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》

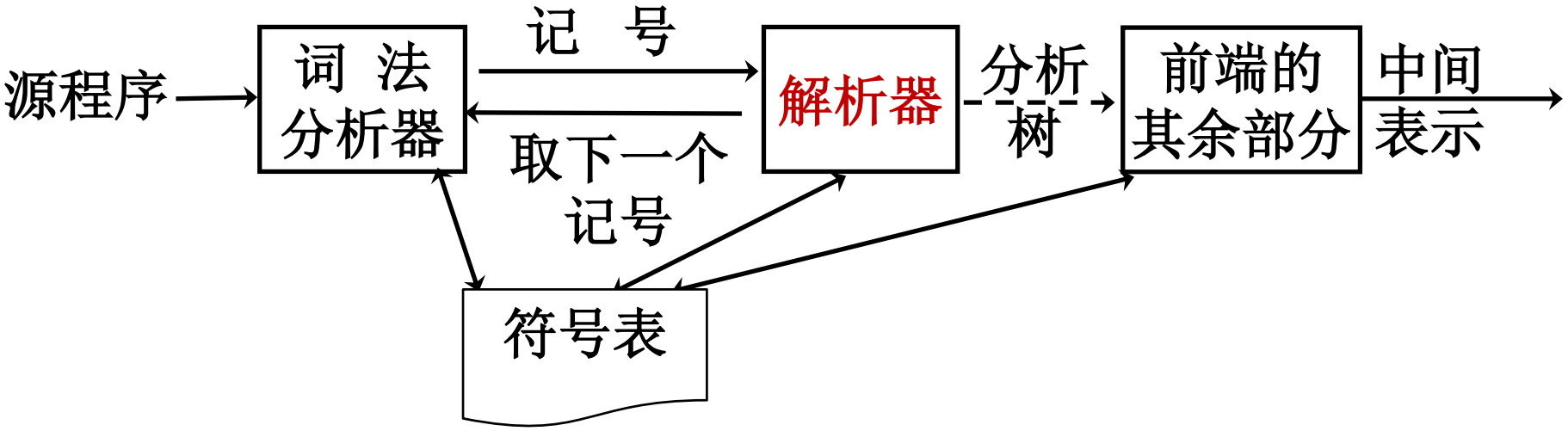
张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

本章内容



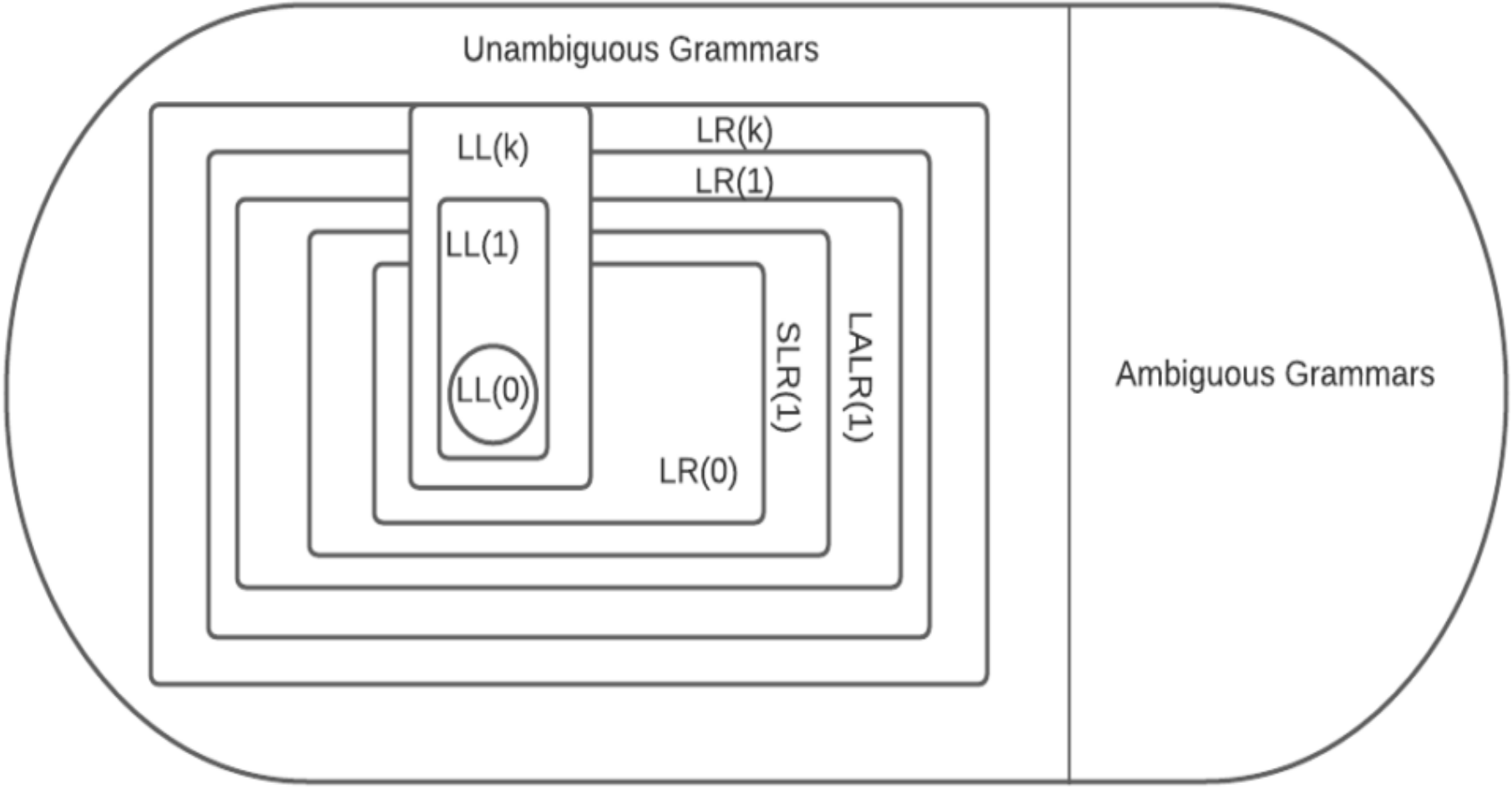
- 语法的形式描述：上下文无关文法
- 语法分析：自上而下、自下而上
- 语法分析器(parser、syntax analyzer)的自动生成
  - LL(k)、LL(*)、SLR(k)、LR(k)、LALR(k)

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

3.6 二义文法的应用

- 通过其他手段消除二义文法的二义性
- LR分析的错误恢复

文法及其表达能力


Unambiguous Grammars
LL(k)
LR(k)
LR(1)
LL(1)
LL(0)
SLR(1)
LALR(1)
LR(0)
Ambiguous Grammars

二义文法的特点

- 特点
  - 绝不是LR 文法
  - 简洁、自然

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

非二义的文法：

$E \to E + T \mid T$

$T \to T * F \mid F$

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

该文法有单个非终结符为右部的产生式（简称单非产生式）

单非产生式会增加
分析树的高度
=> 分析效率降低

二义文法的消除

- 特点
  - 绝不是LR 文法
  - 简洁、自然
  - 可以用文法以外的信息来消除二义

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

二义文法的消除

- 特点
  - 绝不是LR 文法
  - 简洁、自然
  - 可以用文法以外的信息来消除二义

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

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

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

$E \to E \cdot * E$

面临+，归约　面临*，移进

面临)和$，归约

二义文法的消除

- 特点
  - 绝不是LR 文法
  - 简洁、自然
  - 可以用文法以外的信息来消除二义

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_8

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

$E \to E \cdot + E$

$E \to E \cdot * E$

面临+，归约

二义文法的消除

- 特点
  - 绝不是LR 文法
  - 简洁、自然
  - 可以用文法以外的信息来消除二义

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_8

$E \to E * E \cdot$

$E \to E \cdot + E$

$E \to E \cdot * E$

id * id + id

id * id * id

面临+，归约

面临*，归约

面临)和$，归约

特殊情况引起的二义性

$E \rightarrow E$ sub E sup E

$E \rightarrow E$ sub E

$E \rightarrow E$ sup E

$E \rightarrow \{E\}$

$E \rightarrow c$

从定义形式语言的角度说，第一个产生式是多余的

但联系到语义处理，第一个产生式是必要的

对a sub i sup 2，需要下面第一种输出

$$a_i^2 \qquad a_i{}^2 \qquad a_{i^2}$$

特殊情况引起的二义性

$E \rightarrow E$ sub E sup E

$E \rightarrow E$ sub E

$E \rightarrow E$ sup E

$E \rightarrow \{E\}$

$E \rightarrow c$

I_{11}:

$E \rightarrow E$ sub E sup E ·

$E \rightarrow E$ sup E ·

…

按前面一个产生式归约

LR分析的错误恢复

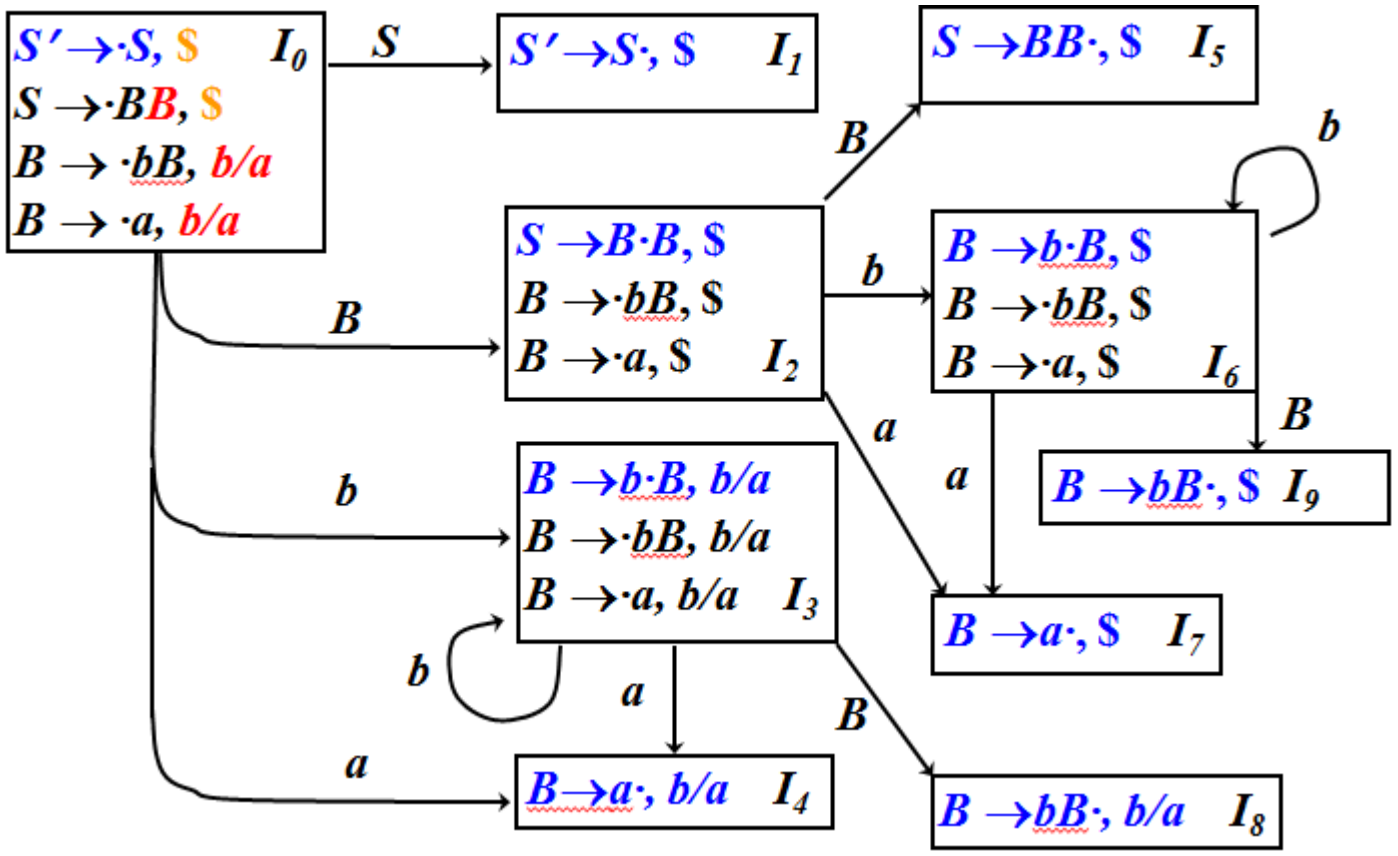
- LR分析器在什么情况下发现错误
  - 访问action表时遇到出错条目
  - 访问goto表时绝不会遇到出错条目
  - 绝不会把不正确的后继移进栈
  - 规范的LR分析器在报告错误之前决不做任何无效归约
    - SLR和LALR在报告错误前有可能执行几步无效归约

规范的LR分析不会把错误移进

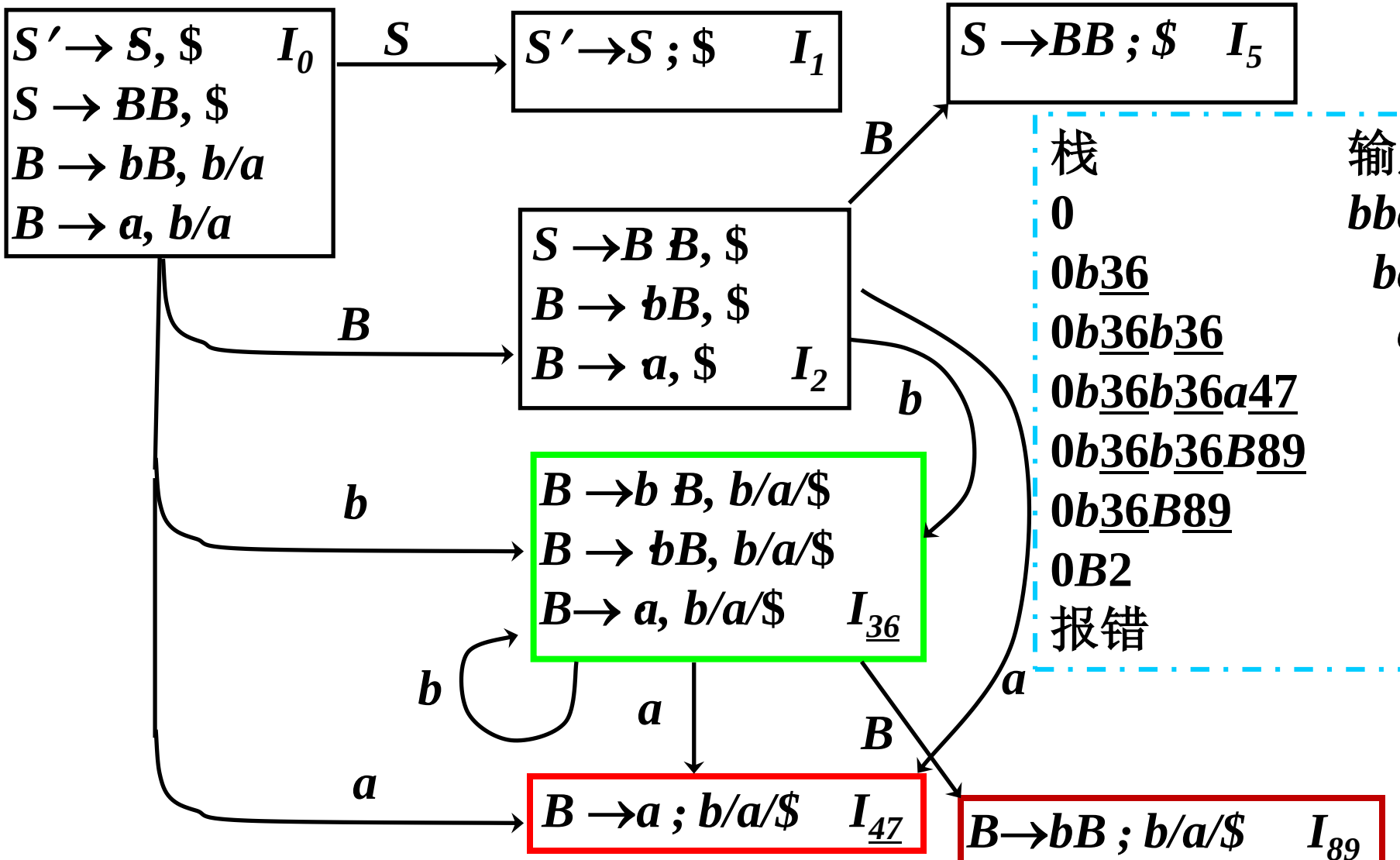
给出在以下两种输入下的LR分析过程

bbabba$

bba$



LALR分析也不会把错误移进



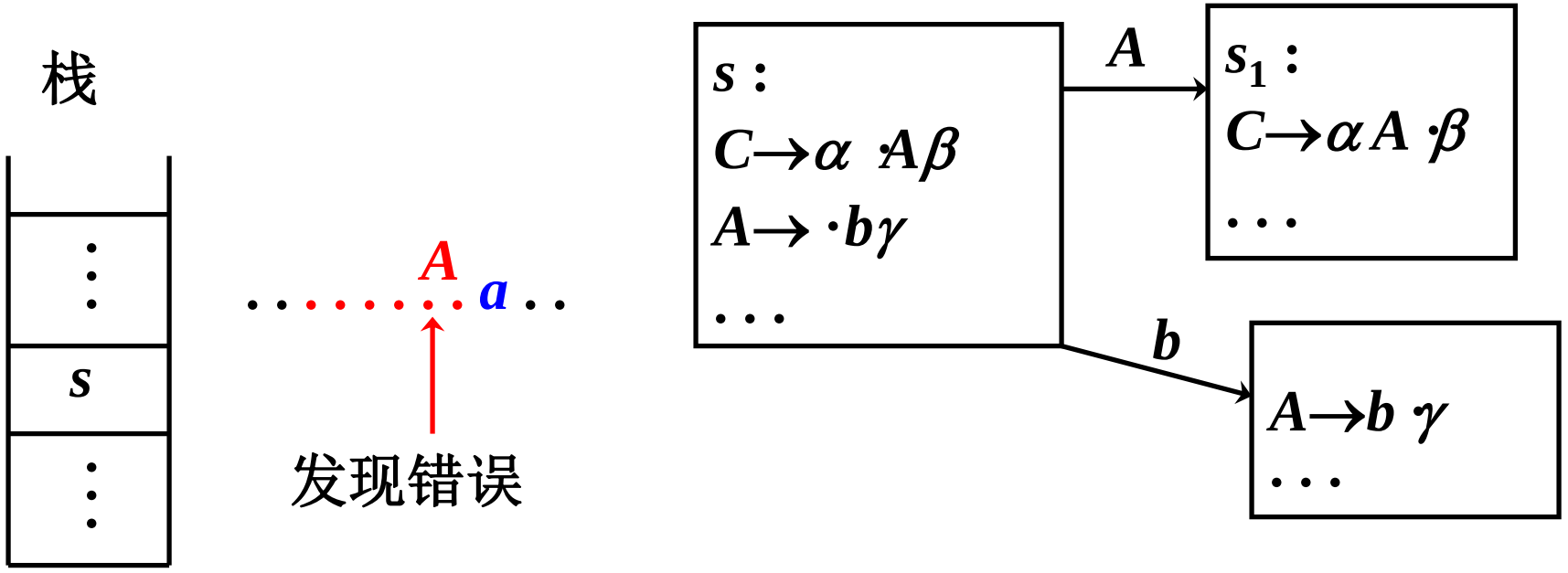
栈	输入
0	bba$
0b36	ba$
0b36b36	a$
0b36b36a47	$
0b36b36B89	$
0b36B89	$
0B2	$
报错	

紧急方式的错误恢复

错误恢复策略

- 试图忽略含语法错误的短语：A推出的串含错误

1. 退栈，直至出现状态 s，它有预先确定的A的转移

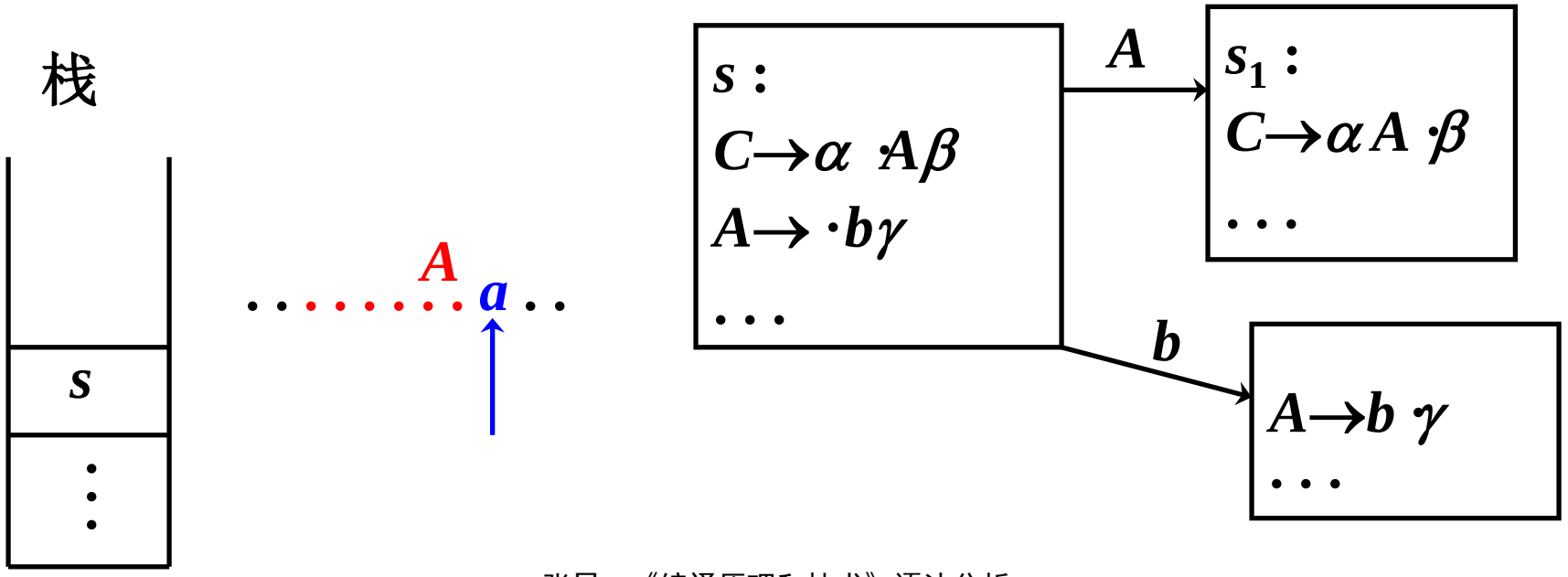


紧急方式的错误恢复

错误恢复策略

- 试图忽略含语法错误的短语：A推出的串含错误

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

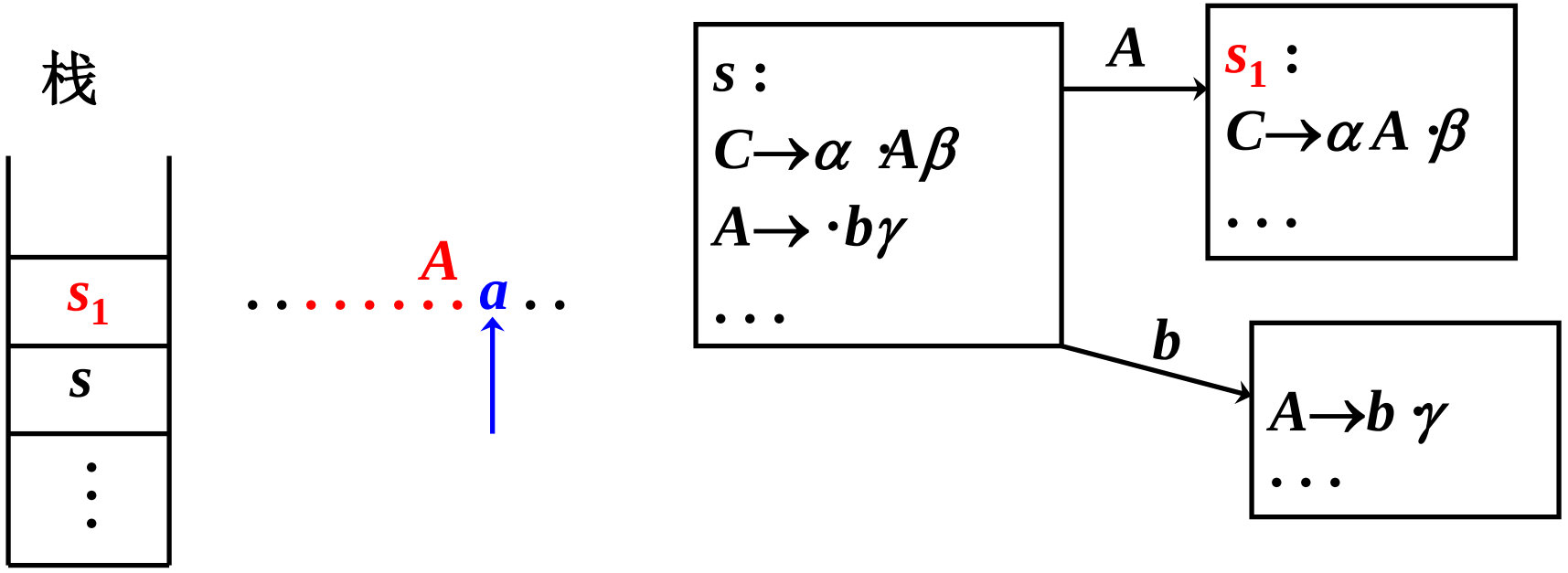


紧急方式的错误恢复

错误恢复策略

- 试图忽略含语法错误的短语：A推出的串含错误

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



短语级恢复

- 发现错误时，对剩余输入作局部纠正

  如用分号代替逗号，删除多余的分号，插入遗漏的分号

缺点：难以解决实际错误出现在诊断点以前的情况

- 实现方法

  在action表的每个空白条目填上指示器，指向错误处理例程

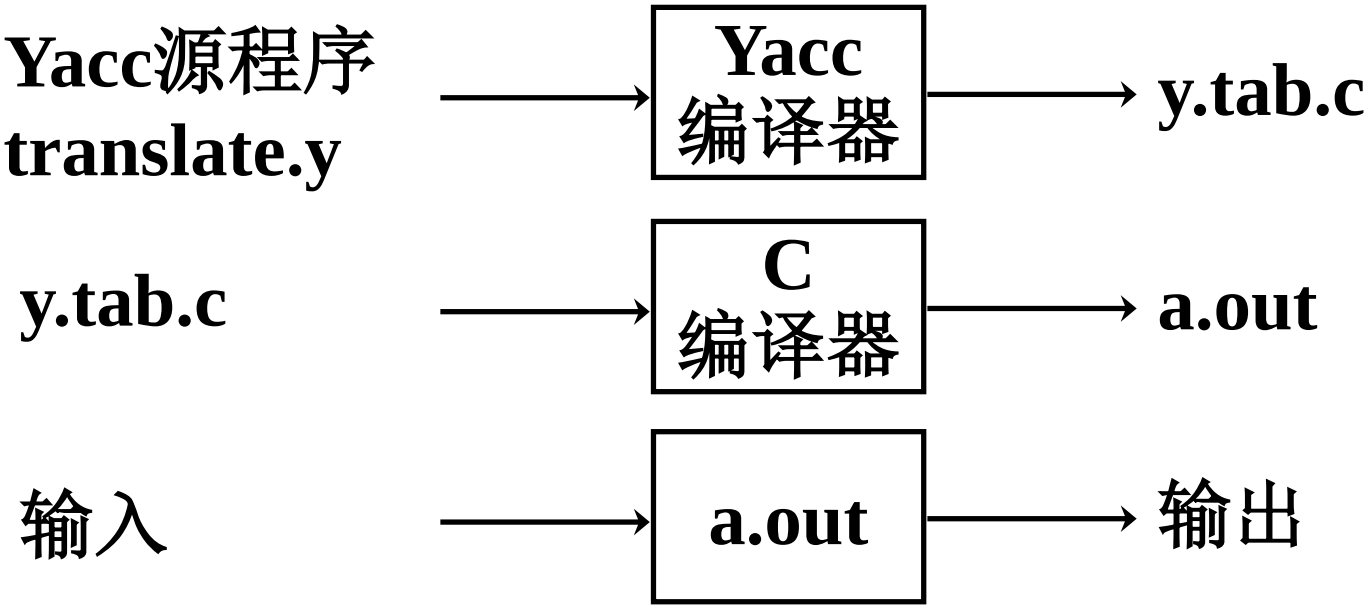
中国科学技术大学
University of Science and Technology of China

3.7 分析器的生成器

- YACC(LR分析器的生成器)
- ANTLR(LL分析器的生成器)

YACC

- YACC（Yet Another Compiler Compiler）



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

声明部分

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

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

例　简单计算器

翻译规则部分

```
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
            ;
%%
```

例　简单计算器

翻译规则部分

```
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

例　简单计算器

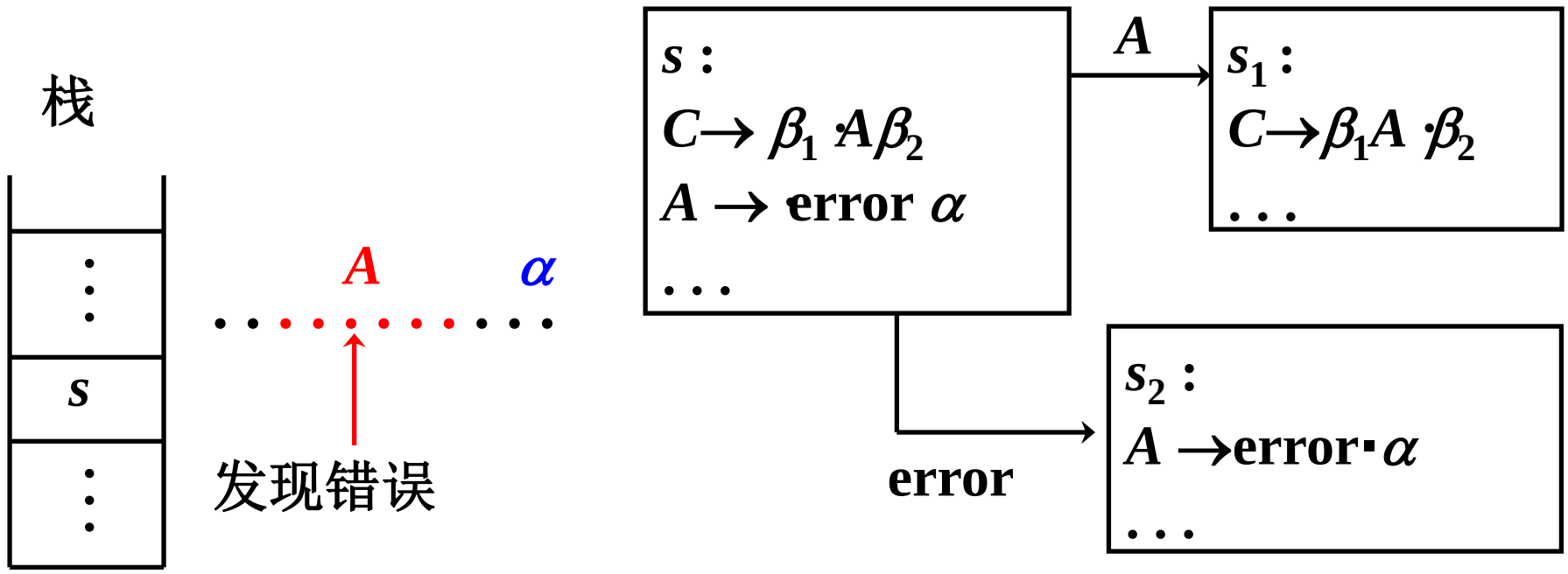
C例程部分

```
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

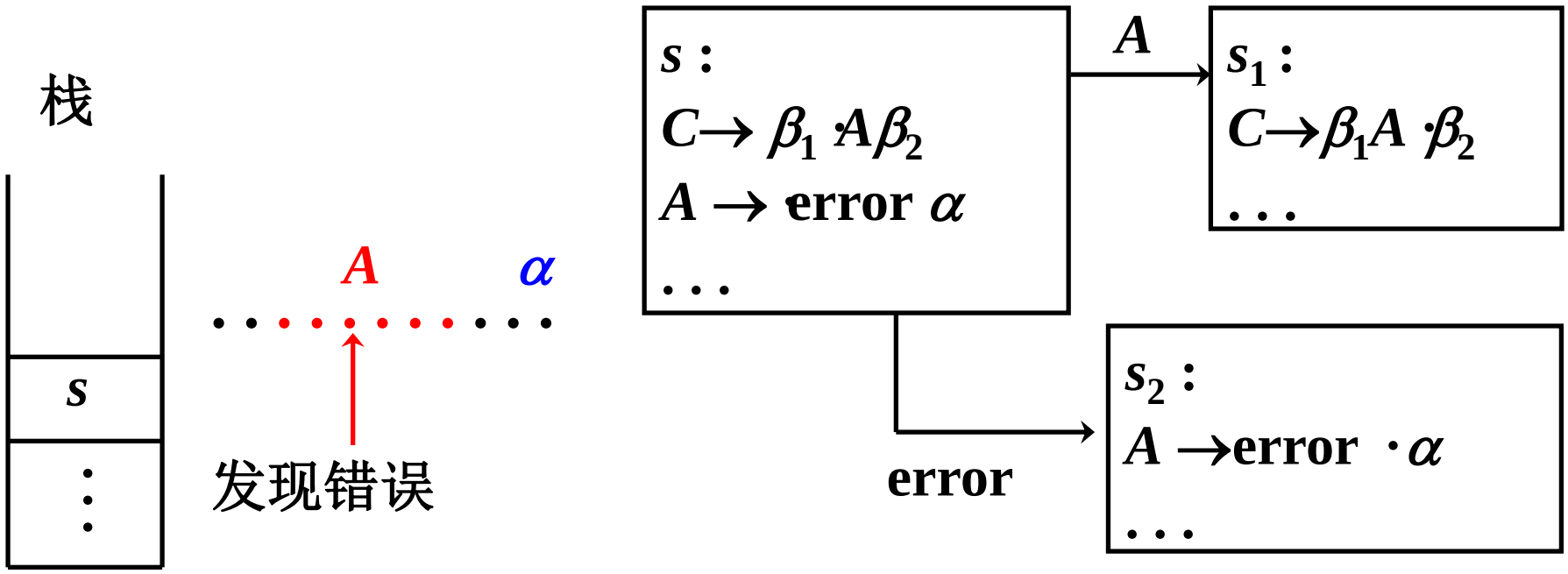
- 增加错误产生式 $A \rightarrow \text{error}\ \alpha$
- 遇到语法错误时



YACC的错误恢复

遇到语法错误时

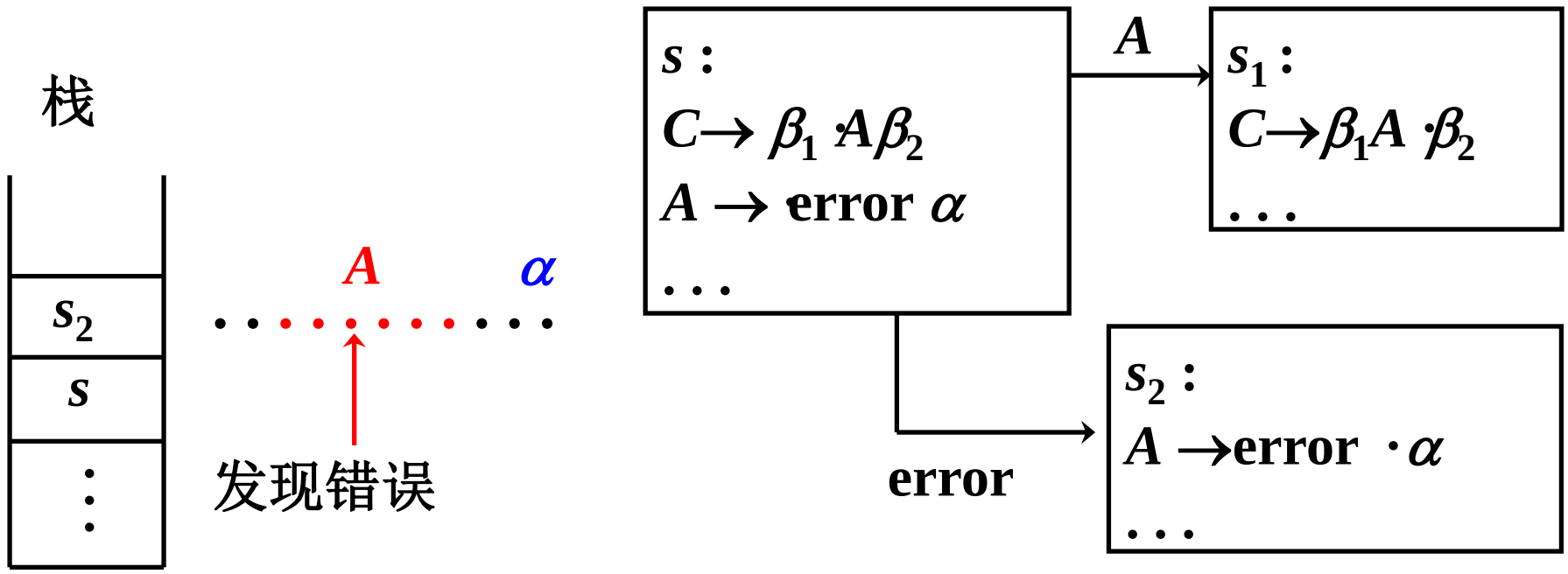
- 从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \text{error}\ \alpha$ 的项目为止



YACC的错误恢复

遇到语法错误时

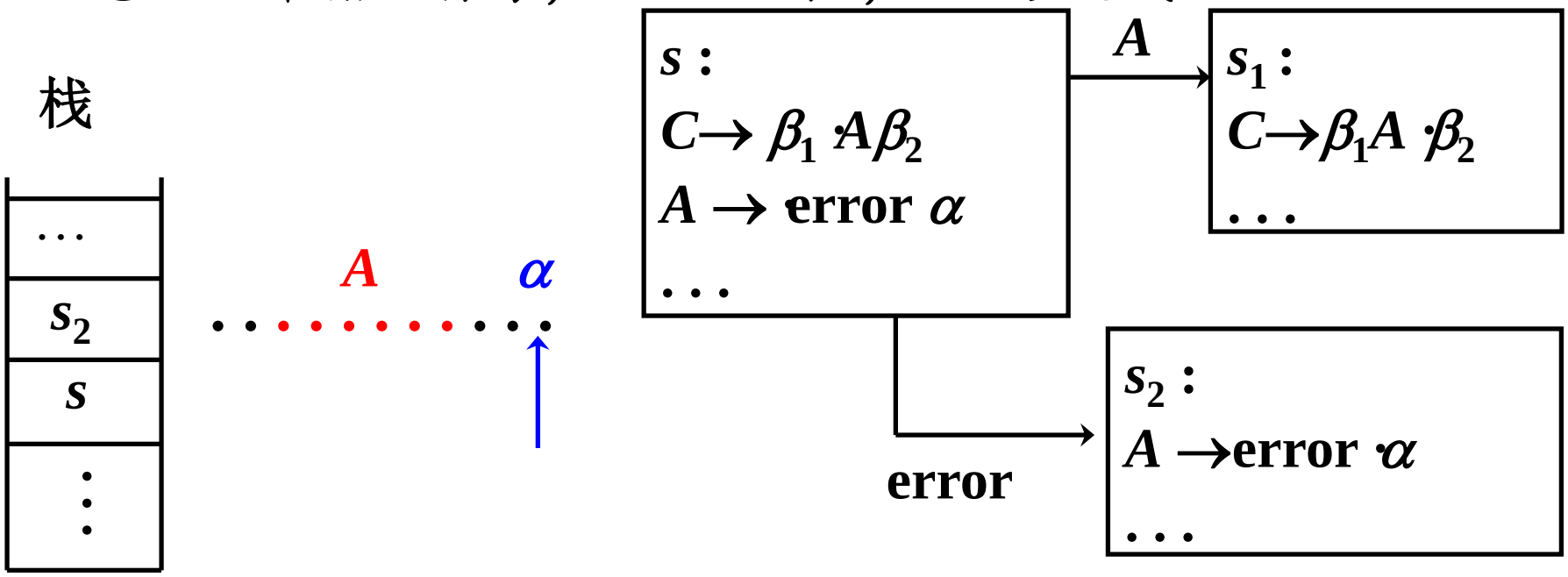
- 从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \text{error}\ \alpha$的项目为止
- 把虚构的终结符error“移进”栈



YACC的错误恢复

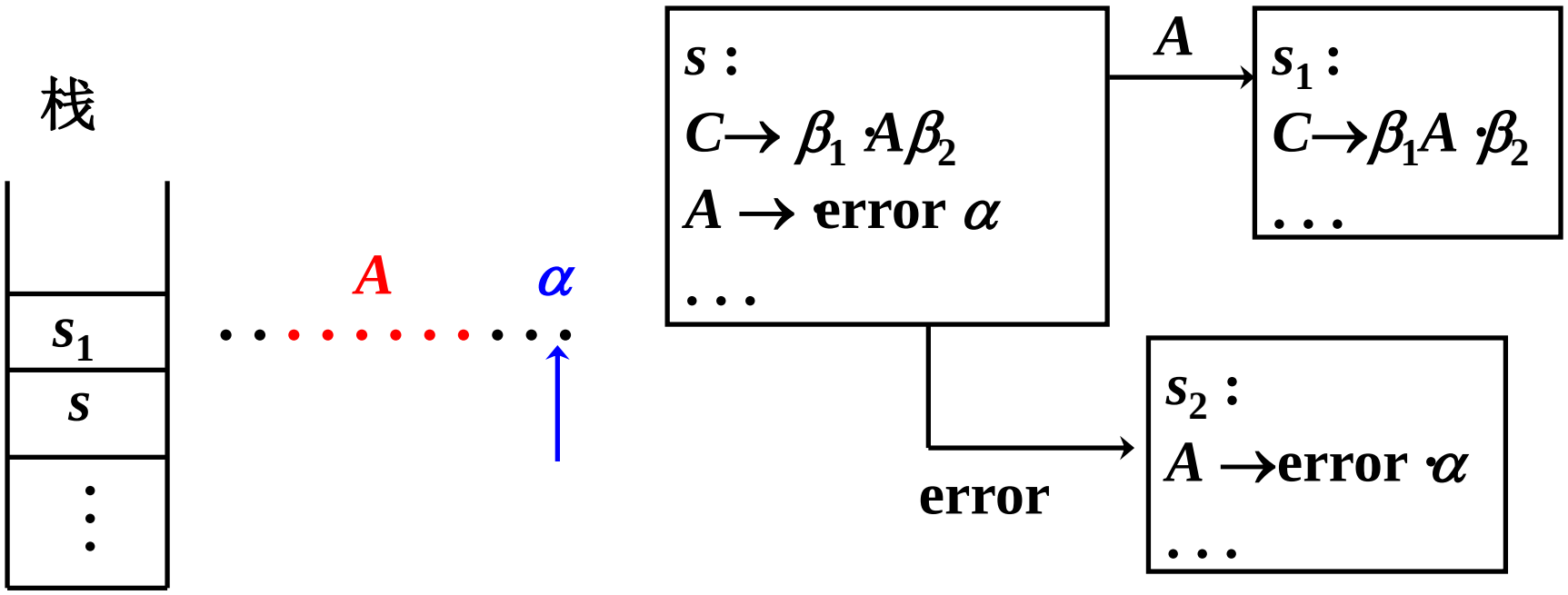
遇到语法错误时

- 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
- 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



YACC的错误恢复

- 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
- 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



例　简单计算器

- 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;}
            ;
```