

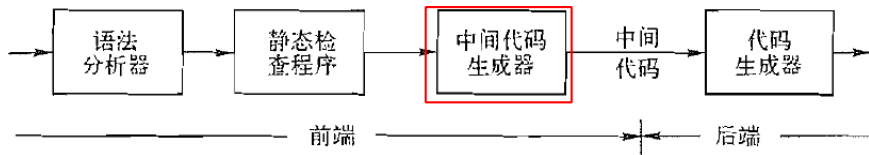
# 中间代码生成

魏恒峰

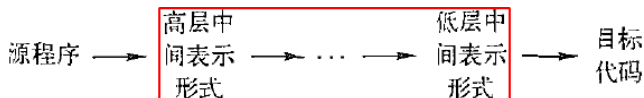
hfwei@nju.edu.cn

2020 年 12 月 24 日

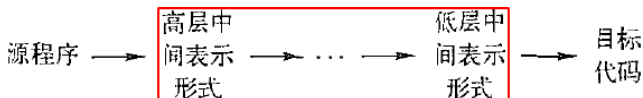




## Intermediate Representation (IR)



## Intermediate Representation (IR)



**精确:** 不能丢失源程序的信息

**独立:** 不依赖特定的源语言与目标语言  
(如, 没有复杂的寻址方式)

## Intermediate Representation (IR)

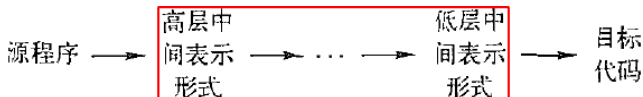
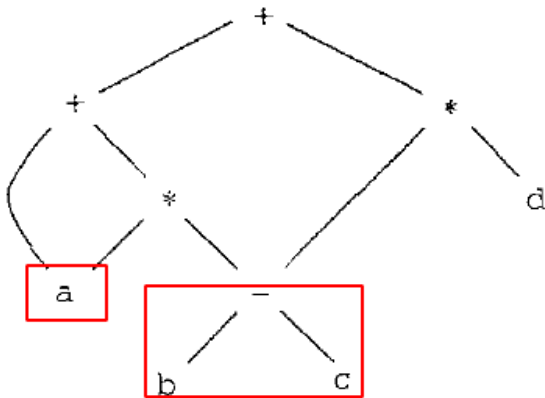


图 (抽象语法树)、三地址代码、C 语言

## 表达式的有向无环图



$$a + a * (b - c) + (b - c) * d$$

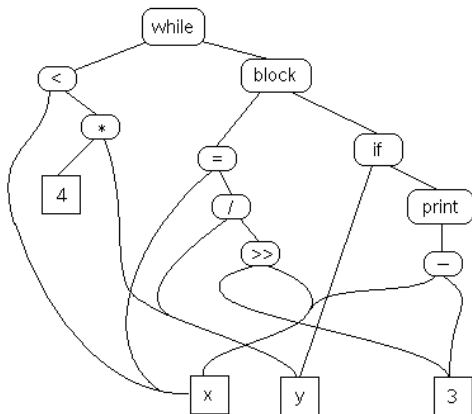
产生式	语义规则
1) $E \rightarrow E_1 + T$	$E.node = \text{new Node}('+', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \text{new Node}('-', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
$T \rightarrow T_1 * F$	$T.node = \text{new Node}('*', T_1.node, F.node)$
4) $T \rightarrow ( E )$	$T.node = E.node$
5) $T \rightarrow \text{id}$	$T.node = \text{new Leaf}(\text{id}, \text{id.entry})$
6) $T \rightarrow \text{num}$	$T.node = \text{new Leaf}(\text{num}, \text{num.val})$

产生式	语义规则
1) $E \rightarrow E_1 + T$	$E.node = \text{new Node}('+', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \text{new Node}('-', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
$T \rightarrow T_1 * F$	$T.node = \text{new Node}('*', T_1.node, F.node)$
4) $T \rightarrow ( E )$	$T.node = E.node$
5) $T \rightarrow \text{id}$	$T.node = \text{new Leaf}(\text{id}, \text{id.entry})$
6) $T \rightarrow \text{num}$	$T.node = \text{new Leaf}(\text{num}, \text{num.val})$

在创建节点之前, 先判断是否已存在 (哈希表)



```
while (x < 4 * y) {
    x = y / 3 >> x;
    if (y) print x - 3;
}
```



Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y \textbf{ op } z \quad (1)$$

$$x = \textbf{ op } y \quad (2)$$

$$x = y \quad (3)$$

## Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y \text{ op } z \quad (1)$$

$$x = \text{op } y \quad (2)$$

$$x = y \quad (3)$$

$$\text{goto } L \quad (4)$$

$$\text{if } x \text{ goto } L \quad (5)$$

$$\text{if False } x \text{ goto } L \quad (6)$$

$$\text{if } x \text{ relop } y \text{ goto } L \quad (7)$$

## Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

`param x` (8)  
`call p, n` (9)  
`y = call p, n` (10)  
`return y` (11)

`param  $x_1$`   
`param  $x_2$`   
`...`

`param  $x_n$`   
`call p, n`

$p(x_1, x_2, \dots, x_n)$

Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y[i] \quad (12)$$

$$x[i] = y \quad (13)$$

距离位置  $y$  处  $i$  个内存单元

Definition (三地址代码 (Three-Address Code (TAC; 3AC)))

每个 **TAC** 指令**最多**包含三个操作数。

$$x = y[i] \quad (12)$$

$$x[i] = y \quad (13)$$

$$x = \&y \quad (14)$$

$$x = *y \quad (15)$$

$$*x = y \quad (16)$$

距离位置  $y$  处  $i$  个内存单元

```
do i = i + 1; while (a[i] < v);
```

```
L:  t1 = i + 1  
    i = t1  
    t2 = i * 8  
    t3 = a [ t2 ]  
    if t3 < v goto L
```

do i = i + 1; while (a[i] < v);

```
L:  t1 = i + 1  
    i = t1  
    t2 = i * 8  
    t3 = a [ t2 ]  
    if t3 < v goto L
```

```
100: t1 = i + 1  
101: i = t1  
102: t2 = i * 8  
103: t3 = a [ t2 ]  
104: if t3 < v goto 100
```



## 三地址代码的四元式表示

### Definition (四元式 (Quadruple))

一个四元式包含四个字段, 分别为  $op$ 、 $arg_1$ 、 $arg_2$  与  $result$ 。

## 三地址代码的**四元式**表示

### Definition (四元式 (Quadruple))

一个四元式包含四个字段, 分别为  $op$ 、 $arg_1$ 、 $arg_2$  与  $result$ 。

$$a + a * (b - c) + (b - c) * d$$

$t_1 = \text{minus } c$

$t_2 = b * t_1$

$t_3 = \text{minus } c$

$t_4 = b * t_3$

$t_5 = t_2 + t_4$

$a = t_5$

	$op$	$arg_1$	$arg_2$	$result$
0	minus	c		$t_1$
1	*	b	$t_1$	$t_2$
2	minus	c		$t_3$
3	*	b	$t_3$	$t_4$
4	+	$t_2$	$t_4$	$t_5$
5	=	$t_5$		a
		...		

$$x = y[i]$$

$$x[i] = y$$

$$=[ ] \quad y \quad i \quad x$$

$$[ ] = \quad i \quad y \quad x$$

$$x = y[i]$$

$$x[i] = y$$

$$=[ ] \quad y \quad i \quad x$$

$$[ ] = \quad i \quad y \quad x$$

$$x = \&y$$

$$x = *y$$

$$*x = y$$

$x = y[i]$

$x[i] = y$

$=[]$        $y$        $i$        $x$

$[]=$        $i$        $y$        $x$

$x = \&y$

$x = *y$

$*x = y$

$=\&$        $y$        $x$

$=*$        $y$        $x$

$*=$        $y$        $x$

## 表达式的中间代码翻译

产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) '=' E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr '=' E_1.addr '+' E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr '=' 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ $E.code = ''$

符号表条目

综合属性  $E.code$  与  $E.addr$

产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) '=' E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr '=' E_1.addr '+' E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr '=' 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ $E.code = ''$

符号表条目

$t_1 = \text{minus } c$   
 $t_2 = b + t_1$   
 $a = t_2$

$a = b + -c$

## 表达式的中间代码翻译 (增量式)

$S \rightarrow id = E ;$	$\{ gen( top.get(id.lexeme) \neq E.addr); \}$
$E \rightarrow E_1 + E_2$	$\{ E.addr = new Temp();$ $gen(E.addr \neq E_1.addr '+' E_2.addr); \}$
$  - E_1$	$\{ E.addr = new Temp();$ $gen(E.addr \neq 'minus' E_1.addr); \}$
$  ( E_1 )$	$\{ E.addr = E_1.addr; \}$
$  id$	$\{ E.addr = top.get(id.lexeme); \}$

综合属性  $E.addr$



## 数组引用的中间代码翻译

声明 : `int a[2][3]`

数组引用 :  $x = a[1][2]; a[1][2] = x$

## 数组引用的中间代码翻译

声明 : `int a[2][3]`

数组引用 : `x = a[1][2]; a[1][2] = x`

需要计算 `a[1][2]` 的相对于**数组基地址** `a` 的**偏移地址**

## 数组引用的中间代码翻译

```
int a[2][3]
```

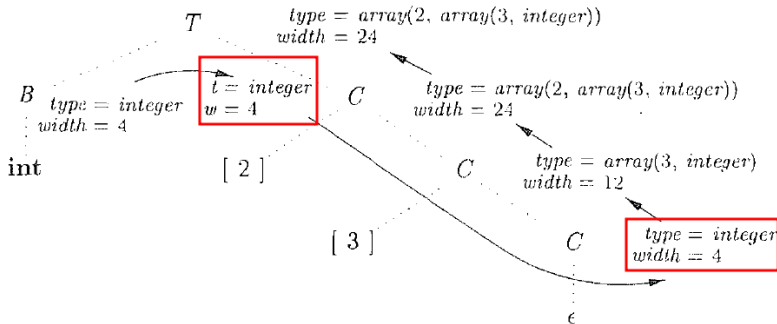


图 6-16 数组类型的语法制导翻译

## 数组类型声明

int  $a[2][3]$

$array(2, array(3, integer))$

	类型	宽度
$a$	$array(2, array(3, integer))$	24
$a[i]$	$array(3, integer)$	12
$a[i][j]$	integer	4

`int a[2][3]`

`array(2, array(3, integer))`

	类型	宽度
$a$	<code>array(2, array(3, integer))</code>	24
$a[i]$	<code>array(3, integer)</code>	12
$a[i][j]$	<code>integer</code>	4

$$\text{addr}(a[1][2]) = \text{base} + 1 \times 12 + 2 \times 4$$

```

S → id = E ; { gen( top.get(id.lexeme) '=' E.addr); }

| L = E ; { gen(L.array.base '[' L.addr ']' '=' E.addr); }

E → E1 + E2 { E.addr = new Temp();
                gen(E.addr '=' E1.addr '+' E2.addr); }

| id { E.addr = top.get(id.lexeme); }

| L { E.addr = new Temp();
      gen(E.addr '=' L.array.base '[' L.addr ']'); }

L → id [ E ] { L.array = top.get(id.lexeme);
               L.type = L.array.type.elem;
               L.addr = new Temp();
               gen(L.addr '=' E.addr '*' L.type.width); }

| L1 [ E ] { L.array = L1.array;
               L.type = L1.type.elem;
               t = new Temp();
               L.addr = new Temp();
               gen(t '=' E.addr '*' L.type.width);
               gen(L.addr '=' L1.addr '+' t); }

```

int a[2][3]

## 综合属性 $L.array.base$ : 数组基地址 (即, 数组名)

$S \rightarrow id = E ; \quad \{ gen(top.get(id.lexeme) \neq E.addr); \}$

$\quad | \quad L = E ; \quad \{ gen(L.array.base '[' L.addr ']' \neq E.addr); \}$

$E \rightarrow E_1 + E_2 \quad \{ E.addr = new Temp();$   
 $\quad \quad \quad gen(E.addr \neq E_1.addr + E_2.addr); \}$

$\quad | \quad id \quad \quad \quad \{ E.addr = top.get(id.lexeme); \}$

$\quad | \quad L \quad \quad \quad \{ E.addr = new Temp();$   
 $\quad \quad \quad gen(E.addr \neq L.array.base '[' L.addr ']); \}$

## 综合属性 $L.addr$ : 偏移地址

## 综合属性 $L.array$ : 数组名 $id$ 对应的符号表条目

```
 $L \rightarrow id [ E ] \quad \{ L.array = top.get(id.lexeme);$   
     $L.type = L.array.type.elem;$   
     $L.addr = new Temp();$   
     $gen(L.addr '=' E.addr '*' L.type.width); \}$   
  
|  $L_1 [ E ] \quad \{ L.array = L_1.array;$   
     $L.type = L_1.type.elem;$   
     $t = new Temp();$   
     $L.addr = new Temp();$   
     $gen(t '=' E.addr '*' L.type.width);$   
     $gen(L.addr '=' L_1.addr '+' t); \}$ 
```

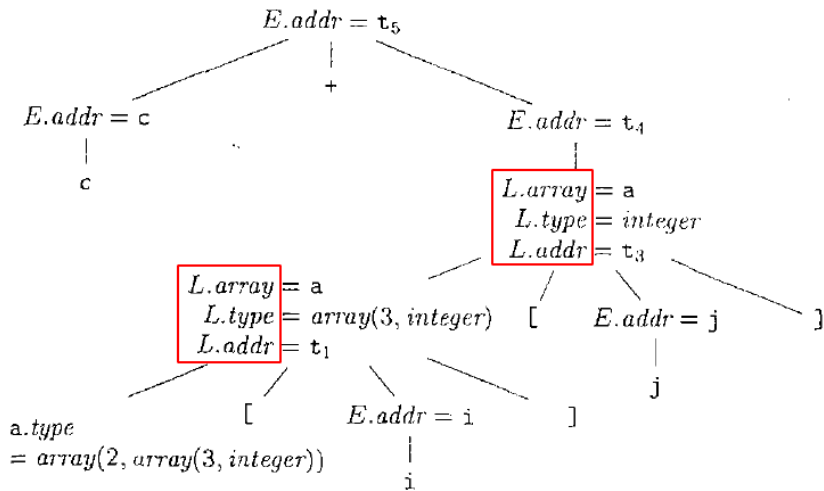


## 综合属性 $L.type$ : (当前) 元素类型

```
 $L \rightarrow id [ E ] \quad \{ L.array = top.get(id.lexeme);$   
     $L.type = L.array.type.elem;$   
     $L.addr = new Temp();$   
     $gen(L.addr '=' E.addr '*' L.type.width); \}$   
  
|  $L_1 [ E ] \quad \{ L.array = L_1.array;$   
     $L.type = L_1.type.elem;$   
     $t = new Temp();$   
     $L.addr = new Temp();$   
     $gen(t '=' E.addr '*' L.type.width);$   
     $gen(L.addr '=' L_1.addr '+' t); \}$ 
```

## 综合属性 $L.addr$ : (当前) 偏移地址

```
 $L \rightarrow id [ E ] \quad \{ L.array = top.get(id.lexeme);$   
     $L.type = L.array.type.elem;$   
     $L.addr = new Temp();$   
     $gen(L.addr '=' E.addr '*' L.type.width); \}$   
  
 $| L_1 [ E ] \quad \{ L.array = L_1.array;$   
     $L.type = L_1.type.elem;$   
     $t = new Temp();$   
     $L.addr = new Temp();$   
     $gen(t '=' E.addr '*' L.type.width);$   
     $gen(L.addr '=' L_1.addr '+' t); \}$ 
```



`int a[2][3]`

```
t1 = i * 12  
t2 = j * 4  
t3 = t1 + t2  
t4 = a [ t3 ]  
t5 = c + t4
```

`int a[2][3]`

## 控制流语句与布尔表达式的中间代码翻译

$$S \rightarrow \text{if } (B) \ S_1$$
$$S \rightarrow \text{if } (B) \ S_1 \ \text{else } S_2$$
$$S \rightarrow \text{while } (B) \ S_1$$

## 控制流语句与布尔表达式的中间代码翻译



产生式	语义规则
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow assign$	$S.code = assign.code$
$S \rightarrow if ( B ) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow if ( B ) S_1 else S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' S.next)$ $\parallel label(B.false) \parallel S_2.code$
$S \rightarrow while ( B ) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \parallel B.code$ $\parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' begin)$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$

继承属性  $S.next$  :  $S$  的下一条指令

$$P \rightarrow S \quad \left| \begin{array}{l} S.next = newlabel() \\ P.code = S.code || label(S.next) \end{array} \right.$$

$S.next$  为语句  $S$  指明了“跳出”  $S$  的目标



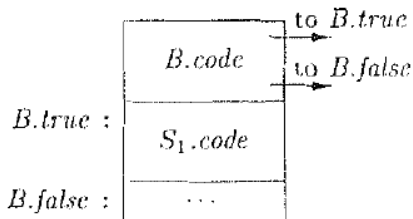
$$S \rightarrow \text{assign} \quad | \quad S.\text{code} = \text{assign}.\text{code}$$

代表了表达式的翻译, 包括数组引用

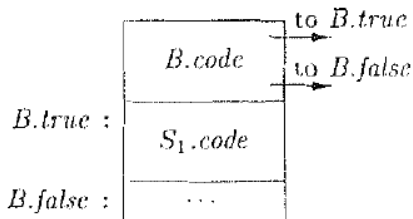
$S \rightarrow \text{if} ( B ) S_1$

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

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

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


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

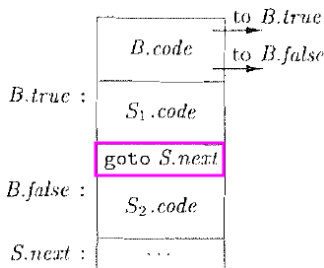
$$\left\{ \begin{array}{l} B.true = \text{newlabel}() \\ B.false = S_1.next = S.next \\ S.code = B.code || \text{label}(B.true) || S_1.code \end{array} \right.$$


`if (true)`

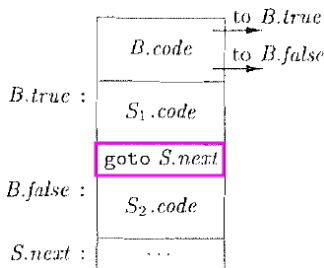
`if (false) assign`

$S \rightarrow \text{if } ( B ) S_1 \text{ else } S_2$	$B.true = \text{newlabel}()$ $B.false = \text{newlabel}()$ <span style="border: 1px solid blue; padding: 2px;"><math>S_1.next = S_2.next = S.next</math></span> $S.code = B.code$ $   \text{label}(B.true)    S_1.code$ $   \text{gen}('goto' S.next)$ $   \text{label}(B.false)    S_2.code$
--	---

$S \rightarrow \text{if} ( B ) S_1 \text{ else } S_2$	$B.true = \text{newlabel}()$ $B.false = \text{newlabel}()$ <span style="border: 1px solid blue; padding: 2px;"><math>S_1.next = S_2.next = S.next</math></span> $S.code = B.code$  $   \text{label}(B.true)    S_1.code$ $   \text{gen}('goto' S.next)$ $   \text{label}(B.false)    S_2.code$
---	---



$S \rightarrow \text{if} ( B ) S_1 \text{ else } S_2$	$B.true = \text{newlabel}()$ $B.false = \text{newlabel}()$ <span style="border: 1px solid blue; padding: 2px;"><math>S_1.next = S_2.next = S.next</math></span> $S.code = B.code$ $   \text{label}(B.true)    S_1.code$ $   \text{gen('goto' } S.next)   $ $   \text{label}(B.false)    S_2.code$
---	---



```

if (true)
    if (true) assign else assign
else
    assign
  
```

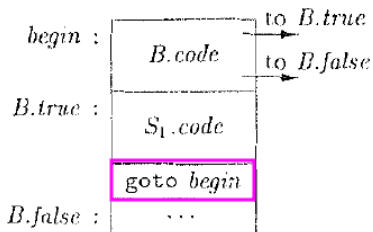
$S \rightarrow \text{while} ( B ) S_1$

```
begin = newlabel()
B.true = newlabel()
B.false = S.next
S1.next = begin
S.code = label(begin) || B.code
        || label(B.true) || S1.code
        || gen('goto' begin)
```



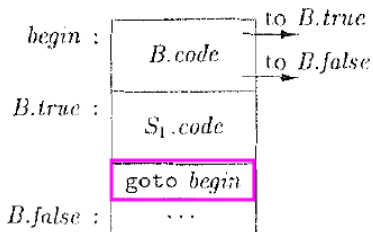
$S \rightarrow \text{while} ( B ) S_1$

```
begin = newlabel()
B.true = newlabel()
B.false = S.next
S1.next = begin
S.code = label(begin) || B.code
           || label(B.true) || S1.code
           || gen('goto' begin)
```



$S \rightarrow \text{while} ( B ) S_1$

```
begin = newlabel()
B.true = newlabel()
B.false = S.next
S1.next = begin
S.code = label(begin) || B.code
           || label(B.true) || S1.code
           || gen('goto' begin)
```



```
while (true)
    if (false) assign else assign
```

$S \rightarrow S_1 S_2$

$S_1.next = newlabel()$   
 $S_2.next = S.next$   
 $S.code = S_1.code || label(S_1.next) || S_2.code$

$S \rightarrow S_1 S_2$

	$S_1.next$	=	$newlabel()$
	$S_2.next$	=	$S.next$
	$S.code$	=	$S_1.code \parallel label(S_1.next) \parallel S_2.code$

if (true) assign else assign assign

产生式	语义规则
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code    label(S.next)$
$S \rightarrow assign$	$S.code = assign.code$
$S \rightarrow if ( B ) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code    label(B.true)    S_1.code$
$S \rightarrow if ( B ) S_1 else S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $   label(B.true)    S_1.code$ $   gen('goto' S.next)$ $   label(B.false)    S_2.code$
$S \rightarrow while ( B ) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin)    B.code$ $   label(B.true)    S_1.code$ $   gen('goto' begin)$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code    label(S_1.next)    S_2.code$

## 布尔表达式的中间代码翻译

产生式	语义规则
$B \rightarrow B_1 \    \ B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \    \ label(B_1.false) \    \ B_2.code$
$B \rightarrow B_1 \ \&\& \ B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \    \ label(B_1.true) \    \ B_2.code$
$B \rightarrow ! B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1 \ rel \ E_2$	$B.code = E_1.code \    \ E_2.code$ $\quad    \ gen('if' \ E_1.addr \ rel.op \ E_2.addr \ 'goto' \ B.true)$ $\quad    \ gen('goto' \ B.false)$
$B \rightarrow true$	$B.code = gen('goto' \ B.true)$
$B \rightarrow false$	$B.code = gen('goto' \ B.false)$

$B \rightarrow \text{true}$       |       $B.\text{code} = \text{gen}(\text{'goto' } B.\text{true})$

$B \rightarrow \text{false}$       |       $B.\text{code} = \text{gen}(\text{'goto' } B.\text{false})$

$B \rightarrow \text{true}$        $B.\text{code} = \text{gen}(\text{'goto' } B.\text{true})$

$B \rightarrow \text{false}$        $B.\text{code} = \text{gen}(\text{'goto' } B.\text{false})$

if (true) assign

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

if (false) assign



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

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

if (!true) assign

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

if (!false) assign

## 短路求值

$$B \rightarrow B_1 \ || \ B_2 \quad \left\{ \begin{array}{l} B_1.true = B.true \\ B_1.false = newlabel() \\ B_2.true = B.true \\ B_2.false = B.false \\ B.code = B_1.code \ || \ label(B_1.false) \ || \ B_2.code \end{array} \right.$$

## 短路求值

$$B \rightarrow B_1 \ || \ B_2 \quad \left| \begin{array}{l} B_1.true = B.true \\ B_1.false = newlabel() \\ B_2.true = B.true \\ B_2.false = B.false \\ B.code = B_1.code \ || \ label(B_1.false) \ || \ B_2.code \end{array} \right.$$

if (true || false) assign

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

if (false || true) assign

## 短路求值

$$B \rightarrow B_1 \ \&\& \ B_2 \quad \left| \begin{array}{l} B_1.true = newlabel() \\ \boxed{B_1.false} = B.false \\ B_2.true = B.true \\ \boxed{B_2.false} = B.false \\ B.code = B_1.code \ || \ label(B_1.true) \ || \ B_2.code \end{array} \right.$$

## 短路求值

$$B \rightarrow B_1 \ \&\& \ B_2 \quad \left| \begin{array}{l} B_1.true = newlabel() \\ B_1.false = B.false \\ B_2.true = B.true \\ B_2.false = B.false \\ B.code = B_1.code \ || \ label(B_1.true) \ || \ B_2.code \end{array} \right.$$

if (true && false) assign

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

if (false && true) assign

$$B \rightarrow E_1 \text{ rel } E_2 \quad \left| \quad \begin{array}{l} B.code = E_1.code \parallel E_2.code \\ \parallel \text{gen('if' } E_1.addr \text{ rel.op } E_2.addr \text{ 'goto' } B.true) \\ \parallel \text{gen('goto' } B.false) \end{array} \right.$$

```
if (x < 100 || x > 200 && x != y) x = 0;
```

```
        if x < 100 goto L2  
        goto L3  
L3:    if x > 200 goto L4  
        goto L1  
L4:    if x != y goto L2  
        goto L1  
L2:    x = 0  
L1:
```



布尔表达式的作用: 布尔值 *vs.* 控制流跳转

$S \rightarrow \text{id} = E; \mid \text{if } (E) \ S \mid \text{while } (E) \ S \mid S \ S$

$E \rightarrow E \parallel E \mid E \&\& E \mid E \text{ rel } E \mid E + E \mid (E) \mid \text{id} \mid \text{true} \mid \text{false}$

布尔表达式的作用: **布尔值** vs. **控制流跳转**

$$\begin{aligned} S &\rightarrow \boxed{\mathbf{id} = E;} \mid \mathbf{if} (E) S \mid \mathbf{while} (E) S \mid S S \\ E &\rightarrow E \parallel E \mid E \&\& E \mid E \mathbf{rel} E \mid E + E \mid (E) \mid \mathbf{id} \mid \mathbf{true} \mid \mathbf{false} \end{aligned}$$

函数  $jump(t, f)$ : 生成控制流代码

函数  $rvalue()$ : 生成计算布尔值的代码, 并将结果存储在临时变量中

产生式	语义规则
$S \rightarrow id = E ;$	$S.code = E.code \parallel$ $gen(top.get(id.lexeme) '=' E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = new Temp()$ $E.code = E_1.code \parallel E_2.code \parallel$ $gen(E.addr '=' E_1.addr '+' E_2.addr)$
$  - E_1$	$E.addr = new Temp()$ $E.code = E_1.code \parallel$ $gen(E.addr '=' 'minus' E_1.addr)$
$  ( E_1 )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$  id$	$E.addr = top.get(id.lexeme)$ 符号表条目 $E.code = ''$

$$E \rightarrow E_1 \&\& E_2$$

为  $E$  生成跳转代码, 在真假出口处将 true 或 false 存储到临时变量

`x = a < b && c < d`

```
    ifFalse a < b goto L1
    ifFalse c < d goto L1
    t = true
    goto L2
L1: t = false
L2: x = t
```

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

$\left\{ \begin{array}{l} B.true = \text{newlabel}() \\ B.false = S_1.next = S.next \\ S.code = B.code \parallel \text{label}(B.true) \parallel S_1.code \end{array} \right.$

*B* 还不知道 *S.next* 的指令地址, 如何跳转?

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

$\left\{ \begin{array}{l} B.true = \text{newlabel}() \\ B.false = S_1.next = S.next \\ S.code = B.code \parallel \text{label}(B.true) \parallel S_1.code \end{array} \right.$

*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

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

*B* 还不知道 *S.next* 的指令地址, 如何跳转?

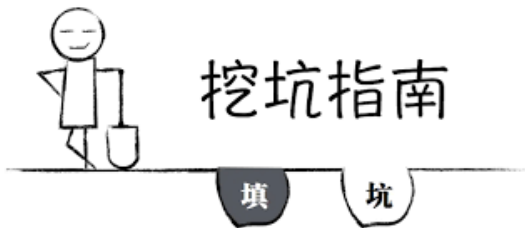
再扫描一遍中间代码, 将标号替换成指令 (相对) 地址

可否在生成中间代码的时候就填入指令地址?

## 回填 (Backpatch) 技术



## 回填 (Backpatch) 技术



子节点挖坑、祖先节点填坑

## 针对布尔表达式的回填技术

- |    |                                      |   |
|----|--------------------------------------|---|
| 1) | $B \rightarrow B_1 \parallel M B_2$  | { <i>backpatch</i> ( <i>B</i> <sub>1</sub> . <i>false</i> list, <i>M.instr</i> );<br><i>B.true</i> list = <i>merge</i> ( <i>B</i> <sub>1</sub> . <i>true</i> list, <i>B</i> <sub>2</sub> . <i>true</i> list);<br><i>B.false</i> list = <i>B</i> <sub>2</sub> . <i>false</i> list; } |
| 2) | $B \rightarrow B_1 \&\& M B_2$       | { <i>backpatch</i> ( <i>B</i> <sub>1</sub> . <i>true</i> list, <i>M.instr</i> );<br><i>B.true</i> list = <i>B</i> <sub>2</sub> . <i>true</i> list;<br><i>B.false</i> list = <i>merge</i> ( <i>B</i> <sub>1</sub> . <i>false</i> list, <i>B</i> <sub>2</sub> . <i>false</i> list); } |
| 3) | $B \rightarrow ! B_1$                | { <i>B.true</i> list = <i>B</i> <sub>1</sub> . <i>false</i> list;<br><i>B.false</i> list = <i>B</i> <sub>1</sub> . <i>true</i> list; }  |
| 4) | $B \rightarrow ( B_1 )$              | { <i>B.true</i> list = <i>B</i> <sub>1</sub> . <i>true</i> list;<br><i>B.false</i> list = <i>B</i> <sub>1</sub> . <i>false</i> list; }  |
| 5) | $B \rightarrow E_1 \text{ rel } E_2$ | { <i>B.true</i> list = <i>makelist</i> ( <i>nextinstr</i> );<br><i>B.false</i> list = <i>makelist</i> ( <i>nextinstr</i> + 1);<br><i>gen</i> ('if' <i>E</i> <sub>1</sub> . <i>addr</i> <i>rel.op</i> <i>E</i> <sub>2</sub> . <i>addr</i> 'goto -');<br><i>gen</i> ('goto -'); }     |
| 6) | $B \rightarrow \text{true}$          | { <i>B.true</i> list = <i>makelist</i> ( <i>nextinstr</i> );<br><i>gen</i> ('goto -'); }  |
| 7) | $B \rightarrow \text{false}$         | { <i>B.false</i> list = <i>makelist</i> ( <i>nextinstr</i> );<br><i>gen</i> ('goto -'); }   |
| 8) | $M \rightarrow \epsilon$             | { <i>M.instr</i> = <i>nextinstr</i> ; }   |

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

- 6)  $B \rightarrow \text{true}$       {  $B.truelist = makelist(nextinstr);$   
                               $gen('goto \_');$  }
- 7)  $B \rightarrow \text{false}$       {  $B.falselist = makelist(nextinstr);$   
                               $gen('goto \_');$  }

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

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

$B \rightarrow true$	$B.code = gen('goto' B.true)$
$B \rightarrow false$	$B.code = gen('goto' B.false)$

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

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

$$3) \quad B \rightarrow ! B_1$$

$$\{ \boxed{B.true\text{list}} = B_1.false\text{list}; \\ \boxed{B.false\text{list}} = B_1.true\text{list}; \}$$

$$4) \quad B \rightarrow ( B_1 )$$

$$\{ \boxed{B.true\text{list}} = B_1.true\text{list}; \\ \boxed{B.false\text{list}} = B_1.false\text{list}; \}$$

$$B \rightarrow ! B_1$$

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

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

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

$B \rightarrow B_1 \ \&\& \ B_2 \quad \begin{cases} B_1.\text{true} = \text{newlabel}() \\ B_1.\text{false} = B.\text{false} \\ B_2.\text{true} = B.\text{true} \\ B_2.\text{false} = B.\text{false} \\ B.\text{code} = B_1.\text{code} \ || \ \text{label}(B_1.\text{true}) \ || \ B_2.\text{code} \end{cases}$

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

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

$B \rightarrow B_1 \parallel B_2$       $\left\{ \begin{array}{l} B_1.true = B.true \\ B_1.false = newlabel() \\ B_2.true = B.true \\ B_2.false = B.false \\ B.code = B_1.code \parallel label(B_1.false) \parallel B_2.code \end{array} \right.$





```
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

$$\begin{aligned} S &\rightarrow \text{if}(B) S \mid \text{if}(B) S \text{ else } S \mid \text{while}(B) S \mid \boxed{\{L\}} \mid A ; \\ L &\rightarrow L S \mid S \end{aligned}$$

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

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

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

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

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

$$S \rightarrow \text{if}(B) M_1 S_1 N \text{ else } M_2 S_2$$

```

{ backpatch(B.truelist, M1.instr);
  backpatch(B.falselist, M2.instr);
  temp = merge(S1.nextlist, N.nextlist);
  S.nextlist = merge(temp, S2.nextlist); }

```

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

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

$$S \rightarrow \text{if}(B) M_1 S_1 N \text{ else } M_2 S_2$$

```

{
  backpatch(B.truelist, M1.instr);
  backpatch(B.falselist, M2.instr);
  temp = merge(S1.nextlist, N.nextlist);
  S.nextlist = merge(temp, S2.nextlist);
}

```

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

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

$S \rightarrow \text{if}(B) S_1 \text{ else } S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $   \text{label}(B.true)    S_1.code$ $   gen('goto' S.next)$ $   \text{label}(B.false)    S_2.code$
--	--

3)  $S \rightarrow \text{while } M_1 (B) M_2 S_1$

```

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

```

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

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

$S \rightarrow \text{while } ( B ) S_1$	<pre> begin = newlabel() B.true = newlabel() B.false = S.next <math>S_1.next = begin</math> S.code = label(begin)    B.code               label(B.true)    <math>S_1.code</math>               gen('goto' begin)</pre>
---	--

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

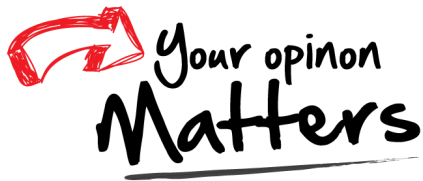
5)  $S \rightarrow A ;$        $\{ S.nextlist = null; \}$

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

Thank  
You!



Office 926

hfwei@nju.edu.cn