# 中间代码生成
# (2. 回填技术)

魏恒峰

hfwei@nju.edu.cn

2021 年 12 月 21 日

$$S \rightarrow \textbf{if}\ (\ B\ )\ S_1 \quad \left| \begin{array}{l} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{array} \right.$$

**$B$ 还不知道 $S.next$ 的指令地址，如何跳转？**

$$S \rightarrow \textbf{if}\ (\ B\ )\ S_1 \quad \left| \begin{array}{l} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{array} \right.$$

**$B$ 还不知道 $S.next$ 的指令地址，如何跳转？**

再扫描一遍中间代码，将标号替换成指令 (相对) 地址

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$

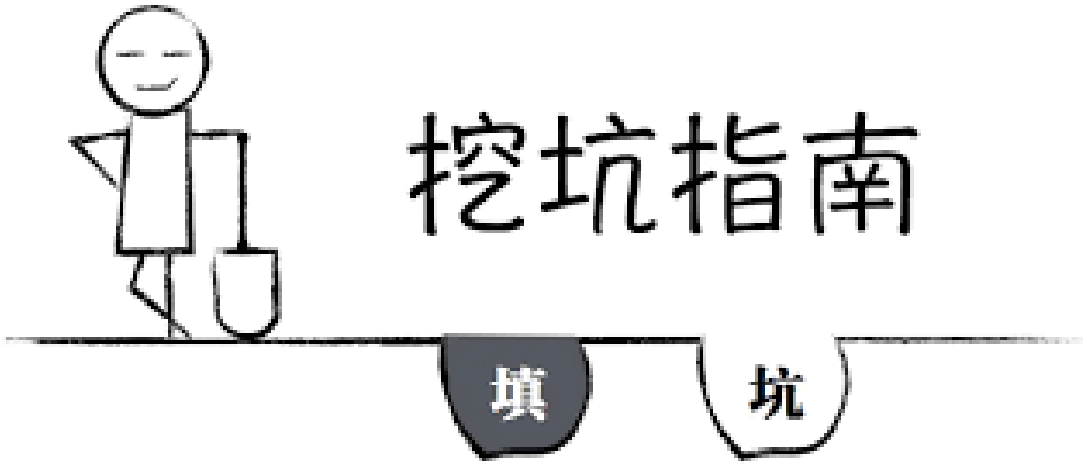
$$
\begin{array}{l}
B.true = newlabel() \\
B.false = S_1.next = S.next \\
S.code = B.code \,||\, label(B.true) \,||\, S_1.code
\end{array}
$$

**$B$ 还不知道 $S.next$ 的指令地址，如何跳转？**

再扫描一遍中间代码，将标号替换成指令 (相对) 地址

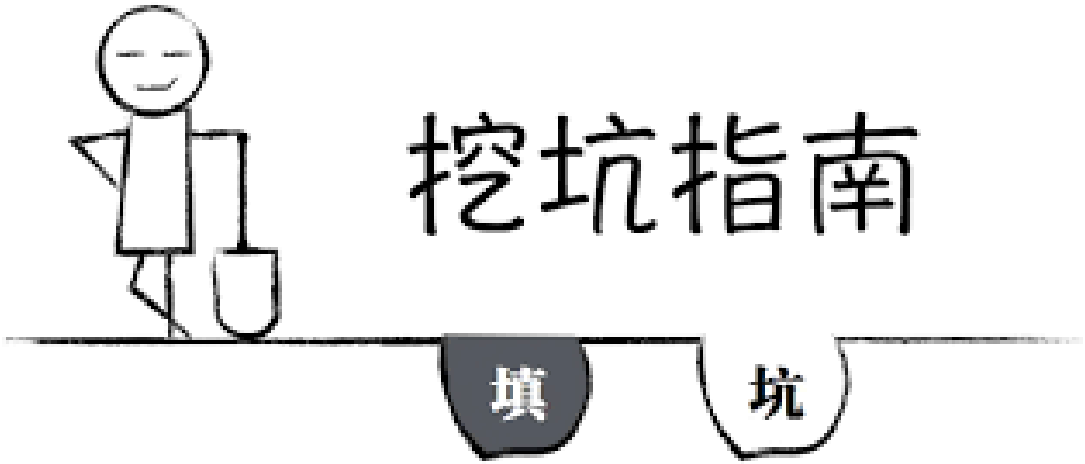
**可否在生成中间代码的时候就填入指令地址？**

# 回填 (Backpatching) 技术




**子节点挖坑、祖先节点填坑**

# 回填 (Backpatching) 技术

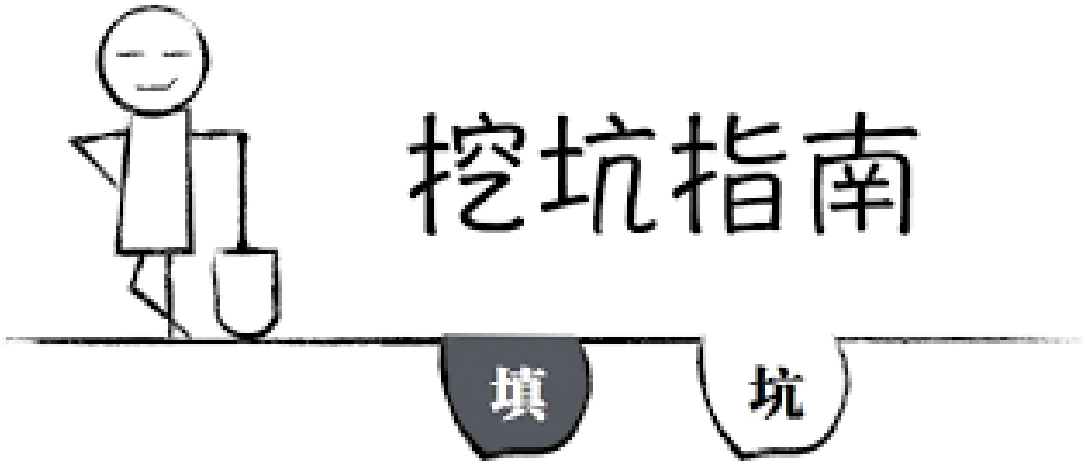



**子节点挖坑、祖先节点填坑**

子节点暂时不指定跳转指令的目标

待父节点能够确定正确的目标地址时回头填充

# 回填 (Backpatching) 技术




**子节点挖坑、祖先节点填坑**

子节点暂时不指定跳转指令的目标

待父节点能够确定正确的目标地址时回头填充

父节点通过**综合属性**收集字节点中具有相同目标的跳转指令

在自底向上的分析过程中

为左部非终结符 $B$ 计算 $B$.truelist 与 $B$.falselist

为左部非终结符 $S$ 计算 $S$.nextlist

并为已能确定目标地址的跳转指令进行回填

# 针对布尔表达式的回填技术

| | | |
|---|---|---|
| 1) | $B \rightarrow B_1 \ \| \ M \ B_2$ | { $backpatch(B_1.falselist, M.instr)$; $B.truelist = merge(B_1.truelist, B_2.truelist)$; $B.falselist = B_2.falselist$; } |
| 2) | $B \rightarrow B_1 \ \&\& \ M \ B_2$ | { $backpatch(B_1.truelist, M.instr)$; $B.truelist = B_2.truelist$; $B.falselist = merge(B_1.falselist, B_2.falselist)$; } |
| 3) | $B \rightarrow \ ! \ B_1$ | { $B.truelist = B_1.falselist$; $B.falselist = B_1.truelist$; } |
| 4) | $B \rightarrow ( \ B_1 \ )$ | { $B.truelist = B_1.truelist$; $B.falselist = B_1.falselist$; } |
| 5) | $B \rightarrow E_1 \ \mathbf{rel} \ E_2$ | { $B.truelist = makelist(nextinstr)$; $B.falselist = makelist(nextinstr + 1)$; $gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto} \ \_')$; $gen('\texttt{goto} \ \_')$; } |
| 6) | $B \rightarrow \mathbf{true}$ | { $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto} \ \_')$; } |
| 7) | $B \rightarrow \mathbf{false}$ | { $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto} \ \_')$; } |
| 8) | $M \rightarrow \epsilon$ | { $M.instr = nextinstr$; } |

综合属性 $B.truelist$ 保存 需要跳转到 $B.true$ 的指令地址

6) $B \rightarrow \textbf{true}$ { $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

7) $B \rightarrow \textbf{false}$ { $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

综合属性 $B.falselist$ 保存 需要跳转到 $B.false$ 的指令地址

综合属性 $B.truelist$ 保存 需要跳转到 $B.true$ 的指令地址

6) $B \rightarrow \textbf{true}$ { $B.truelist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

7) $B \rightarrow \textbf{false}$ { $B.falselist = makelist(nextinstr)$; $gen('\texttt{goto}\ \_')$; }

综合属性 $B.falselist$ 保存 需要跳转到 $B.false$ 的指令地址

| | |
|---|---|
| $B \rightarrow \textbf{true}$ | $B.code = gen('\texttt{goto}'\ B.true)$ |
| $B \rightarrow \textbf{false}$ | $B.code = gen('\texttt{goto}'\ B.false)$ |

5) $B \rightarrow E_1 \ \mathbf{rel} \ E_2$ { $B.truelist = makelist(nextinstr)$;
$B.falselist = makelist(nextinstr + 1)$;
$gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}\ \_')$;
$gen('\texttt{goto}\ \_')$; }

| | |
|---|---|
| $B \rightarrow E_1 \ \mathbf{rel} \ E_2$ | $B.code = E_1.code \ \|\| \ E_2.code$ $\|\| \ gen('\texttt{if}' \ E_1.addr \ \mathbf{rel}.op \ E_2.addr \ '\texttt{goto}' \ B.true)$ $\|\| \ gen('\texttt{goto}' \ B.false)$ |

3) $B \rightarrow\ !\ B_1$ { $B.truelist = B_1.falselist;$ $B.falselist = B_1.truelist;$ }

4) $B \rightarrow (\ B_1\ )$ { $B.truelist = B_1.truelist;$ $B.falselist = B_1.falselist;$ }

$$B \rightarrow\ !\ B_1 \quad \left| \begin{array}{l} B_1.true = B.false \\ B_1.false = B.true \\ B.code = B_1.code \end{array} \right.$$

2) $B \rightarrow B_1 \ \&\& \ M \ B_2$ { $backpatch(B_1.truelist, M.instr)$;
$B.truelist = B_2.truelist$;
$B.falselist = merge(B_1.falselist, B_2.falselist)$; }

8) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$B \rightarrow B_1 \ \&\& \ B_2$

$B_1.true = newlabel()$
$B_1.false = B.false$
$B_2.true = B.true$
$B_2.false = B.false$
$B.code = B_1.code \ || \ label(B_1.true) \ || \ B_2.code$

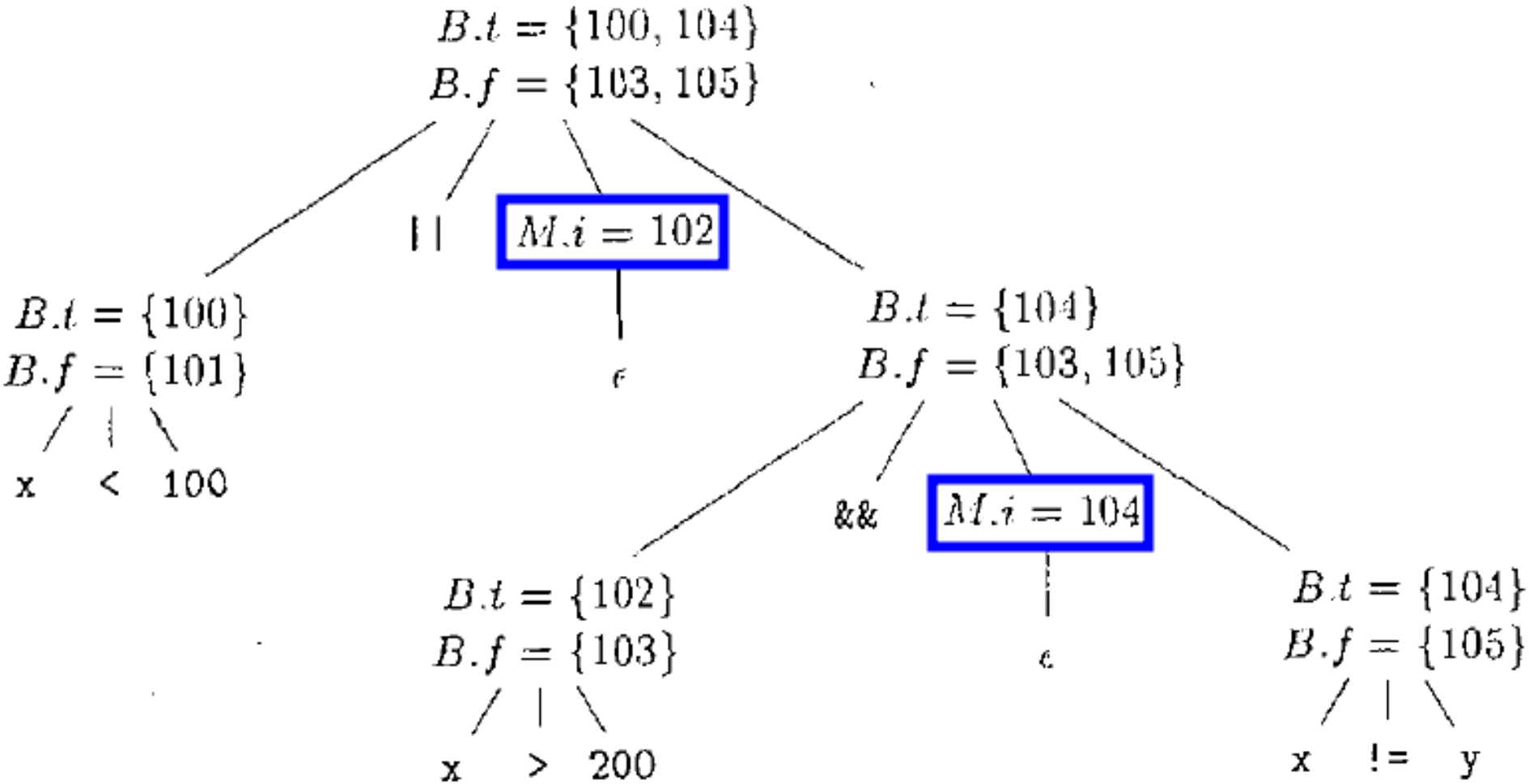
1) $B \rightarrow B_1 \;||\; M \; B_2$ { $backpatch(B_1.falselist, M.instr)$;
$B.truelist = merge(B_1.truelist, B_2.truelist)$;
$B.falselist = B_2.falselist$; }

8) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$B \rightarrow B_1 \;||\; B_2$

$B_1.true = B.true$
$B_1.false = newlabel()$
$B_2.true = B.true$
$B_2.false = B.false$
$B.code = B_1.code \;||\; label(B_1.false) \;||\; B_2.code$

# x < 100 || x > 200 && x != y


B.t = {100, 104}
B.f = {103, 105}
||
M.i = 102
ε
B.t = {100}
B.f = {101}
x < 100
B.t = {104}
B.f = {103, 105}
&&
M.i = 104
ε
B.t = {102}
B.f = {103}
x > 200
B.t = {104}
B.f = {105}
x != y

```
100:   if x < 100 goto _
101:   goto _
102:   if x > 200 goto 104
103:   goto _
104:   if x != y goto _
105:   goto _
```

a) 将 104 回填到指令 102 中之后

```
100:   if x < 100 goto _
101:   goto 102
102:   if x > 200 goto 104
103:   goto _
104:   if x != y goto _
105:   goto _
```

b) 将 102 回填到指令 101 中之后

$$S \rightarrow \textbf{if}\,(\,B\,)\,S \mid \textbf{if}\,(\,B\,)\,S\,\textbf{else}\,S \mid \textbf{while}\,(\,B\,)\,S \mid \{\,L\,\} \mid A\;;$$
$$L \rightarrow L\,S \mid S$$

1) $S \rightarrow \textbf{if}\ (\ B\ )\ M\ S_1$ { $backpatch(B.truelist,\ M.instr)$;
$S.nextlist\ =\ merge(B.falselist,\ S_1.nextlist)$; }

2) $S \rightarrow \textbf{if}\ (\ B\ )\ M_1\ S_1\ N\ \textbf{else}\ M_2\ S_2$
{ $backpatch(B.truelist,\ M_1.instr)$;
$backpatch(B.falselist,\ M_2.instr)$;
$temp\ =\ merge(S_1.nextlist,\ N.nextlist)$;
$S.nextlist\ =\ merge(temp,\ S_2.nextlist)$; }

3) $S \rightarrow \textbf{while}\ M_1\ (\ B\ )\ M_2\ S_1$
{ $backpatch(S_1.nextlist,\ M_1.instr)$;
$backpatch(B.truelist,\ M_2.instr)$;
$S.nextlist\ =\ B.falselist$;
$gen('\texttt{goto}'\ M_1.instr)$; }

4) $S \rightarrow \{\ L\ \}$ { $S.nextlist\ =\ L.nextlist$; }

5) $S \rightarrow A\ ;$ { $S.nextlist\ =\ \textbf{null}$; }

6) $M \rightarrow \epsilon$ { $M.instr\ =\ nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist\ =\ makelist(nextinstr)$;
$gen('\texttt{goto}\ \_')$; }

8) $L \rightarrow L_1\ M\ S$ { $backpatch(L_1.nextlist,\ M.instr)$;
$L.nextlist\ =\ S.nextlist$; }

9) $L \rightarrow S$ { $L.nextlist\ =\ S.nextlist$; }

1) $S \rightarrow \textbf{if}\,(\,B\,)\,M\,S_1\ \{\ backpatch(B.truelist,\ M.instr);$
$S.nextlist = merge(B.falselist,\ S_1.nextlist);\ \}$

6) $M \rightarrow \epsilon \qquad \{\ M.instr = nextinstr;\ \}$

1) $S \rightarrow \textbf{if}\ (\ B\ )\ M\ S_1\ \{\ backpatch(B.truelist,\ M.instr);$
$S.nextlist = merge(B.falselist,\ S_1.nextlist);\ \}$

6) $M \rightarrow \epsilon \qquad \{\ M.instr = nextinstr;\ \}$

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$

$$\begin{cases} B.true = newlabel() \\ B.false = S_1.next = S.next \\ S.code = B.code \,||\, label(B.true) \,||\, S_1.code \end{cases}$$

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

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

6) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist = makelist(nextinstr)$;
$gen('\texttt{goto}\ \_')$; }

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

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

6) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist = makelist(nextinstr)$;
$gen('\texttt{goto}\ \_')$; }

$S \rightarrow$ **if** ( $B$ ) $S_1$ **else** $S_2$

$B.true = newlabel()$
$B.false = newlabel()$
$S_1.next = S_2.next = S.next$
$S.code = B.code$
$\quad || label(B.true) || S_1.code$
$\quad || gen('\texttt{goto}'\ S.next)$
$\quad || label(B.false) || S_2.code$

3) $S \rightarrow$ **while** $M_1$ ( $B$ ) $M_2$ $S_1$

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

6) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

3) $S \rightarrow$ **while** $M_1$ ( $B$ ) $M_2$ $S_1$

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

6) $M \rightarrow \epsilon$ { $M.instr = nextinstr$; }

$S \rightarrow$ **while** ( $B$ ) $S_1$

$begin = newlabel()$
$B.true = newlabel()$
$B.false = S.next$
$S_1.next = begin$
$S.code = label(begin) \,||\, B.code$
$\,||\, label(B.true) \,||\, S_1.code$
$\,||\, gen('\texttt{goto}'\ begin)$

4) $S \rightarrow \{\ L\ \}$ $\{\ S.nextlist = L.nextlist;\ \}$

5) $S \rightarrow A\ ;$ $\{\ S.nextlist = \mathbf{null};\ \}$

6) $M \rightarrow \epsilon$ $\{\ M.instr = nextinstr;\ \}$

8) $L \rightarrow L_1\ M\ S$ $\{\ backpatch(L_1.nextlist,\ M.instr);$
$L.nextlist = S.nextlist;\ \}$

9) $L \rightarrow S$ $\{\ L.nextlist = S.nextlist;\ \}$

1) $S \rightarrow$ **if** ( $B$ ) $M\ S_1$ { $backpatch(B.truelist,\ M.instr)$;
$S.nextlist\ =\ merge(B.falselist,\ S_1.nextlist)$; }

2) $S \rightarrow$ **if** ( $B$ ) $M_1\ S_1\ N$ **else** $M_2\ S_2$
{ $backpatch(B.truelist,\ M_1.instr)$;
$backpatch(B.falselist,\ M_2.instr)$;
$temp\ =\ merge(S_1.nextlist,\ N.nextlist)$;
$S.nextlist\ =\ merge(temp,\ S_2.nextlist)$; }

3) $S \rightarrow$ **while** $M_1$ ( $B$ ) $M_2\ S_1$
{ $backpatch(S_1.nextlist,\ M_1.instr)$;
$backpatch(B.truelist,\ M_2.instr)$;
$S.nextlist\ =\ B.falselist$;
$gen('\texttt{goto}'\ M_1.instr)$; }

4) $S \rightarrow$ { $L$ } { $S.nextlist\ =\ L.nextlist$; }

5) $S \rightarrow A$ ; { $S.nextlist\ =$ **null**; }

6) $M \rightarrow \epsilon$ { $M.instr\ =\ nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist\ =\ makelist(nextinstr)$;
$gen('\texttt{goto}\ \_')$; }

8) $L \rightarrow L_1\ M\ S$ { $backpatch(L_1.nextlist,\ M.instr)$;
$L.nextlist\ =\ S.nextlist$; }

9) $L \rightarrow S$ { $L.nextlist\ =\ S.nextlist$; }

只有 (3) 与 (7) 生成了新的代码, 控制流语句的主要目的是“控制”流。

```
switch ( E ) {
        case V1: S1
        case V2: S2
                 ...
        case Vn-1: Sn-1
        default: Sn
}
```

非 $C$ 语言语义 (break)

```
switch ( E ) {
        case V1: S1
        case V2: S2
                 ...
        case Vn-1: Sn-1
        default: Sn
}
```

非 $C$ 语言语义 (break)

```
        code to evaluate E into t
        goto test
L1:     code for S1
        goto next
L2:     code for S2
        goto next
        ...
Ln-1:   code for Sn-1
        goto next
Ln:     code for Sn
        goto next
test:   if t = V1 goto L1
        if t = V2 goto L2
        ...
        if t = Vn-1 goto Ln-1
        goto Ln
next:
```

```
switch ( E ) {
    case V_1 : S_1
    case V_2 : S_2
             ...
    case V_{n-1} : S_{n-1}
    default : S_n
}
```

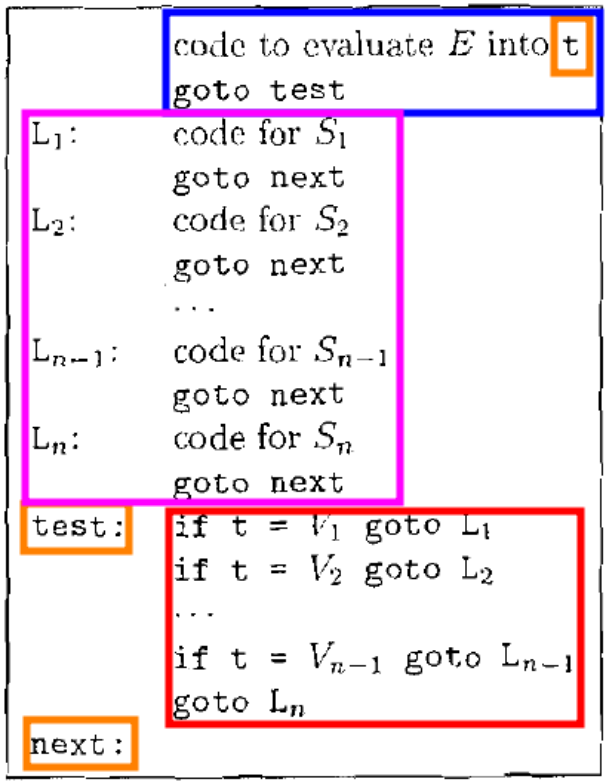
$V_i : L_i$ 队列

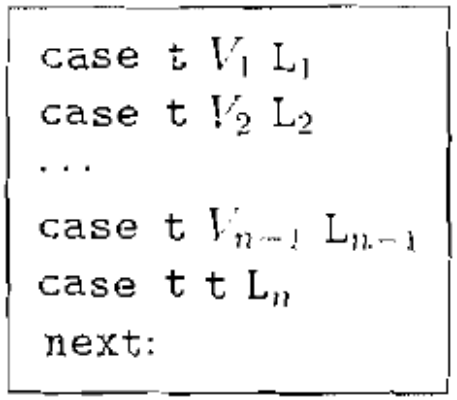
```
        code to evaluate E into t
        goto test
L_1 :   code for S_1
        goto next
L_2 :   code for S_2
        goto next
        ...
L_{n-1} : code for S_{n-1}
        goto next
L_n :   code for S_n
        goto next
test :  if t = V_1 goto L_1
        if t = V_2 goto L_2
        ...
        if t = V_{n-1} goto L_{n-1}
        goto L_n
next :
```

```
        code to evaluate E into t
        goto test
L1:     code for S1
        goto next
L2:     code for S2
        goto next
        ...
Ln-1:   code for Sn-1
        goto next
Ln:     code for Sn
        goto next
test:   if t = V1 goto L1
        if t = V2 goto L2
        ...
        if t = Vn-1 goto Ln-1
        goto Ln
next:
```

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

case 三地址代码


code to evaluate $E$ into t
goto test
$L_1$: code for $S_1$
goto next
$L_2$: code for $S_2$
goto next
...
$L_{n-1}$: code for $S_{n-1}$
goto next
$L_n$: code for $S_n$
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 $L_n$
next:


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:

```
        code to evaluate E into t
        goto test
L1:     code for S1
        goto next
L2:     code for S2
        goto next
        ...
Ln-1:   code for Sn-1
        goto next
Ln:     code for Sn
        goto next
test:   if t = V1 goto L1
        if t = V2 goto L2
        ...
        if t = Vn-1 goto Ln-1
        goto Ln
next:
```

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



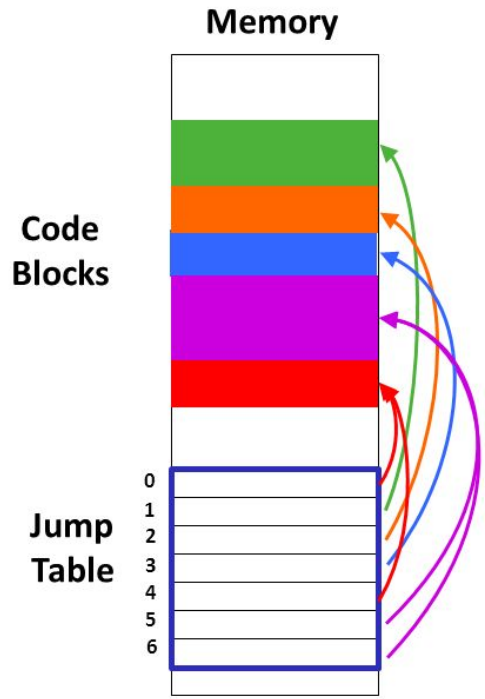
## Jump Table Structure

C code:

```
switch(x) {
  case 1: <some code>
          break;
  case 2: <some code>
  case 3: <some code>
          break;
  case 5:
  case 6: <some code>
          break;
  default: <some code>
}
```

We can use the jump table when x <= 6:

```
if (x <= 6)
  target = JTab[x];
  goto *target;
else
  goto default;
```






Jump Table 优化

# 函数/过程的中间代码翻译

```
n = f(a[i])
```

```
1)   t1 = i * 4
2)   t2 = a [ t1 ]
3)   param t2
4)   t3 = call f, 1
5)   n = t3
```

## 新增文法以支持函数定义与调用

$$
\begin{array}{lcl}
D & \rightarrow & \textbf{define}\ T\ \textbf{id}\ (\ F\ )\ \{\ S\ \} \\
F & \rightarrow & \epsilon \mid T\ \textbf{id}\ ,\ F \\
S & \rightarrow & \textbf{return}\ E\ ; \\
E & \rightarrow & \textbf{id}\ (\ A\ ) \\
A & \rightarrow & \epsilon \mid E\ ,\ A
\end{array}
$$

# 函数定义

$$
\begin{array}{lcl}
D & \rightarrow & \textbf{define}\ T\ \textbf{id}\ (\ F\ )\ \{\ S\ \} \\
F & \rightarrow & \epsilon \mid T\ \textbf{id}\ ,\ F \\
S & \rightarrow & \textbf{return}\ E\ ;
\end{array}
$$

函数名 id 放入当前符号表，建立新的符号表，处理形参 $F$ 与函数体 $S$

# 函数调用

$$E \rightarrow \boxed{\text{id} \ ( \ A \ )}$$

$$A \rightarrow \epsilon \mid E \ , \ A$$

```
param x1
param x2
...
param xn
call p, n
```

# 函数调用

S::= CALL id(Elist) { S.code := Elist.code
A
|| gencode("CALL", id.place, Elist.number) }

Elist::= $Elist_1$, E { Elist.code := E.code || $Elist_1$.code
逆序
|| gencode("PARAM", E.place);
Elist.number := $Elist_1$.number + 1 }

Elist::= E { Elist.code := E.code || gencode("PARAM", E.place);
Elist.number := 1 }

C 语言并未规定参数计算的顺序

# 函数调用

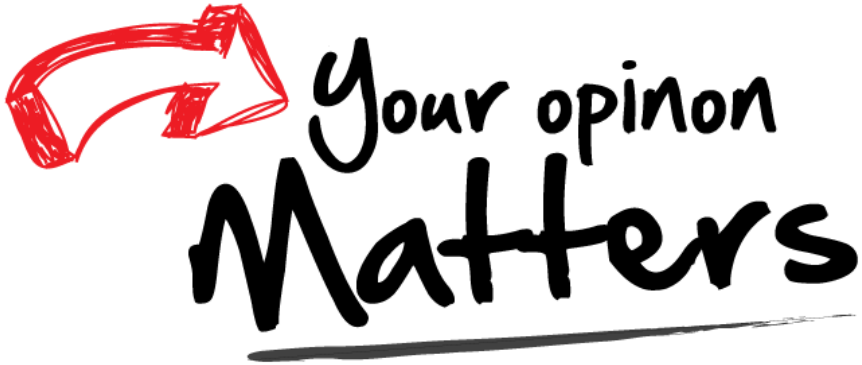
```
S::= CALL id(Elist)A
      { Count := 0; S.code := Elist.code;
        while NOT EmptyQ(q) do
        begin
           t := HeadQ(q);
           S.code := S.code || gencode("PARAM",t);
           DelQ(q); Count := Count + 1
        end;
        S.code := S.code || gencode("CALL",id.place,Coun
      }
```

逆序

```
Elist::= Elist1,E { Elist.code := E.code || Elist1.code;
                    EnterQ(E.place,q)}
Elist::= E        { Elist.code := E.code; CreateQ(q);
                    EnterQ(E.place,q)}
```

集中生成 **param** 指令, 代码更紧凑


Thank
You!




Office 926

hfwei@nju.edu.cn