

回填 (*Backpatching*)

➤ 基本思想

- 生成一个跳转指令时，暂时不指定该跳转指令的**目标标号**。这样的指令都被放入由跳转指令组成的**列表中**。**同一个列表中的所有跳转指令具有相同的目标标号**。等到能够确定正确的目标标号时，才去填充这些指令的目标标号

非终结符 B 的综合属性

- $B.truelist$: 指向一个包含跳转指令的列表, 这些指令最终获得的目标标号就是当 B 为真时控制流应该转向的指令的标号
- $B.falselist$: 指向一个包含跳转指令的列表, 这些指令最终获得的目标标号就是当 B 为假时控制流应该转向的指令的标号

函数

➤ *makelist(i)*

- 创建一个只包含 i 的列表， i 是跳转指令的标号，函数返回指向新创建的列表的指针

➤ *merge(p_1, p_2)*

- 将 p_1 和 p_2 指向的列表进行合并，返回指向合并后的列表的指针

➤ *backpatch(p, i)*

- 将 i 作为目标标号插入到 p 所指列表中的各指令中

布尔表达式的回填

➤ $B \rightarrow E_1 \text{ relop } E_2$

{

$B.\text{truelist} = \text{makelist}(\text{nextquad});$

$B.\text{falselist} = \text{makelist}(\text{nextquad}+1);$

$\text{gen}(\text{'if' } E_1.\text{addr relop } E_2.\text{addr 'goto _'});$

$\text{gen}(\text{'goto _'});$

}

布尔表达式的回填

➤ $B \rightarrow E_1 \text{ relop } E_2$

➤ $B \rightarrow \text{true}$

```
{  
    B.truelist = makelist(nextquad);  
    gen('goto _');  
}
```

布尔表达式的回填

➤ $B \rightarrow E_1 \text{ relop } E_2$

➤ $B \rightarrow \text{true}$

➤ $B \rightarrow \text{false}$

```
{  
    B.falselist = makelist(nextquad);  
    gen('goto _');  
}
```

布尔表达式的回填

➤ $B \rightarrow E_1 \text{ relop } E_2$

➤ $B \rightarrow \text{true}$

➤ $B \rightarrow \text{false}$

➤ $B \rightarrow (B_1)$

{

$B.\text{truelist} = B_1.\text{truelist} ;$

$B.\text{falselist} = B_1.\text{falselist} ;$

}

布尔表达式的回填

➤ $B \rightarrow E_1 \text{ relop } E_2$

➤ $B \rightarrow \text{true}$

➤ $B \rightarrow \text{false}$

➤ $B \rightarrow (B_1)$

➤ $B \rightarrow \text{not } B_1$

{

$B.\text{truelist} = B_1.\text{falselist};$

$B.\text{falselist} = B_1.\text{truelist};$

}

➡ $B \rightarrow B_1 \text{ or } B_2$

$B \rightarrow B_1 \text{ or } M B_2$

{

backpatch(B_1 .falselist, M .quad);

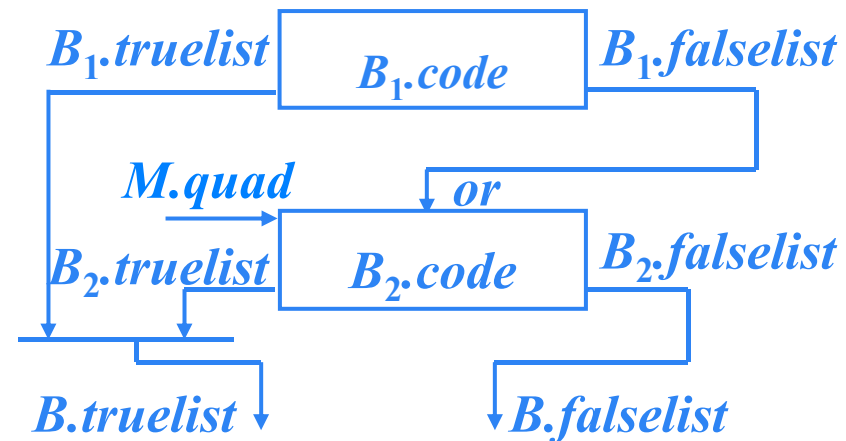
B .truelist = *merge*(B_1 .truelist, B_2 .truelist);

B .falselist = B_2 .falselist;

}

$M \rightarrow \varepsilon$

{ M .quad = *nextquad*; }





$B \rightarrow B_1 \text{ and } B_2$

$B \rightarrow B_1 \text{ and } M B_2$

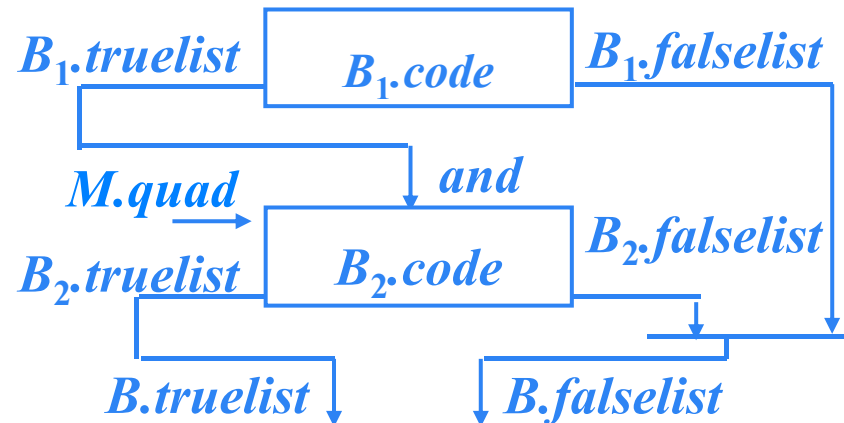
{

backpatch(B_1 .truelist, M .quad);

B .truelist = B_2 .truelist;

B .falselist = merge(B_1 .falselist, B_2 .falselist);

}





例

```
 $B \rightarrow E_1 \text{ relop } E_2$   
{  $B.\text{truelist} = \text{makelist}(\text{nextquad})$ ;  
   $B.\text{falselist} = \text{makelist}(\text{nextquad}+1)$ ;  
   $\text{gen}(\text{'if' } E_1.\text{addr relop } E_2.\text{addr 'goto _'})$ ;  
   $\text{gen}(\text{'goto _'})$ ;  
}
```

```
100:  $\text{if } a < b \text{ goto } \_$   
101:  $\text{goto } \_$ 
```

$t = \{100\}$
 $B \quad f = \{101\}$
 $a < b \quad \text{or} \quad c < d \quad \text{and} \quad e < f$



例

$B \rightarrow B_1 \text{ or } M B_2$

{

backpatch(B_1 .falselist, M .quad);

B .truelist = merge(B_1 .truelist, B_2 .truelist);

B .falselist = B_2 .falselist ;

}

$M \rightarrow \varepsilon$

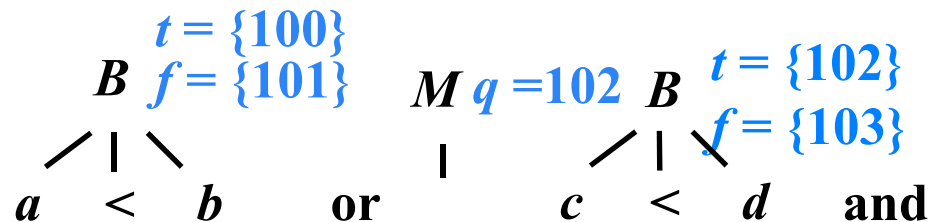
{ *M .quad = nextquad ;* }

100: *if $a < b$ goto _*

101: *goto _*

102: *if $c < d$ goto _*

103: *goto _*



$e < f$



$B \rightarrow B_1 \text{ and } M B_2$

{

$\text{backpatch}(B_1.\text{truelist}, M.\text{quad});$

$B.\text{truelist} = B_2.\text{truelist};$

$B.\text{falselist} = \text{merge}(B_1.\text{falselist}, B_2.\text{falselist});$

}

100: $\text{if } a < b \text{ goto } _$

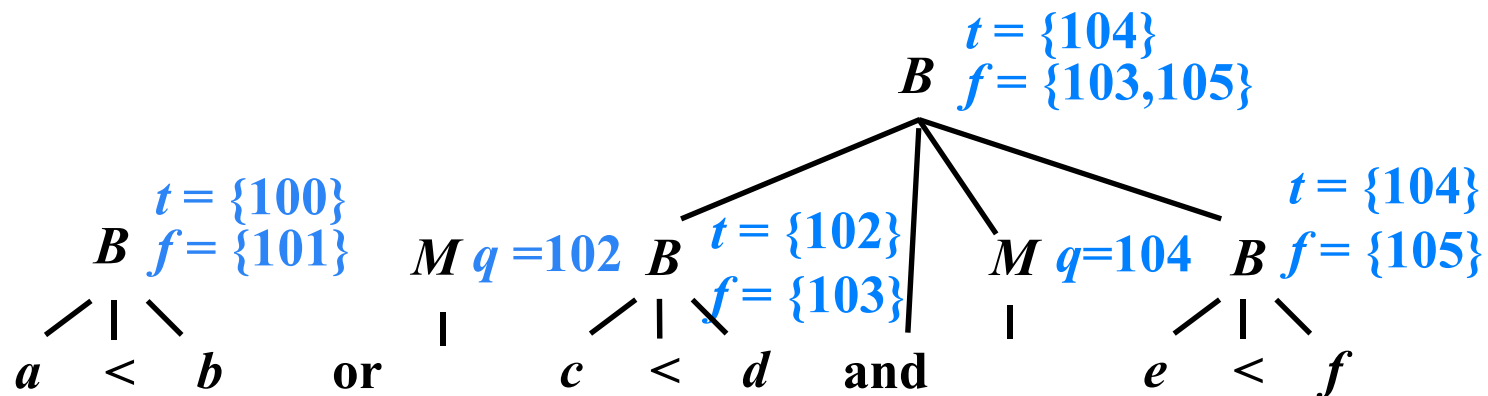
101: $\text{goto } _$

102: $\text{if } c < d \text{ goto } \underline{104}$

103: $\text{goto } _$

104: $\text{if } e < f \text{ goto } _$

105: $\text{goto } _$



例

$B \rightarrow B_1 \text{ or } M B_2$

{

backpatch(B_1 .falselist, M .quad);

B .truelist = *merge*(B_1 .truelist, B_2 .truelist);

B .falselist = B_2 .falselist ;

}

$t = \{100, 104\}$

$f = \{103, 105\}$

100: if $a < b$ goto _

101: goto 102

102: if $c < d$ goto 104

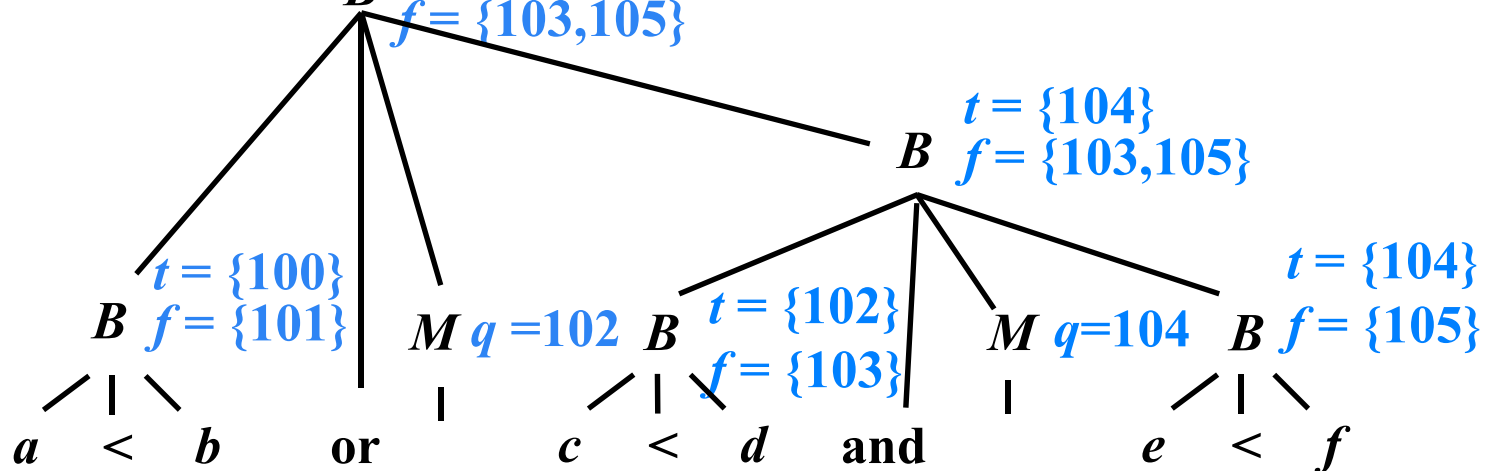
103: goto _

104: if $e < f$ goto _

105: goto _

t

f



控制流语句的回填

➤ 文法

➤ $S \rightarrow S_1 S_2$

➤ $S \rightarrow \text{id} = E ; \mid L = E ;$

➤ $S \rightarrow \text{if } B \text{ then } S_1$
 $\mid \text{if } B \text{ then } S_1 \text{ else } S_2$
 $\mid \text{while } B \text{ do } S_1$

➤ 综合属性

➤ $S.nextlist$: 指向一个包含跳转指令的列表, 这些指令最终获得的目标标号就是按照运行顺序紧跟在S代码之后的指令的标号

➡ $S \rightarrow \text{if } B \text{ then } S_1$

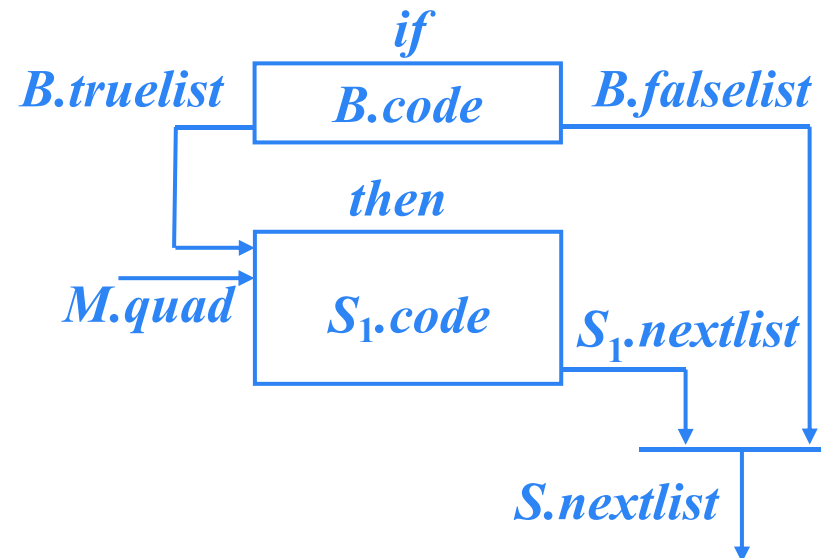
$S \rightarrow \text{if } B \text{ then } M S_1$

{

backpatch(B.truelist, M.quad);

S.nextlist = merge(B.falselist, S₁.nextlist);

}



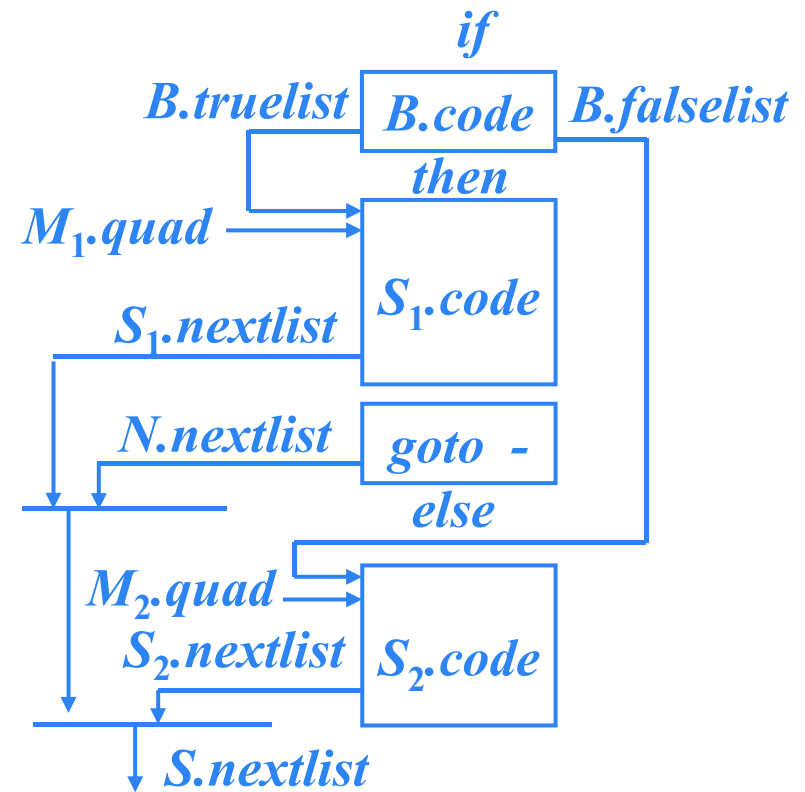
➡ $S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$

```

S → if B then M1 S1 N else M2 S2
{
    backpatch(B.truelist, M1.quad);
    backpatch(B.falselist, M2.quad);
    S.nextlist = merge( merge(S1.nextlist,
        N.nextlist), S2.nextlist );
}

N → ε
{ N.nextlist = makelist(nextquad);
  gen('goto _');
}

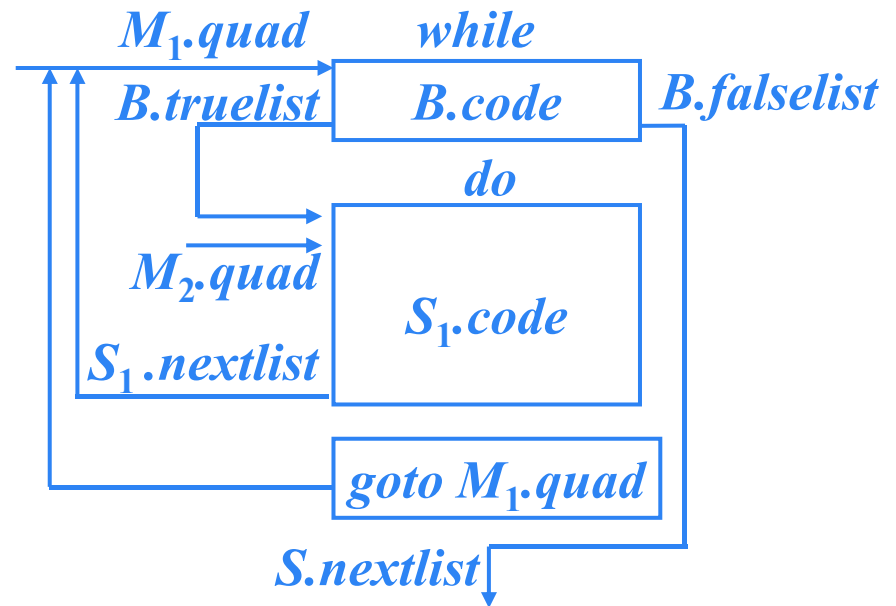
```



➡ $S \rightarrow \text{while } B \text{ do } S_1$

$S \rightarrow \text{while } M_1 B \text{ do } M_2 S_1$

```
{  
  backpatch(  $S_1.nextlist$ ,  $M_1.quad$  );  
  backpatch(  $B.truelist$ ,  $M_2.quad$  );  
   $S.nextlist = B.falselist$ ;  
  gen('goto'  $M_1.quad$ );  
}
```





$S \rightarrow S_1 S_2$

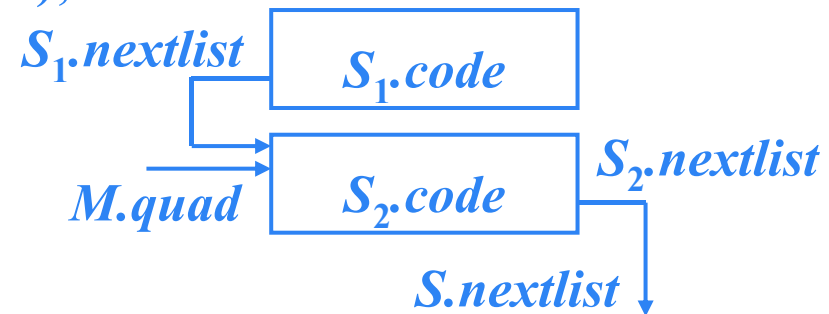
$S \rightarrow S_1 M S_2$


{

backpatch($S_1.nextlist$, $M.quad$);

$S.nextlist = S_2.nextlist$;

}



 $S \rightarrow \text{id} = E ; \mid L = E ;$

$S \rightarrow \text{id} = E ; \mid L = E ; \{ S.nextlist = null; \}$



while a < b do

if c < 5 then

while x > y do z = x + 1;

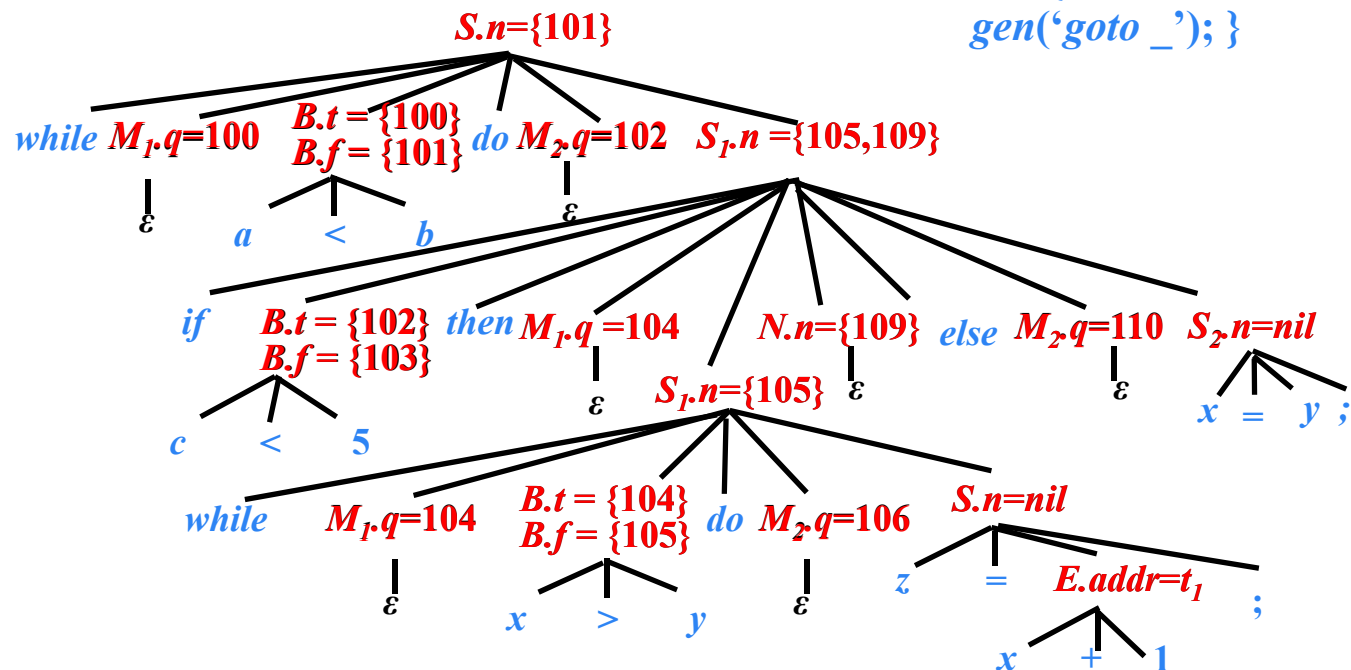
else

x = y;



$S \rightarrow \text{while } M_1 B \text{ do } M_2 S_1$
 $\{ S.\text{nextlist} = B.\text{falselist};$
 $\text{backpatch}(S_1.\text{nextlist}, M_1.\text{quad});$
 $\text{backpatch}(B.\text{truelist}, M_2.\text{quad});$
 $\text{gen}(\text{'goto'} M_1.\text{quad}); \}$

$S \rightarrow \text{if } B \text{ then } M_1 S_1 N \text{ else } M_2 S_2$
 $\{ S.\text{nextlist} = \text{merge}(\text{merge}(S_1.\text{nextlist}, N.\text{nextlist}),$
 $S_2.\text{nextlist});$
 $\text{backpatch}(B.\text{truelist}, M_1.\text{quad});$
 $\text{backpatch}(B.\text{falselist}, M_2.\text{quad}); \}$
 $N \rightarrow \epsilon \{ N.\text{nextlist} = \text{makelist}(\text{nextquad});$
 $\text{gen}(\text{'goto'} _); \}$



100: `if a < b goto 102`
 101: `goto _`
 102: `if c < 5 goto 104`
 103: `goto 110`
 104: `if x > y goto 106`
 105: `goto 100`
 106: `t1 = x + 1`
 107: `z = t1`
 108: `goto 104`
 109: `goto 100`
 110: `x = y`
 111: `goto 100`
 112:

语句 “`while a < b do if c < 5 then while x > y do z = x + 1; else x = y;`” 的注释分析树

 *while a < b do if c < 5 then while x > y do z = x + 1; else x = y;*

100: *if a < b goto 102*

101: *goto _*

102: *if c < 5 goto 104*

103: *goto 110*

104: *if x > y goto 106*

105: *goto 100*

106: $t_1 = x + 1$

107: $z = t_1$

108: *goto 104*

109: *goto 100*

110: $x = y$

111: *goto 100*

112:

100: (*j* <, *a*, *b*, 102)

101: (*j* , - , - , -)

102: (*j* <, *c*, 5, 104)

103: (*j* , - , - , 110)

104: (*j* >, *x*, *y*, 106)

105: (*j* , - , - , 100)

106: (+, *x*, 1, t_1)

107: (=, t_1 , - , z)

108: (*j* , - , - , 104)

109: (*j* , - , - , 100)

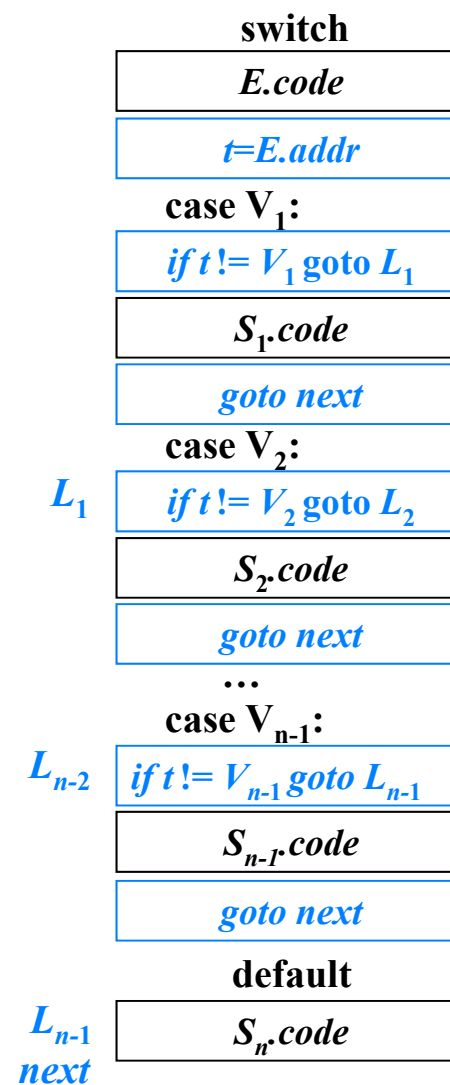
110: (=, *y*, - , *x*)

111: (*j* , - , - , 100)

112:

switch 语句的翻译

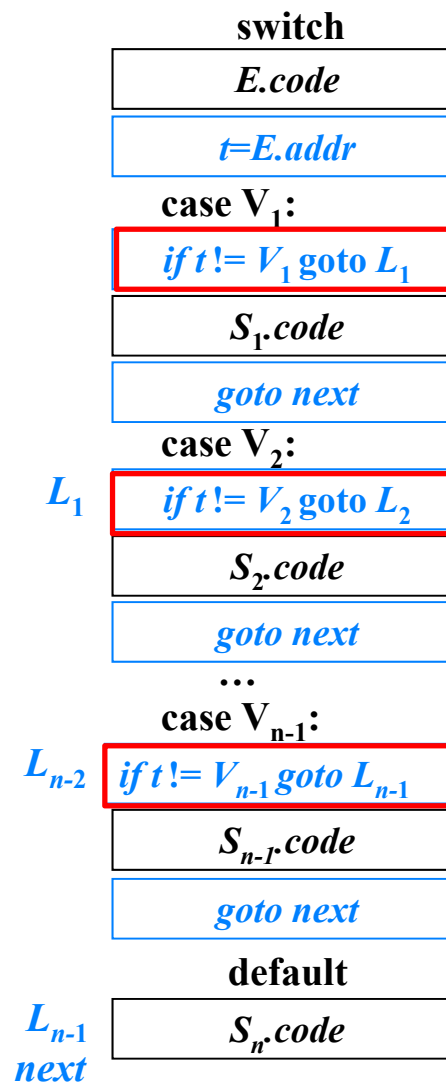
```
switch  $E$ 
begin
    case  $V_1$ :  $S_1$ 
    case  $V_2$ :  $S_2$ 
    ...
    case  $V_{n-1}$ :  $S_{n-1}$ 
    default:  $S_n$ 
end
```



switch 语句的翻译

```

switch E { t = newtemp(); gen( t '=' E.addr ); }
case  $V_1$ : {  $L_1$  = newlabel();
            gen('if' t '!='  $V_1$  'goto'  $L_1$  ); }
 $S_1$  { next = newlabel(); gen('goto' next); }
case  $V_2$ : { label( $L_1$ );  $L_2$  = newlabel();
            gen('if' t '!='  $V_2$  'goto'  $L_2$  ); }
 $S_2$  { gen('goto' next); }
...
case  $V_{n-1}$ : { label( $L_{n-2}$ );  $L_{n-1}$  = newlabel();
              gen('if' t '!='  $V_{n-1}$  'goto'  $L_{n-1}$  ); }
 $S_{n-1}$  { gen('goto' next); }
default: { label( $L_{n-1}$ ); }
 $S_n$  { label(next); }
    
```



*switch*语句的另一种翻译

switch E

begin

case $V_1: S_1$

case $V_2: S_2$

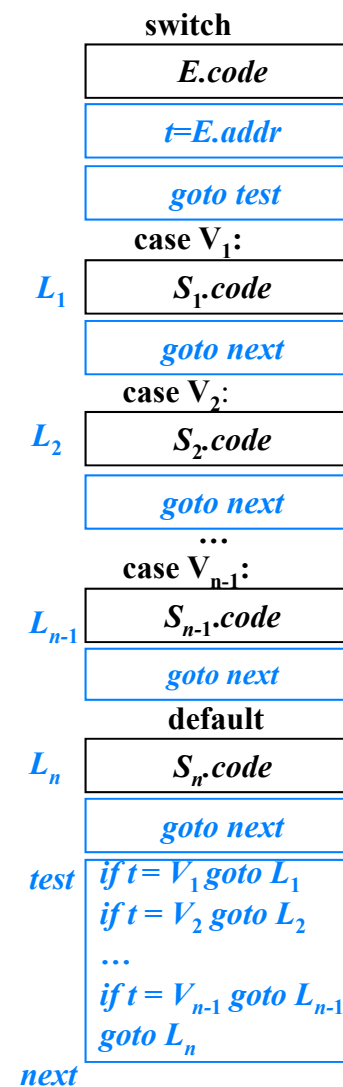
...

case $V_{n-1}: S_{n-1}$

default: S_n

end

在代码生成阶段，根据分支的个数以及这些值是否在一个较小的范围内，这种条件跳转指令序列可以被翻译成最高效的 n 路分支



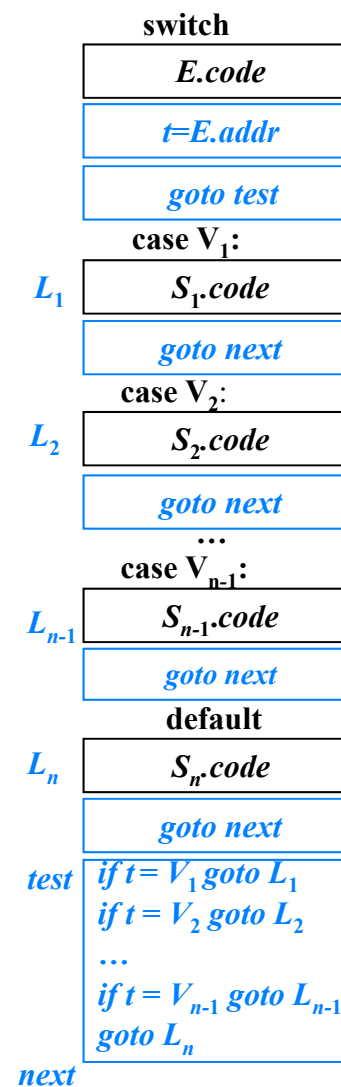


switch语句的另一种翻译

```

switch E { t = newtemp(); gen(t '=' E.addr);
           test = newlabel(); gen('goto' test); }
case V1: { L1 = newlabel(); label(L1); map(V1, L1); }
           S1 { next = newlabel(); gen('goto' next); }
case V2: { L2 = newlabel(); label(L2); map(V2, L2); }
           S2 { gen('goto' next); }
           ...
case Vn-1: { Ln-1 = newlabel(); label(Ln-1); map(Vn-1, Ln-1); }
           Sn-1 { gen('goto' next); }
default: { Ln = newlabel(); label(Ln); }
           Sn { gen('goto' next);
             label(test);
             gen('if' t '=' V1 'goto' L1);
             gen('if' t '=' V2 'goto' L2);
             ...
             gen('if' t '=' Vn-1 'goto' Ln-1);
             gen('goto' Ln);
             label(next);
           }

```



增加一种 *case* 指令

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

```
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 :
```

指令 $\text{case } t \ V_i \ L_i$ 和 $\text{if } t = V_i \text{ goto } L_i$ 的含义相同，
但是 *case* 指令更加容易被最终的代码生成器探测到，
从而对这些指令进行特殊处理

过程调用的翻译

➤ 文法

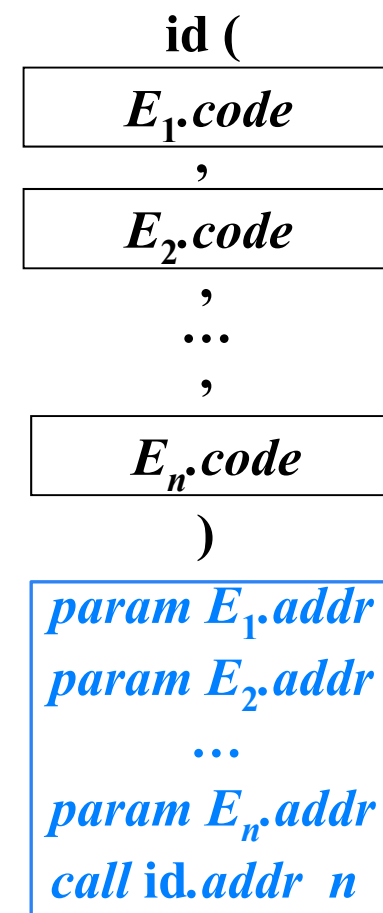
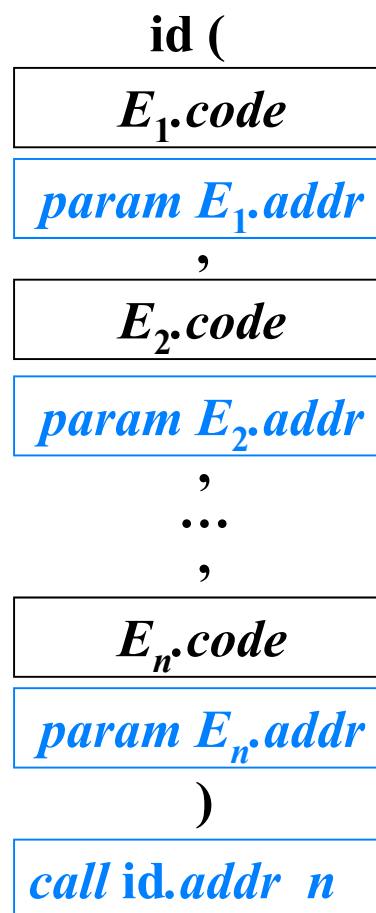
➤ $S \rightarrow \text{call id } (Elist)$

$Elist \rightarrow Elist, E$

$Elist \rightarrow E$

过程调用语句的代码结构

$\text{id}(E_1, E_2, \dots, E_n)$



过程调用语句的代码结构

$\text{id}(E_1, E_2, \dots, E_n)$

$\text{id} ($
 $E_1.\text{code}$
 $,$
 $E_2.\text{code}$
 $,$
 \dots
 $,$
 $E_n.\text{code}$
 $)$

需要一个队列 q 存放 $E_1.\text{addr}$ 、 $E_2.\text{addr}$ 、 \dots 、 $E_n.\text{addr}$ ，以生成

$\{$
 $\text{param } E_1.\text{addr}$
 $\text{param } E_2.\text{addr}$
 \dots
 $\text{param } E_n.\text{addr}$
 $\text{call id.addr } n$
 $\}$

过程调用语句的 *SDD*

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

```

{
     $n=0$ ;
    for  $q$  中的每个  $t$  do
    {
         $gen('param' t)$ ;
         $n = n+1$ ;
    }
     $gen('call' \text{id.addr}, n)$ ;
}

```

➤ $Elist \rightarrow E$

```

{
    将  $q$  初始化为只包含  $E.addr$ ;
}

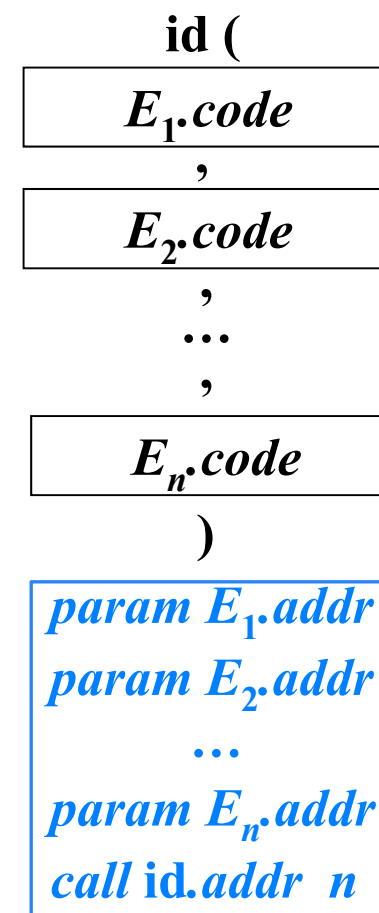
```

➤ $Elist \rightarrow Elist_1, E$

```

{
    将  $E.addr$  添加到  $q$  的队尾;
}

```



 例：翻译以下语句 $f(b * c - 1, x + y, x, y)$

$$t_1 = b * c$$

$$t_2 = t_1 - 1$$

$$t_3 = x + y$$

param t₂

param t₃

param x

param y

call f, 4