

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

\succ merge(p_1, p_2)

 \triangleright 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

\succ 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

```
 False E_1 relop E_2 
 B.true list = make list (next quad); 
 B.false list = make list (next quad + 1); 
 gen('if' E_1.addr relop E_2.addr 'goto _'); 
 gen('goto _'); 
}
```

```
 Partial B \rightarrow E_1 \text{ relop } E_2 
 Partial B \rightarrow \text{true} 
 B. \textit{truelist} = \textit{makelist(nextquad);} 
 gen('goto _'); 
}
```

```
 PB \rightarrow E_1 \text{ relop } E_2 
 PB \rightarrow \text{true} 
 PB \rightarrow \text{false} 
 \{ B. falselist = makelist(nextquad); \\ gen(`goto \_'); 
 \}
```

```
\triangleright B \rightarrow E_1 \text{ relop } E_2
> B \rightarrow \text{true}
\triangleright B \rightarrow \text{false}
\triangleright B \rightarrow (B_1)
\triangleright B \rightarrow \text{not } B_1
    B.truelist = B_1.falselist;
    B.falselist = B_1.truelist;
```

```
B \rightarrow B_1 \text{ or } B_2
B \rightarrow B_1 or MB_2
   backpatch(B_1.falselist, M.quad);
   B.truelist = merge(B_1.truelist, B_2.truelist);
   B.falselist = B_{2}.falselist;
                                             B<sub>1</sub>.truelist
                                                                           B_1-falselist
                                                              B_1.code
M \rightarrow \varepsilon
                                                 M.quad
                                                                   or
\{ M.quad = nextquad; \}
                                                                           B<sub>2</sub>.falselist
                                             B<sub>2</sub>.tru<u>elist</u>
                                                             B_2.code
                                                                     ↓B.falselist
                                            B.truelist
```

```
B \rightarrow B_1 and B_2
B \rightarrow B_1 and MB_2
   backpatch(B_1.truelist, M.quad);
   B.truelist = B_{\gamma}.truelist;
  B.falselist = merge(B_1.falselist, B_2.falselist);
                                                B_1-falselist
                    B_1.truelist
                                    B_1.code
                                         and
                      M.quad
                                                B<sub>2</sub>.falselist
                                   B_2.code
                    B<sub>2</sub>.truelist
                        B.truelist
                                           B.falselist
```

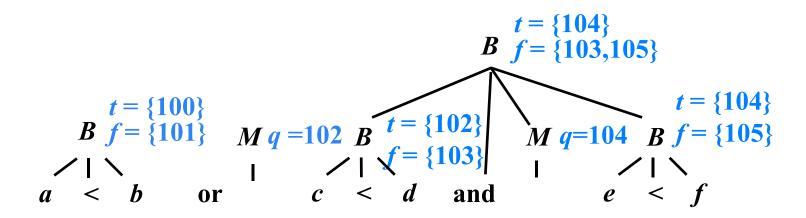
```
B \rightarrow E_1 \text{ relop } E_2 100: if a < b goto _ 101: go
```

$$\begin{array}{c}
t = \{100\} \\
B \ f = \{101\}
\end{array}$$

$$\nearrow \mid \searrow \\
a < b \quad \text{or} \quad c < d \quad \text{and} \quad e < f$$

```
100: if a<b goto
B \rightarrow B_1 \text{ or } M B_2
                                               101: goto _
                                               102: if c<d goto
  backpatch(B_1.falselist, M.quad);
                                               103: goto _
  B.truelist = merge(B_1.truelist, B_2.truelist);
  B.falselist = B_{2}.falselist;
M \rightarrow \varepsilon
\{ M.quad = nextquad; \}
   e < f
```

```
B \rightarrow B_1 \text{ and } MB_2 \\ \{ & 100: if a < b \text{ goto} \\ 101: goto \\ \_ \\ backpatch(B_1.truelist, M.quad); \\ B.truelist = B_2.truelist; \\ B.falselist = merge(B_1.falselist, B_2.falselist); \\ 104: if e < f \text{ goto} \\ \_ \\ 105: goto \\ \_
```



```
100: if a<b goto _-
B \rightarrow B_1 or MB_2
                                                      101: goto 102
                                                      102: if c<d goto 104
  backpatch(B_1.falselist, M.quad);
  B.truelist = merge(B_1.truelist, B_2.truelist);
                                                    103: goto _
  B.falselist = B_{2}.falselist;
                                                      104: if e<f goto 2
                    B^{t} = \{100, 104\}
                                                      105: goto _ _
                                                t = \{104\}
B f = \{103,105\}
                                                                    t = \{104\}
       B f = \{101\} M q = 102 B t = \{102\}
                                                   M q = 104 B f = \{105\}
                                             and
                    or
```





控制流语句的回填

- 〉文法
 - $>S \rightarrow S_1 S_2$
 - $\triangleright S \rightarrow id = E ; | L = E ;$
 - $>S \rightarrow \text{if } B \text{ then } S_1$ | if $B \text{ then } S_1 \text{ else } S_2$

| while B do S_1

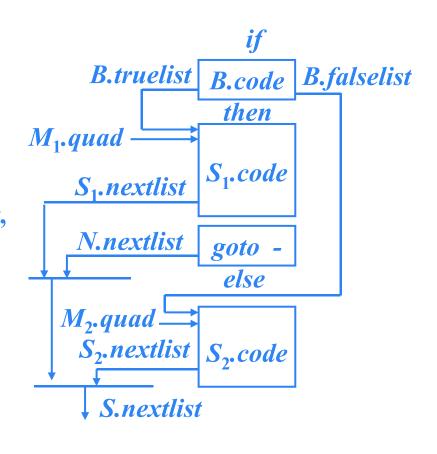
- 户综合属性
 - ▶ S.nextlist: 指向一个包含跳转指令的列表,这些指令最终获得的目标标号就是按照运行顺序紧跟在S代码之后的指令的标号

$S \rightarrow \text{if } B \text{ then } S_1$ $S \rightarrow \text{if } B \text{ then } M S_1$ backpatch(B.truelist, M.quad); $S.nextlist=merge(B.falselist, S_1.nextlist);$ **B.**truelist **B.**falselist **B.**code then M.quad S_1 .code S₁.nextlist

S.nextlist

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

```
S \rightarrow \text{if } B \text{ then } M_1 S_1 N \text{ 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');
```



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

```
S \rightarrow \text{while } M_1 B \text{ do } M_2 S_1
                                               M_1.quad
                                                               while
                                                                           B. falselist
   backpatch(S_1.nextlist, M_1.quad);
                                               B.truelist
                                                              B.code
   backpatch(B.truelist, M2.quad);
                                                                 do
   S.nextlist = B.falselist;
                                                M_2.quad
                                                              S_1.code
  gen('goto' M_1.quad);
                                             S_1.nextlist
                                                           goto M_1.quad
                                                       S.nextlist
```

```
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_1.nextlist

S_1.code

}

S_2.code

S_2.nextlist
```

 $S \rightarrow id = E ; | L = E ;$

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

while a < b do

if c < 5 then

while x > y do z = x + 1;

else

x = y;

```
S \rightarrow \text{if } B \text{ then } M_1 S_1 N \text{ else } M_2 S_2
                S \rightarrow \text{while } M_1 B \text{ do } M_2 S_1
                                                                     S.nextlist = merge(merge(S_1.nextlist, N.nextlist),
                    S.nextlist = B.falselist;
                                                                     S_2.nextlist);
                    backpatch(S_1.nextlist, M_1.quad);
                                                                     backpatch(B.truelist, M_1.quad);
                    backpatch(B.truelist, M2.quad);
                                                                      backpatch(B.falselist, M<sub>2</sub>.quad); }
                    gen(`goto` M_1.quad); \}
                                                                 N \rightarrow \varepsilon { N.nextlist = makelist(nextquad);
                                S.n = \{101\}
                                                                     gen('goto'); }
                                                                                                           100: if a < b goto 102
                      t = \{100\}

f = \{101\} do M_2 q = 102 S_1 n = \{105, 109\}
                                                                                                           101: goto
while M_1.q=100
                                                                                                           102: if c < 5 goto 104
                                                                                                           103: goto 110
                                                                                                           104: if x > y goto 106
                 B.t = \{102\} then M_1.q = 104
                                                                                                           105: goto 100
                                                    N.n=\{109\} else M_2.q=110 S_2.n=nil
                  B.f = \{103\}
                                                                                                           106: t_1 = x + 1
                                              S_{I}.n=\{105\} \varepsilon
                                                                                                           107: z = t_1
                                                                                                           108: goto 104
                                    B.t = \{104\}

B.f = \{105\} do M_2.q = 106
                                                                    S.n=nil
                                                                                                            109: goto 100
                      M_1.q = 104
                                                                                                           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 $a < b ext{ do if } c < 5 ext{ then while } x > y ext{ do } z = x + 1; ext{ else } x = y;$

101:
$$goto_{-}$$
 101: $(j, -, -, -)$

102: if
$$c < 5$$
 goto 104 102: $(j < c, 5, 104)$

103: goto 110
$$(j, -, -, 110)$$

104: if
$$x > y$$
 goto 106 104: $(j>, x, y, 106)$

106:
$$t_1 = x + 1$$
 106: $(+, x, 1, t_1)$

107:
$$z = t_1$$
 107: $(=, t_1, -, z_1)$

108: goto 104 108:
$$(j, -, -, 104)$$

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





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
```

```
switch
                E.code
               t=E.addr
          case V_1:
           if t! = V_1 goto L_1
                S_1.code
              goto next
          case V<sub>2</sub>:
           if t = V_2 goto L_2
                S_2.code
               goto next
          case V_{n-1}:
       | if t! = V_{n-1} goto L_{n-1}
               S_{n-1}.code
               goto next
                default
L_{n-1}
                S_n.code
next
```

switch语句的翻译

```
case V_1:
switch E \{ t = newtemp(); gen(t'='E.addr); \}
                                                                        if t!=V_1 goto L_1
case V_1:\{L_1 = newlabel();
                                                                            S_1.code
              gen('if' t '!=' V_1 'goto' L_1); 
                                                                           goto next
S_1\{ next = newlabel(); gen(`goto' next); \}
                                                                        case V<sub>2</sub>:
case V_2:{ label(L_1); L_2=newlabel();
                                                                        if t = V_2 goto L_2
             gen('if' t '!= 'V_2' goto' L_2); \}
                                                                            S<sub>2</sub>.code
S_{2}{ gen('goto' next); }
                                                                           goto next
                                                                        case V<sub>n-1</sub>:
case V_{n-1}:{ label(L_{n-2}); L_{n-1} = newlabel();
                                                               L_{n-2} [if t != V_{n-1} goto L_{n-1}
               gen('if' t '!=' V_{n-1} 'goto' L_{n-1});
                                                                           S_{n-1}.code
S_{n-1}\{gen('goto'next);\}
                                                                           goto next
default:{ label(L_{n-1});}
                                                                            default
S_n\{label(next);\}
                                                                L_{n-1}
                                                                            S_n.code
                                                               next
```

switch

E.code

t=E.addr

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
              E.code
             t=E.addr
              goto test
          case V_1:
              S_1.code
            goto next
         case V<sub>2</sub>:
              S_{2}.code
             goto next
         case V_{n-1}:
             S_{n-1}.code
             goto next
             default
              S_n.code
             goto next
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<sub>n</sub>
next
```

switch语句的另一种翻译

```
switch E \{ t = newtemp(); gen(t'='E.addr); \}
                                                                                        goto test
               test = newlabel(); gen('goto' test); }
                                                                                     case V_1:
case V_1: { L_1= newlabel(); label(L_1); map(V_1, L_1); }
                                                                                        S_1.code
       S_1 { next = newlabel(); gen('goto' next); }
                                                                                       goto next
case V_2: { L_2 = newlabel(); label(L_2); map(V_2, L_2); }
                                                                                    case V<sub>2</sub>:
       \vec{S}, { gen('goto' next); }
                                                                                        S<sub>2</sub>.code
                                                                                       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); }
                                                                                    case V_{n-1}:
default: {L_n = newlabel(); label(L_n);}
                                                                                       S_{n-1}.code
      S_n = \{gen('goto' next);
                                                                                        goto next
               label(test);
                                                                                        default
              gen(if't'='V_1'goto'L_1);
                                                                                        S_n.code
              gen(if't'='V, goto'L);
                                                                                       goto next
                                                                              test if t = V_1 goto L_1
              gen('if' t'=' V<sub>n-1</sub> 'goto' L<sub>n-1</sub>);
                                                                                   if t = V_2 goto L_2
              gen('goto' L<sub>n</sub>);
                                                                                   if t = V_{n-1} goto L_{n-1}
               label(next);
                                                                                   goto L
                                                                              next
```

switch

E.code

t=E.addr

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

指令 case tV_iL_i 和 $ift=V_i$ 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





过程调用的翻译

```
ightarrow文法

ightarrow S 
ightarrow call id (Elist)
Elist 
ightarrow Elist, E
Elist 
ightarrow E
```

过程调用语句的代码结构

 $id(E_1, E_2, \ldots, E_n)$

```
id (
                             id (
    E_1.code
                           E_1.code
param E_1.addr
                           E_{2}.code
    E_2.code
param E_{2}.addr
                           E_n.code
                       param E_1.addr
    E_n.code
                       param E_2.addr
param E_n.addr
                       param E_n. addr
call id.addr n
                       call id.addr n
```

过程调用语句的代码结构

```
id (
id(E_1, E_2, \ldots, E_n)
                                                    E_1.code
                                                    E_{2}.code
                                                     E_n.code
                                                  param E_1.addr
需要一个队列q存放E_1.addr、E_2.addr、...、
E_m.addr, 以生成
                                                  call id.addr n
```

过程调用语句的SDD

```
\gt S \rightarrow \text{call id } (Elist)
                                                     id (
       n=0;
                                                   E_1.code
       for q中的每个t do
                                                   E_{2}.code
         gen('param' t );
               n = n+1;
                                                    E_n.code
       gen('call' id.addr ',' n);
                                                param E_1.addr
\gt Elist \rightarrow E
                                                param E_2 addr
      将q初始化为只包含E.addr;}
\gt Elist \rightarrow Elist_1, E
                                                param E_n. addr
     将E.addr添加到q的队尾;
                                                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 , f

