



山东大学  
SHANDONG UNIVERSITY

编译原理

# 第十章 优化

授 课 教 师 : 余仲星  
手 机 : 15866821709 (微信同号)  
邮 箱 : zhongxing.yu@sdu.edu.cn

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

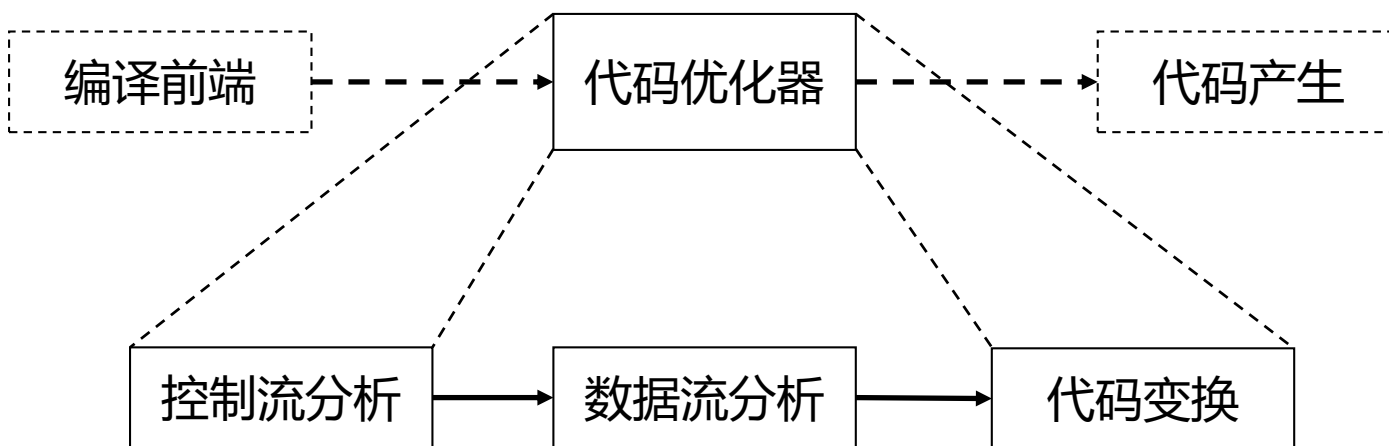
## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

# 第十章 优化

□ **优化**：指对程序进行等价变换，使得从变换后的程序出发，能生成更有效的目标代码。

- **前端优化**：在目标代码生成以前，对语法分析后的目标代码进行优化。
- **后端优化**：在生成目标代码时进行优化，依赖于具体的计算机指令系统。



# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

# 10.1 概述

## □ 优化原则

- 等价原则：经过优化的代码不应改变程序运行的结果。
- 有效原则：使优化后所产生的目标代码运行时间较短，占用的存储空间较小。
- 合算原则：应尽可能以较低的代价取得较好的优化效果。

# 10.1 概述

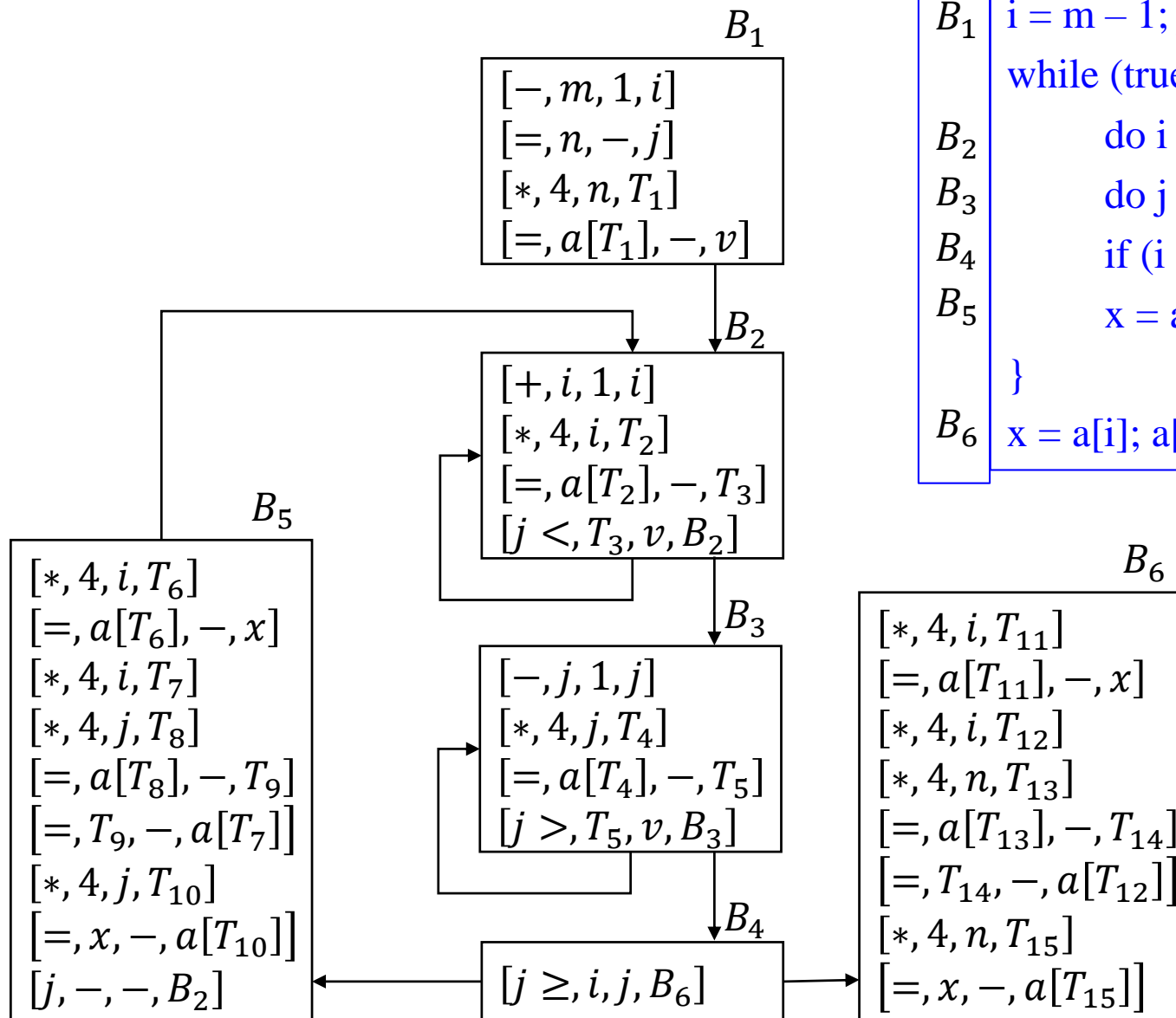
## □ 优化环节

- 源代码级别：选择适当的算法，比如排序算法中，“快排”比“插排”快。
- 语义动作级别：
  - 生成更高效的中间代码。
  - 加入对优化的预备工作。如循环的开头和结尾处打上标记，方便控制流和数据流分析；代码分叉和交汇处打上标记，方便识别流程图中的直接前驱和直接后继。
- 中间代码级别：安排专门的优化阶段。
- 目标代码级别：考虑如何有效的利用寄存器，如果选择指令，以及进行窥孔优化等。

```

void QuickSort(int m, int n)
{
    int i, j, v, x;
    if (n <= m) return;
    i = m - 1; j = n; v = a[n];
    while (true) {
        do i = i + 1; while (a[i] < v);
        do j = j - 1; while (a[j] > v);
        if (i >= j) break;
        x = a[i]; a[i] = a[j]; a[j] = x;
    }
    x = a[i]; a[i] = a[n]; a[n] = x;
    QuickSort(m, j); QuickSort(i + 1, n);
}

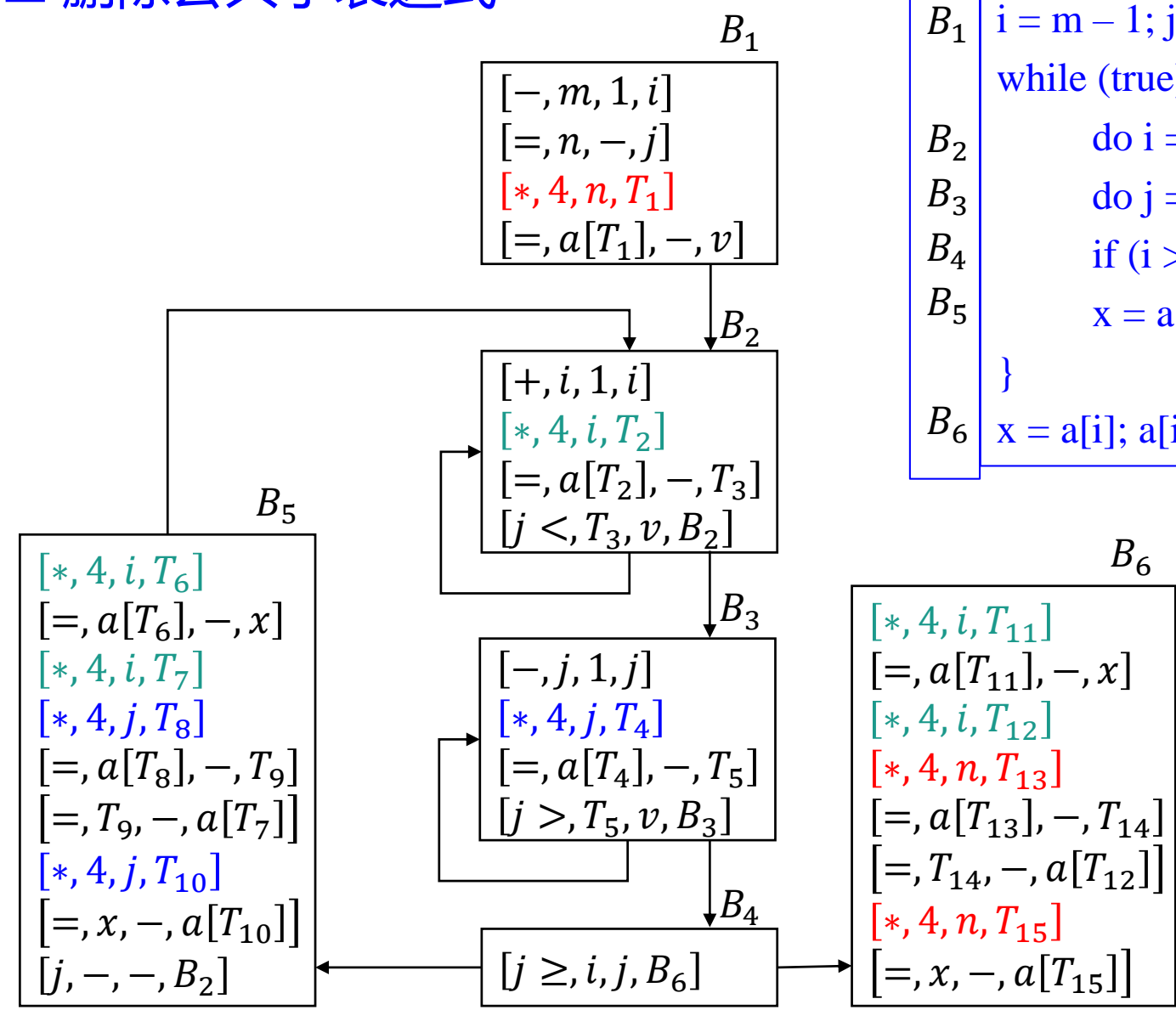
```



$B_1$	$i = m - 1; j = n; v = a[n];$
	<b>while (true) {</b>
$B_2$	do $i = i + 1$ ; while ( $a[i] < v$ );
$B_3$	do $j = j - 1$ ; while ( $a[j] > v$ );
$B_4$	if ( $i \geq j$ ) break;
$B_5$	$x = a[i]; a[i] = a[j]; a[j] = x;$
	<b>}</b>
$B_6$	$x = a[i]; a[i] = a[n]; a[n] = x;$

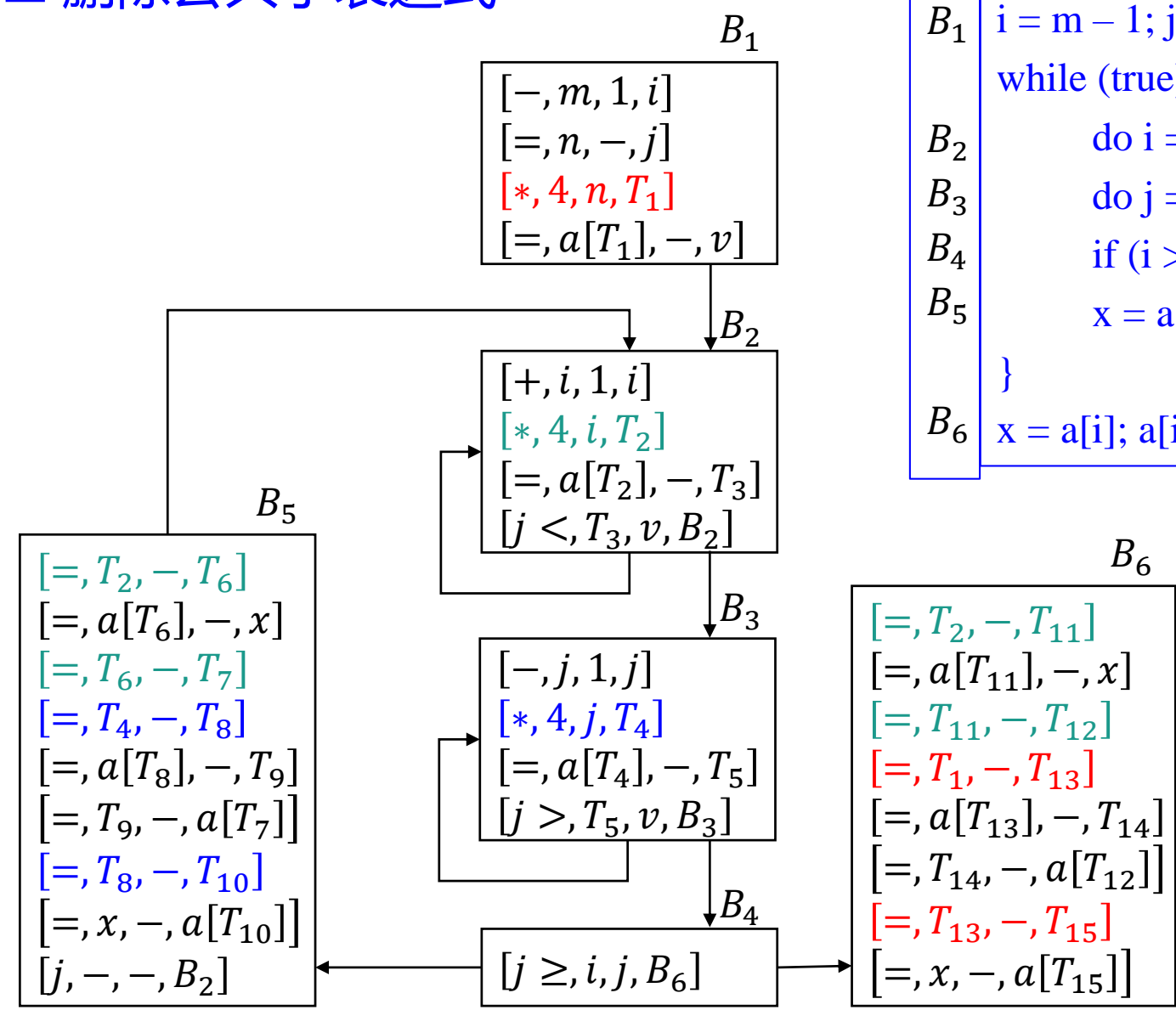


# 删除公共子表达式



B <sub>1</sub>	<code>i = m - 1; j = n; v = a[n];</code>
B <sub>2</sub>	<code>while (true) {</code>
B <sub>3</sub>	<code>do i = i + 1; while (a[i] &lt; v);</code>
B <sub>4</sub>	<code>do j = j - 1; while (a[j] &gt; v);</code>
B <sub>5</sub>	<code>if (i &gt;= j) break;</code>
B <sub>6</sub>	<code>x = a[i]; a[i] = a[j]; a[j] = x;</code>
	<code>}</code>
	<code>x = a[i]; a[i] = a[n]; a[n] = x;</code>

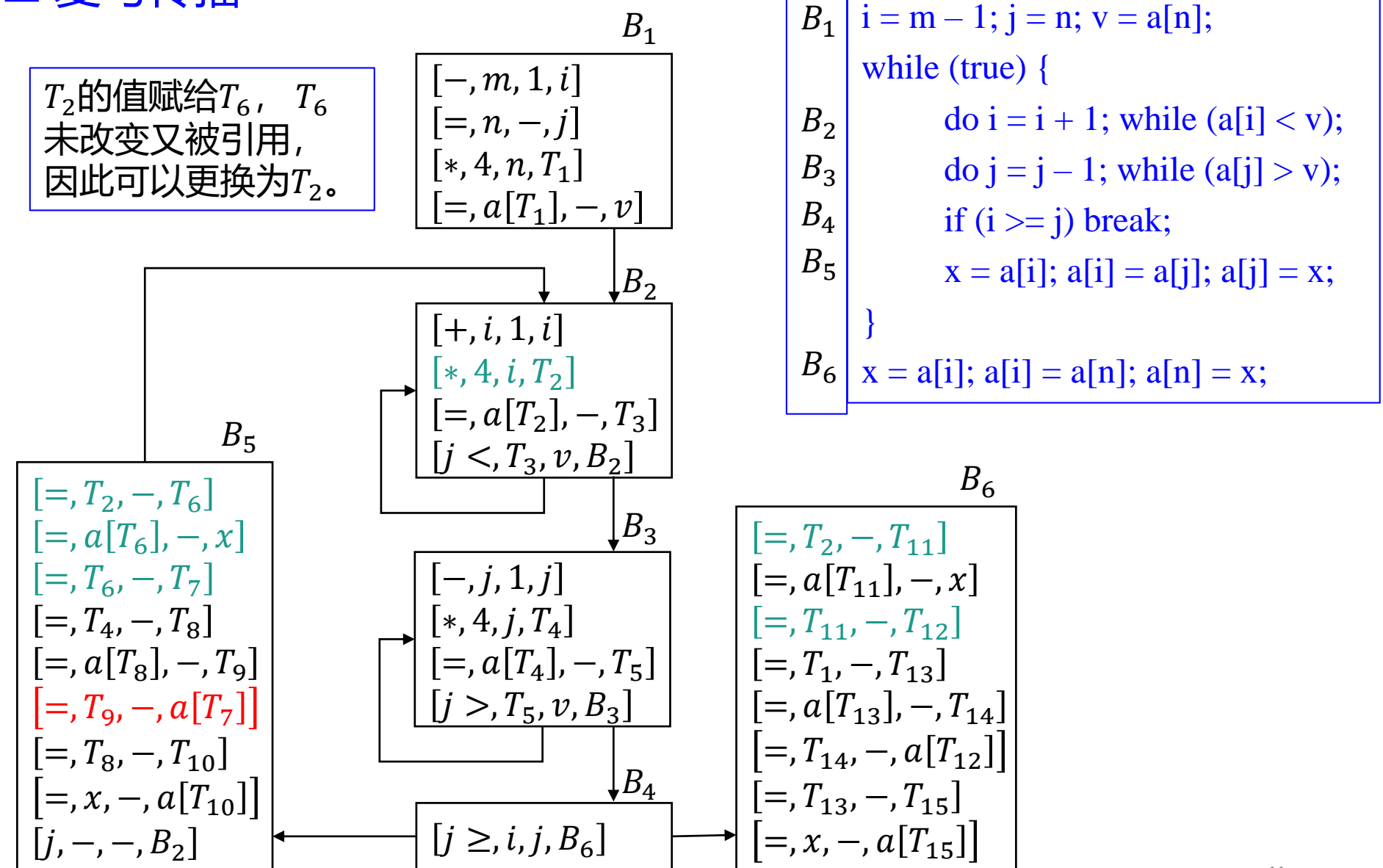
# 删除公共子表达式



B <sub>1</sub>	i = m - 1; j = n; v = a[n];
B <sub>2</sub>	while (true) {
B <sub>3</sub>	do i = i + 1; while (a[i] < v);
B <sub>4</sub>	do j = j - 1; while (a[j] > v);
B <sub>5</sub>	if (i >= j) break;
B <sub>6</sub>	x = a[i]; a[i] = a[j]; a[j] = x;
	}
	x = a[i]; a[i] = a[n]; a[n] = x;

# 复写传播

$T_2$  的值赋给  $T_6$ ,  $T_6$  未改变又被引用, 因此可以更换为  $T_2$ 。



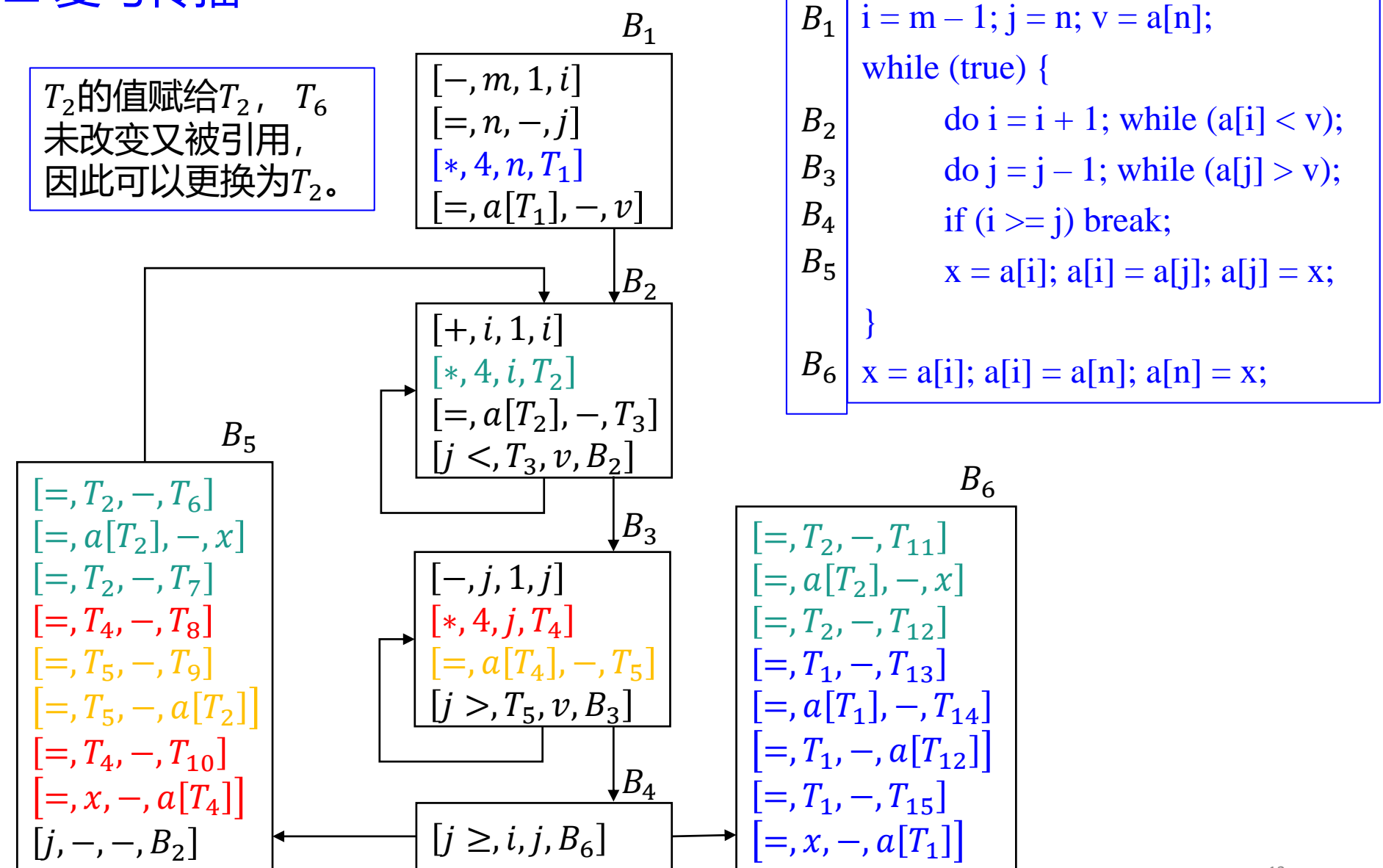
```

B1 i = m - 1; j = n; v = a[n];
while (true) {
B2     do i = i + 1; while (a[i] < v);
B3     do j = j - 1; while (a[j] > v);
B4     if (i >= j) break;
B5     x = a[i]; a[i] = a[j]; a[j] = x;
B6 }
x = a[i]; a[i] = a[n]; a[n] = x;

```

# 复写传播

$T_2$  的值赋给  $T_2$ ,  $T_6$  未改变又被引用, 因此可以更换为  $T_2$ 。



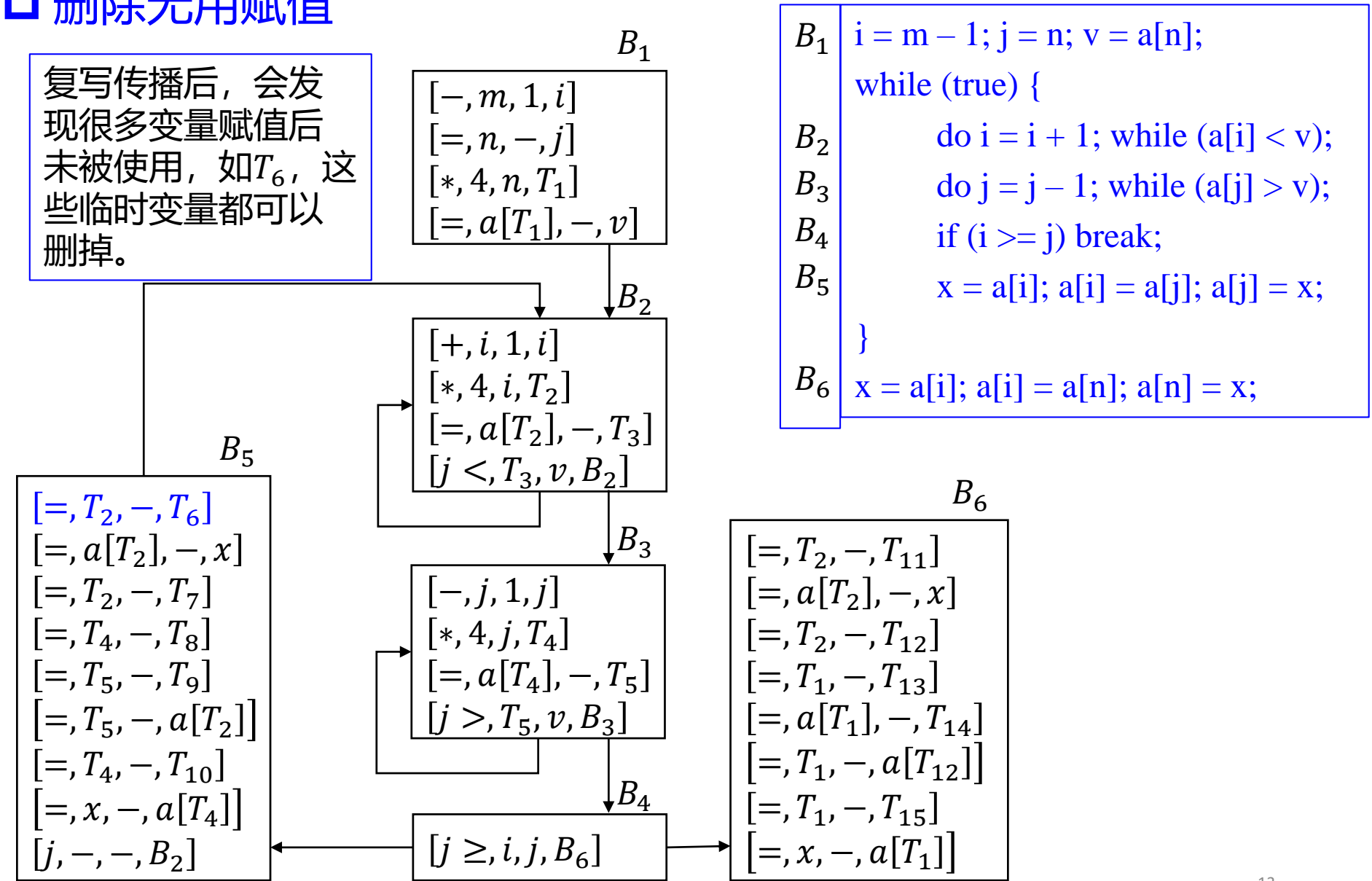
```

B1 i = m - 1; j = n; v = a[n];
while (true) {
B2     do i = i + 1; while (a[i] < v);
B3     do j = j - 1; while (a[j] > v);
B4     if (i >= j) break;
B5     x = a[i]; a[i] = a[j]; a[j] = x;
    }
B6 x = a[i]; a[i] = a[n]; a[n] = x;

```

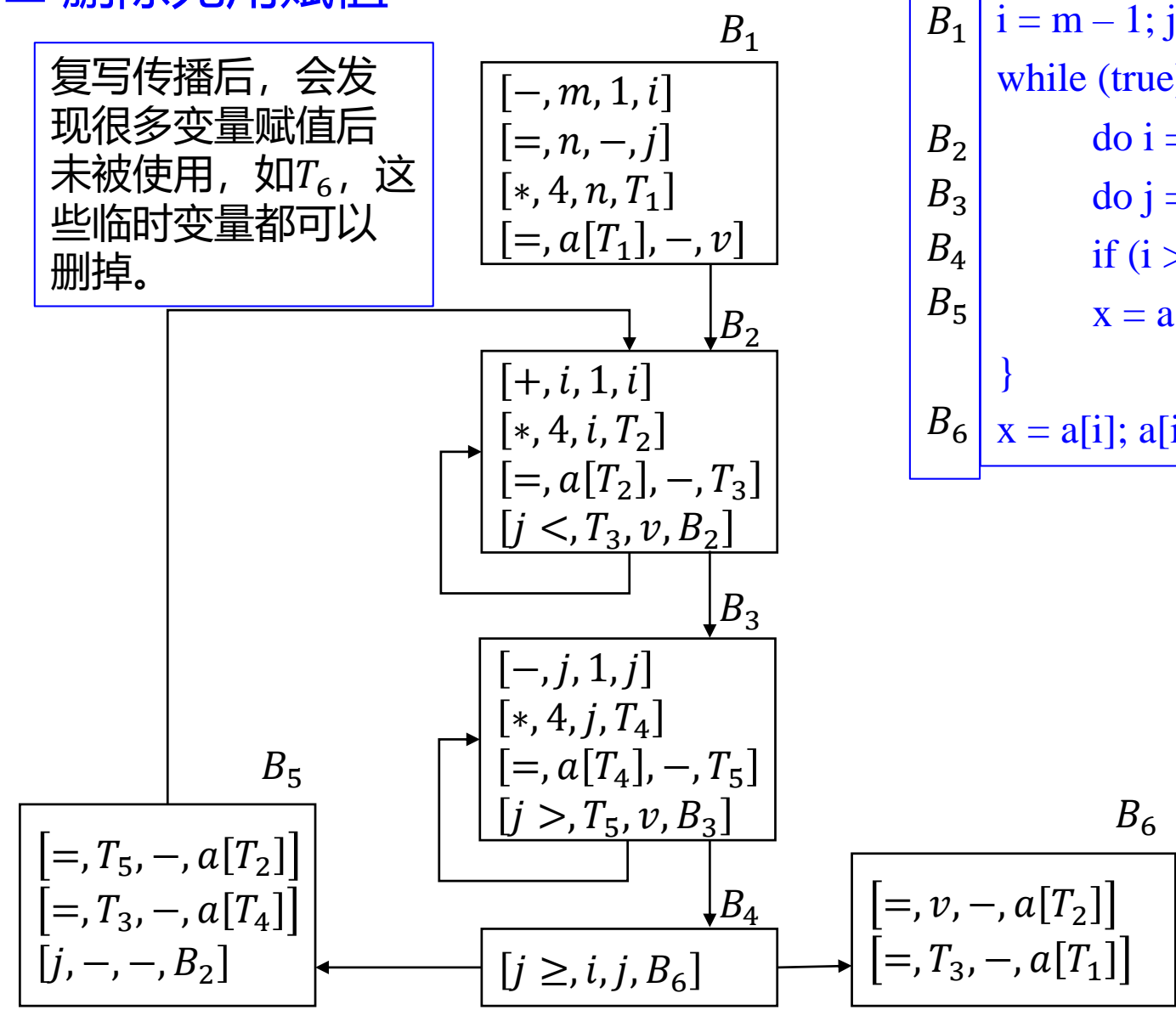
# 删除无用赋值

复写传播后，会发现很多变量赋值后未被使用，如 $T_6$ ，这些临时变量都可以删掉。



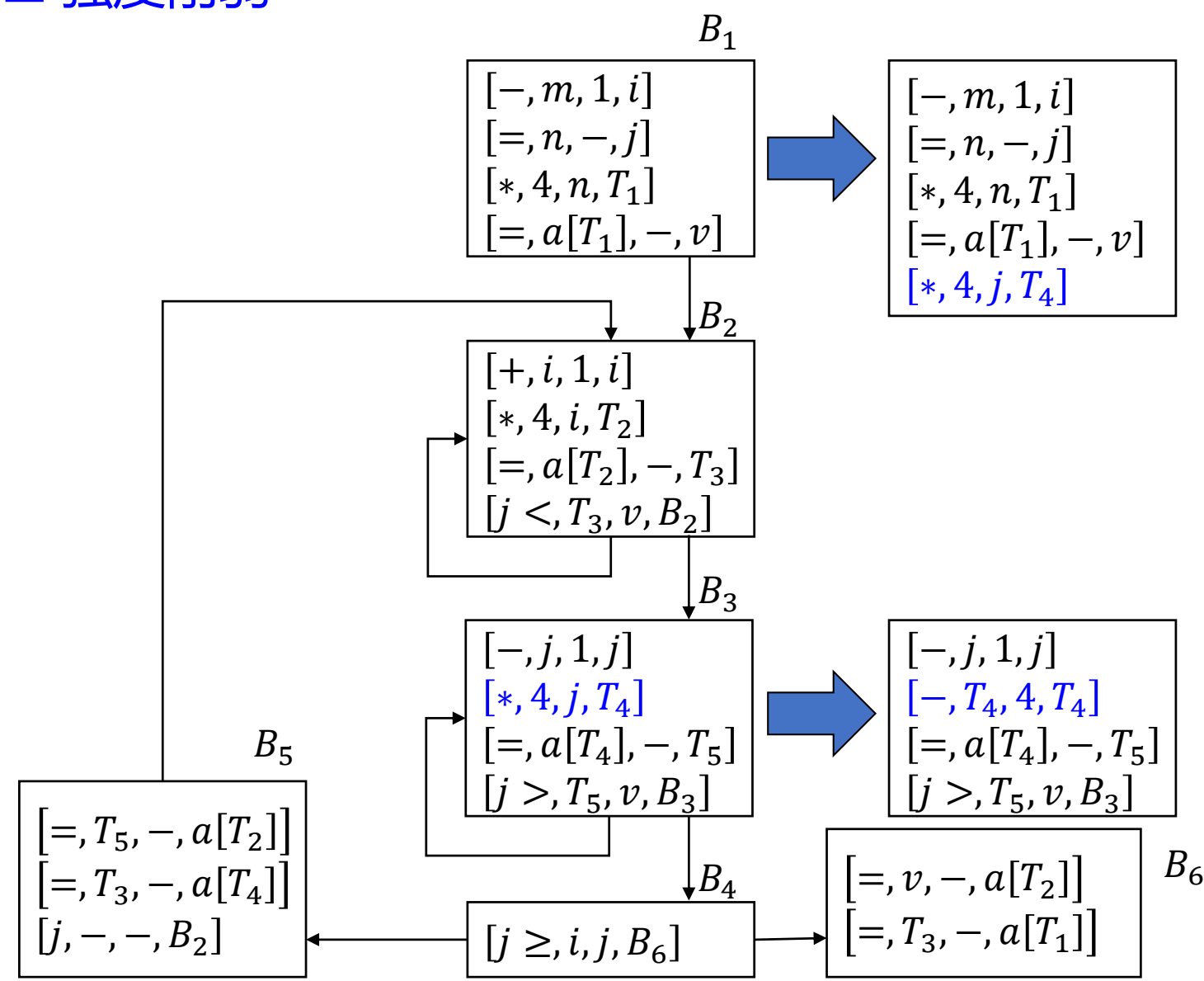
# 删除无用赋值

复写传播后，会发现很多变量赋值后未被使用，如 $T_6$ ，这些临时变量都可以删掉。



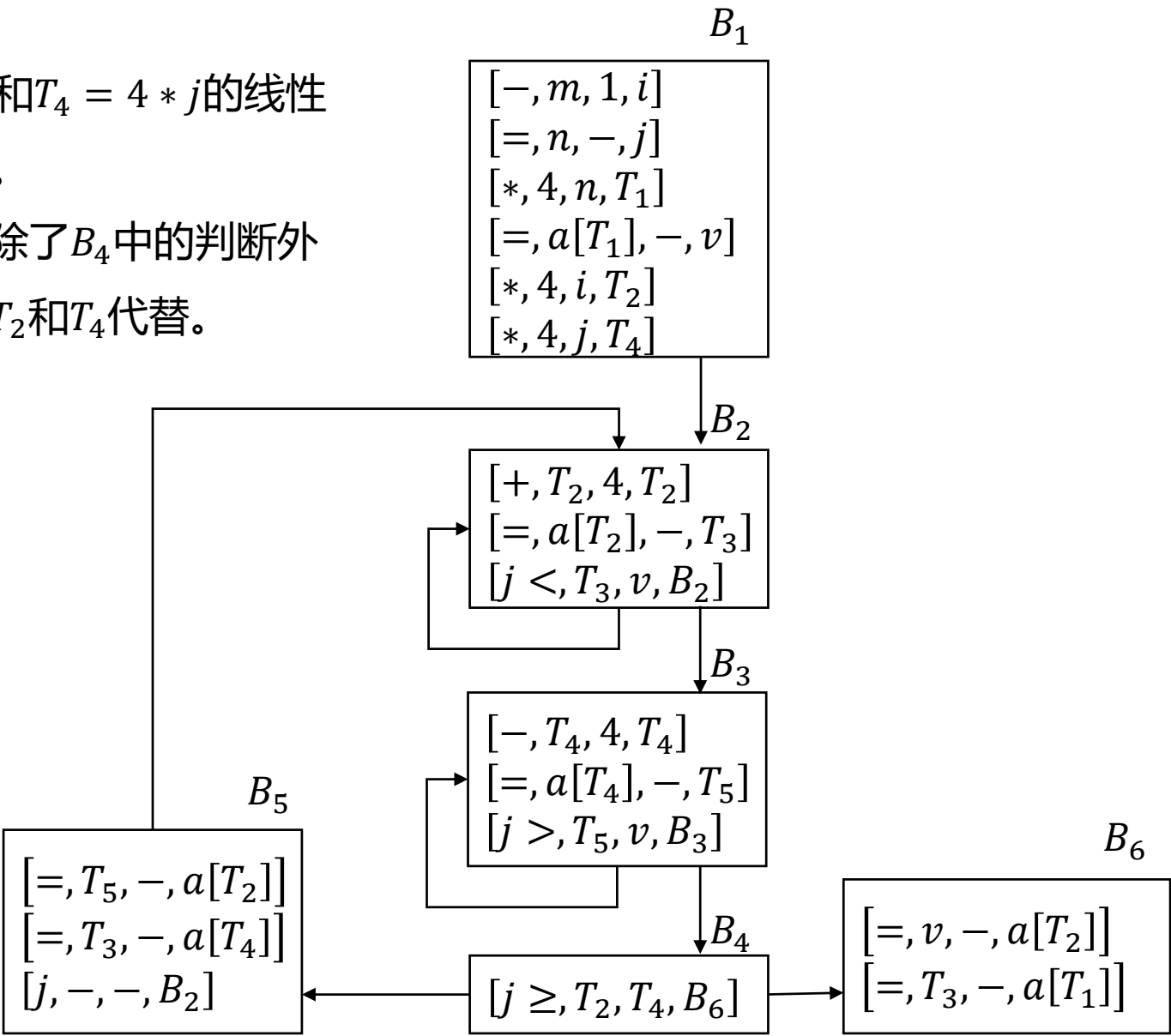
```
B1 i = m - 1; j = n; v = a[n];
while (true) {
  B2   do i = i + 1; while (a[i] < v);
  B3   do j = j - 1; while (a[j] > v);
  B4   if (i >= j) break;
  B5   x = a[i]; a[i] = a[j]; a[j] = x;
}
B6 x = a[i]; a[i] = a[n]; a[n] = x;
```

强度削弱



# 删除归纳变量

- 线性关系 $T_2 = 4 * i$ 和 $T_4 = 4 * j$ 的线性关系称为归纳变量。
- 强度削弱后， $i$ 和 $j$ 除了 $B_4$ 中的判断外不再使用，可以用 $T_2$ 和 $T_4$ 代替。





# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

### ➤ 10.2.1 基本块及流图

### ➤ 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

### ➤ 10.3.1 代码外提

### ➤ 10.3.2 强度削弱

### ➤ 10.3.3 删除归纳变量

## □ 10.4 数据流分析

### ➤ 10.4.1 任意路径数据流分析

### ➤ 10.4.2 全路径数据流分析

### ➤ 10.4.3 数据流问题的分类

### ➤ 10.4.4 其它主要数据流问题

### ➤ 10.4.5 利用数据流信息进行全局优化

## 10.2.1 基本块及流图

□ **基本块**：指程序中一段**顺序执行**的语句序列，其中只有一个**入口**和一个**出口**，入口就是其中的第一条语句，出口是其中最后一条语句。

【例10.2】基本块举例

$$T_1 = a * a$$

$$T_2 = a * b$$

$$T_3 = 2 * T_2$$

$$T_4 = T_1 + T_2$$

$$T_5 = b * b$$

$$T_6 = T_4 + T_5$$

## 10.2.1 基本块及流图

- 如果一条三地址语句为 $x = y + z$ ，则称对 $x$ 定值并引用 $y$ 和 $z$ 。
- 基本块中的一个名字在程序中某个给定点是活跃的，指如果在程序中（包括本基本块或其它基本块），它的值在该点以后被引用。
- 局限于基本块范围内的优化称为基本块内的优化，或称为局部优化。

# 基本块的划分算法

- 求出基本块入口语句，包括以下三种情况之一：
  - 程序的第一个语句；
  - 能由条件转移语句或无条件转移语句转移到的语句；
  - 紧跟在条件转移语句后面的语句。
- 基本块入口语句到以下语句之间部分构成一个基本块：
  - 后续的另一入口语句（不含该入口语句）；
  - 一条转移语句（含该转移语句）；
  - 停止语句（含该停止语句）。
- 未被纳入任一基本块的语句，都是程序中控制流不能到达的语句，可以删除。

# 基本块的划分算法

【例10.3】找出如下三地址语句基本块

(1) <i>read X</i>
(2) <i>read Y</i>
(3) <i>R = X mod Y</i>
(4) <i>if R = 0 goto 8</i>
(5) <i>X = Y</i>
(6) <i>Y = R</i>
(7) <i>goto 3</i>
(8) <i>write Y</i>
(9) <i>halt</i>

(1) <i>read X</i>
(2) <i>read Y</i>

(3) <i>R = X mod Y</i>
(4) <i>if R = 0 goto 8</i>

(5) <i>X = Y</i>
(6) <i>Y = R</i>
(7) <i>goto 3</i>

(8) <i>write Y</i>
(9) <i>halt</i>

# 基本块的优化

□ 删除公共子表达式

□ 删除无用赋值

□ 合并已知量

$$T_1 = 2$$

...

$$T_2 = 4 * T_1$$

$$T_2 = 8$$

□ 临时变量改名

$$T_1 = b + c$$

$$S = b + c$$

# 基本块的优化

## □ 交换语句位置

$$T_1 = b + c$$

$$T_2 = x + y$$

$$T_2 = x + y$$

$$T_1 = b + c$$

## □ 代数变换

$$x = x + 0$$

// 可删除

$$x = x * 1$$

// 可删除

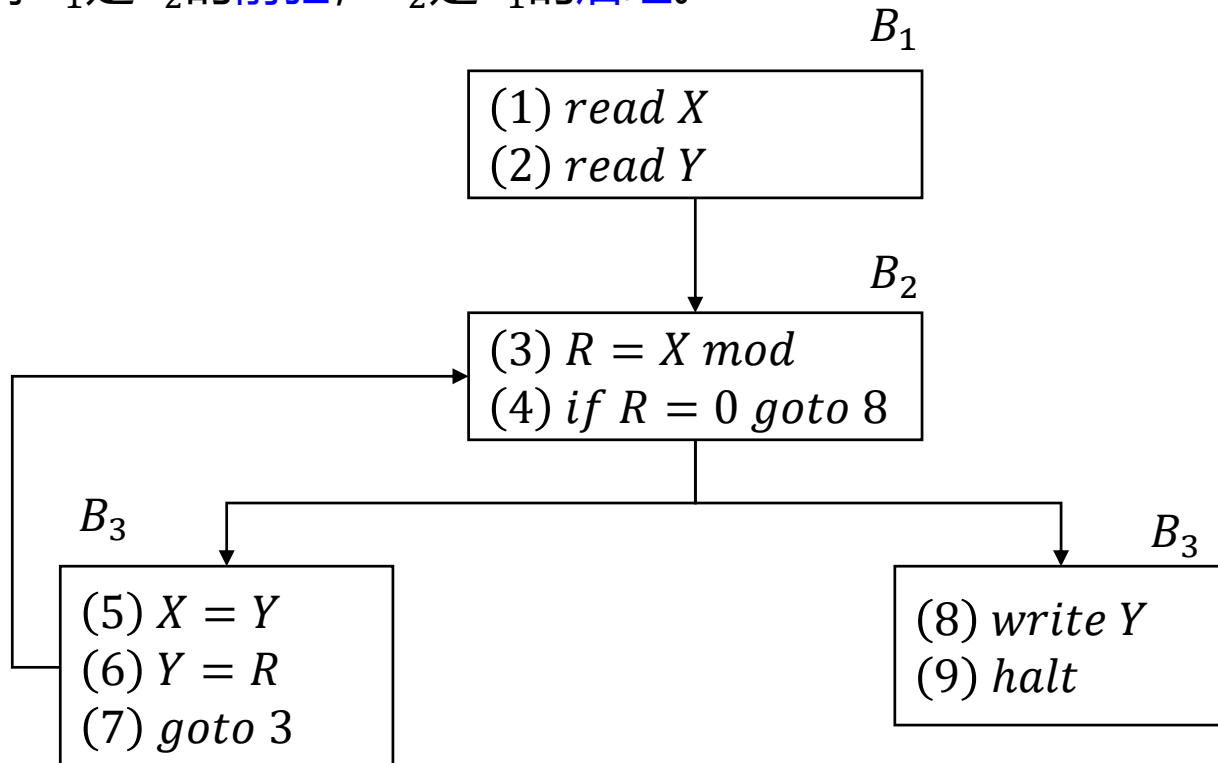
$$x = y ^ 2$$

$$x = y * y$$

# 基本块的优化

□ **流图**：以基本块为结点，通过有向图表示。

- 如果一个结点的基本块的入口语句是程序的第一条语句，则此结点为**首结点**。
- 如果在某个执行顺序中，基本块 $B_2$ 紧接在 $B_1$ 之后执行，则从 $B_1$ 到 $B_2$ 有一条有向边，称 $B_1$ 是 $B_2$ 的**前驱**， $B_2$ 是 $B_1$ 的**后继**。





# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

### ➤ 10.2.1 基本块及流图

### ➤ 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

### ➤ 10.3.1 代码外提

### ➤ 10.3.2 强度削弱

### ➤ 10.3.3 删除归纳变量

## □ 10.4 数据流分析

### ➤ 10.4.1 任意路径数据流分析

### ➤ 10.4.2 全路径数据流分析

### ➤ 10.4.3 数据流问题的分类

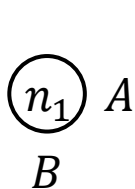
### ➤ 10.4.4 其它主要数据流问题

### ➤ 10.4.5 利用数据流信息进行全局优化

# 基本块的DAG

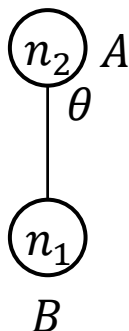
□ 一个基本块的DAG是一种结点带有下列**标记**或**附加信息**的DAG

- **叶结点**: **标记**为变量名或常数, 表示该结点变量或常数的值;
- **内部结点**: **标记**为运算符, 表示后继结点的运算结果;
- 各节点可能**附加**一个或多个变量名, 表示这些变量具有该结点的值。
- 结点圈内为**编号**, 下面为**标记**, 右面为**附加信息**。



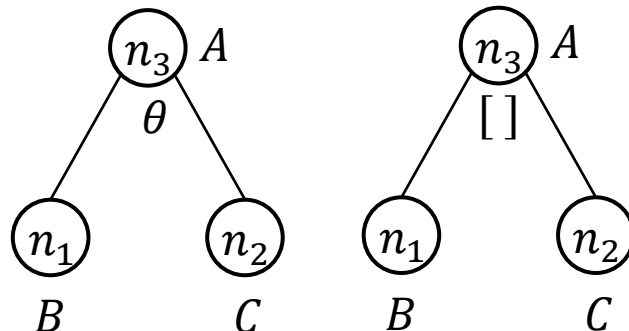
0型四元式

$$A = B$$



1型四元式

$$A = \theta B$$

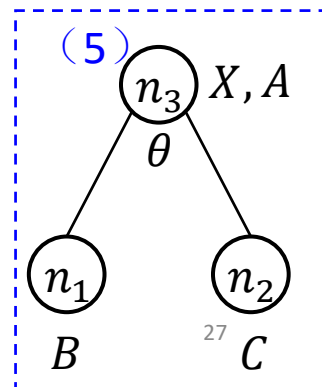
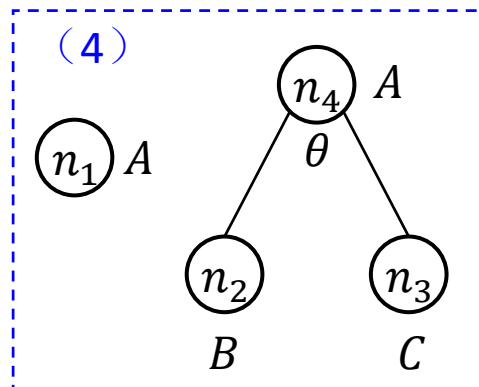
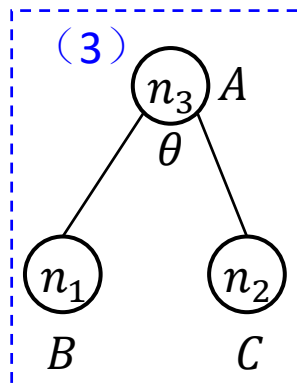
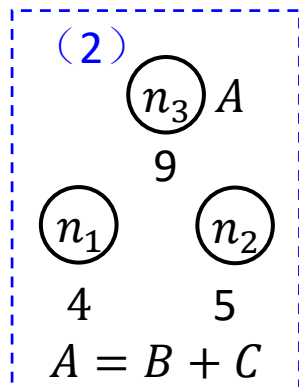
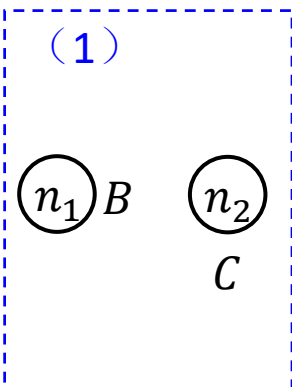


2型四元式

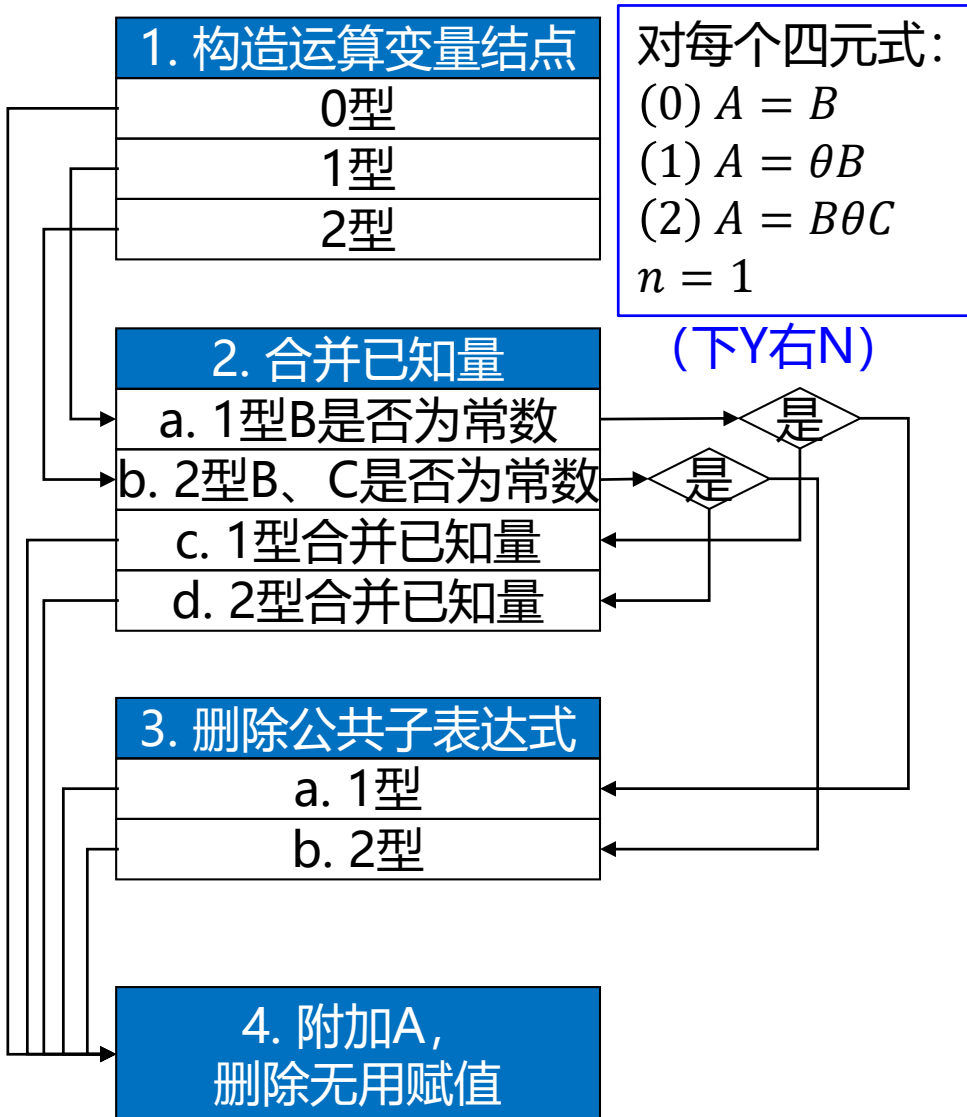
$$A = B \theta C \text{ 或 } A = B [C]$$

## □ DAG优化的基本思想

- (1) 对每个四元式, 如 $A = B \theta C$ , 找出或建立代表 $B$ 和 $C$ 当前值的结点;
- (2) 若 $B$ 和 $C$ 都是叶结点, 且都为常数, 则直接执行 $B \theta C$ , 然后建立以运算结果 $P$ 为标记的叶结点, 并把 $A$ 附加上去, 即合并已知量; ( $B$ 或 $C$ 若新建立, 删除之);
- (3) 若 $B$ 或 $C$ 是内部结点, 或至少一个不是常数, 则建立以 $\theta$ 为标记的新结点, 此结点分别以 $B$ 和 $C$ 为左右直接后继结点, 并把 $A$ 附加上去;
- (4) 若第三步之前, DAG中已有结点有附加标记 $A$ , 且此结点无前驱, 则在建立新结点的同时, 把老结点上附加的 $A$ 删除, 即删除无用赋值;
- (5) 若原来已有代表 $B \theta C$ 的结点, 则不必建立新的结点, 只需把 $A$ 附加到代表 $B \theta C$ 的结点上, 即删除公共子表达式。



# 基本块的DAG优化算法



## 1、构造运算变量的结点

如果 $Node(B)$ 没有定义，构造标记为 $B$ 的叶结点。

a. 0型：记 $Node(B) = n$ ，转4

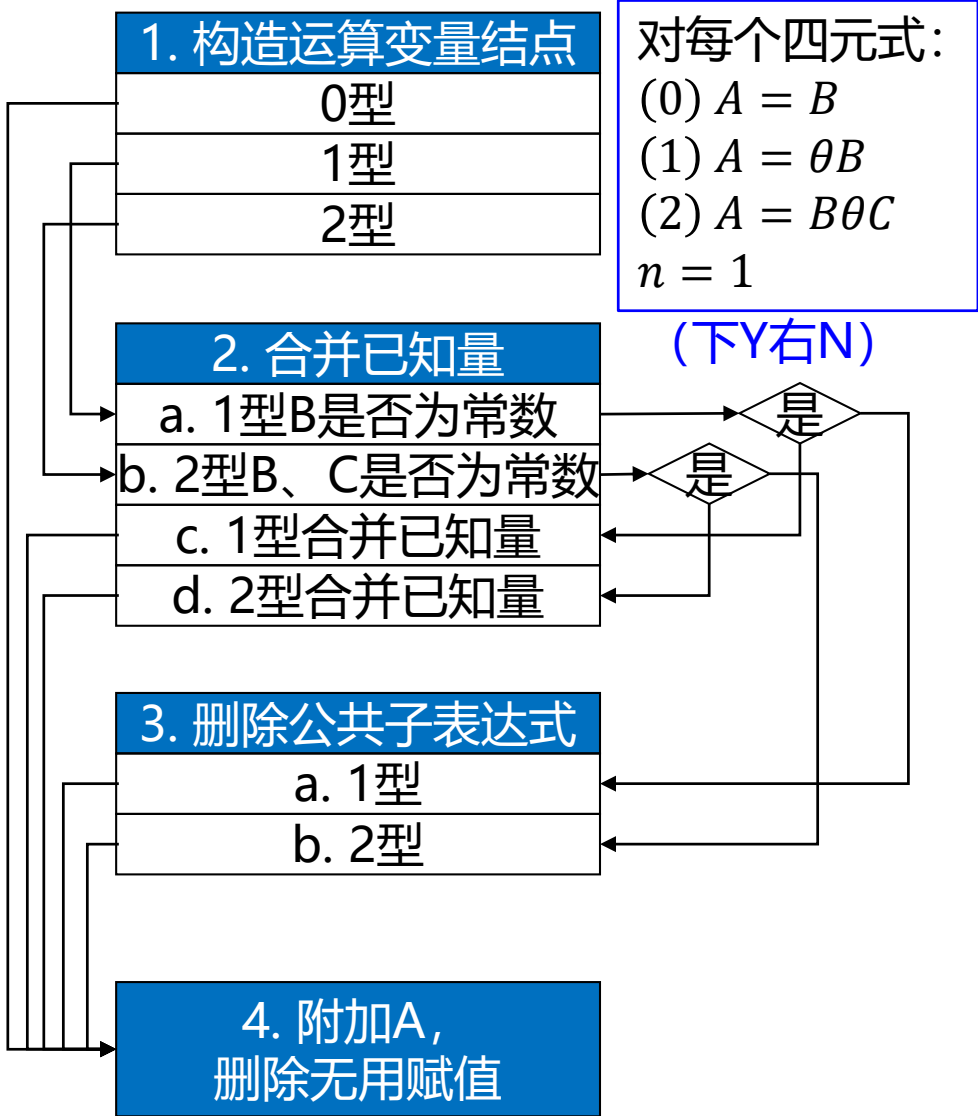
b. 1型：转2.a。

c. 2型：

- 如果 $Node(C)$ 没有定义，构造标记为 $C$ 的叶结点；

- 转2.b。

# 基本块的DAG优化算法



## 2、合并已知量

- a. 如果 $Node(B)$ 是标记为常数的叶结点, 则转2.c, 否则转3.a。
- b. 如果 $Node(B)$ 和 $Node(C)$ 都是标记为常数的叶结点, 则转2.d, 否则转3.b。
- c. 令 $p = \theta B$ , 如果 $Node(B)$ 是新构造出来的, 删除之; 如果 $Node(p)$ 无定义, 构造标记为 $p$ 的叶结点,  $Node(p) = n$ , 转4
- d. 令 $p = B\theta C$ , 如果 $Node(B)$ 或 $Node(C)$ 是新构造出来的, 删除之; 如果 $Node(p)$ 无定义, 构造标记为 $p$ 的叶结点,  $Node(p) = n$ , 转4。

# 基本块的DAG优化算法

## 1. 构造运算变量结点

0型

1型

2型

对每个四元式:

(0)  $A = B$

(1)  $A = \theta B$

(2)  $A = B \theta C$

$n = 1$

(下Y右N)

## 2. 合并已知量

a. 1型B是否为常数

b. 2型B、C是否为常数

c. 1型合并已知量

d. 2型合并已知量

## 3. 删除公共子表达式

a. 1型

b. 2型

4. 附加A,  
删除无用赋值

## 3. 删除公共子表达式

- a. 1型: 检查是否有结点, 其唯一后继为 $Node(B)$ 且标记为 $\theta$ .
  - 若没有, 创建之, 置为 $n$ , 转4
  - 若有, 作为它的结点, 转4
- b. 2型: 检查是否有结点, 其左右后继为 $Node(B)$ 和 $Node(C)$ , 且标记为 $\theta$ .
  - 若没有, 创建之, 置为 $n$ , 转4
  - 若有, 作为它的结点, 转4

# 基本块的DAG优化算法

## 1. 构造运算变量结点

0型

1型

2型

对每个四元式：

(0)  $A = B$

(1)  $A = \theta B$

(2)  $A = B \theta C$

$n = 1$

## 2. 合并已知量

a. 1型B是否为常数

b. 2型B、C是否为常数

c. 1型合并已知量

d. 2型合并已知量

(下Y右N)

是

是

## 3. 删除公共子表达式

a. 1型

b. 2型

## 4. 附加A, 删除无用赋值

## 4、附加A，删除无用赋值

- 若 $Node(A)$ 无定义把A附加到结点 $n$
- 若 $Node(A)$ 已定义，则把A从原 $Node(A)$ 删除，再把A附加到新结点 $n$ ，以下情况不删除：
  - $A$ 是叶结点标记

重新生成四元式前，把有前驱但没有附加信息的内部结点，生成新的临时变量附加上去。

# 基本块的DAG优化算法

【例10.4】构造以下基本块的DAG

(1)  $T_0 = 3.14$

(2)  $T_1 = 2 * T_0$

(3)  $T_2 = R * r$

(4)  $A = T_1 * T_2$

(5)  $B = A$

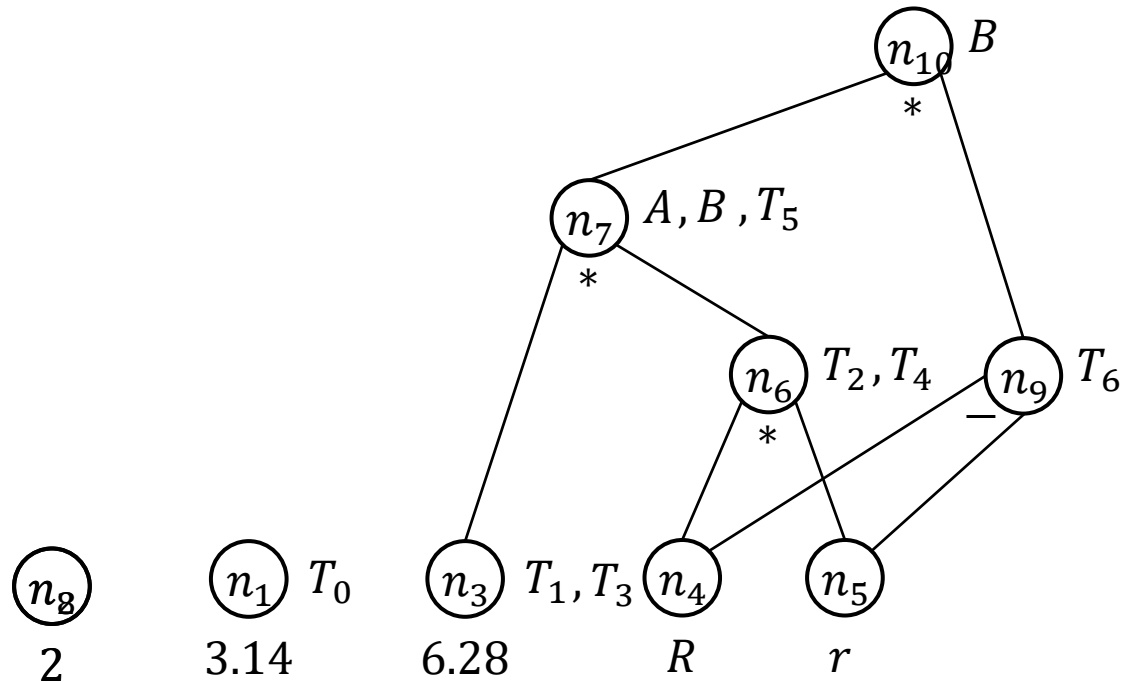
(6)  $T_3 = 2 * T_0$

(7)  $T_4 = R * r$

(8)  $T_5 = T_3 * T_4$

(9)  $T_6 = R - r$

(10)  $B = T_5 * T_6$

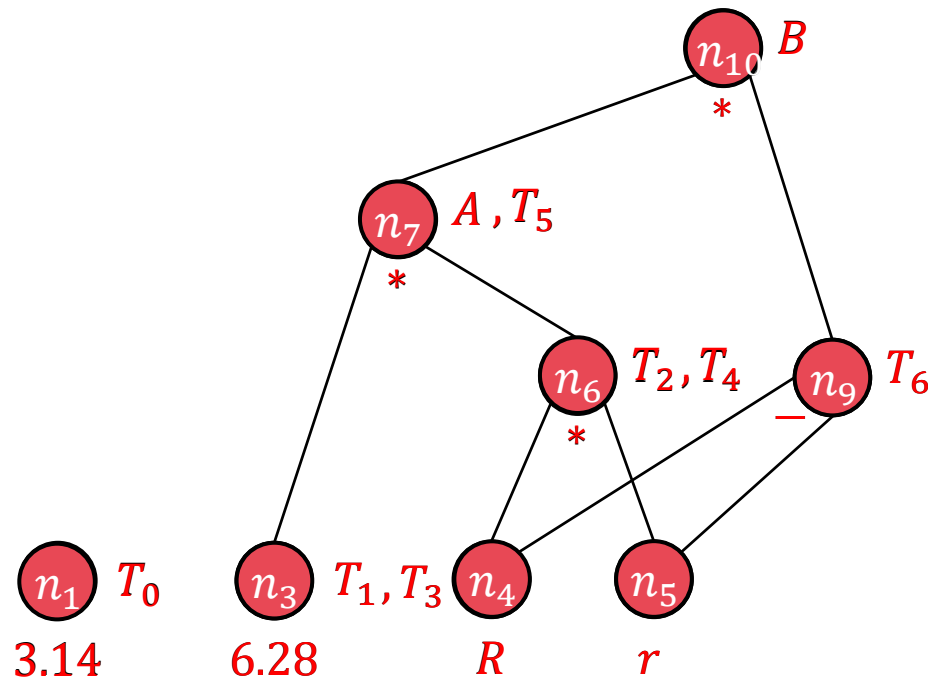




# 基本块的DAG优化算法

【例10.5】按构造DAG结点的顺序重构中间代码。

(1) $T_0 = 3.14$
(2) $T_1 = 2 * T_0$
(3) $T_2 = R * r$
(4) $A = T_1 * T_2$
(5) $B = A$
(6) $T_3 = 2 * T_0$
(7) $T_4 = R * r$
(8) $T_5 = T_3 * T_4$
(9) $T_6 = R - r$
(10) $B = T_5 * T_6$



(1) $T_0 = 3.14$
(2) $T_1 = 6.28$
(3) $T_3 = 6.28$
(4) $T_2 = R * r$
(5) $T_4 = T_2$
(6) $A = 6.28 * T_2$
(7) $T_5 = A$
(8) $T_6 = R - r$
(9) $B = A * T_6$

# 基本块的DAG优化算法

【例10.6】对如下基本块进行DAG优化。

(1)  $T_1 = a * b$

(2)  $T_2 = 2 * 3$

(3)  $T_3 = T_1 - T_2$

(4)  $x = T_3$

(5)  $c = 5$

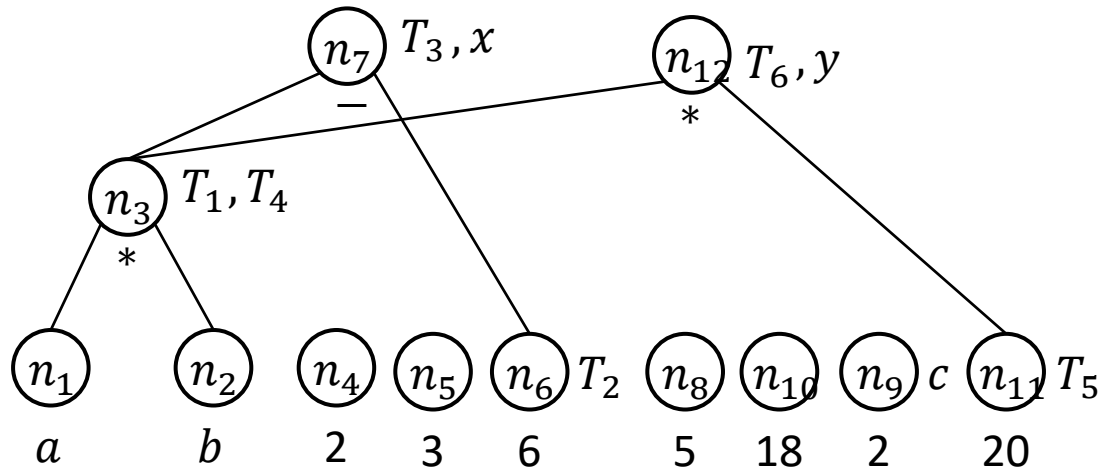
(6)  $T_4 = a * b$

(7)  $c = 2$

(8)  $T_5 = 18 + c$

(9)  $T_6 = T_4 * T_5$

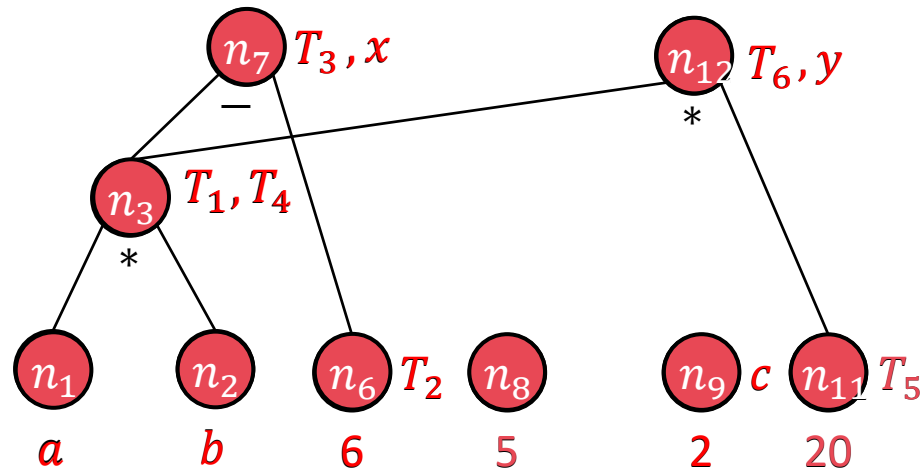
(10)  $y = T_6$



# 基本块的DAG优化算法

【例10.6】对如下基本块进行DAG优化。

(1) $T_1 = a * b$
(2) $T_2 = 2 * 3$
(3) $T_3 = T_1 - T_2$
(4) $x = T_3$
(5) $c = 5$
(6) $T_4 = a * b$
(7) $c = 2$
(8) $T_5 = 18 + c$
(9) $T_6 = T_4 * T_5$
(10) $y = T_6$



(1) $T_1 = a * b$
(2) $T_4 = T_1$
(3) $T_2 = 6$
(4) $T_3 = T_1 - 6$
(5) $x = T_3$
(6) $c = 2$
(7) $T_5 = 20$
(8) $T_6 = T_1 * 20$
(9) $y = T_6$

- ❑ 叶结点上标记的标识符：是基本块外定值而基本块内引用的变量。
- ❑ 各结点上附加的标识符：是基本块内定值而可能在基本块外引用的变量。

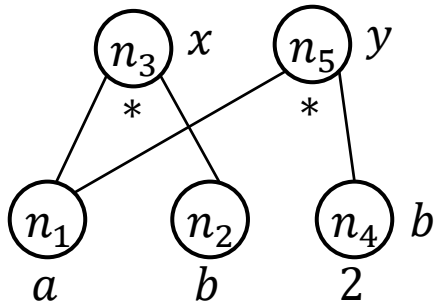
# 出现多个可用结点

【例10.7】构造如下两个基本块的DAG

(1)  $x = a * b$

(2)  $b = 2$

(3)  $y = a * b$

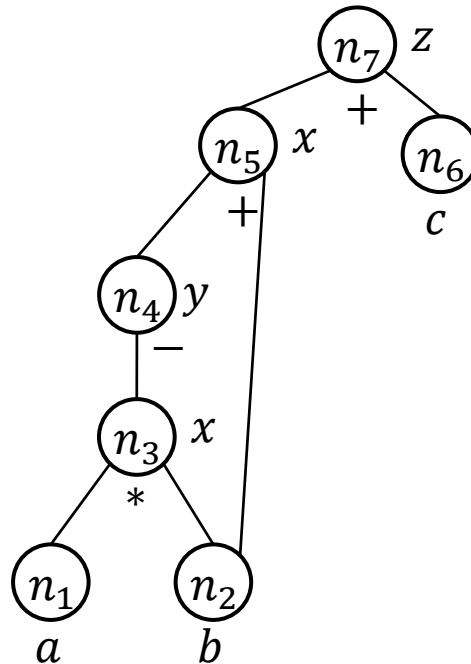


(1)  $x = a * b$

(2)  $y = -x$

(3)  $x = y + b$

(4)  $z = x + c$

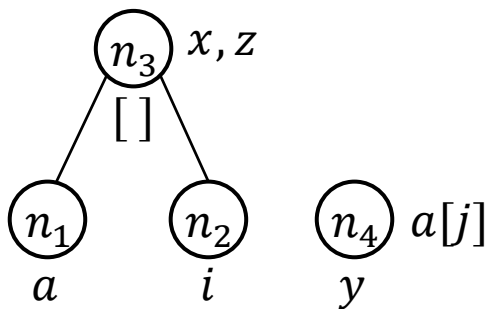


□ 搜索结点应从后往前搜索。

# 数组的处理

【例10.8】构造如下两个基本块的DAG

(1) $x = a[i]$	(1) $x = a[i]$
(2) $a[j] = y$	(2) $z = x$
(3) $z = a[i]$	(3) $a[j] = y$



□ 当 $i = j, y \neq a[i]$ 时, 两者 $z$ 值不一致

➤ 原因: 当对数组赋值时, 即使 $a$ 和 $i$ 都未改变, 也可能改变了 $a[i]$ 的右值

□ 数组做限制

➤ 当对数组 $a$ 的元素赋值时, 注销数组 $a$ 作为公共子表达式资格;

➤ 即标记为 $[ ]$ , 左变元为 $a$ 的结点, 禁止再添加附加变元;

➤ 每个结点需要有个标志位标记是否已注销。

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

## 10.3.1 代码外提

### 【例10.9】代码外提

```
for (i = 0; i < 100; i++)
```

```
{
```

```
    u = a + b;
```

```
    s = u + i;
```

```
}
```



```
u = a + b;
```

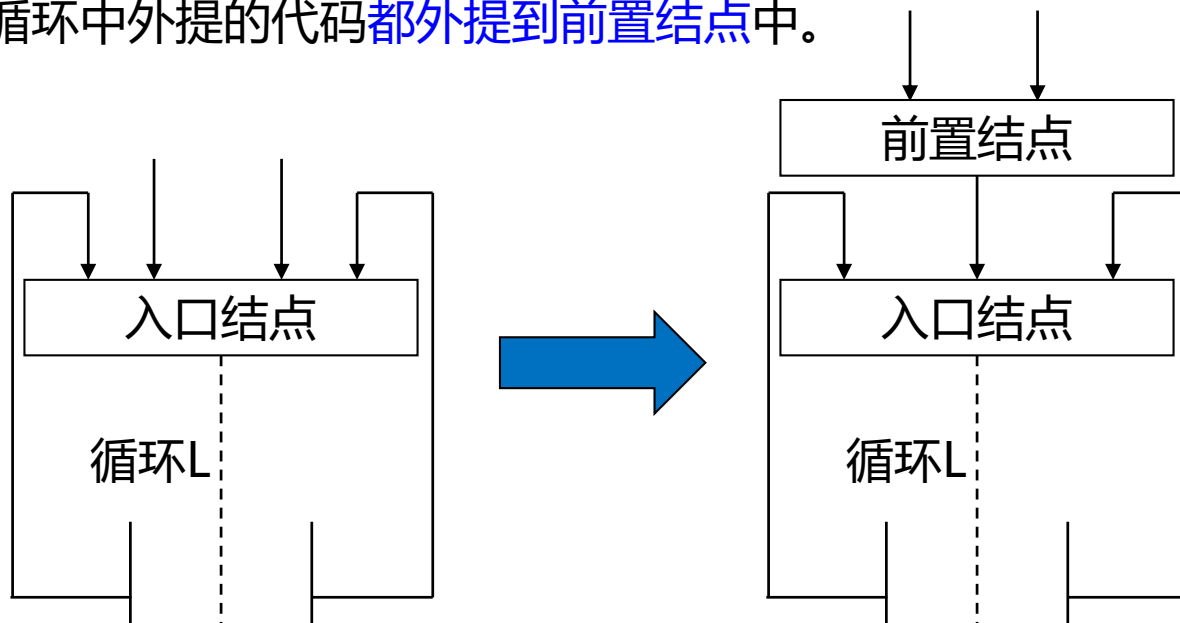
```
for (i = 0; i < 100; i++)
```

```
    s = u + i;
```

- 变量 $A$ 在某点**定值**：即在该点对 $A$ 赋值
- 变量 $A$ 在某点 $d$ 的**定值到达**另一点 $u$ ：指流图中从 $d$ 有一条通路到达 $u$ ，且该通路上没有 $A$ 的其它定值。
- 在循环中定值不变的运算称为**循环不变运算**。

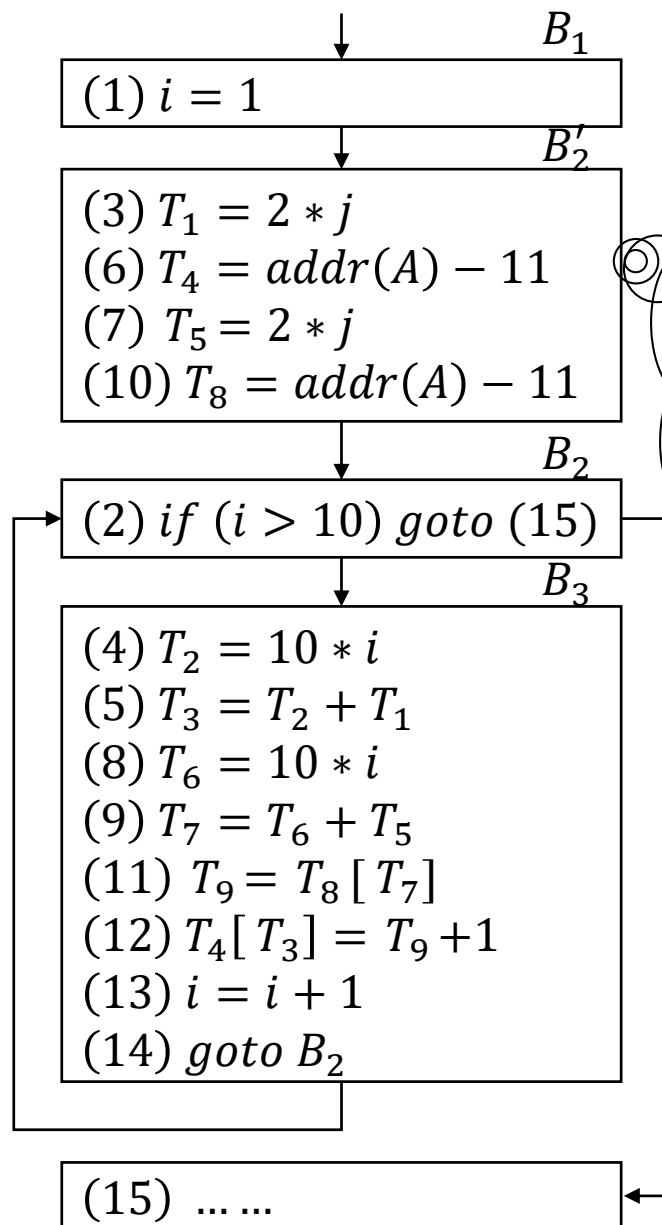
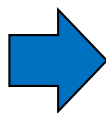
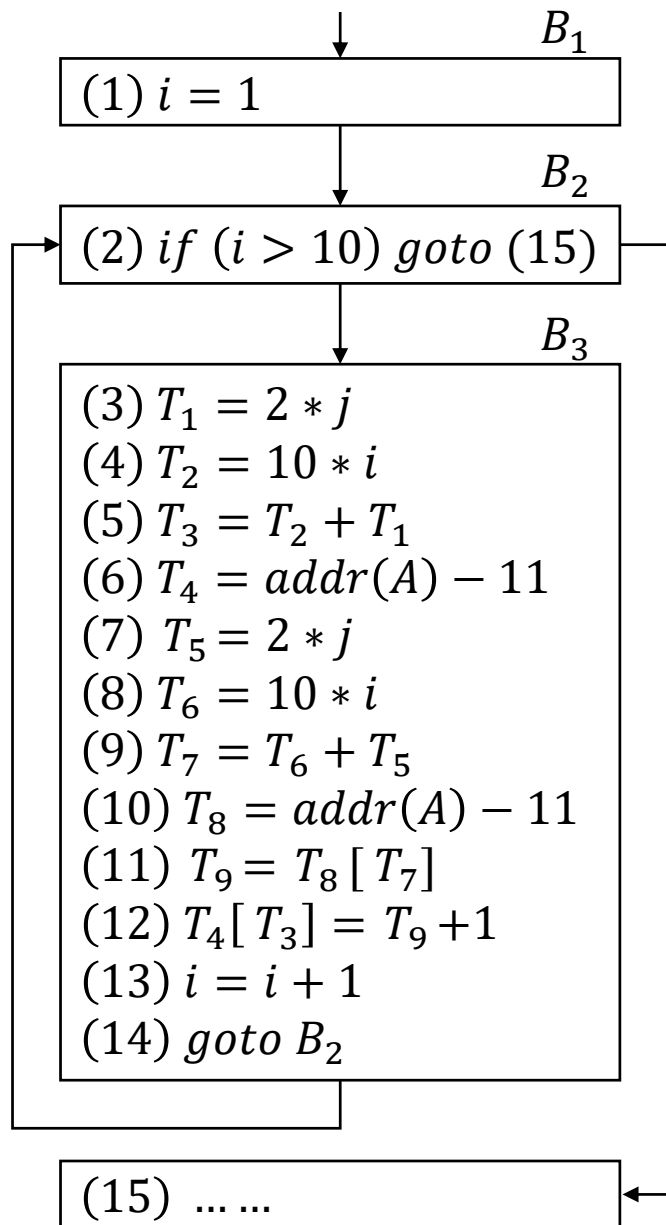
## 10.3.1 代码外提

- 代码外提时，在循环入口点前面建立一个新结点（基本块），称为循环的**前置结点**。
- 循环**前置结点**以**循环入口结点**为其**唯一后继**，原来流图中从**循环外引入**（不包括循环内引入）到循环入口结点的有向边，改成引入到循环前置结点。
- 循环入口结点是**唯一**的，所以前置结点也是**唯一**的。
- 循环中外提的代码都外提到前置结点中。



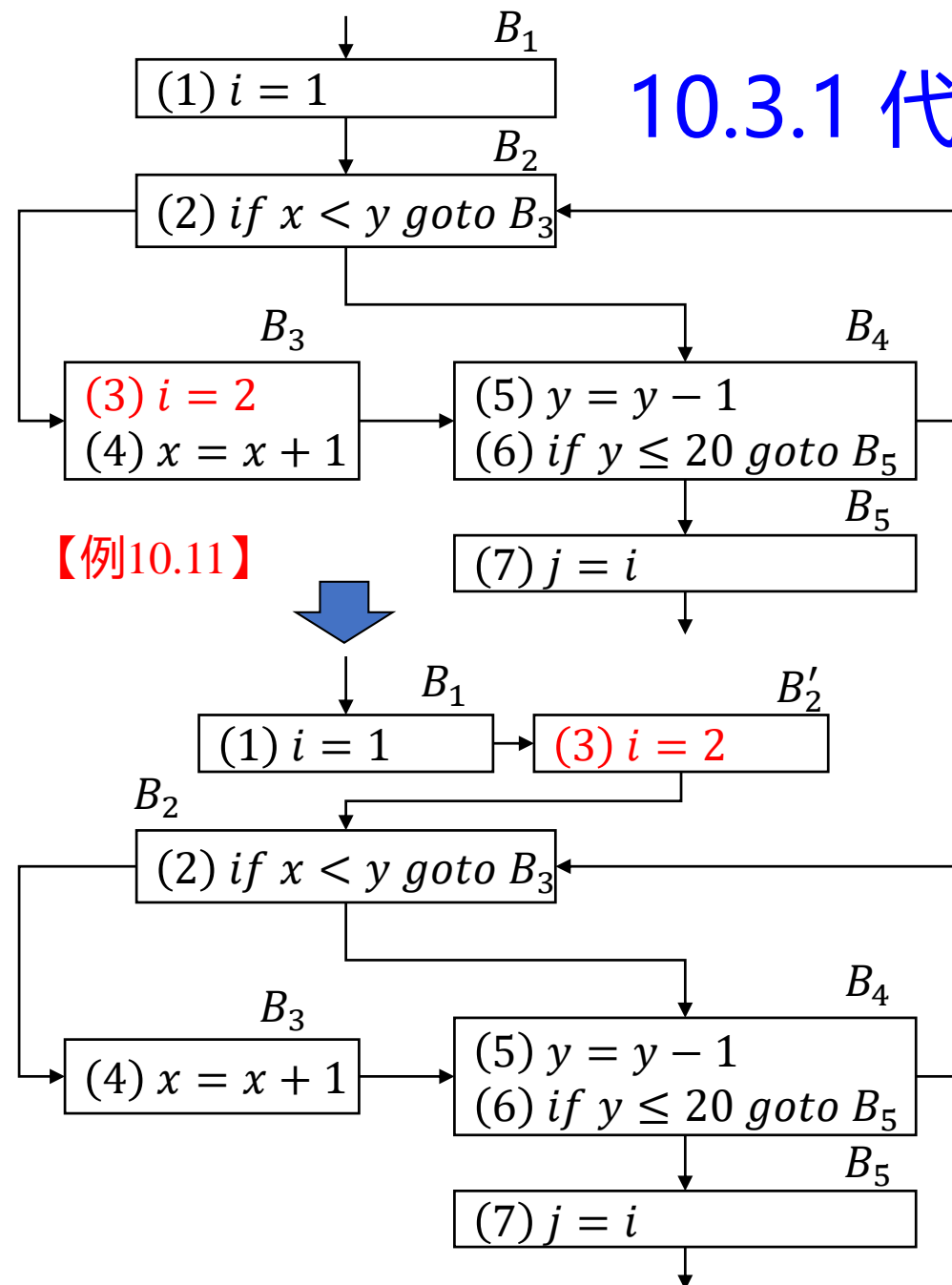


【例10.10】  $i = 1$ ; for (;  $i \leq 10$ ;  $i = i + 1$ )  $A[i, 2*j] = A[i, 2*j] + 1$ ; 假设  $A[10, 10]$



是否任何  
情况下都  
可以把循  
环不变运  
算代码外  
提?

## 10.3.1 代码外提



□ 循环不变量  $i = 2$  提到循环外

- 代码外提后，最后总有  $j = 2$
- 代码外提前，如果  $x \geq y$ ，则  $j = 1$

□ 循环外提条件

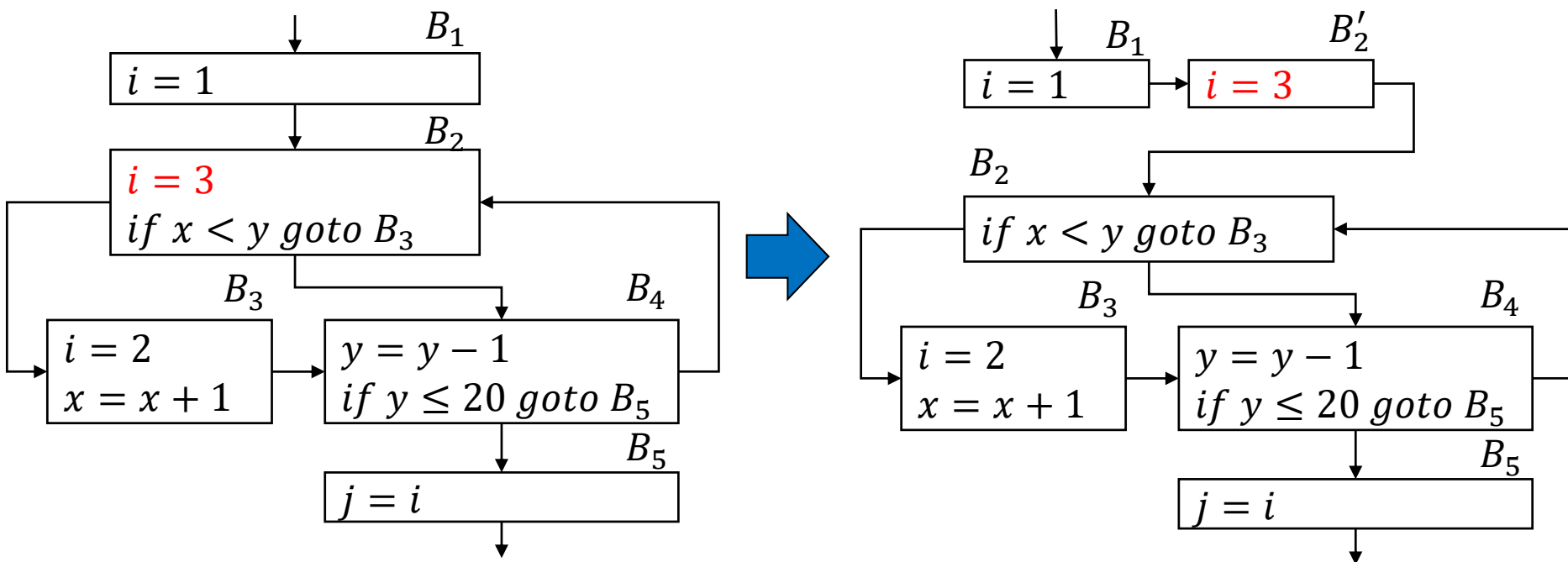
- $B_3$  必须是出口结点  $B_4$  的必经结点；
- 或者，虽然不是必经结点，但出循环后不再引用该定点值。

□ 活跃变量（变量A在点p活跃）

- 指A要在从  $p$  开始的某通路上被引用
- 前述等价于：在循环外的循环后继结点入口，变量  $i$  不活跃才能外提。

【例10.12】

## 10.3.1 代码外提



□  $i = 3$ 所在的 $B_2$ 是出口结点的**必经结点**，是否可外提？

□ **路径**：  $B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$

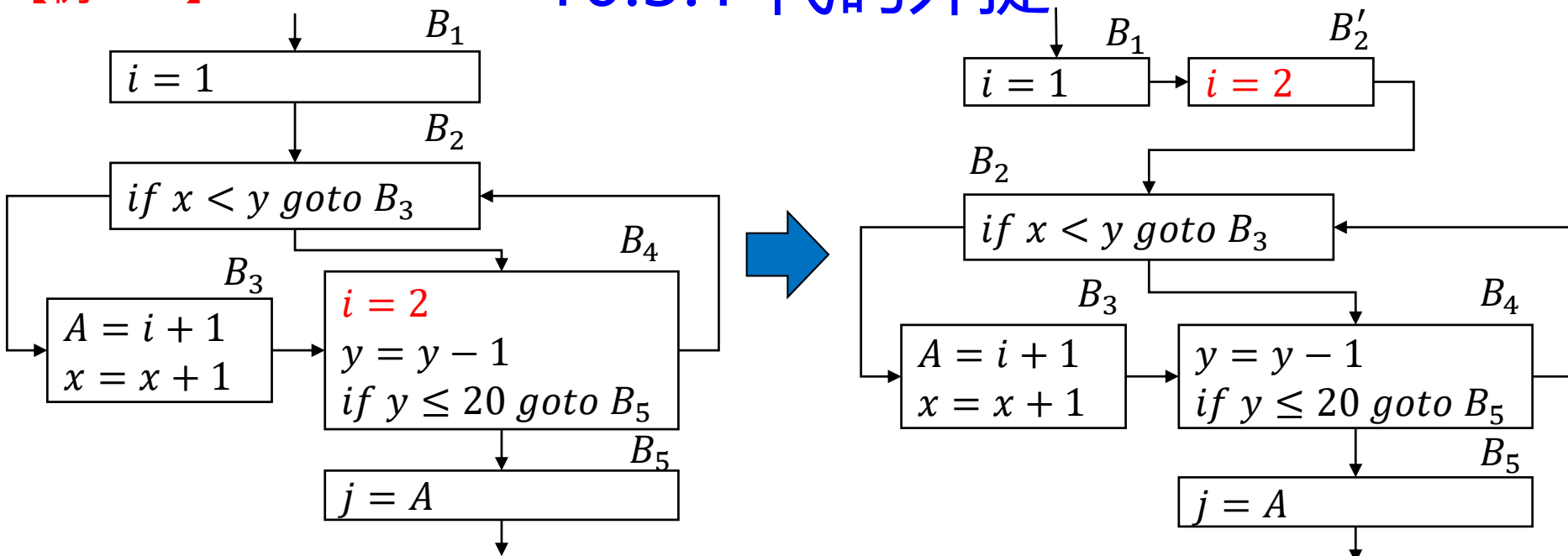
➤ 代码外提前，  $j = 3$

➤ 代码外提后，  $j = 2$

□ **结论**：循环不变量  $A = B \theta C$  外提时，要求循环中其它地方不再有  $A$  的**定值点**。<sup>43</sup>

【例10.13】

## 10.3.1 代码外提



□  $i = 2$ 所在的 $B_4$ 本身是出口结点，且其它地方没有 $i$ 的定值，是否可外提？

□ **路径：**  $B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_2 \rightarrow B_4 \rightarrow B_5$

➤ 代码外提前， $j = 2$

➤ 代码外提后， $j = 3$

□ **结论：** 循环不变量 $A = B \theta C$ 外提时，要求循环中 $A$ 的所有引用点都是而且仅仅是这个定值点所能到达的。

## 10.3.1 代码外提

### □ 查找循环L的不变运算的算法

- (1) 依次查看L的各基本块的代码，如果它的每个运算对象或为常数，或者定值点在L外，则将此代码标记为“不变运算”。
- (2) 重复第(3)步，直至没有新的代码被标记为“不变运算”为止。
- (3) 依次查看尚未被标记为“不变运算”的代码，如果它的每个运算对象或为常数，或者定值点在L之外，或只有一个能到达的定值点且该点上的代码已标记为“不变运算”，则把查看的代码标记为“不变运算”。

## 10.3.1 代码外提

### □ 代码外提算法

1. 求出循环L的所有不变运算。
2. 对步骤1所求得的每一个不变运算 $s: A = B \theta C$ 或 $A = \theta B$ ，检查是否满足条件：
  - 2.1
    - 2.1.1  $s$ 所在的结点是L的所有出口结点的必经结点；
    - 2.1.2  $A$ 在L中其它地方未再定值；
    - 2.1.3 L中所有 $A$ 的引用点只有 $s$ 中 $A$ 的定值才能到达。
  - 2.2  $A$ 在离开L后不再是活跃的，并且条件2.1.2和2.1.3成立。  $A$ 离开L后不活跃，指 $A$ 在L的任何出口结点的后继结点的入口处是不活跃的。
3. 按步骤1所找出的不变运算的查找顺序，依次把符合条件2.1和2.2的不变运算 $s$ 外提到L的前置结点；但如果 $s$ 的运算对象（ $B$ 或 $C$ ）在L中定值，那么只有当这些定值代码都外提到前置结点中时，才可能把 $s$ 外提。

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

## 10.3.2 强度削弱

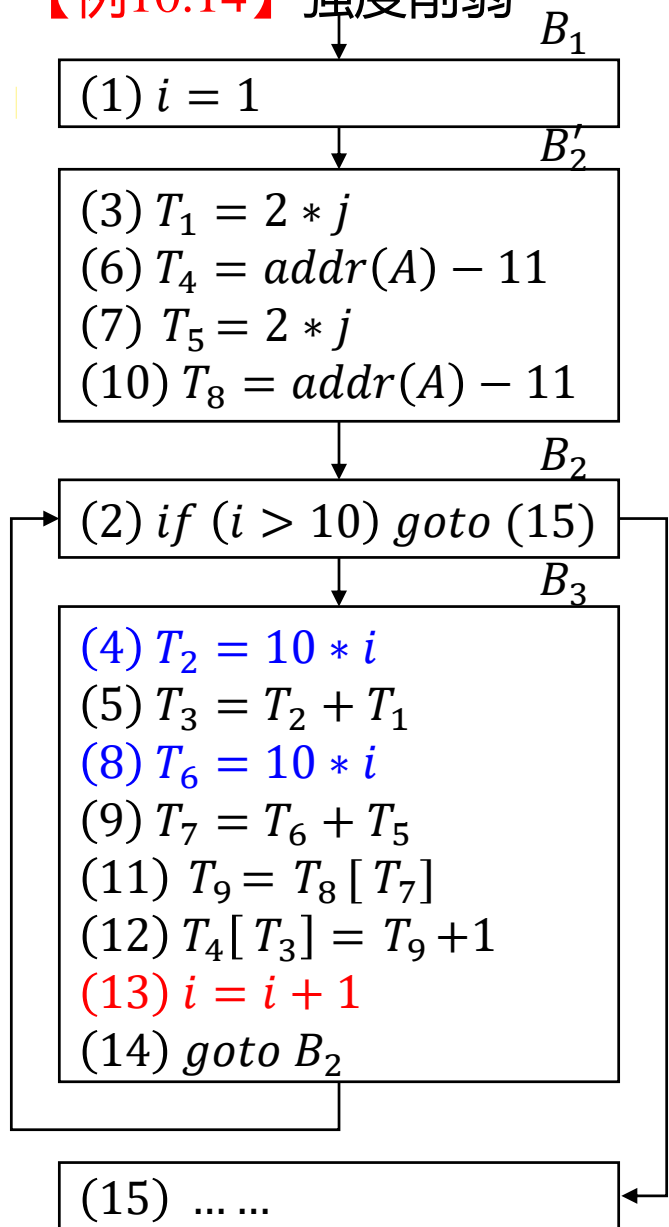
□ **强度削弱**：指把程序中执行时间较长的运算替换为执行时间较短的运算。

➤ 乘法替换为递归加法

➤ 变量替换为常量



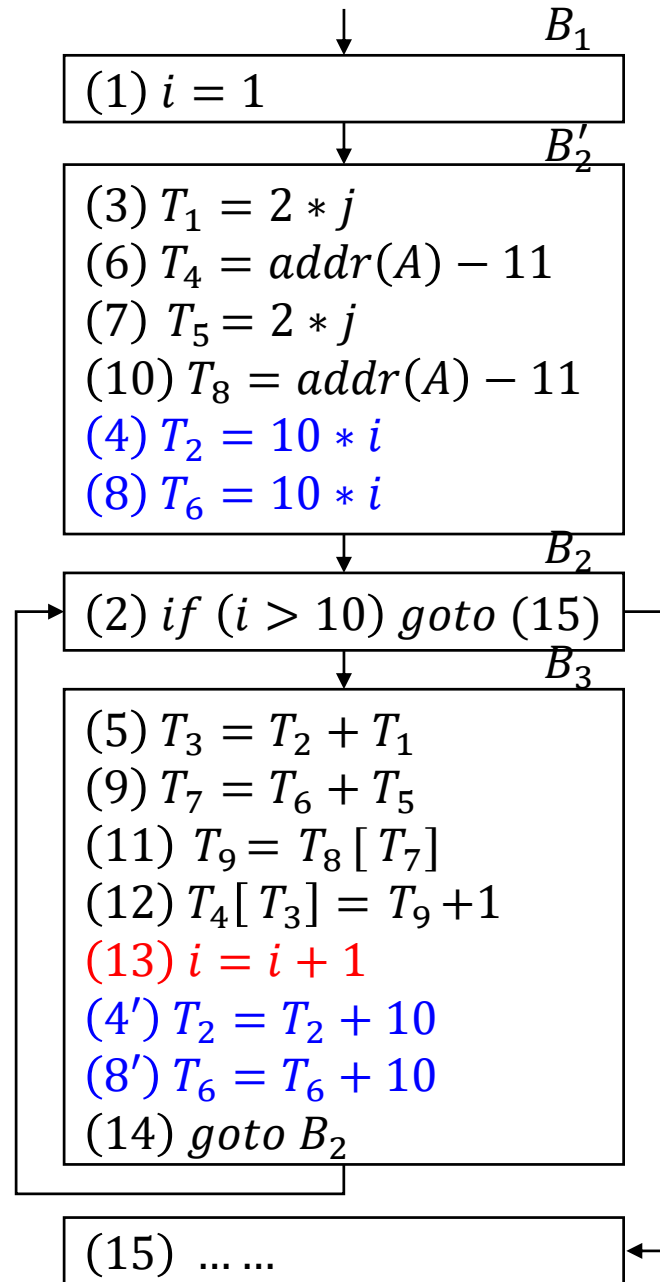
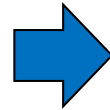
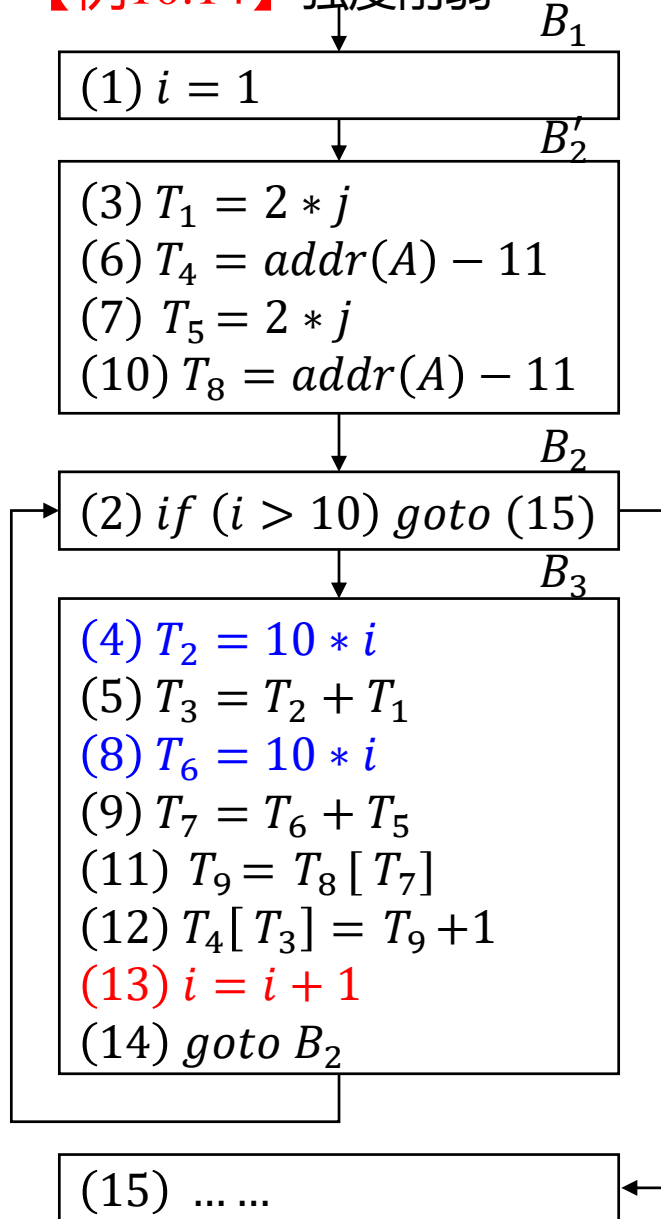
### 【例10.14】强度削弱

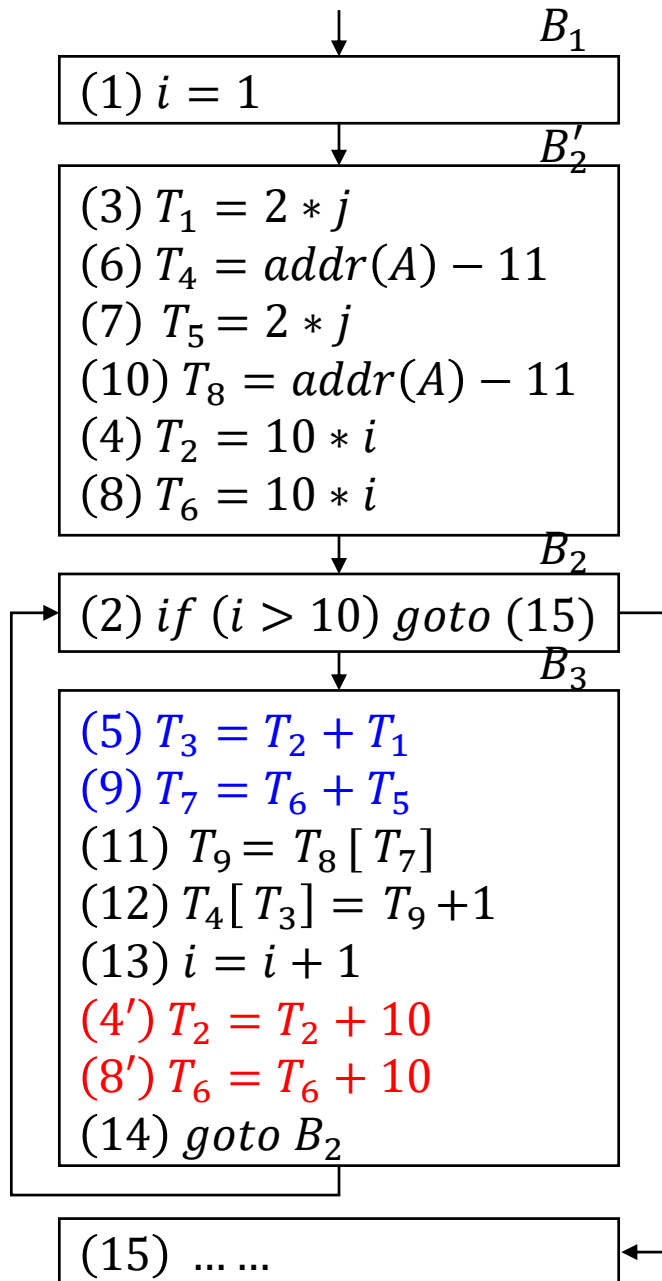


□  $\{B_2, B_3\}$  是循环,  $B_2$  是入口结点。

- (13) 中的  $i$  是一个递归赋值的变量, 每循环一次, 其值增加常量1。
- (4) 和 (8) 中的  $T_2$  和  $T_6$  要引用  $i$  点值, 且都是  $i$  的线性函数。
- 即每循环一次,  $i$  增加常量1,  $T_2$  和  $T_6$  增加常量10。
- 优化: 把 (4) 和 (8) 外提, 在 (13) 后面  $T_2$  和  $T_6$  增10。

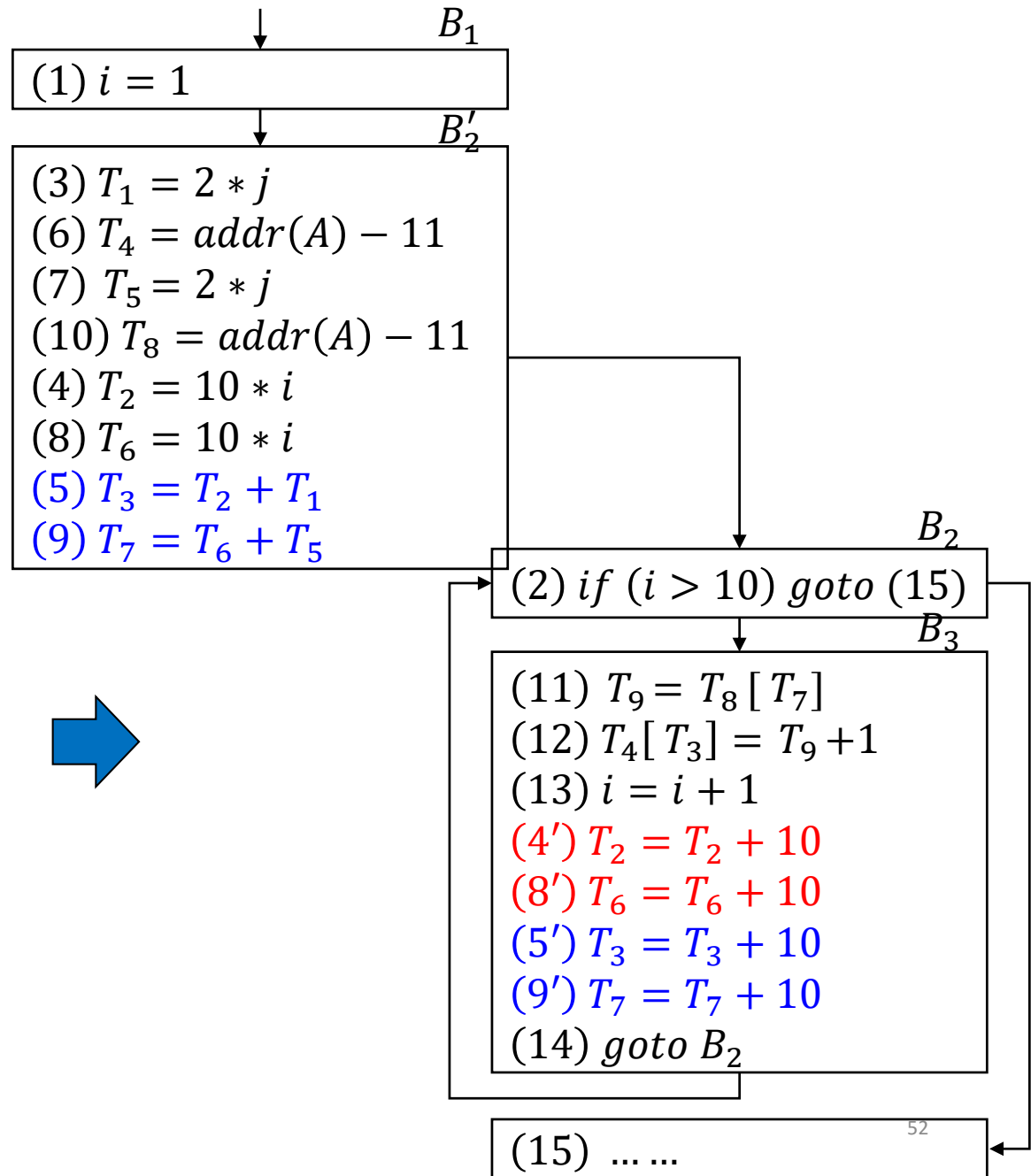
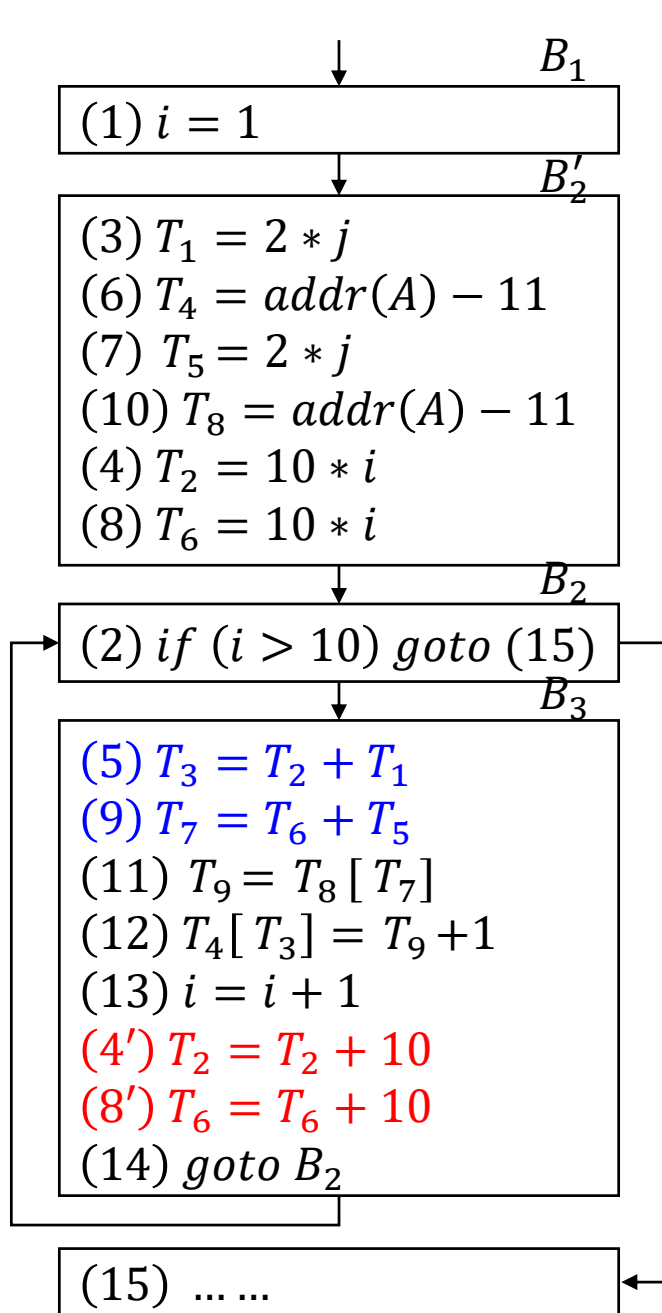
【例10.14】强度削弱





## □ 对加法实行强度削弱

- (4') 和 (8') 中的  $T_2$  和  $T_6$  是递归赋值变量，每循环一次，分别增加一个常量 10。
- (5) 中的  $T_3$  要引用  $T_2$  点值，它的另一个运算对象是循环不变量  $T_1$ ，故每次循环增量也是常量 10。
- (9) 中  $T_7$  增量与  $T_6$  相同。
- 优化：把 (5) 和 (9) 外提，在 (8') 后面分别给  $T_3$  和  $T_7$  增 10。

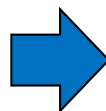


## 10.3.2 强度削弱

### □ 强度削弱

- 如果循环中有 $i$ 的递归赋值 $i = i \pm C$ ，并且循环中 $T$ 的赋值运算可以化归为 $T = k * i \pm C_1$ ，其中 $C, C_1, k$ 均为循环不变量，那么 $T$ 的赋值运算可以进行强度削弱。
- 进行强度削弱后，循环中可能出现一些新的无用赋值，如(4')和(8')，如果 $T_2$ 和 $T_6$ 在循环出口之后不再是活跃变量，则可以删除。
- 循环中下标变量的计算很费时，强度削弱对下标变量的计算强度优化非常有效。

```
(5)  $T_3 = T_2 + T_1$   
(9)  $T_7 = T_6 + T_5$   
(11)  $T_9 = T_8[T_7]$   
(12)  $T_4[T_3] = T_9 + 1$   
(13)  $i = i + 1$   
(4')  $T_2 = T_2 + 10$   
(8')  $T_6 = T_6 + 10$   
(14) goto  $B_2$ 
```



```
(11)  $T_9 = T_8[T_7]$   
(12)  $T_4[T_3] = T_9 + 1$   
(13)  $i = i + 1$   
(4')  $T_2 = T_2 + 10$   
(8')  $T_6 = T_6 + 10$   
(5')  $T_3 = T_3 + 10$   
(9')  $T_7 = T_7 + 10$   
(14) goto  $B_2$ 
```

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

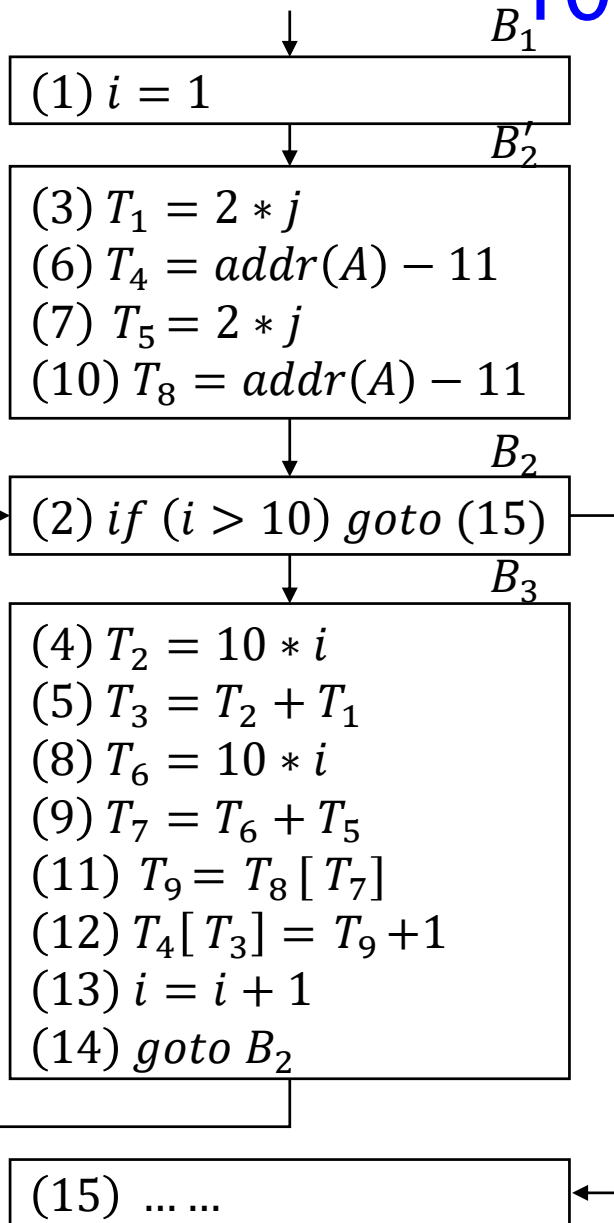
## 10.3.3 删除归纳变量

- **基本归纳变量**：如果循环中对变量 $i$ 只有唯一的形如 $i = i \pm C$ 的赋值，其中 $C$ 是循环不变量，则称 $i$ 为循环中的**基本归纳变量**。
- **归纳变量**：如果 $i$ 是循环中的基本归纳变量， $j$ 在循环中的定值总是可以化归为 $i$ 的线性函数，即 $j = C_1 * i + C_2$ ，其中 $C_1, C_2$ 是循环不变量，则称 $j$ 是**归纳变量**，并称它与 $i$ **同族**。
  - 一个基本归纳变量也是一个归纳变量。

## 10.3.3 删除归纳变量

### 【例10.16】归纳变量

- $i$ 是循环 $\{B_2, B_3\}$ 的**基本归纳变量**。
- $T_2$ 和 $T_6$ 是循环中与 $i$ **同族**的**归纳变量**。
- $T_3$ 在(5)中被唯一定值，根据(5)(4)，有 $T_3 = 10 * i + T_1$ ，而 $T_1$ 是循环不变量，因此 $T_3$ 是循环中与 $i$ **同族**的**归纳变量**。
- $T_7$ 在(9)中被唯一定值，根据(9)(8)，有 $T_7 = 10 * i + T_5$ ，而 $T_5$ 是循环不变量，因此 $T_7$ 是循环中与 $i$ **同族**的**归纳变量**。





## 10.3.3 删除归纳变量

□ 一个基本归纳变量除用于自身的递归定值外，往往只在循环中用来计算其它归纳变量，以及用来控制循环的进行。

➤ 左图为强度削弱后， $i$ 只用来控制循环的进行。

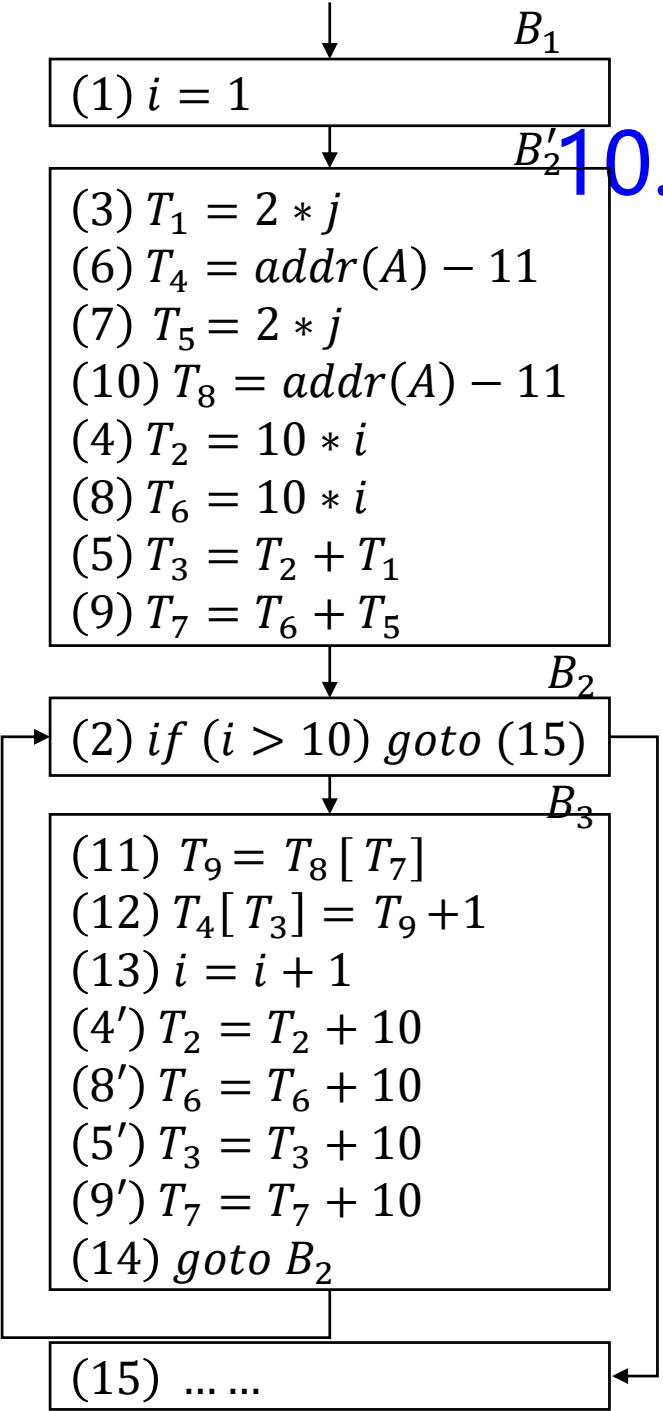
□ 考虑用与 $i$ 同族的某一归纳变量替换循环中的控制条件。

➤ 可选 $T_2, T_3, T_6, T_7$ 。

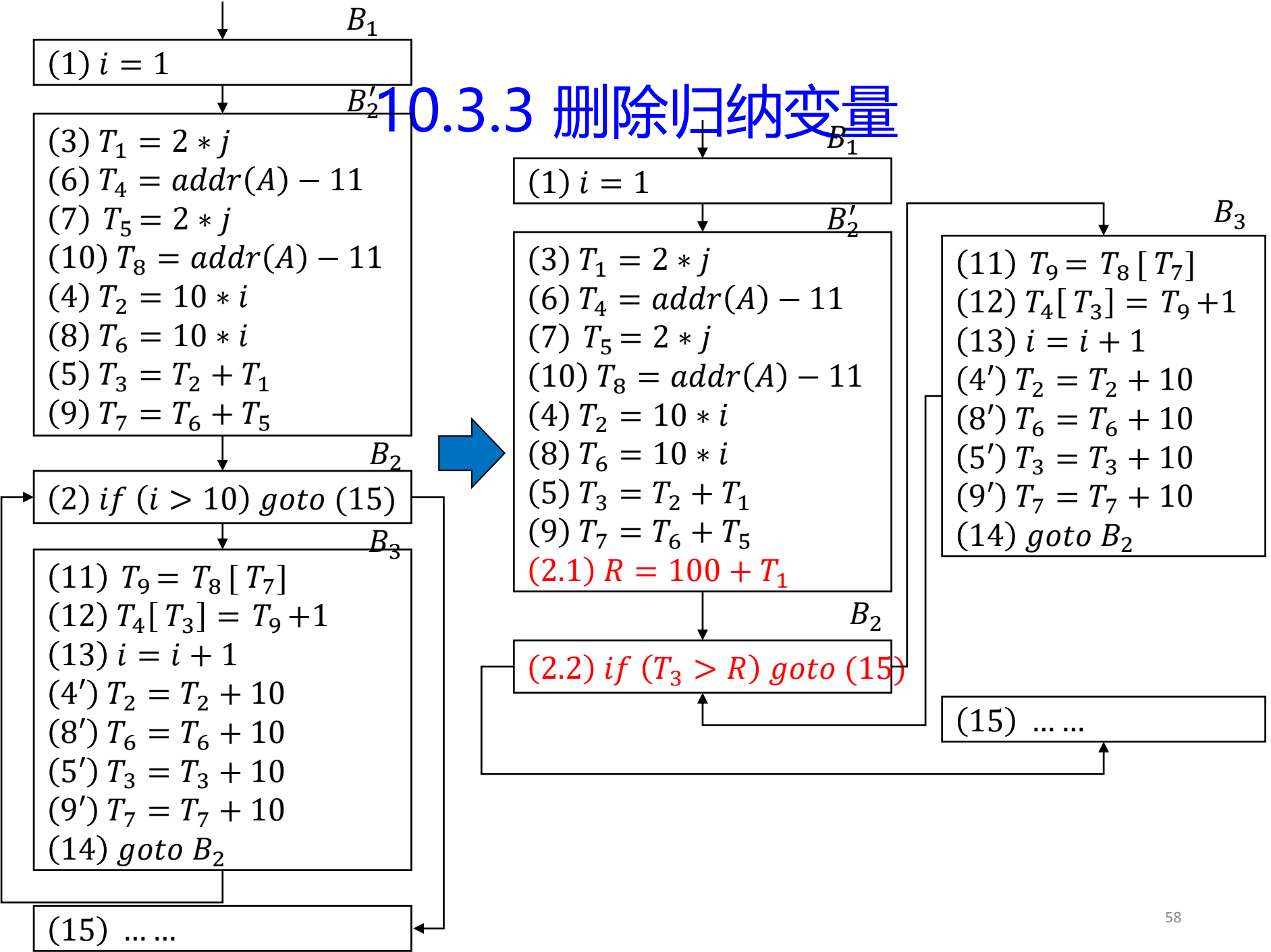
➤ 比如选 $T_3 = 10 * i + T_1$ ，所以： $i > 10 \Leftrightarrow T_3 > 100 + T_1$ ，因此可以替换为：

$$2.1 \ R = 100 + T_1$$

$$2.2 \ \text{if } (T_3 > R) \text{ goto } (15)$$



# 10.3.3 删除归纳变量



## 10.3.3 删除归纳变量

□ 删除归纳变量在强度削弱以后进行，其统一框架为：

- ① 利用循环不变信息，找出循环中的所有基本归纳量。
- ② 找出所有其它归纳变量 $A$ ，并找出 $A$ 与已知基本归纳变量 $X$ 的同族线性函数关系 $F_A(X)$ 。
- ③ 对②中找出的每个归纳变量 $A$ ，进行强度削弱。
- ④ 删除对归纳变量的无用赋值。
- ⑤ 删除基本归纳变量。如果基本归纳变量 $B$ 在循环出口后不是活跃的，并且在循环中，除在自身的递归赋值被引用外，只在形如 $if\ B\ \theta\ y\ goto\ L$ 中被引用，则可选取一与 $B$ 同族的归纳变量 $M$ 来替换 $B$ 进行条件控制。最后删除循环中对 $B$ 的递归赋值代码。

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

## 10.4.1 任意路径数据流分析

□ **全局数据流分析**：分析数据的数值如何在基本块之间进行引用和修改。

- 通常一个程序中基本块的**确切执行次序不能预知**，因此执行数据流分析时，**假定**流图中**所有路径都可能执行**。

□ **以全局活跃变量分析为例-活跃变量集合定义**

- $LiveIn(B)$ ：基本块 $B$ 入口处为活跃的变量的集合；
- $LiveOut(B)$ ：基本块 $B$ 出口处为活跃的变量的集合；
- 令 $S(B)$ 为流图中基本块 $B$ 的**后继**的集合，则有：

$$LiveOut(B) = \bigcup_{B_i \in S(B)} LiveIn(B_i)$$

## 10.4.1 任意路径数据流分析

### □ 定值前引用变量集合

- $LiveUse(B)$ : 基本块 $B$ 中被定值之前引用的变量集合, 由 $B$ 中的语句唯一确定。
- 如果 $v \in LiveUse(B)$ , 则 $v \in LiveIn(B)$ , 即 $LiveIn(B) \supseteq LiveUse(B)$ 。

### □ 定值变量集合

- $Def(B)$ : 基本块 $B$ 中被定值的变量集合, 由 $B$ 中的语句唯一确定。
- 如果 $v \in LiveOut(B) \wedge v \notin Def(B)$ , 则 $v \in LiveIn(B)$ , 即

$$LiveIn(B) \supseteq LiveOut(B) - Def(B)$$

## 10.4.1 任意路径数据流分析

### □ 由基本块语句唯一确定的集合

- $LiveUse(B)$ : 基本块 $B$ 中被定值之前引用的变量集合;
- $Def(B)$ : 基本块 $B$ 中被定值的变量集合。

### □ 计算方程

- $LiveOut(B) = \bigcup_{B_i \in S(B)} LiveIn(B_i)$ , 没有后继则为空。
- $LiveIn(B) \supseteq LiveUse(B)$
- $LiveIn(B) \supseteq LiveOut(B) - Def(B)$

### □ 求解

- $LiveIn(B) = LiveUse(B) \cup (LiveOut(B) - Def(B))$

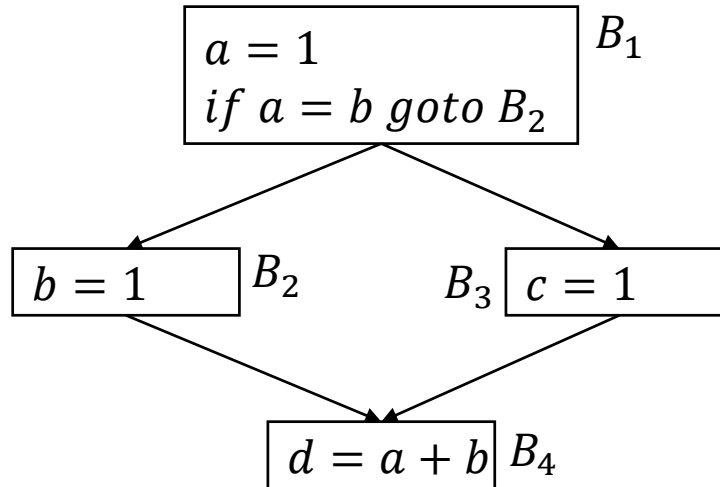
### □ 求解方向

- 反向流(backward-flow): 信息流的方向与控制流相反
- 前向流(forward-flow): 信息流的方向与控制流一致

# 10.4.1 任意路径数据流分析

## 【例10.17】程序段

```
a = 1;
if a = b then b = 1;
else c = 1;
d = a + b;
```



基本块	Def	LiveUse
$B_1$	$\{a\}$	$\{b\}$
$B_2$	$\{b\}$	$\phi$
$B_3$	$\{c\}$	$\phi$
$B_4$	$\{d\}$	$\{a, b\}$

基本块	LiveIn	LiveOut
$B_1$	$\{b\}$	$\{a, b\}$
$B_2$	$\{a\}$	$\{a, b\}$
$B_3$	$\{a, b\}$	$\{a, b\}$
$B_4$	$\{a, b\}$	$\phi$

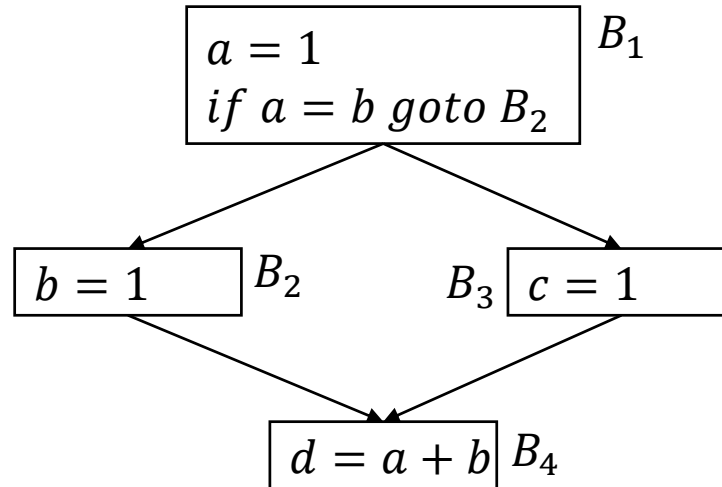
- $LiveIn(B_4) = LiveUse(B_4) \cup (LiveOut(B_4) - Def(B_4)) = \{a,$
- $LiveOut(B_2) = LiveIn(B_4) = \{a, b\}$
- $LiveOut(B_3) = LiveIn(B_4) = \{a, b\}$
- $LiveIn(B_2) = LiveUse(B_2) \cup (LiveOut(B_2) - Def(B_2)) = \{a\}$
- $LiveIn(B_3) = LiveUse(B_3) \cup (LiveOut(B_3) - Def(B_3)) = \{a, b\}$
- $LiveOut(B_1) = LiveIn(B_2) \cup LiveIn(B_3) = \{a, b\}$
- $LiveIn(B_1) = LiveUse(B_1) \cup (LiveOut(B_1) - Def(B_1)) = \{b\} \cup (\{a, b\} - \{a\}) = \{b\}$



## 10.4.1 任意路径数据流分析

### 【例10.17】程序段

```
a = 1;  
if a = b then b = 1;  
else c = 1;  
d = a + b;
```



基本块	Def	LiveUse
$B_1$	$\{a\}$	$\{b\}$
$B_2$	$\{b\}$	$\phi$
$B_3$	$\{c\}$	$\phi$
$B_4$	$\{d\}$	$\{a, b\}$

基本块	LiveIn	LiveOut
$B_1$	$\{b\}$	$\{a, b\}$
$B_2$	$\{a\}$	$\{a, b\}$
$B_3$	$\{a, b\}$	$\{a, b\}$
$B_4$	$\{a, b\}$	$\phi$

### □ 活跃变量分析的另一种用法

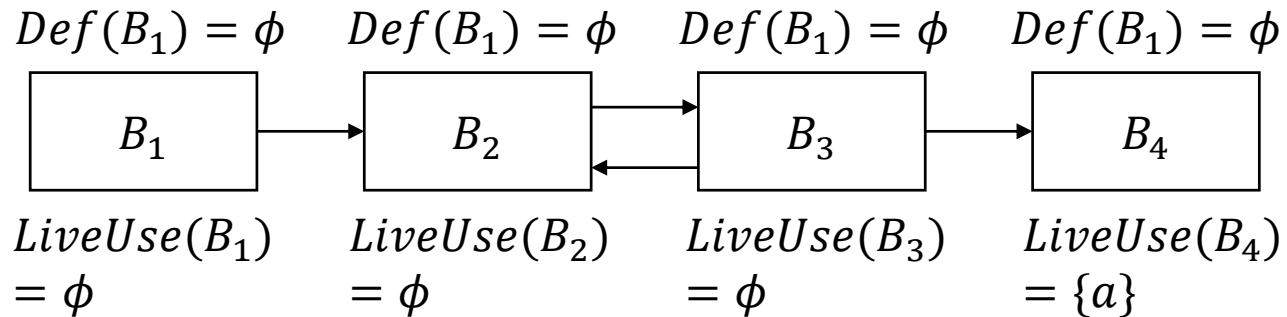
- $LiveIn(B_1) = \{b\}$ : 变量在程序起始基本块的入口处是活跃的, 则变量可能在定值之前被引用。

## 10.4.1 任意路径数据流分析

- 如果规定流图只有一个**唯一的开始结点**（无前驱），并且有一个或多个**结束结点**（无后继），则数据流方程是**可解**的。
  - 从基本块所产生的值`LiveUse`开始；
  - 这些值**反向传播**；
  - 除掉基本块内死了的值（`Def`）；
  - 一直迭代直到求出所有集合。

## 10.4.1 任意路径数据流分析

【例10.18】数据流问题的解不唯一



□ 一个明显的解:  $LiveIn(B_i) = \{a\}$

□ 一个不太合理的解

基本块	LiveIn	LiveOut
$B_1$	$\{a, b\}$	$\{a, b\}$
$B_2$	$\{a, b\}$	$\{a, b\}$
$B_3$	$\{a, b\}$	$\{a, b\}$
$B_4$	$\{a\}$	$\phi$

## 10.4.1 任意路径数据流分析

### □ 数据流方程的两个观点

- 悲观观点：如果在所有后继结点中**没有**看到明显的**定值**，就认为这些变量是活跃的；
- 乐观观点：只有看到一个变量在某个后继基本块中**被引用**了，才认为这个变量是活跃的。
- 乐观观点是**最小有效解**。就优化目的而言，最小有效解是合理的，因为活跃变量的值要保存，而死变量的值可以忽略。

## 10.4.1 任意路径数据流分析

### □ 前向数据流分析：变量初始化检测问题

- $UninitIn(B)$ ：基本块 $B$ 入口处可能未被初始化的变量集，如果一个基本块没有前驱，则其 $UninitIn$ 集合包含所有变量；
- $UninitOut(B)$ ：基本块 $B$ 出口处可能未被初始化的变量集。
- 令 $P(B)$ 为流图中基本块 $B$ 的前驱的集合，则有：

$$UninitIn(B) = \bigcup_{B_i \in P(B)} UninitOut(B_i)$$

## 10.4.1 任意路径数据流分析

### □ 初始化变量集合

- $Init(B)$ : 在 $B$ 的出口处已被初始化的变量集合。
- $UninitOut(B) \supseteq UninitIn(B) - Init(B)$

### □ 基本块中变为未初始化集合 $Uninit(B)$

- 被赋予一个非法值, 如 $null$ ;
- 一个操作的副作用, 如释放一个对象;
- 刚刚建立一个变量。
- 显然:  $UninitOut(B) \supseteq Uninit(B)$

## 10.4.1 任意路径数据流分析

### □ 由基本块语句唯一确定的集合

- $Init(B)$ : 在 $B$ 中被初始化的变量集合;
- $Uninit(B)$ : 在 $B$ 中变为未初始化的变量集合。

### □ 计算方程

- $UninitIn(B) = \bigcup_{B_i \in P(B)} UninitOut(B_i)$ , 没有前驱则包含所有变量。
- $UninitOut(B) \supseteq UninitIn(B) - Init(B)$
- $UninitOut(B) \supseteq Uninit(B)$

### □ 求解

- $UninitOut(B) = Uninit(B) \cup (UninitIn(B) - Init(B))$

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化



## 10.4.2 全路径数据流分析

□ **任意路径问题**：存在某条路径为真，则问题为真。

- **活跃变量检测**：存在某条路径上变量被引用，则认为变量是活跃的。
- **变量初始化检测**：存在任何一条路径上变量没有适当初始化，就认为这个变量是未初始化的。

□ **全路径问题**：所需性质需要在所有路径上都满足。

- 确定**表达式的可用性**（availability），是一个全路径前向数据流问题；
- 称**表达式是可用的**（available），如果它已经被计算且重新计算是多余的；
- 可用性信息对实现全局公共子表达式优化非常重要。

## 10.4.2 全路径数据流分析

□ 确定基本块内对表达式的计算，在基本块出口处是否可用。

- $RelVar(T)$ ：表达式计算的相关变量集合，结果保存在临时变量 $T$ 中。
- 函数 $ComputeRelVar(T)$ ：递归的把临时变量替换为计算这个临时变量的变量和临时变量，直到只剩下变量。

```
function ComputeRelVar(T)
```

```
{  
     $RealVar(T) = \{T\}$ ;  
    while 存在临时变量  $T' \in RealVar(T)$   
        把  $RealVar(T)$  中的  $T'$  替换为计算  $T'$  的变量和临时变量;  
}
```

## 10.4.2 全路径数据流分析

### □ 可用临时变量集合

- $AvailOut(B)$ : 在 $B$ 的出口处可用的临时变量集合;
- $AvailIn(B)$ : 在 $B$ 的入口处可用的临时变量集合, 第一个基本块入口处无可用表达式;
- $AvailIn(B) = \bigcap_{B_i \in P(B)} AvailOut(B_i)$

### □ 基本块语句唯一确定的集合

- $Killed(B)$ : 在基本块 $B$ 内对相关变量赋值而被杀死的表达式集合;
- $Computed(B)$ : 在基本块 $B$ 内被计算而且没被杀死的表达式集合;

### □ 求解

- $AvailOut(B) = Computed(B) \cup (AvailIn(B) - Killed(B))$

## 10.4.2 全路径数据流分析

### □ 反向全路径数据流分析：确定非常忙表达式

- 非常忙表达式：表达式在被杀死之前，所有路径上都要引用这个表达式的值；
- 非常忙表达式为寄存器分配的主要候选，因为知道它的值是必须要引用点；
- 非常忙表达式也用于指导代码外提。

## 10.4.2 全路径数据流分析

### □ 由基本块语句唯一确定的集合

- $Used(B)$ : 在 $B$ 中被杀死之前引用点表达式集合;
- $Killed(B)$ : 在 $B$ 中被引用之前杀死了的表达式集合。

### □ 计算方程

- $VeryBusyOut(B)$ , 在基本块 $B$ 的出口处非常忙的表达式集合;
- $VeryBusyIn(B)$ , 在基本块 $B$ 的入口处非常忙的表达式集合;
- $VeryBusyOut(B) = \bigcap_{B_i \in S(B)} VeryBusyIn(B_i)$

### □ 求解

- $VeryBusyIn(B) = Used(B) \cup (VeryBusyOut(B) - Killed(B))$

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

## 10.4.3 数据流问题的分类

### □ 数据流问题的四个集合

- $Gen(B)$ : 在 $B$ 中生成的集合, 由基本块语句唯一确定;
- $Killed(B)$ : 在 $B$ 中被杀死的集合, 由基本块语句唯一确定;
- $In(B)$ :  $B$ 的入口处的集合, 前向流需要确定开始基本块的该集合;
- $Out(B)$ :  $B$ 的出口处的集合, 反向流需要确定结束基本块的该集合。

	前向流	反向流
任意路径	$Out(B) = Gen(B) \cup (In(B) - Killed(B))$ $In(B) = \bigcup_{B_i \in P(B)} Out(B_i)$	$In(B) = Gen(B) \cup (Out(B) - Killed(B))$ $Out(B) = \bigcup_{B_i \in S(B)} In(B_i)$
全路径	$Out(B) = Gen(B) \cup (In(B) - Killed(B))$ $In(B) = \bigcap_{B_i \in P(B)} Out(B_i)$	$In(B) = Gen(B) \cup (Out(B) - Killed(B))$ $Out(B) = \bigcap_{B_i \in S(B)} In(B_i)$

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化



# 引用-定值链 (UD链)

□ **到达定值集合**：一个变量 $v$ 的定值到达 $v$ 的某个引用点，如果存在一条从 $v$ 的这个定值到引用点的路径，并且中间没有对 $v$ 的重新定值。

- **规则1**：在基本块 $B$ 中，变量 $A$ 的引用点 $u$ 之前有 $A$ 的定值点 $d$ ，且 $d$ 点所定 $A$ 值能到达点 $u$ ，则 $A$ 在 $u$ 点的UD链为 $\{d\}$ 。
- **规则2**：在基本块 $B$ 中，变量 $A$ 的引用点 $u$ 之前没有 $A$ 的定值点，则包含在 $In(B)$ 中的全部 $A$ 的定值点均可到达 $u$ ，故 $In(B)$ 中的这些 $A$ 的定值点组成 $A$ 在 $u$ 点的UD链。

## □ 引用-定制链问题为前向流问题

- $Gen(B)$ ：出现在 $B$ 中且到达基本块尾的定值，由基本块语句唯一确定；
- $Killed(B)$ ：出现在 $B$ 中排除掉 $Gen(B)$ 的定值，擦掉了被基本块内局部定值取代了的那些定值，由基本块语句唯一确定；
- $In(B)$ ： $B$ 的入口处的集合，第一个基本块的该集合为空；
- $Out(B)$ ： $B$ 的出口处的集合。

# 定值-引用链 (DU链)

- **定值-引用链**：一个变量 $A$ 的定值点 $p$ ，从 $p$ 出发能到达的全部 $A$ 的引用点组成的集合，称为 $A$ 在定值点 $P$ 的定值-引用链，简称DU链。
- **定制-引用链问题为反向流问题**
  - $Gen(B)$ ：形如 $(s, A)$ 的元素，其中 $s$ 是变量 $A$ 在基本块 $B$ 中点引用点，且从 $B$ 的入口点到 $s$ 之前无变量 $A$ 的定值点；
  - $Killed(B)$ ：形如 $(s, A)$ 的元素，其中变量 $A$ 是在基本块 $B$ 中定值，而 $s$ 是基本块 $B$ 外变量 $A$ 的引用点；
  - $In(B)$ ： $B$ 的入口处的、可能引用变量当前值的那些中间代码集合；
  - $Out(B)$ ： $B$ 的出口处的、可能引用变量当前值的那些中间代码集合，最后一个基本块的该集合为空。

# 第十章 优化

## □ 10.1 概述

## □ 10.2 局部优化

- 10.2.1 基本块及流图
- 10.2.2 基本块的DAG表示及其应用

## □ 10.3 循环优化

- 10.3.1 代码外提
- 10.3.2 强度削弱
- 10.3.3 删除归纳变量

## □ 10.4 数据流分析

- 10.4.1 任意路径数据流分析
- 10.4.2 全路径数据流分析
- 10.4.3 数据流问题的分类
- 10.4.4 其它主要数据流问题
- 10.4.5 利用数据流信息进行全局优化

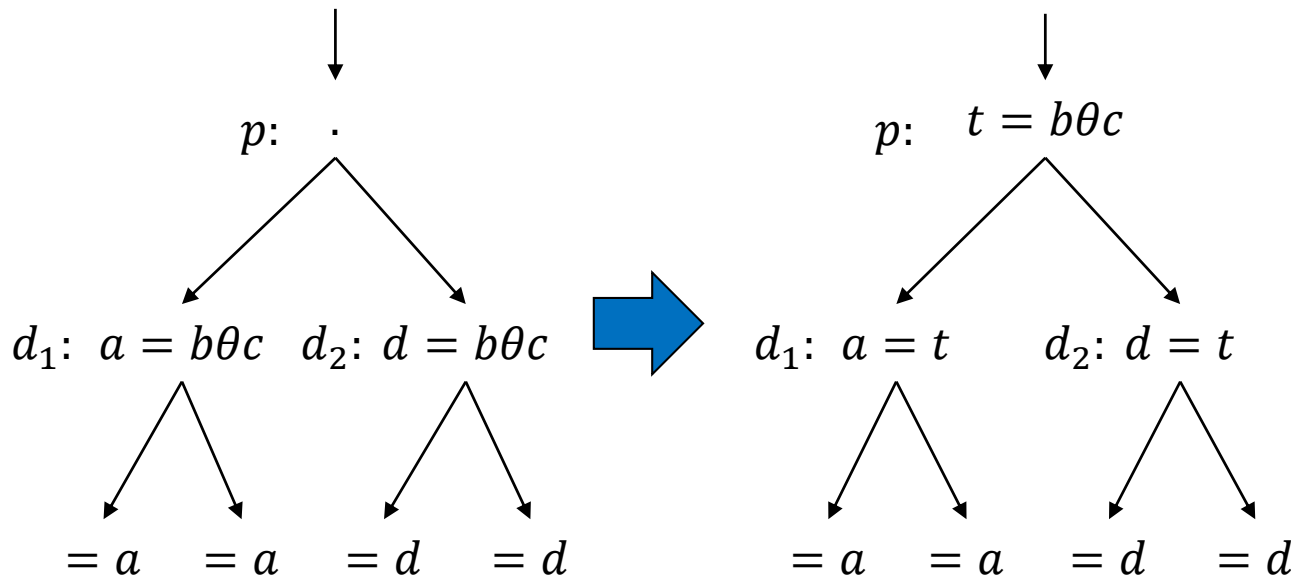
## 10.4.5 利用数据流信息进行全局优化

	前向流		反向流	
	问题	初始值	问题	初始值
任意路径	到达定值 (UD链)	$\emptyset$	活跃变量	$\emptyset$
	未初始化变量	所有变量	DU链	$\emptyset$
全路径	有效表达式	$\emptyset$	非常忙表达式	$\emptyset$
	复写传播	$\emptyset$		

# 非常忙表达式

❑ **非常忙表达式**：如果从程序中某点 $p$ 开始的任何一条通路上，在对 $b$ 或 $c$ 进行定值前，都要计算表达式 $b \theta c$ ，则称表达式在 $p$ 点非常忙。

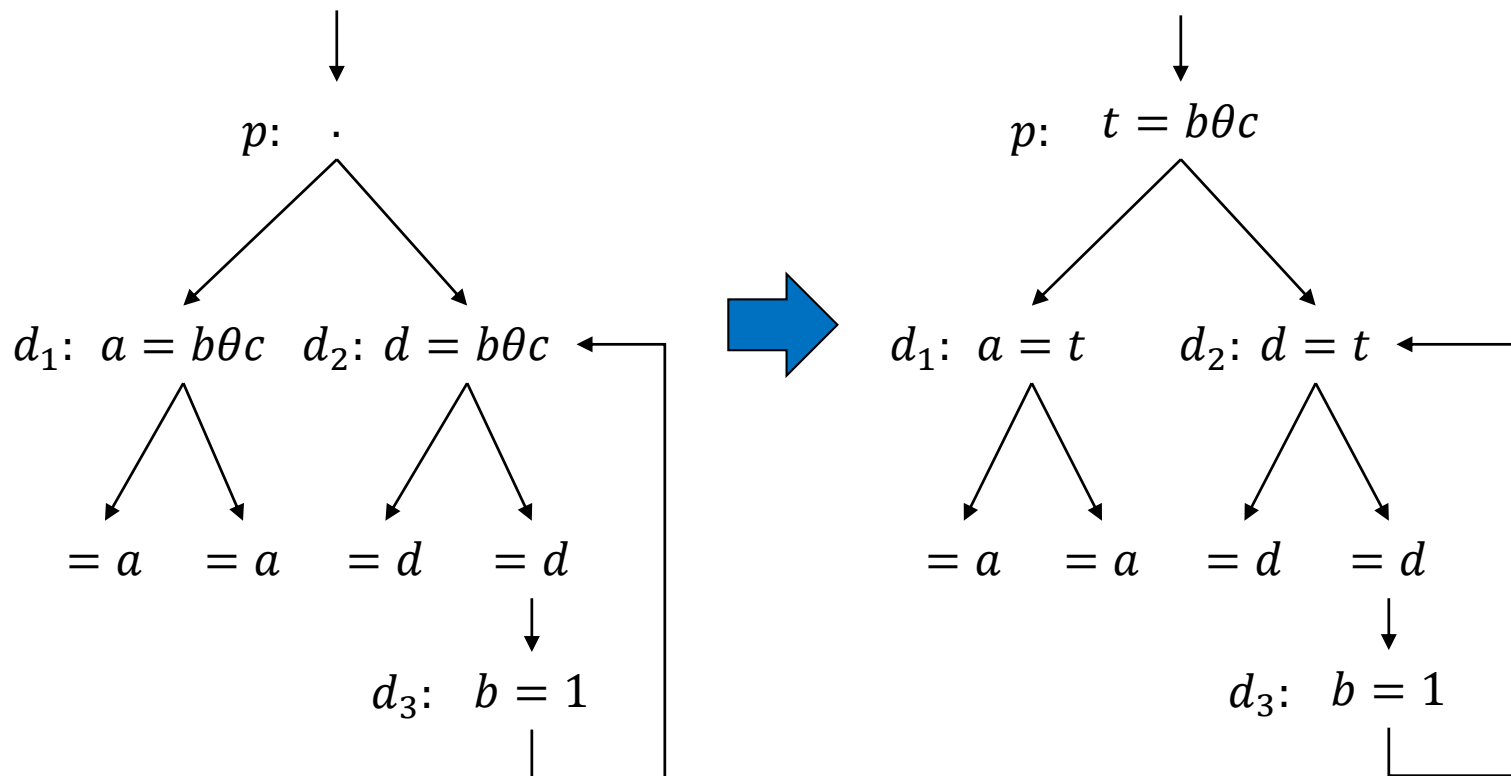
- 非常忙的循环不变运算是**代码外提**到极好候选；
- 非常忙表达式可以用来进行**代码提升**。



这步本身未使程序得到改善，但为复写传播优化提供了基础。

# 非常忙表达式

❑ 该变换不能保证变换后的程序与变换前的程序等价。



对变换的每一 $(d)a = b\theta c$ , 除了 $b\theta c$ 在点 $p$ 非常忙的条件外, 还要求任何能到达 $d$ 的 $b$ 和 $c$ 的定值, 必须首先经过 $p$ 。

# 删除全局公共子表达式

```
procedure RemoveGlobalCSEs()
{
    计算全局公共子表达式集合GlobalCSE;
    foreach 表达式E in GlobalCSE
    {
        对E进行可用表达式数据流分析;
        给E分配一个临时单元, 记作t(E);
    }

    foreach 基本块B
        foreach 表达式E in GlobalCSE
            if  $E \in B \wedge E$ 在B的入口处可用
                删除B中E的第一次计算, 并把它替换为引用t(E)。
}
```

# 活跃变量分析

```
procedure RemoveDeadStores() // 删除死变量赋值
{
    foreach 基本块B
        foreach 代表变量或公共子表达式的单元v
        {
            if B中对v的赋值是在所有对v的引用之后
                对v进行活跃变量分析;
            if B出口处v不再活跃
                从B中删除对v的赋值;
        }
    }
```



# 未初始化变量分析

```
procedure FindUninitializedVars()
```

```
{
```

```
    进行未初始化变量数据流分析;
```

```
    foreach 基本块B
```

```
        for B中变量v的每次使用
```

```
        {
```

```
            if ((这是B中v的第一次引用 且 在B的入口处v未初始化)
```

```
                or (v的最后一次引用使v变为未初始化) )
```

```
                发出v未被初始化的警告;
```

```
        }
```

```
}
```

# 常量传播和复写传播

```
procedure Propagate()
```

```
{
```

```
    进行到达定值数据流分析;
```

```
    进行UD链数据流分析;
```

```
    进行复写传播数据流分析;
```

```
    标记程序中所有变量引用;
```

```
    for 变量 $v$ 的每个被标记的引用
```

```
        去掉 $v$ 的这个引用标记;
```

```
        if 到达 $v$ 的这个引用的唯一定值为 $v = c$ , 这里 $c$ 为常量 {
```

```
            用 $c$ 代替 $v$ 的这个引用, 并尽量简化表达式;
```

```
            if 这个替换和简化建立了一个常量赋值 $x = k$  {
```

```
                用 $x = k$ 代替原来的赋值;
```

```
                标记这个赋值可以到达的 $x$ 的所有引用;
```

```
            }
```

```
        从 $v = c$ 的DU链中去掉对 $v$ 的这个引用;
```

```
}
```

# 常量传播和复写传播

else if 复写传播分析表明到达 $v$ 的这个引用的唯一定值为 $v = x$ , 这里 $x$ 为变量

```
{
```

```
    用 $x$ 代替 $v$ 的这个引用;
```

```
    从 $v = x$ 的DU链中去掉对 $v$ 的这个引用;
```

```
}
```

for 变量 $v$ 的每个定值

if 这个定值的所有引用都因常量或复写传播而消除

从程序中删除这个定值;

for 每个变量

if 这个变量的所有引用已被消除

从程序中删除这个变量;

```
}
```