

# 第七章 语法制导翻译并产生中间代码

廖力

xobjects@seu.edu.cn

3793235

# 第一节 概述

- 语法分析之后，编译的任务是由已识别为正确的源程序生成一组规格一致，便于计算机加工的指令形式。

## 一、中间代码生成方法

语法制导翻译，属性文法制导翻译

## 二、中间代码

- 中间代码：不是机器语言，便于生成机器语言，便于代码优化。
- 中间代码的形式：
  - 逆波兰式
  - 树形表示法
  - 三元式
  - 四元式：最常用的形式

# 第一节 概述

## 二、翻译方法

### 1、属性文法制导翻译

将文法的终结符、非终结符及动作符号附加以语义参数，这些参数称作为文法符号的属性，从而构成属性文法。

- 注：1)在属性文法中是利用各种属性作为语义动作之间的通信介质

2)一个文法符号可以和多个语义参数相关联

3)属性可分为两种：继承属性和综合属性。继承属性值的计算规则是自上而下——产生式右部符号的某些值是根据其左部符号的属性和右部其它符号的某些属性计算而得。

# 第一节 概述

## 二、翻译方法

### 1、属性文法制导翻译

- 注：3) 综合属性值的计算规则是自下而上——产生式左部符号的某些属性值是根据其右部符号的属性和左部其它符号的某些属性计算而得。
- 4) 属性翻译的依据是属性规则。
- 5) 属性规则嵌入在产生式右部适当位置。

# 第一节 概述

## 二、翻译方法

### 2、语法制导翻译

- 在语法分析的基础上进行边分析边翻译。
- 注：1)语法制导翻译时会根据文法产生式右部符号串的含义，进行翻译，翻译的结果是生成相应中间代码。
- 2)语法制导翻译的依据是语义子程序。
- 3)具体做法：为每个产生式配置一个语义子程序，当语法分析进行归约或推导时，调用语义子程序，完成一部分翻译任务。
- 4)语法分析完成，翻译工作也告结束。

# 第一节 概述

## 二、翻译方法

- 2、语法制导翻译

- 语法制导翻译适用于多种语法分析。

- 语法制导翻译种类

- 1、自上而下语法制导翻译：对每个文法符号配以语义动作。
- 2、自下而上语法制导翻译：我们主要讨论LR语法制导翻译。

# 第一节 概述

## 三、语义子程序

### 1、作用

- 用来描述一个产生式所对应的翻译工作。
  - 如：改变某些变量的值；查填各种符号表；发现并报告源程序错误；产生中间代码等。
- 注：这些翻译工作很大程度上决定了要产生什么形式的中间代码。

# 第一节 概述

## 三、语义子程序

### 2、写法

- 语义子程序写在该产生式后面的花括号内。

$X \rightarrow \alpha \quad \{\text{语义子程序1}\}$

- 注：在一个产生式中同一个文法符号可能出现多次，但他们代表的是不同的语义值，要区分可以加上角标。

如： $E \rightarrow E^{(1)} + E^{(2)}$

### 3、语义值

- 为了描述语义动作，需要为每个文法符号赋予不同的语义值：类型、地址、代码值等。



# 第一节 概述

## 三、语义子程序

### 4、语义栈

- 各个符号的语义值放在语义栈中
  - 当产生式进行归约时，需对产生式右部符号的语义值进行综合，其结果作为左部符号的语义值保存到语义栈中。
- 下推栈包含3部分：
  - 状态栈、符号栈和语义栈
  - 注：语义栈与状态栈和符号栈是同步变化的。

# 第一节 概述

## 三、语义子程序

注：1)若把语义子程序改成产生某种中间代码的动作，就能在语法分析制导下，随着分析的进展逐步生成中间代码。

2)若把语义子程序改成产生某种机器的汇编语言指令，就能随着分析的进展逐步生成某机器的汇编语言代码。

# 第一节 概述

例如：

- | 产生式                                     | 语义子程序  |
|---|--|
| • $(0) S' \rightarrow E$                | $\{\text{PRINT } E \bullet \text{VAL}\}$   |
| • $(1) E \rightarrow E^{(1)} + E^{(2)}$ | $\{E \bullet \text{VAL} = E^{(1)} \bullet \text{VAL} + E^{(2)} \bullet \text{VAL}\}$ |
| • $(2) E \rightarrow E^{(1)} * E^{(2)}$ | $\{E \bullet \text{VAL} = E^{(1)} \bullet \text{VAL} * E^{(2)} \bullet \text{VAL}\}$ |
| • $(3) E \rightarrow (E^{(1)})$         | $\{E \bullet \text{VAL} = E^{(1)} \bullet \text{VAL}\}$                              |
| • $(4) E \rightarrow i$                 | $\{E \bullet \text{VAL} = \text{LEXVAL}\}$   |
| • 注：LEXVAL指的是词法分析送来的机内二进制整数。            |  |

$S_m$	$E^{(2)}$	$E^{(2)} \bullet VAL$	<div>←</div> <div>TOP</div>
$S_{m-1}$	+		
	$E^{(1)}$	$E^{(1)} \bullet VAL$	
.....	.....	.....	
$S_0$	#		
状态	符号	VAL (语义)	

## 第一节 概述

- 注：由于语义值是放在语义栈中的，它也可以用栈顶指针TOP指出，语义子程序改写为：
- | 产生式                                   | 语义子程序                              |
|---------------------------------------|------------------------------------|
| $(0) S' \rightarrow E$                | {PRINT VAL[TOP]}                   |
| $(1) E \rightarrow E^{(1)} + E^{(2)}$ | {VAL[TOP] = VAL[TOP] + VAL[TOP+2]} |
| $(2) E \rightarrow E^{(1)} * E^{(2)}$ | {VAL[TOP] = VAL[TOP] * VAL[TOP+2]} |
| $(3) E \rightarrow (E^{(1)})$         | {VAL[TOP] = VAL[TOP+1]}            |
| $(4) E \rightarrow i$                 | {VAL[TOP] = LEXVAL }               |

# 第一节 概述

- 例如：分析输入串  $(7 + 9) * 5 \#$ , 并给出它的计值过程。分析表如下：
- P119 图6.6

状态	ACTION						GOTO
	i	+	*	(	)	#	S
0	S <sub>3</sub>			S <sub>2</sub>			1
1		S <sub>4</sub>	S <sub>5</sub>			acc	
2	S <sub>3</sub>			S <sub>2</sub>			6
3		r <sub>4</sub>	r <sub>4</sub>		r <sub>4</sub>	r <sub>4</sub>	
4	S <sub>3</sub>			S <sub>2</sub>			7
5	S <sub>3</sub>			S <sub>2</sub>			8
6		S <sub>4</sub>	S <sub>5</sub>		S <sub>9</sub>		
7		r <sub>1</sub> (S <sub>4</sub> )	S <sub>5</sub> (r <sub>1</sub> )		r <sub>1</sub>	r <sub>1</sub>	
8		r <sub>2</sub> (S <sub>4</sub> )	r <sub>2</sub> (S <sub>5</sub> )		r <sub>2</sub>	r <sub>2</sub>	
9		r <sub>3</sub>	r <sub>3</sub>		r <sub>3</sub>	r <sub>3</sub>	

步骤	状态	SYM	VAL	INPUT	ACTION
1	0	#	-	( 7+9 ) *5#	
2	0,2	#(	- -	7+9 ) *5#	移进
3	0,2,3	#(7	- - -	+9 ) *5#	移进
4	0,2,6	#(E	- - 7	+9 ) *5#	r4
5	0,2,6,4	#(E+	- - 7 -	9 ) *5#	移进
6	0,2,6,4,3	#(E+9	- - 7 - -	) *5#	移进
7	0,2,6,4,7	#(E+E	- - 7 - 9	) *5#	r4
8	0,2,6	#(E	- - 16	) *5#	r1
9	0,2,6,9	#(E)	- - 16 -	*5#	移进
10	0,1	#E	- 16	*5#	r3
11	0,1,5	#E*	- 16 -	5#	移进



# 第一节 概述

## 四、常见的中间代码形式

### 1、四元式形式:

(Operator,Operand1,Operand2,Result)

注：1) Operand1,Operand2,Result 可能是用户自定义的变量，也可能是编译时引进的变量。这里Operator是双目运算符，若只有一个运算量，则是单目运算符。

- 2) 四元式中变量采用符号表入口的地址，而不用变量的地址，因为语义分析不仅需要变量的地址，还需要从符号表查到的变量的属性、类型和地址等。
- 3) 四元式的优点是容易转换为目标代码和容易进行优化。

# 第一节 概述

## 四、常见的中间代码形式

### 2、三元式

(Operator,Operand1,Operand2)

注：1)这里三元式本身作为存放结果的单元。

2)为了在其它三元式中利用当前三元式的结果，需要对三元式进行编号。三元式的编号就作为相应三元式的结果值。

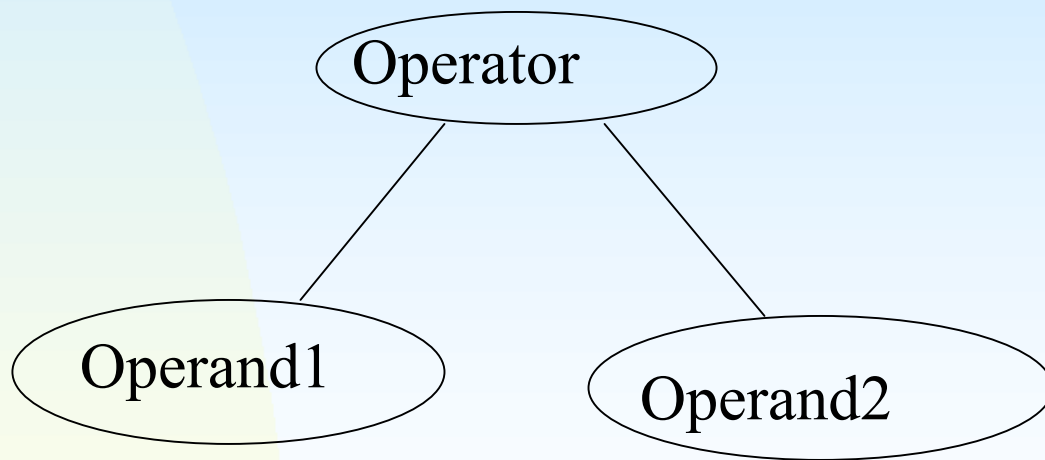
# 第一节 概述

## 四、常见的中间代码形式

### 3、后缀表示式(逆波兰表达式)

Operand1 Operand2 Operator

### 4、树形表示法



注：常用中间代码表示法是四元式。

## 第二节 简单算术表达式和赋值语句的翻译

### 一、赋值语句的翻译

#### – 仅含简单变量的表达式和赋值语句的翻译

#### 1、赋值语句的文法

- $A \rightarrow i=E$
- $E \rightarrow E+E|E * E|-E|(E)|i$

#### 2、需要的语义过程

- NEWTEMP函数：每次调用送回一个代表新临时变量的序号，可认为是送回 $T_1$ 、 $T_2$ 这样的一些临时变量；
- ENTRY(i)函数：用于查变量i的符号表入口地址；
- GEN ( OP,ARG<sub>1</sub>,ARG<sub>2</sub>,RESULT)过程：产生一个四元式，并填入四元式序列表。

## 第二节 简单算术表达式和赋值语句的翻译

### 一、赋值语句的翻译

#### 3、需要的语义变量

- $E \bullet PLACE$ ：与非终结符 $E$ 相联系的语义变量
  - 值为某变量的符号表入口地址或临时变量序号。
  - 它与文法的非终结符相联，分析过程需要就建立，不需要就消亡。

- | 产生式                                     | 语义子程序   |
|---|---|
| • (1) $A \rightarrow i = E$             | $\{ \text{GEN}(=, E \bullet \text{PLACE}, \_, \text{ENTRY}(i)) \}$              |
| • (2) $E \rightarrow -E^{(1)}$          | $\{ T = \text{NEWTEMP};$  |
| —                                       | $\text{GEN}(@, E^{(1)} \bullet \text{PLACE}, \_, T);$                           |
| —                                       | $E \bullet \text{PLACE} = T \}$   |
| • (3) $E \rightarrow E^{(1)} * E^{(2)}$ | $\{ T = \text{NEWTEMP};$  |
| —                                       | $\text{GEN}(*, E^{(1)} \bullet \text{PLACE}, E^{(2)} \bullet \text{PLACE}, T);$ |
| —                                       | $E \bullet \text{PLACE} = T \}$   |
| • (4) $E \rightarrow E^{(1)} + E^{(2)}$ | $\{ T = \text{NEWTEMP};$  |
| —                                       | $\text{GEN}(+, E^{(1)} \bullet \text{PLACE}, E^{(2)} \bullet \text{PLACE}, T);$ |
| —                                       | $E \bullet \text{PLACE} = T \}$   |
| • (5) $E \rightarrow (E^{(1)})$         | $\{ E \bullet \text{PLACE} = E^{(1)} \bullet \text{PLACE} \}$                   |
| • (6) $E \rightarrow i$                 | $\{ E \bullet \text{PLACE} = \text{ENTRY}(i) \}$                                |

输入符号串	SYM栈	PLACE栈	生成四元式
A=-B*(C+D)#			
=-B*(C+D)#	i	-	
-B*(C+D)#	i=	- -	
B*(C+D)#	i=-	- - -	
*(C+D)#	i=-i	- - -	
*(C+D)#	i=-E	- - - B	
*(C+D)#	i=E	- - T <sub>1</sub>	((@,B, -, T <sub>1</sub> ))
(C+D)#	i=E*	- - T <sub>1</sub> -	
C+D)#	i=E*(	- - T <sub>1</sub> - -	
+D)#	i=E*(i	- - T <sub>1</sub> - - C	
+D)#	i=E*(E	- - T <sub>1</sub> - - C	

- 注:1、符号栈是为了介绍才列出的，实际上并不存在。
- 2、由于语义分析是在语法分析的过程中进行的，所以状态栈实际上是需要，这里没有写出来。
- 3、这个分析过程是针对整型变量做的。实际上在一个表达式中，可能出现各种不同类型的变量或常量，所以，编译程序要么拒绝接受某种混合运算，要么能产生有关类型转换的指令。



## 第二节 简单算术表达式和赋值语句的翻译

### 二、类型转换

- 对于表达式文法中的 $i$ 既是实型又可以是整型。
- 对这种混合运算，我们要先把整型量转化为实型量，就需要为每个非终结符的语义值添加类型信息，用 $E \cdot \text{MODE}$ 表示非终结符 $E$ 的类型信息。

#### 1、定义各非终结符的类型信息

- 产生式 $E \rightarrow E^{(1)} \text{ op } E^{(2)}$ 的语义动作中要加入关于 $E \cdot \text{MODE}$ 的语义规则的定义：
  - $\{ \text{IF } E^{(1)} \cdot \text{MODE} = \text{int AND } E^{(2)} \cdot \text{MODE} = \text{int}$
  - $\text{ THEN } E \cdot \text{MODE} = \text{int ELSE } E \cdot \text{MODE} = \text{r} \}$

## 第二节 简单算术表达式和赋值语句的翻译

### 2、对运算量进行类型转换

- 例如： $X=Y+I*J$ ,其中 $X$ 、 $Y$ 是实型， $I$ 、 $J$ 是整型。其四元式为：

- $(*^i, I, J, T_1)$

- $(itr, T_1, \_, T_2)$

- $(+^r, Y, T_2, T_3)$

- $(=, T_3, \_, X)$

## 第二节 简单算术表达式和赋值语句的翻译

- 注：1) 对运算符也要指出相应的类型（运算符上加角标），说明是定点还是浮点运算。
- 2) 由于非终结符 $E$ 的语义值有 $E \cdot \text{PLACE}$  和  $E \cdot \text{MODE}$  两个，这两者都要保存在语义栈中。
- 3) 在运算量的类型较多的情况下，需要仔细推敲语义规则，否则语义子程序会变得累赘不堪。

## 第三节 布尔表达式的翻译

### 一、布尔表达式在程序设计语言中的作用

- 用作控制语句中的条件表达式；
- 用于逻辑赋值语句中布尔表达式演算。

### 二、布尔表达式的组成

- 由布尔算符作用于布尔变量（或常量）或关系表达式而形成的。
- 布尔算符：  $\neg$  ,  $\wedge$  ,  $\vee$  ,  $\rightarrow$  ,  $\leftrightarrow$  ,
- 关系表达式的形式：  $E^{(1)} \text{ rop } E^{(2)}$  , 其中rop是关系运算符(如 $<$ ,  $\leq$ ,  $=$ ,  $<>$ ,  $\geq$ ,  $>$ ) ,  $E^{(1)}$  和 $E^{(2)}$ 是算术表达式。

## 第三节 布尔表达式的翻译

### 三、布尔表达式文法：

- $G(B): E \rightarrow E \mid E \mid E \mid (E) \mid i \mid E_a \text{ rop } E_a$
- $i$  为布尔变量或常量； $E_a$  为算术表达式。
- 注：1) 此文法为二义文法，一般布尔算符的优先顺序为：，，；且，服从左结合律；关系运算优先级别高于布尔运算。
- 2) 由于布尔表达式有两种用途，所以有两种不同的翻译方法。

## 第三节 布尔表达式的翻译

### 四、布尔表达式在逻辑演算中的翻译

#### 1、语义子程序

- 由于布尔表达式演算与算术表达式计算非常类似，可以仿照算术表达式的翻译方法，为每个产生式写出语义子程序：

## 产生式

## 语义子程序

- (1)  $E \rightarrow E_a^{(1)} \text{ rop } E_a^{(2)}$        $\{T = \text{NEWTEMP};$   
    $\text{GEN}(\text{rop}, E_a^{(1)} \bullet \text{PLACE}, E_a^{(2)} \bullet \text{PLACE}, T);$   
    $E \bullet \text{PLACE} = T \}$
- (2)  $E \rightarrow E^{(1)} \text{ bop } E^{(2)}$        $\{T = \text{NEWTEMP};$   
    $\text{GEN}(\text{bop}, E_a^{(1)} \bullet \text{PLACE}, E_a^{(2)} \bullet \text{PLACE}, T);$   
    $E \bullet \text{PLACE} = T \}$
- (3)  $E \rightarrow E^{(1)}$        $\{T = \text{NEWTEMP};$   
    $\text{GEN}(\_, E^{(1)} \bullet \text{PLACE}, \_, T);$   
    $E \bullet \text{PLACE} = T \}$
- (4)  $E \rightarrow (E^{(1)})$        $\{E \bullet \text{PLACE} = E^{(1)} \bullet \text{PLACE} \}$
- (5)  $E \rightarrow i$        $\{E \bullet \text{PLACE} = \text{ENTRY}(i) \}$

## 第三节 布尔表达式的翻译

### 四、布尔表达式在逻辑演算中的翻译

2、实例：对布尔式 $X+Y>Z \wedge (B \vee C)$ 进行翻译：

- 解：语法制导得翻译是在语法分析的过程中进行的，若利用G(B)文法的LR分析表对上句进行LR分析，在其过程中进行语义分析；则得到的四元式序列是：
- $(+, X, Y, T_1)$  ; E+E进行归约，识别了算术运算
- $(>, T_1, Z, T_2)$  ; E>E进行归约，识别了关系运算
- $(\vee, B, \_, T_3)$  ; E归约
- $(\vee, T_3, C, T_4)$  ; E E进行归约
- $(\wedge, A, T_4, T_5)$  ; E E进行归约
- $(\wedge, T_2, T_5, T_6)$  ; E E进行归约



## 第三节 布尔表达式的翻译

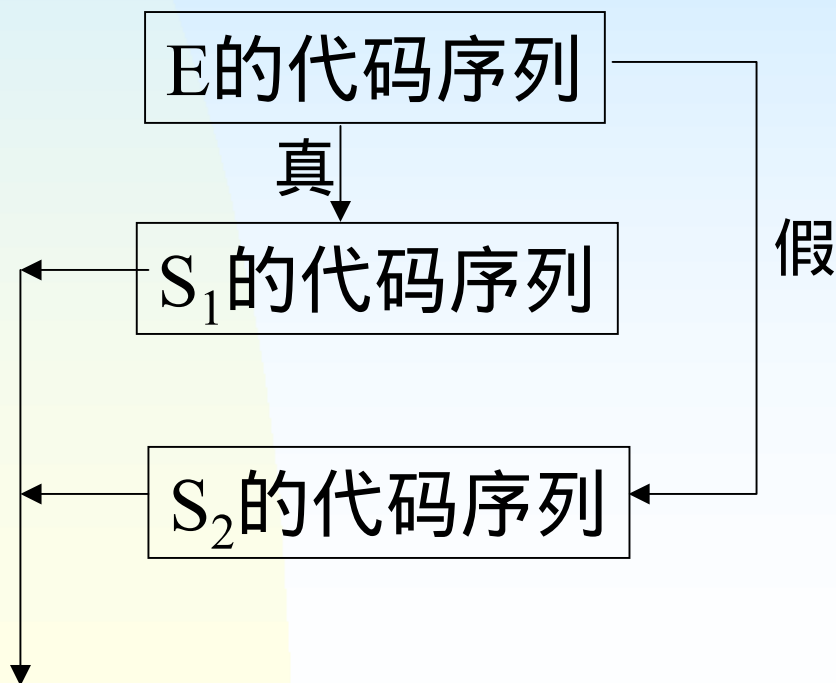
- 注：上面的翻译质量并不高，可以采取某种优化措施：
- 例如：
  - 对  $A \wedge B$  若已知A的值是1，那么B的值就不需要知道就能得出  $A \wedge B = 1$ ；
    - 可以表示为：IF A THEN TRUE ELSE B.
  - 对  $A \wedge B$  若已知A的值是0，那么B的值就不需要知道就能得出  $A \wedge B = 0$ ；
    - 可以表示为：IF A THEN B ELSE FALSE
  - 对  $\neg A$  ；
    - 可以表示为：IF A THEN FALSE ELSE TRUE.

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

1、控制语句中的布尔式并不需要计算表达式的值，只用来控制程序流向。

- if E then  $S_1$  else  $S_2$



## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

2、控制语句中布尔式E翻译成的四元式为以下三种：

- $(jnz, A_1, \_, P)$ 
  - If  $A_1$  then goto P
- $(j\theta, A_1, A_2, P)$ 
  - If  $A_1 \theta A_2$  then goto P
- $(j, \_, \_ P)$ 
  - Goto P
- 注：这些四元式都是转移四元式，其中P为出口的四元式序号。与E的真假值相对应的分别为“真出口”和“假出口”两类四元式。

- 例如：把语句if A B<D then S<sub>1</sub> else S<sub>2</sub> 翻译成四元式
- 解：(1)(jnz,A,\_,(5)) ;真出口；若A为真，执行S<sub>1</sub>代码
- (2)(j,\_,\_(3)) ;若A为假，看 右边的表达式值
- (3)(j<,B,D,(5)) ;真出口； 右边的表达式值为真
- (4)(j,\_,\_(P+1)) ;假出口； 右边的表达式值为假
- (5) S<sub>1</sub>语句的第一个四元式
- .....
- (P)(j,\_,\_(q)) ;执行完S<sub>1</sub>代码后跳过S<sub>2</sub>代码段
- (p+1) S<sub>2</sub>语句的第一个四元式
- .....
- (q)此语句的后继语句

## 第三节 布尔表达式的翻译

- 注：1、上面的1 ~ 4式是由布尔式“ $A \leq B < D$ ”产生的中间代码，但它们和逻辑演算中的翻译不同，消除了布尔运算，变成转移四元式。
- 2、翻译时每个布尔量是译为两个二元式的，但有些并不必要，所以可以在后续阶段进行优化。
- 3、在自下而上的分析过程中，一个布尔式的真假出口的转向序号不可能在产生转移四元式时就填上。此时将待填的转移四元式暂时保留，等出口明朗了，在进行回填。
- 4、真出口或假出口的四元式可能不止一个，所以要把他们拉链链接在一起，等到出口知道了再一起回填。

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 3、对文法的改写

- 原G(B):  $E \rightarrow E \quad E \mid E \quad E \mid \quad E \mid (E) \mid i \mid E_a \text{ rop } E_a$
- 由于产生式  $E \rightarrow E^{(1)} \quad E^{(2)}$  在  $E^{(1)} \rightarrow i$  归约时翻译成两个四元式，一个表示“真”转移，一个表示“假”转移；“假”转移要转到“ ”号后，即读完“ ”就可以回填假出口，所以可以进行产生式的改写
- $E \rightarrow E^{(1)} \quad E^{(2)}$  改写为  $E \rightarrow E^0 E^{(2)}$  ,  $E^0 \rightarrow E^{(1)}$
- $E \rightarrow E^{(1)} \quad E^{(2)}$  改写为  $E \rightarrow E^A E^{(2)}$  ,  $E^A \rightarrow E^{(1)}$

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 3、对文法的改写

故文法改写为： $G'(B)$ :

- $E \rightarrow E^A E \mid E^0 E \mid E \mid (E) \mid i \mid E_a \text{ rop } E_a$
- $E^A \rightarrow E$
- $E^0 \rightarrow E$

## 五、控制语句中布尔式的翻译

### 4、增加语义值：

- 由于归约每个非终结符时，产生两个四元式，分别对应于其语义为真和为假的情况，所以为了实现对转移四元式的拉链和回填，为每个非终结符赋予两个语义值： $E \cdot TC$ 、 $E \cdot FC$ 。
- $E \cdot TC$ 、 $E \cdot FC$ 分别记录非终结符 $E$ 需要回填“真”、“假”出口四元式的序号所构成的链。



## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 4、增加语义值：

- $E \cdot TC$ 、 $E \cdot FC$ 的值也可以放在语义栈中，即下推栈又增加了两栏。

	$E^{(1)}$			
	...			
$S_0$	#	—	—	—
S	SYM	PLACE	TC	FC
分析栈		语义栈		

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 5、文法G(B)各产生式的语义子程序

– 文法： $E \rightarrow E^A E \mid E^0 E \mid E \mid (E) \mid i \mid E_a \text{ rop } E_a$

–  $E^A \rightarrow E$

–  $E^0 \rightarrow E$

#### • 语义子程序：

– (1)  $E \rightarrow i$        $\{E \cdot TC = NXQ; E \cdot FC = NXQ + 1;$

–  $GEN(jnz, ENTRY(i), \_, 0);$

–  $GEN(j, \_, \_, 0)\}$

– 注：将布尔型操作数进行归约，产生两个四元式；  
无论真假都不知该转到哪个四元式上去，故转向0。

- 例如：将控制语句中的  $A \quad B < D$
- 翻译成四元式
- 解：(1)(jnz,A,\_,0) ;真出口；
- (2)(j,\_,\_,0) ;若A为假，看 右边的表达式值
- TC:1;
- FC:2;

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 5、文法G(B)各产生式的语义子程序

(7)  $E^0 \rightarrow E^{(1)}$

- $\{\text{BACKPATCH}(E^{(1)} \bullet \text{FC}, \text{NXQ});$
- $E^0 \bullet \text{TC} = E^{(1)} \bullet \text{TC};\}$

- 注：对 进行归约之后，若  $E^{(1)} = 0$ ，则 运算的结果就要看 右边式子的值了，所以表示  $E^{(1)} = 0$  的四元式应转移到 归约后的下一个四元式，即判断 右边式子的值的第一个四元式。

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 5、文法G(B)各产生式的语义子程序

PROCEDURE *BACKPATCH*(p,t)

- {Q=p;
- WHILE (Q<>0)
- {q=四元式Q第4段的内容;
- 把t填入四元式Q的第四段 ;
- Q=q;}}
- 注：其算法思想是:从链头填起，边找边填，直到链尾为止。

- 例如：将控制语句中的  $A \leq B < D$  翻译成四元式
- 解：(1)(jnz,A,\_,0) ;真出口；
- (2)(j,\_,\_,(3)) ;若A为假，看 右边的表达式值
- $E^0 \bullet TC:1$ ;
- FC已回填；

## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 5、文法G(B)各产生式的语义子程序

(2)  $E \rightarrow E_a \text{ rop } E_a$

- $\{E \cdot TC = NXQ; E \cdot FC = NXQ + 1;$
- $\text{GEN}(j_{\text{rop}}, E_a^{(1)} \cdot \text{PLACE}, E_a^{(2)} \cdot \text{PLACE}, 0);$
- $\text{GEN}(j, \_, \_, 0)\}$

- 例如：将控制语句中的  $A \quad B < D$  翻译成四元式
- 解：(1)(jnz,A,\_,0) ;真出口
- (2)(j,\_,\_,(3)) ;若A为假，看 右边的表达式值
- (3)(j<,B,D,0) ;真出口; 右边的表达式值为真
- (4)(j,\_,\_,0) ;假出口; 右边的表达式值为假
- E•TC:3;
- E•FC:4;



## 第三节 布尔表达式的翻译

### 五、控制语句中布尔式的翻译

#### 5、文法G(B)各产生式的语义子程序

$$(8) E \rightarrow E^0 E^{(2)}$$

$$- \quad \{E \cdot FC = E^{(2)} \cdot FC;$$

$$- \quad E \cdot TC = \text{MERG}(E^0 \cdot TC, E^{(2)} \cdot TC)\}$$

- 注：由于E的真假决定于 $E^{(2)}$ ，所以 $E^{(2)}$ 为假时它们的转移相同；由于 $E^{(1)}$ 或 $E^{(2)}$ 为真时都导致E为真，所以它们为真的情况要合并起来。

- 例如：将控制语句中的  $A \quad B < D$  翻译成四元式
- 解：(1)(jnz,A,\_,0) ;真出口
- (2)(j,\_,\_,(3)) ;若A为假，看 右边的表达式值
- (3)(j<,B,D,1) ;真出口; 右边的表达式值为真
- (4)(j,\_,\_,0) ;假出口; 右边的表达式值为假
- E•TC: 3、 1;
- E•FC:4;

- FUNCTION *MERG*(p1,p2);
- { IF p2=0
- MERG=p1
- ELSE
- {p=p2;
- WHILE 四元式p的第4段内容不为0
- p=四元式p的第4段内容;
- 把p1填进四元式p的第四段 ;
- MERG=p2;}}
- 注：算法思想：找到p2的链尾，然后将p1连接到p2的链尾。

## 第三节 布尔表达式的翻译

- (3)  $E \rightarrow (E^{(1)})$ 
  - $\{E \bullet TC = E^{(1)} \bullet TC; E \bullet FC = E^{(1)} \bullet FC\}$
- (4)  $E \rightarrow E^{(1)}$ 
  - $\{E \bullet TC = E^{(1)} \bullet FC; E \bullet FC = E^{(1)} \bullet TC\}$
- (5)  $E^A \rightarrow E^{(1)}$ 
  - $\{\text{BACKPATCH}(E^{(1)} \bullet TC, NXQ);$
  - $E^A \bullet FC = E^{(1)} \bullet FC;\}$
- (6)  $E \rightarrow E^A E^{(2)}$ 
  - $\{E \bullet TC = E^{(2)} \bullet TC;$
  - $E \bullet FC = \text{MERG}(E^A \bullet FC, E^{(2)} \bullet FC)\}$

# 第三节 布尔表达式的翻译

## 五、控制语句中布尔式的翻译

- 例如：将布尔式  $A \wedge B \vee C$  在语法制导下翻译成四元式。

INPUT	SYM	TC	FC	四元式
A   B   C#	#	-	-	
B   C#	#i	- -	- -	
B   C#	#E	- 1	- 2	1.(jnz,a, -,0) (3)
B   C#	#E	- 1 -	- 2 -	2.(j, -, -, 0) (5)
B   C#	# E^A	- -	- 2	
C#	# E^Ai	- - -	- 2 -	

INPUT	SYM	TC	FC	四元式
C #	# E <sup>A</sup> E	- - 3	- 2 4	3.(jnz,B, - ,0)
C #	#E	- 3	- 4	4.(j, - , - ,2) (5)
C #	#E	- 3 -	- 4 -	
C #	# E <sup>0</sup>	- 3	- -	
C #	# E <sup>0</sup>	- 3 -	- - -	
#	# E <sup>0</sup> i	- 3 - -	- - - -	
#	# E <sup>0</sup> E	- 3 - 5	- - - 6	5.(jnz,C, - ,0)
#	# E <sup>0</sup> E	- 3 6	- - 5	6.(j, - , - ,0)
#	#E	- 6	- 5	6.(j, - , - ,3)
成功				

## 第四节 控制语句的翻译

### 一、控制语句的种类：

- 无条件转移语句
- 条件语句
- 迭代语句
- 循环语句
- 分情语句

# 第四节 控制语句的翻译

## 二、标号和转移语句(GOTO)

### 1、标号

- (1) 标号语句
  - GOTO语句实现无条件转移，要确定转移的目的语句，需要给语句加标号。
  - 带标号的语句形式：L:S
    - L是标号
- (2) 标号的定义和使用：
  - 先定义后使用 L:S ... GOTO L
  - 先使用后定义 GOTO L ... L:S



## 第四节 控制语句的翻译

### 2、先定义后使用

- (1)形式：L:S

- .....
  - GOTO L
  - .....

- (2)定义和使用标号的文法：

- $S \rightarrow \text{goto } L$
  - $\text{Label} \rightarrow i:$

- (3)翻译过程：

- 当遇到定义性的标号语句时，将L归约为Label，再将L填入符号表，如下：

NAME	INFORMATION			
	CAT	...	定义否	地址
...		...		
L	标号		已	S.QUAD
		...		

S.QUAD :即S对应的入口四元式序号。

当后面遇到GOTO L语句时，便产生四元式：  
 $(j, -, -, p)$ ，其中  $p = S.QUAD$

## 第四节 控制语句的翻译

### 2、先使用后定义

(1)形式： GOTO L`

```
—          .....  
—          GOTO L`  
—          .....  
—          GOTO L`  
—          .....  
—          L`:S
```

• (2)翻译过程：

- 由于这里是先使用，所以当遇到标号L`时，它还未定义，故而填入符号表时情况与上不同，表如下：

NAME	INFORMATION			
	CAT	...	定义否	地址
...		...		
L'	标号		未	p
		...		

(2)翻译过程：

(a)填符号表：将定义否栏填“未”，地址栏暂填即将生成的四元式序号p，CAT栏填“标号”；

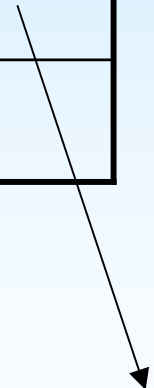
(b)生成四元式：(p) (j, -, -, 0)，等待回填；

NAME	INFORMATION			
	CAT	...	定义否	地址
...		...		
L'	标号		未	q
		...		

(c)当遇到第2个使用性标号L'时，修改符号表，仅将L'这一行的地址栏内容修改为即将生成的四元式序号q；  
 (d)生成四元式：(q) (j, -, -, p)，其中第四段的p取自符号表的地址栏在修改前的内容，即形成一个需要回填的链。不断重复(c)(d)。

NAME	INFORMATION			
	CAT	...	定义否	地址
...		...		
L'	标号		未	r
		...		

(p) (j, -, -, 0)  
 ↑  
 (q) (j, -, -, p)  
 ↑  
 (r) (j, -, -, q)



# 第四节 控制语句的翻译

## 二、标号和转移语句(GOTO)

- 2、先使用后定义
- (2)翻译过程：
  - e)当定义性语句L':S出现时，用S语句所对应的第一个四元式序号回填这个链，即使用BACKPATCH 过程。
  - 注：此链表的链首在符号表的相应行的地址栏内。
- (3)形式化的语义子程序：

– 产生式	语义子程序
– $S \rightarrow \text{goto } L$	<u>程序1</u>
– $\text{Label} \rightarrow i:$	<u>程序2</u>

- {查符号表:
- 若L不在表中，在表中建L这行，CAT栏置标号，定义否填‘未’，地址为NXQ；GEN(j, -, -, 0);
- 若L在表中，但未定义，则：p=L.地址; L.地址为NXQ；GEN(j, -, -, p);
- 若L在表中，已定义，则：p=L.地址; GEN(j, -, -, p);
- }



- {查符号表:
- 若L不在表中，在表中建L这行，CAT栏置标号，  
定义否填‘已’，地址为NXQ；
- 若L已在表中，但定义否已填‘已’或CAT<>标号，  
则出错；
- 否则，{将定义否改为‘已’；
- q=L.地址；
- BACKPATCH(q,NXQ);
- L.地址=NXQ;}
- }

# 第四节 控制语句的翻译

## 三、IF语句的翻译

### 1、描述IF语句的文法

- $S \rightarrow \text{if } E \text{ then } S^{(1)}$
- $S \rightarrow \text{if } E \text{ then } S^{(1)} \text{ else } S^{(2)}$

### 2、IF语句的翻译

- (1)完成对布尔式E的翻译，获得一组四元式，并留下两个待填的语义值 $E \cdot TC, E \cdot FC$ ；
- (2)扫描完then之后就得到了E的真出口，用 $\text{BACKPATCH}(E \cdot TC, NXQ)$ 过程回填；
- (3)翻译 $S^{(1)}$ ，它可能会是任何一种语句，总可以递归地调用语句翻译过程来完成，得到一组四元式序列；

## 第四节 控制语句的翻译

### 三、IF语句的翻译

#### 2、IF语句的翻译

- (4)若遇到else，表示 $S^{(1)}$ 已翻译完，应无条件生成一条GOTO四元式( $j, -, -, 0$ )转到 $S^{(2)}$ 语句后面，以示这个IF语句已执行结束。但这个无条件转移语句究竟转到什么地方去，是不确定的，甚至在翻译完 $S^{(2)}$ 语句之后也不一定确定下来。遇到else还表示已获得E的假出口。
- 例如：语句
  - if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$  else  $S_3$
  - 注：由于转移指令的转移目标只能等到出口明朗了才能回填，此时应把该待填的四元式序号并链后存在与代表整个语句的非终结符S相联系的语义栈S•CHAIN中。

## 第四节 控制语句的翻译

### 三、IF语句的翻译

#### 2、IF语句的翻译

- (4')若不遇到else，那么布尔式的假出口与 $S^{(1)}$ 的结束出口都表示IF语句的结束。
- (5)翻译 $S^{(2)}$ 语句成四元式序列。它执行过后也表示IF语句应该结束了，它的结束出口和 $S^{(1)}$ 是一样的，可将它们出口并链，链首置于 $S \cdot CHAIN$ 中。
- 故，条件语句翻译之后，除了生成相应 $E, S^{(1)}, S^{(2)}$ 的四元式之外，还剩下一个待填语句链，链首在 $S \cdot CHAIN$ 中，出口明确后按此回填。

## 第四节 控制语句的翻译

### 三、IF语句的翻译

#### 3、改写产生式

- $S \rightarrow \text{if } E \text{ then } S^{(1)} \text{ else } S^{(2)}$  改写为：
  - $C \rightarrow \text{if } E \text{ then}$
  - $T \rightarrow C S^{(1)} \text{ else}$
  - $S \rightarrow T S^{(2)}$
- $S \rightarrow \text{if } E \text{ then } S^{(1)}$  改写为：
  - $C \rightarrow \text{if } E \text{ then}$
  - $S \rightarrow C S^{(1)}$

## 第四节 控制语句的翻译

### 三、IF语句的翻译

#### 4、语义子程序

- $C \rightarrow \text{if } E \text{ then}$       $\{\text{BACKPATCH}(E \cdot \text{TC}, \text{NXQ});$   
    —                      $C \cdot \text{CHAIN} = E \cdot \text{FC};\}$
- $T \rightarrow C S^{(1)} \text{ else}$     $\{q = \text{NXQ}; \text{GEN}(j, -, - 0);$   
    —                      $\text{BACKPATCH}(C \cdot \text{CHAIN}, \text{NXQ});$   
    —                      $T \cdot \text{CHAIN} = \text{MERG}(S^{(1)} \cdot \text{CHAIN}, q)\}$
- $S \rightarrow T S^{(2)}$           $\{S \cdot \text{CHAIN} = \text{MERG}(T \cdot \text{CHAIN}, S^{(2)} \cdot \text{CHAIN})\}$
- $S \rightarrow C S^{(1)}$           $\{S \cdot \text{CHAIN} = \text{MERG}(C \cdot \text{CHAIN}, S^{(1)} \cdot \text{CHAIN})\}$

## 第四节 控制语句的翻译

- 三、IF语句的翻译

- 4、语义子程序

- 注：1)这里的基本思想是拉链回填；

2)当if E then归约为C时，布尔式E已不在栈内，故其语义值E.FC也不复存在，但E.FC尚未回填，可将它暂存于非终结符C的CHAIN中，通过后来的拉链回填C.CHAIN，实际上就是对E.FC的回填；

3)语句翻译完了，但S•CHAIN还没回填，只有当整个嵌套句全部翻译完，才能进行回填。

## 第四节 控制语句的翻译

- 例如:翻译条件嵌套语句：
- If a then (1) (jnz,a,\_,0)
- if b then (2) (j,\_,\_,0)
- A:=2
- else A:=3
- Else if c then
- A=4
- Else A=5
- 翻译成四元式为：



## 第四节 控制语句的翻译

- 例如:翻译条件嵌套语句：

- If a **then**

- if b then

- A:=2

- else A:=3

- Else if c then

- A=4

- Else a=5

- 翻译成四元式为：

(1)(jnz,a,\_,3)

(2)(j,\_,\_,0)

(3)(jnz,b,\_,0)

(4)(j,\_,\_,0)

C•CHAIN->2

## 第四节 控制语句的翻译

- 例如:翻译条件嵌套语句：

- If a then
- if b then
- A:=2
- else A:=3
- Else if c then
- A=4
- Else a=5
- 翻译成四元式为：

(1)(jnz,a,\_,3)

(2)(j,\_,\_,0)

(3)(jnz,b,\_,5)

(4)(j,\_,\_,0)

(5)(:=,2,\_,A)

Ca•CHAIN->2

Cb•CHAIN->4

## 第四节 控制语句的翻译

- 例如:翻译条件嵌套语句：

- If a then
- if b then
- A:=2
- else A:=3
- Else if c then
- A=4
- Else a=5
- 翻译成四元式为：

(1)(jnz,a,\_,3)

(2)(j,\_,\_,0)

(3)(jnz,b,\_,5)

(4)(j,\_,\_,7)

(5)(:=,2,\_,A)

(6)(j,\_,\_,0)

Ca•CHAIN->2

Tb•CHAIN->6

## 第四节 控制语句的翻译

- 例如:翻译条件嵌套语句：

- If a then
- if b then
- A:=2
- else A:=3
- Else if c then
- A=4
- Else a=5

- 翻译成四元式为：

Sb•CHAIN->8,6

(1)(jnz,a,\_,3)

(2)(j,\_,\_,0)

(3)(jnz,b,\_,5)

(4)(j,\_,\_,7)

(5)(:=,2,\_,A)

(6)(j,\_,\_,0)

(7)(:=,3,\_,A)

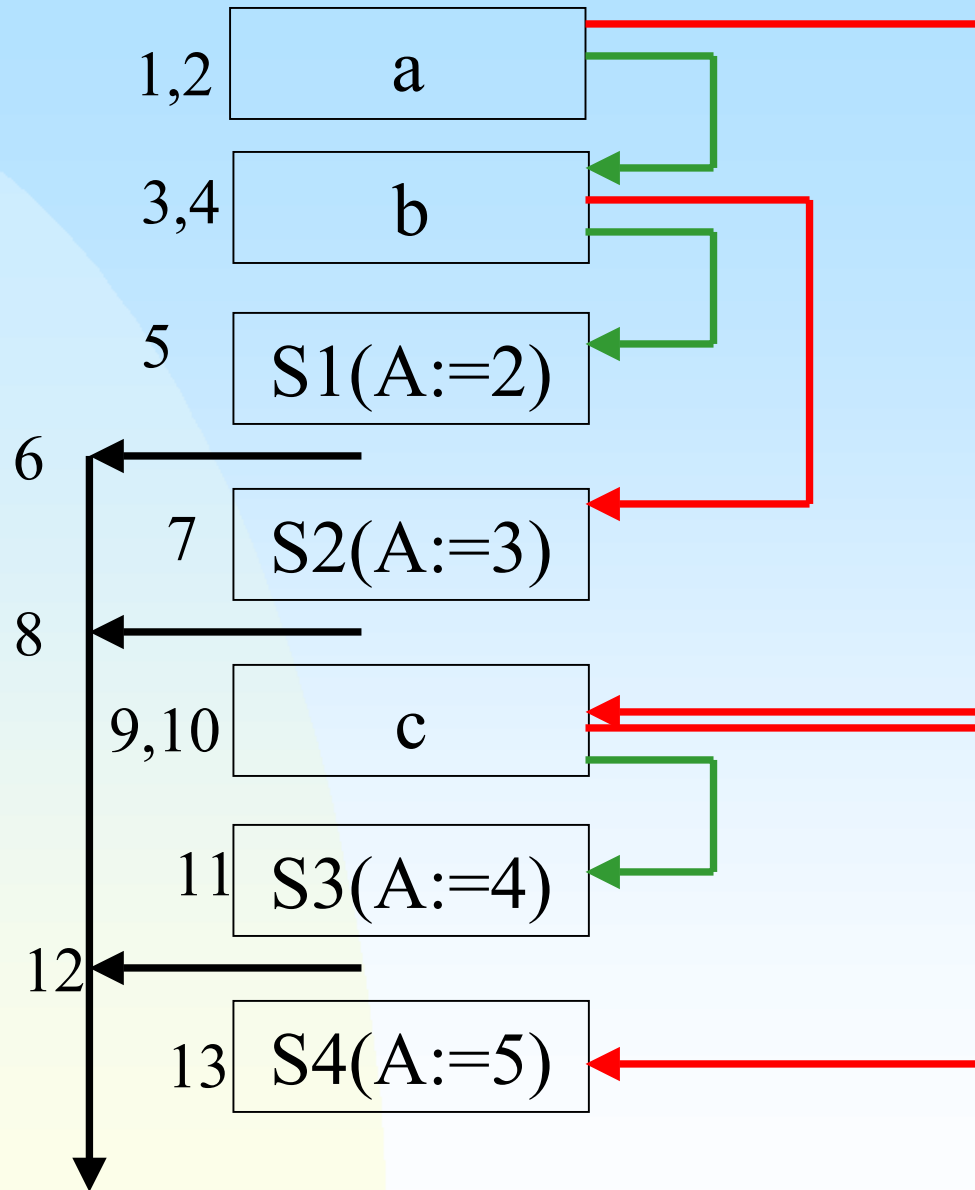
(8)(j,\_,\_,0)

Ca•CHAIN->2

## 第四节 控制语句的翻译

- 可得四元式：

- (1)(jnz,a,\_,3)      (8)(j,\_,\_,6)
- (2)(j,\_,\_,9)      (9)(jnz,c,\_,11)
- (3)(jnz,b,\_,5)      (10)(j,\_,\_,13)
- (4)(j,\_,\_,7)      (11)(:=,4,\_,A)
- (5)(:=,2,\_,A)      (12)(j,\_,\_,8)
- (6)(j,\_,\_,0)      (13)(:=,5,\_,A)
- (7)(:=,3,\_,A)



# 第四节 控制语句的翻译

## 四、WHILE语句的翻译

### 1、WHILE语句的文法

- $S \rightarrow \text{while } E \text{ do } S^{(1)}$

### 2、WHILE语句的翻译思想

- (1)翻译E代码段，并留两个待填的E•TC, E•FC链；
- (2)扫描过do之后，就可以回填E•TC；
- (3)翻译 $S^{(1)}$ 成代码段， $S^{(1)}$ 翻译之后，应无条件转到E代码段的第一条四元式。
- 注：1) 因此，开始翻译while语句时，应留下第一条四元式序号，以备 $S^{(1)}$ 翻译之后用。
- 2) E为假时需回填，此时，E.FC作为S.CHAIN一部分，以便回填。

## 第四节 控制语句的翻译

### 四、WHILE语句的翻译

#### 3、改写产生式

- $S \rightarrow \text{while } E \text{ do } S^{(1)}$  改写为：
  - $W \rightarrow \text{while}$
  - $W^d \rightarrow WE \text{ do}$
  - $S \rightarrow W^d S^{(1)}$



## 第四节 控制语句的翻译

### 四、WHILE语句的翻译

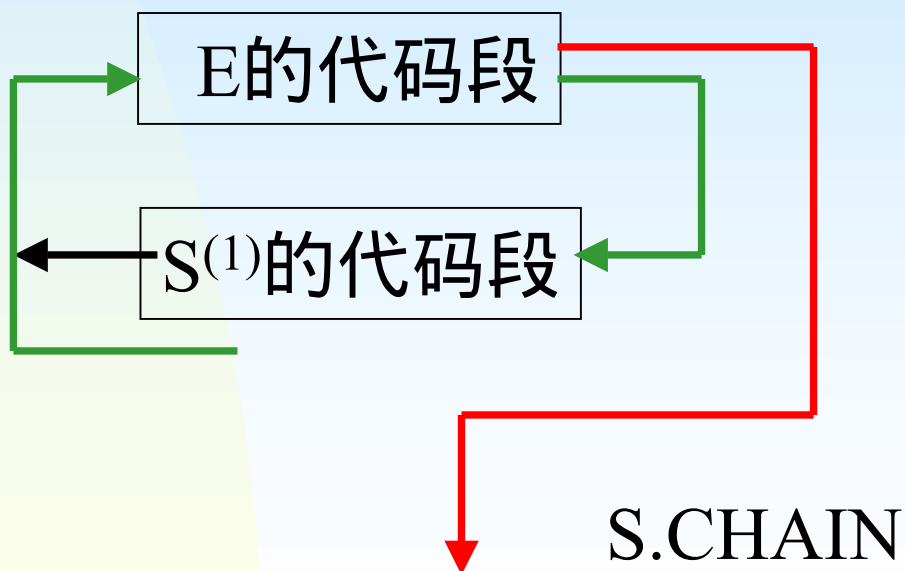
#### 4、语义子程序

- $W \rightarrow \text{while} \quad \{W \cdot \text{QUAD} = \text{NXQ}\}$
- $W^d \rightarrow WE \text{ do} \quad \{\text{BACKPATCH}(E \cdot \text{TC}, \text{NXQ});$ 
  - $W^d \cdot \text{CHAIN} = E \cdot \text{FC};$
  - $W^d \cdot \text{QUAD} = W \cdot \text{QUAD};\}$
- $S \rightarrow W^d S^{(1)} \quad \{\text{BACKPATCH}(S^{(1)} \cdot \text{CHAIN}, W^d \cdot \text{QUAD});$ 
  - $\text{GEN}(j, \_, \_, W^d \cdot \text{QUAD});$
  - $S \cdot \text{CHAIN} = W^d \cdot \text{CHAIN}\}$

## 第四节 控制语句的翻译

### 四、WHILE语句的翻译

### 5、翻译成四元式的结构



## 第四节 控制语句的翻译

- 例如：翻译
- While ( $A < B$ ) do
- if ( $C < D$ ) then
- $X := Y + Z;$
- 翻译为：(100) ( $j <, A, B, 0$ )
- (101)( $j, \_, \_, 0$ )

## 第四节 控制语句的翻译

- 例如：翻译
- While ( $A < B$ ) do
- if ( $C < D$ ) then
- $X := Y + Z;$
- 翻译为：(100) ( $j <, A, B, (102)$ )
- (101) ( $j, \_, \_, 0$ )
- (102) ( $j <, C, D, 0$ )
- (103) ( $j, \_, \_, (100)$ )

## 第四节 控制语句的翻译

- 例如：翻译
- While ( $A < B$ ) do
- if ( $C < D$ ) then
- $X := Y + Z$ ;
- 翻译为：(100) ( $j <, A, B, (102)$ )
- (101) ( $j, \_, \_, 0$ )
- (102) ( $j <, C, D, (104)$ )
- (103) ( $j, \_, \_, (100)$ )
- (104) ( $+, Y, Z, T_1$ )
- (105) ( $:=, T_1, \_, X$ )

## 第四节 控制语句的翻译

- 例如：翻译
- While (A<B) do
- if (C<D) then
- X:=Y+Z;
- 翻译为：(100) (j<,A,B,(102)) (106)(j,\_,\_(100))
- (101)(j,\_,\_(107))
- (102)(j<,C,D,(104))
- (103)(j,\_,\_(100))
- (104)(+,Y,Z,T<sub>1</sub>)
- (105)(:=, T<sub>1</sub>,\_,X)

## 第四节 控制语句的翻译

### 五、REPEAT语句的翻译

#### 1、文法

- $S \rightarrow \text{repeat } S^{(1)} \text{ until } E$

#### 2、翻译思想

- (1)翻译 $S^{(1)}$ 的代码段，留下待回填的语句链 $S^{(1)} \cdot \text{CHAIN}$ ;
- (2)扫描过until之后，就可以回填 $S^{(1)} \cdot \text{CHAIN}$ ;
- (3)翻译E代码段，并留两个待填的 $E \cdot \text{TC}$ ,  $E \cdot \text{FC}$ 链。
- 注： $E \cdot \text{FC}$ 应无条件回到本REPEAT语句的第一条四元式，所以进入REPEAT时应将它的四元式序号留下。  
 $E \cdot \text{TC}$ 是整个语句结束，留在 $S \cdot \text{CHAIN}$ 中等待回填。

## 第四节 控制语句的翻译

### 五、REPEAT语句的翻译

#### 3、改写产生式

- $S \rightarrow \text{repeat } S^{(1)} \text{ until } E$  改写为：
  - $R \rightarrow \text{repeat}$
  - $U \rightarrow RS^{(1)} \text{ until}$
  - $S \rightarrow UE$



## 第四节 控制语句的翻译

### 五、REPEAT语句的翻译

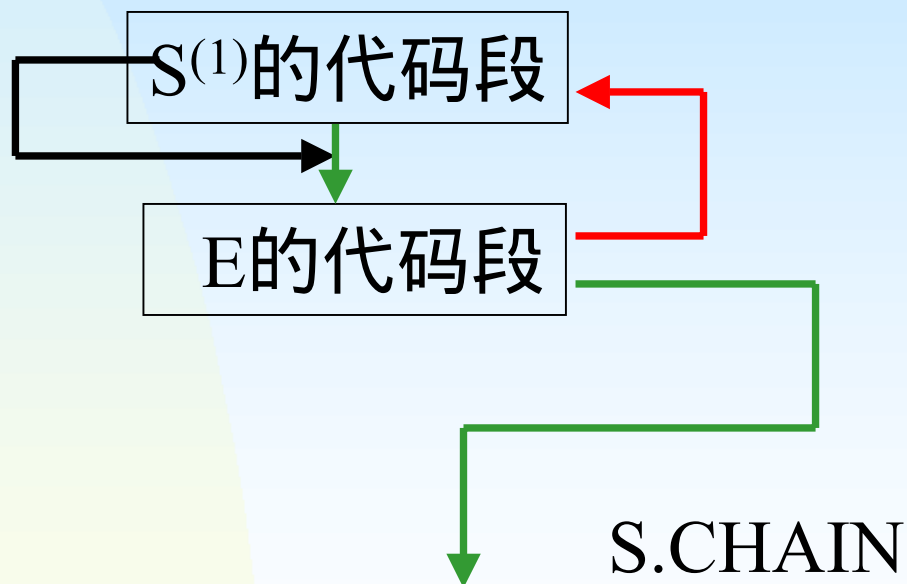
#### 4、语义子程序

- $R \rightarrow \text{repeat}$   $\{R \cdot \text{QUAD} = \text{NXQ}\}$
- $U \rightarrow RS^{(1)} \text{ until}$   $\{U \cdot \text{QUAD} = R \cdot \text{QUAD};$   
—  $\text{BACKPATCH}(S^{(1)} \cdot \text{CHAIN}, \text{NXQ})\}$
- $S \rightarrow UE$   $\{\text{BACKPATCH}(E \cdot \text{FC}, U \cdot \text{QUAD})$   
—  $S \cdot \text{CHAIN} = E \cdot \text{TC}\}$

## 第四节 控制语句的翻译

### 五、REPEAT语句的翻译

#### 5、翻译成四元式的结构



## 第四节 控制语句的翻译

- 例如：将下面语句翻译成四元式
  - repeat
  - $X := X + 1$
  - until  $X > Y$ ;
- 翻译:  $(m) (+, X, 1, T_1)$ 
  - $(m+1)(:=, T_1, \_, X)$
  - $(m+2)(j>, X, Y, (m+4))$
  - $(m+3)(j, \_, \_, (m))$

## 第四节 控制语句的翻译

### 六、循环FOR语句的翻译

#### 1、文法

- $S \rightarrow \text{for } i := E^{(1)} \text{ step } E^{(2)} \text{ until } E^{(3)} \text{ do } S^{(1)}$

#### 2、翻译思想

- 1) 按PL/1将FOR 语句翻译成：

- $i = E^{(1)};$   $\{S^{(1)};$
- $INCR = E^{(2)};$   $\text{goto AGAIN};\}$
- $LIMIT = E^{(3)};$
- $\text{Goto OVER};$
- $\text{AGAIN: } i = i + INCR$
- $\text{OVER: } \text{if } i \leq LIMIT \text{ then}$

## 第四节 控制语句的翻译

### 六、循环FOR语句的翻译

#### 2、翻译思想

- 注：由于 $E^{(2)}$ 、 $E^{(3)}$ 在循环期间一般是不会改变的，即使有改变也以改变前为准，所以可提前翻译，只把结果拿来用。
- 按这种解释改写文法：
  - (1)  $F \rightarrow \text{for } i := E^{(1)} \text{ step } E^{(2)} \text{ until } E^{(3)}$
  - (2)  $S \rightarrow F \text{ do } S^{(1)}$
- 把这种解释改写为FOR语句文法的语义子程序为：

- $F \rightarrow \text{for } i := E^{(1)} \text{ step } E^{(2)} \text{ until } E^{(3)}$  的语义子程序
  - $\{F \cdot \text{PLACE} = \text{ENTRY}(i);$
  - $\text{GEN}(=, E^{(1)} \cdot \text{PLACE}, \_, F \cdot \text{PLACE});$
  - $q = \text{NXQ};$
  - $F \cdot \text{QUAD} = q + 1;$
  - $\text{GEN}(j, \_, \_, q + 2);$
  - $\text{GEN}(+, F \cdot \text{PLACE}, E^{(2)} \cdot \text{PLACE}, F \cdot \text{PLACE});$
  - $\text{GEN}(j \leq, F \cdot \text{PLACE}, E^{(3)} \cdot \text{PLACE}, q + 4);$
  - $F \cdot \text{CHAIN} = \text{NXQ};$
  - $\text{GEN}(j, \_, \_, 0); \}$

- $S \rightarrow F \text{ do } S^{(1)}$ 的语义子程序
- $\{\text{BACKPATCH}(S^{(1)} \cdot \text{CHAIN}, F \cdot \text{QUAD});$
- $\text{GEN}(j, \_, \_, F \cdot \text{QUAD});$
- $S \cdot \text{CHAIN} = F \cdot \text{CHAIN};\}$

# 第四节 控制语句的翻译

## 六、循环FOR语句的翻译

### 2、翻译思想

- 2) 若FOR语句至少做一次循环体，则FOR语句解释成

- $i = E^{(1)};$
- $INCR = E^{(2)};$
- $LIMIT = E^{(3)};$
- AGAIN:  $S^{(1)};$
- $i = i + INCR$
- if  $i \leq LIMIT$  then
- goto AGAIN;



## 第四节 控制语句的翻译

- 把这种解释改写为FOR语句文法的语义子程序为：
- (1)  $F \rightarrow \text{for } i := E^{(1)} \text{ step } E^{(2)} \text{ until } E^{(3)}$  的语义子程序
  - $\{F \cdot \text{PLACE} = \text{ENTRY}(i);$
  - $\text{GEN}(=, E^{(1)} \cdot \text{PLACE}, \_, F \cdot \text{PLACE});$
  - $\text{INCR} = E^{(2)} \cdot \text{PLACE};$
  - $\text{LIMIT} = E^{(3)} \cdot \text{PLACE};$
  - $F \cdot \text{QUAD} = \text{NXQ}; \}$

- (2)  $S \rightarrow F \text{ do } S^{(1)}$ 的语义子程序
  - $\{\text{BACKPATCH}(S^{(1)} \cdot \text{CHAIN}, \text{NXQ});$
  - $\text{GEN}(+, F \cdot \text{PLACE}, \text{INCR}, F \cdot \text{PLACE});$
  - $\text{GEN}(j \leq, F \cdot \text{PLACE}, \text{LIMIT}, F \cdot \text{QUAD});$
  - $S \cdot \text{CHAIN} = 0;\}$
- 注：这种翻译不留语句链，因为该语句结束后自动转后继语句执行。这样目标代码质量较高，但它要求循环体至少做一次，有时不满足用户编程要求。

# 第四节 控制语句的翻译

## 六、循环FOR语句的翻译

### 2、翻译思想

- 3 ) FOR 语句一种较优的解释：
  - $i = E^{(1)};$
  - $INCR = E^{(2)};$
  - $LIMIT = E^{(3)};$
  - AGAIN: if  $i > LIMIT$  goto NEXT;
  - {  $S^{(1)};$
  - $i = i + INCR$
  - goto AGAIN;}
  - NEXT:.....

- 例如：翻译
- for i:=1 step 2 until 2\*X do A:=A+2
  - 按第3种语义子程序翻译为：
  - 1.(\*,2,X,T<sub>1</sub>)
  - 2.(:=,1,\_,i)
  - 3.(j>,i, T<sub>1</sub>,0)
  - 4.(+,A,2, T<sub>2</sub>)
  - 5.(:=, T<sub>2</sub>,\_,A)
  - 6.(+,i,2,i)
  - 7.(j,\_,\_,3)

## 第四节 控制语句的翻译

### 七、复合语句的翻译

#### 1、文法

- $S \rightarrow \text{begin } L \text{ end}$
- $L \rightarrow S$
- $L \rightarrow L^S S$
- $L^S \rightarrow L;$

## 第四节 控制语句的翻译

### 七、复合语句的翻译

#### 2、语义子程序

- (1)  $L \rightarrow S$                        $\{L \cdot \text{CHAIN} = S \cdot \text{CHAIN};\}$
- (2)  $L^S \rightarrow L;$                      $\{\text{BACKPATCH}(L \cdot \text{CHAIN}, \text{NXQ});\}$
- (3)  $L \rightarrow L^S S$                     $\{L \cdot \text{CHAIN} = S \cdot \text{CHAIN};\}$
- (4)  $S \rightarrow \text{begin } L \text{ end}$ 
  - $\{\text{BACKPATCH}(L \cdot \text{CHAIN}, \text{NXQ});$
  - $S \cdot \text{CHAIN} = 0;\}$

# 第五节 数组元素及其在赋值语句中的翻译

## 一、数组及其下标变量地址的计算

### 1、数组分类：

- 一维数组、二维数组、多维数组。
- 确定数组、可变数组：
  - 根据数组所需存储空间是否在编译时就已知道。

### 2、多维数组的存放

- 按行存放
- 按列存放(FORTRAN)
- 只要知道数组的首地址，及每个数组元素占多少内存单元，就可计算出任一数组元素的地址。

- 例如：n维数组定义为：
  - $\text{Array}[l_1:u_1, l_2:u_2, \dots, l_n:u_n]$
  - 令  $d_i = u_i - l_i + 1, i=1, 2, \dots, n$ , 称为每一维尺寸
  - $D = a + ((i_1 - l_1)d_2d_3\dots d_n + (i_2 - l_2)d_3d_4\dots d_n + (i_{n-1} - l_{n-1})d_n + (i_n - l_n))m$
- 改写为  $D = \text{CONSPART} + \text{VARPART}$ 
  - $\text{CONSPART} = a - C$ 
    - $C = ((\dots(l_1d_2 + l_2)d_3 + l_3)d_3\dots + l_{n-1})d_n + l_n)m$
  - $\text{VARPART} = ((\dots(i_1d_2 + i_2)d_3 + i_3)d_3\dots + i_{n-1})d_n + i_n)m$



- 注：CONSPART称作不变部分，它和下标 $i_1$ 、 $i_2 \dots i_n$ 无关,只和各维尺寸和下界有关；只需计算一次，其中C在编译时就可算出。
- VARPART称作可变部分，它和下标 $i_1$ 、 $i_2 \dots i_n$ 有关,它必须根据下标值，转换成相应代码，待运行时计算出。

# 第五节 数组元素及其在赋值语句中的翻译

## 一、数组及其下标变量地址的计算

### 3、编译程序对数组说明的处理

- 当遇到数组说明时，必须把数组的有关信息记录在一张“内情向量”表中，以便以后计算数组下标变量地址时引用这些信息。
- 内情向量表的结构：
  - 维数、各维上下界、首地址、类型等。

$l_1$	$u_1$	$d_1$
$l_2$	$u_2$	$d_2$
...	...	...
$l_n$	$u_n$	$d_n$
$n$	$C$	
type	$a$	

# 第五节 数组元素及其在赋值语句中的翻译

## 一、数组及其下标变量地址的计算

### 4、编译程序对数组说明的处理

- 1) 对于确定的数组来说：内情向量可登记在编译时的符号表中；
- 2) 对于可变数组来说：
  - (1) 内情向量在编译时无法知道，只有在运行时才能算出来；因此，编译程序必须为可变数组设置一个存储空间，以便运行时建立相应的内情向量表。
  - (2) 这样，就必须在编译时为运行程序准备好调用的运行子程序，执行到数组A所在的分程序时，就把内情向量的各有关参数填进表内，然后动态申请数组所需空间。（[算法](#)）

注：不论对哪种数组，其地址计算公式都一样，算法也一样。

# 第五节 数组元素及其在赋值语句中的翻译

## 一、数组及其下标变量地址的计算

### 5、数组的内情向量表建立算法

P153(按行优先存放)

```
{ i=1;N=1;C=0;
  while i<=n do {
    di=ui-li+1; N=N*di; C=C*di+li;
    将li,ui,di填进内情向量表中;i=i+1}
  N=N*m; /*每元素占m字节*/
  C=C*m;
  申请N个单元的数组空间，令这片空间的首地址为a;
  将N和C，以及TYPE和a填进这内情向量表中
}
```

# 第五节 数组元素及其在赋值语句中的翻译

## 二、确定数组的数组元素引用的中间代码形式

### 1、访问数组元素可设想为：

- 把它的VARPART计算在某一变址单元T中，用CONSPART作为“基址”，然后以变址方式访问存储单元—— $\text{CONSPART}[T]$ ，静态数组 $\text{CONSPART} = a - C$ ，C可以从内情向量表中查到，而a在运行时一定可确定下来，所以可以生成a-C代码，待运行时再行确定。假定 $T_1$ 是用于存放CONSPART的临时单元， $T_1 = a - C$ 那么用 $T_1[T]$ 表示数组元素的地址。

# 第五节 数组元素及其在赋值语句中的翻译

## 二、确定数组的数组元素引用的中间代码形式

### 2、访问数组元素

- 引用数组元素：
  - $(=[ ], T_1[T], \_, X)$
- 注：这是一个变址取数四元式，含义相当于： $X = T_1[T]$ .
- 对数组元素赋值：
  - $([ ] =, X, \_, T_1[T])$
- 注：这是一个变址存数四元式，含义相当于： $T_1[T] = X$ .

# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 1、文法

- $A \rightarrow V := E$
- $V \rightarrow i[Elist] \mid i$
- $Elist \rightarrow Elist, E \mid E$
- $E \rightarrow E \text{ op } E \mid (E) \mid V$  /\*op表示各种算数运算\*/

- 注：V可以是简单变量也可以是下标变量。

这个递归定义可形成数组嵌套数组结构。

# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 2、文法改写

- $A \rightarrow V := E$
  - $V \rightarrow \text{Elist}] \mid i$
  - $\text{Elist} \rightarrow \text{Elist}^{(1)}, E \mid i[ E$
  - $E \rightarrow E \text{ op } E \mid (E) \mid V$
- 注：把数组名*i*和最左下标表达式写在一起就表示为数组*i*开始计算第一个下标，同时使我们在整个下标串Elist的翻译过程中随时都能知道数组*i*的符号表入口地址，及表中相应信息。



# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 3、相关语义变量和函数过程

- 语义变量：

- ARRAY：数组名在符号表的入口地址
- DIM：数组下标维数计数器
- PLACE：语义变量，存符号表入口地址或临时变量的序号
- OFFSET：
  - 简单变量：OFFSET=null
  - 下标变量：OFFSET保存已计算的VARPART

# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 3、相关语义变量和函数过程

- 函数过程：

- $\text{LIMIT}[\text{ARRAY}, k]$ ：通过符号表查内情向量表，返回第 $k$ 维的尺寸 $d_k$
- $\text{HEAD}[\text{ARRAY}]$ ：取得数组首地址 $a$ ,或查内情向量表或等到运行时分配得到
- $\text{CONS}[\text{ARRAY}]$ ：查内情向量表，得 $C$ 值
- $\text{TYPE}[\text{ARRAY}]$ ：查内情向量表 $\text{TYPE}$ 项，返回一个数组元素占有的字单元数 $m$ 值。

# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 4、语义子程序

- (1)  $A \rightarrow V := E$
- {IF ( $V \cdot \text{OFFSET} = \text{null}$ )
- $\text{GEN}(=, E \cdot \text{PLACE}, \_, V \cdot \text{PLACE});$
- ELSE
- $\text{GEN}([ ] =, E \cdot \text{PLACE}, \_, V \cdot \text{PLACE}[V \cdot \text{OFFSET}])$ }

# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 4、语义子程序

(2)  $E \rightarrow E^{(1)} \text{ op } E^{(2)}$      $\{T = \text{NEWTEMP};$   
—                     $\text{GEN}(\text{op}, E^{(1)} \cdot \text{PLACE}, E^{(2)} \cdot \text{PLACE}, T);$   
—                     $E \cdot \text{PLACE} = T\}$

(3)  $E \rightarrow (E^{(1)})$      $\{E \cdot \text{PLACE} = E^{(1)} \cdot \text{PLACE}\}$

(4)  $E \rightarrow V$      $\{\text{IF } (V \cdot \text{OFFSET} = \text{null})$   
—                     $E \cdot \text{PLACE} = V \cdot \text{PLACE};$   
—                     $\text{ELSE}$   
—                     $\{T = \text{NEWTEMP};$   
—                     $\text{GEN}(=[ ], E \cdot \text{PLACE}[V \cdot \text{OFFSET}], \_, T);$   
—                     $E \cdot \text{PLACE} = T;\}\}$

## 第五节 数组元素及其在赋值语句中的翻译

### 三、按行存放的赋值语句中的数组元素的翻译

#### 4、语义子程序

(5)  $V \rightarrow \text{Elist}$      $\{ \text{IF (TYPE[ARRAY]} \neq 1)$   
—                     $\{ \text{T=NEWTEMP};$   
—                     $\text{GEN}(*, \text{Elist} \bullet \text{PLACE}, \text{TYPE[ARRAY]}, \text{T});$   
—                     $\text{Elist} \bullet \text{PLACE} = \text{T}; \}$   
—                     $\text{V} \bullet \text{OFFSET} = \text{Elist} \bullet \text{PLACE};$   
—                     $\text{T} = \text{NEWTEMP};$   
—                     $\text{GEN}(-, \text{HEAD[ARRAY]}, \text{CONS[ARRAY]}, \text{T});$   
—                     $\text{V} \bullet \text{PLACE} = \text{T} \}$   
(6)  $V \rightarrow i$              $\{ \text{V} \bullet \text{PLACE} = \text{ENTRY}[i];$   
—                     $\text{V} \bullet \text{OFFSET} = \text{null} \}$

## 第五节 数组元素及其在赋值语句中的翻译

### 三、按行存放的赋值语句中的数组元素的翻译

#### 4、语义子程序

(7)  $Elist \rightarrow Elist^{(1)}, E$

- $\{T = NEWTEMP;$
- $k = Elist^{(1)} \bullet DIM + 1;$
- $d_k = LIMIT(Elist^{(1)} \bullet ARRAY, k);$
- $GEN(*, Elist^{(1)} \bullet PLACE, d_k, T);$
- $T_1 = NEWTEMP;$
- $GEN(+, T, E \bullet PLACE, T_1);$
- $Elist \bullet ARRAY = Elist^{(1)} \bullet ARRAY;$
- $Elist \bullet PLACE = T_1;$
- $Elist \bullet DIM = k;\}$

# 第五节 数组元素及其在赋值语句中的翻译

## 三、按行存放的赋值语句中的数组元素的翻译

### 4、语义子程序

(8)  $Elist \rightarrow i[ E$

- $\{Elist \bullet PLACE = E \bullet PLACE;$
- $Elist \bullet DIM = 1;$
- $Elist \bullet ARRAY = ENTRY(i)\}$

## 第五节 数组元素及其在赋值语句中的翻译

- 例如：数组说明为：ARRAY[1:10,1:20];  
且从内情向量表查得数组首地址为a,  
 $C=21, d_2=20, m=1$ . 为  $X:=A[I,J]$  生成四元式序列：
- 解：四元式序列为：
  - (1)  $(*, I, 20, T_1)$
  - (2)  $(+, T_1, J, T_2)$
  - (3)  $(-, a, 21, T_3)$
  - (4)  $(=[ ], T_3[T_2], \_, T_4)$
  - (5)  $(:=, T_4, \_, X)$



# 第六节 过程调用语句

## 一、过程调用

### 1、过程：

- 定义和调用过程是程序语言的主要特征之一。过程是模块化程序设计的主要手段，同时也是节省程序代码和扩充语言能力的主要途径。

### 2、过程调用：

- 实质是把程序控制转到子程序。

### 3、过程调用中遇到的问题：

- 子程序完成后如何回到主调程序；
- 子程序回到主调程序时，运行环境的恢复；
- 如何进行参数传递；

# 第六节 过程调用语句

## 二、参数传递

### 1、参数

- 实在参数
- 形式参数

### 2、传递方式

- 1)传地址(call by reference)：把实在参数的地址传递给相应的形式参数。
- 2)传值(call by value)：把实在参数的值计算出来传递给相应的形式参数。
- 3)传名(call by name)：ALGOL 60 所定义的一种特殊的形 - 实参结合方式。

# 第六节 过程调用语句

## 二、参数传递

### 3、传地址：

#### 1) 基本方式

- A)若实参是一个变量（包括下标变量），则直接传递它的地址；
- B)若实参是一个常数或其他表达式，就把值计算出来并存放在某临时单元中，然后传递这个临时单元的地址。

# 第六节 过程调用语句

## 二、参数传递

### 3、传地址：

#### 2) 传址过程

- A)当程序控制转入被调用过程之后，被调用段首先把实参地址从连接数据区中读入形参单元中。
  - 注：若已直接传至形参单元，不做此步。
- B)在过程体内，对形参的任何引用或赋值都被看作对形参单元的间接访问。这间接赋值将直接修改实参单元的内容。

## 第六节 过程调用语句

### 二、参数传递

#### 4、传值：

- 调用时把实在参数的值计算出来，把值直接传送给形式单元中。在过程体中应用这些单元不会修改实参单元的值。

#### 5、传名

- 过程调用相当于把被调用段的过程代替调用语句，并且过程体内形参名皆用相应的实参名代替。

## 第六节 过程调用语句

### 三、过程调用语句的翻译（传址）

#### 1、文法

- $S \rightarrow \text{call } i(\text{Arglist})$
- $\text{Arglist} \rightarrow \text{Arglist}, E$
- $\text{Arglist} \rightarrow E$

#### 2、相关四元式

- 1)(par,\_,\_,T)
  - 含义是：将参数T传递到一个公共的区域或直接传递到过程的形参单元
- 2)(jsr,\_,n,S)
  - 含义是：转子指令，转移到S过程的四元式入口序号。

## 第六节 过程调用语句

### 三、过程调用语句的翻译（传址）

#### 3、语义子程序

(1)  $S \rightarrow \text{call } i(\text{Arglist})$

- $\{n=0;$
- $\text{for( 队列Arglist} \cdot \text{QUEUE中的每个} p )$
- $\text{GEN}(\text{par}, \_, \_, p);$
- $n=n+1;$
- $\text{GEN}(\text{jsr}, \_, n, \text{ENTRY}(i))\}$

(2)  $\text{Arglist} \rightarrow \text{Arglist}^{(1)}, E$

- $\{\text{把} E \cdot \text{PLACE} \text{ 添加入 } \text{Arglist}^{(1)} \cdot \text{QUEUE} \text{ 末端} ;$
- $\text{Arglist} \cdot \text{QUEUE} = \text{Arglist}^{(1)} \cdot \text{QUEUE}\}$

## 第六节 过程调用语句

### 三、过程调用语句的翻译（传址）

#### 3、语义子程序

- (3)  $\text{Arglist} \rightarrow E$       {建立一个 $\text{Arglist} \cdot \text{QUEUE}$ 空队列；
- $\text{Arglist} \cdot \text{QUEUE} = \text{队列头}$ ；
- 将 $E \cdot \text{PLACE}$ 添入队列}
- 注： $\text{Arglist} \cdot \text{QUEUE}$ 存放队列头指针，它存于 $\text{QUEUE}$ 语义栈。



## 第六节 过程调用语句

- 例如：过程调用：CALL S(A+B,Z)
- 译为：
  - k ( $\_$ ,A,B,T)
  - k+1 (par, $\_$ , $\_$ ,T)
  - k+2 (par,  $\_$ , $\_$ ,Z)
  - k+3 (jsr,  $\_$ ,2,S)

## 第六节 过程调用语句

### 四、过程调用和数组元素相混淆的处理

1、数组元素和过程（或函数）调用语句在外部形式上没什么区别

– 如： $X := A(I, J)$

2、解决方法

- 1) 查符号表

- 即在把 $A(I, J)$ 中的第一个表达式 $I$ 归约为Elist或Arglist之前要查符号表，根据查到的结果决定用哪个产生式进行归约。

## 第六节 过程调用语句

### 四、过程调用和数组元素相混淆的处理

#### 2、解决方法

- 2) 词法分析解决

- 即让词法分析器在发送单词A的类号前先查询符号表。若A是过程（函数）名，就把A作为过程类号送出，否则就是一般标识符。
- 注：这就要求数组元素或过程（函数）要先定义，后使用；否则编译程序至少需要两遍扫描。

## 第七节 说明语句的翻译

### 一、说明语句的编译任务

- 1) 登记符号表，包括填名字及有关属性；
- 2) 为变量分配内存地址（相对地址），包括内情向量表地址及构造部分或全部向量表。

注：1) 绝大多数程序语言的说明语句不产生中间代码。

2) 词法分析虽然也查造符号表，但它不知道标识符的属性，对该标识符所处的环境也一无所知，符号表很不成熟。

# 第七节 说明语句的翻译

## 二、分程序结构的符号表

- 某些高级语言允许过程递归调用和过程嵌套，这就需要建立分程序结构。
- 1、这些语言的标识符具有以下性质：
  - 1 ) 所有标识符必须先定义后使用；
  - 2 ) 标识符的使用遵守最小作用域原则；
    - 即标识符的使用范围不超过定义它的分程序；
  - 3 ) 同一分程序内标识符不允许重名，不同分程序内的相同标识符表示不同的名字。

## 第七节 说明语句的翻译

### 二、分程序结构的符号表

#### 2、过程嵌套关系的表示：

##### 1)使用顺序号(BLKN)和嵌套层次号(LN)

- 给分程序编两种号码，
  - BLKN：过程出现的先后排的顺序号(从1开始)；
  - LN：嵌套层次号。用来实现对直接外层外部量的引用，并规定主程序main的嵌套层次号为0。

##### 2)建立一个分程序总表(BLKLIST)

- 用以管理各个分程序中的说明信息和各分程序变量间的相互关系。

# 第七节 说明语句的翻译

## 二、分程序结构的符号表

### 2、过程嵌套关系的表示：

#### 2)建立一个分程序总表(BLKLIST)

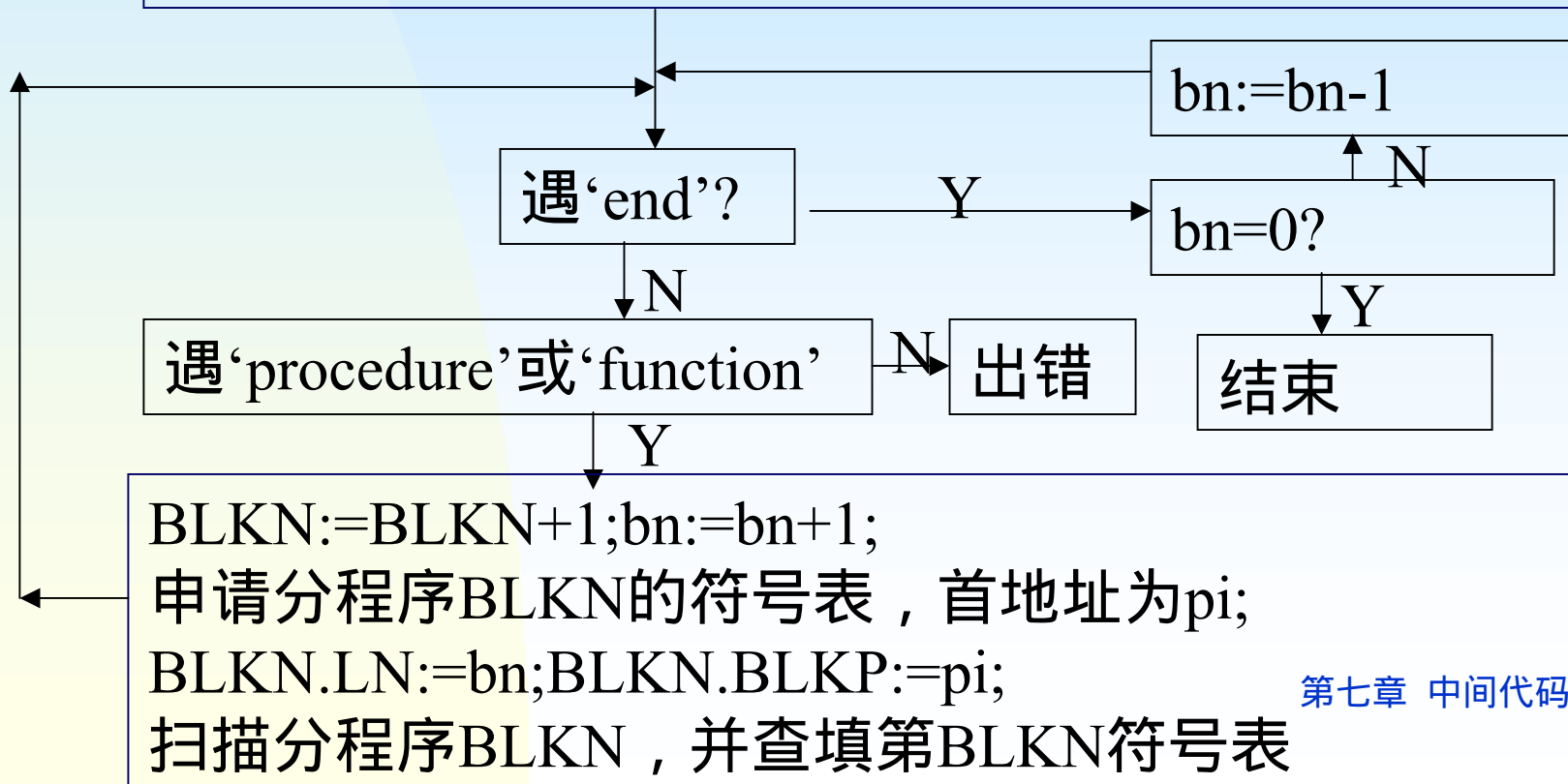
注：1)每扫描一个分程序，就在分程序总表(BLKLIST)建立一项，同时建立一张该分程序符号表。

2)分程序总表中包含各分程序顺序号(BLKN)、层号(LN)以及指向各符号表的指针BLKP。

3)分程序符号表主要由名字(NAME)、种属(CAT)、类型(TYPE)、值(VAL)、和地址(ADDR)组成。

4)有关分程序符号表的填写算法如下：

遇‘program’在分程序总表BLKLIST建立一项，  
BLK:=1;  
bn:=0; /\*bn用作层号计数器\*/  
BLKN.LN:=bn;  
申请主程序符号表区，首地址为p;  
BLKN.BLKP:=p;  
扫描主程序的说明语句，并查填第BLKN符号表





# 第七节 说明语句的翻译

## 二、分程序结构的符号表

### 3、符号表管理技术：

- 1) 符号表采取线性表，其管理采用线性查找技术。
- 2) 要提高线性表的查找速度，在线性表中增加一栏——指示器，它将所有的项按“最新最近”访问原则连成一条链。
  - 这种线性表叫自适应线性表，这种“最新最近”访问原则访问概率较大。
- 3) 除了线性表之外，还采用二叉排序树，和散列表等，它们适于大型表格的查造。

- Program MAIN
- var A,B,C:real;
- procedure P(var X:integer);
- var A,Y:integer;
- B:array[1..10,10..20]of real;
- function Q(var T:integer):boolean;
- var X:integer;
- begin         .....         end;
- begin
- Q(Y);
- end;
- procedure R(var Y:integer);
- var X:integer;

```

begin
    P(X);
end;

begin
    R(A);
End.

```

分程序总表BLKLIST

顺序号 BLKN	层号 LN	指针 BLKP
1	0	
2	1	
3	2	
4	1	
...	...	

主程序MAIN的符号表

过程P的符号表

NAME	CAT	TYPE	VAL	ADDR
P	PROC			
X	形参	int		
A	简变	int		
Y	简变	int		
B	数组	real		

过程R的符号表

函数Q的符号表

第七章 中间代码生成  
B的内情向量表

## 第七节 说明语句的翻译

### 三、整型、实型说明语句的翻译

#### 1、文法：

–  $D \rightarrow \text{integer Namelist} \mid \text{real Namelist}$

–  $\text{Namelist} \rightarrow \text{Namelist}, i \mid i$

- 注:这个文法进行制导翻译时只有把所有的名字都归约成Namelist之后才能把它们属性登记进符号表。

#### 2、改写文法：

- $D \rightarrow D, i \mid \text{integer } i \mid \text{real } i$

## 第七节 说明语句的翻译

### 三、整型、实型说明语句的翻译

#### 3、语义子程序的相关过程

- ENTRY(i):
  - 查造符号表；
- FILL(P,A)：
  - 将属性A填到P所指符号表项相应栏内，可填CAT,TYPE;
- ALLOCATION(T)：
  - 根据T的属性为变量分配存储单元数，并回送数据区指针。同时让数据区指针指向下一个可用单元。

## 第七节 说明语句的翻译

### 三、整型、实型说明语句的翻译

#### 4、语义子程序

- $D \rightarrow \text{integer } i$        $\{\text{FILL}(\text{ENTRY}(i), \text{int});$ 
  - $D \bullet \text{ATT} = \text{int};$
  - $\text{FILL}(\text{ENTRY}(i), \text{ALLOCATION}(D \bullet \text{ATT}));\}$
- $D \rightarrow \text{real } i$        $\{\text{FILL}(\text{ENTRY}(i), \text{real});$ 
  - $D \bullet \text{ATT} = \text{real};$
  - $\text{FILL}(\text{ENTRY}(i), \text{ALLOCATION}(D \bullet \text{ATT}));\}$
- $D \rightarrow D^{(1)}, i$        $\{\text{FILL}(\text{ENTRY}(i), D \bullet \text{ATT});$ 
  - $D \bullet \text{ATT} = D^{(1)} \bullet \text{ATT};$
  - $\text{FILL}(\text{ENTRY}(i), \text{ALLOCATION}(D \bullet \text{ATT}));\}$

## 第七节 说明语句的翻译

### 四、常量定义语句的翻译

- 1、文法

- $\text{CONSTD} \rightarrow \text{CONST Constlist} \mid \varepsilon$
- $\text{Constlist} \rightarrow i=\text{INT};\text{Constlist} \mid i=\text{STR};\text{Constlist} \mid$   
 $i=\text{INT} \mid i=\text{STR}$

- 注：INT代表某常量，值在常量表中；STR代表字符常量，在符号表中可查到。

- 2、语义动作

- 在符号表中为变量i填写其值、类型和种属。

## 第七节 说明语句的翻译

### 四、常量定义语句的翻译

#### 3、语义子程序

- $\text{CONSTD} \rightarrow \text{CONST Constlist} \mid \varepsilon$  无需语义子程序
- $\text{Constlist} \rightarrow i=\text{INT}; \text{Constlist} \mid i=\text{INT}$ 
  - $\{\text{ENTRY}(i) \cdot \text{VAL} = \text{ENTRY}(\text{INT});$
  - $\text{ENTRY}(i) \cdot \text{TYPE} = \text{int};$
  - $\text{ENTRY}(i) \cdot \text{CAT} = \text{数值常量}; \}$
- $\text{Constlist} \rightarrow i=\text{STR}; \text{Constlist} \mid i=\text{STR}$ 
  - $\{\text{ENTRY}(i) \cdot \text{VAL} = \text{ENTRY}(\text{STR});$
  - $\text{ENTRY}(i) \cdot \text{TYPE} = \text{char};$
  - $\text{ENTRY}(i) \cdot \text{CAT} = \text{字符常量}; \}$



# 第七节 说明语句的翻译

## 五、数组说明语句的翻译

### 1、文法

- $A \rightarrow i:\text{array } [Alists] \text{ of } TYPE$
- $Alists \rightarrow Alist \mid Alists, Alist$
- $Alist \rightarrow \text{int} \mid \text{int}^{(1)}..\text{int}^{(2)}$
- $TYPE \rightarrow \text{integer} \mid \text{real} \mid \text{bool} \mid \text{char}$
- 注：int表示该维的上界，隐含下界为1；  
int<sup>(1)</sup>..int<sup>(2)</sup>分别表示该维下、上界；  
Alists是下标界表,Alist是单个下标界。

## 第七节 说明语句的翻译

### 五、数组说明语句的翻译

#### 2、语义动作

- 填写符号表和构造内情向量表P。

#### 3、语义子程序

- $A \rightarrow A_1 \text{ of TYPE}$        $\{\text{FILL}(P \bullet \text{TYPE}, t)\}$
- $A_1 \rightarrow A_2 [Alists]$        $\{\text{FILL}(P \bullet C, C) ;$ 
  - $\text{FILL}(P \bullet n, \text{DIM}) ; \}$
- $A_2 \rightarrow i:\text{array}$        $\{L = \text{ENTRY}(i);$ 
  - $\text{FILL}(L \bullet \text{CAT}, 'ARRAY');$
  - 申请一张内情向量表，首址为p,
  - $P = p; \text{FILL}(L \bullet \text{ADDR}, P);$
  - $\text{DIM} = 0; \}$

# 第七节 说明语句的翻译

## 五、数组说明语句的翻译

### 3、语义子程序

- $Alists \rightarrow Alist$   $\{DIM=DIM+1\}$
- $Alists \rightarrow Alists, Alist$   $\{DIM=DIM+1\}$
- $Alist \rightarrow int$   $\{FILL (P \bullet l_{dim}, 1);$   
—  $FILL (P \bullet u_{dim}, int);$   
—  $FILL (P \bullet d_{dim}, int); \}$
- $Alist \rightarrow Alt \ int^{(2)}$   $\{ FILL (P \bullet u_{dim}, int^{(2)});$   
—  $FILL (P \bullet d_{dim}, (int^{(2)} - int^{(1)} + 1)); \}$
- $Alt \rightarrow int^{(1)} \ ..$   $\{FILL (P \bullet l_{dim}, int^{(1)}) \}$
- $TYPE \rightarrow integer \mid real \mid bool \mid char$   
—  $\{ \}$

## 第七节 说明语句的翻译

### 六、过程说明语句的翻译

#### 1、过程说明的首部定义

- PROCEDURE <过程名>(<形参表>);
- FUNCTION <函数名>(<形参表>) : TYPE;
- 注：当分析到它们时，应该产生计数的层号LN和顺序号BLKN，建立分程序符号表，并在BLKLIST表中增加一个表项。在分程序符号表中填写过程名（或函数名）、形参名及其有关的属性。

# 第七节 说明语句的翻译

## 六、过程说明语句的翻译

### 2、语义子程序

- $P \rightarrow PP(\text{Parg}) \quad \{\}$
- $P \rightarrow PF(\text{Parg}):TYPE \quad \{\text{FILL}(\text{L.TYPE}, TYPE)\}$
- $PP \rightarrow \text{procedure } i$ 
  - $\{LN=LN+1;$
  - $BLKN=BLKN+1;$

为过程*i*建立一个分程序符号表，设其首址为*p*，在BLKLIST表中增加一行，将BLKN, LN和*p*填入；

$L=ENTRY(i);$   
 $FILL(L.CAT, 'proc')\}$

# 第七节 说明语句的翻译

## 六、过程说明语句的翻译

### 2、语义子程序

- $PF \rightarrow \text{function } i$

- $\{LN=LN+1;$

- $BLKN=BLKN+1;$

为过程*i*建立一个分程序符号表，设其首址为*p*，在BLKLIST表中增加一行，将BLKN, LN和*p*填入；

$L=ENTRY(i);$

$FILL(L.CAT, 'function')\}$

# 第七节 说明语句的翻译

## 六、过程说明语句的翻译

### 2、语义子程序

- $\text{Parg} \rightarrow \text{Arg:TYPE}$ 
  - $\{\text{FILL}(\text{K.TYPE}, \text{TYPE})\}$
- $\text{Parg} \rightarrow \text{Arg:TYPE}, \text{Parg}$ 
  - $\{\text{FILL}(\text{K.TYPE}, \text{TYPE})\}$
- $\text{Arg} \rightarrow i$   $\{\text{K}=\text{ENTRY}(i);$ 
  - $\text{FILL}(\text{K.CAT}, \text{“形参”})\}$

# 第八节 输入/输出语句的翻译

## 一、输入输出语句的处理方式

- 以设备为中心
- 以文件为中心

## 二、输入输出语句的执行

- I/O控制系统
- 基本I/O系统

## 三、编译对I/O语句的处理

- 将这些语句转换为系统子程序所要求的参数形式,然后调用这些系统子程序完成I/O功能。
- 注：不同机器、不同系统，要求编译程序将I/O语句加工的结果不同。故，无通用的翻译方法。



# 第八节 输入/输出语句的翻译

## 四、PASCAL语言中的输入输出操作

- Input
- Output
- Read
- Write
- Readln
- Writeln

## 第八节 输入/输出语句的翻译

五、对输入/输出语句的几点规定：

- 1、所有出现在参数表中的变量必须是说明过的；
- 2、所有出现在write或writeln的表达式中的变量必须是已赋值的；
- 3、所以I/O文件都是顺序文件，也即仅考虑顺序访问，不考虑直接访问和索引访问；
- 4、忽略参数表中的格式说明。

## 第八节 输入/输出语句的翻译

### 六、文法:

- 1、PROG    program i ( $\epsilon$ |'('PROG\_PAR')');
- 2、PROG\_PAR    input|output|input,output;
- 3、READST    (read|readln)('ID')
- 4、WRITEST    (write|writeln)('ELIST')

## 第八节 输入/输出语句的翻译

### 七、语义子程序

- READST read(ID)
- {n := 0 ;
- repeat
- 依指针n , 从队列ID.QUEUE取出一项 $i_n$ ;
- GEN(par,\_,\_,  $i_n$ );
- n:=n+1
- until ID.QUEUE为空;
- GEN(call,0,n,SY SIN);} /\*0代表read,n代表系统  
读n个数据,SY SIN为系统输入子程序\*/

## 第八节 输入/输出语句的翻译

### 七、语义子程序

- ID  $i$
- {建立一个ID.QUEUE空队列;并将添入队列}
- ID  $i, ID$
- {把ENTRY( $i$ )添入队列的末端}

## 第八节 输入/输出语句的翻译

### 七、语义子程序

- WRITEST write(ELIST)
  - {n := 0 ;
  - repeat
  - 依指针n , 从队列ID.QUEUE取出一项 $i_n$ ;
  - GEN(par,\_,\_,  $i_n$ );
  - n:=n+1
  - until ID.QUEUE为空;
  - GEN(call,w,n,SYSOUT);} /\*w代表read,n代表系统读n个数据,SYSOUT为系统输入子程序\*/

# 第九节 自上而下分析制导的翻译

## 一、自上而下分析制导翻译

通过推导自左而右地产生句子的各终结符号。再按产生式推导的过程添加语义子程序，无需待到整个候选式匹配完后，即不必改写文法。

注：1)在按产生式推导的中间任何一步都可以添加语义子程序.所以不必改写文法。

2)实际上，程序设计语言绝大部分均可使用自上而下分析，并生成相应中间代码。

## 二、自上而下分析制导翻译种类

1、递归下降分析制导翻译

2、LL(1)分析制导翻译

# 第九节 自上而下分析制导的翻译

## 三、递归下降分析制导翻译

### 1、基本思想

给每个文法符号配以语义动作，用到的语义变量不用语义栈单元，而是用局部变量。

### 2、算术表达式的翻译

#### 1) 文法

- $E \rightarrow E + T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow ( E ) \mid i$

消除左递归并改写成扩充BNF表示法：

- $E \rightarrow T \{ + T \}$
- $T \rightarrow F \{ * F \}$
- $F \rightarrow ( E ) \mid i$



# 第九节 自上而下分析制导的翻译

## 三、递归下降分析制导翻译

### 2、算术表达式的翻译

#### 2) 翻译： $E \rightarrow T \{+T\}$ 的语法分析过程

```
– FUNCTION E: INTEGER
–   BEGIN
–       T;
–       WHILE SYM="+" DO
–           BEGIN
–               ADVANCE;
–               T;
–           END
–       END
–   END
```

- 语法制导翻译过程

- FUNCTION E: INTEGER

- BEGIN

- $\{E^{(1)}.PLACE := \} T;$

- WHILE SYM = “+” DO

- BEGIN

- ADVANCE;

- $\{E^{(2)}.PLACE := \} T;$

- $\{T_1 := NEWTEMP;$

- $GEN(+, E^{(1)}.PLACE, E^{(2)}.PLACE, T_1 \}$

- END

- $\{E := E^{(1)}.PLACE \}$

- END

- 根据 $F \rightarrow ( E ) | i$ 的语法制导翻译过程：
  - FUNCTION F: INTEGER
  - BEGIN
  - IF SYM='i' THEN
  - BEGIN {F:=ENTRY(i);} ADVANCE END
  - ELSE IF SYM='d' THEN
  - BEGIN {F:=ENTRY(d)+c;} ADVANCE END
  - ELSE IF SYM='(' THEN
  - BEGIN     ADVANCE;
  - {F:=}E;
  - IF SYM=')' THEN ADVANCE
  - ELSE ERROR(1)
  - END

- ELSE ERROR(2)
- END
- 注：‘i’表示变量；
- ‘d’表示常量，ENTRY(d)+c表示常量表入口地址。
- ERROR(1)表示缺“)”
- ERROR(2)表示缺因子。

# 第九节 自上而下分析制导的翻译

## 二、布尔表达式的翻译

- $E_b \rightarrow E_b \quad T_b | T_b$
- $T_b \rightarrow T_b \quad F_b | F_b$
- $F_b \rightarrow i_b | F_b | (E \text{ rop } E)$

消除左递归并改写成扩充BNF表示法：

$$\bullet E_b \rightarrow T_b \{ \quad T_b \}$$

$$\bullet T_b \rightarrow F_b \{ \quad F_b \}$$

$$\bullet F_b \rightarrow i_b | F_b | (E \text{ rop } E)$$

- 注：按控制语句中的布尔表达式翻译，主要是产生一组转移四元式，并留下待回填的真假链的链首。

- $E_b \rightarrow T_b \{ T_b \}$  的语法制导的翻译过程：
  - PROC  $E_b$ (VAR E.TC,E.FC:INTEGER);
  - BEGIN
  - $T_b(E^{(1)}.TC, E^{(1)}.FC)$ ;
  - WHILE SYM=“ ” DO
  - BEGIN
  - ADVANCE;
  - $BACKPATCH(E^{(1)}.FC, NXQ)$ ;
  - $T_b(E^{(2)}.TC, E^{(2)}.FC)$ ;
  - $E^{(1)}.TC := MERG(E^{(1)}.TC, E^{(2)}.TC)$ ;
  - $E^{(1)}.FC := E^{(2)}.FC$ ;
  - END;
  - $E.TC := E^{(1)}.TC$ ;  $E.FC := E^{(1)}.FC$ ;
  - END

- $T_b \rightarrow T_b$   $F_b|F_b$ 的语法制导的翻译过程：
  - PROC  $T_b$ (VAR T.TC,T.FC:INTEGER);
  - BEGIN
  - $F_b(T^{(1)}.TC, T^{(1)}.FC);$
  - WHILE SYM=“ ” DO
  - BEGIN
  - ADVANCE;
  - $BACKPATCH(T^{(1)}.FC, NXQ);$
  - $F_b(T^{(2)}.TC, T^{(2)}.FC);$
  - $T^{(1)}.TC := MERG(T^{(1)}.TC, T^{(2)}.TC);$
  - $T^{(1)}.TC := T^{(2)}.TC;$
  - END;
  - $T.TC := T^{(1)}.TC; T.FC := T^{(1)}.FC;$

•  $F_b \rightarrow i_b \mid F_b(E \text{ rop } E)$ 的语法制导的翻译过程：

```
– PROC  $F_b$ (VAR F.TC,F.FC:INTEGER);  
– BEGIN  
–   IF SYM=' $i_b$ ' THEN  
–     BEGIN  $F.TC := NXQ$ ;  
–            $GEN(jnz, ENTRY(i_b), \_, 0)$ ;  
–            $F.FC := NXQ$ ;  
–            $GEN(j, \_, \_, 0)$ ;  
–           ADVANCE  
–     END;  
–   ELSE IF SYM='  ' THEN  
–     .....
```



# 第九节 自上而下分析制导的翻译

## 三、递归下降分析制导翻译

### 4、简单语句的翻译

- $S \rightarrow \text{if } E_b \text{ then } S^{(1)}$
- $| \text{if } E_b \text{ then } S^{(1)} \text{ else } S^{(2)}$
- $| \text{while } E_b \text{ do } S^{(1)}$
- $| \text{begin } L \text{ end}$
- $| i := E$
- $| \text{read}(\text{IDT})$
- $| \text{write}(\text{ET})$
- $| \varepsilon$
- $L \rightarrow L; S | S$

# 第九节 自上而下分析制导的翻译

## 三、递归下降分析制导翻译

### 4、简单语句的翻译

消除左递归后文法改写为：

- $S \rightarrow \text{if } E_b \text{ then } S^{(1)} \text{ TAIL}$
- $\quad | \text{while } E_b \text{ do } S^{(1)}$
- $\quad | \text{begin } L \text{ end}$
- $\quad | i := E$
- $\quad | \text{read}(\text{IDT})$
- $\quad | \text{write}(\text{ET})$
- $\quad | \varepsilon$
- $\text{TAIL} \rightarrow \text{else } S^{(2)} | \varepsilon$
- $L \rightarrow S \{ ; S \}$

```

– PROC S(VAR S.CHAIN:INTEGER);
–   BEGIN ADVANCE;
–     CASE SYM OF
–   ‘if’:BEGIN
–     ADVANCE;
–     Eb (TC,FC);
–     IF SYM=‘then’ THEN
–       BEGIN ADVANCE;
–         BACKPATCH (TC,NXQ);
–         S(S(1).CHAIN);
–     END
–   ELSE ERROR;

```

```

— IF SYM='else' THEN
—   BEGIN ADVANCE;
—     q:=NXQ;
—     GEN(j,_,_,0);
—     BACKPATCH(FC,NXQ);
—     S(1).CHAIN:=MERG(S(1).CHAIN,q)
—     S(S(2).CHAIN);
—     S(1).CHAIN:=MERG(S(1).CHAIN, S(2).CHAIN);
—   END
— ELSE
—   S.CHAIN:=MERG(S(1).CHAIN, S(2).CHAIN);
— END;
— “while”:BEGIN .....

```

# 第九节 自上而下分析制导的翻译

## 四、LL(1)语法制导翻译

### 1、算法思想：

- 预先在文法产生式的相应位置上嵌入**语义动作符号**，用来提示语法分析程序，当分析到达这些位置时，应调用相应的语义子程序。动作符号通常表示为语义子程序的入口。
- 注：1、这些带有动作符号的文法称为翻译文法。为了和文法符号区别开，把翻译文法中的动作符号用{}括起来。
- 2、与递归下降法不同，LL(1)分析法需要使用语义栈记录分析过程中的语义信息，但其语义栈的操作与自下而上分析法中的语义栈操作不同。

## 第九节 自上而下分析制导的翻译

- 例如：赋值语句文法：
- $A \rightarrow i\{a_1\} := \{a_2\} E \{a_3\}$
- $E \rightarrow TE'$
- $E' \rightarrow + \{a_2\} T \{a_4\} E' \mid \varepsilon$
- $T \rightarrow FT'$
- $T' \rightarrow * \{a_2\} F \{a_4\} T' \mid \varepsilon$
- $F \rightarrow (E) \mid i \{a_1\}$

•注：在自上而下的分析中，进行语法分析时动作符号一起进入语法分析栈，语义栈和语法栈 是分开操作的，语义动作主要是针对语义栈操作的。

## 第九节 自上而下分析制导的翻译

- 其语义动作的含义为：
  - $a_1$  : 查找符号表，取变量 $i$ 的入口地址。即：取ENTRY( $i$ ),并将其移进语义栈。
  - $a_2$  : 将分析栈栈顶与读头下匹配符号移进语义栈。
  - $a_3$  : 弹出语义栈栈顶三项，生成赋值四元式。
  - $a_4$  : 弹出语义栈栈顶三项，生成二目运算四元式。
- 例如：采用LL(1)分析法，将语句 $A:=B*(C+D)$ 翻译成中间代码。

步骤	分析栈	语义栈	输入串	四元式/动作
0	A		A:=B*(C+D)#	
1	$\{a_3\}$ E $\{a_2\}$ := $\{a_1\}$ i		A:=B*(C+D)#	推导
2	$\{a_3\}$ E $\{a_2\}$ :=	A	:=B*(C+D)#	匹配
3	$\{a_3\}$ E	A:=	B*(C+D)#	移进
4	$\{a_3\}$ E`T`F	A:=	B*(C+D)#	推导2步
5	$\{a_3\}$ E`T` $\{a_1\}$ i	A:=	B*(C+D)#	推导
6	$\{a_3\}$ E`T`	A:=B	*(C+D)#	匹配
7	$\{a_3\}$ E`T` $\{a_4\}$ F $\{a_2\}$ *	A:=B	*(C+D)#	推导
8	$\{a_3\}$ E`T` $\{a_4\}$ F	A:=B*	(C+D)#	移进
9	$\{a_3\}$ E`T` $\{a_4\}$ )E(	A:=B*	(C+D)#	推导
10	$\{a_3\}$ E`T` $\{a_4\}$ )E	A:=B*	C+D)#	匹配不移进
11	$\{a_3\}$ E`T` $\{a_4\}$ )E`T`F	A:=B*	C+D)#	推导2步
12	$\{a_3\}$ E`T` $\{a_4\}$ )E`T` $\{a_1\}$ i	A:=B*	C+D)#	推导



步骤	分析栈	语义栈	输入串	四元式/动作
13	$\{a_3\}E'T'\{a_4\})E'T'$	$A:=B*C$	$+D)\#$	匹配
14	$\{a_3\}E'T'\{a_4\})E'$	$A:=B*C$	$+D)\#$	推导空串
15	$\{a_3\}E'T'\{a_4\})E'\{a_4\}T\{a_2\}+$	$A:=B*C$	$+D)\#$	推导
16	$\{a_3\}E'T'\{a_4\})E'\{a_4\}T$	$A:=B*C+$	$D)\#$	匹配
17	$\{a_3\}E'T'\{a_4\})E'\{a_4\}T'\{a_1\}i$	$A:=B*C+$	$D)\#$	推导两步
18	$\{a_3\}E'T'\{a_4\})E'\{a_4\}T'$	$A:=B*C+D$	$)\#$	匹配
19	$\{a_3\}E'T'\{a_4\})E'\{a_4\}$	$A:=B*C+D$	$)\#$	推导空串
20	$\{a_3\}E'T'\{a_4\})$	$A:=B* T_1$	$)\#$	$(+,C,D,T_1)$
21	$\{a_3\}E'T'\{a_4\}$	$A:=B* T_1$	$\#$	匹配不移进
22	$\{a_3\}E'$	$A:=T_2$	$\#$	$(*,B, T_1, T_2)$
23			$\#$	$(:=, T_2, \_, A)$

## 第九节 自上而下分析制导的翻译

- 注：用同样的方法，也可以翻译其他语句。
- 例如： $S \rightarrow \text{while}\{w_1\}E \text{ do } \{w_2\}S^{(1)} \{w_3\}$ 
  - 其中 $w_1$ ：记住WHILE语句的入口四元式序号。
  - $w_2$ ：布尔式E翻译后留E.TC在栈顶，E.FC在次栈顶，出栈，回填TC。
  - $w_3$ ： $S^{(1)}$ 翻译后，回填 $S^{(1)}$ .CHAIN的内容（若需要回填的话），程序转移到while语句开始处；E.FC留待回填，保留在S.CHAIN中放在栈顶。

# 第十节 中间代码的其它形式

## 一、后缀表示法（逆波兰表示法）

### 1、后缀表示法的文法

- $\langle \text{后缀式} \rangle ::= \langle \text{标识符} \rangle | \langle \text{后缀式} \rangle \langle \text{单目算符} \rangle | \langle \text{后缀式} \rangle \langle \text{后缀式} \rangle \langle \text{双目算符} \rangle$
- $\langle \text{单目算符} \rangle ::= @$
- $\langle \text{双目算符} \rangle ::= + | - | * | /$
- 注：这种表示法不带括号，根据运算量和运算符出现的先后位置，以及每个运算符的目数，就完全决定了一个表达式的计算顺序。

## 第十节 中间代码的其它形式

### 2、后缀表示法的特点：

- (1)运算量的排列顺序与中缀表示法相同；
- (2)运算符是按运算的顺序排列的；
- (3)运算符紧跟在被运算的对象之后出现。
- 注：它虽然不符合人的习惯，但对计算机来说，很容易使用一个栈来计算它的值，或转换成另一种代码。

# 第11节 中间代码的其它形式

## 二、语法制导产生后缀式

- 首先，除了下推栈外，为分析过程设置一个一维数组 POST 来存放后缀式；初始时，其下标  $k$  为 1。
- 然后，利用某种分析法进行语法制导翻译。
  - 例如：利用算符优先分析法

# 第十节 中间代码的其它形式

## 二、语法制导产生后缀式

### 1、利用算符优先分析法进行语法分析

#### 语义子程序

- | 产生式   | 语义子程序   |
|---|---|
| • $(1) E \rightarrow E^{(1)} \text{ op } E^{(2)}$ | $\{\text{POST}[k]=\text{OP}; k=k+1; \}$         |
| • $(2) E \rightarrow (E^{(1)})$                   | $\{ \}$   |
| • $(3) E \rightarrow i$                           | $\{\text{POST}[k] = \text{ENTRY}(i); k=k+1; \}$ |
- 
- 例如：假定算符优先分析表已造好，对表达式  $A*(B+C)$  翻译成后缀式。

步骤	下推栈	输入串	后缀式
0	#	$A*(B+C)\#$	
1	#E	$*(B+C)\#$	A
2	#E*	$(B+C)\#$	A
3	#E*(	$B+C)\#$	A
4	#E*(E	$+C)\#$	AB
5	#E*(E+	$C)\#$	AB
6	#E*(E+E	)#	ABC
7	#E*(E	)#	ABC+
8	#E*(E)	#	ABC+
9	#E*E	#	ABC+
10	#E	#	$ABC+*$

# 第十节 中间代码的其它形式

## 二、语法制导产生后缀式

### 2、后缀表示法的计值和产生中间代码

- 1) 计值过程：自左至右扫描后缀式，每碰到运算量就将其入栈，每遇到 $k$ 目运算符就将它作用于栈顶的 $k$ 项，并将运算结果来代替这 $k$ 项。
  - 例如： $ab+c*$
- 2) 产生四元式过程：自左至右扫描后缀式，每碰到运算量就将其入栈，每遇到 $k$ 目运算符就将它作用于栈顶的 $k$ 项，并生成相应的中间代码，以结果的临时变量序号代替该栈顶的 $k$ 项。



# 第11节 中间代码的其它形式

## 二、语法制导产生后缀式

### 3、后缀式的推广


- 方法：遵守运算符紧跟在被作用的运算量之后。

- 例如：(1)赋值语句 $A:=E$ ,后缀式为： $AE:=$

(2)转向语句GOTO L, 后缀式为： $L' \text{ jmp}$

– 其中 $L'$ 是指示器，用POST中的下标值表示。

- (3)条件语句if  $x>y$  then  $m:=x$  else  $m:=y$ ;;其后缀值为:



1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	y	>	11	jez	m	x	:=	14	j	m	y	:=	...

Begin

k:=100;

L: if k>i+j then

begin k:=k-1; goto L end

else k:=i<sup>2</sup>-j<sup>2</sup>;

i:=0;

End.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
k	100	:=	k	i	j	+	>	20	jez	k	k	1	-	:=	4	j

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
29	j	k	i	2	^	j	2	^	-	:=	i	0	:=	...

注：翻译成后缀式时也存在拉链回填的问题。 第七章 中间代码生成 186

# 小结

- 1、算术表达式及赋值语句翻译
- 2、控制语句中布尔表达式翻译
  - 留有真、假出口
  - 回填过程与并链函数
- 3、标号和转移语句翻译
- 4、IF语句的翻译
- 5、WHILE语句的翻译
- 6、数组元素及其在赋值语句中的翻译
  - 内情向量表：a,C(必须会计算),n(维数),TYPE
  - V.PLACE[V.OFFSET]
  - V.PLACE=a-C, V.OFFSET与具体下标值有关

# 小结

- 7、分程序结构的符号表  
分程序控制总表、分程序符号表  
BLKN, LN
- 8、算术表达式的自上而下语法制导翻译  
在按产生式推导的中间任何一步均可添加语义子程序，不必改写文法
- 9、LL(1)语法制导翻译  
预先在文法产生式的相应位置上嵌入语义动作符号(相当于一段语义子程序)，用来提示语法分析程序，当分析到达这些位置时，应调用相应的语义子程序。
- 10、语法制导生成后缀式(逆波兰表达式)