第六章 中间代码生成

布尔表达式的回填

哈尔滨工业大学 陈鄞

之前的做法 最后还要进行标号和地址的绑定

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

➢ $B \rightarrow E_1$ relop E_2

```
{
    B.truelist = makelist(nextquad);
    B.falselist = makelist(nextquad+1);
    gen('if' E₁.addr relop E₂.addr 'goto _');
    gen('goto _');
}
```

等待回填

布尔表达式的回填

- $B \rightarrow E_1$ relop E_2
- $B \rightarrow$ true

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

布尔表达式的回填

- $B \rightarrow E_1$ relop E_2
- $B \rightarrow$ true
- $B \rightarrow$ false

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

布尔表达式的回填

- $B \rightarrow E_1$ relop E_2
- $B \rightarrow$ true
- $B \rightarrow$ false
- $B \rightarrow (B_1)$

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

布尔表达式的回填

- $B \rightarrow E_1$ relop E_2
- $B \rightarrow$ true
- $B \rightarrow$ false
- $B \rightarrow (B_1)$
- $B \rightarrow$ not B_1

{

   $B.truelist = B_1.falselist;$

   $B.falselist = B_1.truelist;$

}

$B \rightarrow B_1$ or B_2

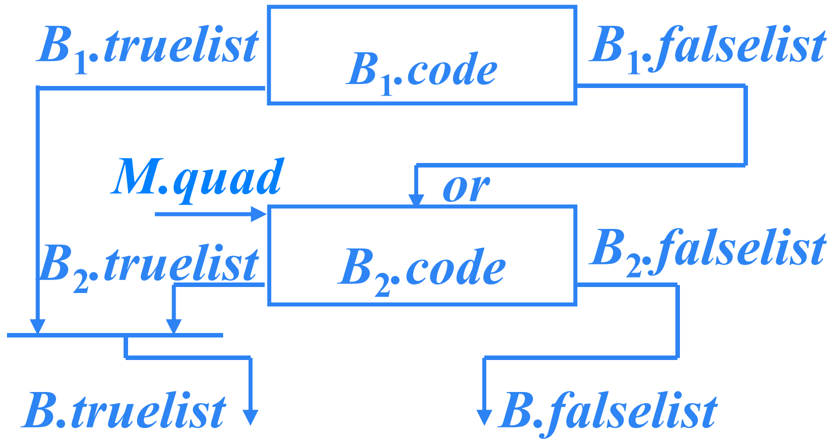
$B \rightarrow B_1$ or $M\ B_2$

```
{
    backpatch(B_1.falselist, M.quad );    回填
    B.truelist= merge(B_1.truelist,B_2.truelist );
    B.falselist = B_2.falselist ;
}
```

记录B2第一条指令的标号

$M \rightarrow \varepsilon$

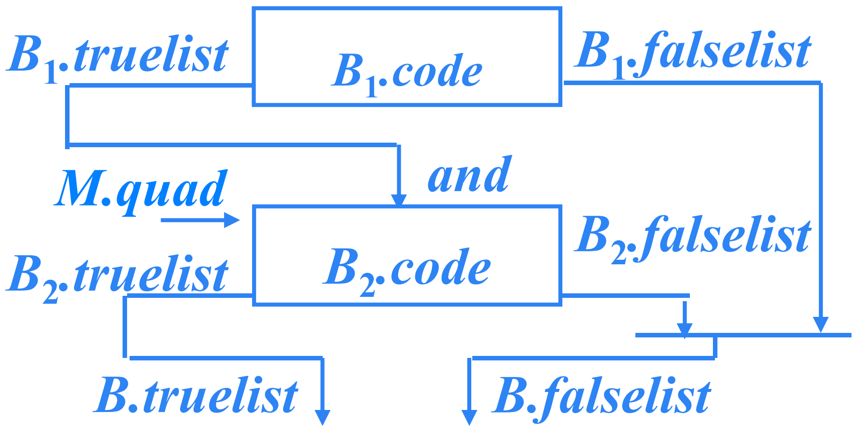
$\{\ M.quad = nextquad\ ;\ \}$



$B \rightarrow B_1$ **and** B_2

$B \rightarrow B_1$ **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

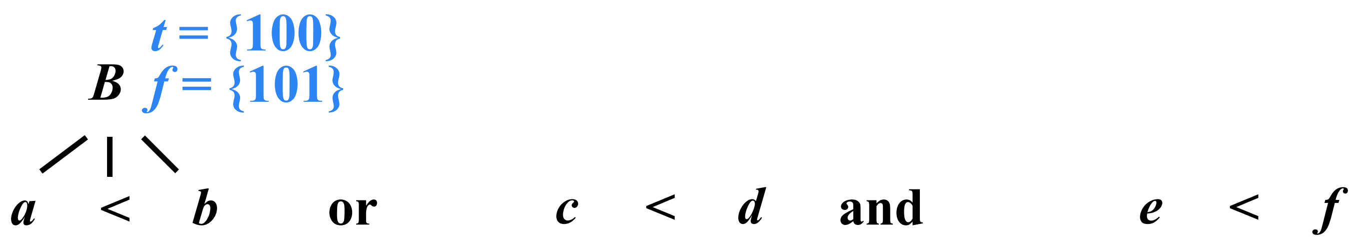
_表示待回填的目标标号

$B \rightarrow E_1$ **relop** E_2

```
{  B.truelist = makelist(nextquad);
   B.falselist = makelist(nextquad+1);
   gen('if' E₁.addr relop E₂.addr 'goto _');
   gen('goto _');
}
```

100: *if a<b goto* _

101: *goto* _



例

$B \rightarrow B_1$ or M B_2

{

$\quad$ *backpatch*(B_1.*falselist*, *M.quad*);

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

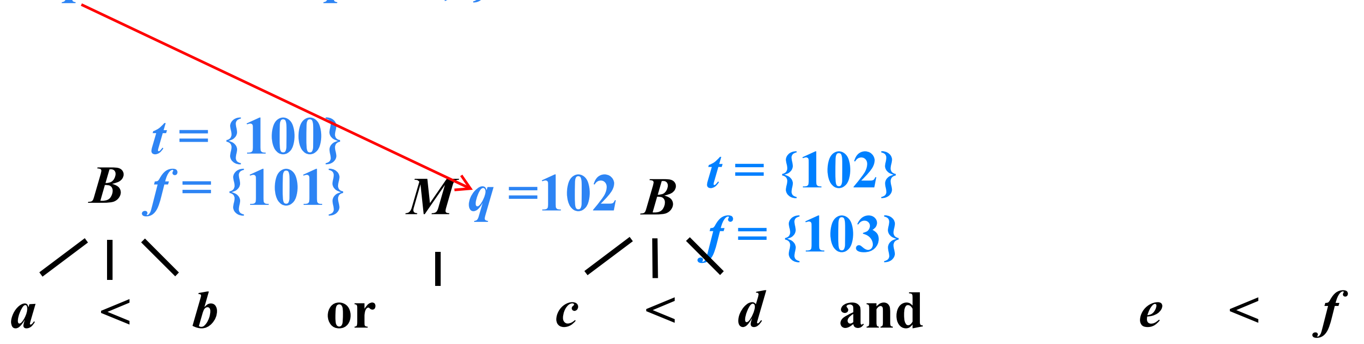
$\quad$ *B.falselist* = B_2.*falselist* ;

}

$M \rightarrow \varepsilon$

{ *M.quad* = *nextquad* ; }

```
100: if a<b goto _
101: goto _
102: if c<d goto _
103: goto _
```



例

$B \rightarrow B_1$ and M B_2

{

  backpatch(B_1.truelist, M.quad);

  B.truelist = B_2.truelist;

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

}

100: *if a<b goto* _

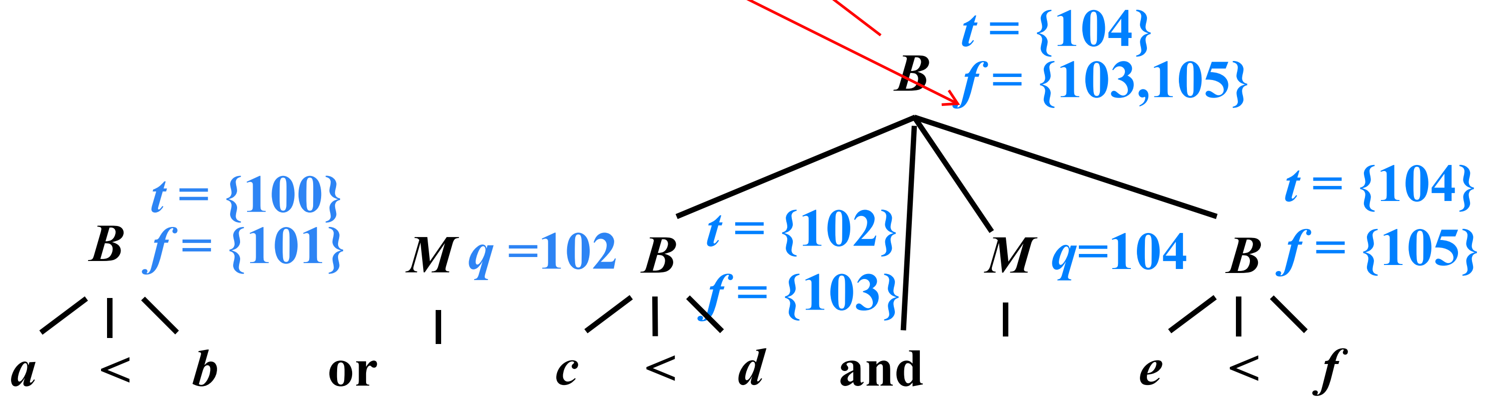
101: *goto* _

102: *if c<d goto* 104

103: *goto* _

104: *if e<f goto* _

105: *goto* _



例

$B \rightarrow B_1$ or M B_2

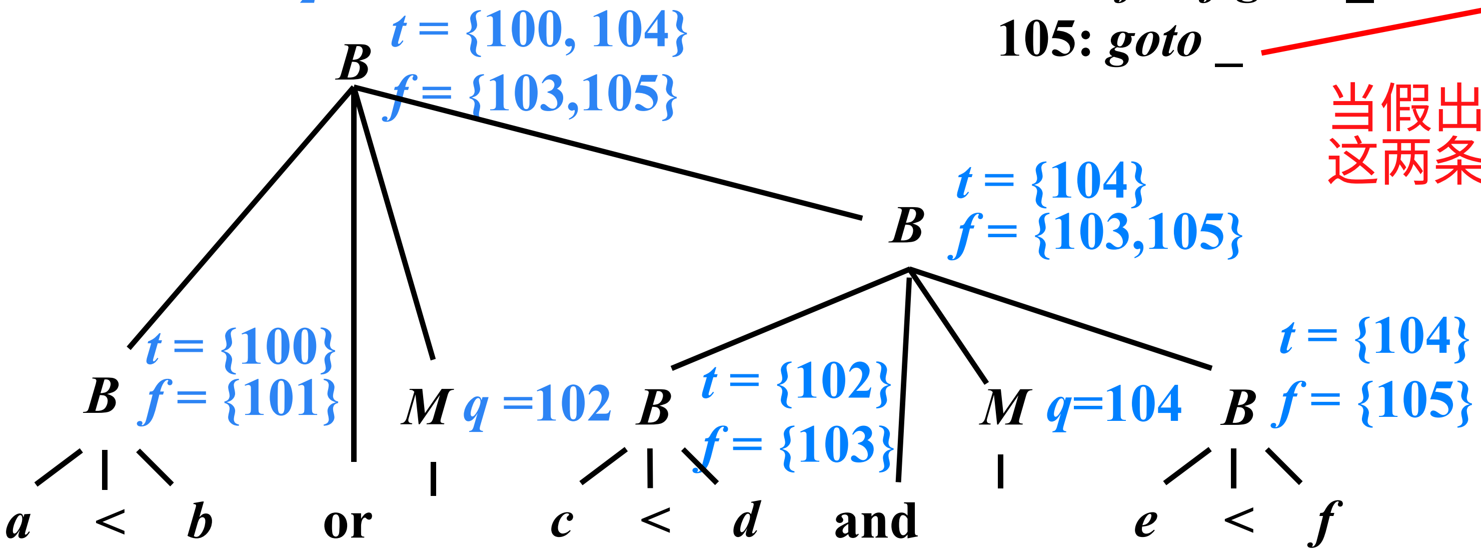
{

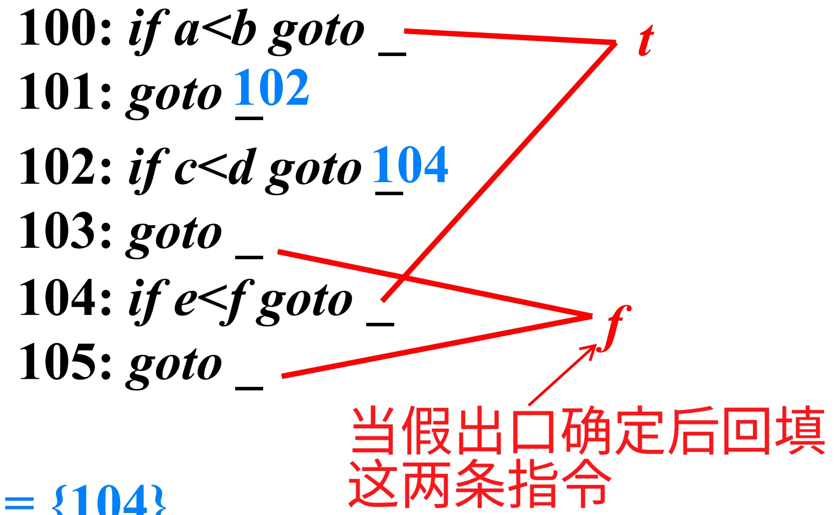
backpatch(B_1.falselist, M.quad);

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

B.falselist = B_2.falselist ;

}





100: *if a<b goto* _

101: *goto* 102

102: *if c<d goto* 104

103: *goto* _

104: *if e<f goto* _

105: *goto* _

当假出口确定后回填这两条指令

第六章 中间代码生成

布尔表达式的回填

哈尔滨工业大学 陈鄞

第六章 中间代码生成

控制流语句的回填

哈尔滨工业大学 陈鄞

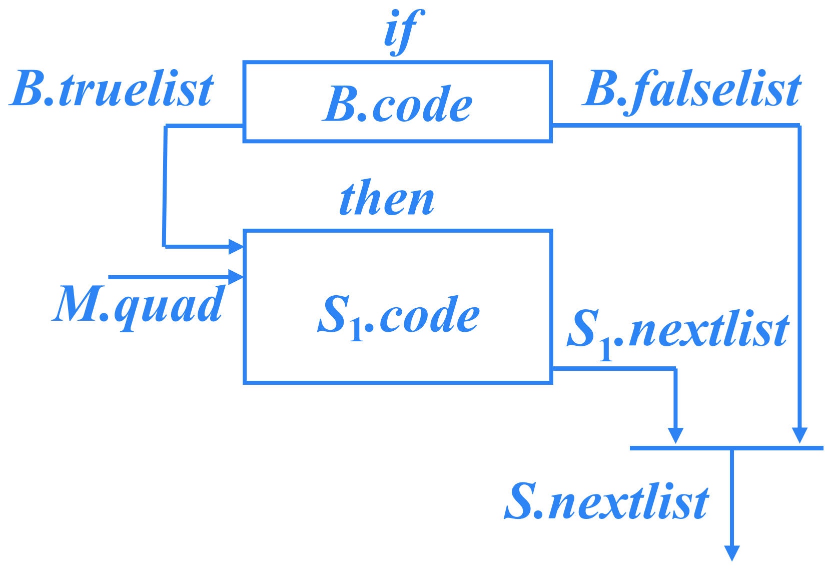
控制流语句的回填

- 文法
  - $S \rightarrow S_1 S_2$
  - $S \rightarrow \mathrm{id} = E \,;\ |\ L = E\,;$
  - $S \rightarrow \mathbf{if}\ B\ \mathbf{then}\ S_1$
    $|\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2$
    $|\ \mathbf{while}\ B\ \mathbf{do}\ S_1$
- 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

$S \rightarrow$ if B then S_1

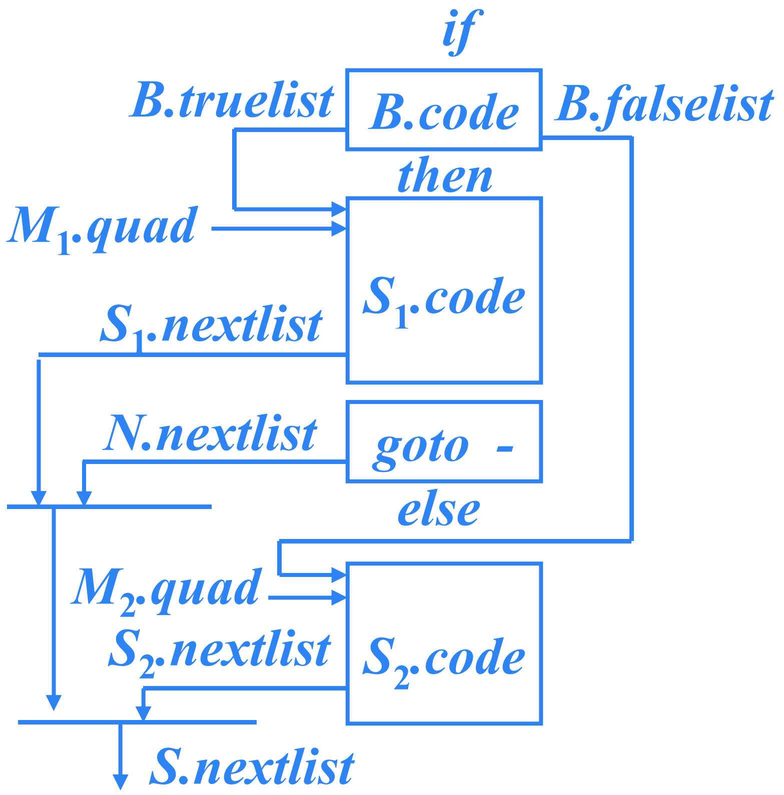
$S \rightarrow$ if B then M S_1

```
{
    backpatch(B.truelist, M.quad);
    S.nextlist=merge(B.falselist, S_1.nextlist);
}
```



$S \rightarrow$ if B then S_1 else S_2

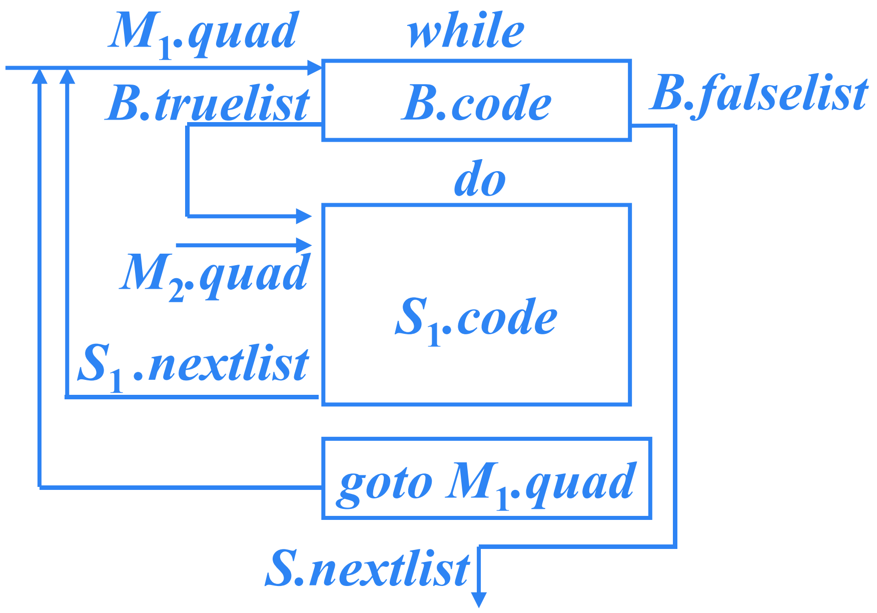
```
S → if B then M1 S1 N else M2 S2
{
    backpatch(B.truelist, M1.quad);
    backpatch(B.falselist, M2.quad);
    S.nextlist = merge( merge(S1.nextlist,
    N.nextlist), S2.nextlist );
}
N → ε
{ N.nextlist = makelist(nextquad);
    gen('goto _');
}
```



S $\rightarrow$ while B do S_1

$S \rightarrow$ while M_1 B 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);
}
```



$S \rightarrow S_1\ S_2$

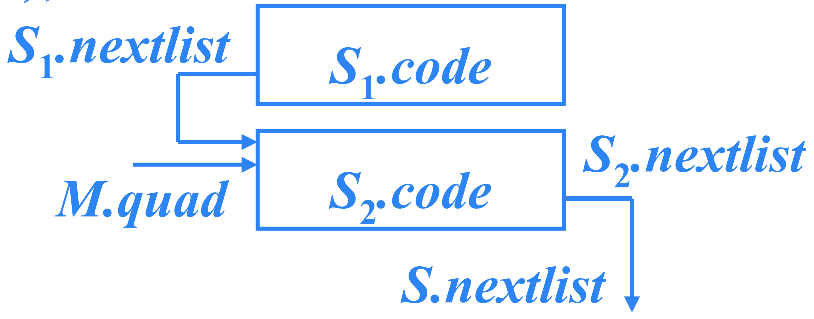
$S \rightarrow S_1\ M\ S_2$

{

backpatch(S_1.nextlist, M.quad);

S.nextlist = S_2.nextlist ;

}



$S \rightarrow \mathbf{id} = E\ ;\ |\ L = E\ ;$

$S \rightarrow \mathbf{id} = E\ ;\ |\ L = E\ ; \{\ S.nextlist = null;\ \}$

因为赋值语句不包含跳转指令

例

```
while  $a < b$  do

   if  $c < 5$  then

         while $x > y$ do $z = x + 1$;
   else
         $x = y$;
```

$S \rightarrow$ **while** M_1 B **do** M_2 S_1

{ *S.nextlist = B.falselist;*

backpatch(S_1.nextlist, M_1.quad);

backpatch(B.truelist, M_2.quad);

gen('goto' M_1.quad); }

$S \rightarrow$ **if** B **then** M_1 S_1 N **else** M_2 S_2

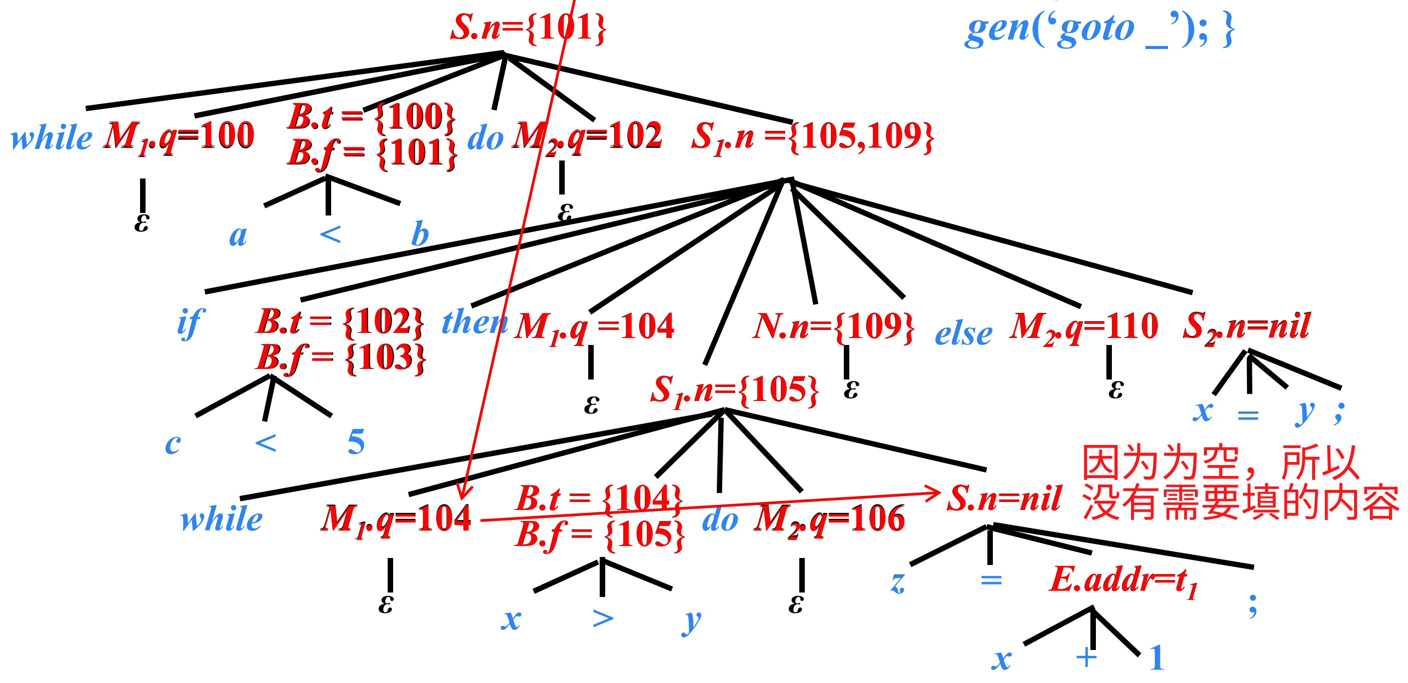
{ *S.nextlist = merge(merge(S_1.nextlist, N.nextlist), S_2.nextlist);*

backpatch(B.truelist, M_1.quad);

backpatch(B.falselist, M_2.quad); }

$N \rightarrow \varepsilon$ { *N.nextlist = makelist(nextquad);*

gen('goto _'); }



100: *if a < b goto* 102

101: *goto* _ s的下一条指令确定后再填入

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:

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	100:	($j<$, a , b , 102)
101:	*goto* _	101:	(j , - , - , -)
102:	*if* $c < 5$ *goto* 104	102:	($j<$, c , 5 , 104)
103:	*goto* 110	103:	(j , - , - , 110)
104:	*if* $x > y$ *goto* 106	104:	($j>$, x , y , 106)
105:	*goto* 100	105:	(j , - , - , 100)
106:	$t_1 = x + 1$	106:	(+ , x , 1 , t_1)
107:	$z = t_1$	107:	(= , t_1 , - , z)
108:	*goto* 104	108:	(j , - , - , 104)
109:	*goto* 100	109:	(j , - , - , 100)
110:	$x = y$	110:	(= , y , - , x)
111:	*goto* 100	111:	(j , - , - , 100)
112:		112:	

第六章 中间代码生成

控制流语句的回填

哈尔滨工业大学 陈鄞

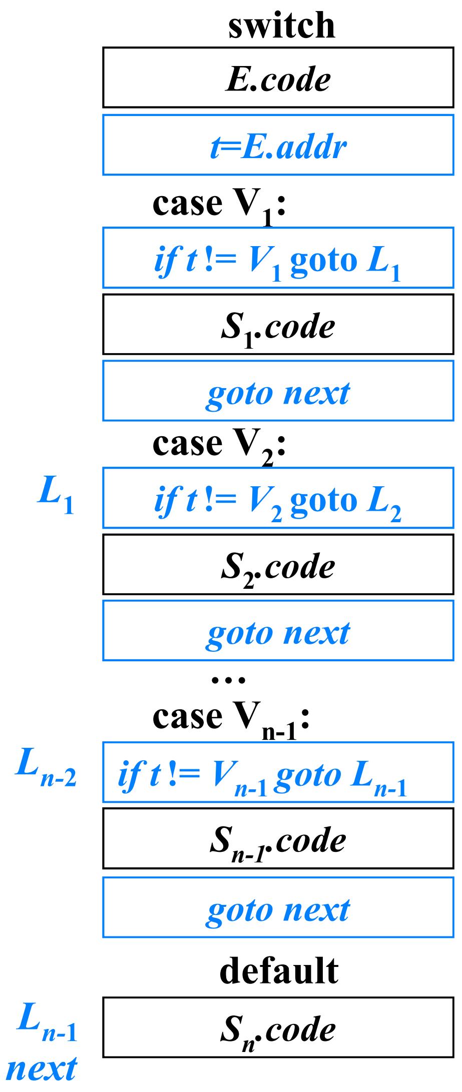
第六章 中间代码生成

switch语句的翻译

哈尔滨工业大学 陈鄞

*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

表达式E的值赋给 t

switch E { $t = newtemp(); gen(\ t\ ‘=’\ E.addr\ );$ }

case V_1:{ $L_1 = newlabel();$

  $gen(‘if’\ t\ ‘!=’\ V_1\ ‘goto’\ L_1\ );$ }

S_1{ $next = newlabel();\quad gen(‘goto’\ next);$ }

case V_2:{ $label(L_1);\ L_2=newlabel();$

  $gen(‘if’\ t\ ‘!=\ ’\ V_2\ ‘goto’\ L_2\ );$ }

S_2{ $gen(‘goto’\ next);$ }

. . .

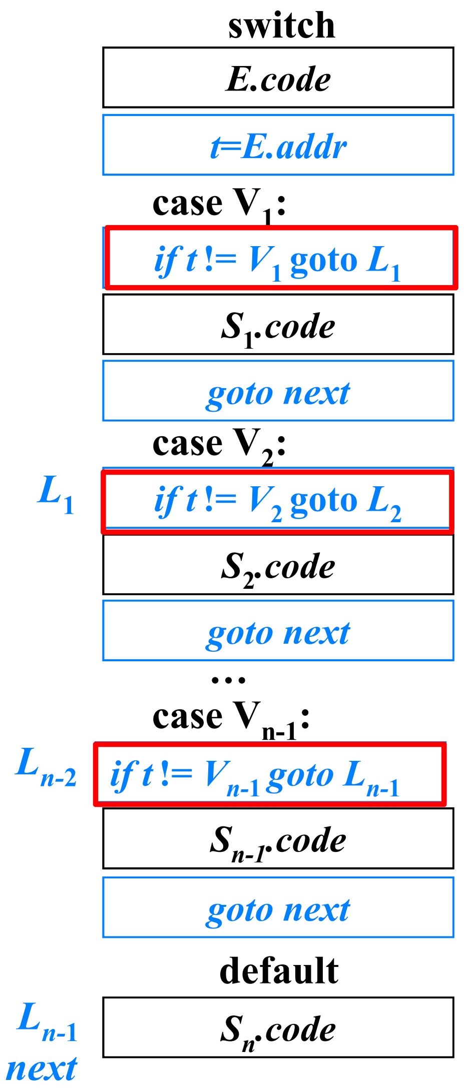
case V_{n-1}:{ $label(L_{n-2});\ L_{n-1}=newlabel();$

  $gen(‘if’\ t\ ‘!=’\ V_{n-1}\ ‘goto’\ L_{n-1}\ );$}

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

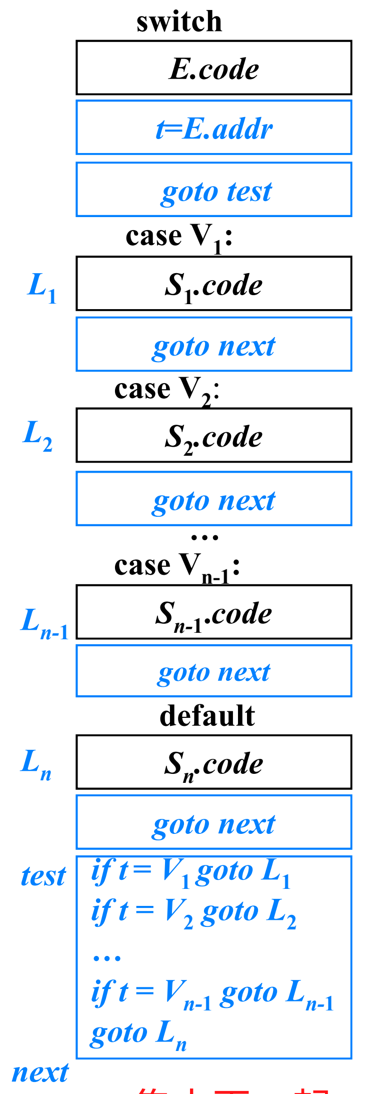
    . . .

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

    default: S_n

  end

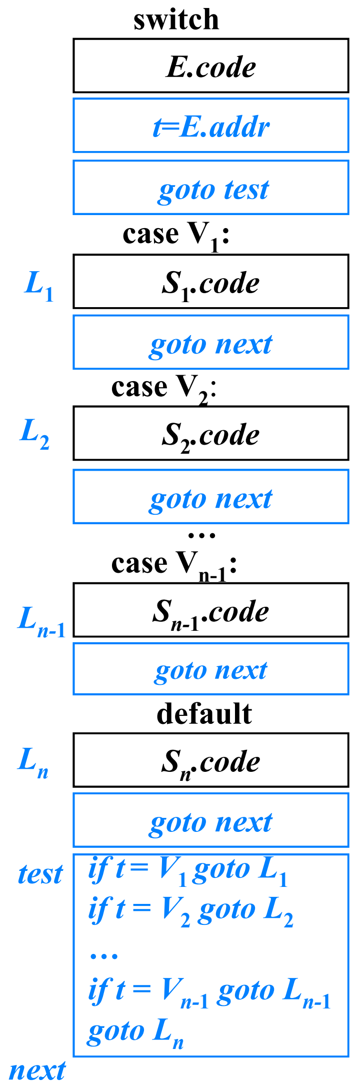
在代码生成阶段，根据分支的个数以及这些值是否在一个较小的范围内，这种条件跳转指令序列可以被翻译成最高效的n路分支



集中再一起，提高效率

*switch*语句的另一种翻译

```
switch E { t = newtemp(); gen(t '=' E.addr);
            test = newlabel(); gen('goto' test); }
case V_1 : { L_1 = newlabel(); label(L_1); map(V_1, L_1); }
       S_1   { next = newlabel();  gen('goto' next); }
case V_2 : { L_2 = newlabel(); label(L_2); map(V_2, L_2); }
       S_2   { gen('goto' next); }
            . . .
case V_n-1 :{ L_n-1 = newlabel(); label(L_n-1); map(V_n-1, L_n-1); }
       S_n-1 { gen('goto' next); }
default :  { L_n = newlabel(); label(L_n); }
       S_n   { gen('goto' next);
               label(test);
               gen('if' t '=' V_1 'goto' L_1);
               gen('if' t '=' V_2 'goto' L_2);
                      . . .
               gen('if' t '=' V_n-1 'goto' L_n-1);
               gen('goto' L_n);
               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 ***case t*** V_i L_i 和 ***if t*** $= V_i$ ***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

第六章 中间代码生成

switch语句的翻译

哈尔滨工业大学 陈鄞

第六章 中间代码生成

过程调用语句的翻译

哈尔滨工业大学 陈鄞

过程调用的翻译

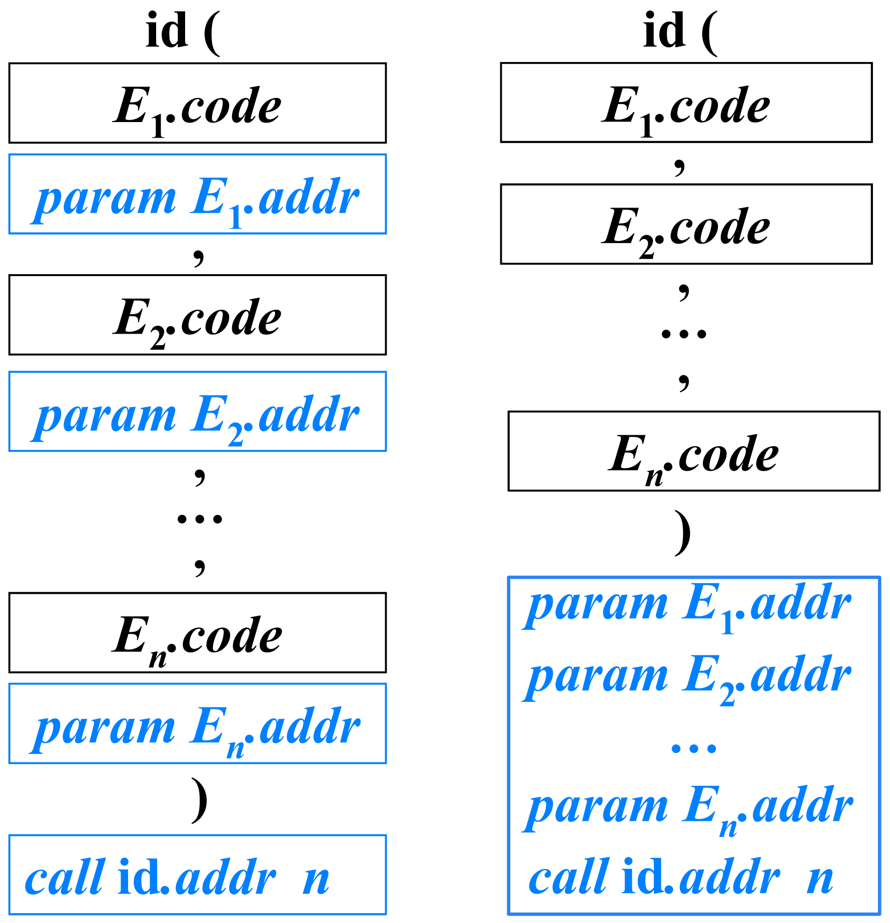
- 文法
  - $S \rightarrow$ **call id** ($Elist$)

    $Elist \rightarrow Elist, E$

    $Elist \rightarrow E$

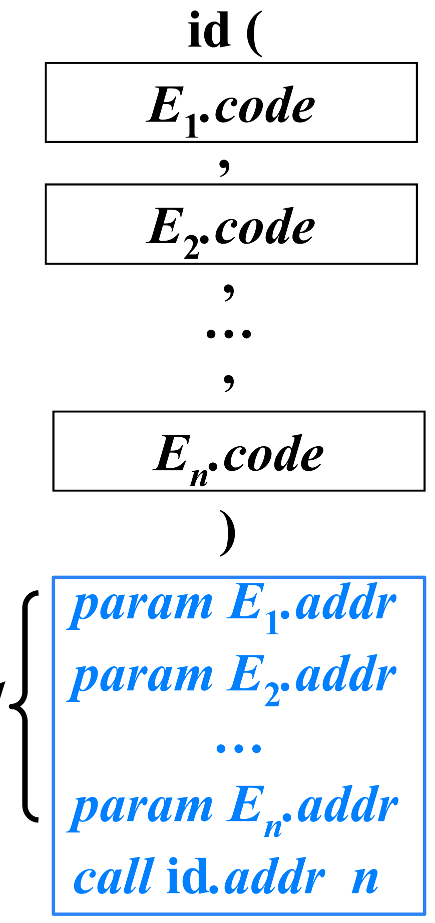
过程调用语句的代码结构

id(E_1, E_2, … , E_n)



过程调用语句的代码结构

id(E_1, E_2, … , E_n)



需要一个队列q存放E_1.addr 、E_2.addr、…、E_n.addr，以生成

过程调用语句的*SDD*

- $S \rightarrow$ call id (*Elist*)

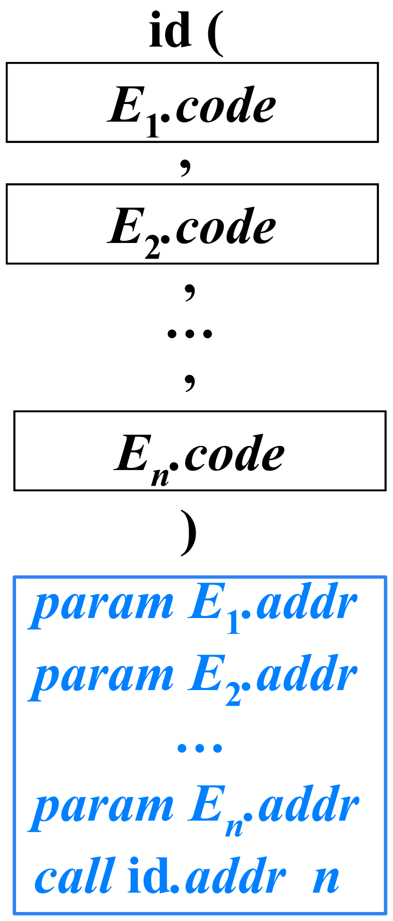
```
{     n=0;
      for q中的每个t do
      {      gen('param' t );
             n = n+1;
      }
      gen('call' id.addr ',' n);
}
```

- *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;    }
```



```
param E_1.addr
param E_2.addr
…
param E_n.addr
call id.addr  n
```

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 t_2

param t_3

param x

param y

call f, 4

第六章 中间代码生成

过程调用语句的翻译

哈尔滨工业大学 陈鄞