

中间代码生成

(1. 表达式的翻译与控制流的翻译)

魏恒峰

hfwei@nju.edu.cn

2021 年 12 月 17 日



Intermediate Representation (IR)

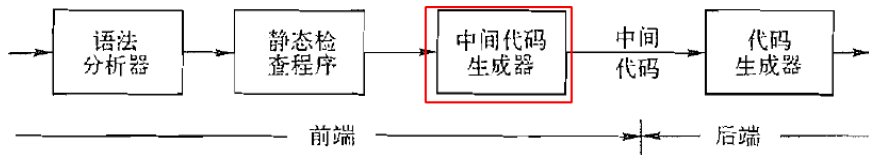
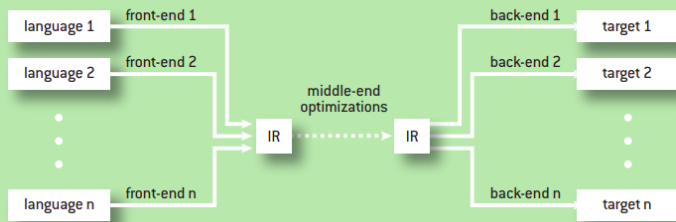


FIGURE 2

A Compiler System Supporting Multiple Languages and Multiple Targets



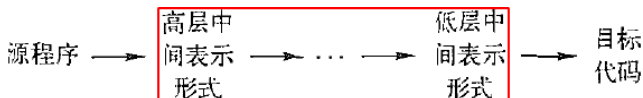
The Increasing Significance of Intermediate Representations in Compilers (Fred Chow; 2013)



LLVM 的核心就是它的 LLVM IR

(希望下一轮授课可以加入 LLVM IR 内容)

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)



华为方舟编译器的 Maple IR 采用多层设计

Intermediate Representation (IR)

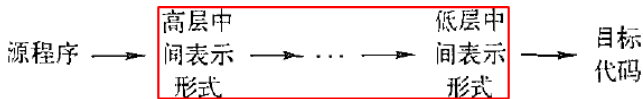


图 (抽象语法树)、**三地址代码**、C 语言

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

每个 TAC **指令** 最多包含三个**地址**。

地址: 源程序中的名字、常量、编译器生成的临时变量

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

每个 TAC **指令** 最多包含三个**地址**。

5 组 16 条**指令**格式 (x, y, z 为**地址**):

$x = y \text{ op } z$	(1)	goto L	(4)
$x = \text{op } y$	(2)	if x goto L	(5)
$x = y$	(3)	if False x goto L	(6)
		if x relop y goto L	(7)

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

每个 TAC **指令**最多包含三个**地址**。

5 组 16 条**指令**格式 (x, y, z 为**地址**):

<code>param x</code>	(8)
<code>call p, n</code>	(9)
<code>$y = \text{call } p, n$</code>	(10)
<code>return y</code>	(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**地址**。

5 组 16 条**指令**格式 (x, y, z 为**地址**):

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

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

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

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

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

距离位置 x/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
```

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

为 (某些) 控制语句生成**带有位置号**的中间代码时, 需要采用**回填技术**

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
		...		

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

$x = \&y$	$=\&$	y	x
$x = *y$	$=*$	y	x
$*x = y$	$*=$	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)

```

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

```

综合属性 $E.addr$

假想一个全局指令缓冲区, 对 gen 的连续调用将生成一个指令序列

数组引用的中间代码翻译

声明 : $\text{int } a[2][3]$

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**

`int a[2][3]`

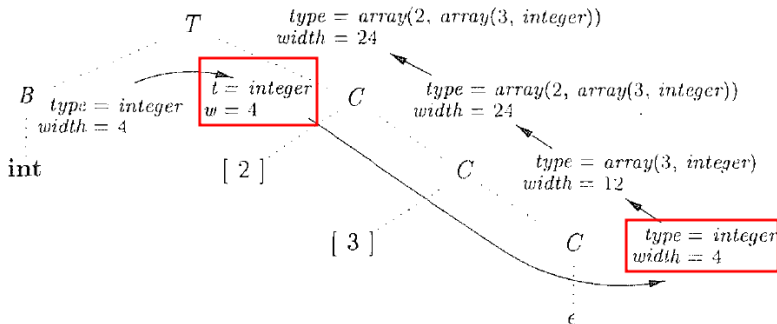


图 6-16 数组类型的语法制导翻译

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

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

```

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

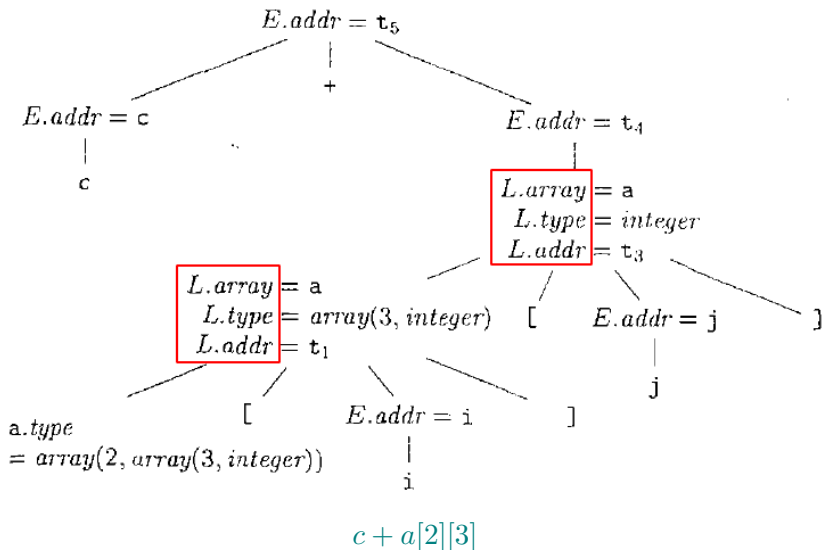
```
 $L \rightarrow id [ E ] \quad \{ L.array = top.get(id.lexeme);$   
     $L.type = L.array.type.elem;$   
     $L.addr = new Temp();$   
     $gen(L.addr '=' E.addr '*' L.type.width); \}$   
  
|  $L_1 [ E ] \quad \{ L.array = L_1.array;$   
     $L.type = L_1.type.elem;$   
     $t = new Temp();$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

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

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



$$\begin{array}{l} t_1 = i * 12 \\ t_2 = j * 4 \\ t_3 = t_1 + t_2 \\ t_4 = a [t_3] \\ t_5 = c + t_4 \end{array}$$

$$c + a[2][3]$$

控制流语句与布尔表达式的中间代码翻译

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

布尔表达式的作用: 布尔值 *vs.* 控制流跳转

$S \rightarrow \text{id} = E; \mid \text{if } (E) \text{ } S \mid \text{while } (E) \text{ } S \mid S \text{ } S$
 $E \rightarrow E \parallel E \mid E \&\& E \mid E \text{ rel } E \mid E + E \mid (E) \mid \text{id} \mid \text{true} \mid \text{false}$

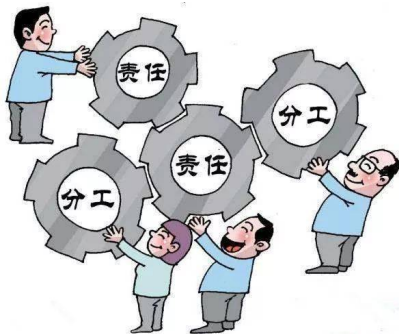
我们先关注“控制流跳转”

控制流语句与布尔表达式的中间代码翻译



分工明确

各司其职



父节点为子节点准备跳转指令的目标标签

子节点通过继承属性确定跳转目标

在自顶向下的分析过程中

为右部的每个 B 计算 $B.true$ 与 $B.false$

为右部的每个 S 计算 $S.next$

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 $S.next$

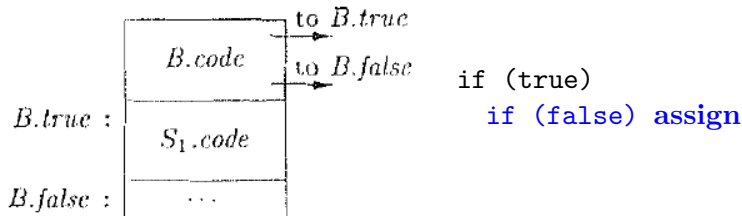
$$P \rightarrow S \quad \left| \begin{array}{l} S.next = newlabel() \\ P.code = S.code || label(S.next) \end{array} \right.$$

$S.next$ 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

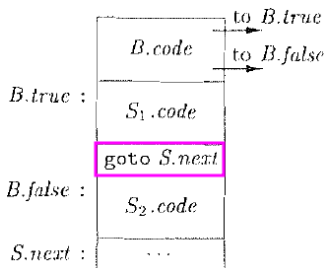
$$S \rightarrow \text{if}(B) S_1$$

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


$$P \rightarrow S$$

$$\begin{array}{l|l} B \rightarrow \text{true} & B.\text{code} = \text{gen}(\text{'goto' } B.\text{true}) \\ B \rightarrow \text{false} & B.\text{code} = \text{gen}(\text{'goto' } B.\text{false}) \end{array}$$

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

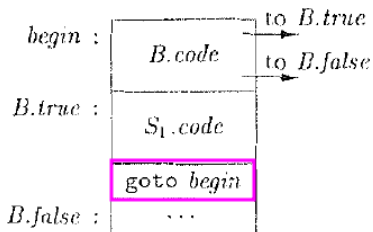


```

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)
```



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

$S \rightarrow S_1 S_2$

$\begin{array}{|l} S_1.next = newlabel() \\ S_2.next = S.next \\ S.code = S_1.code || label(S_1.next) || S_2.code \end{array}$

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

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

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

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

$B \rightarrow ! B_1$

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

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

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

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

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

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

$S \rightarrow \text{id} = E; \mid \text{if} (E) S \mid \text{while} (E) S \mid S S$
 $E \rightarrow E \parallel E \mid E \&\& E \mid E \text{ rel } E \mid E + E \mid (E) \mid \text{id} \mid \text{true} \mid \text{false}$

根据 E 所处的上下文判断 E 所扮演的角色, 调用不同的代码生成函数

函数 $\text{jump}(t, f)$: 生成控制流代码

函数 $\text{rvalue}()$: 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
```

Thank
You!



Office 926

hfwei@nju.edu.cn