



誠朴雄偉
勵學敦行

编译原理习题课（二）

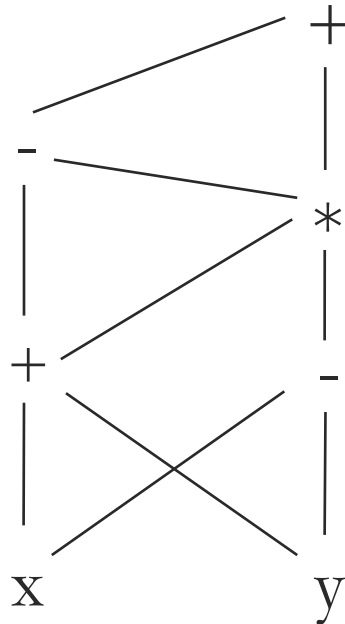


第六章



1. 为下面的表达式构造DFA:

$$((x + y) - ((x + y) * (x - y))) + ((x + y) * (x - y))$$



书6.1.1节
表达式的有向无环图



第六章



2. 将下列将下列赋值语句翻译为四元式序列，三元式序列，间接三元式序列：

$$(2) \ a[i] = b * c - b * d$$

四元式				
0)	*	b	c	t1
1)	*	b	d	t2
2)	-	t1	t2	t3
3)	[]	a	i	t4
4)	=	t3		t4

三元式			
0)	*	b	c
1)	*	b	d
2)	-	(0)	(1)
3)	[]	a	i
4)	=	(3)	(2)

间接三元式 (假设地址为100)			
	0)	*	b c
	1)	*	b d
	2)	-	(0) (1)
	3)	[]	a i
	4)	=	(3) (2)
100	0)		
101	1)		
102	2)		
103	3)		
104	4)		



第六章



3.使用下图所示的翻译方案来翻译赋值语句 $x = a[i][j] + b[i][j]$ 。

```

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); }
    
```

目标：为带数组引用的表达式生成三地址代码

关键字解释：

E.addr: E存放值的地址

gen: 构造一条新的三地址指令，添加至指令序列之后

top: 当前符号表

非终结符号L的三个综合属性：

L.addr: 临时变量，计算数组引用的偏移量

L.array: 指向数组名字的符号表的指针

(*L.array.base*代表基地址)

L.type: L生成子数组的类型

图 6-22 处理数组引用的语义动作



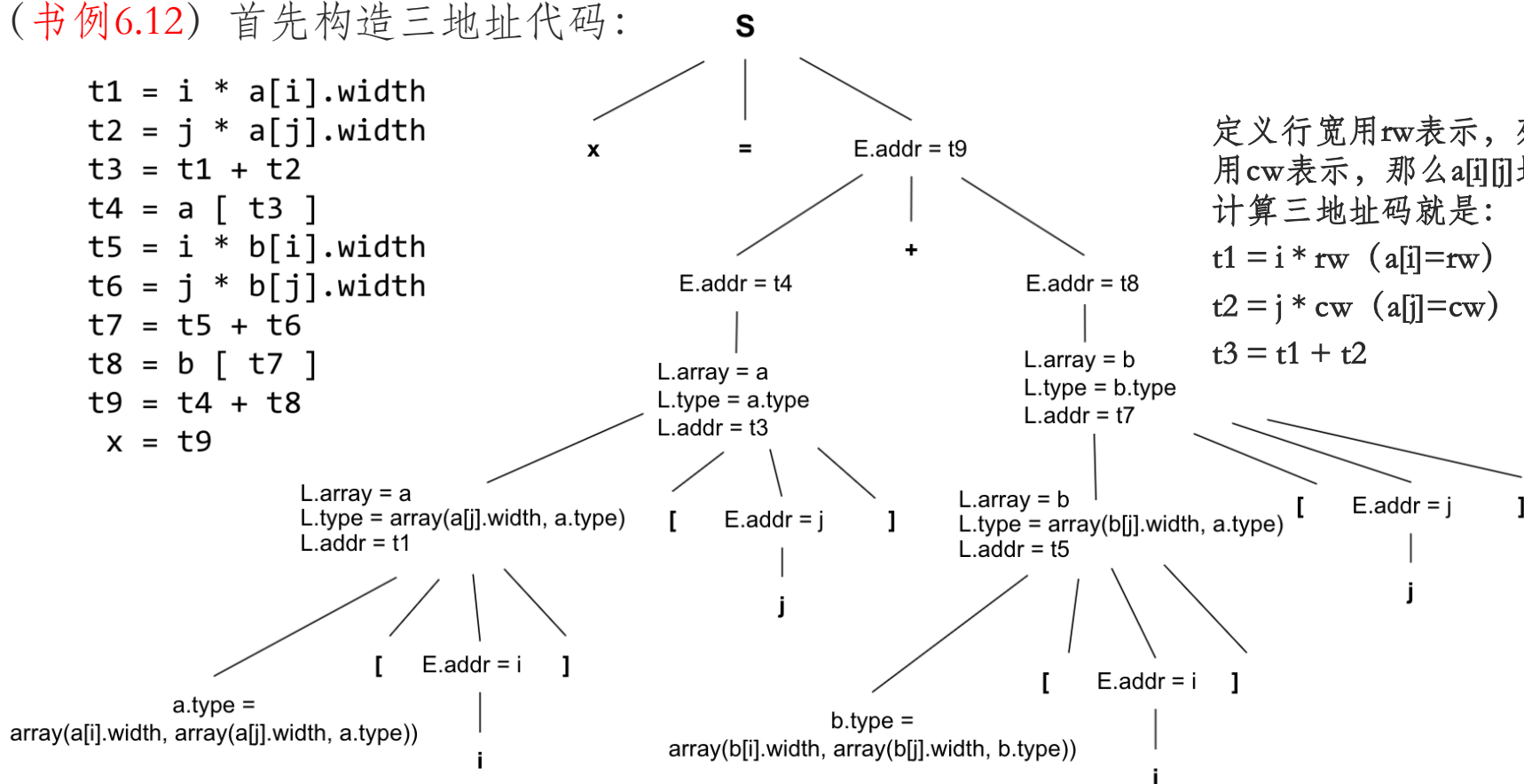
第六章



3.使用下图所示的翻译方案来翻译赋值语句 $x = a[i][j] + b[i][j]$ 。

(书例6.12) 首先构造三地址代码：

```
t1 = i * a[i].width
t2 = j * a[j].width
t3 = t1 + t2
t4 = a [ t3 ]
t5 = i * b[i].width
t6 = j * b[j].width
t7 = t5 + t6
t8 = b [ t7 ]
t9 = t4 + t8
x = t9
```



注释语法分析树



第六章



4. 一个按行存放的实数型数组 $A[i, j, k]$ 的下标 i 的范围为 $1 \sim 4$ ，下标 j 的范围为 $0 \sim 4$ ，且下标 k 的范围为 $5 \sim 10$ 。每个实数占 8 个字节。假设数组 A 从 0 字节开始存放，计算下列元素的位置：

(1) $A[3, 4, 5]$

(书 6.4.3) 数组在 $A[i]$ 上的宽度为 5×6 ，在 $A[i, j]$ 上的宽度为 5，根据公式 6.3 和 6.7，元素位置为： $((3 - 1) \times 5 \times 6 + (4 - 0) \times 6 + (5 - 5)) \times 8 = 672$

(2) $A[1, 2, 7]$

元素位置为： $((1 - 1) \times 5 \times 6 + (2 - 0) \times 6 + (7 - 5)) \times 8 = 112$

(3) $A[4, 3, 9]$

元素位置为： $((4 - 1) \times 5 \times 6 + (3 - 0) \times 6 + (9 - 5)) \times 8 = 896$



第六章



5. 使用下图中的翻译方案翻译表达式 $a == b \ \&\& \ (c == d \ || \ e == f)$ ，并给出每个子表达式的真值列表与假值列表，你可以假设第一条被生成的指令的地址是100:

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) $B \rightarrow E_1 \ rel \ E_2$	{ $B.truelist = makelist(nextinstr);$ $B.falselist = makelist(nextinstr + 1);$ $gen('if' E_1.addr \ rel.op \ E_2.addr 'goto -');$ $gen('goto -');$ }
6) $B \rightarrow true$	{ $B.truelist = makelist(nextinstr);$ $gen('goto -');$ }
7) $B \rightarrow false$	{ $B.falselist = makelist(nextinstr);$ $gen('goto -');$ }
8) $M \rightarrow \epsilon$	{ $M.instr = nextinstr; \}$

目标：为布尔表达式生成三地址代码

方法：回填完成一趟式目标代码生成

B有两个综合属性：

两个list都是包含跳转指令的列表

$B.truelist$ ：B为真时控制流应该转向的标号

$B.falselist$ ：B为假时控制流应该转向的标号

构建这个列表需要三个函数：

(1)makelist(i): 创建一个只包含指令数组下标i的列表，返回一个新列表的指针

(2)merge(p1, p2): 将p1和p2指向的列表进行合并，返回合并列表的指针

(3)backpatch(p, i): 将i作为目标标号插入p列表

图 6-43 布尔表达式的翻译方案

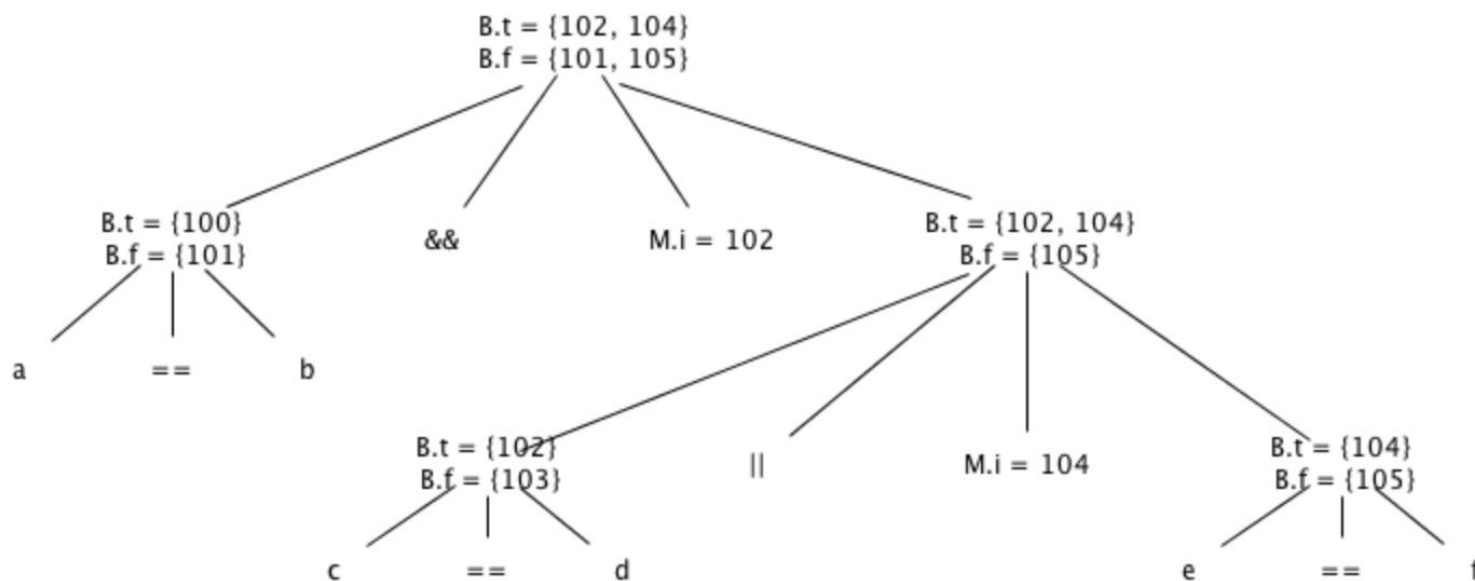


第六章



5. 使用下图中的翻译方案翻译表达式 $a == b \ \&\& \ (c == d \ || \ e == f)$ ，并给出每个子表达式的真值列表与假值列表，你可以假设第一条被生成的指令的地址是100:

书6.7.2 节 例6.17





第八章

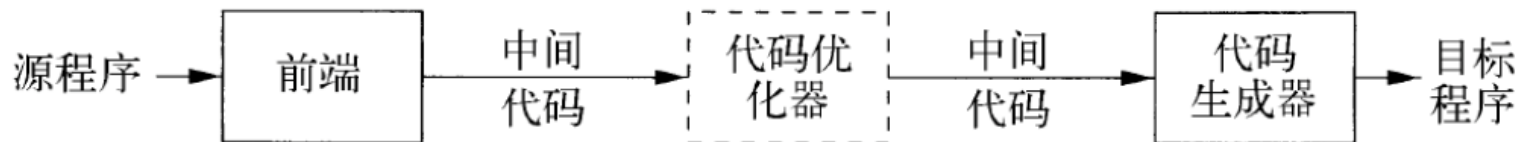


图 8-1 代码生成器的位置

- 根据中间表示生成代码
- 代码生成器之前可能有一个优化组件
- 代码生成器的三个主要任务
 - 指令选择：选择适当的指令实现IR语句
 - 寄存器分配和指派：把哪个值放在哪个寄存器中
 - 指令排序：按照什么顺序安排指令执行



第八章



1. 假设a和b是元素为4字节值的数组，为下面的三地址语句序列生成代码：

(1) 四个语句的序列

$x = a[i]$

$y = b[j]$

$a[i] = y$

$b[j] = x$

答案：(不唯一)

LD R1, i

MUL R1, R1, #4

LD R2, a(R1)

LD R3, j

MUL R3, R3, #4

LD R4, b(R3)

ST a(R1), R4

ST b(R3), R2



第八章



1. 假设a和b是元素为4字节值的数组，为下面的三地址语句序列生成代码：

(2) 三个语句的序列

```
x = a[i]
y = b[i]
z = x * y
```

答案：(不唯一)

```
LD  R1, i
MUL R1, R1, #4
LD  R2, a(R1)
LD  R3, b(R1)
MUL R4, R2 * R3
ST  z, R4
```



第八章



2. 假设x、y和z存放在内存位置中，为下面的三地址语句序列生成代码：

if x < y goto L1

z = 0

goto L2

L1: z = 1

答案：(不唯一)

LD R1, x

LD R2, y

SUB R1, R1, R2

BLTZ R1, L1

LD R3, #0

ST z, R3

BR L2

L1: LD R4, #1

ST z, R4



第八章



3. 假设使用栈式分配而寄存器SP指向栈的顶端，为下列的三地址语句生成代码：

```
call p
call q
return
call r
return
return
```

答案：

```
// 假设起始位置是100，过程p,q,r的代码分别从地址200，400，600开始
100: LD SP, #stackStart
108: ADD SP, SP, #psize
116: ST 0(SP), #132
124: BR 200
132: SUB SP, SP, #psize
140: ADD SP, SP, #qsize
148: ST 0(SP), #164
156: BR 400
164: SUB SP, SP, #qsize
172: BR *0(SP)
180: ADD SP, SP, #rsize
188: ST 0(SP), #204
196: BR 600
204: SUB SP, SP, #rsize
212: BR *0(SP)
220: BR *0(SP)
```



第八章



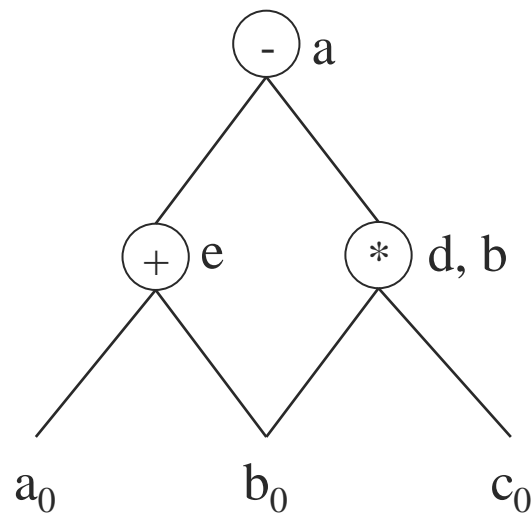
4. 为下面的基本块构造DAG:

$$d = b * c$$

$$e = a + b$$

$$b = b * c$$

$$a = e - d$$





第八章



4. 为下面的基本块构造DAG:

$$d = b * c$$

$$e = a + b$$

$$b = b * c$$

$$a = e - d$$

分别按照下列两种假设简化三地址代码:

(1) 只有a在基本块的出口处活跃。

$$d = b * c$$

$$e = a + b$$

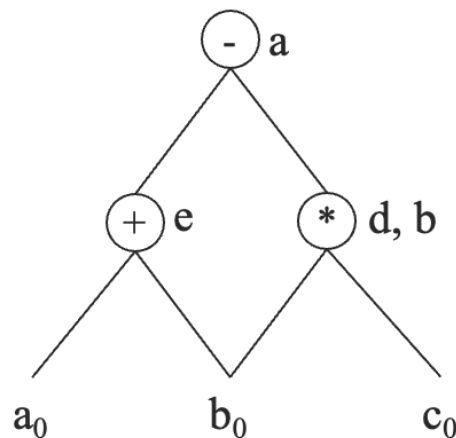
$$a = e - d$$

(2) a、b、c在基本块的出口处活跃。

$$e = a + b$$

$$b = b * c$$

$$a = e - b$$





第八章



5. 为下面的每个C语言赋值语句生成三地址代码：

(1) $x = a + b * c;$

// 先把代码转化为三地址语句

$t = b * c$

$u = a + t$

$x = u$

// 再转化为指令代码

LD R1, b

LD R2, c

MUL R1, R1, R2 //存R1或R2都可以，后面不引用了

LD R3, a

ADD R1, R1, R3

ST x, R1



第八章



5. 为下面的每个C语言赋值语句生成三地址代码:

(2) $x = a[j] + 1;$

// 假设数组长度为4

$t = a[j]$

$u = t + 1$

$x = u$

//

LD R1, j

MUL R1, R1, 4

LD R2, a(R1)

ADD R2, R2, 1

ST x, R2



■ 代码优化或者代码改进

- 在目标代码中消除不必要的指令
- 把一个指令序列替换为一个完成相同功能的更快的指令序列

■ 全局优化

- 具体的优化实现基于数据流分析技术
- 用以收集程序相关信息的算法



第九章



1. 对于图9-10中的流图:

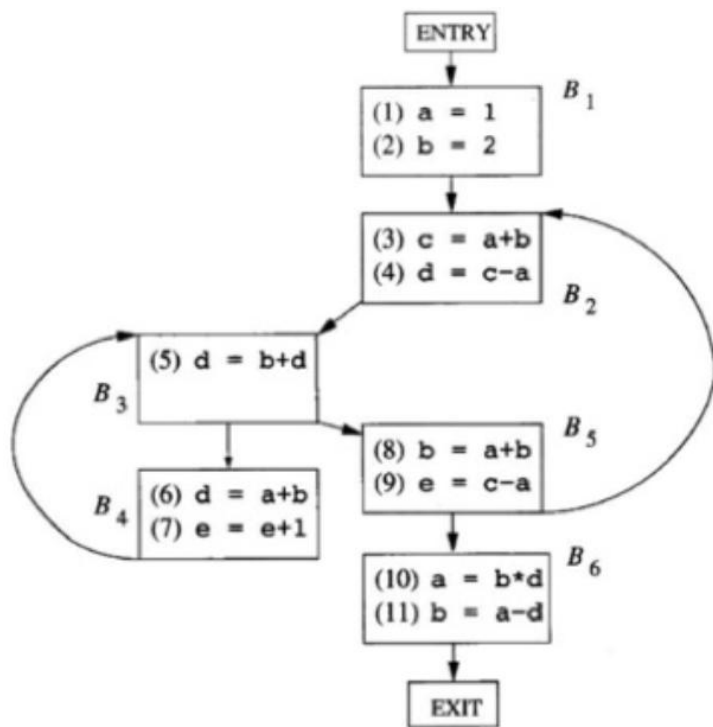
(1) 找出流图中的循环。

(2) B1中的语句(1)和(2)都是复制语句。其中a和b都被赋予了常量值。我们可以对a和b的哪些使用进行复制传播, 并把对它们的使用替换为对一个常量的使用?

(3) 对每个循环, 找出所有的全局公共子表达式。

语义不变的优化方法

- 公共子表达式消除 (题1)
- 复制传播 (题1)
- 死代码消除 (通常在复制传播优化后)
- 常量折叠
- 代码移动
- 归纳变量和强度消减 (题2)





第九章

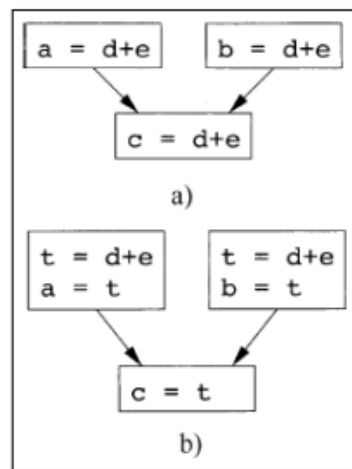


语义不变的优化方法

➤ 复制传播

➤ 死代码消除

- 形如 $u = v$ 的复制语句使得语句后面的程序点上， u 的值等于 v 的值
 - 如果在某个位置上 u 一定等于 v ，那么可以把 u 替换为 v
 - 有时可以彻底消除对 u 的使用，从而消除对 u 的赋值语句
- 右图所示，消除公共子表达式时引入了复制语句
- 如果尽可能用 t 来替换 c ，可能就不需要 $c = t$ 这个语句了





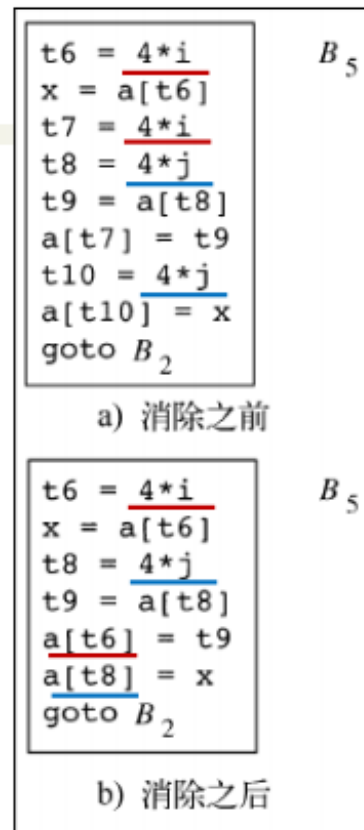
第九章



语义不变的优化方法

公共子表达式消除

- 如果 E
 - 在某次出现之前必然已被计算过，且
 - E 的运算分量在该次计算之后没有被改变
 - 那么， E 的本次出现就是一个公共子表达式
- 如果上一次 E 的值赋给了 x ，且 x 的值没有被修改过，那么我们可以使用 x ，而不需要计算 E



$t7 \Rightarrow t6$
 $t10 \Rightarrow t8$

由于 $4*i$ 和 $4*j$ 的值没变
无需重复计算



第九章



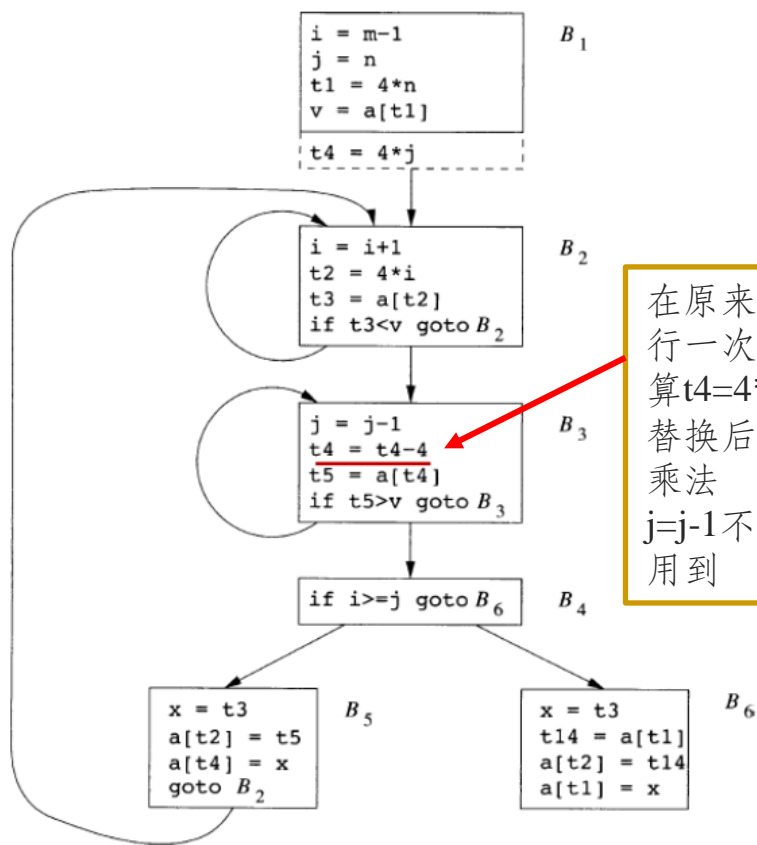
语义不变的优化方法

➤ 归纳变量和强度消减

■ 归纳变量

- 每次对 x 赋值都使 x 增加 c
- 把赋值改为增量操作，可消减计算强度
- 两个归纳变量步调一致，可删除一个
- 例子

- 循环开始保持 $t4 = 4 * j$
- $j = j - 1$ 后面的 $t4 = 4 * j$ 每次赋值使 $t4$ 减4
- 可替换为 $t4 = t4 - 4$
- $t2$ 也可同样处理



在原来的块B3中，循环每执行一次都要进行一次乘法运算 $t4 = 4*j$
替换后只需在B1中运行一次乘法
 $j = j - 1$ 不能删除，因为后续会用到

图 9-8 对基本块 B_3 中的 $4 * j$ 应用强度消减优化



第九章



1. 对于图9-10中的流图:

(1) 找出流图中的循环。

B3-B4

B2-B3-B5

B2-B3-B4-B5

(2) B1中的语句(1)和(2)都是复制语句。其中a和b都被赋予了常量值。我们可以对a和b的哪些使用进行复制传播, 并把对它们的使用替换为对一个常量的使用?

B2: 将语句(3)替换为 $c=1+b$

将语句(4)替换为 $d=c-1$

B4: 将语句(6)替换为 $d=1+b$

B5: 将语句(8)替换为 $b=1+b$

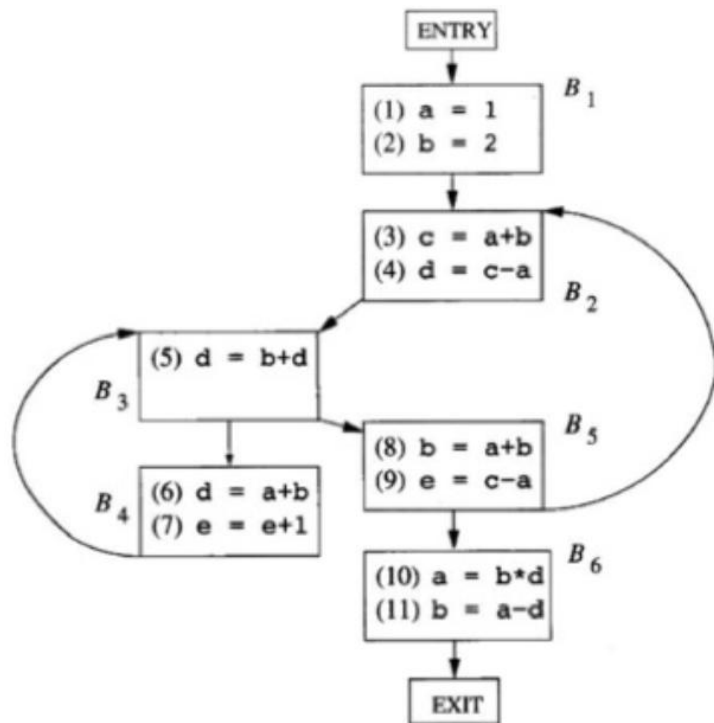
将语句(9)替换为 $e=c-1$

(3) 对每个循环, 找出所有的全局公共子表达式。

B3-B4: 无

B2-B3-B5: $a+b$ $c-a$

B2-B3-B4-B5: 同上





第九章



2. 下图是用来计算两个向量A和B的点积的中间代码。尽你所能，通过下列方式优化这个代码：消除公共子表达式，对归纳变量进行强度消减，消除归纳变量。

```
dp = 0.  
i = 0  
L: t1 = i*8  
t2 = A[t1]  
t3 = i*8  
t4 = B[t3]  
t5 = t2*t4  
dp = dp+t5  
i = i+1  
if i<n goto L
```

消除公共子表达式：

删除 $t3=i*8$

$t4=B[t3]$ 改为 $t4=B[t1]$

强度消减，消除归纳变量（乘法改为加法）：

把 $t1=i*8$ 插入到循环之前

把 $t1=t1+8$ 插入到if语句之前



第九章



■ 数据流分析

- 用于获取数据沿着程序执行路径流动信息的相关技术
- 是优化的基础

■ 例如

- 两个表达式是否一定计算得到相同的值？（公共子表达式）可用表达式分析
- 一个语句的计算结果是否可能被后续语句使用？（死代码消除）活跃变量分析



第九章



	到达定值	活跃变量	可用表达式
域	Sets of definitions	Sets of variables	Sets of expressions
方向	Forwards	Backwards	Forwards
传递函数	$gen_B \cup (x - kill_B)$	$use_B \cup (x - def_B)$	$e_gen_B \cup (x - e_kill_B)$
边界条件	$OUT[ENTRY] = \emptyset$	$IN[EXIT] = \emptyset$	$OUT[ENTRY] = \emptyset$
交汇运算(\wedge)	\cup	\cup	\cap
方程组	$OUT[B] = f_B(IN[B])$ $IN[B] = \bigwedge_{P, pred(B)} OUT[P]$	$IN[B] = f_B(OUT[B])$ $OUT[B] = \bigwedge_{S, succ(B)} IN[S]$	$OUT[B] = f_B(IN[B])$ $IN[B] = \bigwedge_{P, pred(B)} OUT[P]$
初始值	$OUT[B] = \emptyset$	$IN[B] = \emptyset$	$OUT[B] = U$

三种数据流方程的总结

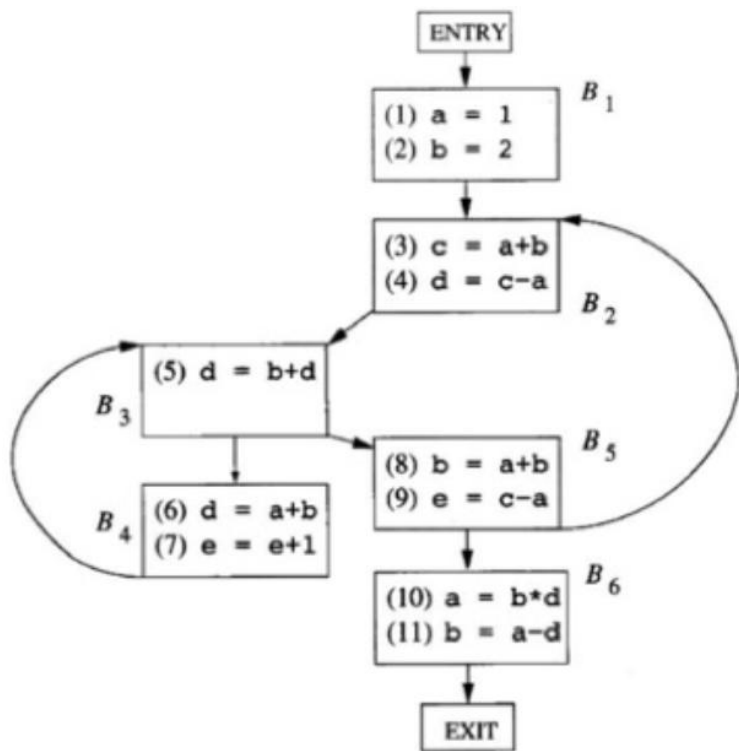


第九章



3. 对图9-10中的流图，计算下列值：

- (1) 每个基本块的gen和kill集合。
- (2) 每个基本块的IN和OUT集合。



■ 到达定值

- 假定 x 有定值 d ，如果存在一个路径，从紧随 d 的点到达某点 p ，并且此路径上面没有 x 的其他定值点，则称 x 的定值 d 到达 p
- 如果在这条路径上有对 x 的其它定值，我们说变量 x 的这个定值 d 被杀死了

Block	<i>gen</i>	<i>kill</i>
B1	$d1, d2$	$d8, d10, d11$
B2	$d3, d4$	$d5, d6$
B3	$d5$	$d4, d6$
B4	$d6, d7$	$d4, d5, d9$
B5	$d8, d9$	$d2, d7, d11$
B6	$d10, d11$	$d1, d2, d8$



第九章

3. 对图9-10中的流图，计算下列值：

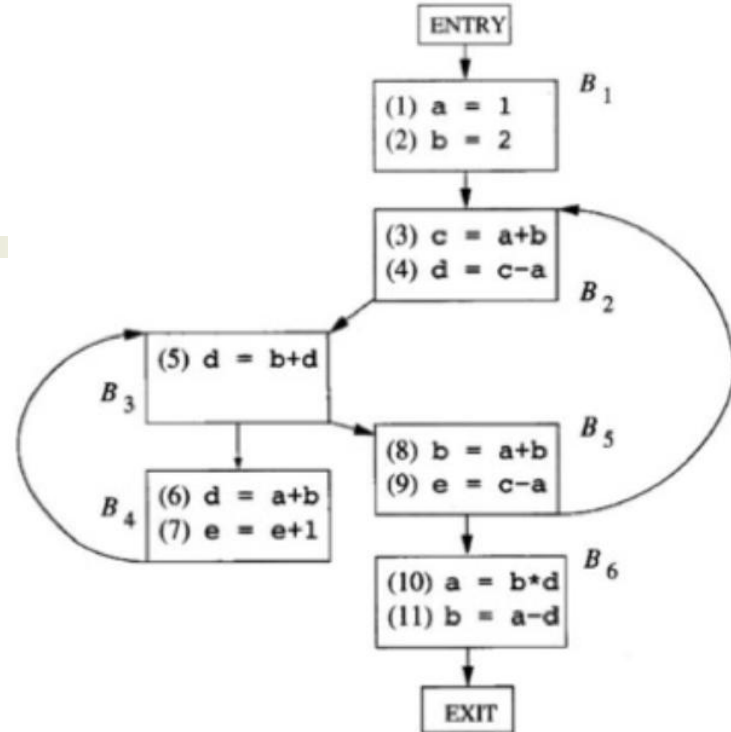
- (1) 每个基本块的gen和kill集合。
- (2) 每个基本块的IN和OUT集合。

- 输入：流图、各基本块的*kill*和*gen*集合
- 输出：IN[B]和OUT[B]
- 方法

```

1) OUT[ENTRY] = ∅;
2) for (除 ENTRY 之外的每个基本块 B) OUT[B] = ∅;
3) while (某个 OUT 值发生了改变)
4)     for (除 ENTRY 之外的每个基本块 B) {
5)         IN[B] =  $\bigcup_{P \text{ 是 } B \text{ 的一个前驱}} \text{OUT}[P]$ ;
6)         OUT[B] =  $\text{gen}_B \cup (\text{IN}[B] - \text{kill}_B)$ ;
    }

```



Block	<i>gen</i>	<i>kill</i>
B1	<i>d1, d2</i>	<i>d8, d10, d11</i>
B2	<i>d3, d4</i>	<i>d5, d6</i>
B3	<i>d5</i>	<i>d4, d6</i>
B4	<i>d6, d7</i>	<i>d4, d5, d9</i>
B5	<i>d8, d9</i>	<i>d2, d7, d11</i>
B6	<i>d10, d11</i>	<i>d1, d2, d8</i>



第九章

3. 对图9-10中的流图，计算下列值：

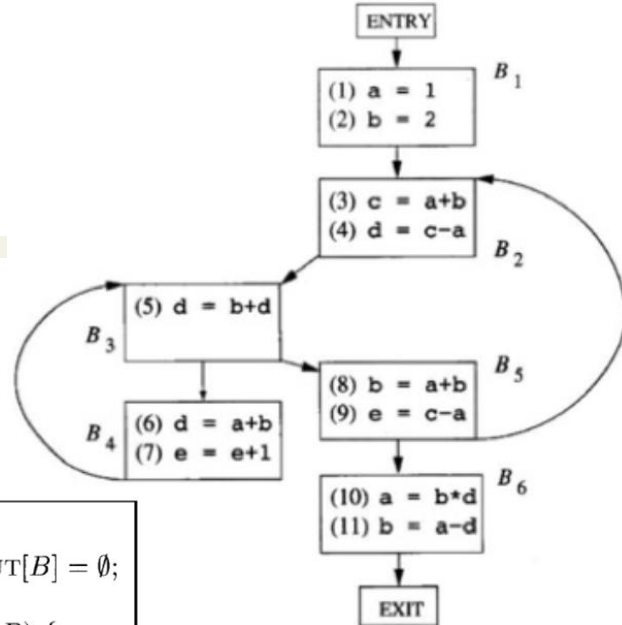
- (1) 每个基本块的gen和kill集合。
- (2) 每个基本块的IN和OUT集合。

Block	gen	kill
B1	d1, d2	d8, d10, d11
B2	d3, d4	d5, d6
B3	d5	d4, d6
B4	d6, d7	d4, d5, d9
B5	d8, d9	d2, d7, d11
B6	d10, d11	d1, d2, d8

```

1) OUT[ENTRY] = ∅;
2) for (除 ENTRY 之外的每个基本块 B) OUT[B] = ∅;
3) while (某个 OUT 值发生了改变)
4)   for (除 ENTRY 之外的每个基本块 B) {
5)     IN[B] =  $\bigcup_{P \text{ 是 } B \text{ 的一个前驱}} \text{OUT}[P]$ ;
6)     OUT[B] =  $\text{gen}_B \cup (\text{IN}[B] - \text{kill}_B)$ ;
   }

```



$$IN[B_2]^1 = OUT[B_1]^1 \cup OUT[B_5]^0 = 110\ 0000\ 0000 + 000\ 0000\ 0000 = 110\ 0000\ 0000$$

$$OUT[B_2]^1 = \text{gen}_{B_2} \cup (IN[B_2]^1 - \text{kill}_{B_2}) = 001\ 1000\ 0000 + (110\ 0000\ 0000 - 000\ 0110\ 0000) = 111\ 1000\ 0000$$

Block	OUT[B] ⁰	IN[B] ¹	OUT[B] ¹	IN[B] ²	OUT[B] ²
B1	0000000000	0000000000	1100000000	0000000000	1100000000
B2	0000000000	1100000000	1111000000	1110100110	1111000110
B3	0000000000	1111000000	1110100000	1111011110	1110101110
B4	0000000000	1110100000	1110011000	1110101110	1110011100
B5	0000000000	1110100000	1010100110	1110101110	1010100110
B6	0000000000	1010100110	0010100011	1010100110	0010100011

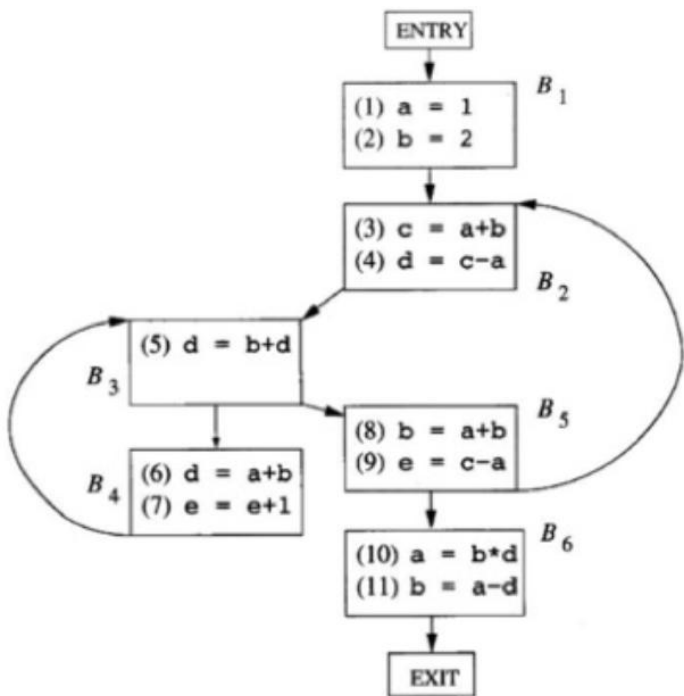
第三轮迭代的结果与第二轮相同，因此第二轮迭代后算法终止



第九章



4. 对图9-10的流图，计算活跃变量分析中的def、use、IN和OUT集合。



■ 活跃变量分析

- x 在 p 上的值是否会在某条从 p 出发的路径中使用
- 变量 x 在 p 上活跃，当且仅当存在一条从 p 开始的路径，该路径的末端使用了 x ，且路径上没有对 x 进行覆盖

■ 基本块的传递函数仍然是生成-杀死形式，但是从OUT值计算出IN值 (逆向)

- use_B : 在 B 中先于定值被使用
- def_B : 在 B 中先于使用被定值

```

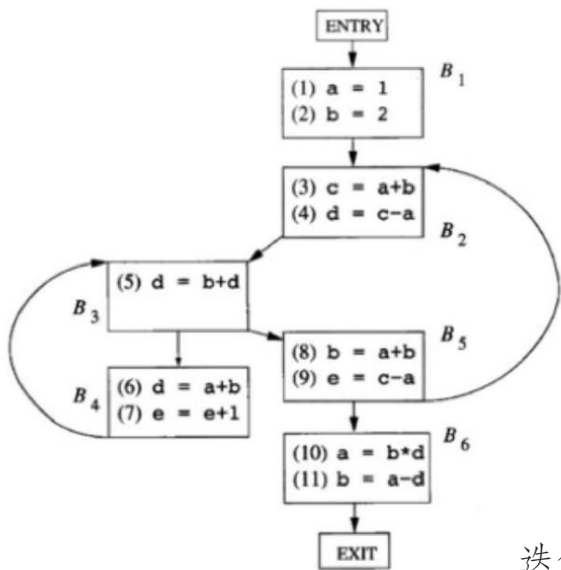
IN[EXIT] = ∅;
for (除 EXIT 之外的每个基本块 B) IN[B] = ∅;
while (某个 IN 值发生了改变)
    for (除 EXIT 之外的每个基本块 B) {
        OUT[B] =  $\bigcup_{S \text{ 是 } B \text{ 的一个后继}} IN[S]$ ;
        IN[B] =  $use_B \cup (OUT[B] - def_B)$ ;
    }
    
```



第九章



4. 对图9-10的流图，计算活跃变量分析中的def、use、IN和OUT集合。



- 基本块的传递函数仍然是生成-杀死形式，但是从OUT值计算出IN值(逆向)
- use_B : 在B中先于定值被使用
- def_B : 在B中先于使用被定值

```

IN[EXIT] =  $\emptyset$ ;
for (除 EXIT 之外的每个基本块 B) IN[B] =  $\emptyset$ ;
while (某个 IN 值发生了改变)
    for (除 EXIT 之外的每个基本块 B) {
        OUT[B] =  $\bigcup_{S \text{ 是 } B \text{ 的一个后继}} IN[S]$ ;
        IN[B] =  $use_B \cup (OUT[B] - def_B)$ ;
    }

```

迭代开始: $IN[B6]^1 = \{b, d\} \cup (\emptyset - \{a\})$

Block	def	use	IN[B] ⁰	IN[B] ¹	OUT[B] ¹	IN[B] ²	OUT[B] ²
B1	a,b	\emptyset	\emptyset	e	a,b,e	e	a,b,e
B2	c,d	a,b	\emptyset	a,b,e	a,b,c,d,e	a,b,e	a,b,c,d,e
B3	\emptyset	b,d	\emptyset	a,b,c,d,e	a,b,c,d,e	a,b,c,d,e	a,b,c,d,e
B4	d	a,b,e	\emptyset	a,b,e	\emptyset	a,b,c,e	a,b,c,d,e
B5	e	a,b,c	\emptyset	a,b,c,d	b,d	a,b,c,d	a,b,d,e
B6	a	b,d	\emptyset	b,d	\emptyset	b,d	\emptyset

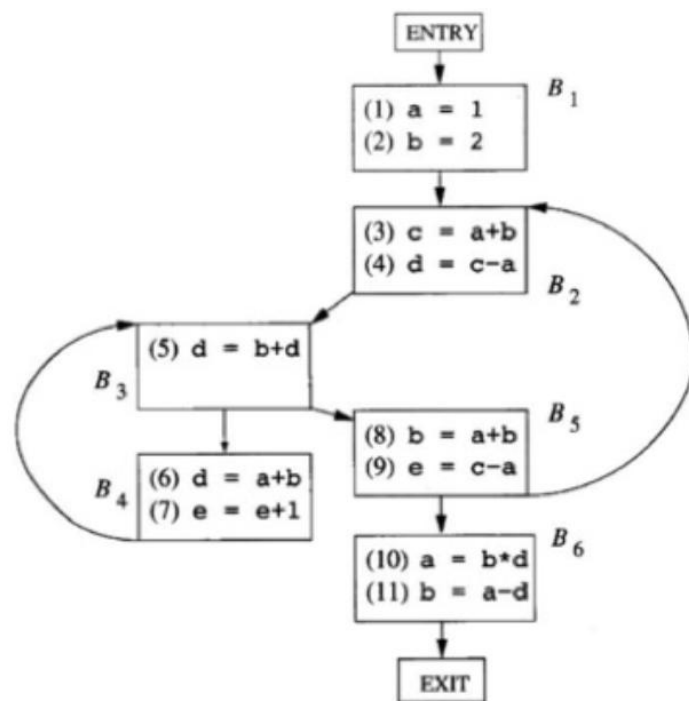


第九章



5. 对图9-10的流图：

- (1) 计算支配关系。
- (2) 寻找每个结点的直接支配结点。
- (3) 构造支配结点树。
- (4) 找出该流图的一个深度优先排序。
- (5) 指明(4)中的前进、后退和交叉边以及树的边。
- (6) 这个流图是可规约的吗？
- (7) 计算这个流图的深度。
- (8) 找出这个流图的自然循环。





第九章



5. 对图9-10的流图：

(1) 计算支配关系。

$D(B1) = \{B1\}$

$D(B2) = \{B1, B2\}$

$D(B3) = \{B1, B2, B3\}$

$D(B4) = \{B1, B2, B3, B4\}$

$D(B5) = \{B1, B2, B3, B5\}$

$D(B6) = \{B1, B2, B3, B5, B6\}$

支配 (Dominate)

- 如果每条从入口结点到达 n 的路径都经过 d ，那么 d 支配 n ，记为 $d \text{ dom } n$

直接支配结点

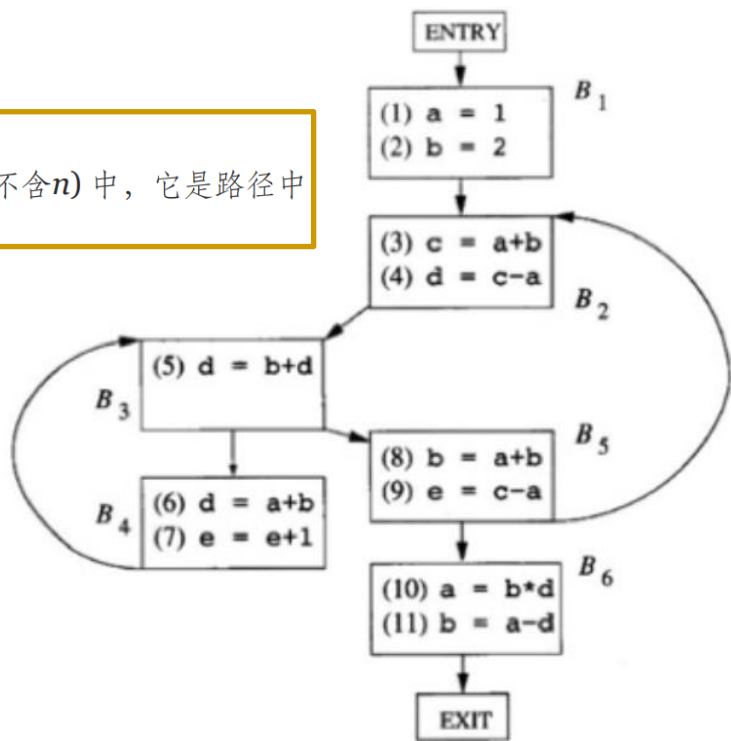
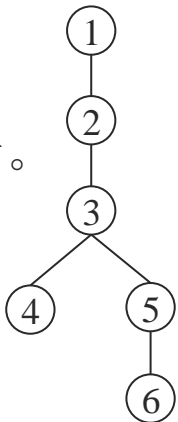
- 从入口结点到达 n 的任何路径 (不含 n) 中，它是路径中最后一个支配 n 的结点

(2) 寻找每个结点的直接支配结点。

$B1$: 无 $B2$: $B1$ $B3$: $B2$

$B4$: $B3$ $B5$: $B3$ $B6$: $B5$

(3) 构造支配结点树。

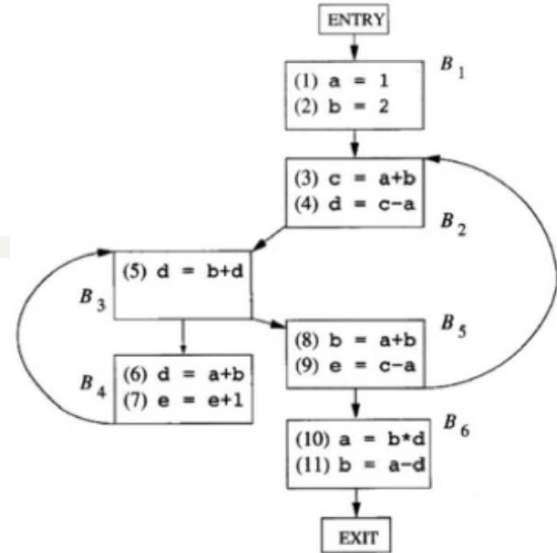




第九章

5. 对图9-10的流图：

- (4) 找出该流图的一个深度优先排序。
- (5) 指明(4)中的前进、后退和交叉边以及树的边。
- (6) 这个流图是可规约的吗？
- (7) 计算这个流图的深度。
- (8) 找出这个流图的自然循环。



- 为一个流图构造出DFST之后，流图中的边可以分为三类
 - 前进边：从结点 m 到达 m 在DFST树中的一个真后代结点的边 (DFST中的所有边都是前进边)
 - 后退边：从 m 到达 m 在DFST树中的某个祖先 (包括 m) 的边
 - 交叉边：边的 src 和 $dest$ 都不是对方的祖先

- 回边
 - 边 $a \rightarrow b$ ，头 b 支配了尾 a
 - 每条回边都是后退边，但不是所有后退边都是回边
- 如果一个流图的任何深度优先生成树中的所有后退边都是回边，那么该流图就是可归约的
 - 可归约流图的DFST的后退边集合就是回边集合
 - 不可归约流图的DFST中可能有一些后退边不是回边



第九章



5. 对图9-10的流图：

(4) 找出该流图的一个深度优先排序。

B1-B2-B3-B4-B5-B6

(5) 指明(4)中的前进、后退和交叉边以及树的边。

前进边: B1-B2 B2-B3 B3-B4 B3-B5 B5-B6

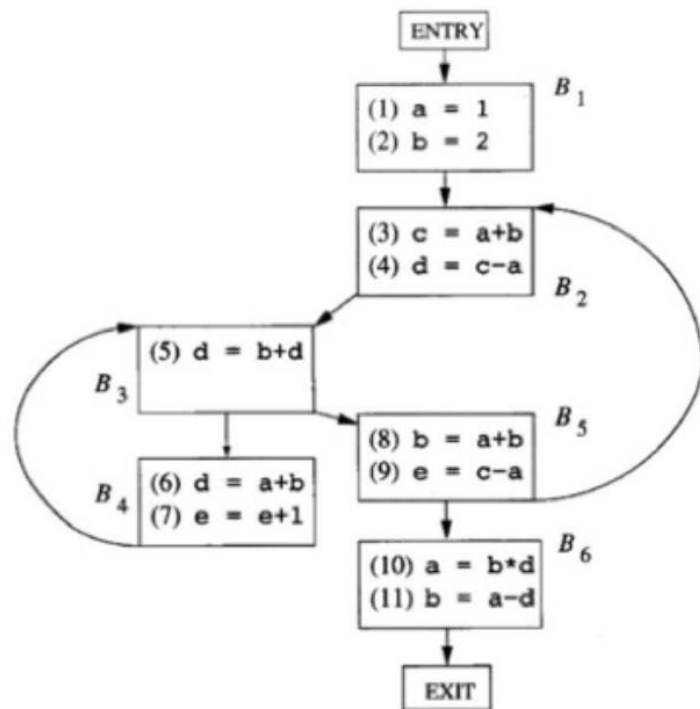
后退边: B4-B3 B5-B2

交叉边: 无

树(DFST)的边: 所有前进边

(6) 这个流图是可规约的吗?

是可归约的, 因为后退边B4-B3, B5-B2都是回边

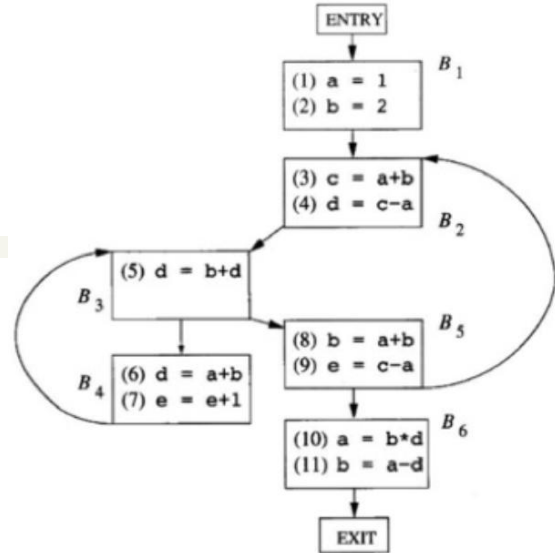




第九章

5. 对图9-10的流图：

- (4) 找出该流图的一个深度优先排序。
- (5) 指明(4)中的前进、后退和交叉边以及树的边。
- (6) 这个流图是可规约的吗？
- (7) 计算这个流图的深度。
- (8) 找出这个流图的自然循环。



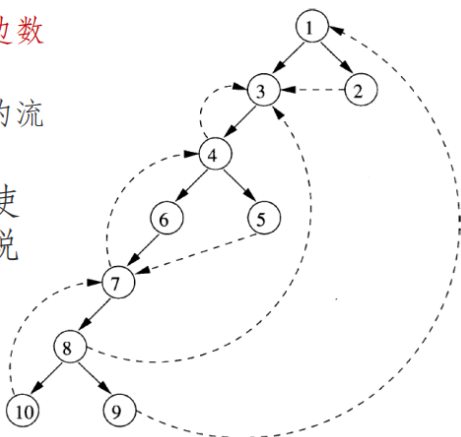
■ 流图相对于DFST的深度

- 各条无环路径上后退边数中的最大值
- 不会大于直观上所说的流图中的循环嵌套深度

■ 对可归约的流图，可使用回边来定义，且可说是流图的深度

■ 右边的流图深度为3

- $10 \rightarrow 7 \rightarrow 4 \rightarrow 3$



自然循环构造算法

- 输入：流图 G 和回边 $n \rightarrow d$
- 输出：给定回边 $n \rightarrow d$ 的自然循环中的所有结点的集合 $loop$
- 方法
 - $loop = \{n, d\}$, d 标记为visited
 - 从 n 开始，逆向对流图进行深度优先搜索，把所有访问到的结点都加入 $loop$ ，加入 $loop$ 的结点都标记为visited
 - 搜索过程中，不越过标记为visited的结点



第九章



5. 对图9-10的流图：

(7) 计算这个流图的深度。

深度为2，存在无环路径B4-B3-B5-B2，其中有两边后退边

(8) 找出这个流图的自然循环。

对于回边B4-B3，自然循环为{B4, B3}

对于回边B5-B2，自然循环为{B2, B3, B4, B5}

