



# 中间代码生成

## Part1: 简单语句的翻译

李 诚

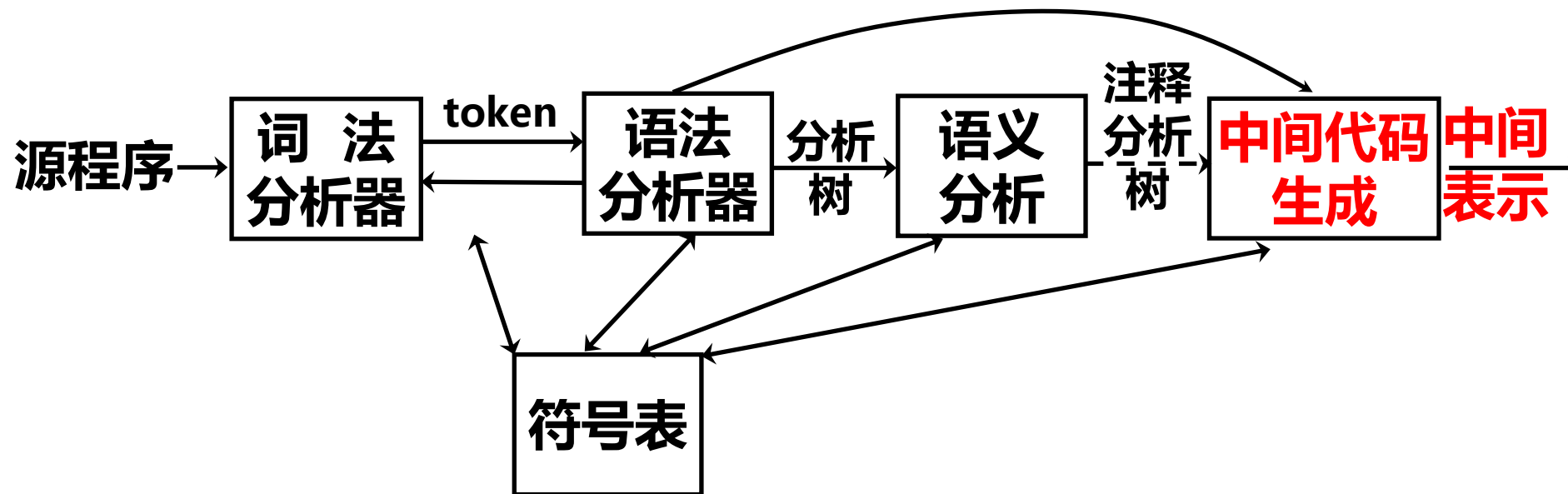
国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2023年10月17日



# 本节提纲



- 中间代码生成简介
- 表达式、switch语句、过程或函数的翻译



## • 常用的三地址语句

- 运算/赋值语句  $x = y \text{ op } z, \quad x = \text{op } y, \quad x = y$
- 无条件转移  $\text{goto } L$
- 条件转移1  $\text{if } x \text{ goto } L, \text{ if False } x \text{ goto } L$
- 条件转移2  $\text{if } x \text{ relop } y \text{ goto } L$



## • 常用的三地址语句

### • 过程调用

- **param  $x_1$**  //设置参数
- **param  $x_2$**
- ...
- **param  $x_n$**
- **call p, n** //调用子过程p, n为参数个数

### • 过程返回 **return y**

### • 索引赋值 **$x = y[i]$ 和 $x[i] = y$**

- 注意:  $i$ 表示距离 $y$ 处 $i$ 个内存单元

### • 地址和指针赋值 **$x = \&y$ , $x = *y$ 和 $*x = y$**



- 表达式的翻译
- switch语句的翻译
- 过程或函数的翻译
- 控制流语句的翻译
- 类型检查
- 声明语句的翻译
- 记录或结构体的翻译
- 数组寻址的翻译

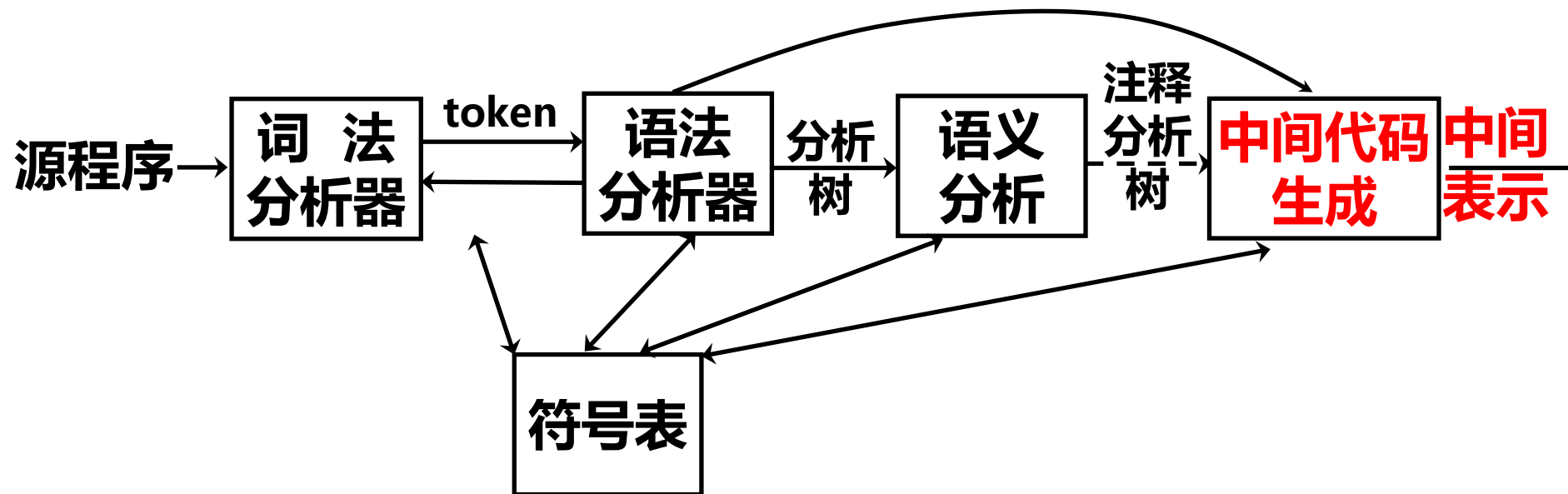
包含布尔表达式计算，较复杂，因此，单独讲

涉及到类型表达式、静态相对地址分配组织，因此，单独设计一个章节来讲

核心技术：  
语法制导翻译



# 本节提纲



- 中间代码生成简介
- 表达式、switch语句、过程或函数的翻译



## • 表达式文法

$S \rightarrow \text{id} := E \quad E \rightarrow E_1 + E_2 \mid -E_1 \mid (E_1) \mid \text{id}$

## • 语法制导翻译方案

- 属性:  $E.place$  存放结果的地址
- 语义动作: 从符号表中获取id的地址
  - $lookup(id.lexeme)$ ; 如果不存在, 返回  $nil$
- 语义动作: 产生临时变量
  - $newTemp()$ ; 保存中间结果
- 语义动作: 输出翻译后的三地址指令
  - $gen(addr, op, arg1, arg2)$
  - 该动作将地址和运算符及临时变量拼接为字符串



# 表达式的中间代码翻译方案



$S \rightarrow id := E$        $\{p = \textit{lookup}(id.lexeme);$   
                          if  $p \neq nil$  then  
                                     $\textit{gen} (p, '=', E.place)$   
                          else  $\textit{error}$  }

$E \rightarrow E_1 + E_2$   
           $\{E.place = \textit{newTemp}();$   
           $\textit{gen} (E.place, '=', E_1.place, '+', E_2.place) \}$





# 表达式的中间代码翻译方案

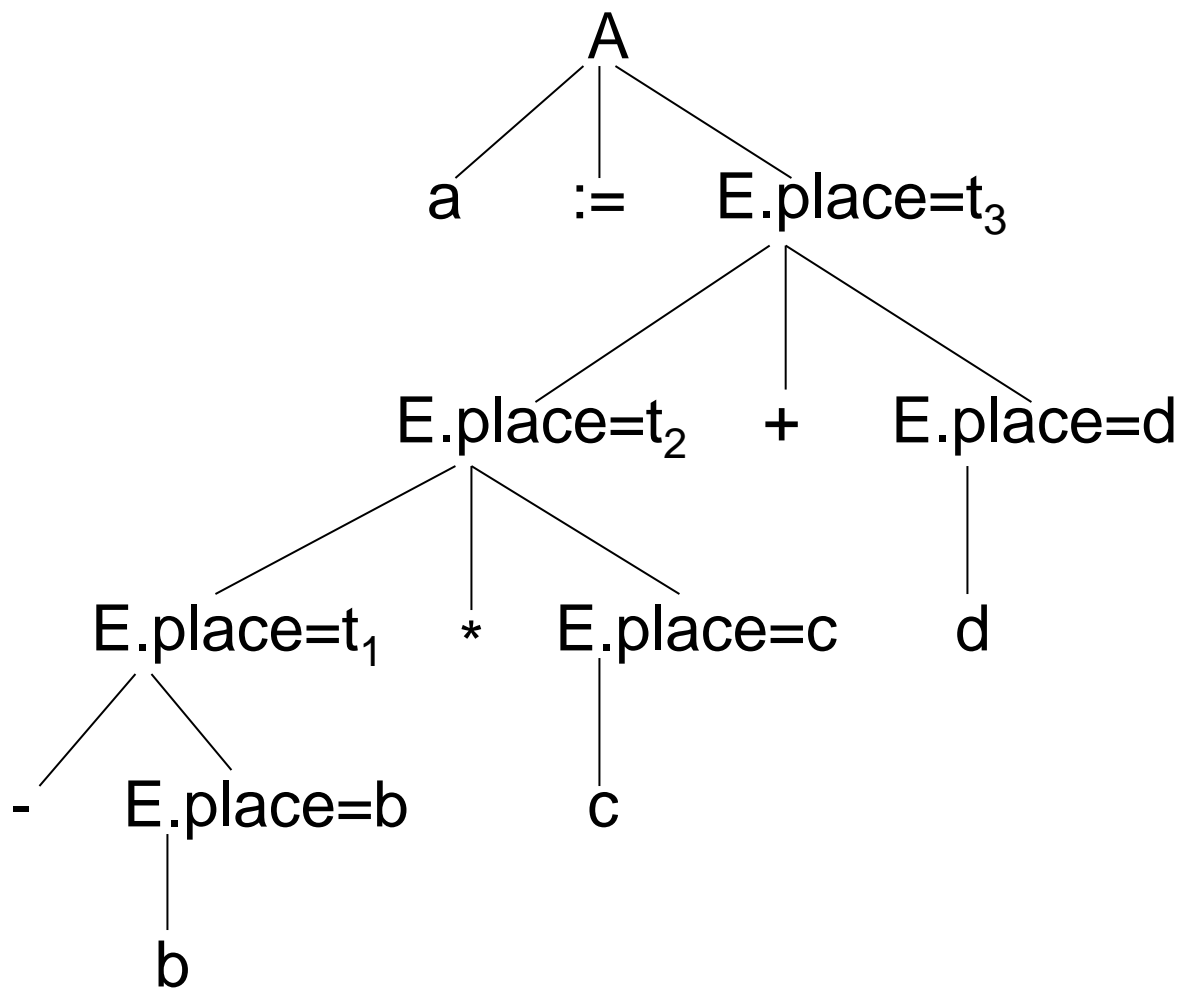

$$E \rightarrow -E_1 \{ \textcolor{blue}{E.place = newTemp()};$$
$$\textcolor{red}{gen (E.place, '=', 'uminus', E_1.place) } \}$$
$$E \rightarrow (E_1) \{ \textcolor{blue}{E.place = E_1.place} \}$$
$$E \rightarrow \text{id} \quad \{ \textcolor{blue}{p = lookup(id.lexeme)};$$
$$\text{if } p \neq nil \text{ then}$$
$$\quad \textcolor{blue}{E.place = p}$$
$$\text{else } \textcolor{red}{error} \}$$



# 举例：表达式翻译



## • $a := -b * c + d$ 的翻译



TAC:

1)  $t_1 := -b$

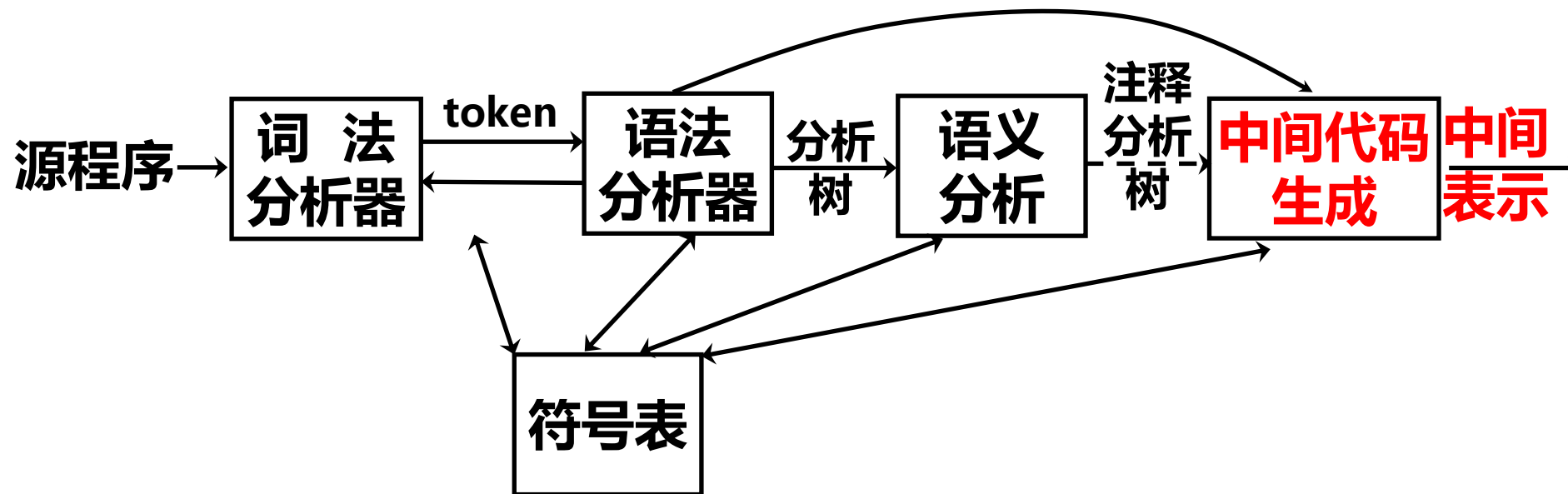
2)  $t_2 := t_1 * c$

3)  $t_3 := t_2 + d$

4)  $a := t_3$



# 本节提纲



- 中间代码生成简介
- 表达式、switch语句、过程或函数的翻译



# switch语句的文法



```
switch ( $E$ ){ //E是一个选择表达式
    case  $V_1$ :  $S_1$ //  $V$ 是常量,  $S$ 是语句
    case  $V_2$ :  $S_2$ 
    ...
    case  $V_{n-1}$ :  $S_{n-1}$ 
    default:  $S_n$ 
}
```



```
switch (E){ //E是一个选择表达式  
    case  $V_1$ :  $S_1$  // V是常量, S是语句  
    case  $V_2$ :  $S_2$   
    ...  
    case  $V_{n-1}$ :  $S_{n-1}$   
    default:  $S_n$   
}
```

三地址代码形态:

- 计算E的值的代码
- $S_j$ 语句的代码
- 匹配E的值与 $V_j$ , 并执行对应 $S_j$ 的逻辑



- 分支数较少时

*E*的代码

$t = E.place$

if  $t \neq V_1$  goto  $L_1$

$S_1$ 的代码

goto next

$L_1$ : if  $t \neq V_2$  goto  $L_2$

$S_2$ 的代码

goto next

$L_2$ : ...

...

|  $L_{n-2}$ : if  $t \neq V_{n-1}$  goto  $L_{n-1}$

|  $S_{n-1}$ 的代码

| goto next

|  $L_{n-1}$ :  $S_n$ 的代码

| next:



- 分支较多时，将分支测试代码集中在一起，便于生成较好的分支测试代码

	$t = E.place$	$L_n: S_n$ 的代码
	goto test	goto next
$L_1:$	$S_1$ 的代码	test: if $t == V_1$ goto $L_1$
	goto next	if $t == V_2$ goto $L_2$
$L_2:$	$S_2$ 的代码	...
	goto next	if $t == V_{n-1}$ goto $L_{n-1}$
	...	goto $L_n$
$L_{n-1}:$	$S_{n-1}$ 的代码	next:
	goto next	



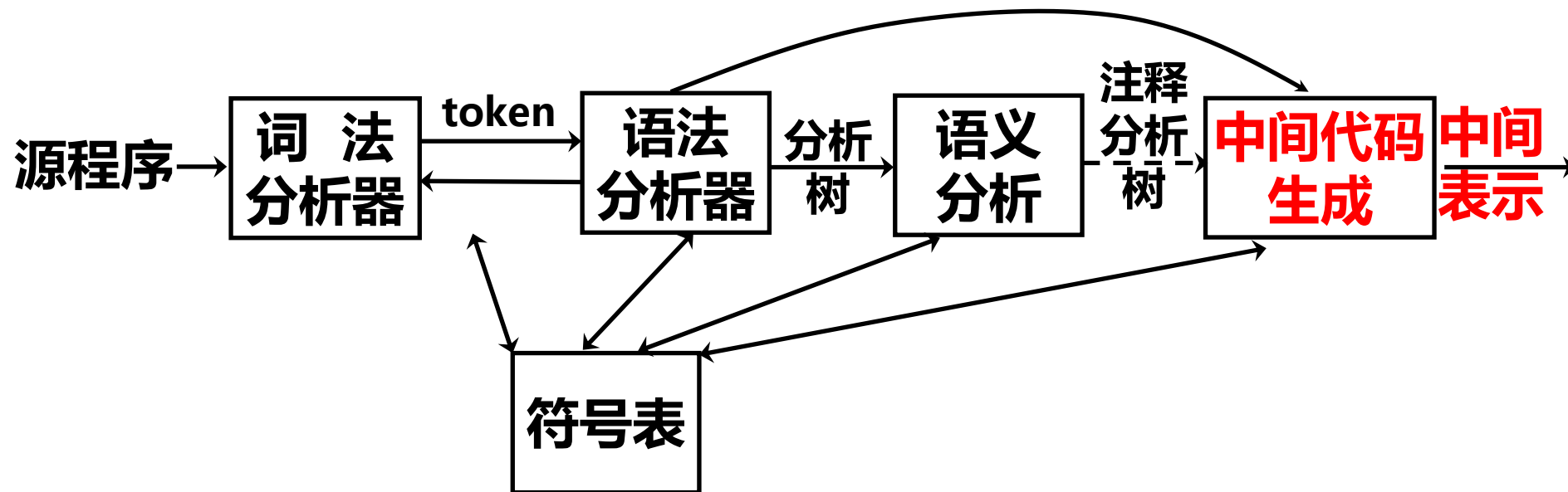
- 中间代码表示增加一种case语句，与之前的翻译等价，便于代码生成器对它进行特别处理

```
test: case t  $V_1$     $L_1$   
      case t  $V_2$     $L_2$   
      ...  
      case t  $V_{n-1}$   $L_{n-1}$   
      case t t       $L_n$   
next:
```





# 本节提纲



- 中间代码生成简介
- 表达式、switch语句、过程或函数的翻译



# 过程调用的翻译



$S \rightarrow \text{call id } (Elist)$

$Elist \rightarrow Elist, E$

$Elist \rightarrow E$

此处先忽略函数及过程的定义，会在后面的课程中讲解。



• 过程调用  $\text{id}(E_1, E_2, \dots, E_n)$  的中间代码结构

$E_1$  的代码

$E_2$  的代码

...

$E_n$  的代码

param  $E_1.place$

param  $E_2.place$

...

param  $E_n.place$

call  $\text{id}.place, n$



为 $Elist$ 设计一个综合属性 $paramlist$ , 该列表记录函数调用的所有参数, 且参数排列顺序与传参顺序一致

$Elist \rightarrow E$

$\{Elist.paramlist = createEmptyList();$

$Elist.push\_back(E.place); \}$



为 $Elist$ 设计一个综合属性 $paramlist$ ，该列表记录函数调用的所有参数，且参数排列顺序与传参顺序一致

$Elist \rightarrow E$

```
{ $Elist.paramlist = createEmptyList();$   
   $Elist.paramlist.push\_back(E.place);$ }
```

$Elist \rightarrow Elist_1, E$

```
{ $Elist.paramlist = Elist_1.paramlist;$   
   $Elist.paramlist.push\_back(E.place);$ }
```



# 过程调用的翻译方案



$S \rightarrow \text{call id } (Elist)$

{for  $E_i$  in  $Elist.paramlist$ :

$gen(\text{'param'}, E_i.place);$

$gen(\text{'call'}, id.place, Elist.paramlist.size()) ; \}$



# 一起努力 打造国产基础软硬件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2023年10月17日