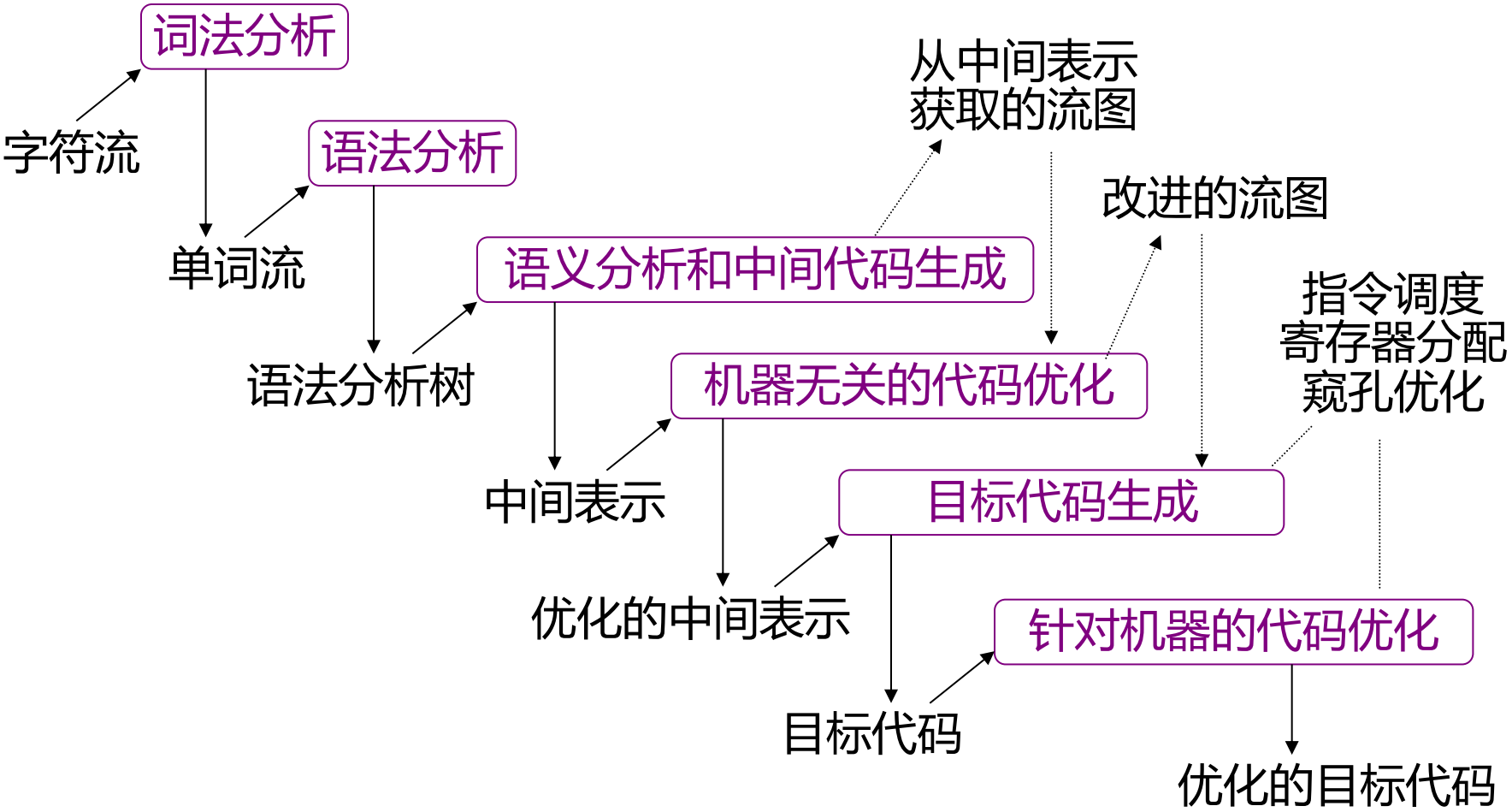
第10章

代码优化和目标代码生成

✧ 二者在编译程序中的逻辑位置



什么是代码优化？

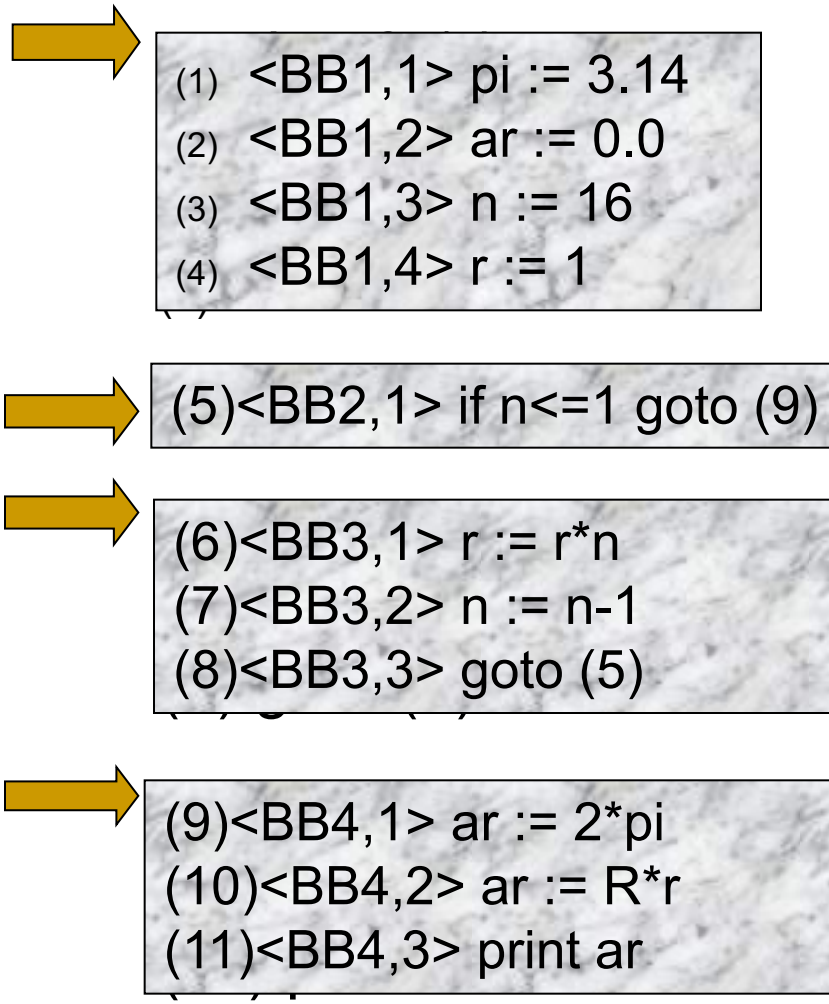
在各级中间代码以及目标代码的层次上，**进行等价变换,使得变换前后的代码在逻辑上是相等的，但后者比前者效率更高(运行速度更快，或占用空间更少，或使用能量更少)。**

10.1 基本块、流图和循环

基本块

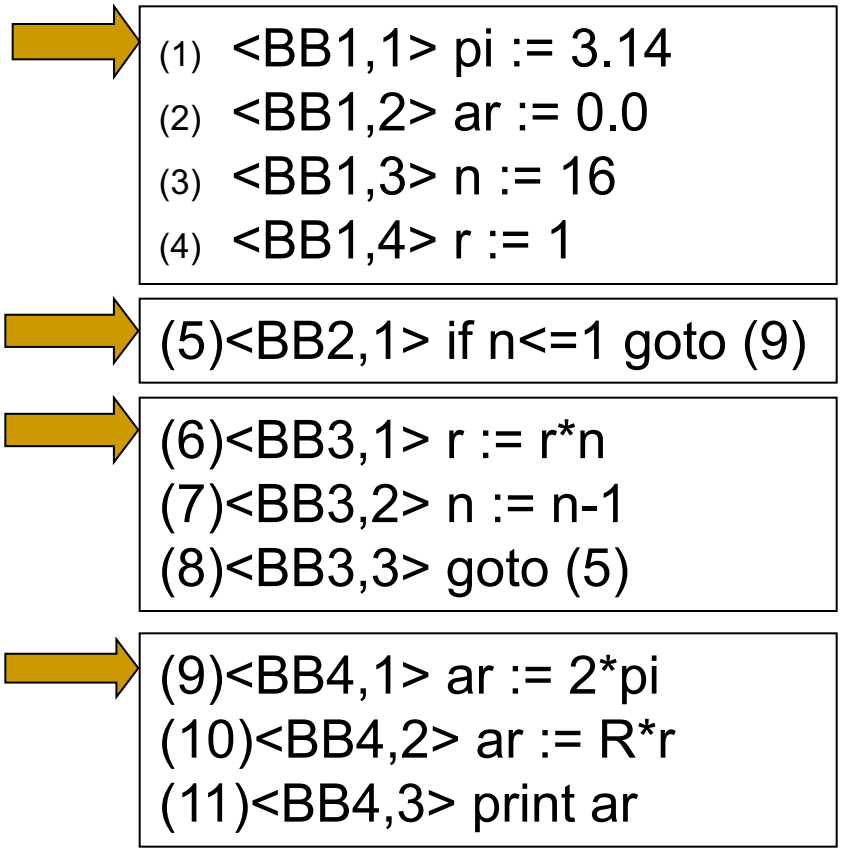
- 程序中一个顺序执行的语句序列
- 只有一个入口语句（第一个语句）和一个出口语句（最后一个语句）
- 除入口语句外其他语句均不可以带标号
- 除出口语句外其他语句均不可能是转移或停语句
- 对于一个基本块来说，执行时只能从其入口进入，从其出口退出。

- 入口语句
  - 程序的第一个语句；或者，
  - 条件转移语句或无条件转移语句的转移目标语句；或者，
  - 紧跟在条件转移语句后面的语句。



划分基本块的算法

- 求出四元式程序之中各个基本块的入口语句
- 对每一入口语句，构造其所属的基本块。它是由该语句到下一入口语句（不包括下一入口语句），或到一转移语句（包括该转移语句），或到一停语句（包括该停语句）之间的语句序列组成的
- 凡未被纳入某一基本块的语句，都是程序中控制流程无法到达的语句，因而也是不会被执行到的语句，我们可以把它们删除。



流图

- 以构成程序的基本块为结点，添加控制流信息而得到的有向图，称为流图。

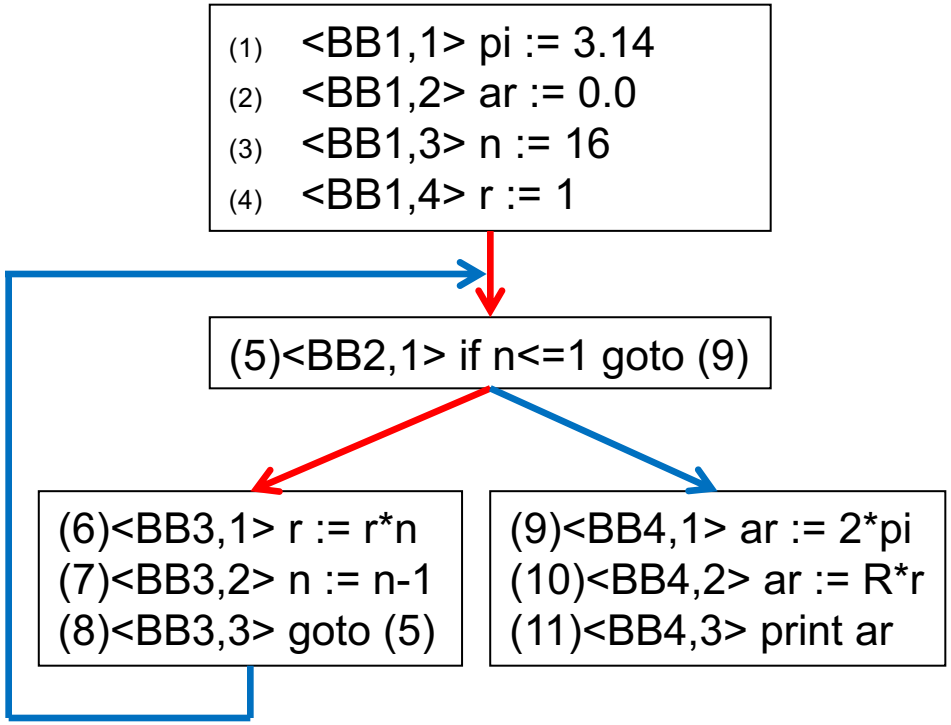
首结点：

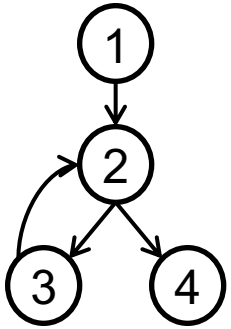
包含程序第一个语句的基本块，具有唯一性。

有向边(i → j)：

j是i之后的相邻基本块，且i的出口语句不是无条件跳转语句或停语句或返回语句。

i的出口是goto L或if … goto L，而L是j的入口语句 。





2的前趋：1，3
2的后继：3，4

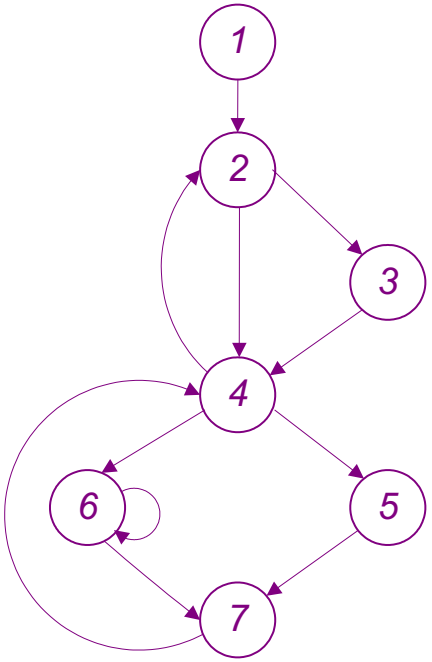
循环

支配结点集

如果从流图的首结点出发,到达 n 的任意通路都要经过 m，则称 m 支配 n, 或 m 是 n 的支配结点，记为 m DOM n。

结点n 的所有支配结点的集合, 称为结点 n 的支配结点集，记为D(n)。

D(1)={1}
D(2)={1,2}
D(3)={1,2,3}
D(4)={1,2,4}
D(5)={1,2,4,5}
D(6)={1,2,4,6}
D(7)={1,2,4,7}



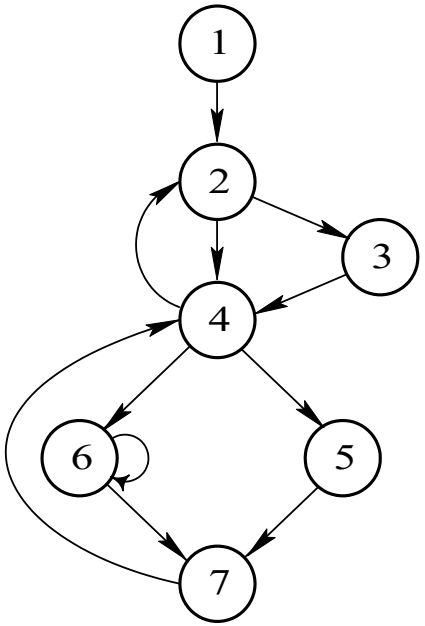
- 回边

  假设a→b是流图中的一条有向边，如果b DOM a，则称a→b是流图中的一条回边。

- 循环

  对于回边n→d，组成的循环是由结点d和结点n以及有通路到达n而该通路不经过d的所有结点组成，并且d是该循环的唯一入口结点。

回边　　循环

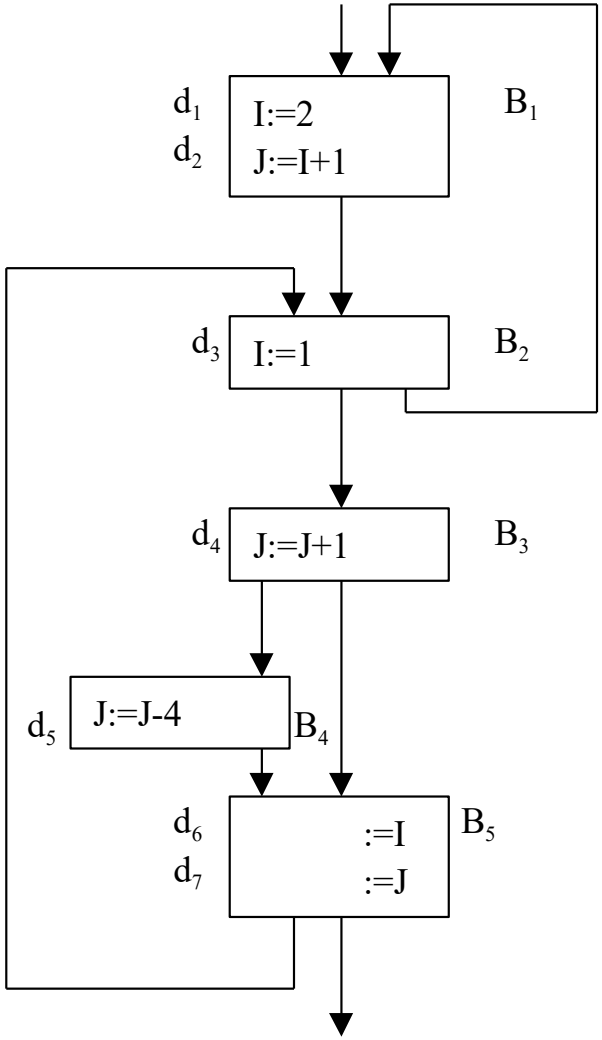


- 循环的性质：

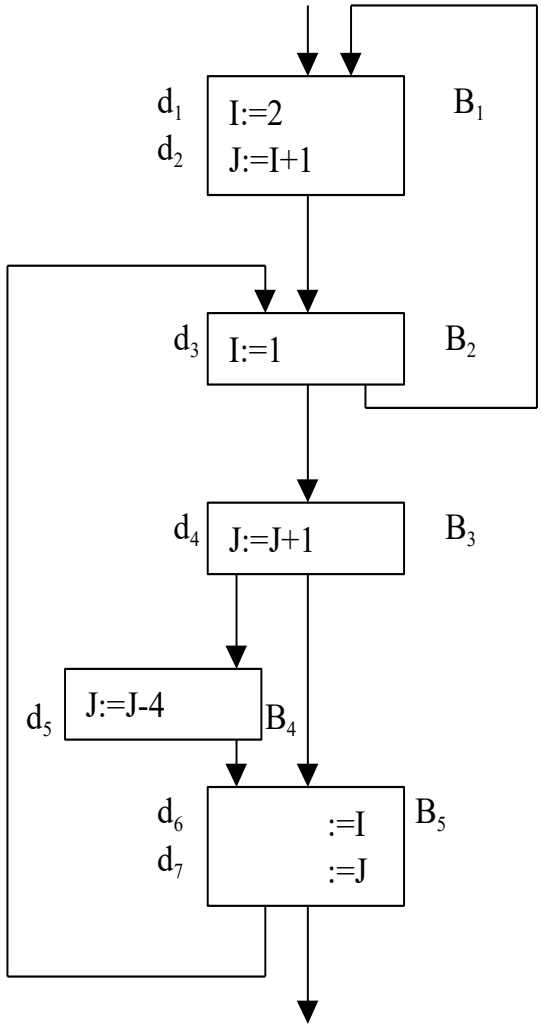
1.强连通的（任意两结点间，必有一条通路，且该通路上各结点都属于该结点序列）

2.它们中间有且只有一个是入口结点。

数据流分析简介



到达一定值



到达一定值

ud链例：变量I，J的ud链

变量	引用点	ud 链
I	d2	d1
	d6	d3
J	d4	d2,d4,d5
	d5	d4
	d7	d4,d5

10.3 代码优化技术

- 引例

P:=0； i:=1 ；

do

{P:=P+A[I]*B[I];i:=i+1； }

while i<=20;

(1)P:=0
(2)I:=1
(3)T_1:=4*I
(4)T_2:=addr(A)-4
(5)T_3:=T_2[T_1]
(6)T4:=4*I
(7)T_5:=addr(B)-4
(8)T_6:=T5[T4]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(11)I:=I+1
(12)if I<=20 goto(3)

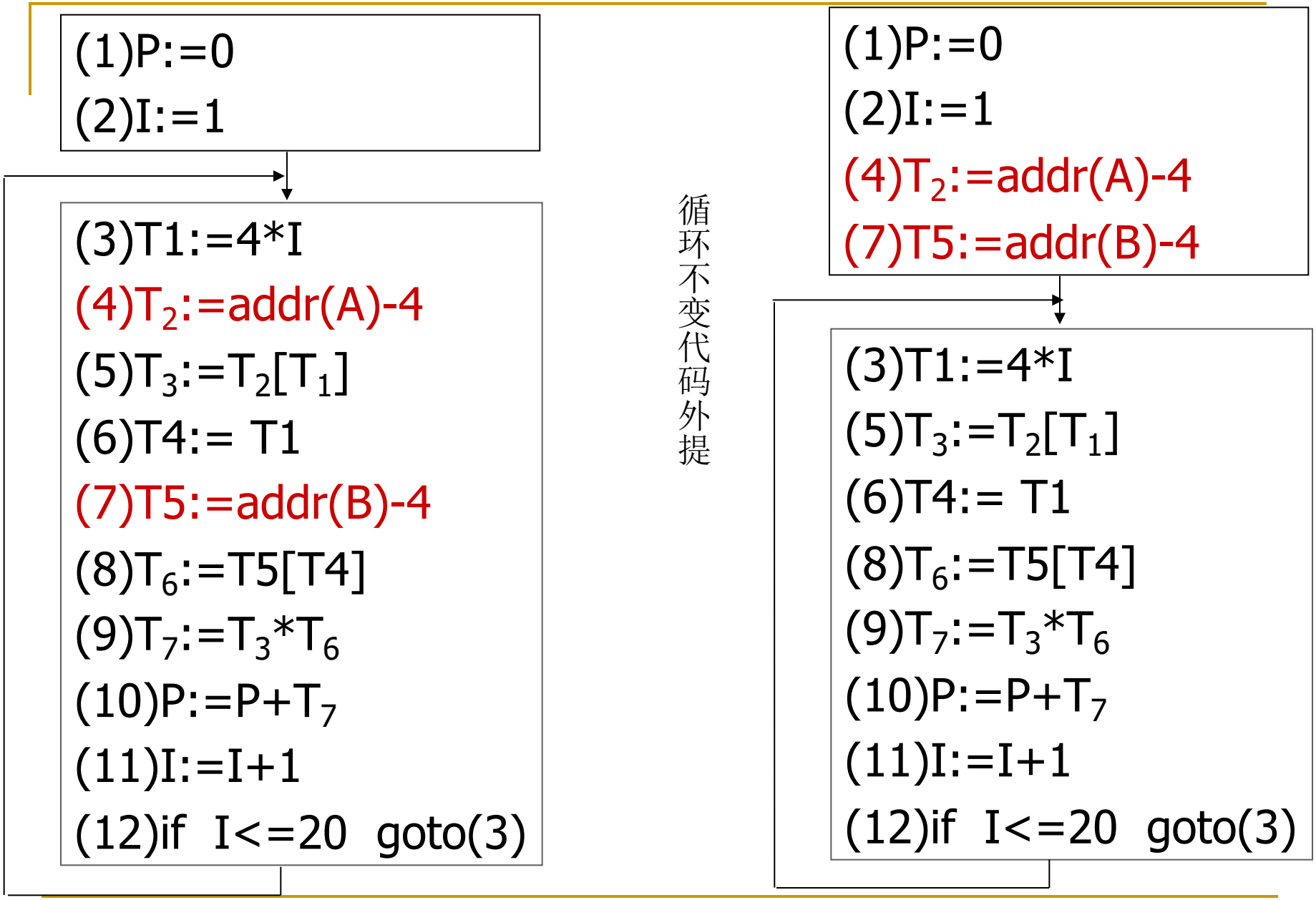
(1)P:=0
(2)I:=1

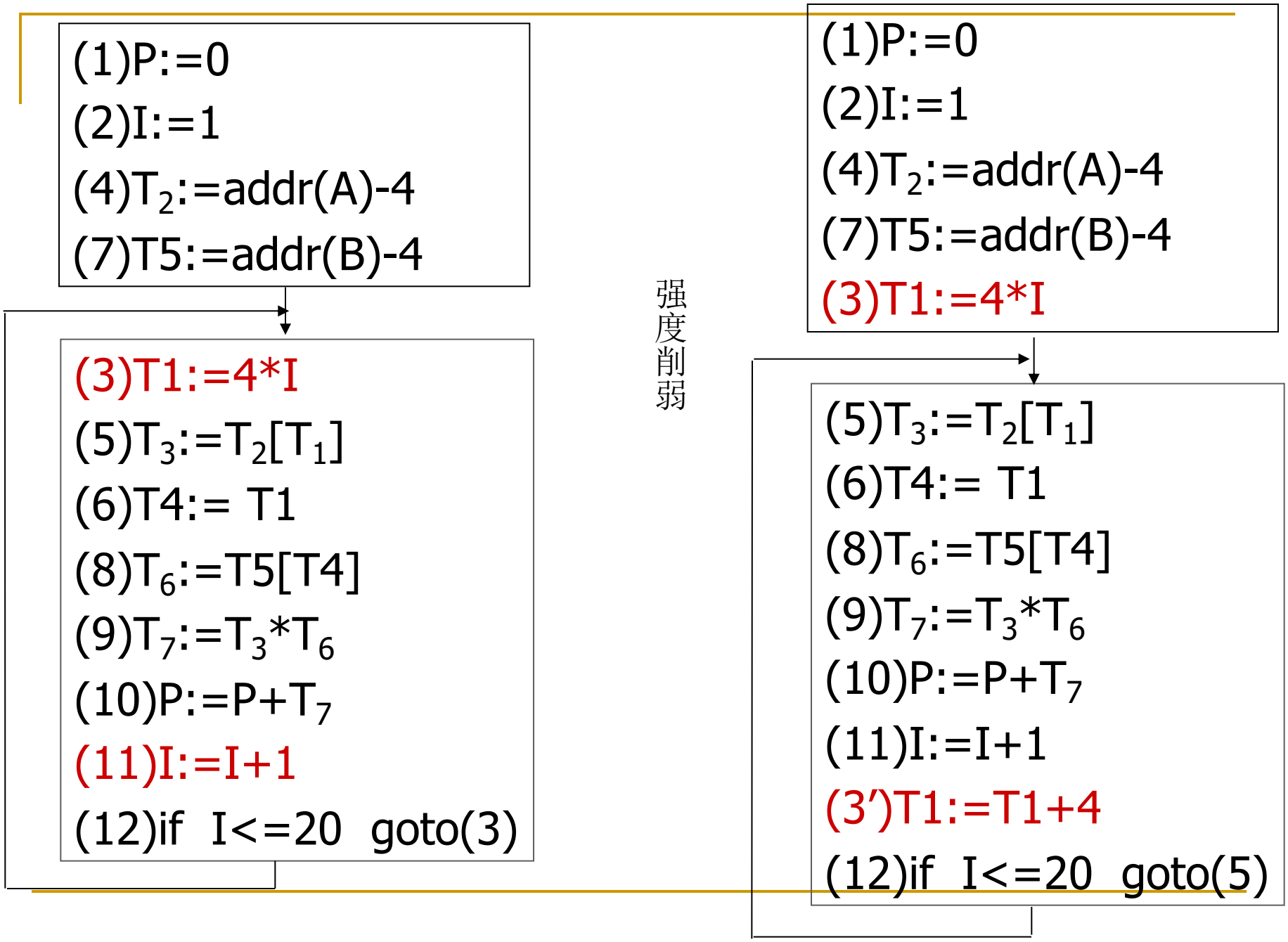
(3)T_1:=4*I
(4)T_2:=addr(A)-4
(5)T_3:=T_2[T_1]
(6)T4:=4*I
(7)T_5:=addr(B)-4
(8)T_6:=T5[T4]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(11)I:=I+1
(12)if I<=20 goto(3)

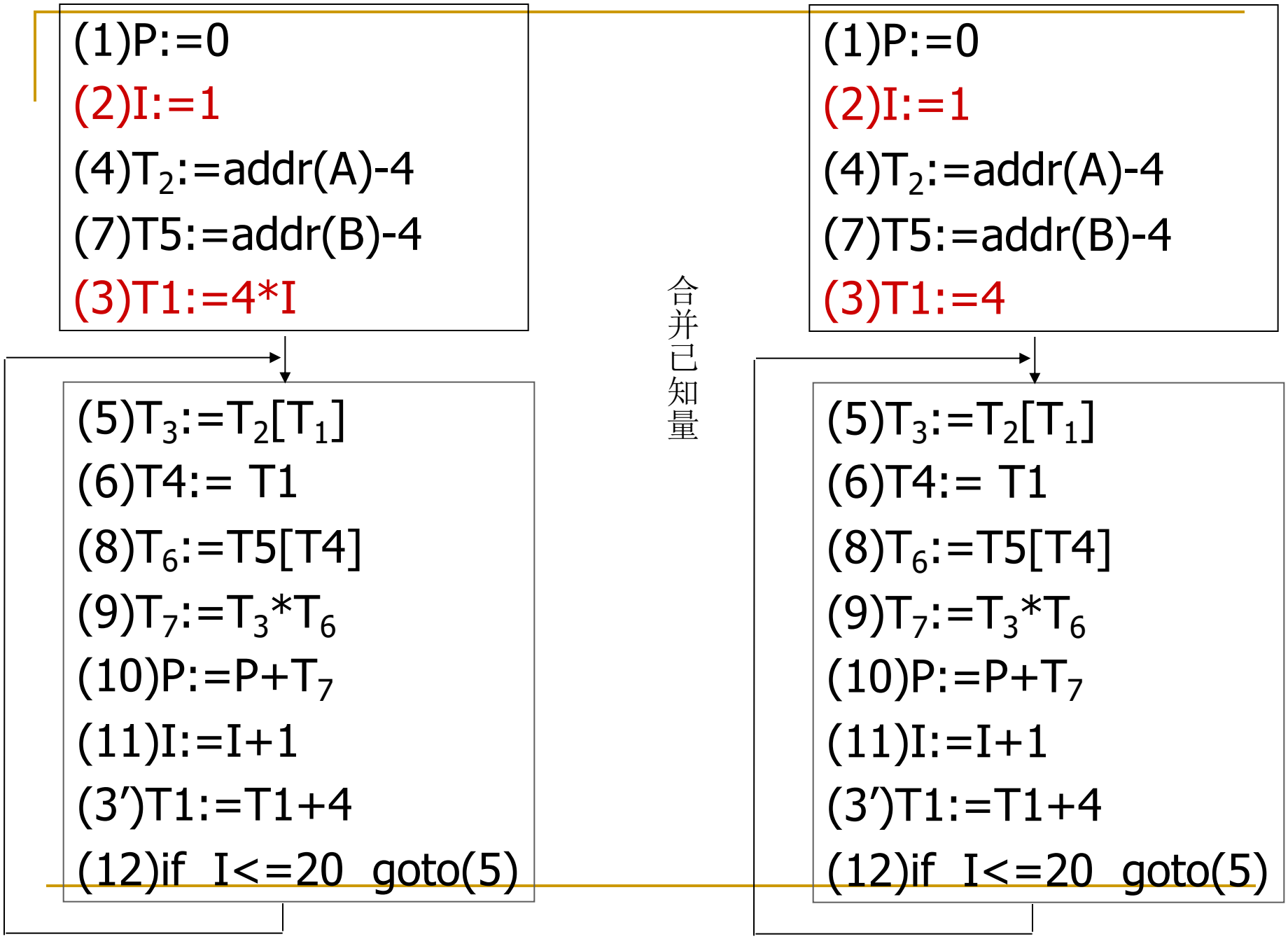
删除多余运算(公共子表达式)

(1)P:=0
(2)I:=1

(3)T_1:=4*I
(4)T_2:=addr(A)-4
(5)T_3:=T_2[T_1]
(6)T4:= T_1
(7)T_5:=addr(B)-4
(8)T_6:=T5[T4]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(11)I:=I+1
(12)if I<=20 goto(3)


(1)P:=0
(2)I:=1
(3)T1:=4*I
(4)T2:=addr(A)-4
(5)T3:=T2[T1]
(6)T4:= T1
(7)T5:=addr(B)-4
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(12)if I<=20 goto(3)
循环不变代码外提
(1)P:=0
(2)I:=1
(4)T2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4*I
(5)T3:=T2[T1]
(6)T4:= T1
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(12)if I<=20 goto(3)


(1)P:=0
(2)I:=1
(4)T2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4*I
(5)T3:=T2[T1]
(6)T4:= T1
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(12)if I<=20 goto(3)
强度削弱
(1)P:=0
(2)I:=1
(4)T2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4*I
(5)T3:=T2[T1]
(6)T4:= T1
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(3′)T1:=T1+4
(12)if I<=20 goto(5)


(1)P:=0
(2)I:=1
(4)T2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4*I
(5)T3:=T2[T1]
(6)T4:= T1
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(3′)T1:=T1+4
(12)if I<=20 goto(5)
合并已知量
(1)P:=0
(2)I:=1
(4)T2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4
(5)T3:=T2[T1]
(6)T4:= T1
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(3′)T1:=T1+4
(12)if I<=20 goto(5)

(1)P:=0
(2)I:=1
(4)T_2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4

(5)T_3:=T_2[T_1]
(6)T4:= T1
(8)T6:=T5[T4]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(11)I:=I+1
(3′)T1:=T1+4
(12)if I<=20 goto(5)

复写传播

(1)P:=0
(2)I:=1
(4)T_2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4

(5)T_3:=T_2[T_1]
(6)T4:= T1
(8)T6:=T5[T1]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(11)I:=I+1
(3′)T1:=T1+4
(12)if I<=20 goto(5)

(1)P:=0
(2)I:=1
(4)T_2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4

(5)T_3:=T_2[T_1]
(6)T4:= T1
(8)T6:=T5[T1]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(11)I:=I+1
(3′)T1:=T1+4
(12)if I<=20 goto(5)

变换循环控制条件

(1)P:=0
(2)I:=1
(4)T_2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4

(5)T_3:=T_2[T_1]
(6)T4:= T1
(8)T6:=T5[T1]
(9)T_7:=T_3*T_6
(10)P:=P+T_7
(3′)T1:=T1+4
(12)if T1<=80 goto(5)

(1)P:=0

(2)I:=1

(4)T_2:=addr(A)-4

(7)T5:=addr(B)-4

(3)T1:=4

(5)T_3:=T_2[T_1]

(6)T4:= T1

(8)T6:=T5[T1]

(9)T_7:=T_3*T_6

(10)P:=P+T_7

(3′)T1:=T1+4

(12)if T1<=80 goto(5)

删除无用赋值

(1)P:=0

(4)T_2:=addr(A)-4

(7)T5:=addr(B)-4

(3)T1:=4

(5)T_3:=T_2[T_1]

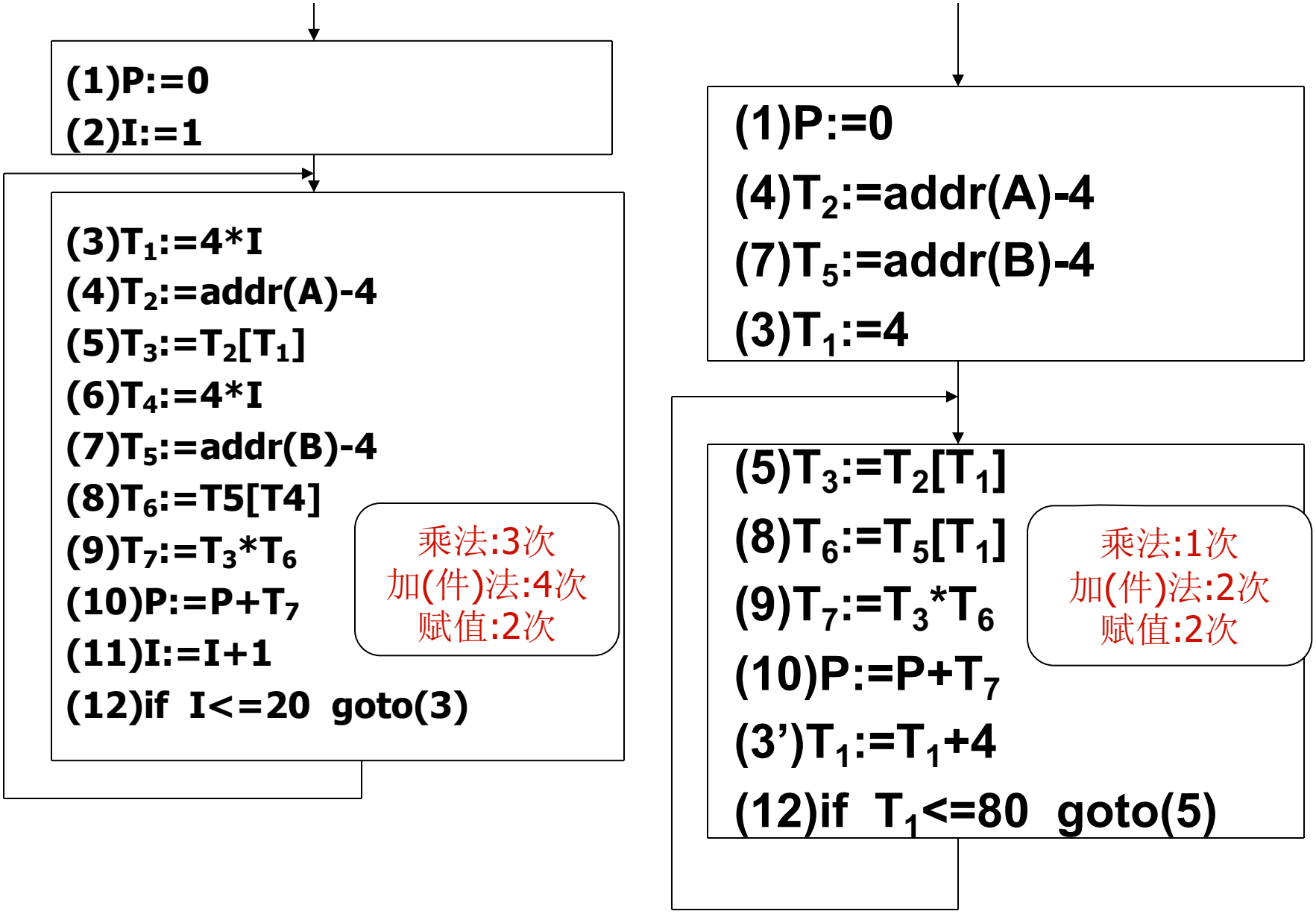
(8)T6:=T5[T1]

(9)T_7:=T_3*T_6

(10)P:=P+T_7

(3′)T1:=T1+4

(12)if T1<=80 goto(5)


(1)P:=0
(2)I:=1
(3)T1:=4*I
(4)T2:=addr(A)-4
(5)T3:=T2[T1]
(6)T4:=4*I
(7)T5:=addr(B)-4
(8)T6:=T5[T4]
(9)T7:=T3*T6
(10)P:=P+T7
(11)I:=I+1
(12)if I<=20 goto(3)
乘法:3次
加(件)法:4次
赋值:2次
(1)P:=0
(4)T2:=addr(A)-4
(7)T5:=addr(B)-4
(3)T1:=4
(5)T3:=T2[T1]
(8)T6:=T5[T1]
(9)T7:=T3*T6
(10)P:=P+T7
(3')T1:=T1+4
(12)if T1<=80 goto(5)
乘法:1次
加(件)法:2次
赋值:2次

优化技术

- 删除多余运算
- 循环不变代码外提
- 强度削弱
- 变换循环控制条件
- 合并已知量与复写传播
- 删除无用赋值

- 依优化范围划分
  - 窥孔优化
    局部的几条指令范围内的优化
  - 局部优化
    基本块范围内的优化
  - 超局部优化
  - 循环优化
  - 过程内全局优化
    流图范围内的优化
  - 过程间优化
    整个程序范围内的优化

窥孔优化

- 工作方式 在目标指令序列上滑动一个包含几条指令的窗口（称为窥孔），发现其中不够优化的指令序列，用一段更短或更有效的指令序列来替代它，使整个代码得到改进
- 举例
  - 删除冗余的“取”和“存”（*redundant loads and stores*）

    指令序列

    （1） MOV R0，a
    （2） MOV a，R0

    可优化为

    （1） MOV R0，a

窥孔优化

- 举例
  - 合并已知量（*constant folding*）

    代码序列

    （1） r2:=3*2

    可优化为

    （1） r2:=6

窥孔优化

– 举例

- 常量传播（*constants propagating*）

  代码序列

  （1） r2:=4
  （2） r3:=r1+r2

  可优化为

  （1） r2:=4
  （2） r3:=r1+ 4

注：虽然条数未少，但若是知道r2不再活跃时，可删除（1）

窥孔优化

- 举例
  - 代数化简（*algebraic simplification*）

    代码序列

    （1） x:=x+0
    （2） ……
    （n） y:=y*1

    中的（1），（n）可在窥孔优化时删除

窥孔优化

- 举例
  - 控制流优化（*flow-of-control optimization*）

    代码序列

```
            goto L1
            ……
    L1:     goto L2
```

    可替换为

```
            goto L2
            ……
    L1:     goto L2
```

窥孔优化

- 举例
  - 死代码删除（*dead-code elimination*）

    代码序列

    ```
    debug := false
    if (debug) print …
    ……
    ```

    可替换为

    ```
    debug := false
    ……
    ```

窥孔优化

- 举例
  - 强度削弱（*reduction in strength*）

    x:=2.0*f　可替换为　x:=f+f

    x:=f/ 2.0 可替换为　x:=f*0.5

窥孔优化

- 举例
  - 使用目标机惯用指令（*use of idioms*）

    某个操作数与*1* 相加，通常用“加*1*”指令，而不是用“加”指令

    某个定点数乘以*2* ，可以采用“左移”指令；而除以*2* ，则可以采用“右移”指令

    …

局部优化---基本块的DAG

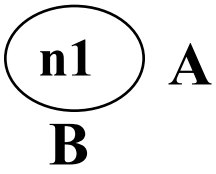
- 基本块的DAG表示和应用
  - 叶结点用标识符(变量名)或常数作为其唯一的标记，当叶结点是标识符时，代表名字的初值，给它加下标0;
  - 内部结点用运算符标记，它表示计算的值;
  - 各结点可能附加有一个或若干个标识符，附加于同一个结点上的若干个标识符有相同的值。

(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

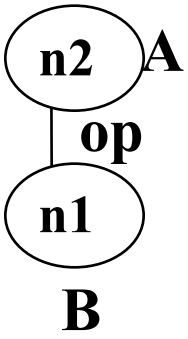
各种三地址码对应的DAG结点

DAG结点

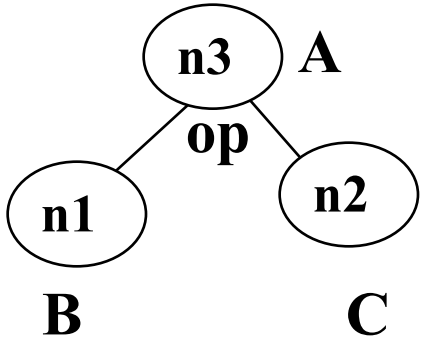
0型: A:=B



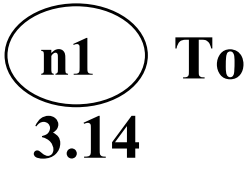
1型: A:=op B



2型: A:=B op C

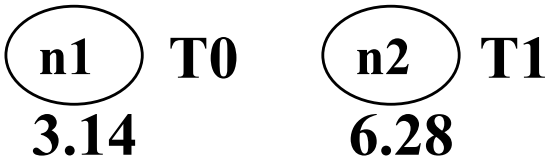


(1) T0:=3.14 (2) T1:=2*T0



(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r

常量合并

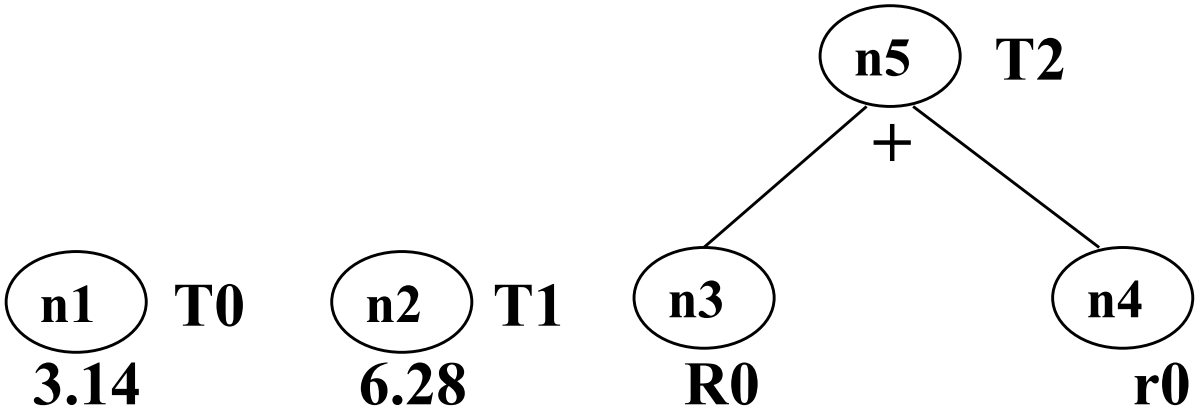


(1) T0:=3.14

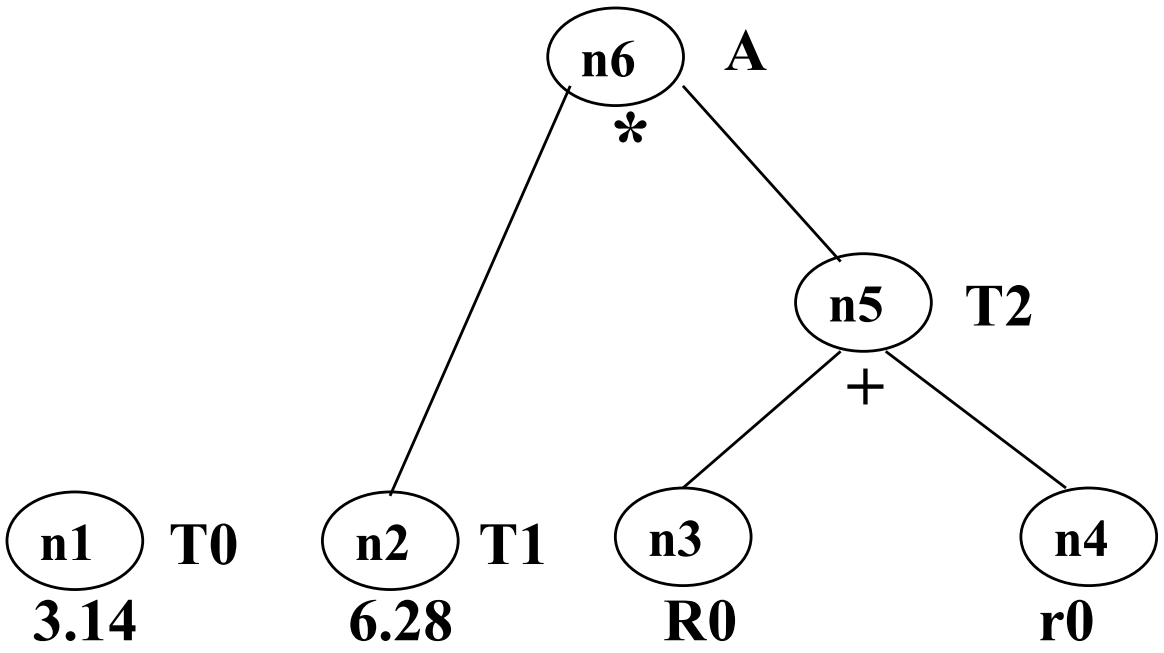
(2) T1:=2*T0

(3) T2:=R+r

(4) A:=T1*T2

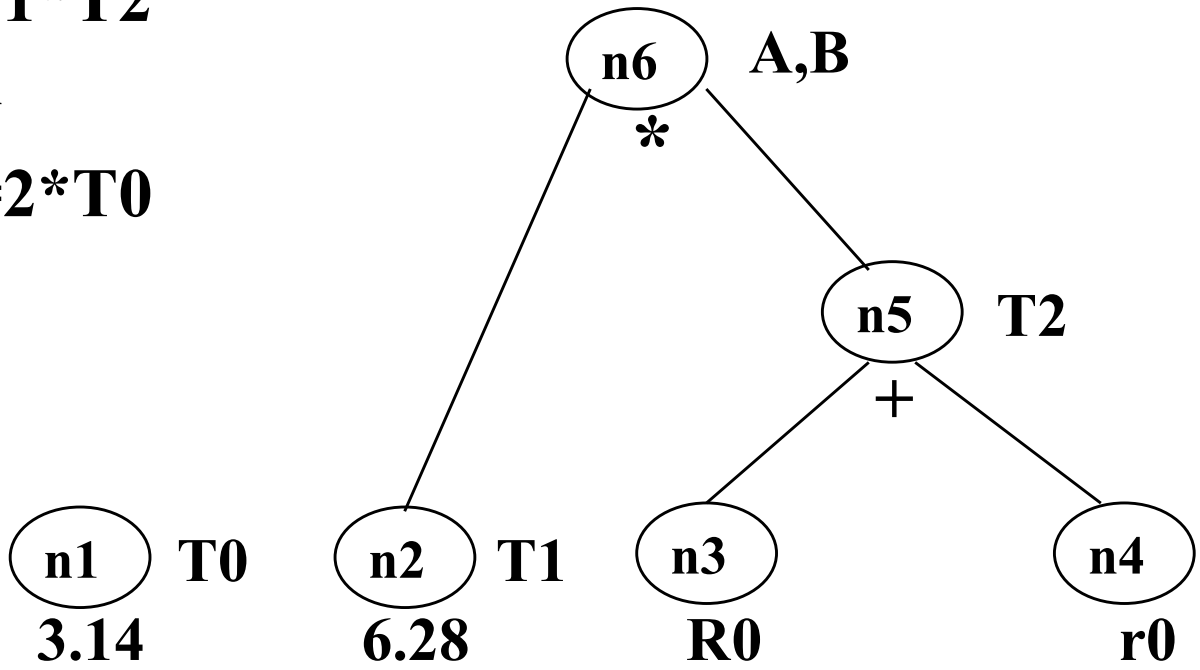


(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A

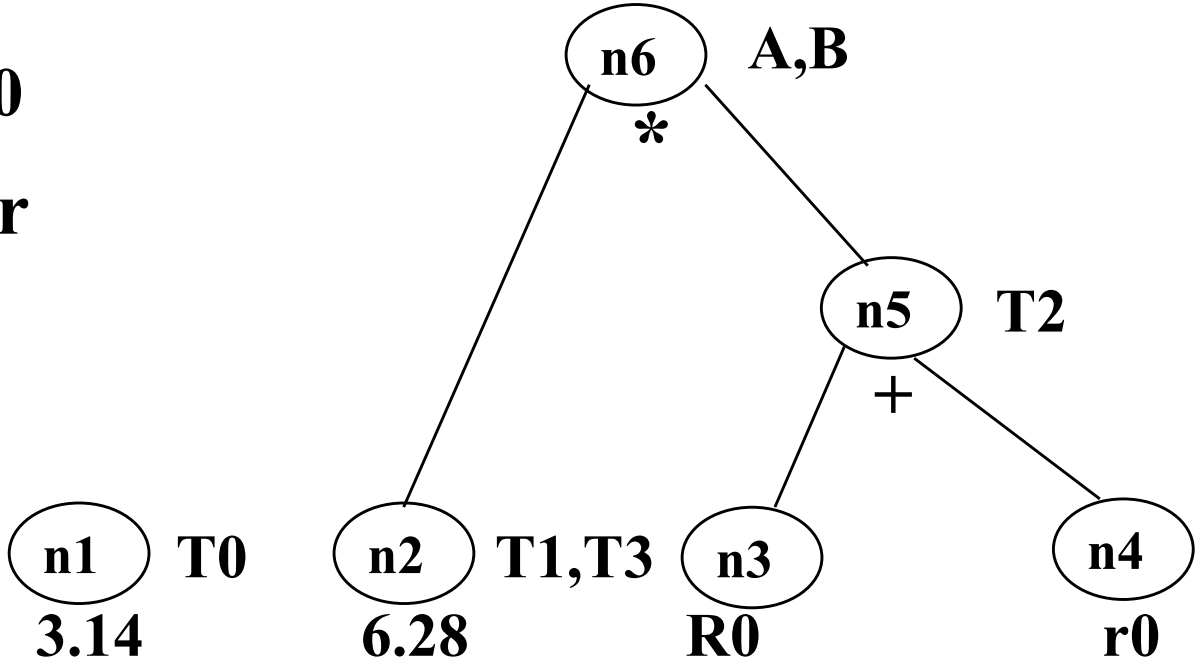


复写传播

(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A
(6) T3:=2*T0



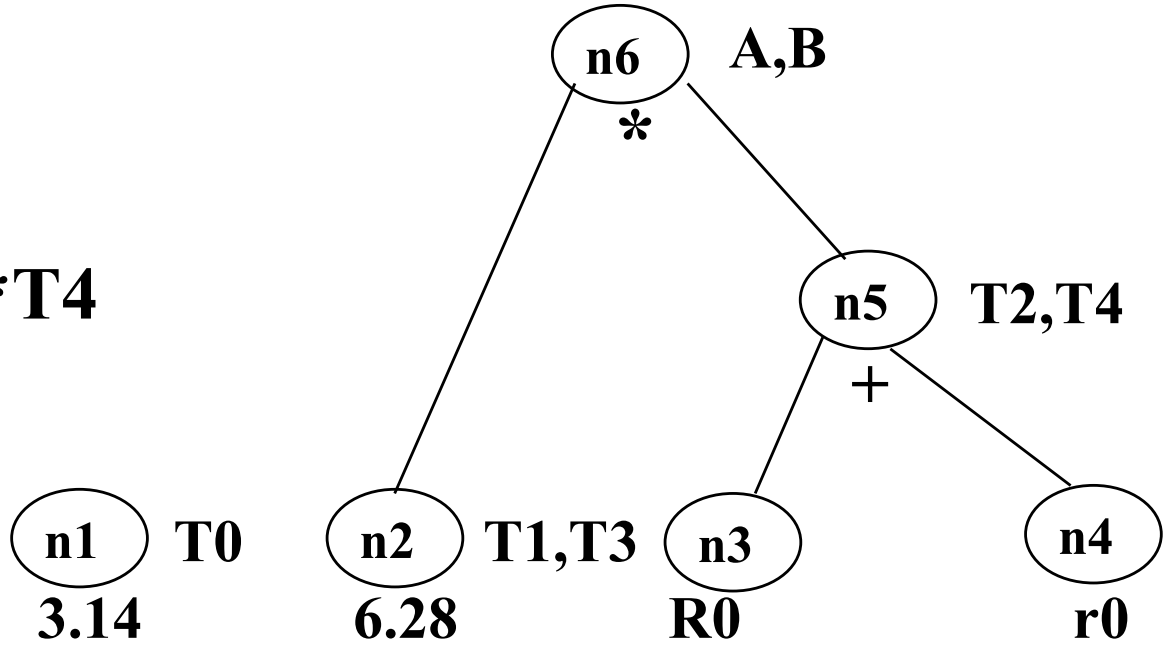
(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A
(6) T3:=2*T0
(7) T4:=R+r



删除公共子表达式

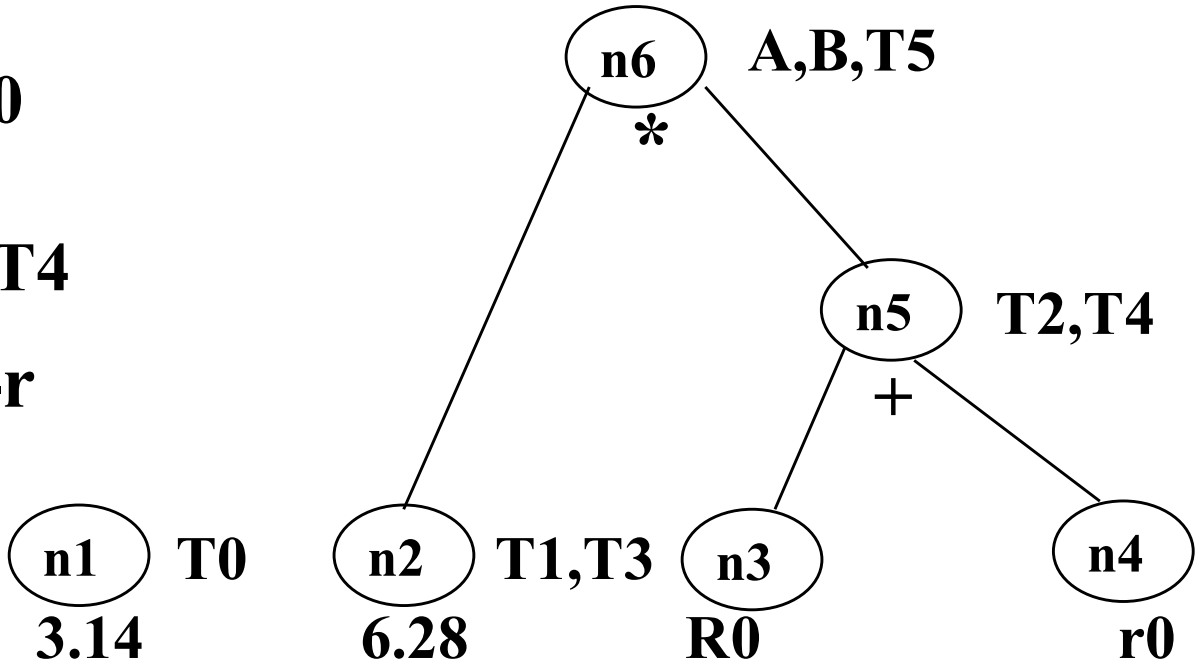
(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A
(6) T3:=2*T0
(7) T4:=R+r

(8) T5:=T3*T4



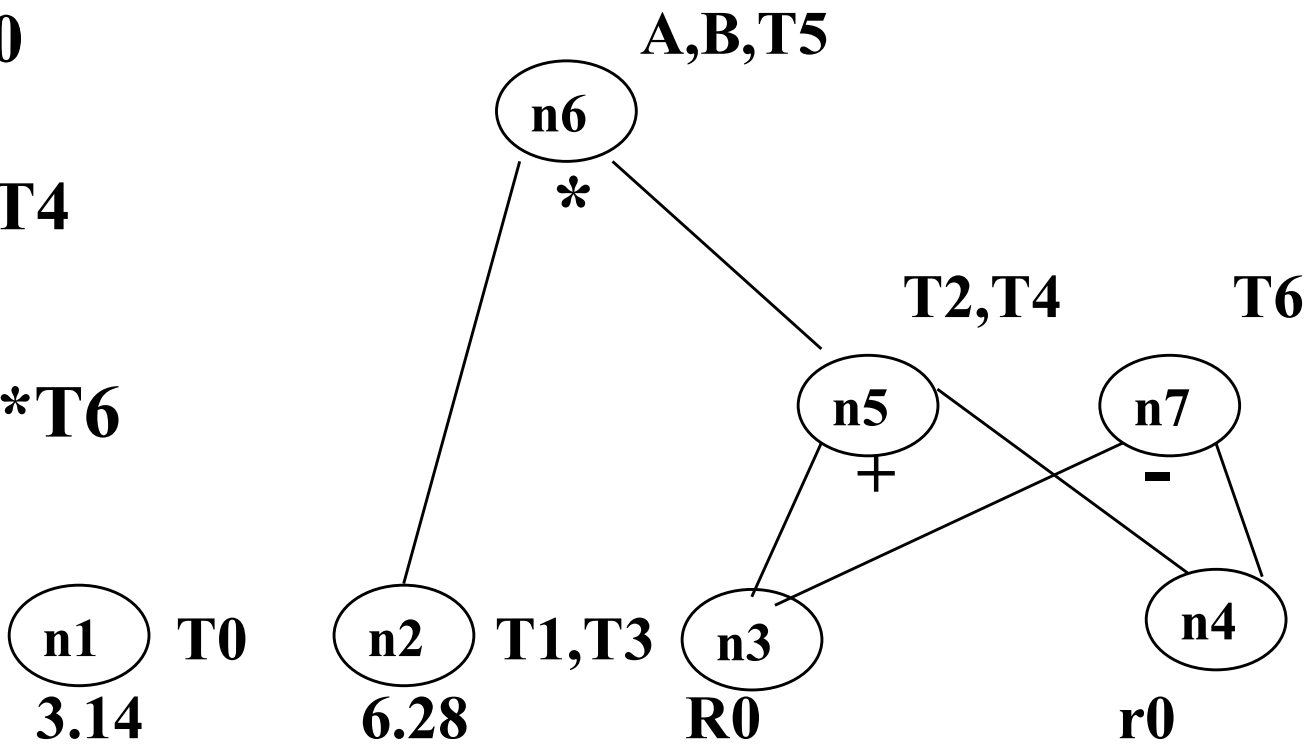
(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A
(6) T3:=2*T0
(7) T4:=R+r
(8) T5:=T3*T4

(9) T6:=R-r



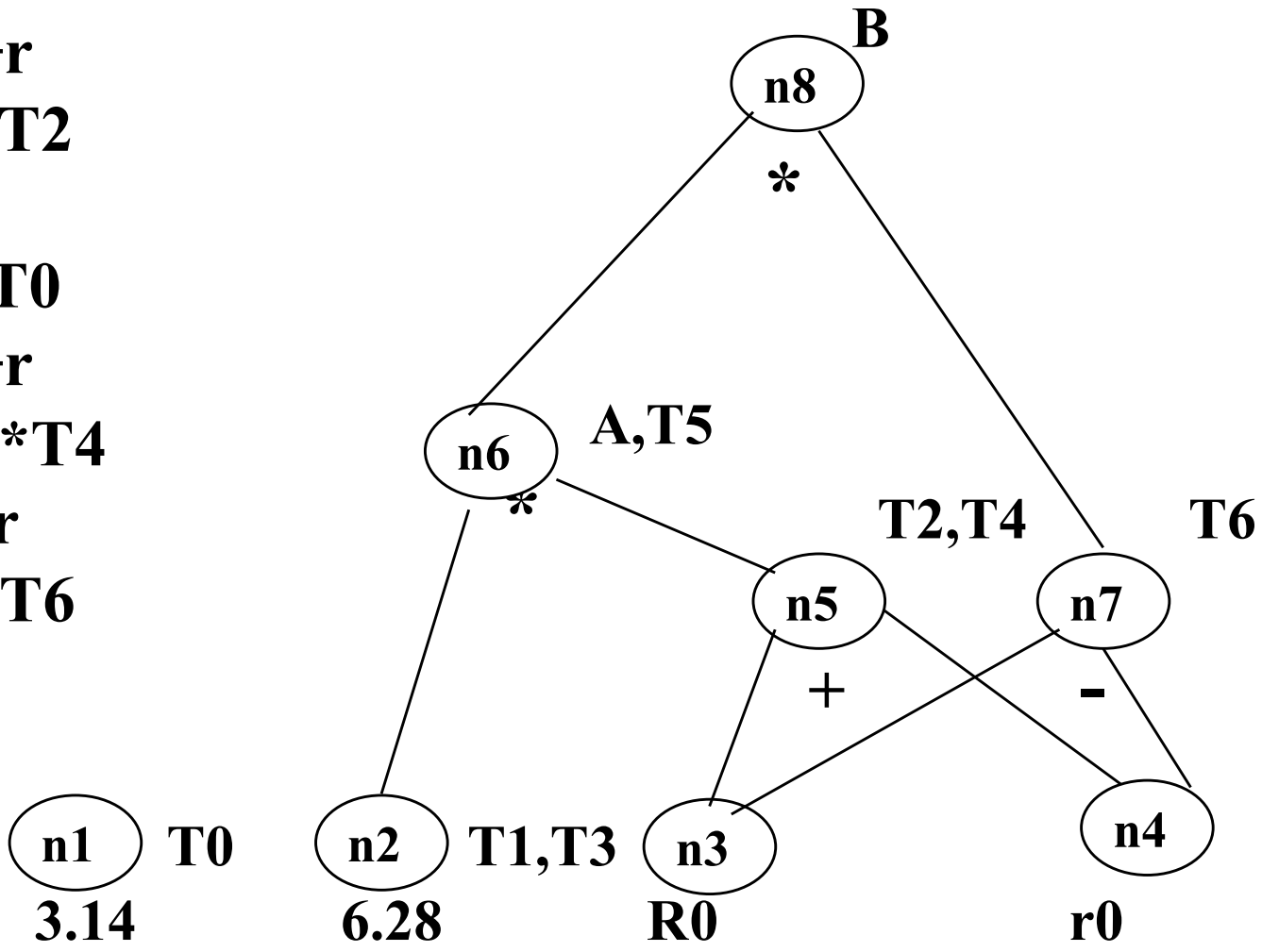
(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A
(6) T3:=2*T0
(7) T4:=R+r
(8) T5:=T3*T4
(9) T6:=R-r

(10) B:=T5*T6



(1) T0:=3.14
(2) T1:=2*T0
(3) T2:=R+r
(4) A:=T1*T2
(5) B:=A
(6) T3:=2*T0
(7) T4:=R+r
(8) T5:=T3*T4
(9) T6:=R-r
(10) B:=T5*T6

删除无用赋值



(1) T_0:=3.14

(2) T_1:=6.28 （合并已知量）

(3) T_3:=6.28 （合并已知量）

(4) T_2:=R+r

(5) T_4:=T_2 （删除公共子表达式）

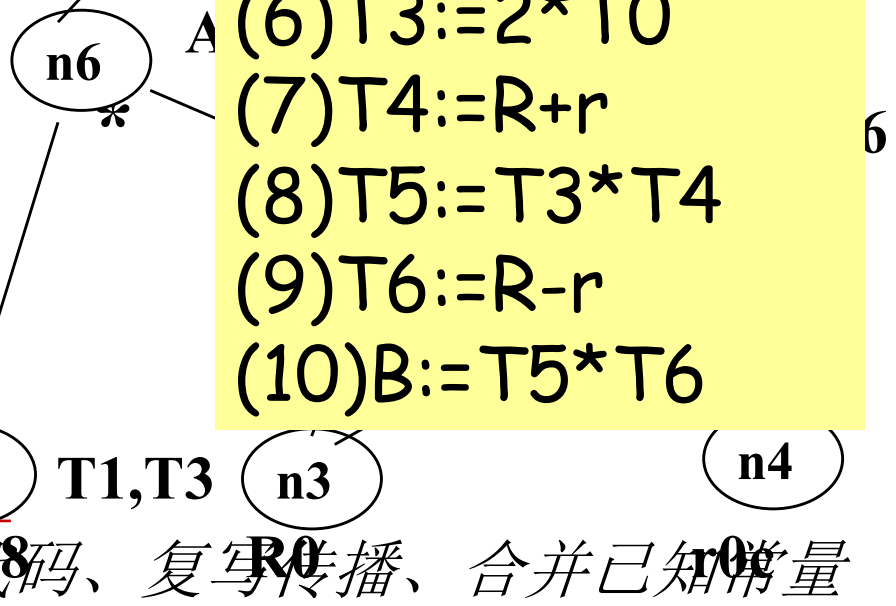
(6) A:=6.28*T_2

(7) T_5:=A （删除公共子表达式））

(8) T_6:=R-r

(9) B:=A*T_6 （删除无用代码）

(1)T0:=3.14
(2)T1:=2*T0
(3)T2:=R+r
(4)A:=T1*T2
(5)B:=A
(6)T3:=2*T0
(7)T4:=R+r
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6



基本块内可进行的优化有：

删除公共子表达式、删除无用代码、复写传播、合并已知常量

例题：对下列基本块B应用DAG进行优化，

B:
T1 := a+b
T2 := 3.14
S := T1*T2
T3 := a+b
T4 := 2.56
T5 := T2+T4
W := T5*T1
S := T5*T3

课堂练习

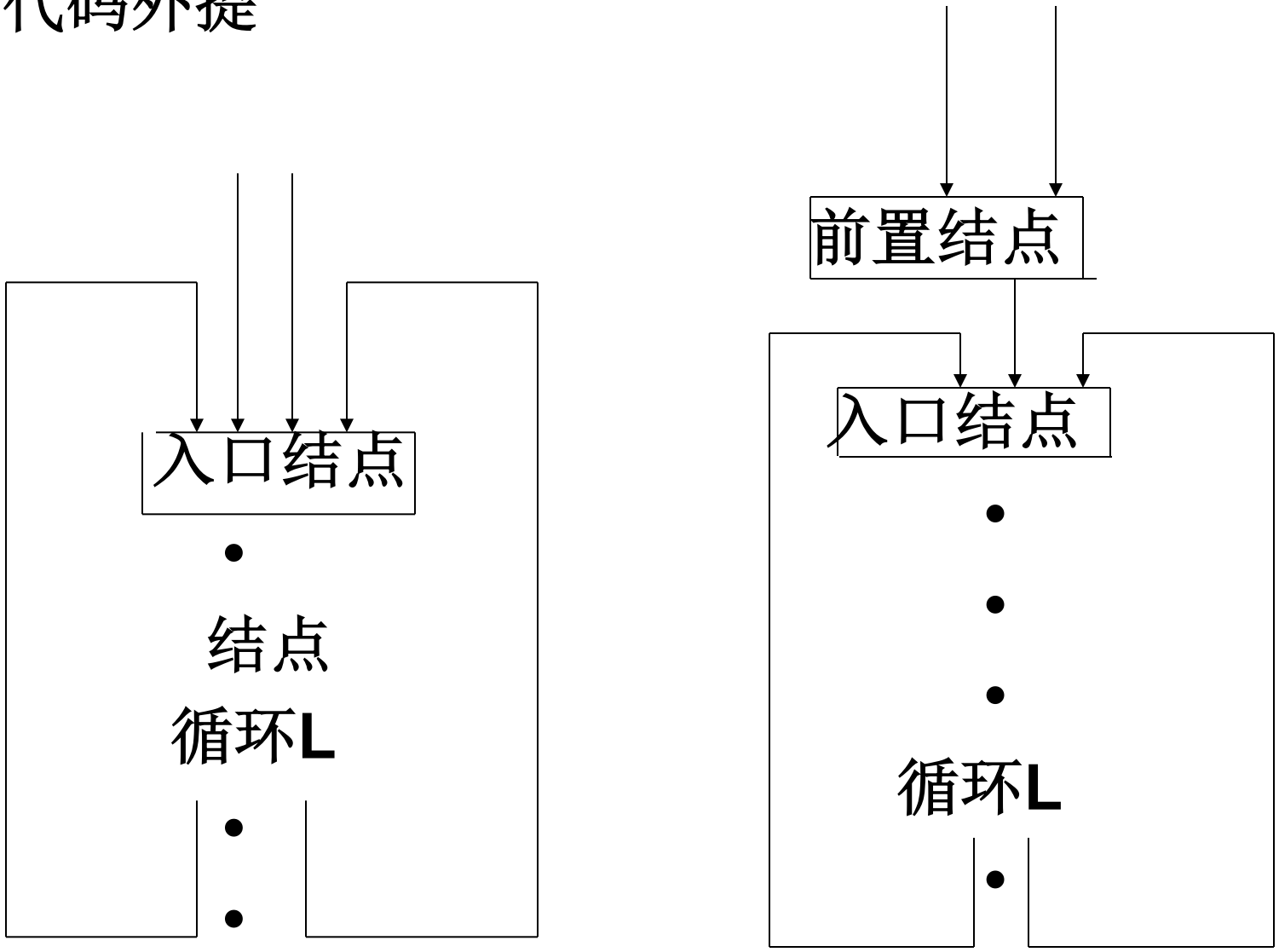
- **7+b*c-(c*b+9-5)/(b*c+d)**

- **（a*b+c）/(a*b-c)+(2*3+7-8)/(a*b+c)**

循环优化

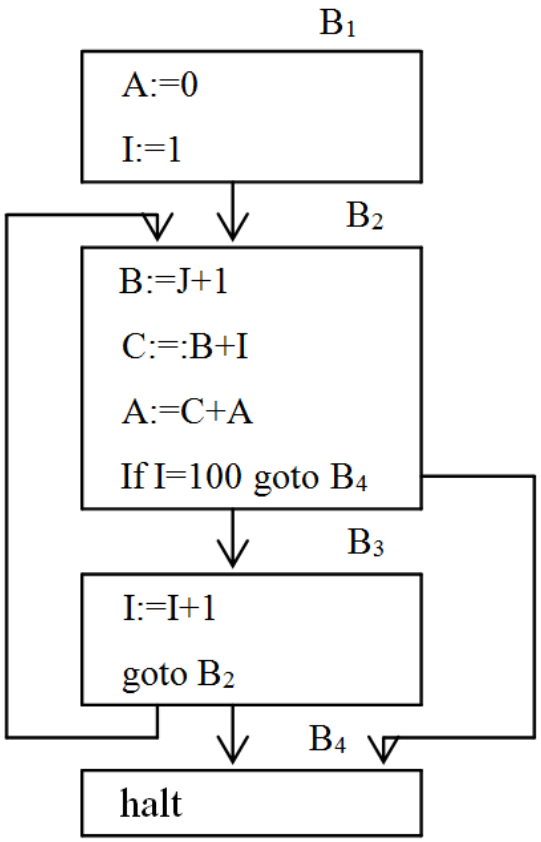
- 代码外提
- 归纳变量的删除

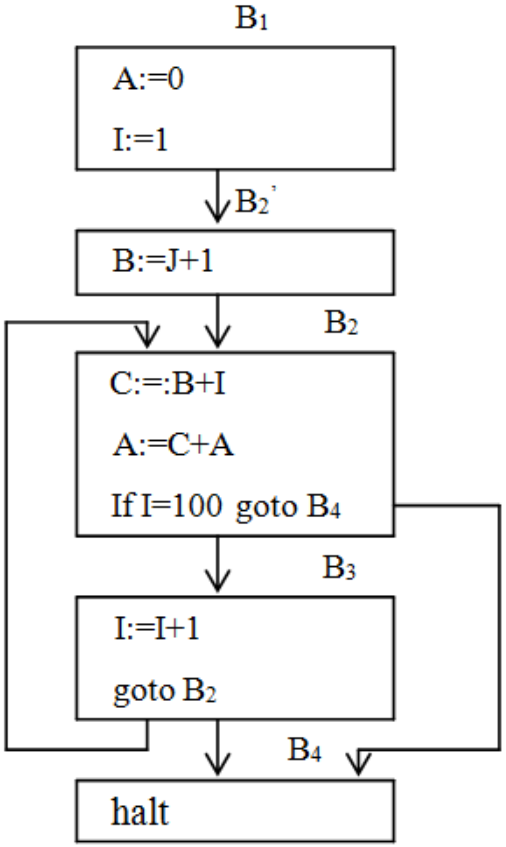
代码外提



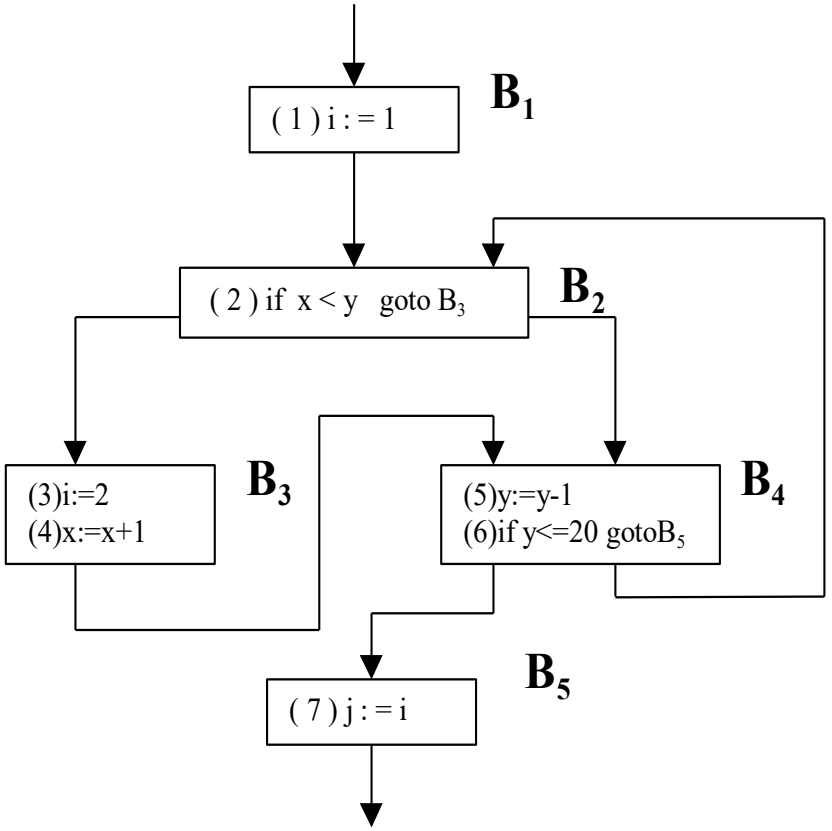
循环不变式外提

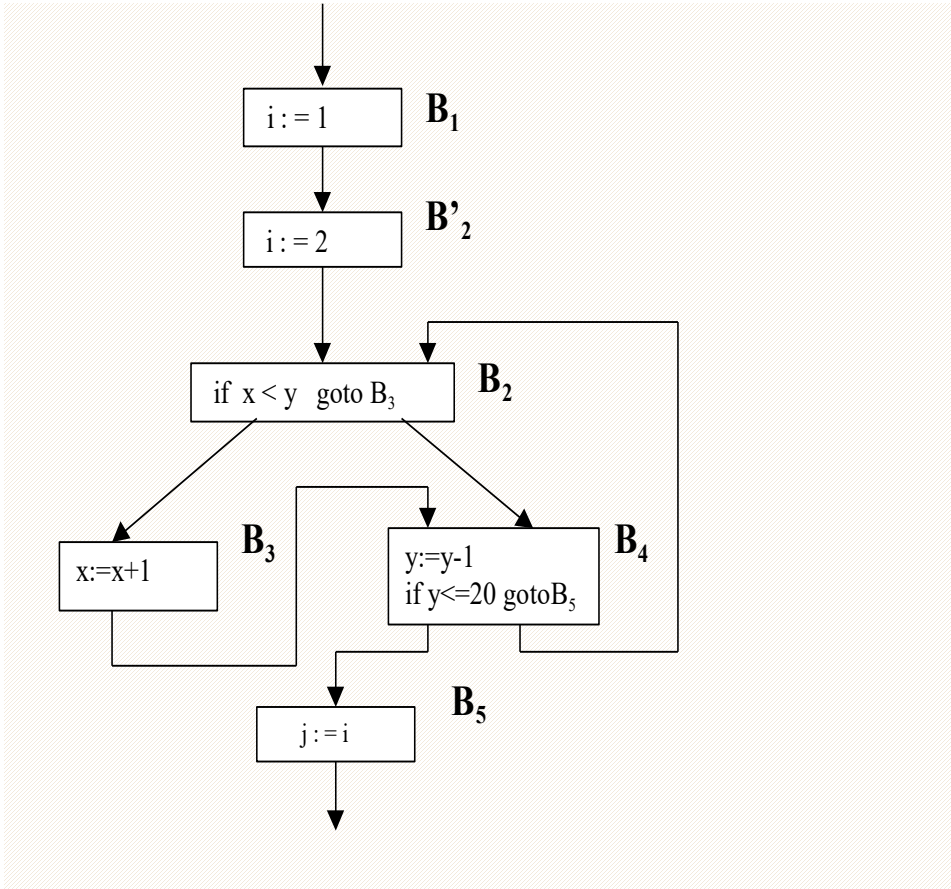
- 把循环不变运算，即产生的结果独立于循环执行次数的表达式，放到循环的前面。





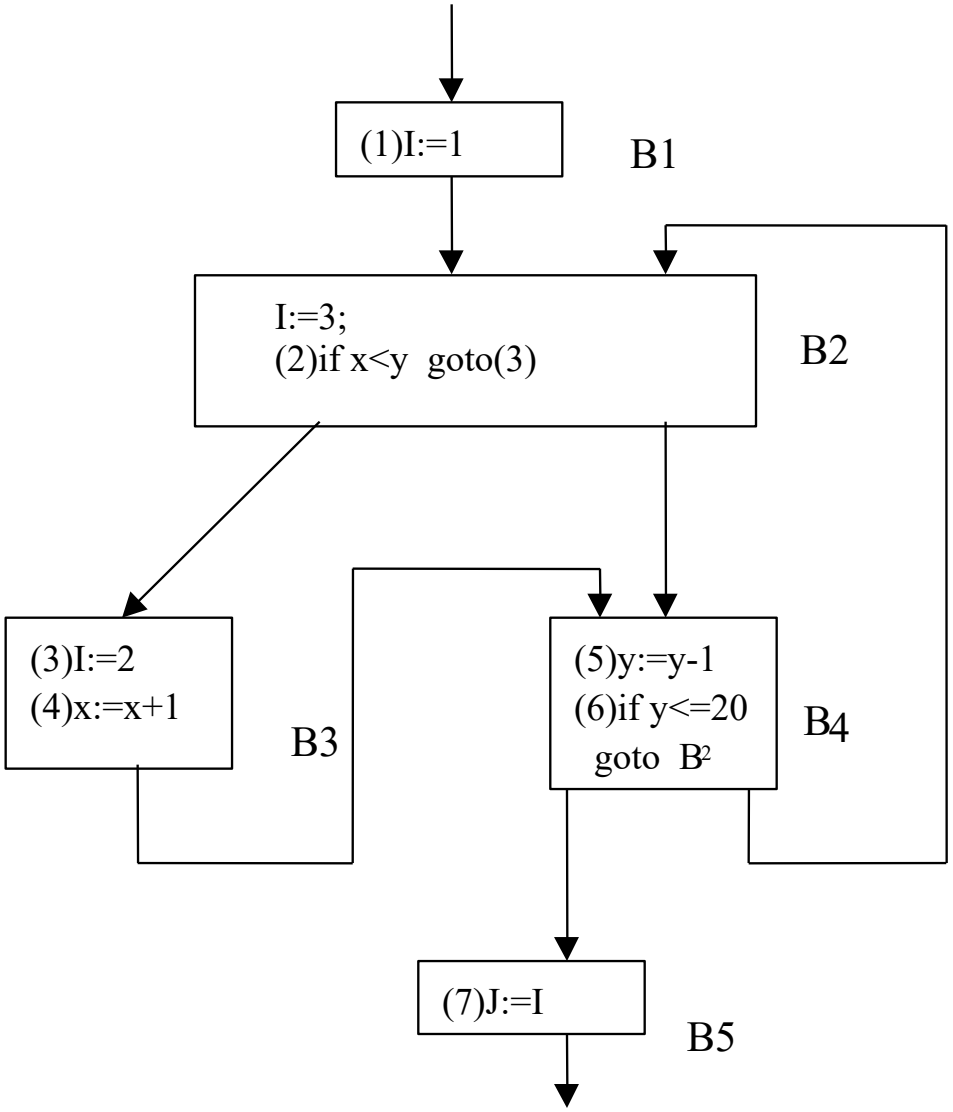
- 是否一个循环中的全部不变运算均可外提?





当把一循环中的不变运算外提时，要求该不变运算所在的结点是循环所有出口结点的支配结点。

是否一个循环中的全部不变运算均可外提?



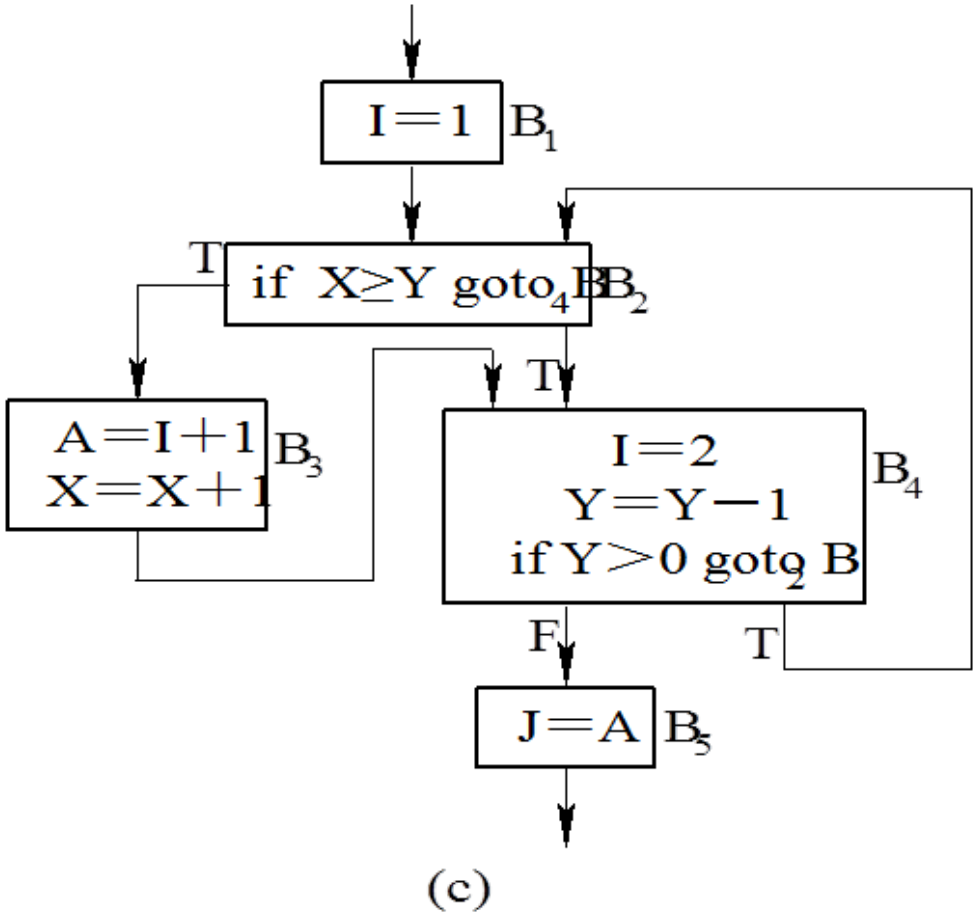
当把循环中的不变运算A=B op C外提时，要求循环中其它地方不再有A的定值点。

“点”指某一四元式的位置。

对变量的“定值”指对变量赋值或输入值。

定值点指变量在该点被赋值或输入值。

- 是否一个循环中的全部不变运算均可外提?



(c)

当把循环不变运算A=B op C外提时，要求循环中A的所有引用点都是而且仅仅是该定值所能到达的。

引用点则指在该点使用了该变量。

到达-定值点是指变量在某点定值后到达的一点，通路上没有其它该变量的定值。

循环不变运算外提时的注意点

循环不变量代码 x:=y+z 可以外提的一个充分条件：

1）所在结点是循环的所有出口结点的支配结点

2）循环中其它地方不再有 x 的定值点

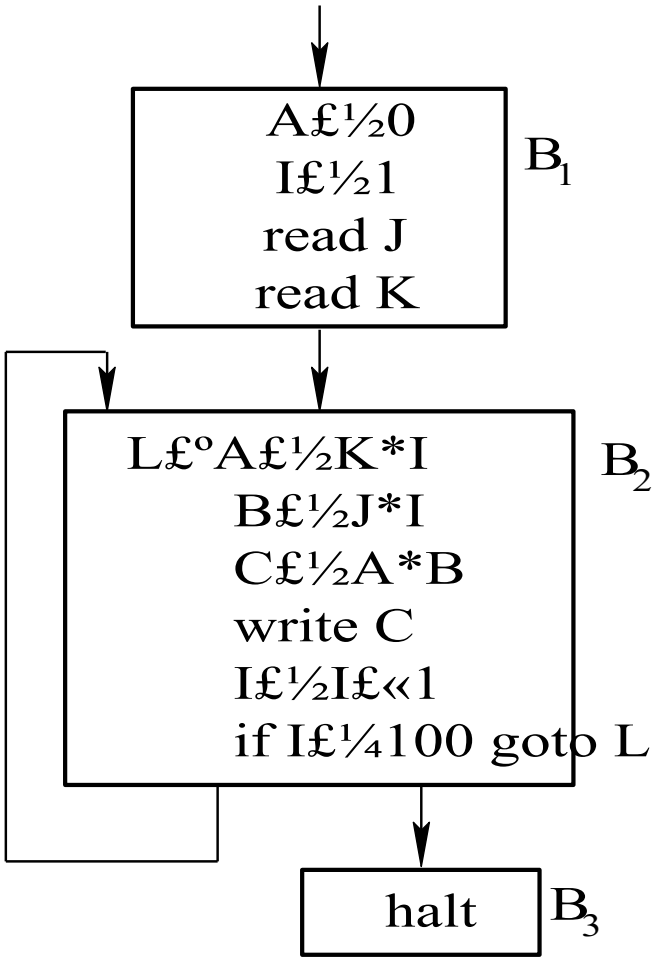
3）循环中 x 的所有引用点都是且仅是这个定值所能达到的

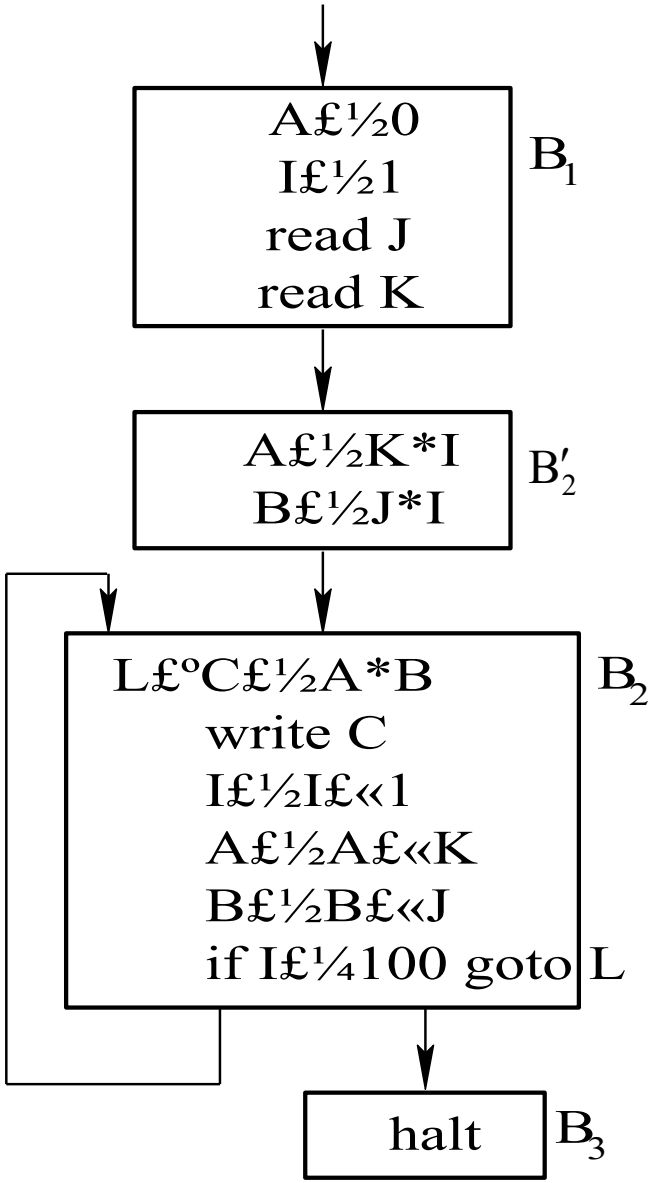
4）若y或z是在循环中定值的，则只有当这些定值点的语句（一定也是循环不变量）已经被执行过代码外提

或者，在满足上述 第 2、3 和 4 条的前提下，将第1条替换为：

5）要求 x 在离开循环之后不再是活跃的

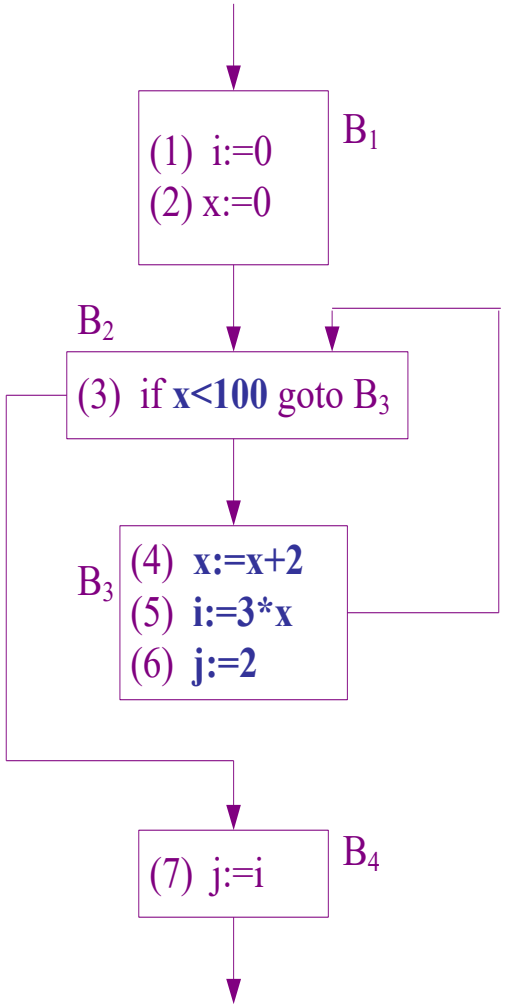
- 计算强度削减
  - 把强度大的运算换算成强度小的运算。强度削弱不仅可对乘法运算实行（将循环中的乘法运算用递归加法运算来替换），对加法运算也可实行。
  - 如果循环中有I的递归赋值**I=I±C**（**C**为循环不变量），并且循环中**T**的赋值运算可化归为**T=K*I±C1**（**K**和**C1**为循环不变量），那么**T**的赋值运算可以进行强度削弱。

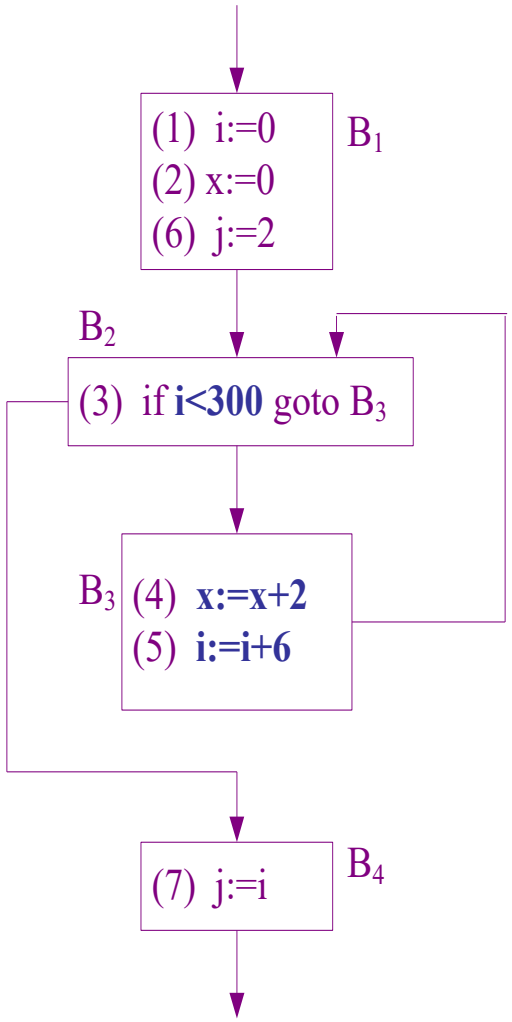

A£½0
I£½1
read J
read K
B1
L£ºA£½K*I
B£½J*I
C£½A*B
write C
I£½I£«1
if I£¼100 goto L
B2
halt
B3

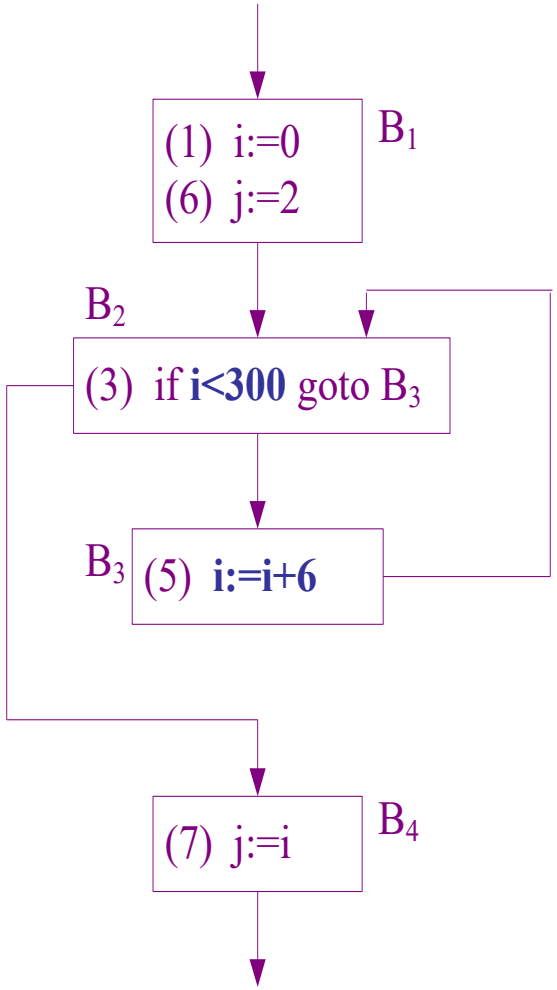

A£½0
I£½1
read J
read K
B1
A£½K*I
B£½J*I
B′2
L£ºC£½A*B
write C
I£½I£«1
A£½A£«K
B£½B£«J
if I£¼100 goto L
B2
halt
B3

归纳变量的删除

- 如果循环中对变量I只有惟一的形如I=I±C的赋值，且其中C为循环不变量，则称I为循环中的基本归纳变量。
- 如果I是循环中一基本归纳变量，J在循环中的定值总是可化归为I的同一线性函数，也即J=C1*I±C2，其中C1和C2都是循环不变量，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归纳变量。
- 一个基本归纳变量往往只在循环中用来计算其它归纳变量以及控制循环的进行。此时，可以用同族的某一归纳变量来替换循环控制条件中的这个基本归纳变量，从而达到将这个基本归纳变量从流图中删去的目的。这种优化称为删除归纳变量或变换循环控制条件。


(1) i:=0
(2) x:=0
B1
B2
(3) if x<100 goto B3
B3
(4) x:=x+2
(5) i:=3*x
(6) j:=2
(7) j:=i
B4


(1) i:=0
(2) x:=0
(6) j:=2
B1
B2
(3) if i<300 goto B3
B3
(4) x:=x+2
(5) i:=i+6
(7) j:=i
B4


(1) i:=0
(6) j:=2
B1
B2
(3) if i<300 goto B3
B3
(5) i:=i+6
(7) j:=i
B4

全局优化

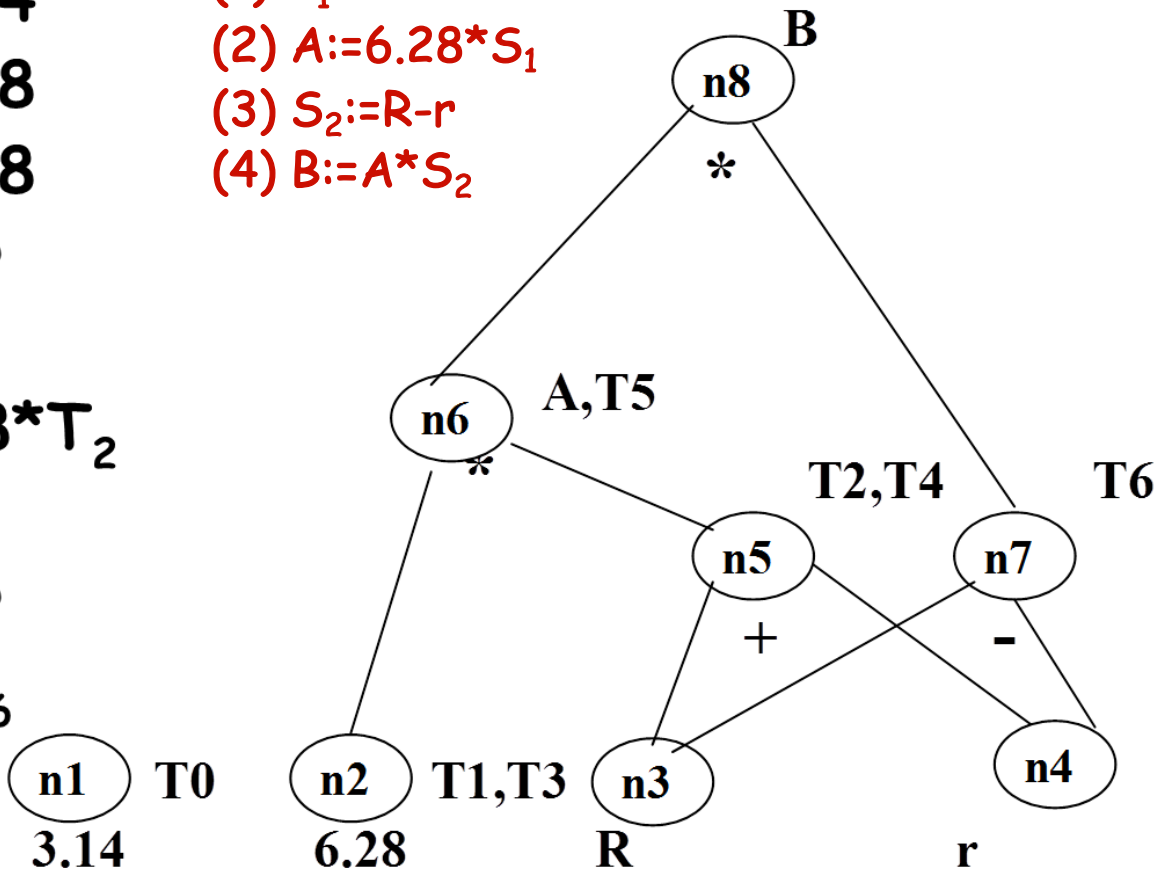
- 全局优化是在整个程序范围内进行的优化，需要把程序作为一个整体来收集信息 并把这些分配给流图中的各个基本块。因此，要进行数据流分析工作。

活跃变量

- 对程序中的某变量A和某点p而言，如果存在一条从p开始的通路，其中引用了A在点P的值，则称A在点p是活跃的。否则称A在点p是死亡的。
  - 无论是基本块优化或是循环优化，都可能引起某些变量的定值在该基本块或循环内不会被引用；只要这些变量在基本块或循环的出口之后也不是活跃的，那么，这些变量在该基本块或循环内的定值就是无用赋值，从而可以予以删除。因此，活跃变量的分析对于删除无用赋值是很有意义的。

(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

(1) S_1:=R+r
(2) A:=6.28*S_1
(3) S_2:=R-r
(4) B:=A*S_2



如果只有A和B在基本块后是活跃的，如何将DAG重写为四元式序列？

练习

```
        F:=H*G
        L:=F
        M:=L
B2:     B:=3
        D:=A+C
        E:=A*C
        F:=D+E
        G:=B*F
        H:=A+C
        I:=A*C
        J:=H+I
        K:=B*5
        L:=K+J
        M:=L
```

假设G、L、M在基本块后面要被引用