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

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' E_1.addr relop E_2.addr 'goto _');

gen('goto _');

}

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*);

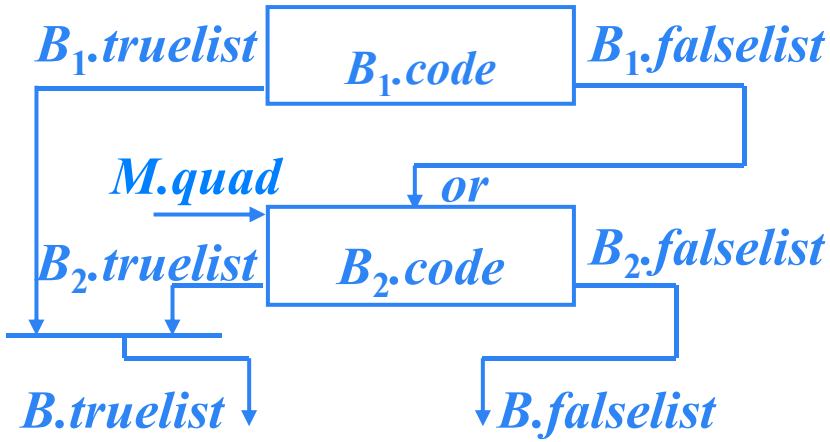
B.truelist= *merge*(B_1.*truelist*,B_2.*truelist*);

B.falselist = B_2.*falselist* ;

}

$M \rightarrow \varepsilon$

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



$B \rightarrow B_1$ and B_2

$B \rightarrow B_1$ and M B_2

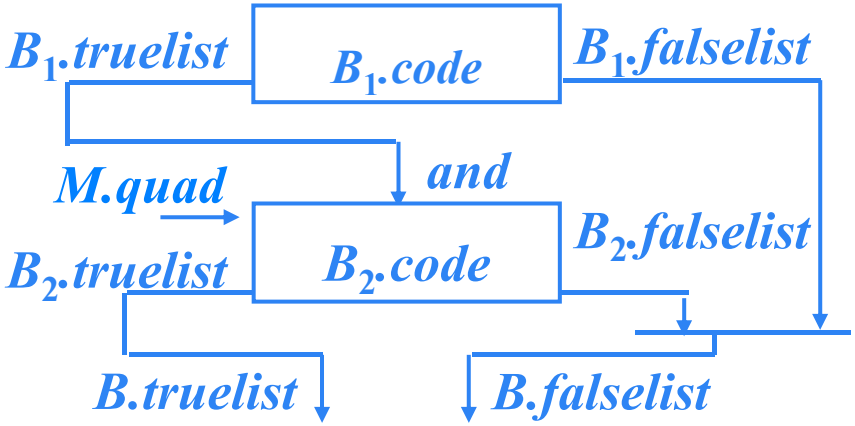
{

backpatch(B_1.truelist, M.quad);

B.truelist = B_2.truelist;

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

}

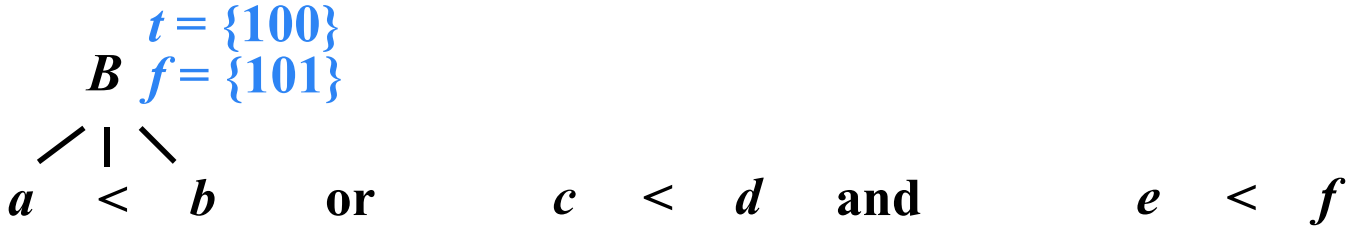


例

$B \rightarrow E_1$ relop E_2

{ *B.truelist = makelist(nextquad);*
B.falselist = makelist(nextquad+1);
gen('if' E_1.addr relop E_2.addr 'goto _');
gen('goto _');
}

100: *if a<b goto _*
101: *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* ;

}

$M \rightarrow \varepsilon$

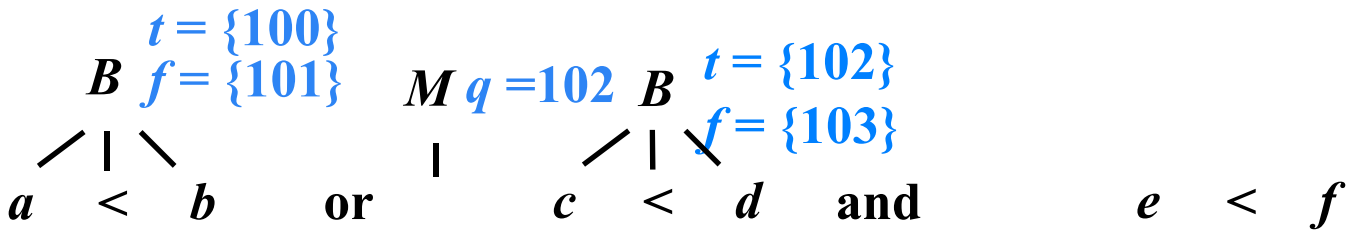
{ *M.quad* = *nextquad* ; }

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* _

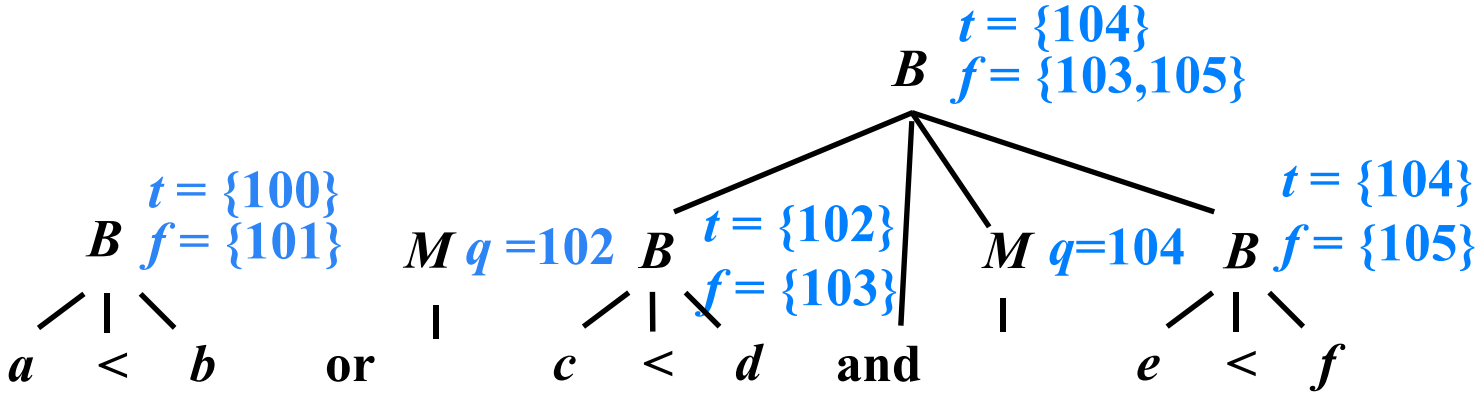
101: *goto* _

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

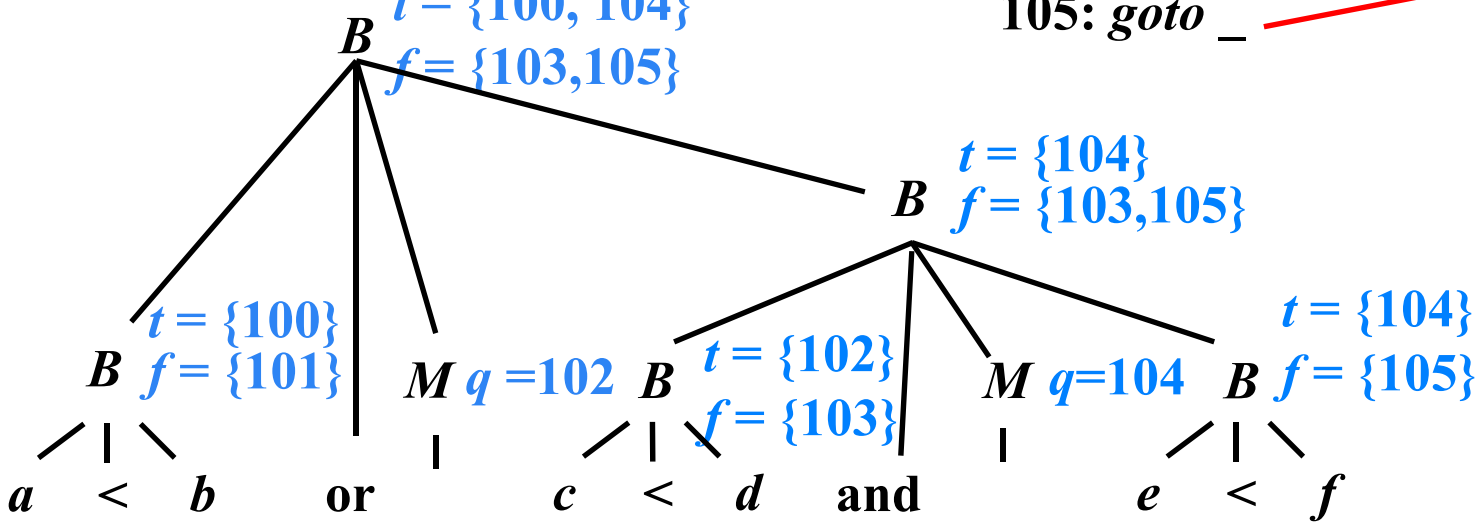
103: *goto* _

104: *if e<f goto* _

105: *goto* _

t

f



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
  - $S \rightarrow S_1 S_2$
  - $S \rightarrow \text{id} = E\ ; \mid L = E\ ;$
  - $S \rightarrow \text{if } B \text{ then } S_1$
    $\mid \text{if } B \text{ then } S_1 \text{ else } S_2$
    $\mid \text{while } B \text{ do } S_1$
- 综合属性
  - *S.next1ist*：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

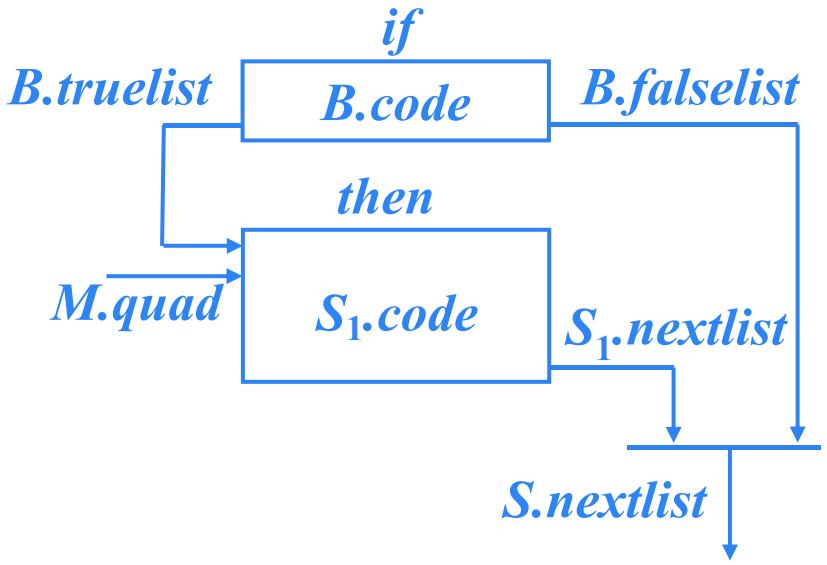
$S \rightarrow$ if B then M S_1

{

backpatch(B.truelist, M.quad);

S.nextlist=merge(B.falselist, S_1.nextlist);

}



$S \rightarrow$ if B then S_1 else S_2

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

{

backpatch(B.truelist, M_1.quad);

backpatch(B.falselist, M_2.quad);

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

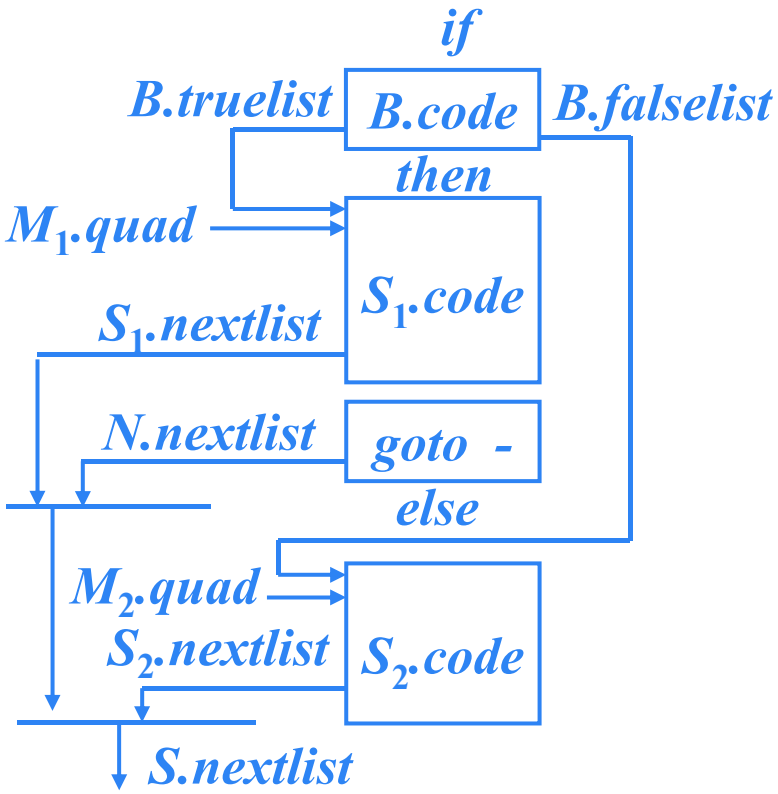
}

$N \rightarrow \varepsilon$

{ *N.nextlist = makelist(nextquad);*

gen('goto _');

}



S → while *B* do S_1

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

{

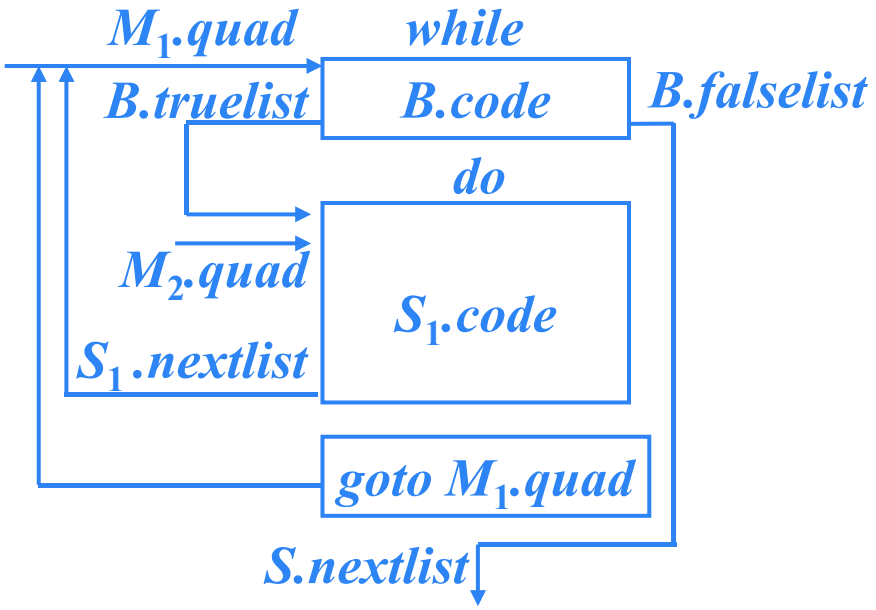
backpatch(S_1.*nextlist*, M_1.*quad*);

backpatch(*B.truelist*, M_2.*quad*);

S.nextlist = *B.falselist*;

gen('*goto*' M_1.*quad*);

}



$S \rightarrow S_1\ S_2$

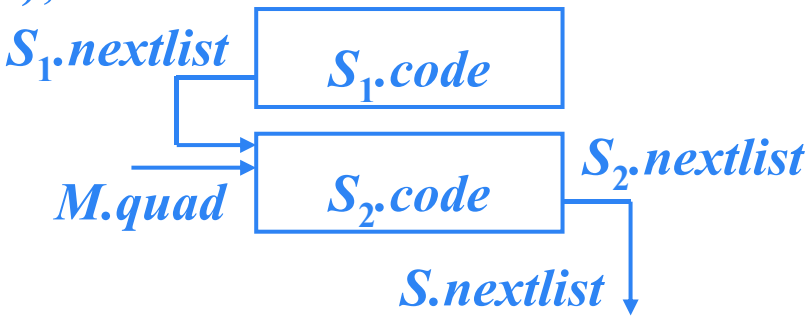
$S \rightarrow S_1\ M\ S_2$

{

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

S.*nextlist* = S_2.*nextlist* ;

}



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

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

例

```
while  a < b  do

    if  c < 5  then

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

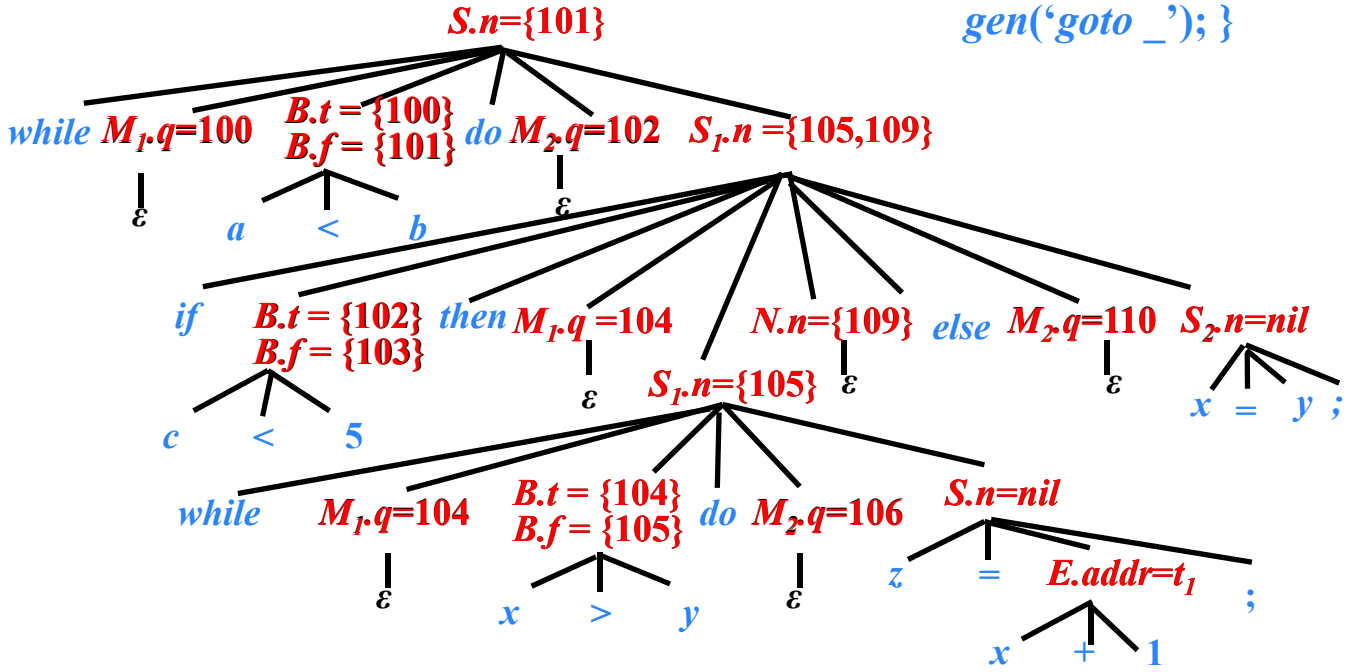
$S \rightarrow$ while M_1 B do M_2 S_1

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

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

{ $S.nextlist = merge(merge(S_1.nextlist, N.nextlist), S_2.nextlist)$;
$backpatch(B.truelist, M_1.quad)$;
$backpatch(B.falselist, M_2.quad)$; }

$N \rightarrow \varepsilon$ { $N.nextlist = makelist(nextquad)$;
$gen(\text{'goto}\ \_\text{'})$; }



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:

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

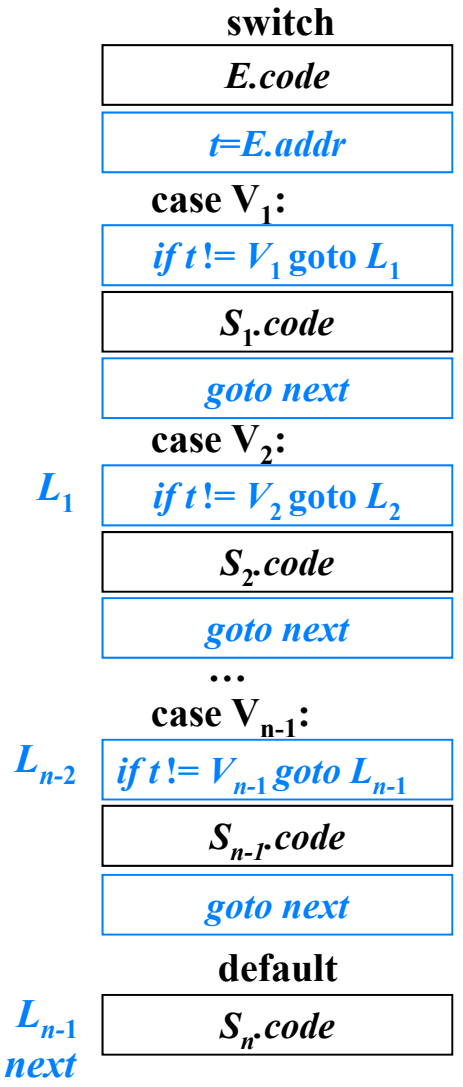
第六章 中间代码生成

switch语句的翻译

哈尔滨工业大学 陈鄞

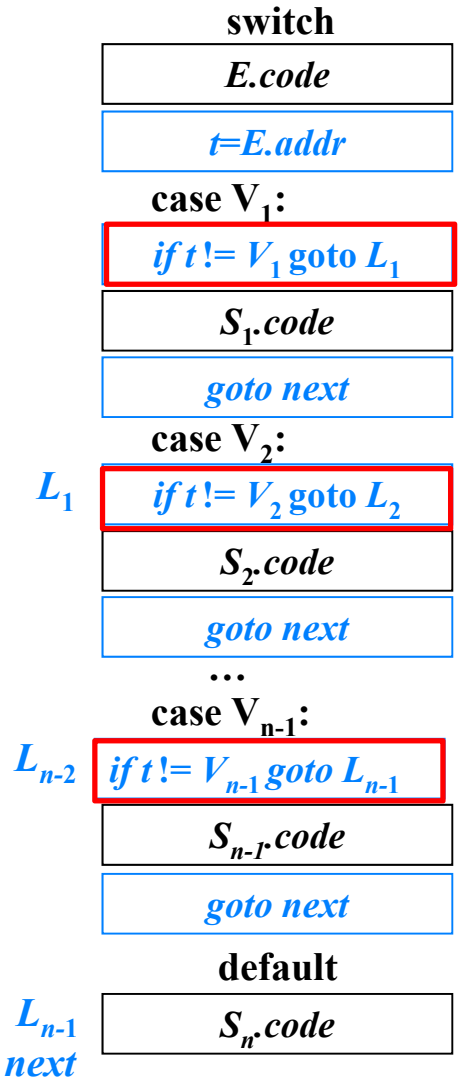
*switch*语句的翻译

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

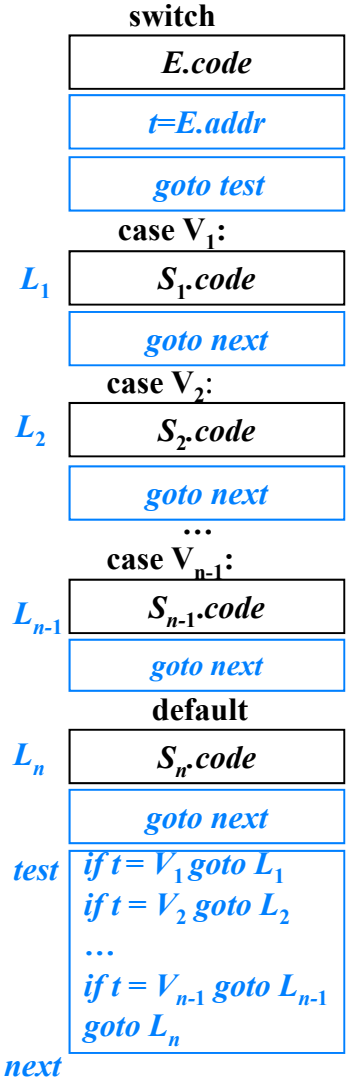
switch *E* { *t* = *newtemp*(); *gen*(*t* '=' *E.addr*); }
case V_1:{ L_1= *newlabel*();
gen('*if*' *t* '!=' V_1 '*goto*' L_1); }
S_1 { *next* = *newlabel*(); *gen*('*goto*' *next*); }
case V_2:{ *label*(L_1); L_2=*newlabel*();
gen('*if*' *t* '!= ' V_2 '*goto*' L_2); }
S_2 { *gen*('*goto*' *next*); }
. . .
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

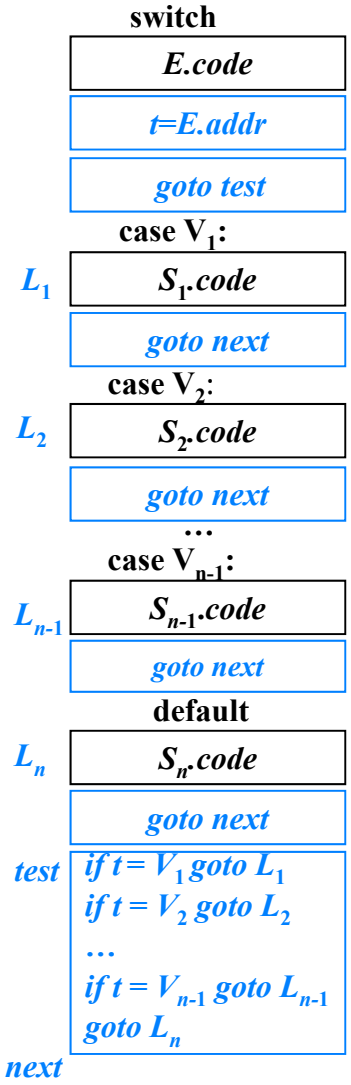
switch *E*
  begin
    case V_1: S_1
    case V_2: S_2
    . . .
    case V_{n-1}: S_{n-1}
    default: S_n
  end

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 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 = V1 goto L1
        if t = V2 goto L2
        . . .
        if t = Vn-1 goto Ln-1
        goto Ln
next :
```

```
test :  case t  V1    L1
        case t  V2    L2
           . . .
        case t  Vn-1  Ln-1
        case t  t     Ln
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

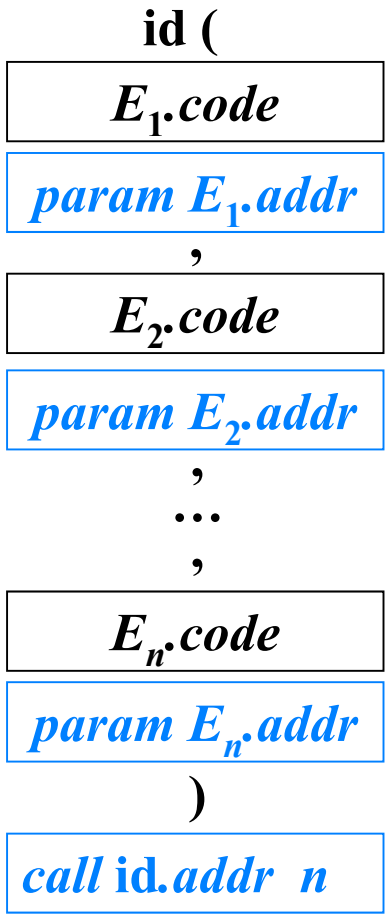
- 文法
  - $S \rightarrow$ call id (*Elist*)

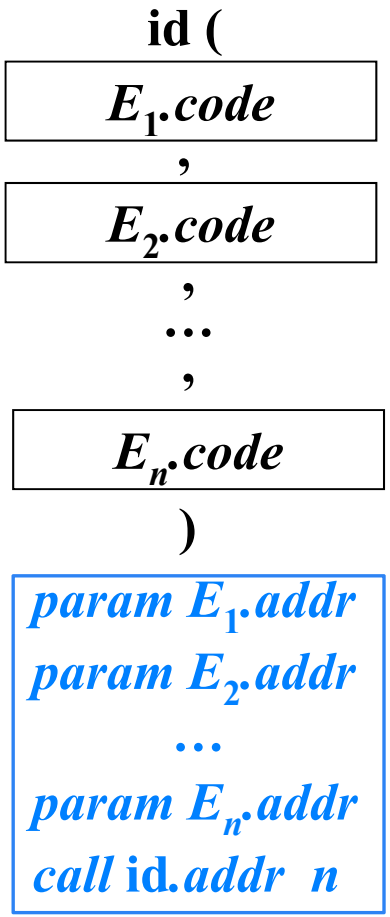
    $Elist \rightarrow Elist, E$

    $Elist \rightarrow E$

过程调用语句的代码结构

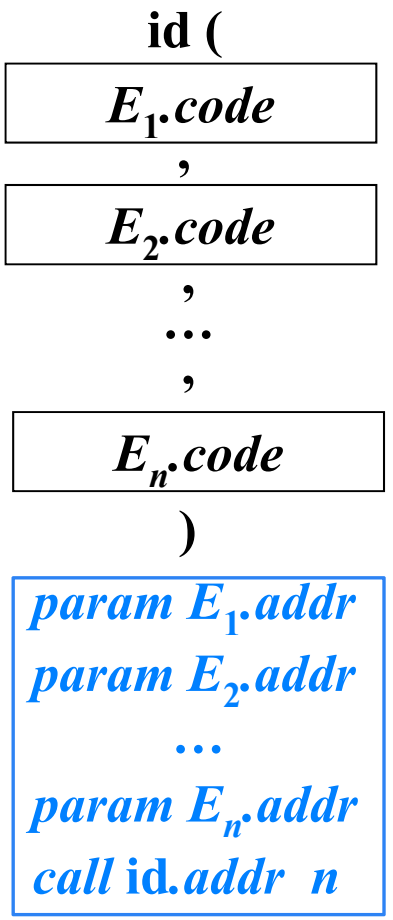
id(E_1, E_2, … , E_n)





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; }

id (
E_1.*code*
,
E_2.*code*
,
…
,
E_n.*code*
)

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