

## 四、中间代码生成 (2. 表达式的翻译)

魏恒峰

hfwei@nju.edu.cn

2023 年 05 月 06 日



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

$S \rightarrow id = E ;$	{ $gen(top.get(id.lexeme) '==' E.addr);$ }
$  L = E ;$	{ $gen(L.array.base '[' L.addr ']' '==' E.addr);$ }
$E \rightarrow E_1 + E_2$	{ $E.addr = new Temp();$ $gen(E.addr '==' E_1.addr +' E_2.addr);$ }
$  id$	{ $E.addr = top.get(id.lexeme);$ }
$  L$	{ $E.addr = new Temp();$ $gen(E.addr '==' L.array.base '[' L.addr ']');$ }
$L \rightarrow id [ E ]$	{ $L.array = top.get(id.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = new Temp();$ $gen(L.addr '==' E.addr '*' L.type.width);$ }
$  L_1 [ E ]$	{ $L.array = L_1.array;$ $L.type = L_1.type.elem;$ $t = new Temp();$ $L.addr = new Temp();$ $gen(t '==' E.addr '*' L.type.width);$ $gen(L.addr '==' L_1.addr +' t);$ }

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

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

# 表达式的中间代码翻译

## 综合属性 $E.code$ : 中间代码

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

$$a = b + -c$$

$t_1 = minus c$

$t_2 = b + t_1$

$a = t_2$

综合属性  $E.addr$ : 变量名 (包括临时变量)、常量

```
int main() {  
    int a = 0, b = 1, c = 2;  
    a = b + -c;  
    return 0;  
}
```

%7 = sub nsw i32 0, %6  
%8 = add nsw i32 %5, %7  
store i32 %8, i32\* %2, align 4

## Compiler Explorer

# 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  $a[1][2]$  相对于数组基地址  $a$  的偏移地址

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

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

int  $a[2][3]$

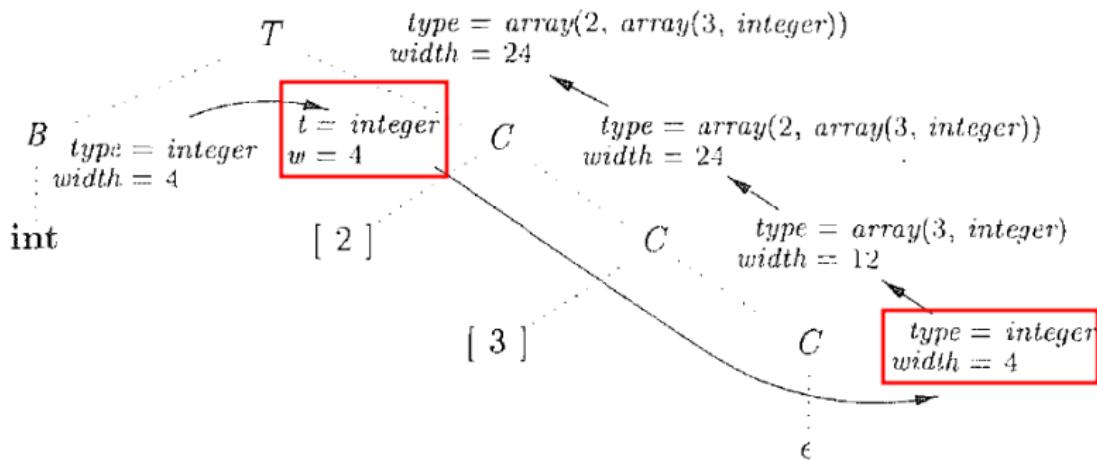
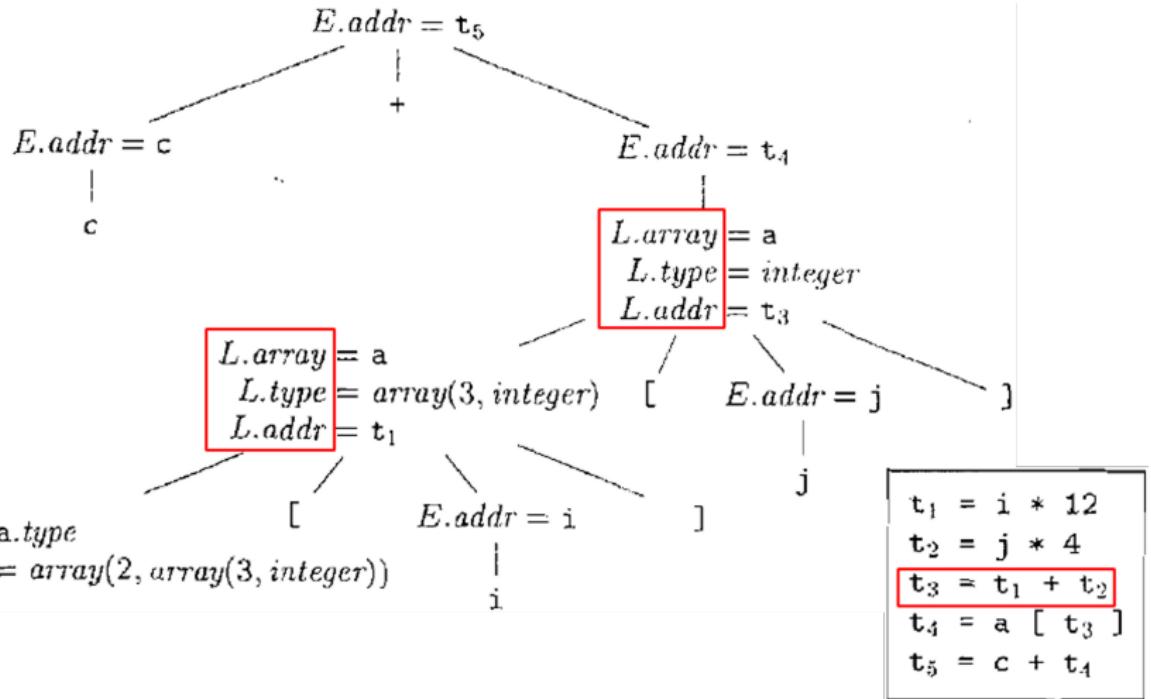


图 6-16 数组类型的语法制导翻译

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

$S \rightarrow id = E ; \quad \{ gen(\text{top.get}(id.lexeme) '==' E.addr); \}$	
	$L = E ; \quad \{ gen(L.array.base '[' L.addr ']' '==' E.addr); \}$
$E \rightarrow E_1 + E_2 \quad \{ E.addr = \text{new Temp}();$	$\quad gen(E.addr '==' E_1.addr +' E_2.addr); \}$
	$id \quad \{ E.addr = \text{top.get}(id.lexeme); \}$
	$L \quad \{ E.addr = \text{new Temp}();$
	$\quad gen(E.addr '==' L.array.base '[' L.addr ']'); \}$
$L \rightarrow id [ E ] \quad \{ L.array = \text{top.get}(id.lexeme);$	$\quad L.type = L.array.type.elem;$
	$\quad L.addr = \text{new Temp}();$
	$\quad gen(L.addr '==' E.addr '*' L.type.width); \}$
$L_1 [ E ] \quad \{ L.array = L_1.array;$	$\quad L.type = L_1.type.elem;$
	$\quad t = \text{new Temp}();$
	$\quad L.addr = \text{new Temp}();$
	$\quad gen(t '==' E.addr '*' L.type.width);$
	$\quad gen(L.addr '==' L_1.addr '+' t); \}$

## 综合属性 $L.Addr$ : 偏移地址



$c + a[i][j]$

```
%2 = alloca [2 x [3 x i32]], align 16

    int main() {
        int a[2][3] = { 0 };

        int i = 1, j = 2;
        int c = 10, d = 20;

        d = c + a[i][j];

        return 0;
    }

%8 = load i32, i32* %5, align 4 %8:c
%9 = load i32, i32* %3, align 4 %9:i
%10 = sext i32 %9 to i64
%11 = getelementptr inbounds [2 x [3 x i32]], [2 x [3 x i32]]* %2, i64 0, i64 %10
%12 = load i32, i32* %4, align 4 %12:j
%13 = sext i32 %12 to i64
%14 = getelementptr inbounds [3 x i32], [3 x i32]* %11, i64 0, i64 %13
%15 = load i32, i32* %14, align 4 %15:a[i][j]
%16 = add nsw i32 %8, %15
store i32 %16, i32* %6, align 4
```

```

%2 = alloca [2 x [3 x i32]], align 16

    int main() {
        int a[2][3] = { 0 };

        int i = 1, j = 2;
        int c = 10, d = 20;

        d = c + a[i][j];

        return 0;
    }

%8 = load i32, i32* %5, align 4 %8:c
%9 = load i32, i32* %3, align 4 %9:i
%10 = sext i32 %9 to i64
%11 = getelementptr inbounds [2 x [3 x i32]], [2 x [3 x i32]]* %2, i64 0, i64 %10
%12 = load i32, i32* %4, align 4 %12:j
%13 = sext i32 %12 to i64
%14 = getelementptr inbounds [3 x i32], [3 x i32]* %11, i64 0, i64 %13
%15 = load i32, i32* %14, align 4 %15: a[i][j]
%16 = add nsw i32 %8, %15
store i32 %16, i32* %6, align 4

```

## The Often Misunderstood GEP Instruction

# 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
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \text{assign}$	$S.code = \text{assign}.code$
$S \rightarrow \text{if } (B) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\quad \parallel label(B.true) \parallel S_1.code$ $\quad \parallel gen('goto' S.next)$ $\quad \parallel label(B.false) \parallel S_2.code$
$S \rightarrow \text{while } (B) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \parallel B.code$ $\quad \parallel label(B.true) \parallel S_1.code$ $\quad \parallel gen('goto' begin)$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$

继承属性  $B.true$ ,  $B.false$ ,  $S.next$  指明了控制流跳转目标

## 继承属性 $S.next$

$P \rightarrow S$

$$\begin{cases} S.next = newlabel() \\ P.code = S.code \parallel label(S.next) \end{cases}$$

$S.next$  为语句  $S$  指明了“跳出” $S$  的目标

## 继承属性 $S.next$

$P \rightarrow S$

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

$S.next$  为语句  $S$  指明了“跳出”  $S$  的目标

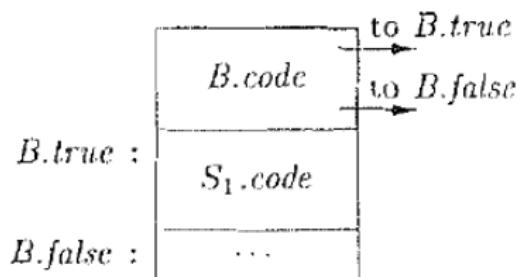
$S \rightarrow \text{assign}$

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

代表了表达式的翻译，包括数组引用

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

$B.\text{true} = \text{newlabel}()$   
 $B.\text{false} = S_1.\text{next} = S.\text{next}$   
 $S.\text{code} = B.\text{code} \parallel \text{label}(B.\text{true}) \parallel S_1.\text{code}$

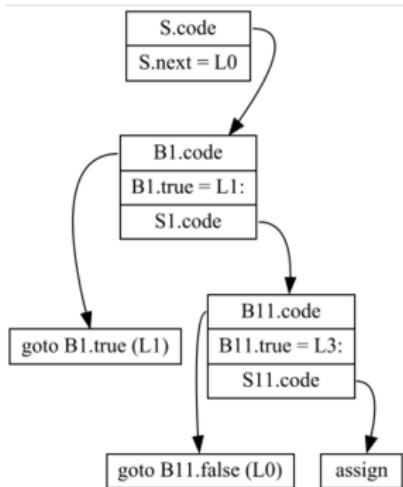


$S \rightarrow \text{if} (B) S_1$

$B.\text{true} = \text{newlabel}()$

$B.\text{false} = S_1.\text{next} = S.\text{next}$

$S.\text{code} = B.\text{code} \parallel \text{label}(B.\text{true}) \parallel S_1.\text{code}$



if (true)  
if (false) assign

$B \rightarrow \text{true}$

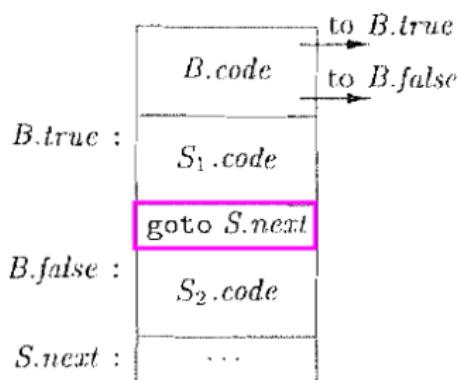
$B.\text{code} = \text{gen('goto' } B.\text{true})$

$B \rightarrow \text{false}$

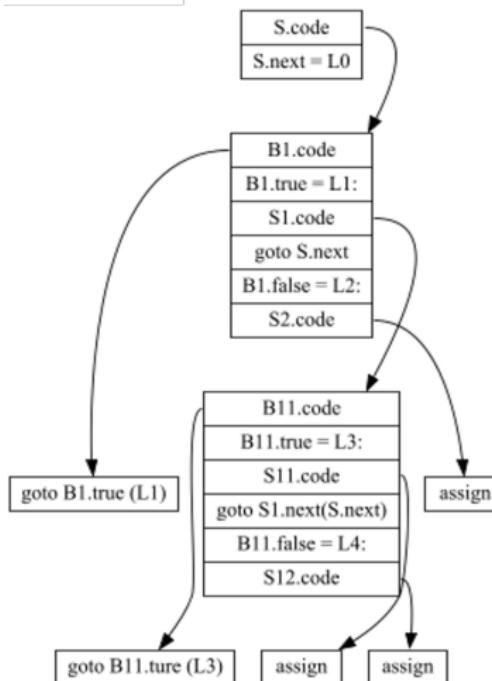
$B.\text{code} = \text{gen('goto' } B.\text{false})$

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

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



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



$B.\text{true} = \text{newlabel}()$

$B.\text{false} = \text{newlabel}()$

$S_1.\text{next} = S_2.\text{next} = S.\text{next}$

$S.\text{code} = B.\text{code}$

$\| \text{label}(B.\text{true}) \| S_1.\text{code}$

$\| \text{gen('goto' } S.\text{next})$

$\| \text{label}(B.\text{false}) \| S_2.\text{code}$

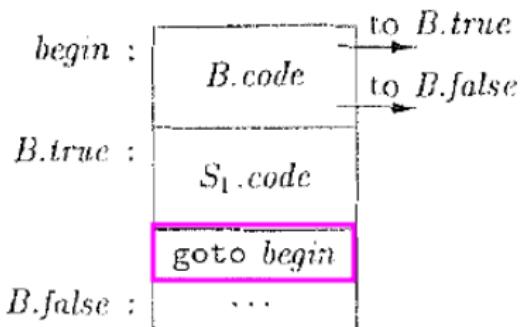
```

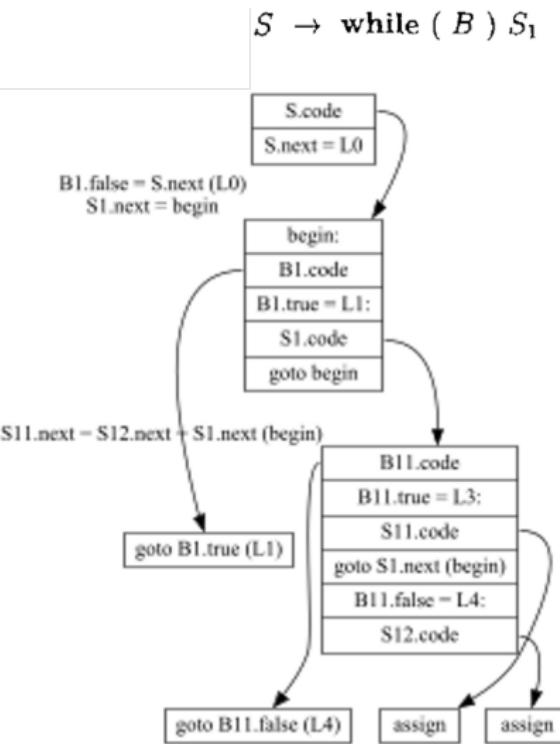
if (true)
  if (true) assign else assign
else
  assign

```

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

$\begin{aligned}begin &= \text{newlabel}() \\B.\text{true} &= \text{newlabel}() \\B.\text{false} &= S.\text{next} \\S_1.\text{next} &= begin \\S.\text{code} &= \text{label}(begin) \parallel B.\text{code} \\&\quad \parallel \text{label}(B.\text{true}) \parallel S_1.\text{code} \\&\quad \parallel \text{gen('goto' begin)}\end{aligned}$





```

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

while (true)  
if (false) assign else assign

$S \rightarrow S_1 S_2$

$S_1.next = newlabel()$

$S_2.next = S.next$

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

$S \rightarrow S_1 S_2$

$S_1.next = newlabel()$

$S_2.next = S.next$

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

if (true) assign else assign assign

# 布尔表达式的中间代码翻译

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

$B \rightarrow \text{true}$

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

$B \rightarrow \text{false}$

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

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

if (true) assign

 $S \rightarrow \text{if ( } B \text{ ) } S_1$  $B.\text{true} = \text{newlabel()}$  $B.\text{false} = S_1.\text{next} = S.\text{next}$  $S.\text{code} = B.\text{code} \parallel \text{label}(B.\text{true}) \parallel S_1.\text{code}$ 

if (false) assign

$B \rightarrow !B_1$

|  $B_1.true = B.false$   
|  $B_1.false = B.true$   
|  $B.code = B_1.code$

$B \rightarrow !B_1$ 
$$\begin{cases} B_1.\text{true} = B.\text{false} \\ B_1.\text{false} = B.\text{true} \\ B.\text{code} = B_1.\text{code} \end{cases}$$

if (!true) assign

 $S \rightarrow \text{if} ( B ) S_1$ 
$$\begin{cases} B.\text{true} = \text{newlabel}() \\ B.\text{false} = S_1.\text{next} = S.\text{next} \\ S.\text{code} = B.\text{code} \parallel \text{label}(B.\text{true}) \parallel S_1.\text{code} \end{cases}$$

if (!false) assign

## 短路求值

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

## 短路求值

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

if (true || false) assign

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

if (false || true) assign

## 短路求值

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

## 短路求值

$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$
	$B_1.false = B.false$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \parallel label(B_1.true) \parallel B_2.code$

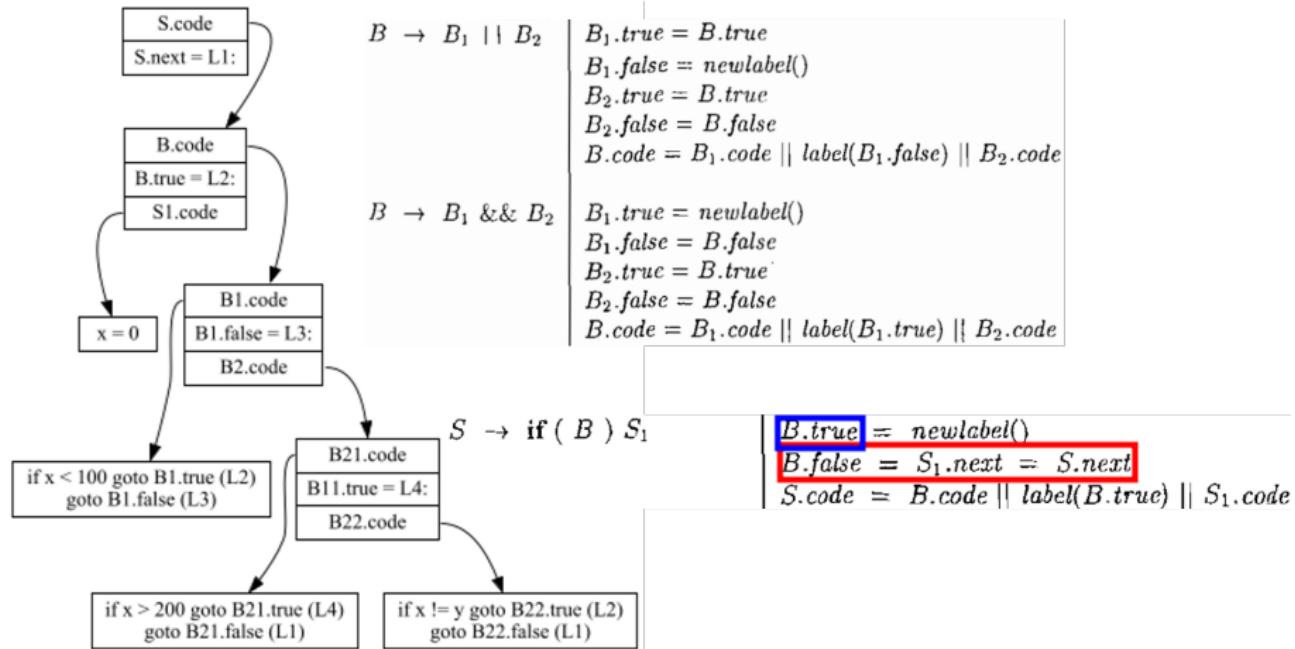
if (true && false) assign

$S \rightarrow \text{if} (B) S_1$	$B.true = newlabel()$
	$B.false = S_1.next = S.next$
	$S.code = B.code \parallel label(B.true) \parallel S_1.code$

if (false && true) assign

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

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



```
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:
```

```
clang -S -emit-llvm if-boolexpr.c -o if-boolexpr-opt0.ll
```

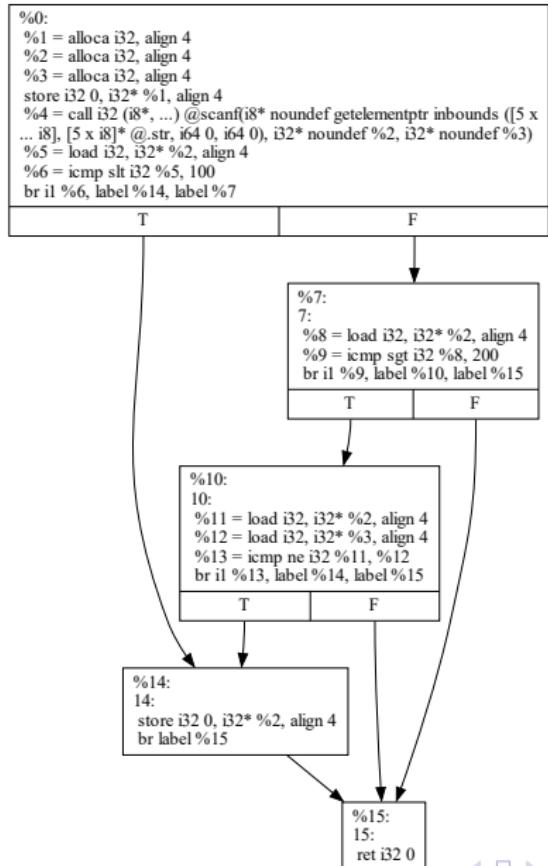
```
int main() {
    int x, y;
    scanf(Format: "%d%d", &x, &y);

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

    return 0;
}
```

```
opt -dot-cfg if-boolexpr-opt0.ll
```

```
dot -Tpdf .main.dot -o if-boolexpr-opt0-cfg.pdf
```



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

$S \rightarrow id = E ;$	{ $gen(top.get(id.lexeme) '==' E.addr);$ }
$  L = E ;$	{ $gen(L.array.base '[' L.addr ']' '==' E.addr);$ }
$E \rightarrow E_1 + E_2$	{ $E.addr = new Temp();$ $gen(E.addr '==' E_1.addr +' E_2.addr);$ }
$  id$	{ $E.addr = top.get(id.lexeme);$ }
$  L$	{ $E.addr = new Temp();$ $gen(E.addr '==' L.array.base '[' L.addr ']');$ }
$L \rightarrow id [ E ]$	{ $L.array = top.get(id.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = new Temp();$ $gen(L.addr '==' E.addr '*' L.type.width);$ }
$  L_1 [ E ]$	{ $L.array = L_1.array;$ $L.type = L_1.type.elem;$ $t = new Temp();$ $L.addr = new Temp();$ $gen(t '==' E.addr '*' L.type.width);$ $gen(L.addr '==' L_1.addr +' t);$ }

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

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

Control.g4

CodeGenListener.java



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

## 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

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

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

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

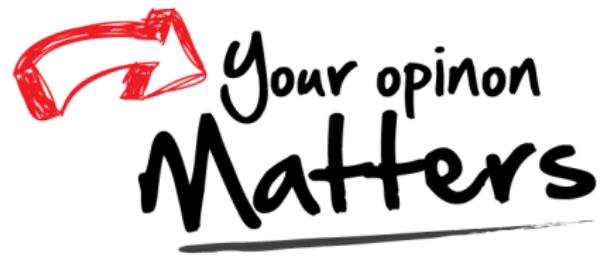
$x = a < b \&\& c < d$

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

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

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

# Thank You!



Office 926

hfwei@nju.edu.cn