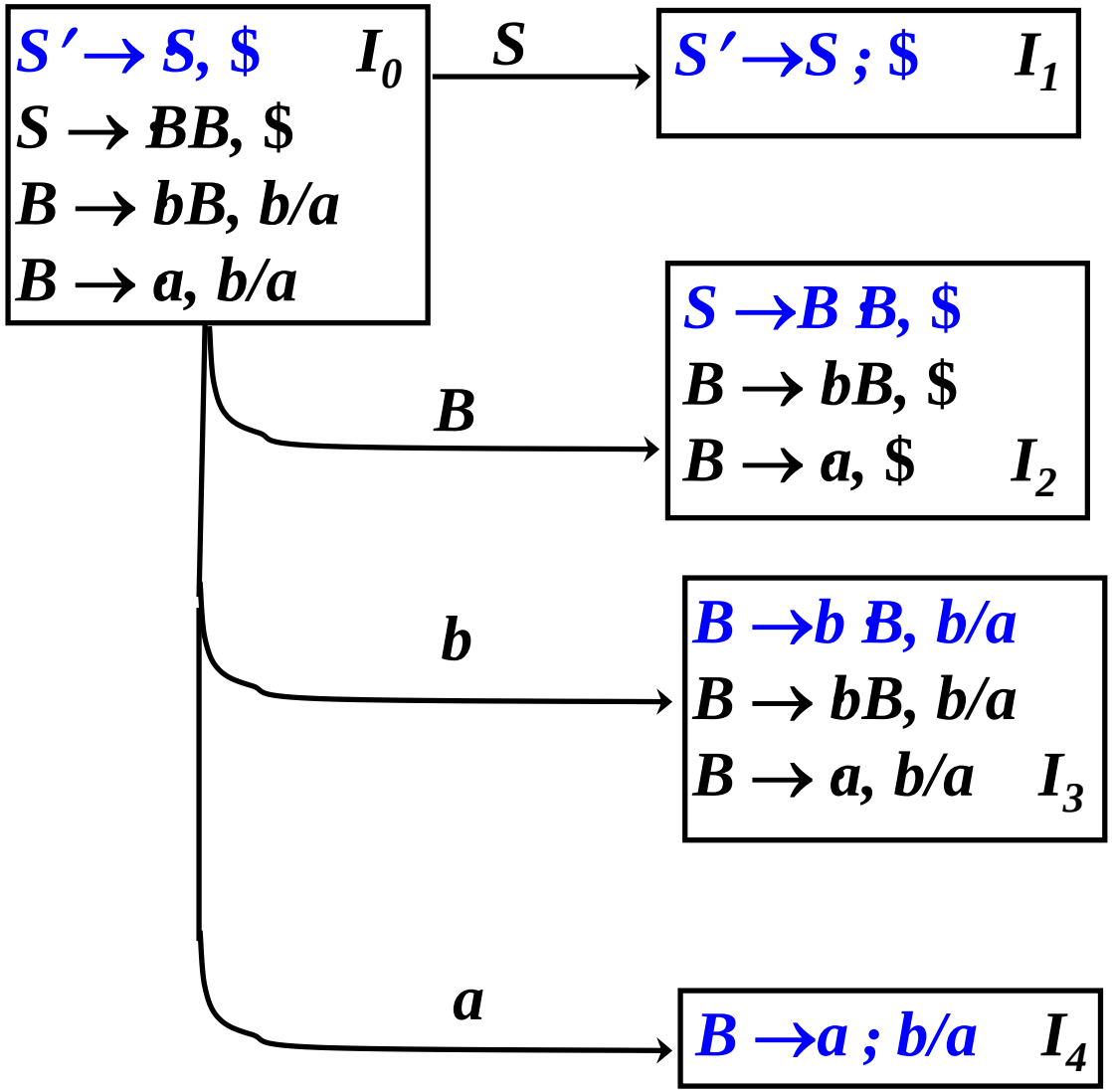
保证在用$B \rightarrow \gamma$进行归约后，

- 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。

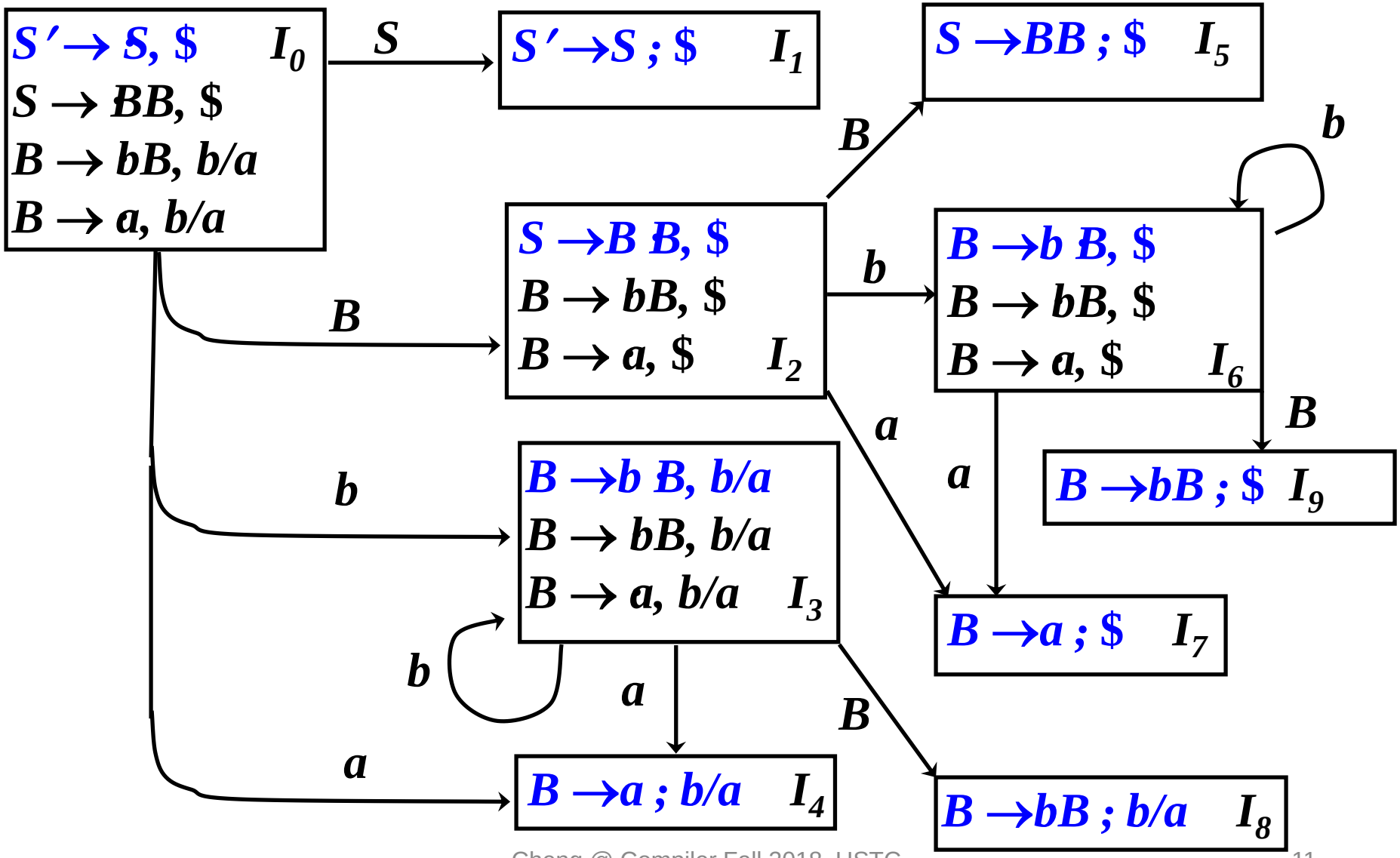
规范的LR分析：举例

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

规范的LR分析：举例


S′→S, $ I0
S→BB, $
B→bB, b/a
B→a, b/a
S
S′→S ; $ I1
B
S→B B, $
B→bB, $
B→a, $ I2
b
B→b B, b/a
B→bB, b/a
B→a, b/a I3
a
B→a ; b/a I4

规范的LR分析：举例


S′→S, $ I0
S → BB, $
B → bB, b/a
B → a, b/a
S
S′→S ; $ I1
B
S →BB ; $ I5
B
S →B B, $
B → bB, $
B → a, $ I2
b
B →b B, $
B → bB, $
B → a, $ I6
b
B
B →bB ; $ I9
a
a
b
B →b B, b/a
B → bB, b/a
B → a, b/a I3
B →a ; $ I7
b
a
B
a
B →a ; b/a I4
B →bB ; b/a I8

构造规范的LR分析表

- 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$[A \rightarrow \alpha\ a\beta, b]$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$[S' \rightarrow S\ ;\ \$]$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

构造规范的LR分析表

- 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$goto[i, A] = j$

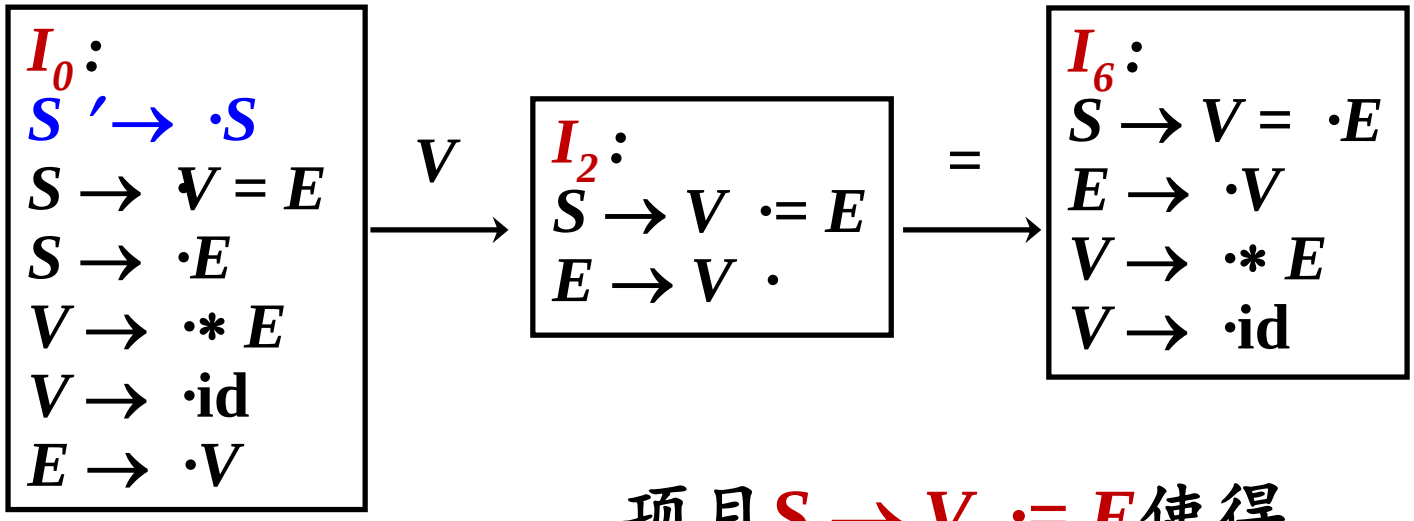
构造规范的LR分析表

- 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

非SLR(1)文法

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



项目 $S \to V \cdot = E$ 使得
action[2, =] = s6

项目 $E \to V \cdot$ 使得
action[2, =] = r5
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

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

I_0:
$S' \to \cdot S, \$$
$S \to \cdot V = E, \$$
$S \to \cdot E, \$$
$V \to \cdot * E, =/\$$
$V \to \cdot \text{id}, =/\$$
$E \to \cdot V, \$$

$V \longrightarrow$

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

无移进归约冲突

计算闭包：

定义里：$[A \to \alpha \cdot B \beta, a]$

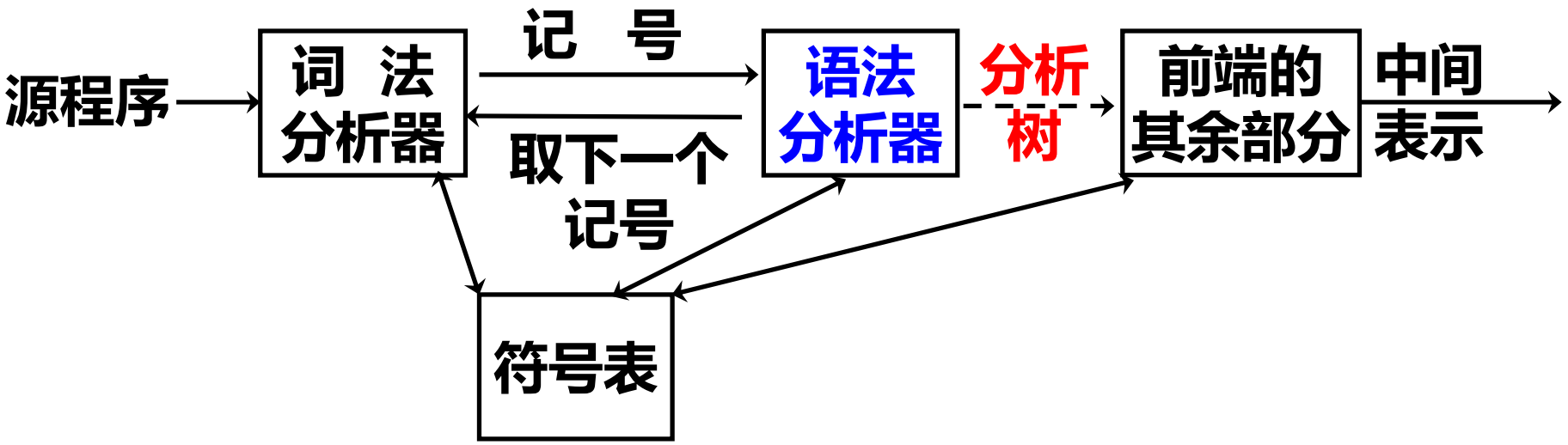
这里：$[S \to \cdot V = E, \$]$

FIRST(βa)

⇓

FIRST(=E$)={=}

主要内容

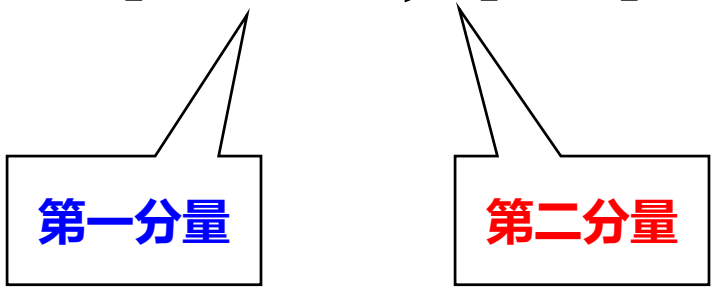


- LR(k)分析技术
  - 规范的LR方法、向前看的LR方法
  - 二义性文法的使用、错误恢复
  - 分析器的生成器(Yacc)
- 语法分析技术总结

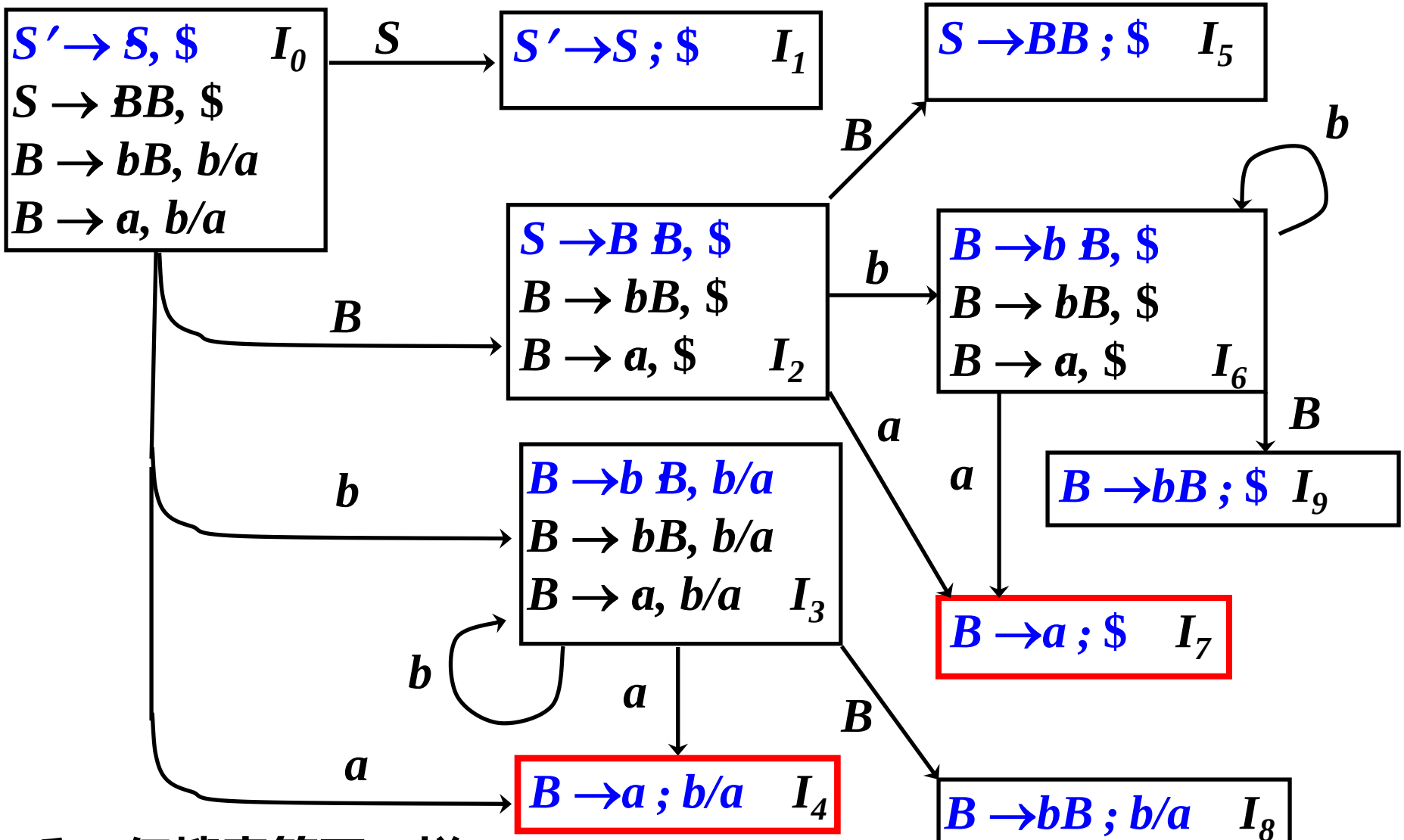
LALR分析方法

- 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中的相同核心项目集(同心项目集)
- 同心的LR(1)项目集
  - 核心：项目集中第一分量的集合
  - 略去搜索符后它们是相同的集合
  - 例： [B → ·bB, $] 与 [B → ·bB, b/a]

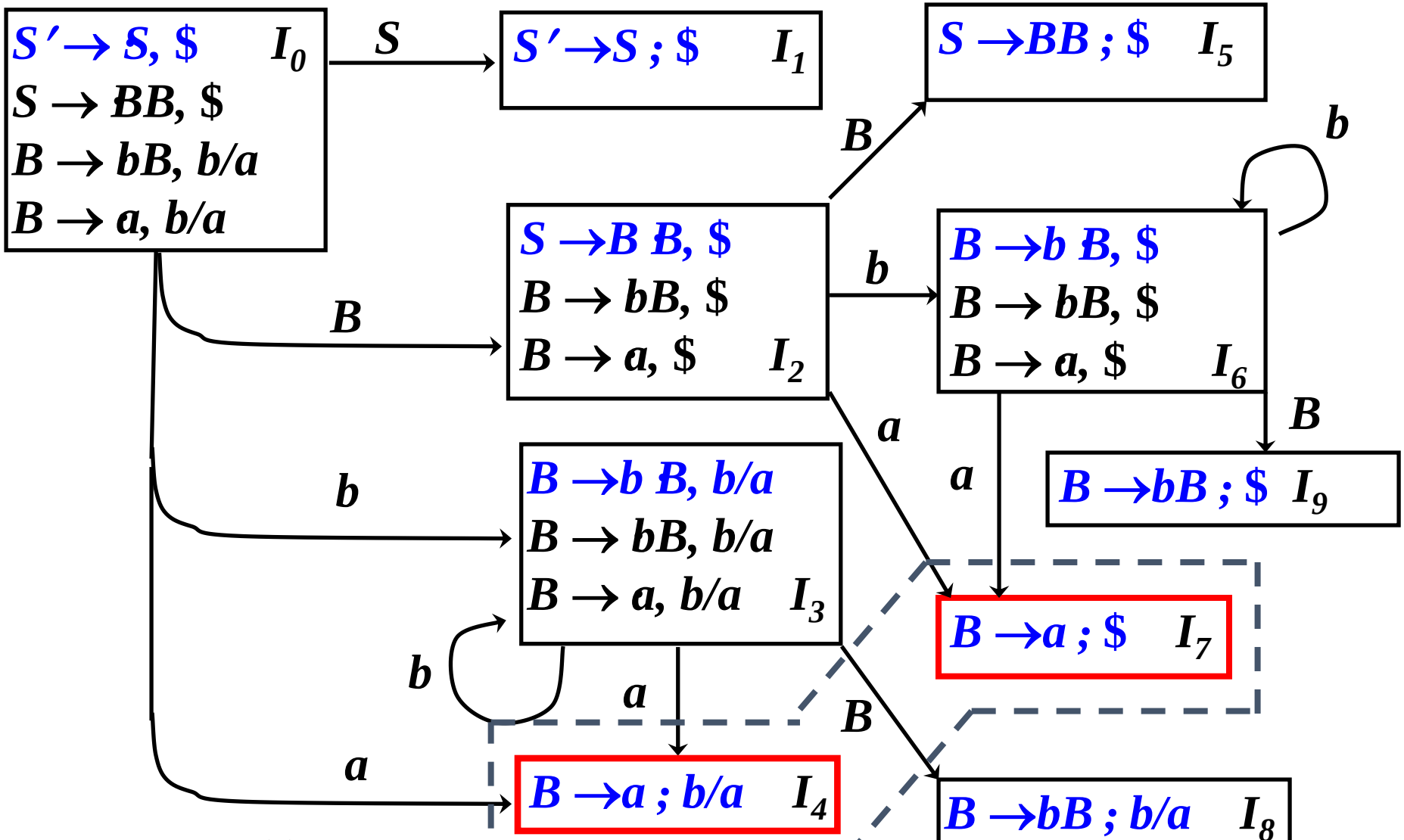


识别活前缀的DFA



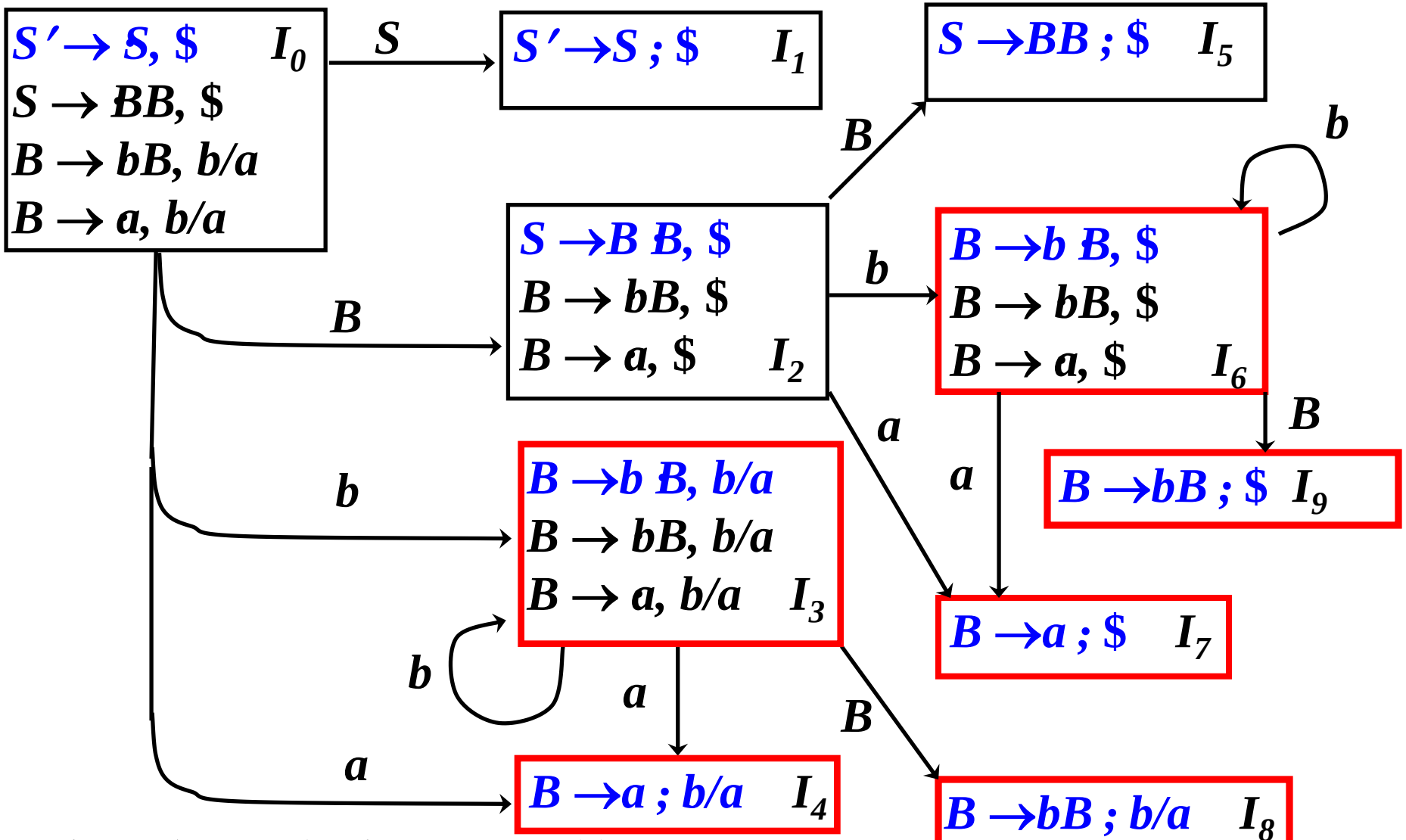
I_4和I_7仅搜索符不一样

合并同心项目集



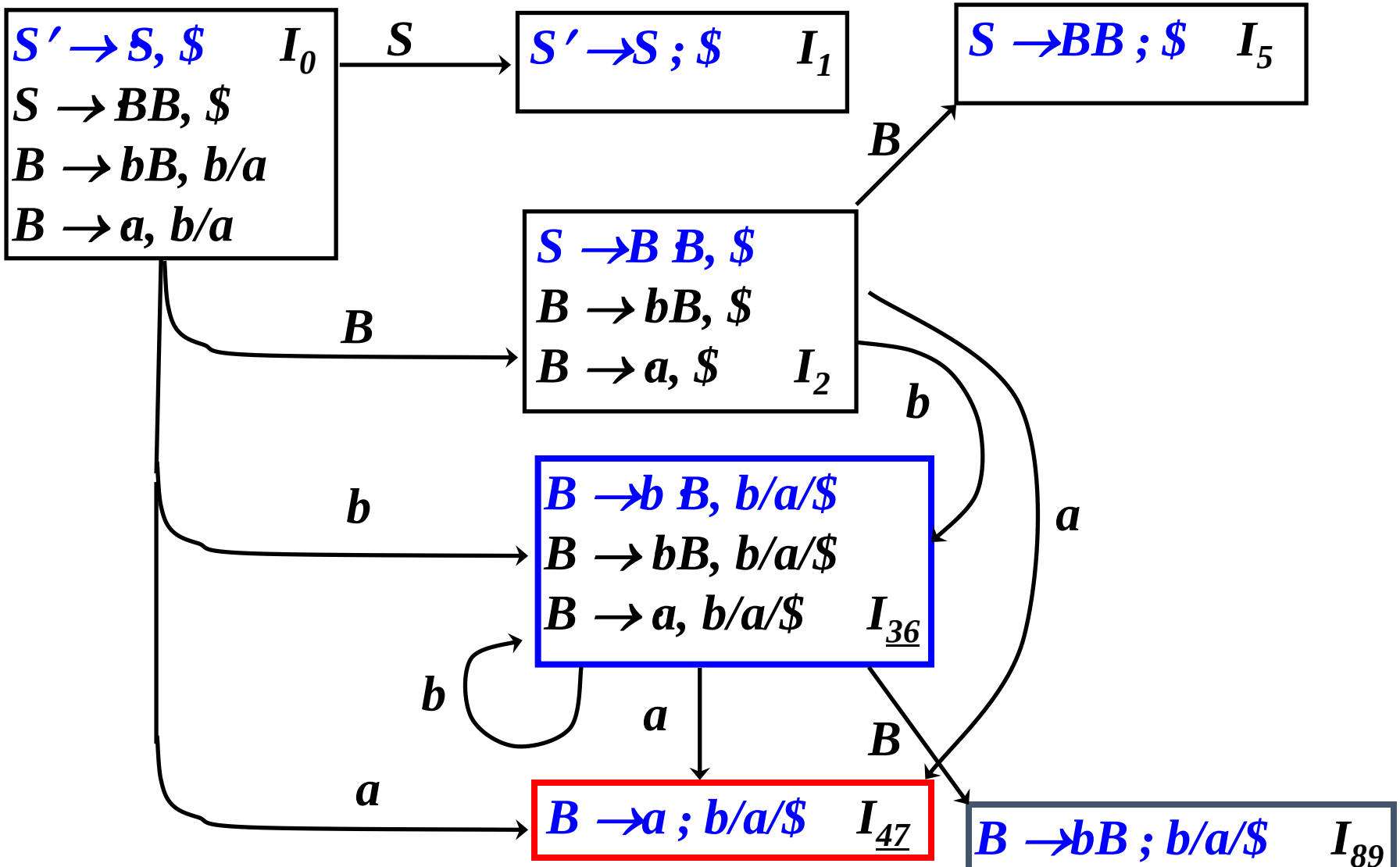
I_4和I_7合并

合并同心项目集

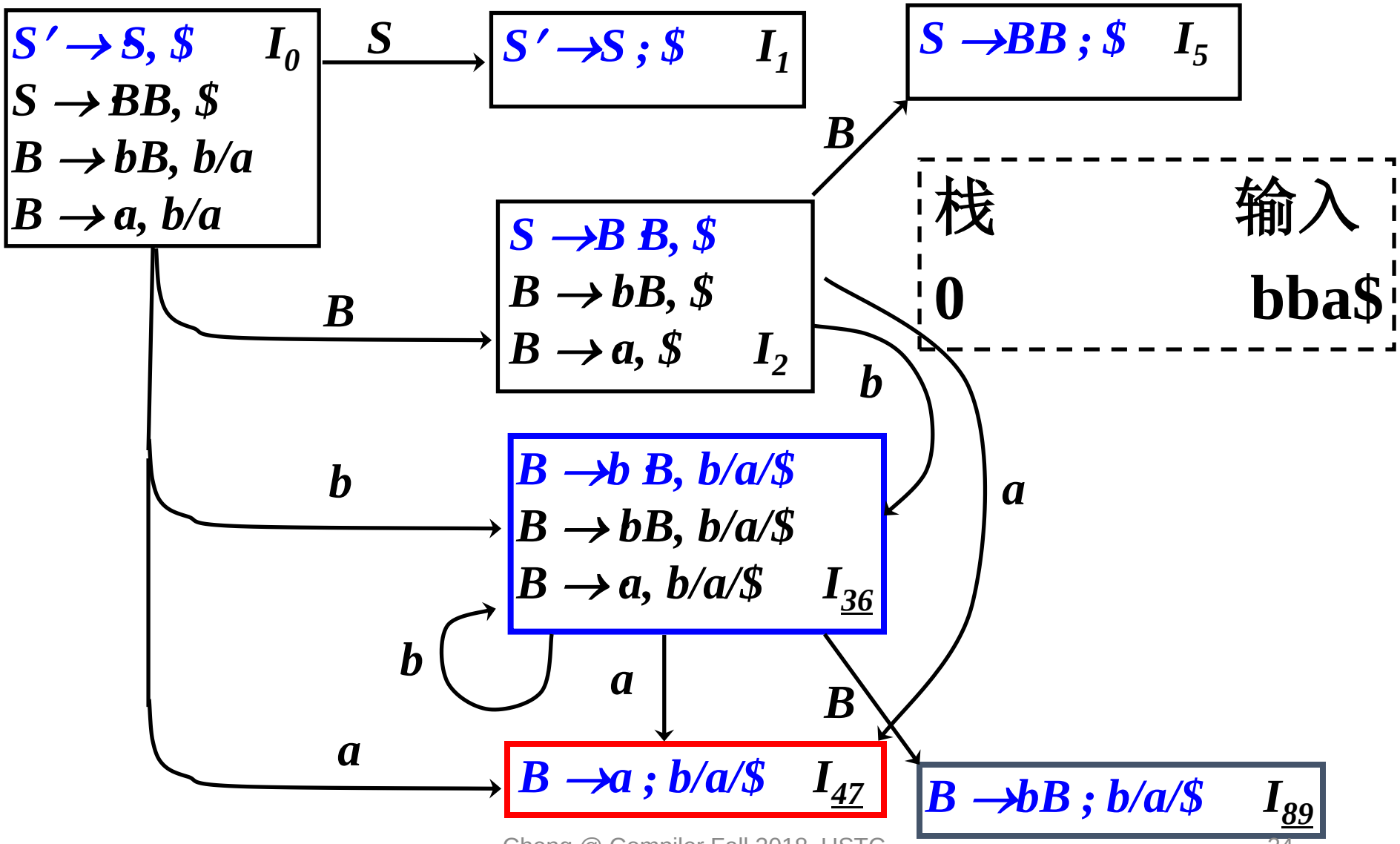


有三组同心集，都合并

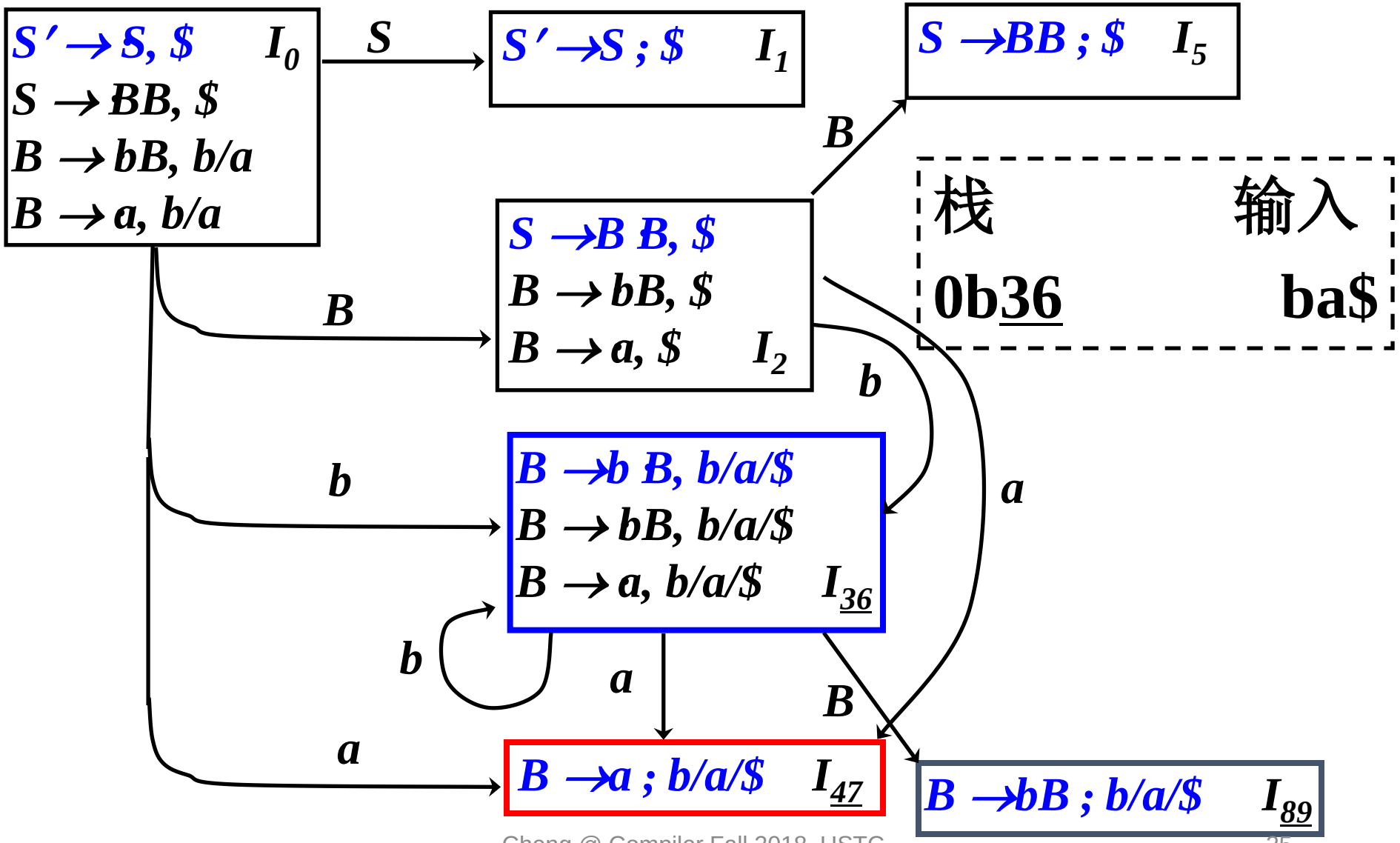
合并同心项目集


S′ → S, $ I0
S → BB, $
B → bB, b/a
B → a, b/a
S
S′ → S ; $ I1
S → BB ; $ I5
B
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
a
b
a
B
a
B → a ; b/a/$ I47
B → bB ; b/a/$ I89

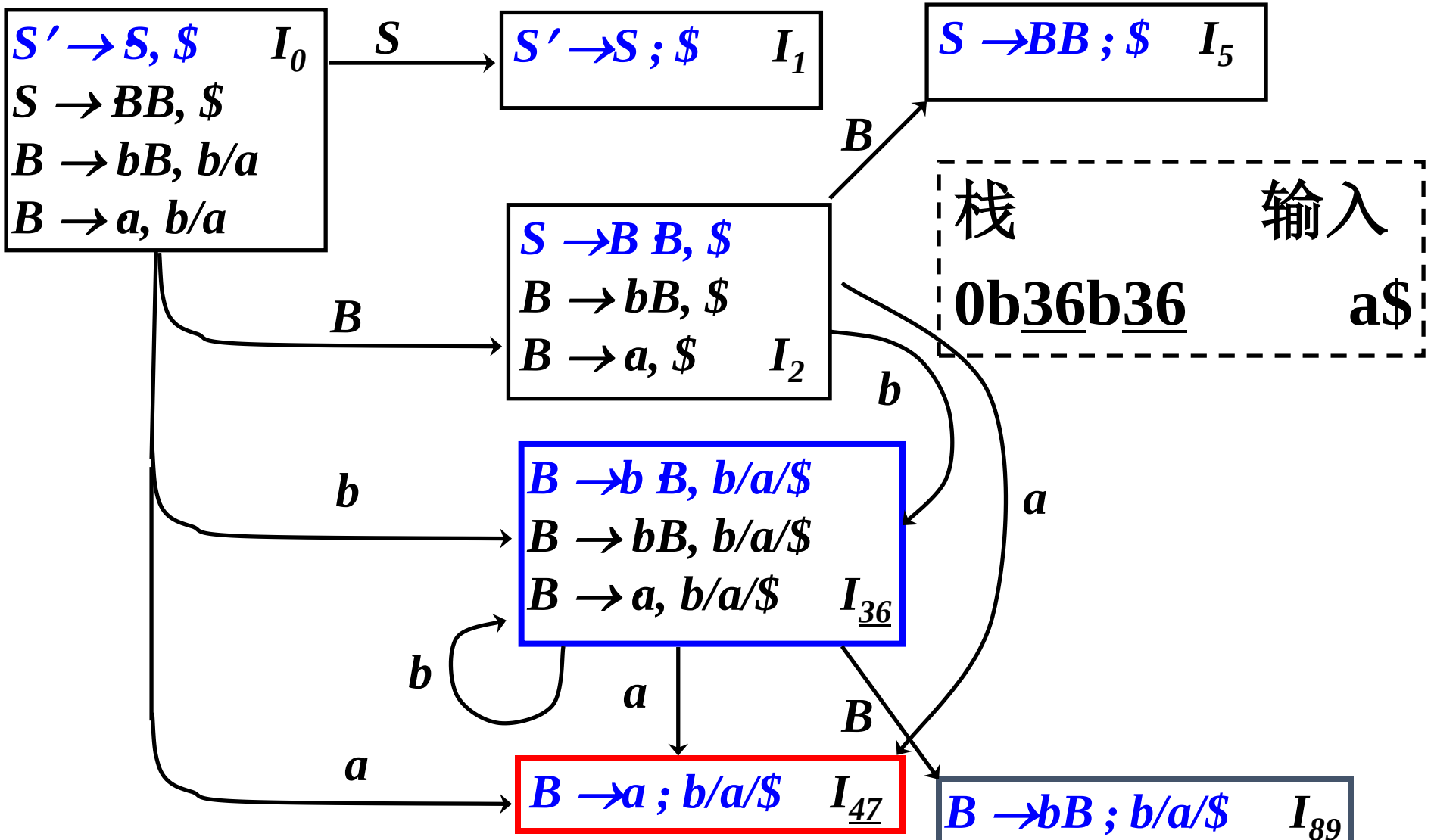
LALR分析：举例


S′ → S, $ I0
S → BB, $
B → bB, b/a
B → a, b/a
S
S′ → S ; $ I1
S → BB ; $ I5
B
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
a
b
a
B
a
B → a ; b/a/$ I47
B → bB ; b/a/$ I89
栈 输入
0 bba$

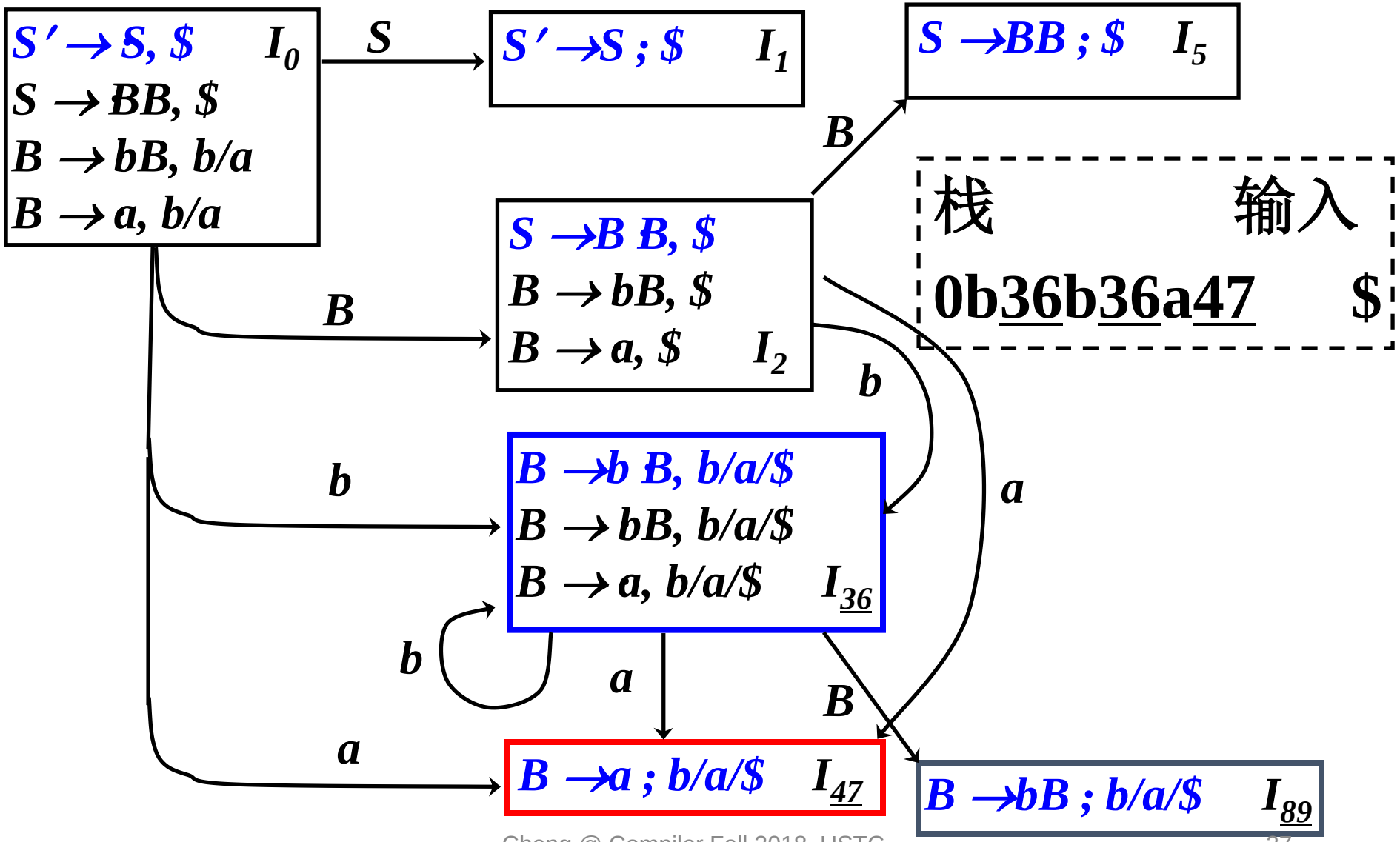
LALR分析：举例


S′ → S, $
S → BB, $
B → bB, b/a
B → a, b/a
I0
S
S′ → S ; $
I1
S → BB ; $
I5
B
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
a
b
a
B
a
B → a ; b/a/$
I47
B → bB ; b/a/$
I89
栈
输入
0b36
ba$

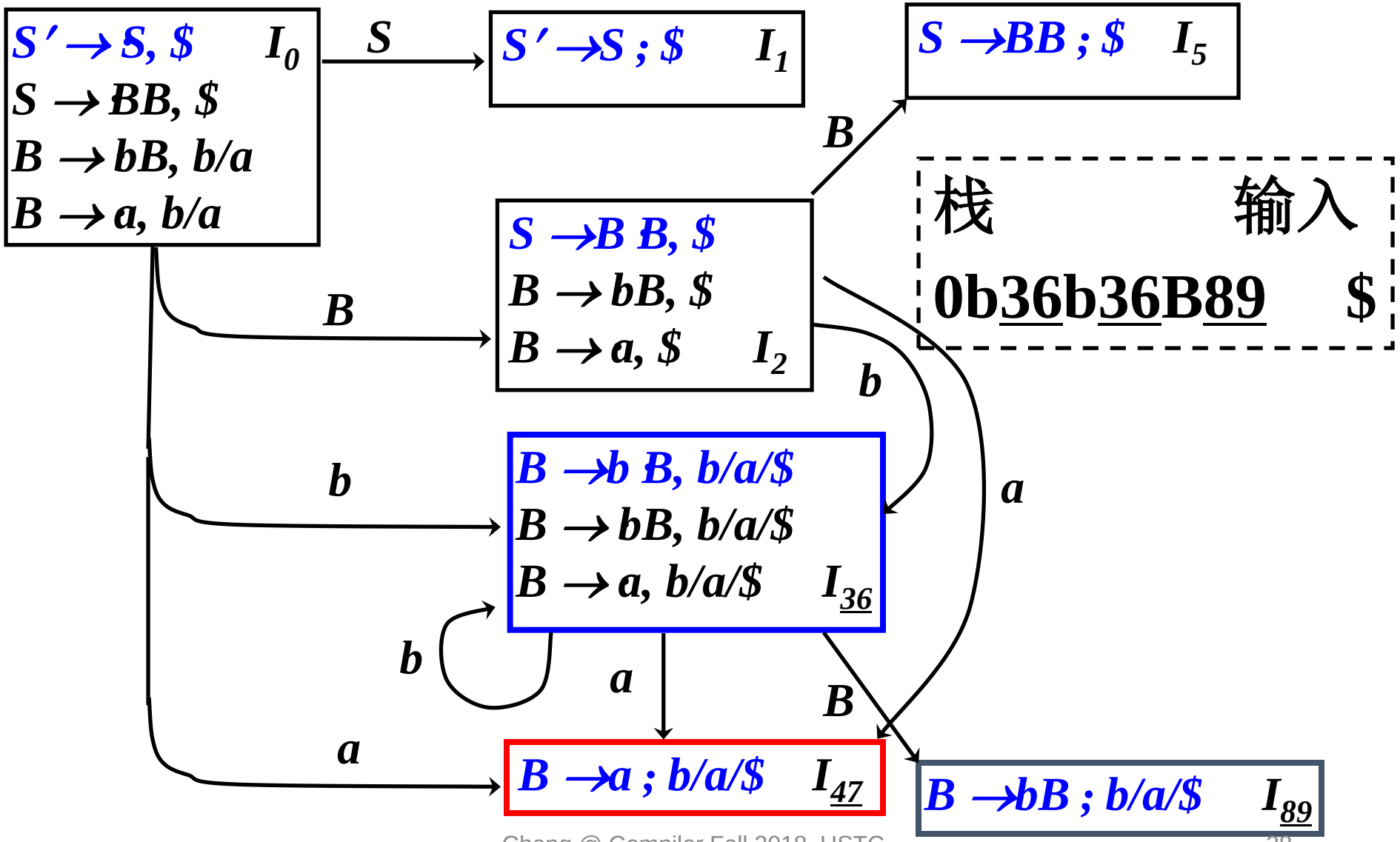
LALR分析：举例


S′ → S, $ I0
S → BB, $
B → bB, b/a
B → a, b/a
S
S′ → S ; $ I1
S → BB ; $ I5
B
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
a
b
a
B
a
B → a ; b/a/$ I47
B → bB ; b/a/$ I89
栈 输入
0b36b36 a$

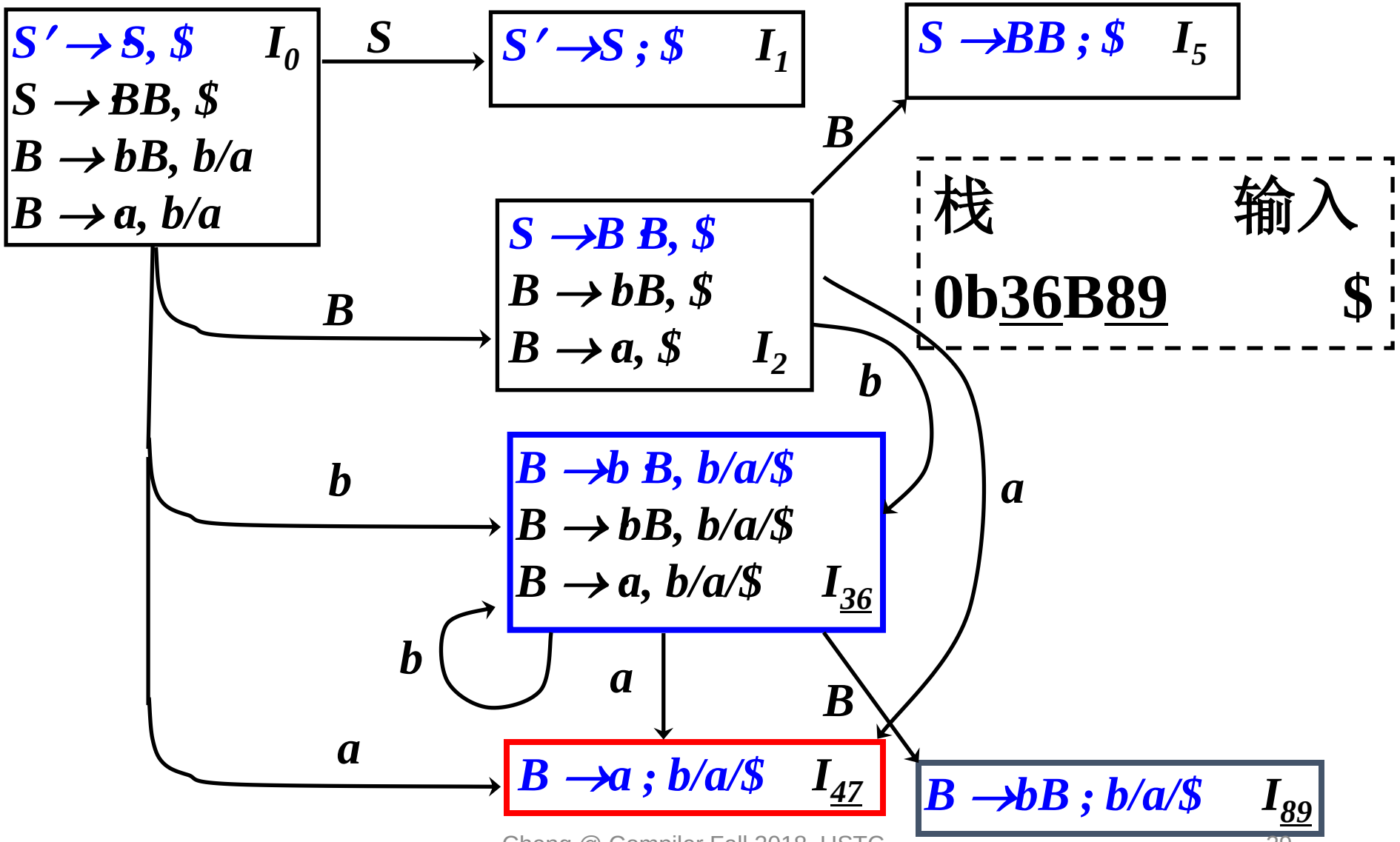
LALR分析：举例


S′ → S, $
S → BB, $
B → bB, b/a
B → a, b/a
I0
S
S′ → S ; $
I1
S → BB ; $
I5
B
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
a
b
a
B
a
B → a ; b/a/$
I47
B → bB ; b/a/$
I89
栈 输入
0b36b36a47 $

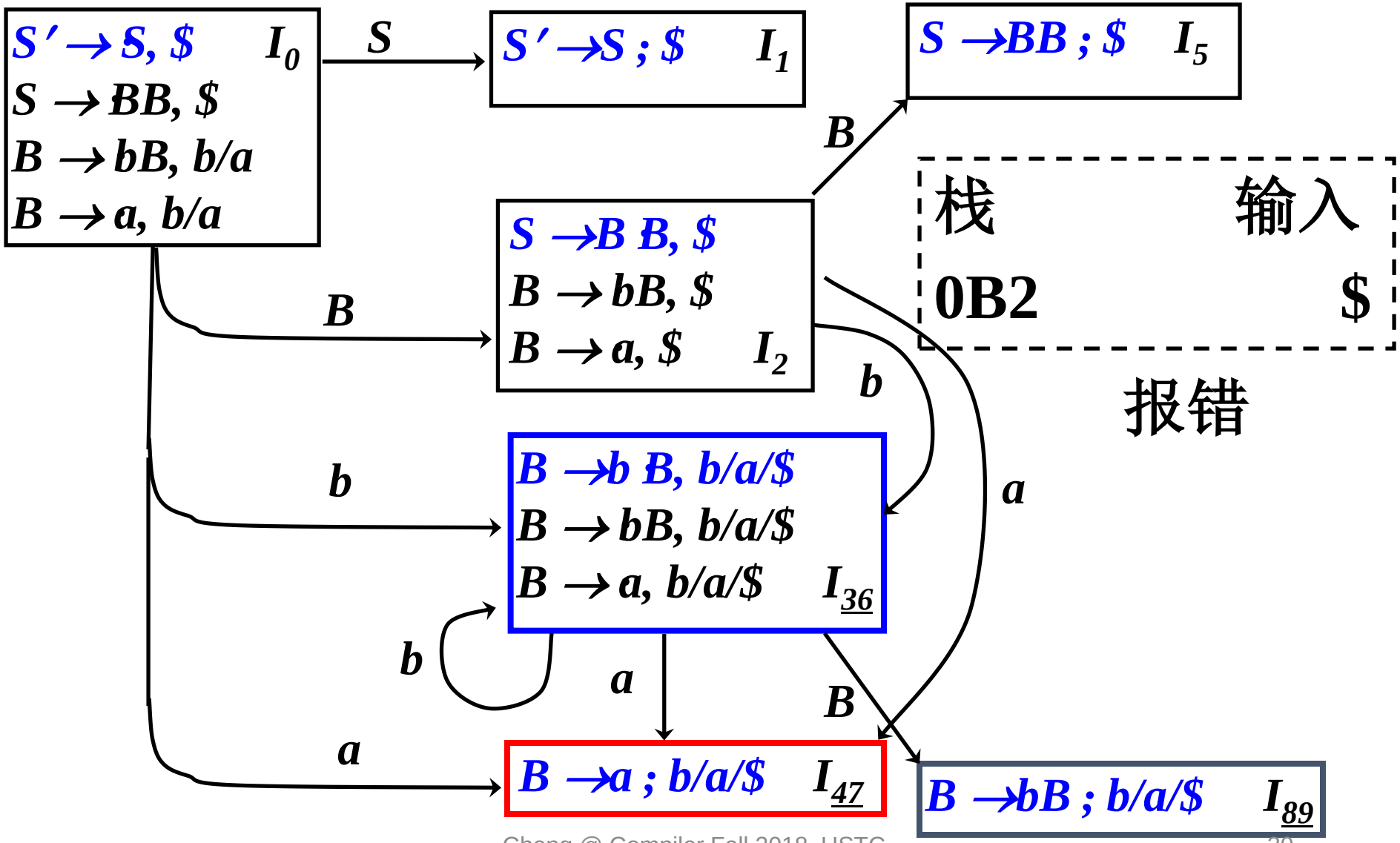
LALR分析：举例


S′ → S, $ I0
S → BB, $
B → bB, b/a
B → a, b/a
S
S′ → S ; $ I1
B
S → BB ; $ I5
B
S → B B, $
B → bB, $
B → a, $ I2
b
a
b
B → b B, b/a/$
B → bB, b/a/$
B → a, b/a/$ I36
b
a
B
a
B → a ; b/a/$ I47
B → bB ; b/a/$ I89
栈 输入
0b36b36B89 $

LALR分析：举例


S′ → S, $ I0
S → BB, $
B → bB, b/a
B → a, b/a
S
S′ → S ; $ I1
B
S → BB ; $ I5
B
S → B B, $
B → bB, $
B → a, $ I2
b
a
b
B → b B, b/a/$
B → bB, b/a/$
B → a, b/a/$ I36
b
a
B
a
B → a ; b/a/$ I47
B → bB ; b/a/$ I89
栈 输入
0b36B89 $

LALR分析：举例


S′ → S, $
S → BB, $
B → bB, b/a
B → a, b/a
I0
S
S′ → S ; $
I1
S → BB ; $
I5
B
B
S → B B, $
B → bB, $
B → a, $
I2
b
a
b
B → b B, b/a/$
B → bB, b/a/$
B → a, b/a/$
I36
b
a
B
a
B → a ; b/a/$
I47
B → bB ; b/a/$
I89
栈
输入
0B2
$
报错

2、构造LALR(1)分析表

- 构造LR(1)项目集规范族$C = \{I_0, I_1, \ldots, I_n\}$
- 构造LALR(1)项目集规范族$C' = \{J_0, J_1, \ldots, J_k\}$，其中任意项目集$J_i = I_n \cup I_m \cup \ldots \cup I_t$
  - $I_n, I_m, \ldots, I_t \in C$且具有共同的核心
- 按构造规范LR(1)分析表的方式构造分析表

如没有语法分析动作冲突，那么给定文法就是**LALR(1)文法**

LALR分析方法

- 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

$[A \rightarrow \alpha \ ; a]$

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

. . .

项目集2

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

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

. . .

则合并前就有冲突

❑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

$S' \to S$

$S \to aAd \mid bBd \mid aBe \mid bAe$

$A \to c$

$B \to c$

对ac有效的项目集

$A \to c \ ; d$

$B \to c \ ; e$

对bc有效的项目集

$A \to c \ ; e$

$B \to c \ ; d$

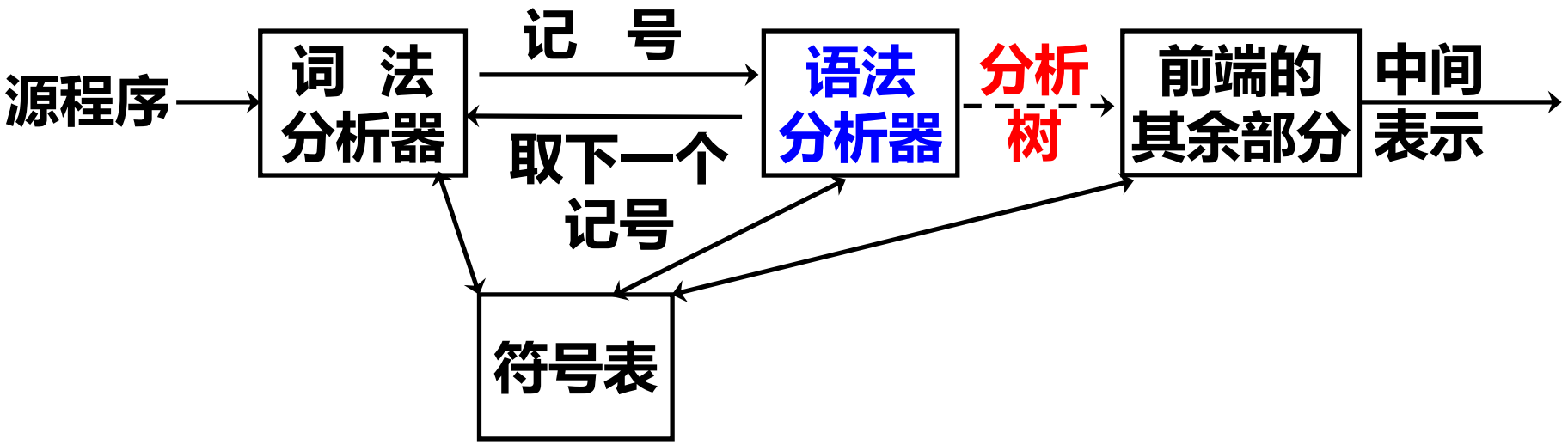
合并同心集后

$A \to c \ ; d/e$

$B \to c \ ; d/e$

该文法是LR(1)的
但不是LALR(1)的

主要内容



- ❑LR(k)分析技术
  - ❖规范的LR方法、向前看的LR方法
  - ❖二义性文法的使用、错误恢复
  - ❖分析器的生成器(Yacc)
- ❑语法分析技术总结

二义文法的特点

- 二义文法决不是LR文法
- 简洁、自然

  二义文法：$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

  非二义的文法：

  $E \rightarrow E + T \mid T$

  $T \rightarrow T * F \mid F$

  $F \rightarrow (E) \mid \mathbf{id}$
- 可以用文法以外的信息来消除二义
- 语法分析的效率高（基于消除二义后得到的分析表）

二义文法的应用

- 使用文法以外信息来解决分析动作冲突
  - 例 二义文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \mathrm{id}$

    规定： *优先级高于+，两者都是左结合

二义文法的应用

- 使用文法以外信息来解决分析动作冲突
  - 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

    规定： *优先级高于+，两者都是左结合

LR(0)项目集 I_7

$E \to E + E \cdot$

$E \to E \cdot + E$

$E \to E \cdot * E$

二义文法的应用

- 使用文法以外信息来解决分析动作冲突
  - 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

    规定： *优先级高于+，两者都是左结合

LR(0)项目集 I_7

$E \to E + E \cdot$

$E \to E \cdot + E$

$E \to E \cdot * E$

id + id + id

面临+，归约

二义文法的应用

❑使用文法以外信息来解决分析动作冲突

❖例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

规定： *优先级高于+，两者都是左结合

LR(0)项目集I_7

$E \to E + E \cdot$

$E \to E \cdot + E$ id + id + id

$E \to E \cdot * E$ id + id * id

面临+，归约 面临*，移进

二义文法的应用

❑使用文法以外信息来解决分析动作冲突

❖例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

规定： *优先级高于+，两者都是左结合

LR(0)项目集I_7

$E \to E + E \cdot$

$E \to E \cdot + E$ id + id + id

$E \to E \cdot * E$ id + id * id

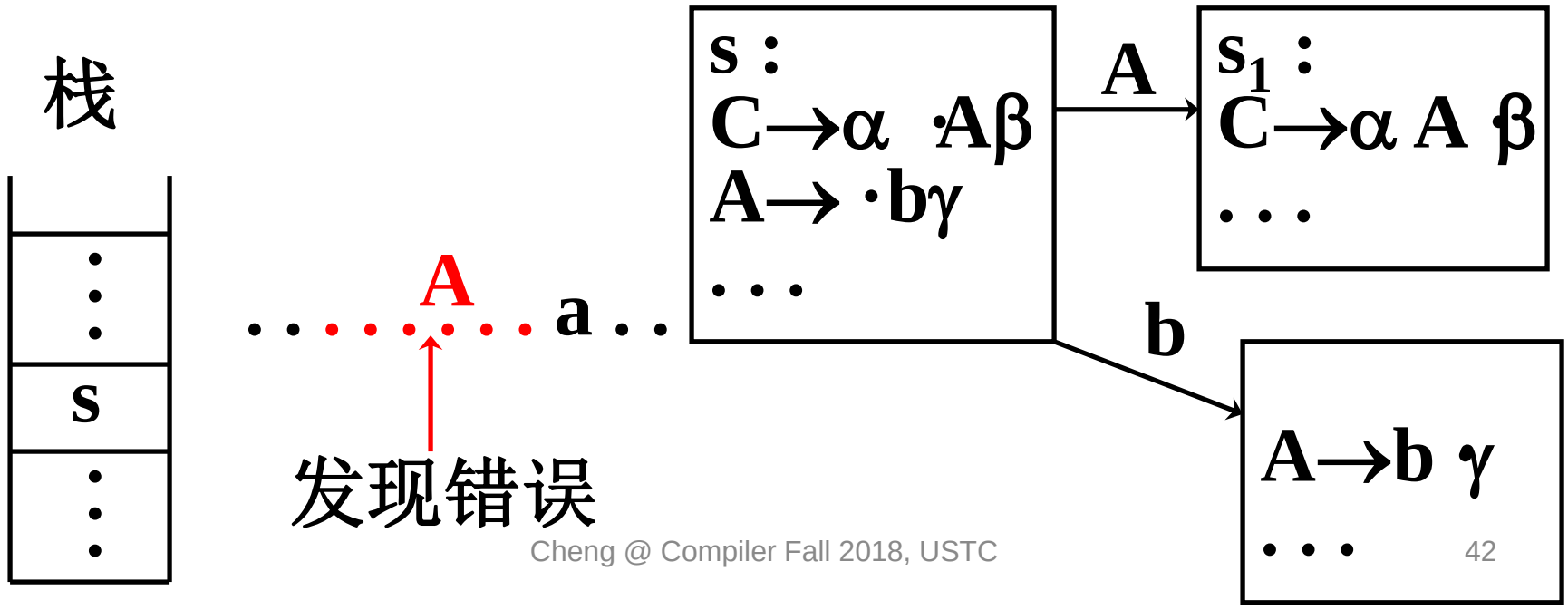
面临+，归约 面临*，移进

面临) 和$，归约

LR分析的错误恢复

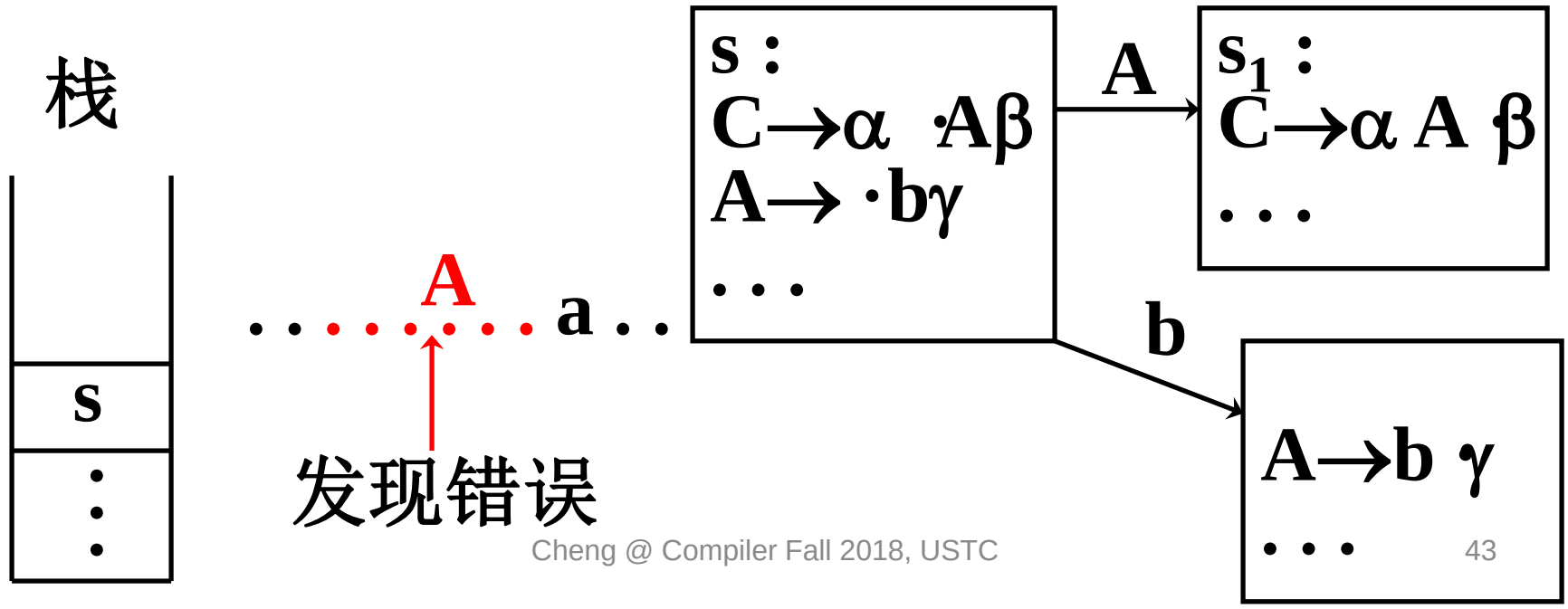
- LR分析器在什么情况下发现错误
  - 访问动作表时若遇到出错条目
  - 访问转移表时它决不会遇到出错条目
  - 决不会把不正确的后继移进栈
  - 规范的LR分析器甚至在报告错误之前决不做任何无效归约

❑紧急方式错误恢复



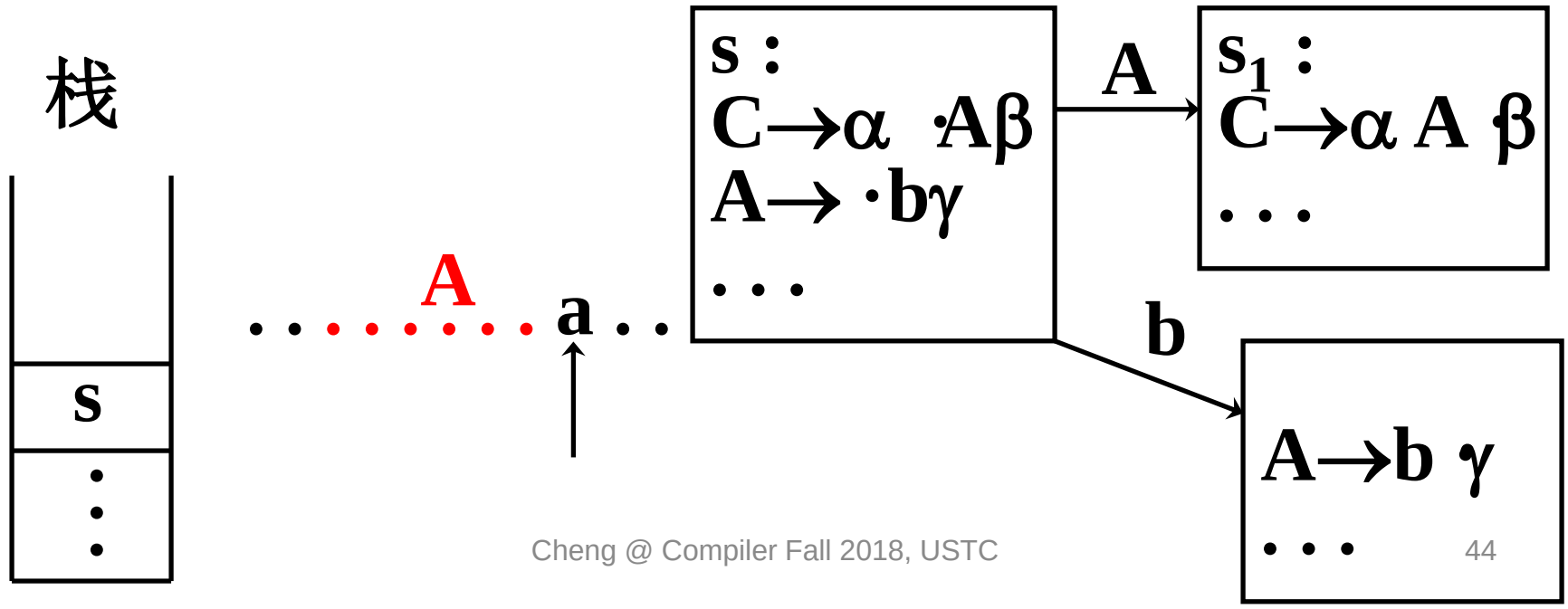
❑紧急方式错误恢复

❖退栈，直至出现状态s，它有预先确定的A的转移



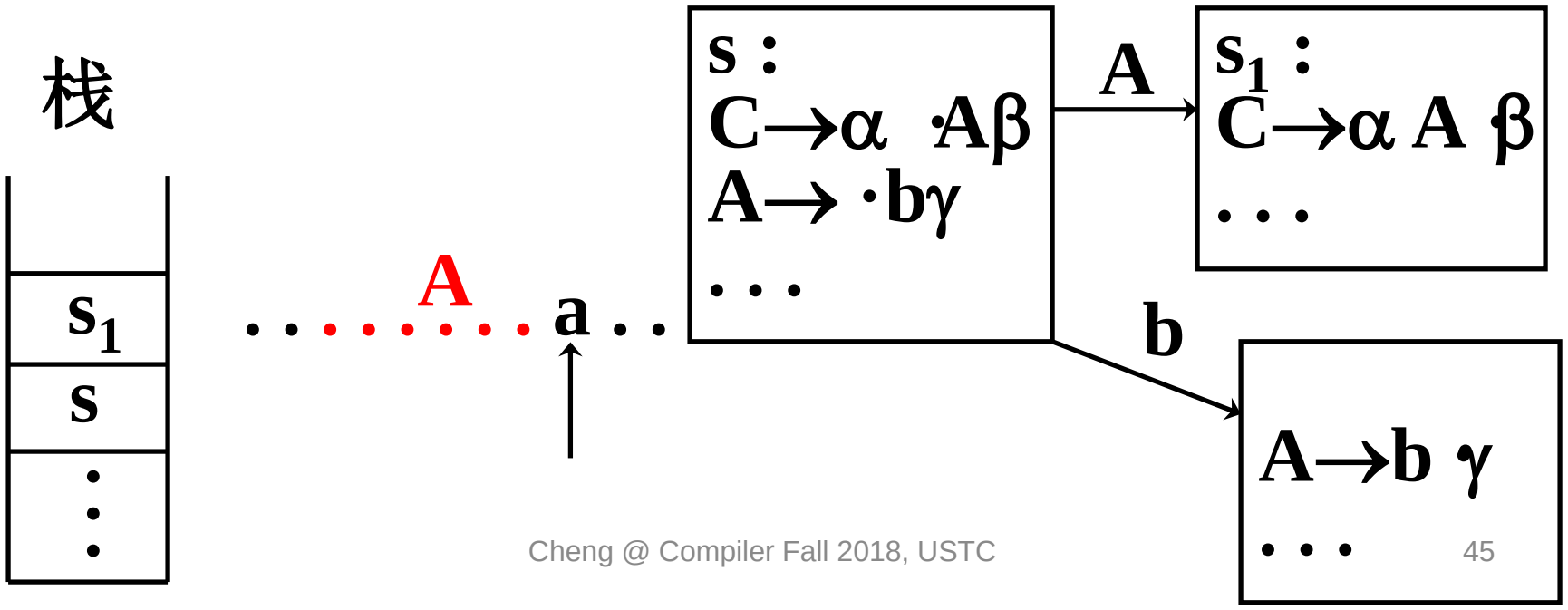
❑紧急方式错误恢复

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

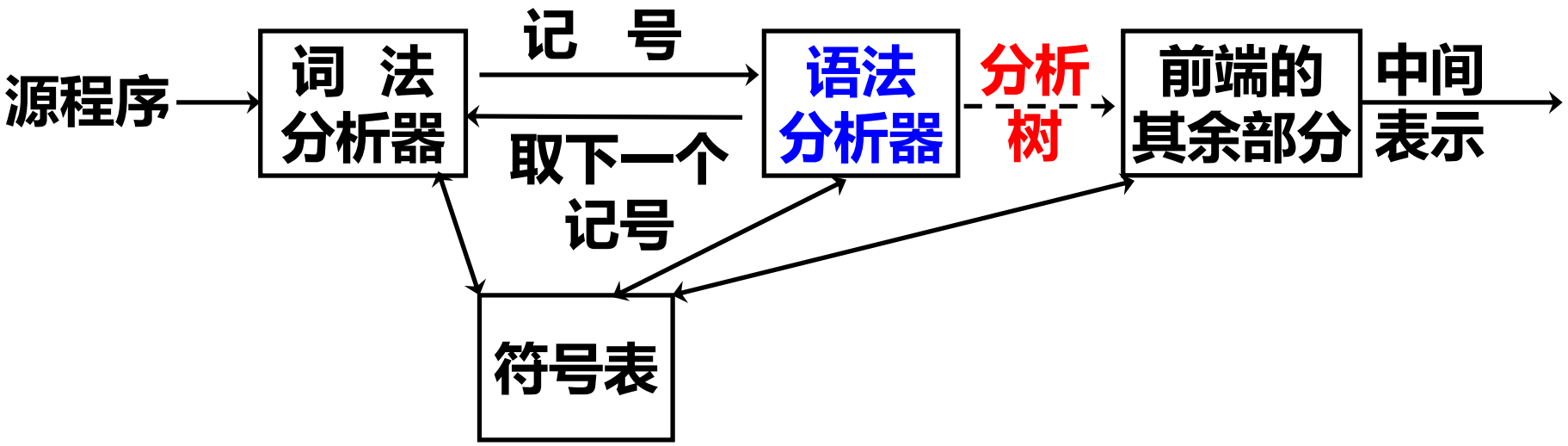


❑紧急方式错误恢复

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

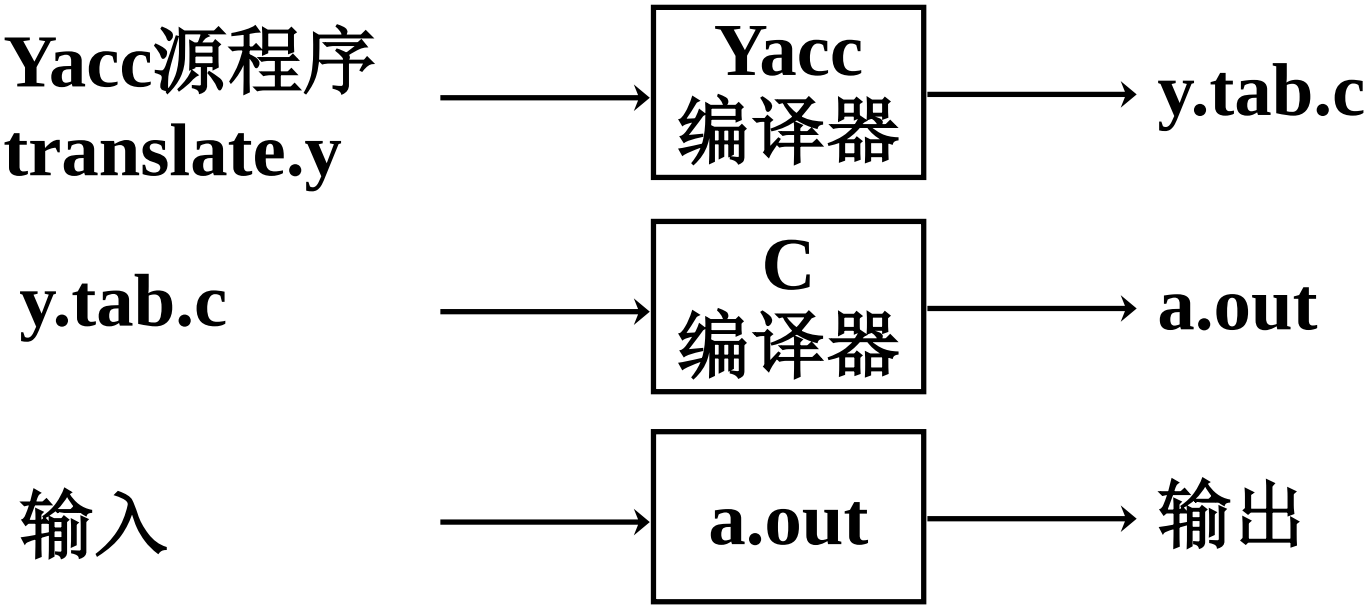


主要内容



- LR(k)分析技术
  - 规范的LR方法、向前看的LR方法
  - 二义性文法的使用、错误恢复
  - 分析器的生成器(Yacc)
- 语法分析技术总结

- Yacc (Yet another compiler-compiler)
  - 生成LALR语法分析器



- 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

分析器的生成器


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

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

分析器的生成器


```
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
            ;
%%
```

分析器的生成器


```
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

分析器的生成器


```
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

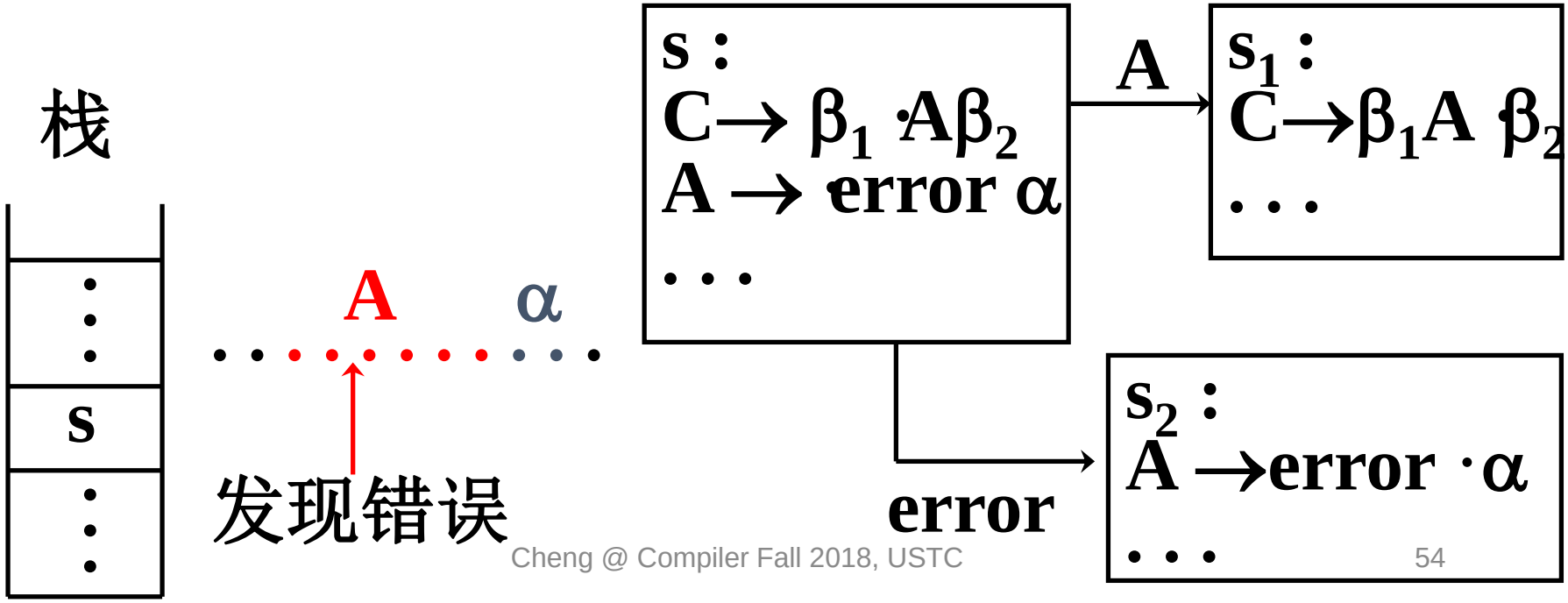
❑Yacc的错误恢复

❑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$\alpha$是文法符号串
- ❖为这样的产生式配上语义动作

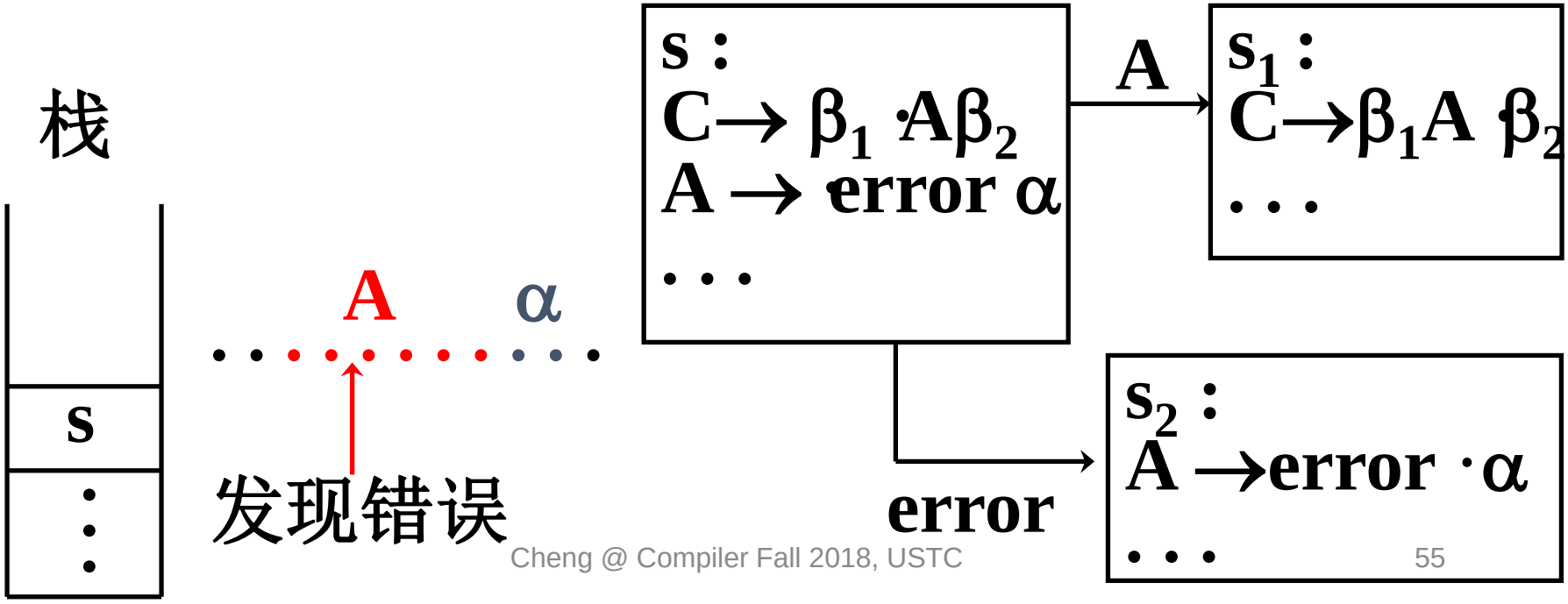
❑Yacc把错误产生式当作普通产生式处理

❑遇到语法错误时



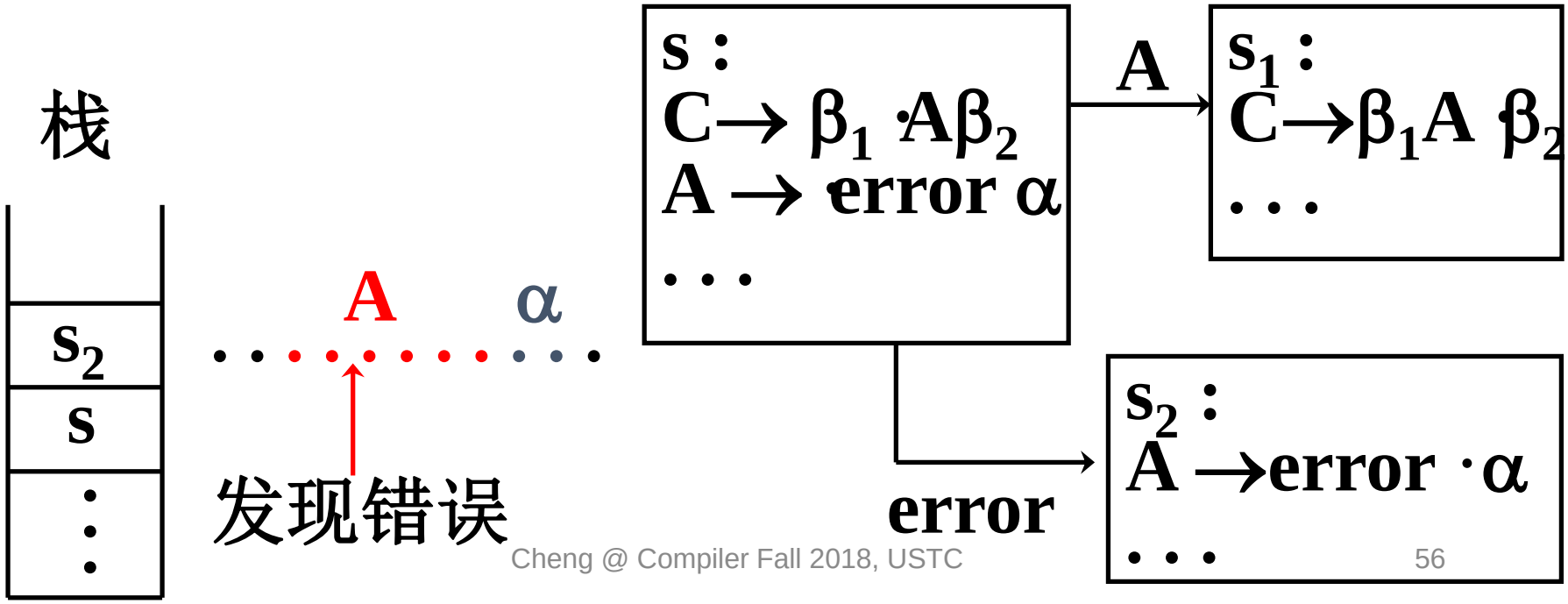
❑遇到语法错误时

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



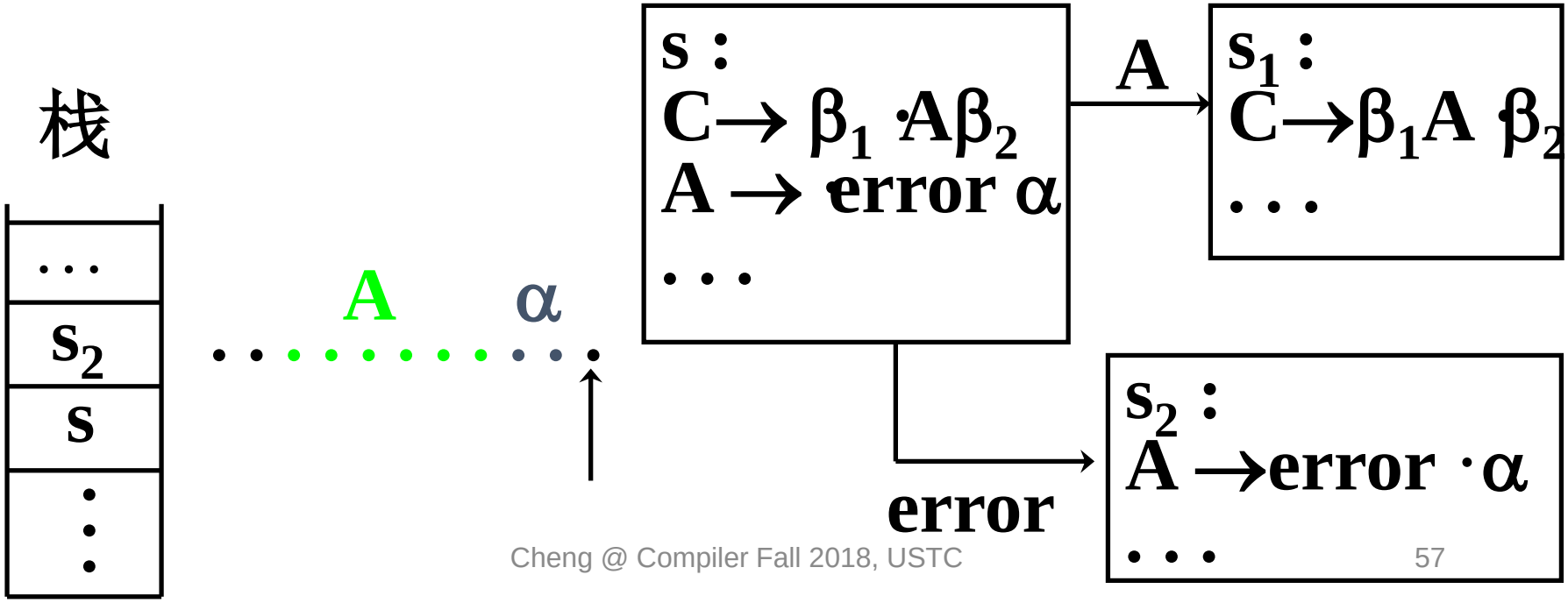
❑遇到语法错误时

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



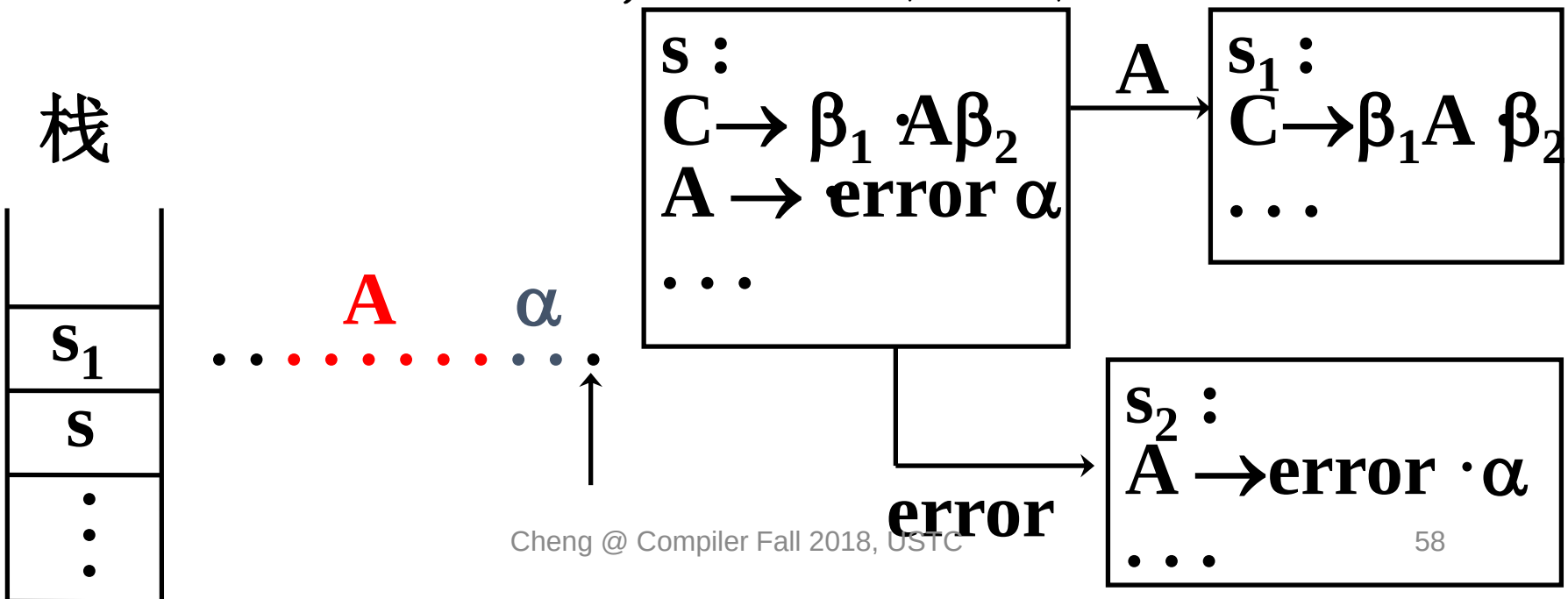
❑遇到语法错误时

- ❖从栈中弹出状态，直到发现栈顶状态的项目集包含形为$A \rightarrow$ error α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

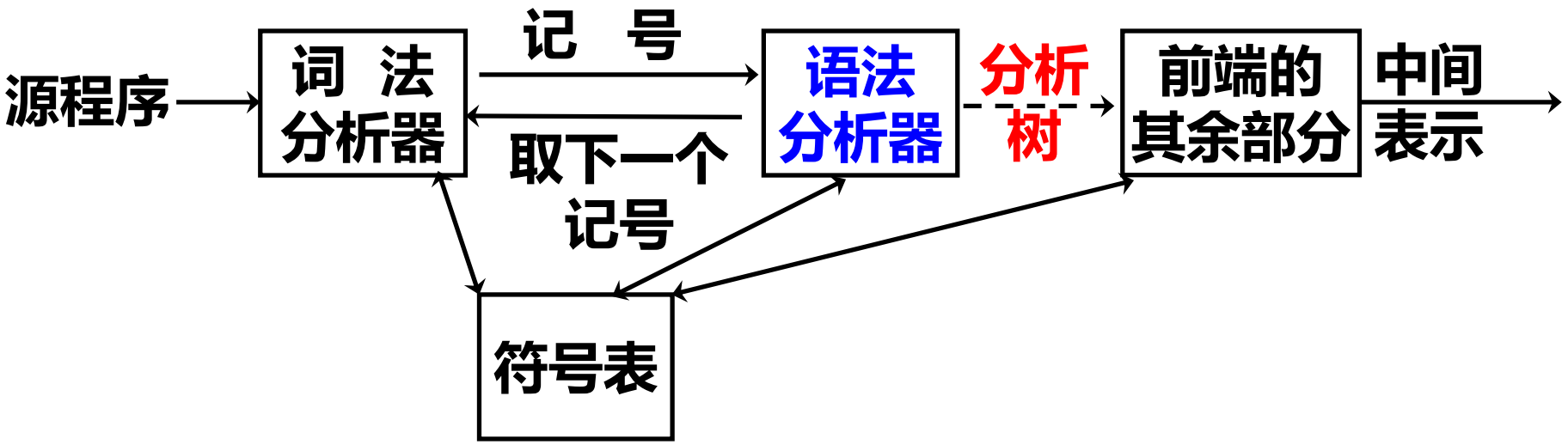
- ❖从栈中弹出状态，直到发现栈顶状态的项目集包含形为$A \rightarrow$ error α的项目为止
- ❖把虚构的终结符error“移进”栈
- ❖忽略若干输入符号，直至找到α，把α移进栈
- ❖把error α归约为A，恢复正常分析



❑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;}
        ;
```

主要内容

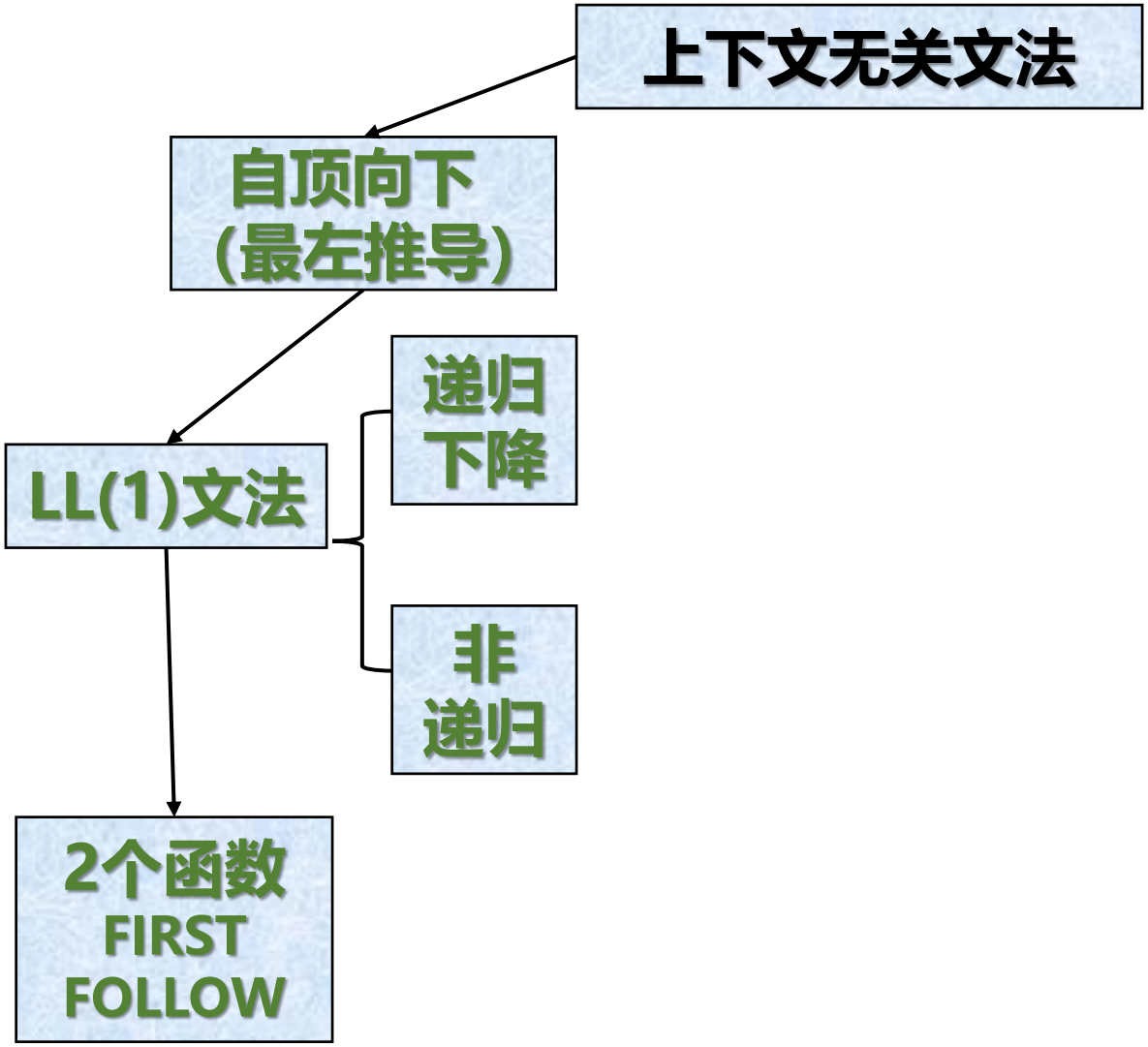


- LR(k)分析技术
  - 规范的LR方法、向前看的LR方法
  - 二义性文法的使用、错误恢复
  - 分析器的生成器(Yacc)
- **语法分析技术总结**

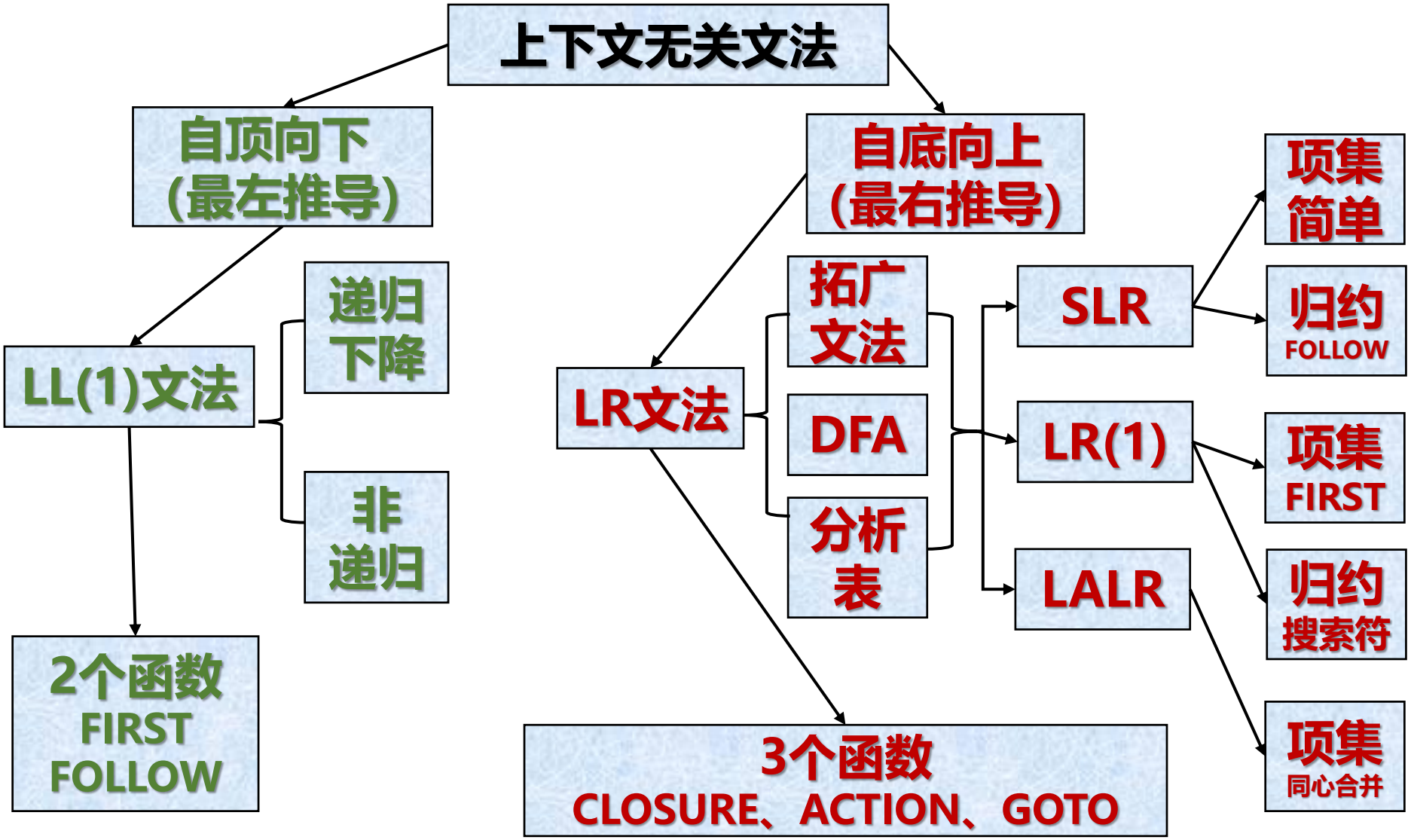
LR分析法总结

		SLR	LALR	LR(1)
初始状态		$[S' \rightarrow S]$	$[S' \rightarrow S, \$]$	$[S' \rightarrow S, \$]$
项目集		LR(0) CLOSURE(I)	合并LR(1)项目集族的同心项目集	LR(1), CLOSURE(I) 搜索符考虑FISRT(βa)
动作	移进	$[A \rightarrow \alpha a\beta] \in I_i$ $GOTO(I_i, a) = I_j$ $ACTION[i, a] = sj$	与LR(1) 一致	$[A \rightarrow \alpha a\beta, b] \in I_i$ $GOTO(I_i, a) = I_j$ $ACTION[i, a] = sj$
	归约	$[A \rightarrow \alpha] \in I_i, A \neq S'$ $A \in FOLLOW(A)$ $ACTION[i, a] = rj$	与LR(1) 一致	$[A \rightarrow \alpha; a] \in I_i$ $A \neq S'$ $ACTION[i, a] = rj$
	接受	$[S' \rightarrow S \cdot] \in I_i$ $ACTION[i, \$] = acc$	与LR(1) 一致	$[S' \rightarrow S \cdot, \$] \in I_i$ $ACTION[i, \$] = acc$
	出错	空白条目	与LR(1) 一致	空白条目
GOTO		$GOTO(I_i, A) = I_j$ $GOTO[i, A] = j$	与LR(1) 一致	$GOTO(I_i, A) = I_j$ $GOTO[i, A] = j$
状态量		少(几百)	与SLR一样	多(几千)

语法分析技术总结



语法分析技术总结


上下文无关文法
自顶向下
(最左推导)
自底向上
(最右推导)
LL(1)文法
递归
下降
非
递归
2个函数
FIRST
FOLLOW
LR文法
拓广
文法
DFA
分析
表
SLR
LR(1)
LALR
项集
简单
归约
FOLLOW
项集
FIRST
归约
搜索符
项集
同心合并
3个函数
CLOSURE、ACTION、GOTO

LR和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
建立分析树	自 下 而 上	自 上 而 下
归约or推导	规 范 归 约	最 左 推 导
决定使用产生式的时机	看见产生式整个右部推出的串后	看见产生式推出的第一个终结符后
对文法的限制	无	无左递归、无公共左因子
分析表	状态×文法符号，大	非终结符×终结符，小
分析栈	状态栈，信息更多	文法符号栈
确定句柄	根据栈顶状态和下一个符号便可以确定句柄和归约所用产生式	无句柄概念
语法错误	决不会将出错点后的符号移入分析栈	和LR一样，决不会读过出错点而不报错

《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*