



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

语法分析 VI

《编译原理和技术(H)》、《编译原理(H)》

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

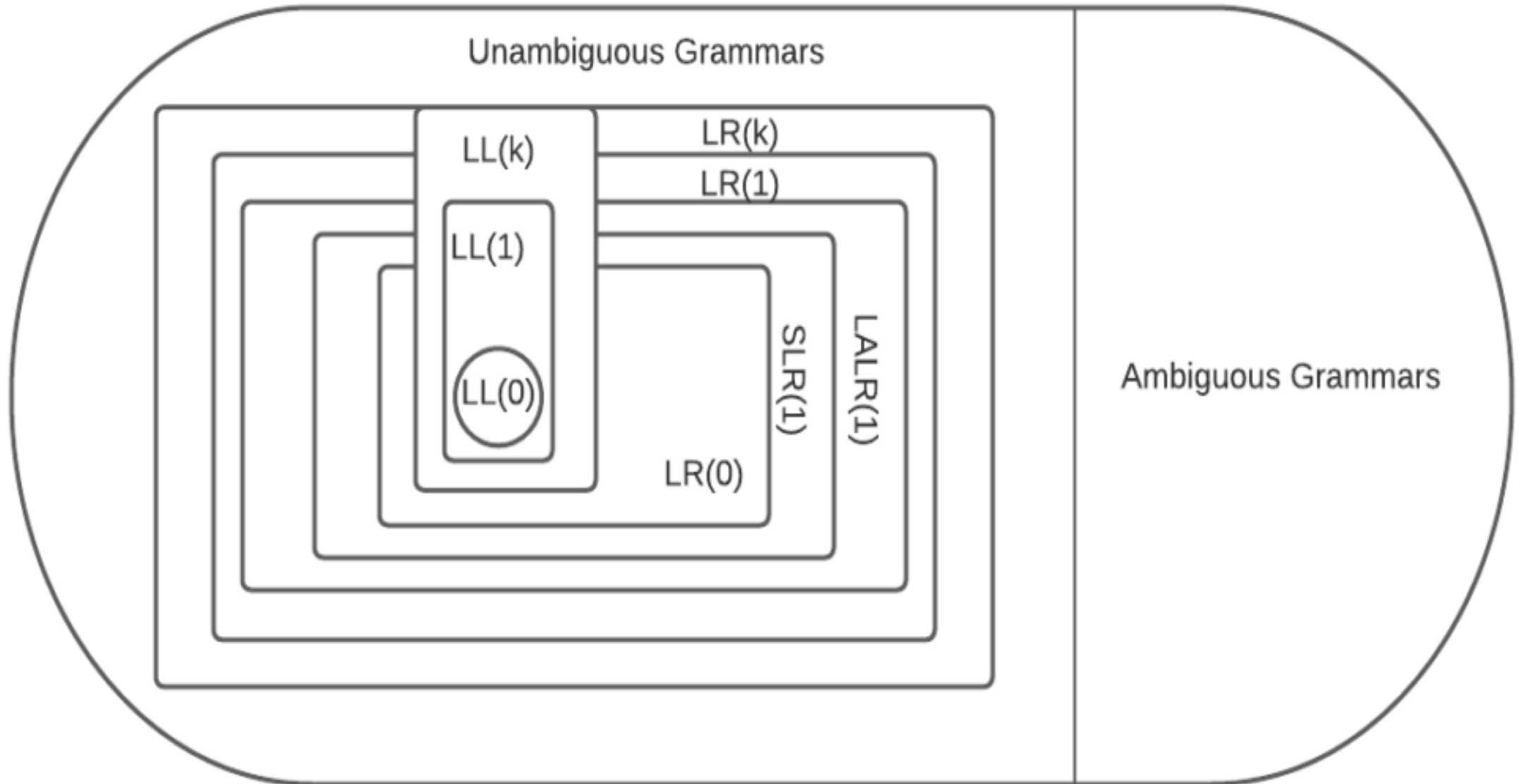


3.7 二义文法的应用

- 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





二义文法的特点

□ 特点

- 绝不是LR 文法
- 简洁、自然

例 二义文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$

非二义的文法:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \text{id}$$

单非产生式会增加
分析树的高度
 \Rightarrow 分析效率降低

该文法有单个非终结符为右部的产生式（简称单非产生式）



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 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$

规定：*优先级高于+，两者都是左结合

LR(0)项目集 I_7

$E \rightarrow E + E \cdot$

$E \rightarrow E + E$

$E \rightarrow E * E$

栈

id + id

输入

+ 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 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$

规定: *优先级高于+, 两者都是左结合

LR(0)项目集 I_7

$E \rightarrow E + E \cdot$

$E \rightarrow E + E$

$E \rightarrow E * E$

栈

id + id

id + id

输入

+ id

* id

面临+, 归约

面临*, 移进

Follow(E)={+, *,), \$}

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 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$

规定：*优先级高于+，两者都是左结合

LR(0)项目集 I_8

$E \rightarrow E * E \cdot$

$E \rightarrow E + E$

$E \rightarrow E * E$

栈

id * id

输入

+ 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 $E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$

规定：*优先级高于+，两者都是左结合

LR(0)项目集 I_8

$E \rightarrow E * E \cdot$

$E \rightarrow E + E$

$E \rightarrow E * E$

栈

id * id

id * id

输入

+ id

* id

面临+, 归约

面临*, 归约

Follow(E)={+, *,), \$}

面临)和\$, 归约



特殊情况引起的二义性

$$E \rightarrow E \text{ sub } E \text{ sup } E$$

$$E \rightarrow E \text{ sub } E$$

$$E \rightarrow E \text{ sup } E$$

$$E \rightarrow \{E\}$$

$$E \rightarrow c$$

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 $a \text{ sub } i \text{ sup } 2$ ，需要下面第一种输出

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



特殊情况引起的二义性

$E \rightarrow E \text{ sub } E \text{ sup } E$

$E \rightarrow E \text{ sub } E$

$E \rightarrow E \text{ sup } E$

$E \rightarrow \{E\}$

$E \rightarrow c$

I_{11} :

$E \rightarrow E \text{ sub } E \text{ sup } E \cdot$

$E \rightarrow E \text{ sup } E \cdot$

...

按前面一个产生式归约



3.7 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



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绝不做任何无效归约
 - 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

给出在以下输入下的

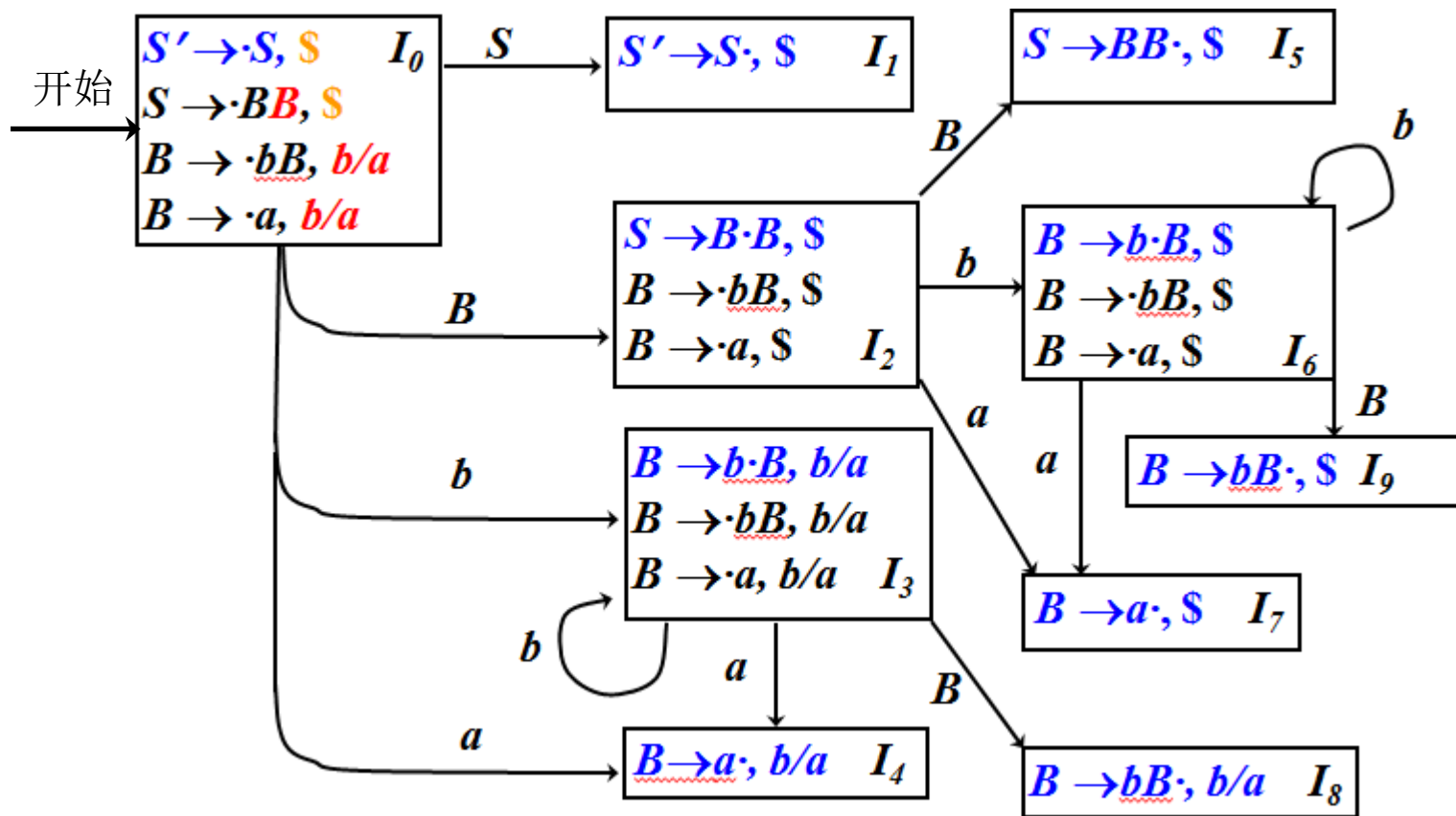
LR分析过程

$aa\$$

栈	输入
0	$aa\$$
0a4	$a\$$
0B2	$a\$$
0B2a7	$\$$
0B2B5	$\$$
0S1	$\$$
acc	

$a\$$

栈	输入
0	$a\$$
0a4	$\$$
报错	



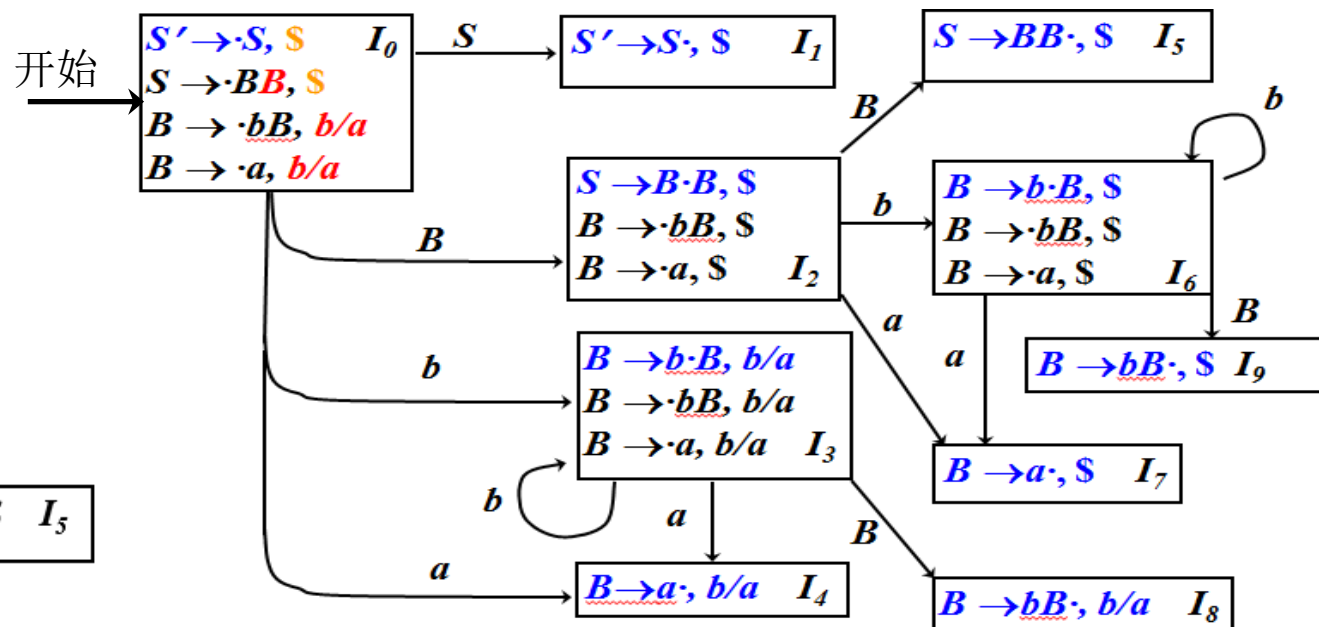
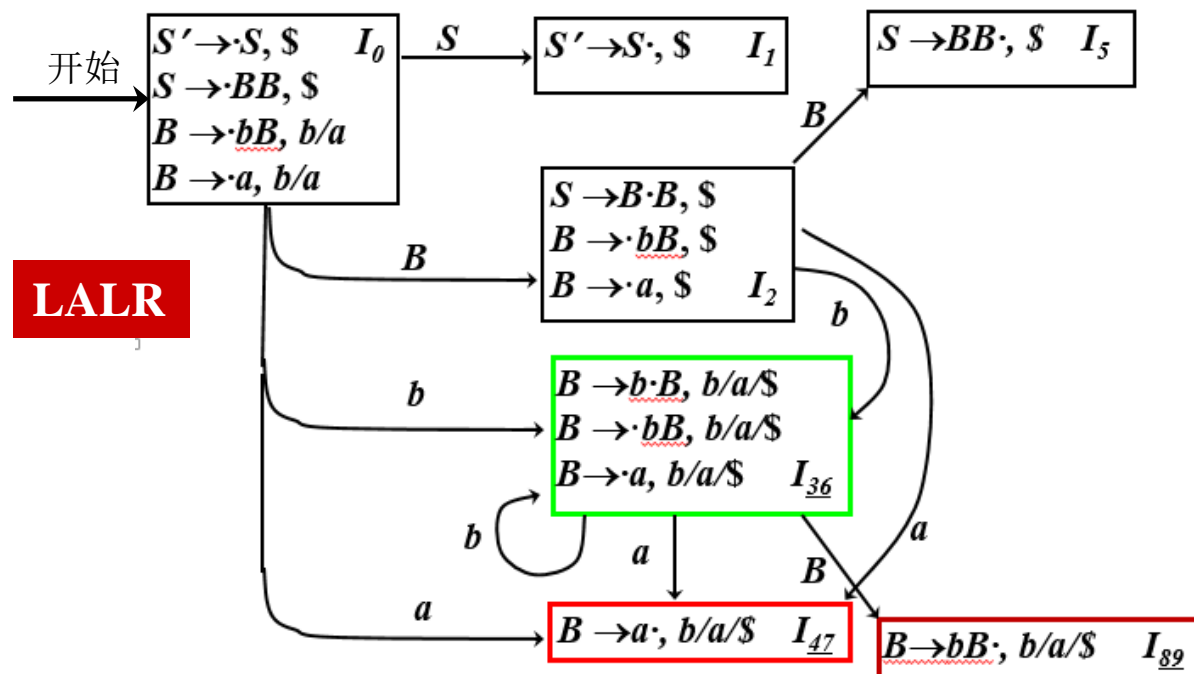


LALR分析不会把错误移进

给出在以下输入下的

LR分析过程

$a\$$



多执行无效归约

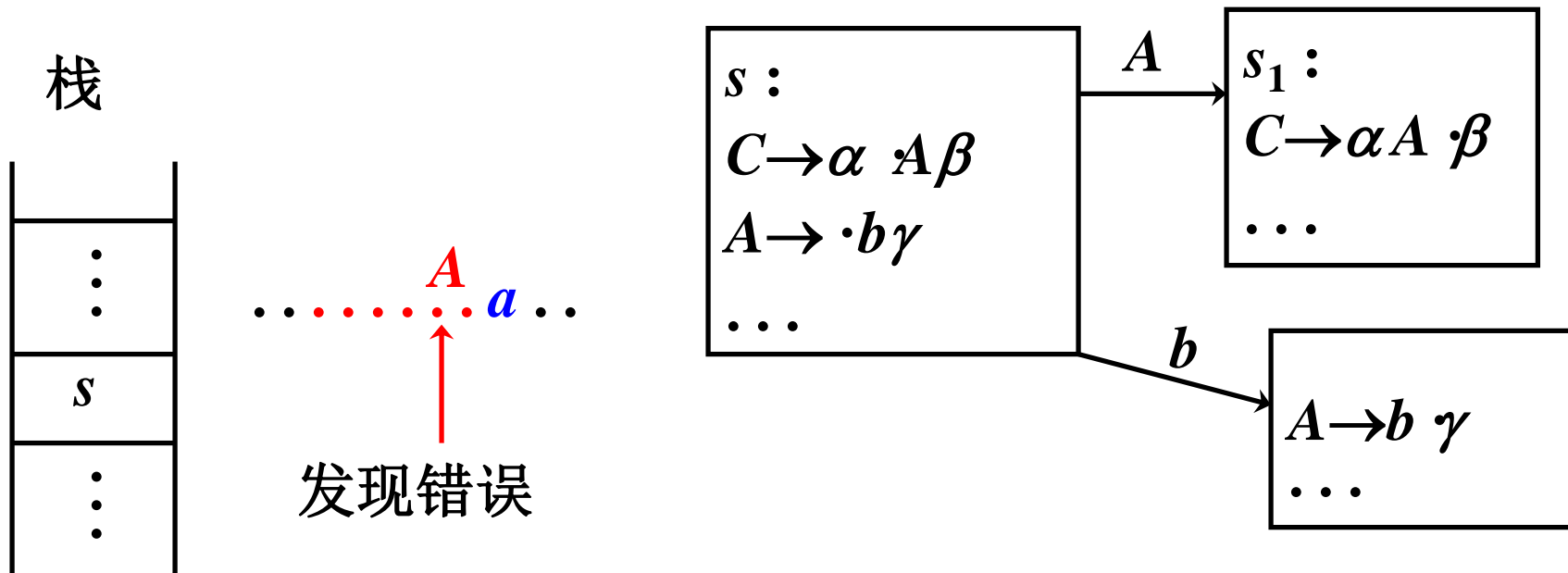


紧急方式的错误恢复

□ 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



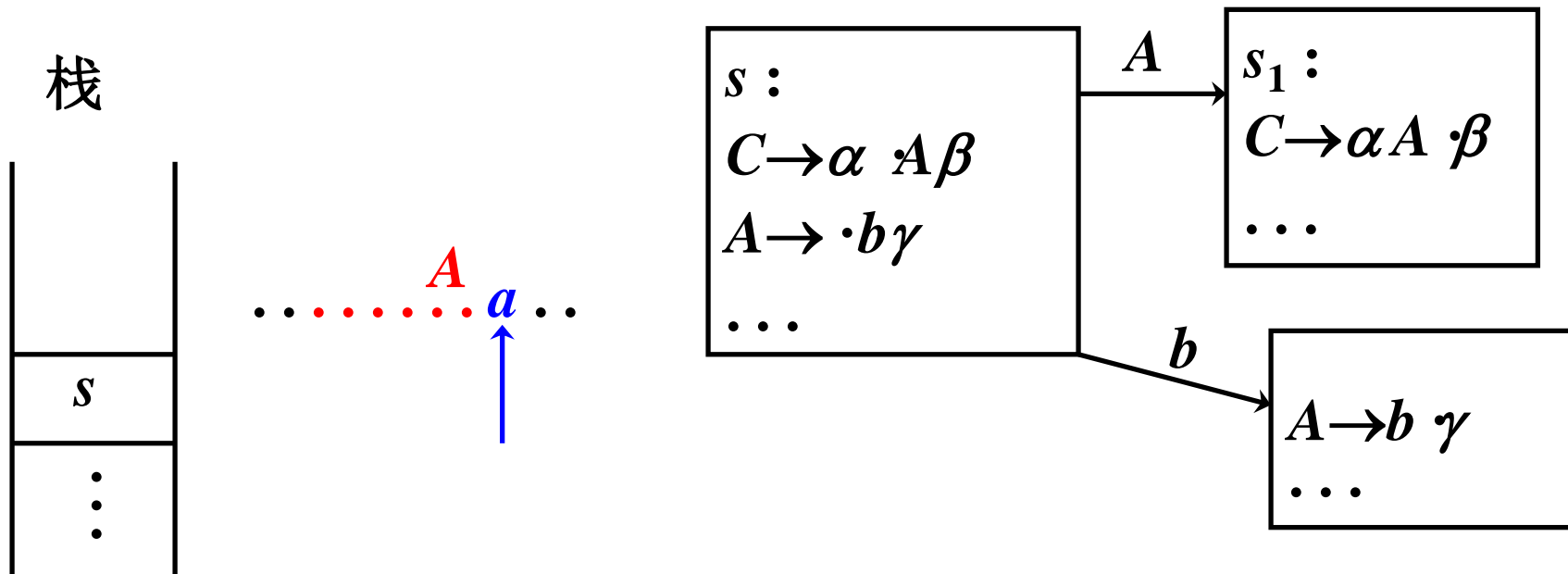


紧急方式的错误恢复

□ 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



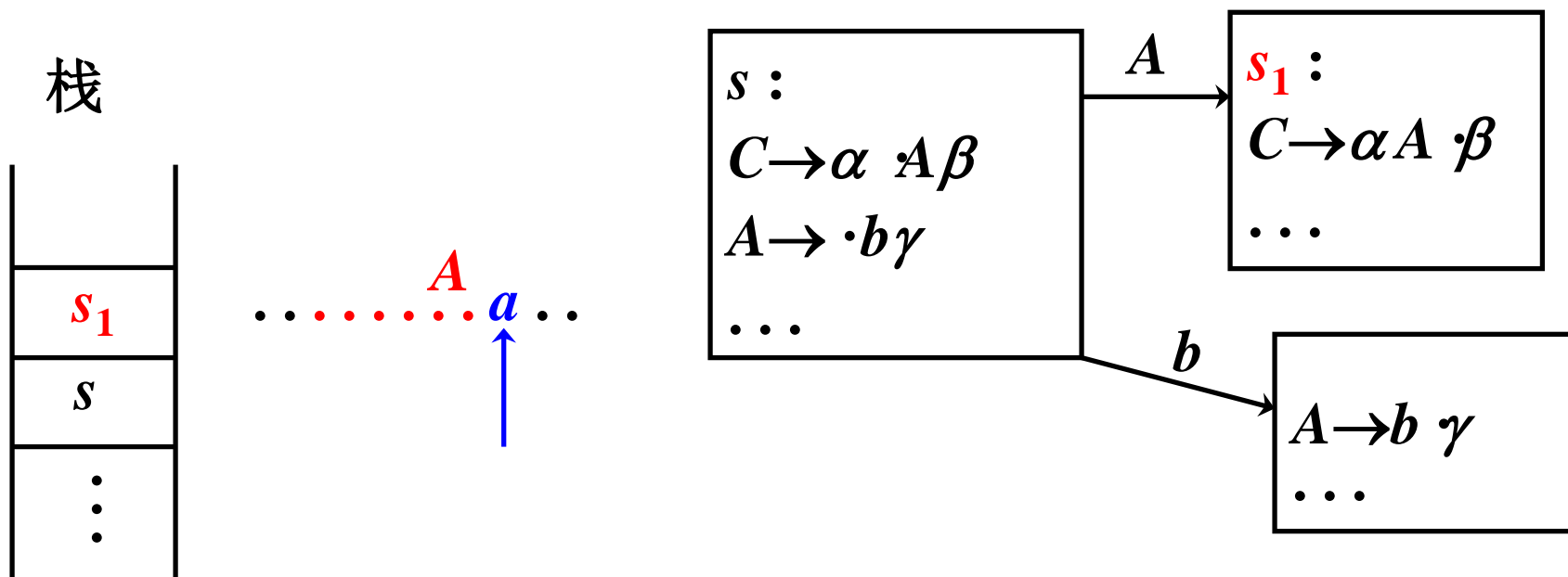


紧急方式的错误恢复

□ 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

□ 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

3.8 LR解析器的生成器：YACC



分析器的生成器

□ 生成器

- 生成Lexer: [Flex](#) ([for windows](#))、[Jflex](#)
- 生成Parser
 - LALR: [Bison](#) ([for windows](#))、[Java CUP](#)
 - LL: [JavaCC](#)、[ANTLR](#) – LL(*)^[PLDI2011], AdaptiveLL(*)^[OOPSLA2014]

□ 文法对Parser的影响

- LR Parser的优势: 速度快、表达能力强
- LL Parser的优势:
代码结构与文法对应, 易理解, 容易增加错误处理和错误恢复



确定的分析技术

确定的分析(deterministic parsing)

□ LL(k)、LR(k)、LALR(k)

□ LR分析对左递归的处理能力

■ 直接左递归: ✓

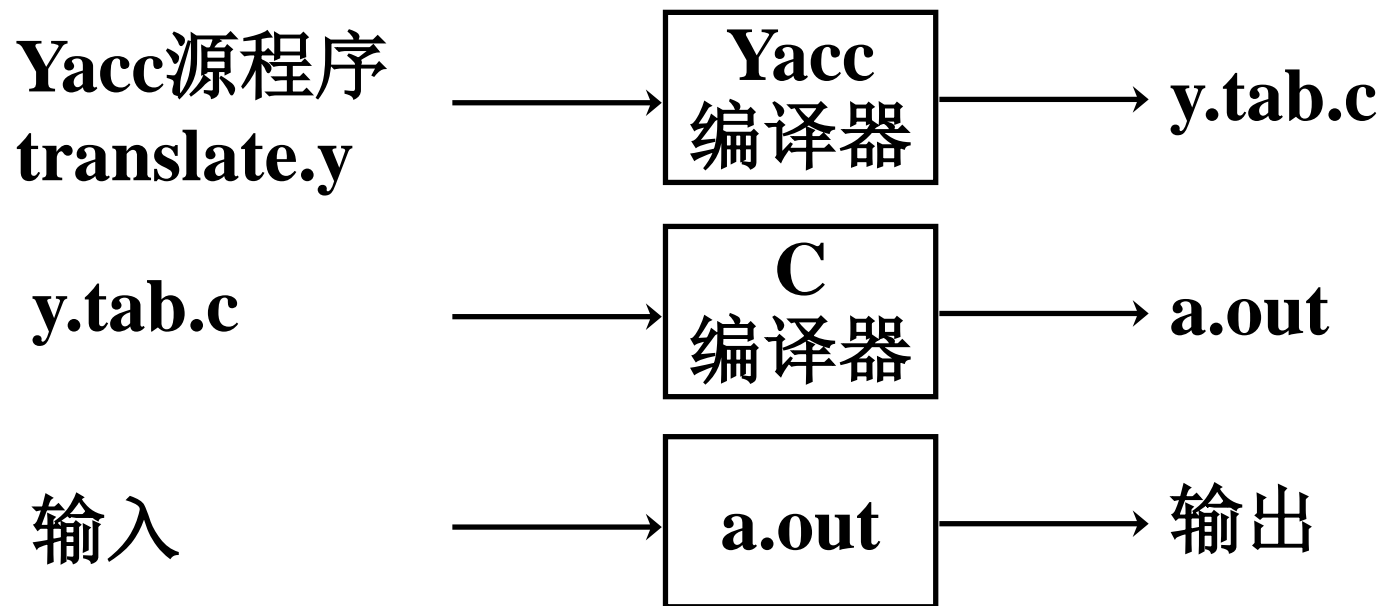
■ 隐藏的左递归 (hidden left recursion) : 可能不终止

□ $A \Rightarrow^* \beta A \mu$ 其中 $\beta \Rightarrow^+ \epsilon$

例如, $S \rightarrow C a / d$ $B \rightarrow \epsilon / a$ $C \rightarrow b / B C b / b b$



□ 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'  
           | /*  $\epsilon$  */  
           ;  
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'  
           | /*  $\epsilon$  */  
           ;  
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

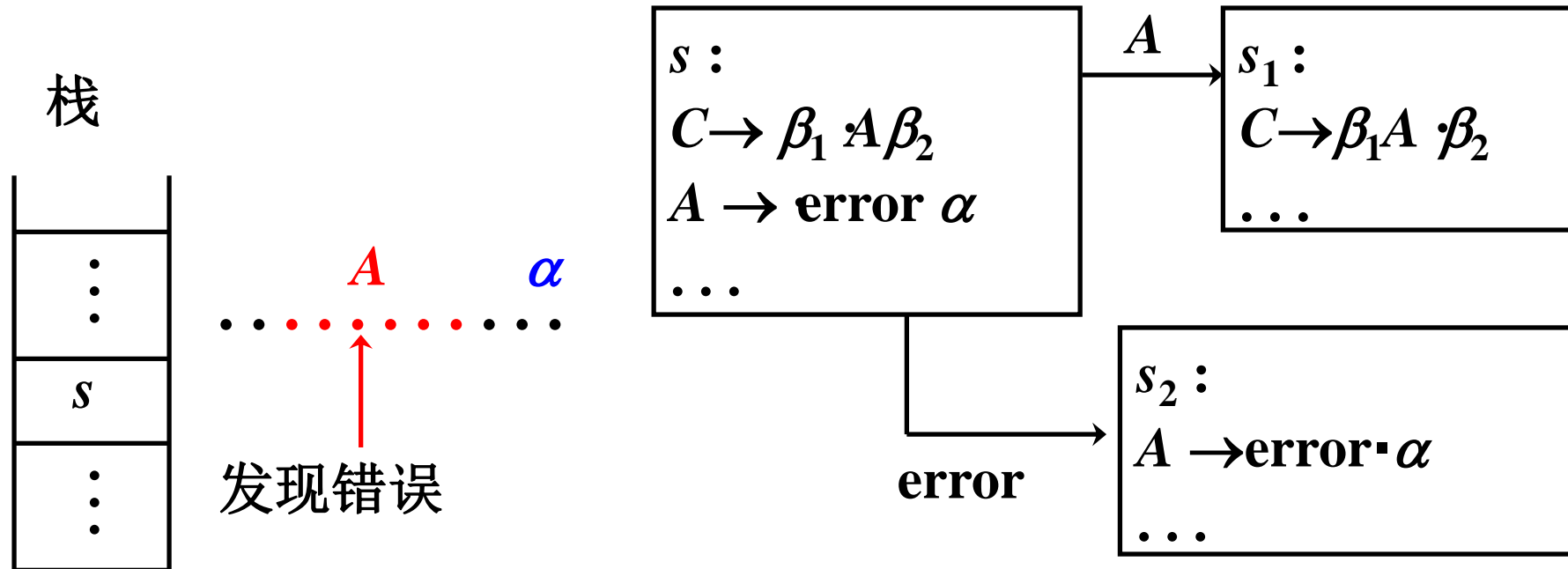
```
yylex ( ) {  
    int c;  
    while ( ( c = getchar ( ) ) == ' ' );  
    if ( ( c == '.' ) || (isdigit (c) ) ) {  
        ungetc (c, stdin);  
        scanf ( "%lf", &yylval);  
        return NUMBER;  
    }  
    return c;  
}
```

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

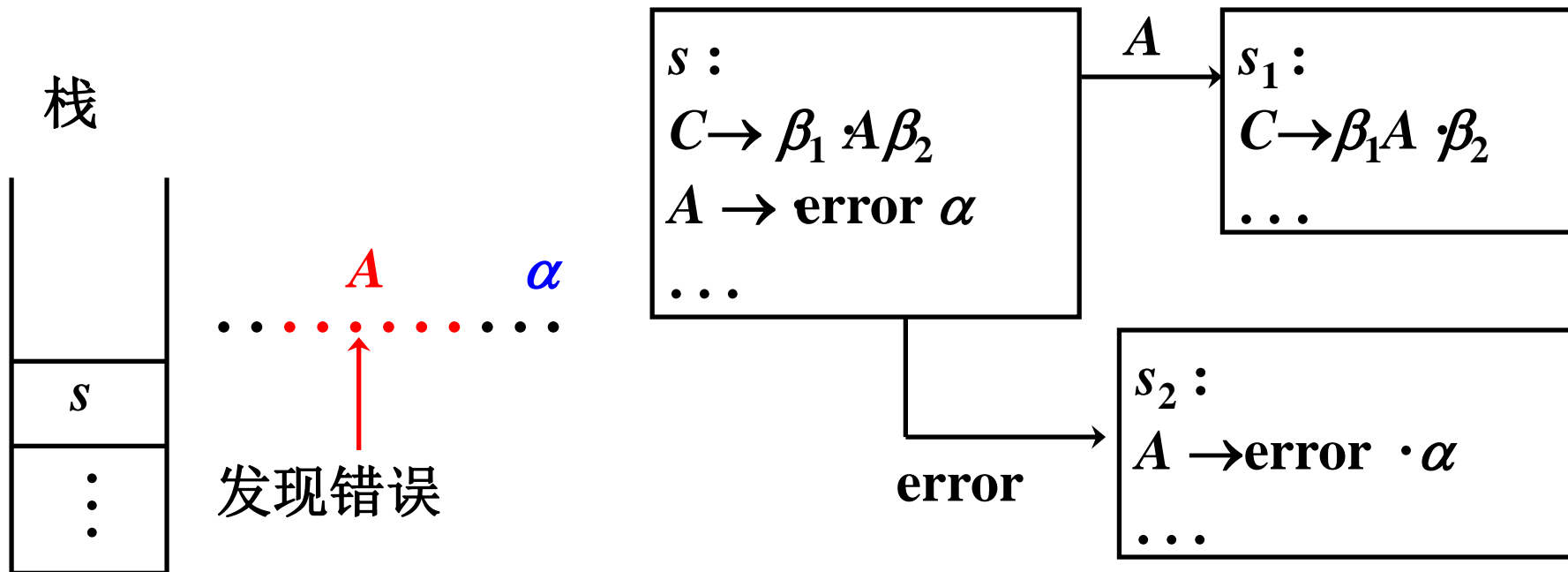




YACC的错误恢复

□ 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

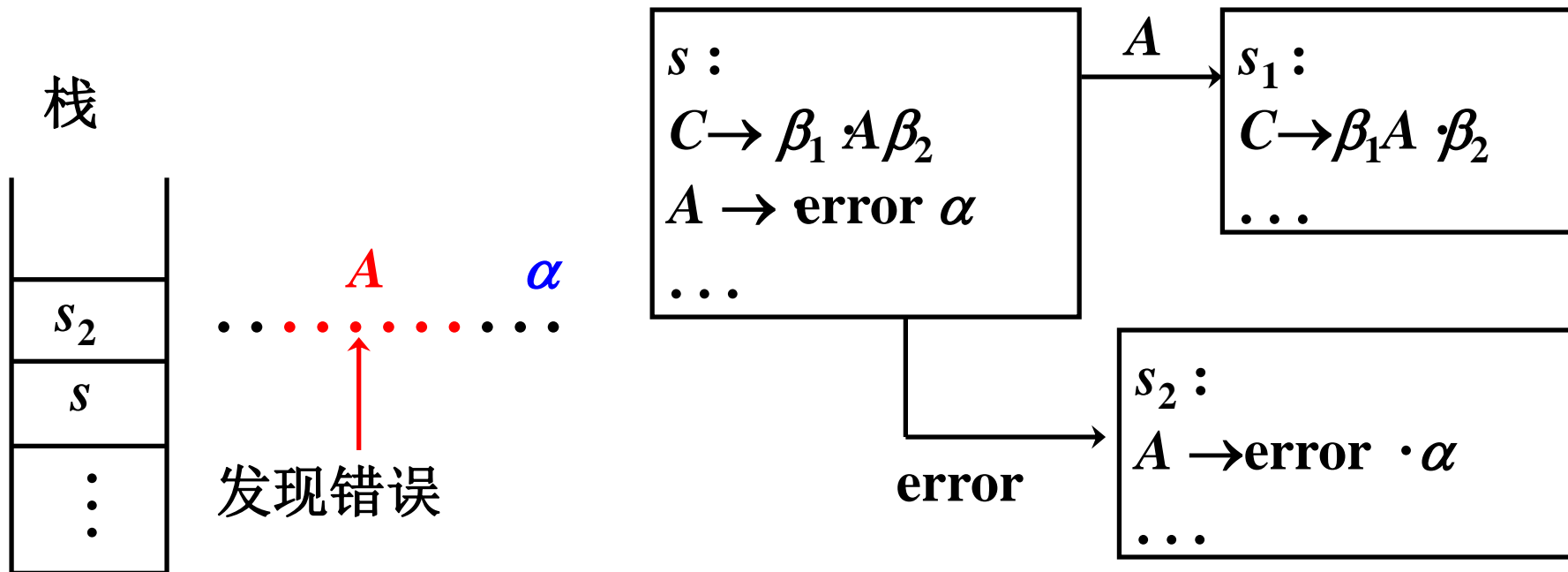




YACC的错误恢复

□ 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

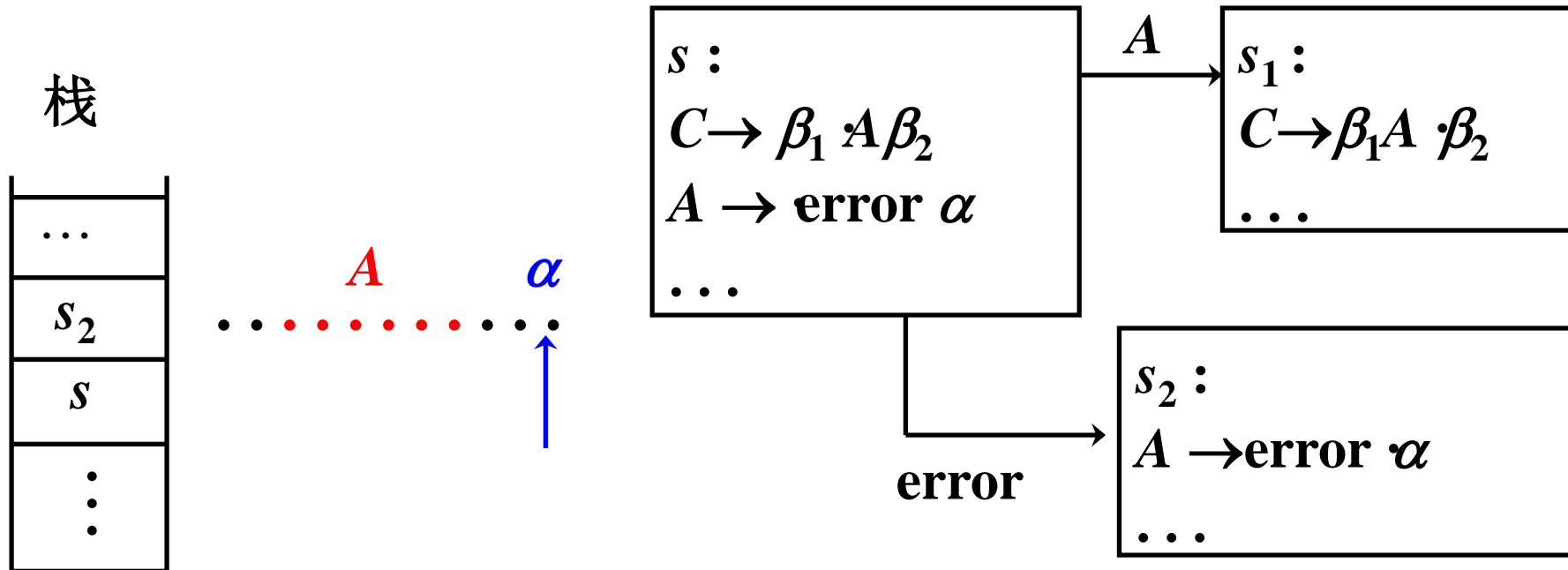




YACC的错误恢复

□ 遇到语法错误时

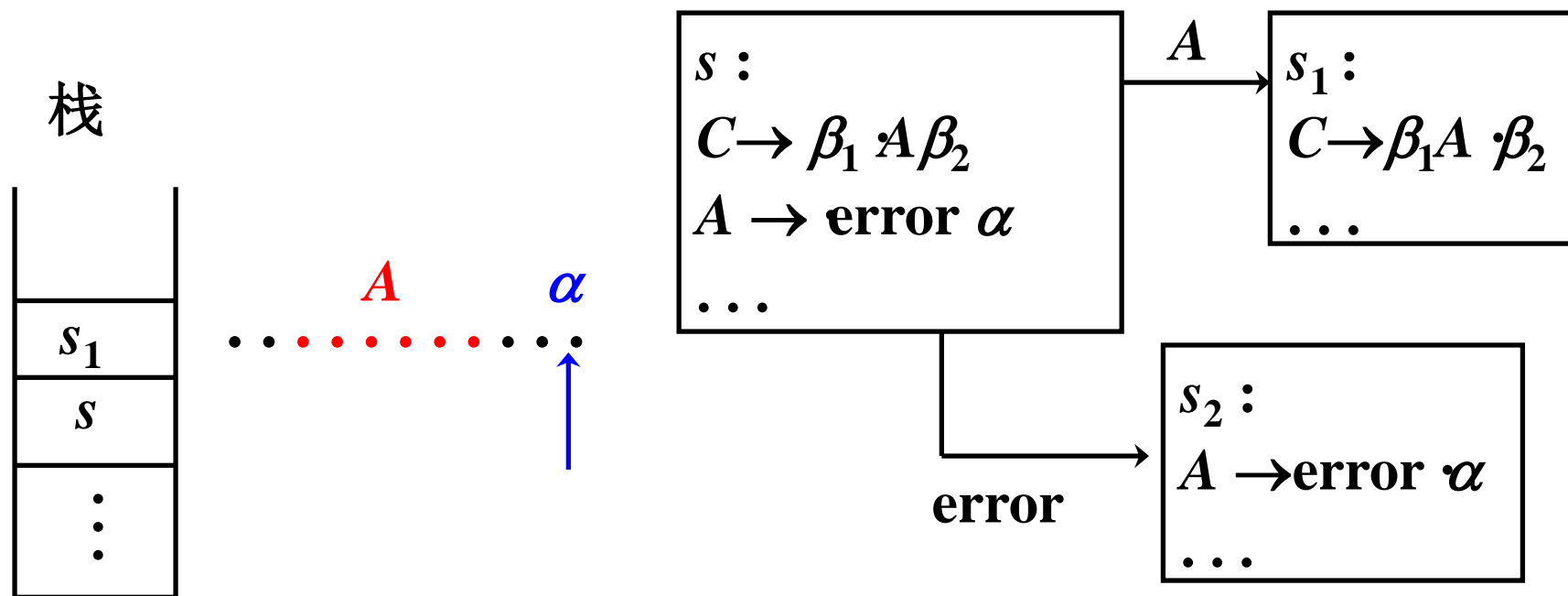
- 从栈中弹出状态，直到发现栈顶状态的项目集包含形为 $A \rightarrow \text{error } \alpha$ 的项目为止
- 把虚构的终结符error “移进” 栈
- 忽略若干输入符号，直至找到 α ，把 α 移进栈





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'  
           | /*  $\epsilon$  */  
           | error '\n'{yyerror ( “重新输入上一行” );  
                               yyerrok;}  
           ;
```




本章要点

- 语法分析器的作用和接口，用高级语言编写语法分析器（递归下降的预测解析器）等
- 掌握下面的相关概念、方法或算法
 - 上下文无关文法：定义、与正规式相比的表达能力
 - 推导、归约；文法、句子、句型；句柄、活前缀等
 - 自上而下的解析：递归下降的预测分析、非递归的预测分析、LL(*)
 - 自下而上的解析：SLR、LR、LALR
 - 二义文法的利用
 - 解析器的生成工具：ANTLR、YACC