编译原理

Compiler Construction Principles

朱 青

信息学院计算机系，

中国人民大学，

zqruc2012@aliyun.com

第9章 优化

- 9.1 概述
- 9.2 局部优化
- 9.3 控制流分析和循环优化
- 9.4 数据流分析

9.1 概述

- 优化：
  - 目的：对程序等价变换，生成更有效的目标代码
  - 原则
    - 等价原则：程序运行结果不变
    - 有效原则：运行速度提高或存储空间减少
    - 合算原则：优化代价低
  - 分类：
    - 按阶段分：
      - 在源代码级由程序员选择更优的算法与数据结构
      - 与机器无关的优化---对中间代码进行：改进循环、地址计算等
      - 依赖于机器的优化--对目标代码进行：分配寄存器、优选指令
    - 根据优化所涉及的程序范围分成：
      - 局部优化:一个入口，一个出口的基本程序
      - 循环优化:对循环中的代码进行优化
      - 全局优化:整个程序范围的优化

概述

- 优化技术：
  - 删除公共子表达式：提高运行速度
  - 复写传播
  - 删除无用代码
  - 代码外提
  - 强度削弱
  - 删除归纳变量

代码优化器的结构:

代码优化器包括三部分:控制流分析,数据流分析和代码变换.

这里讨论以控制流分析为主的基本优化方法.以下列程序段为例:(p273)

```
 i=m-1;  j=n;      v=a[n];
while   (1)  {
    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[j];   a[j]=x;
```

各语句翻译出的中间代码写成如下形式:

(1) i:=m-1
(2) j:=n
(3) t1:=4*n
(4) v:=a[t1]
(5) i:=i+1
(6) t2:=4*i
(7) t3:=a[t2]
(8) if t3 < v goto (5)
(9) j:=j-1
(10) t4:=4*j
(11) t5:=a[t4]
(12) if t5 > v goto (9)
(13) if i >=j goto (23)
(14)t6:=4*i
(15)x:=a[t6]
(16)t7:=4*i
(17)t8:=4*j
(18)t9:=a[t8]
(19)a[t7]:=t9
(20)t10:=4*j

(21) a[t10]:=x

(22) goto (5)

---

(23) t11:=4*i

(24) x:=a[t11]

(25) t12:=4*i

(26) t13:=4*n

(27) t14:=a[t13]

(28) a[t12]:=t14

(29) t15:=4*n

(30) a[t15]:=x

其中访问数组a[i],假定每个元素占4个相邻字节,每次访问其值的三地址码可以表示成:

t1:=4 * i ;

t2:= a[t1] ;

第9章 优化

- 9.1 概述
- 9.2 局部优化
- 9.3 控制流分析和循环优化
- 9.4 数据流分析

9.2 局部优化

基本块与控制流图

基本块及其划分:

定义: 一个基本块是一组相邻语句组成的序列,控制流从序列的开头进入,从序列的末尾退出,没有停止或分支的情况.

例如:(程序段)

(14)-------(22)构成一个基本块.

算法:划分基本块

输入: 三地址语句序列.

输出:基本块的集合,输入的三地址语句恰好在一个基本块里.

方法:

(1)按下列方法确定每个基本块的开头语句:

1)输入的第一个语句是一个基本块的开头语句;

2)由条件或无条件goto语句转向的目标语句是一个基本块的开头语句;

3) 跟在条件或无条件goto语句后面的语句是一个基本块的开头语句。

(2) 由每个开头语句确定的基本块,包括本开头语句及其后相邻的若干语句,直到出现另一开头语句或程序末尾为止,但不包括另一基本块的开头语句。

例如:(程序段)的基本块的划分.

B1={(1)---(4)}

B2={(5)----(8)}

B3={(9)----(12)}

B4={(13)}

B5={(14)----(22)}

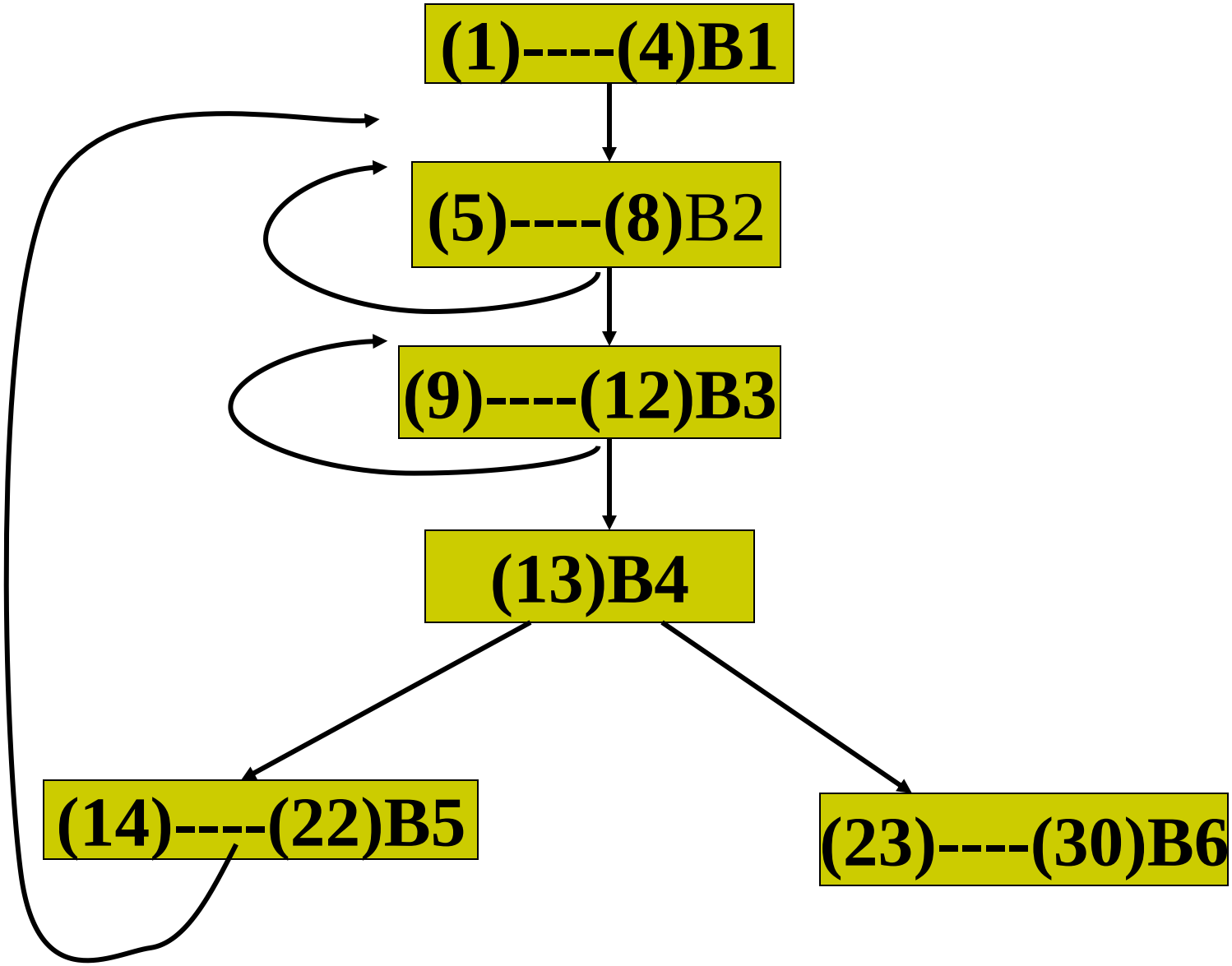
B6={(23)----(30)}

控制流图:

一个控制流图表示一个程序,在控制流图里,结点表示基本块,有向边表示控制流.

由此构成一个有向图.

例如:程序段的控制流图.(p257) 图10.2


(1)----(4)B1
(5)----(8)B2
(9)----(12)B3
(13)B4
(14)----(22)B5
(23)----(30)B6

修改前面的goto语句:

(8) if t3 < v goto B2

(12) if t5 > v goto B3

(13) if i>=j goto B6

(22) goto B2

在图中B2和B3分别自身构成循环,
B2,B3,B4和B5共同构成一个循环,其入口为B2.

基本块的优化:

对基本块进行等价的优化:

称基本块出口处活动名称的值,为基本块的值.

对基本块进行等价变换,要保证基本块的值不变.

♦删除冗余公共表达式

程序段中B5,在i,j不变的情况下,重复计算4*i和4*j:

(14)t6:=4*i

(15)x:=a[t6]

(16)t7:=4*i

(17)t8:=4*j

(18)t9:=a[t8]

(19)a[t7]:=t9

(20)t10:=4*j

(21) a[t10]:=x

(22) goto B2

定义:(冗余公共表达式)

在基本块中,除表达式E第一次出现之外,在E中的变量都没改变的情况下,其余的E都是冗余公共表达式.

解: 删除B5中的(16),(20),并且用t6代替t7 , t8代替t10得到:

(14)t6:=4*i
(15)x:=a[t6]
(17)t8:=4*j
(18)t9:=a[t8]
(19)a[t6]:=t9
(21) a[t8]:=x
(22) goto B2

例如: 设有如下基本块:

(1) a:=b+c
(2) b:=a-d
(3) c:=b+c
(4) d:=a-d

优化为:

(1) a:=b+c
(2) b:=a-d
(3) c:=b+c
(4) d:=b

其中的(3)不是冗余公共表达式,故不能删除.

♦复写传播 （p275）

♦删除无用代码:

定义:(无用代码)

如果一个变量x求值之后再不引用它的值,则称对x求值的代码为无用代码.

删除无用代码不改变基本块的值.

♦削弱计算强度:

是对基本块的代数变换.将复杂运算变换成简单运算.

例如:　x:=y**2 可以替换为:　　x:=y*y

特别删除: x:=x+0;　　x:=x*1;

♦删除归纳变量（p277）

♦临时变量改名:

假定在一个基本块里有语句

t :=b+c ;

其中t是一个临时变量名.把这个语句改成:

u:=b+c;

其中u是一个新的临时变量名.并替换其它t,则不改变基本块的值.

♦交换两个互不依赖的相邻语句的位置

假定在一个基本块里有两个相邻的语句

t1:=b+c

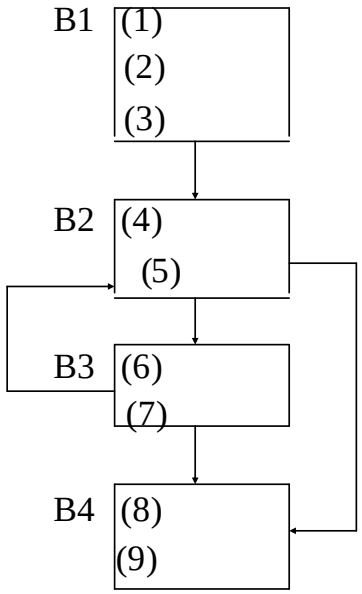
t2:=x+y

如果x,y均不为t1;b,c均不为t2,则交换两个语句的位置不影响基本块的值.

- 例如

```
(1)      read (C)
(2)      A:= 0
(3)      B:= 1
(4) L1: A:=A + B
(5)      if  B>= C  goto  L2
(6)      B:=B+1
(7)      goto  L1
(8) L2:  write (A)
(9)      halt
```

划分成四个基本块 B1，B2，B3，B4



基本块内实行的优化:合并已知量

删除无用赋值

局部优化

- 基本块的DAG表示
  - 叶结点：以标识符或常数作为标记，表示该结点代表变量或常数的值。
  - 内部结点：以运算符作为标记，表示应用该运算符对其后继结点所代表的值进行运算的结果。
  - 结点可附加一个或多个标识符，表示这些变量的值等于该结点的值。

局部优化

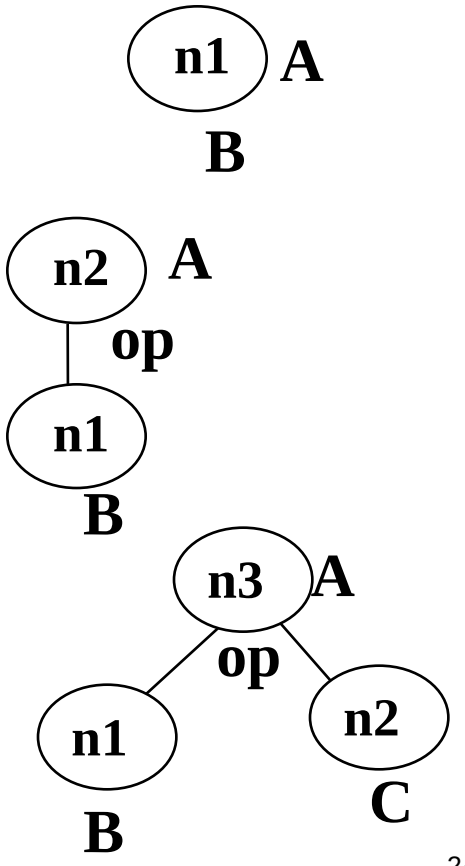
DAG 进行基本块的优化　　四元式

0 型：A:=B(:=,B, —,A)

1 型：A:=op B(op,B, —,A)

2 型: A:=B op C(op, B, C,A)

DAG结点



局部优化

- 仅含0，1，2型四元式的基本块的DAG构造算法:
  - 首先，DAG为空，对基本块的每一四元式，依次执行：

1. 如果NODE（B）无定义，则构造一标记为B的叶结点并定义NODE（B）为这个结点；
   - 如果当前四元式是0型，则记NODE(B)的值为n，转4。
   - 如果当前四元式是1型，则转2(1）。
   - 如果当前四元式是2型，则：如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C) 为这个结点；转2 (2)
2. （1）如果NODE(B)是标记为常数的叶结点，则转2(3)，否则转3（1）。

   （2）如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2（4），否则转3（2）。

   （3）执行op B（即合并已知量），令得到的新常数为P。如果NODE(B)是处理当前四元式时新构造出来的结点，则删除它。如果NODE(P)无定义，则构造一用P做标记的叶结点n。置NODE(P)=n，转4。

   （4）执行B op C（即合并已知量），令得到的新常数为P。如果NODE(B)或NODE(C)是处理当前四元式时新构造出来的结点，则删除它。如果NODE(P)无定义，则构造一用P做标记的叶结点n。置NODE(P)=n，转4。

局部优化

3. （1）检查DAG中是否已有一结点，其唯一后继为NODE(B)，且标记为op（即找公共子表达式）。如果没有，则构造该结点n，否则就把已有的结点作为它的结点并设该结点为n，转4。

- （2）检查中DAG中是否已有一结点，其左后继为NODE(B)，其右后继为NODE(C)，且标记为op（即找公共子表达式）。如果没有，则构造该结点n，否则就把已有的结点作为它的结点并设该结点为n，转4。

4. 如果NODE(A)无定义，则把A附加在结点n上并令NODE(A)=n；否则先把A从NODE(A)结点上附加标识符集中删除，把A附加到新结点n上并令NODE(A)=n。转处理下一四元式。而后，我们可由DAG重新生成原基本块的一个优化的代码序列。

(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

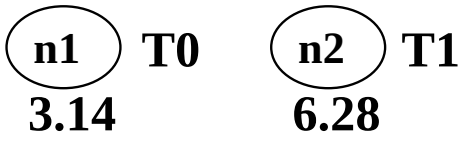
(1) T_0:=3.14
(2) T_1:=2*T_0
(3) T_2:=R+r
(4) A:=T_1*T_2
(5) B:=A
(6) T_3:=2*T_0
(7) T_4:=R+r
(8) T_5:=T_3*T_4
(9) T_6:=R-r
(10) B:=T_5*T_6

To

3.14

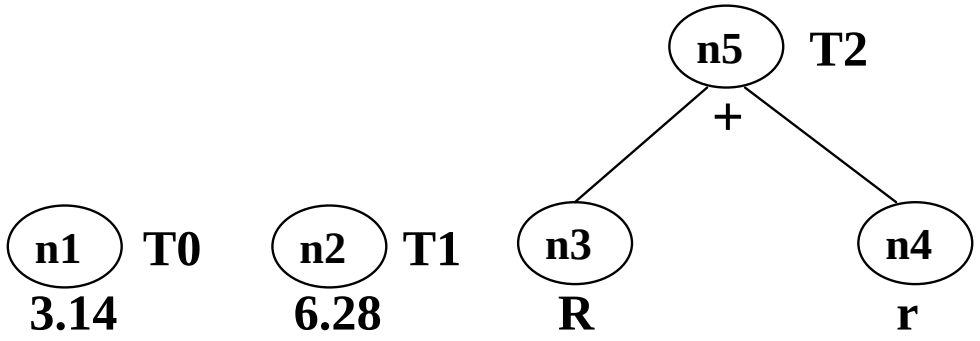
(a)

(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



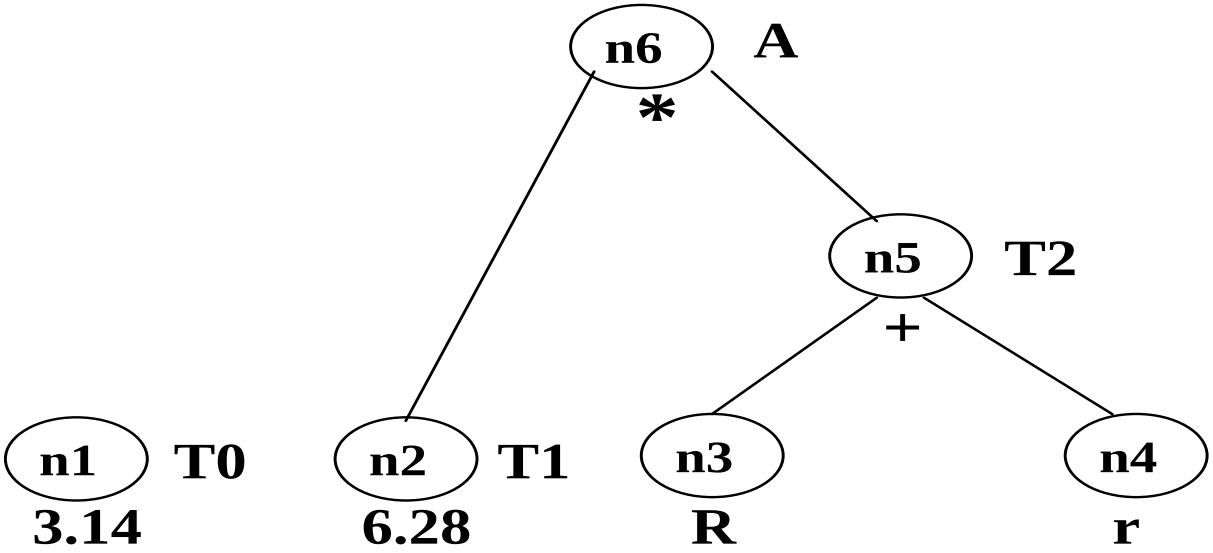
(b)

(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



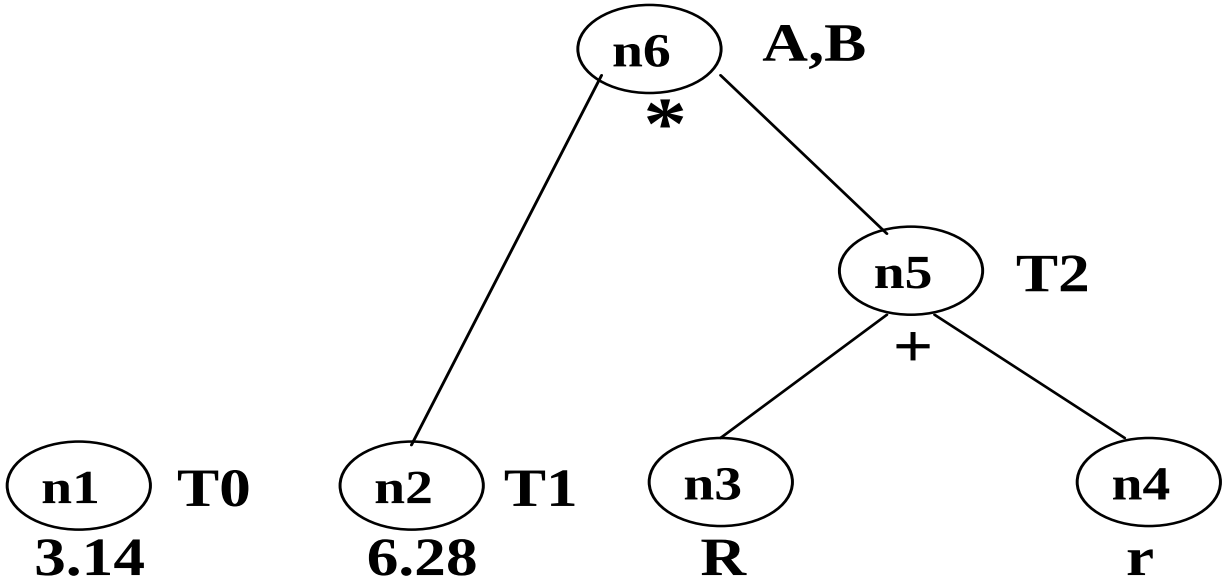
(c)

(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



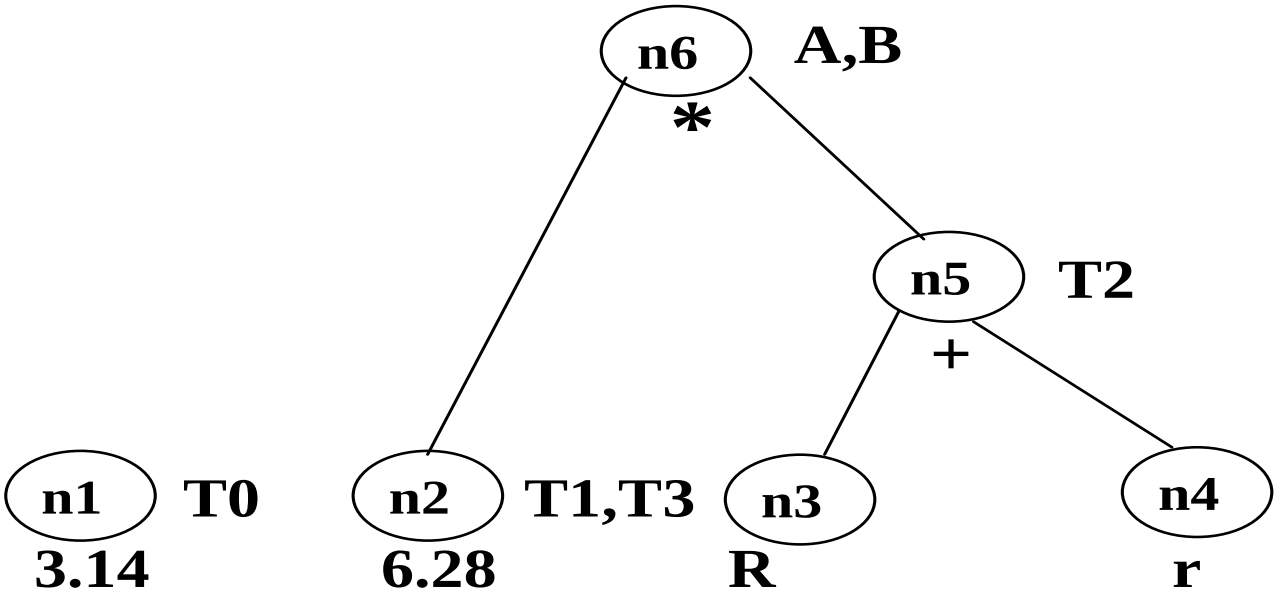
(d)

(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



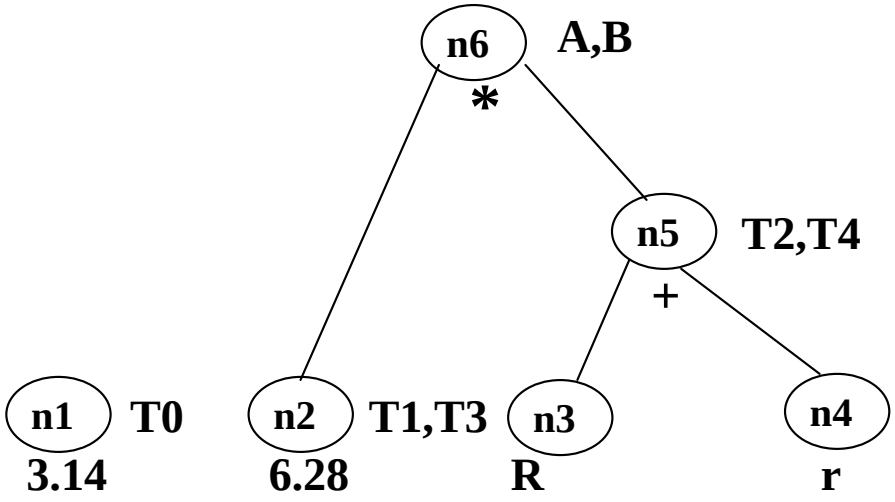
(e)

(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



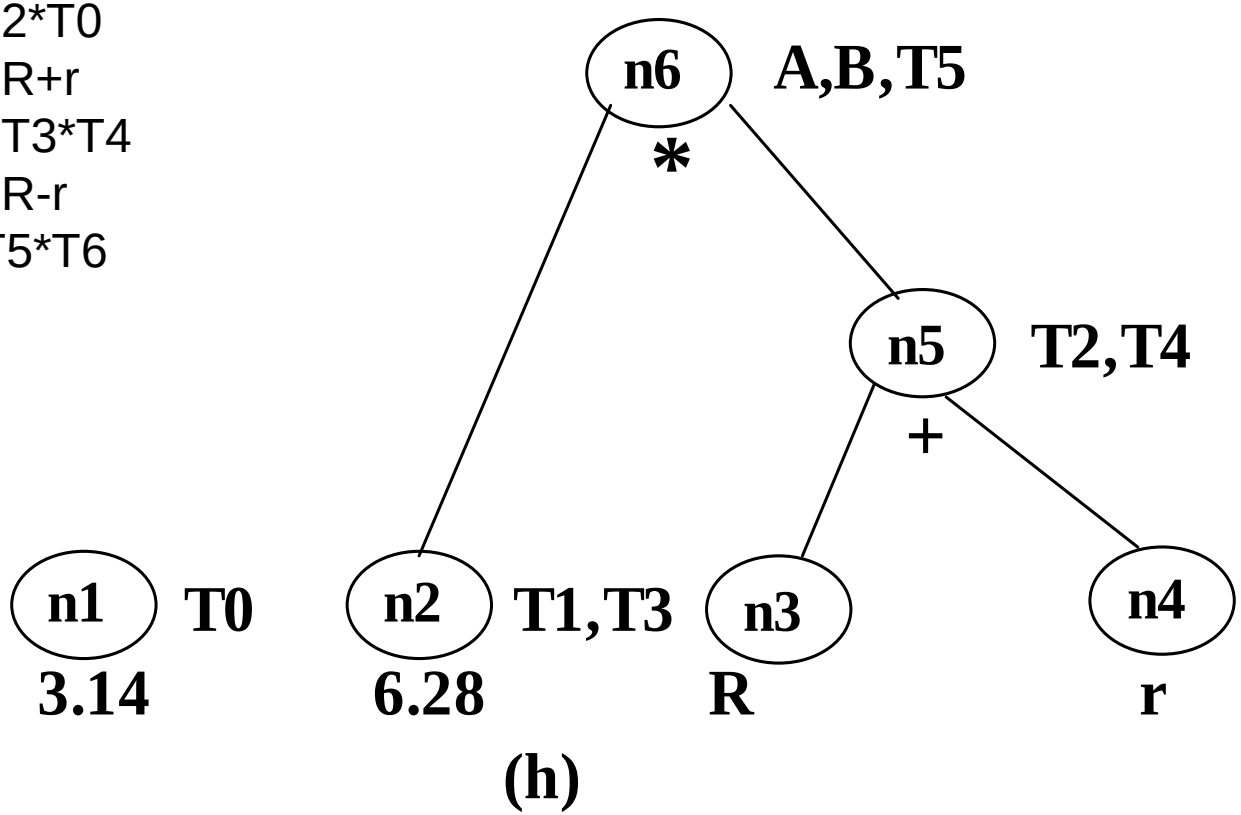
(f)

(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



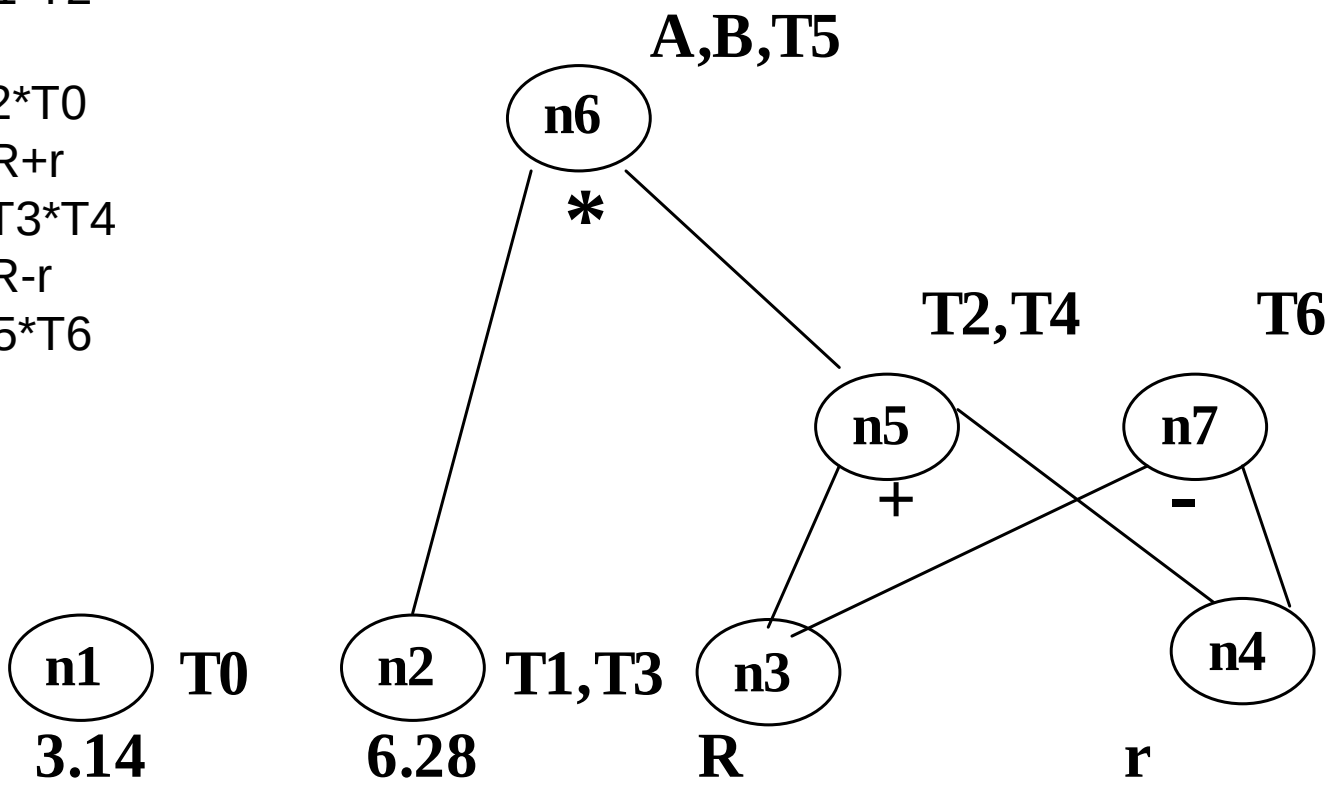
(g)

(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



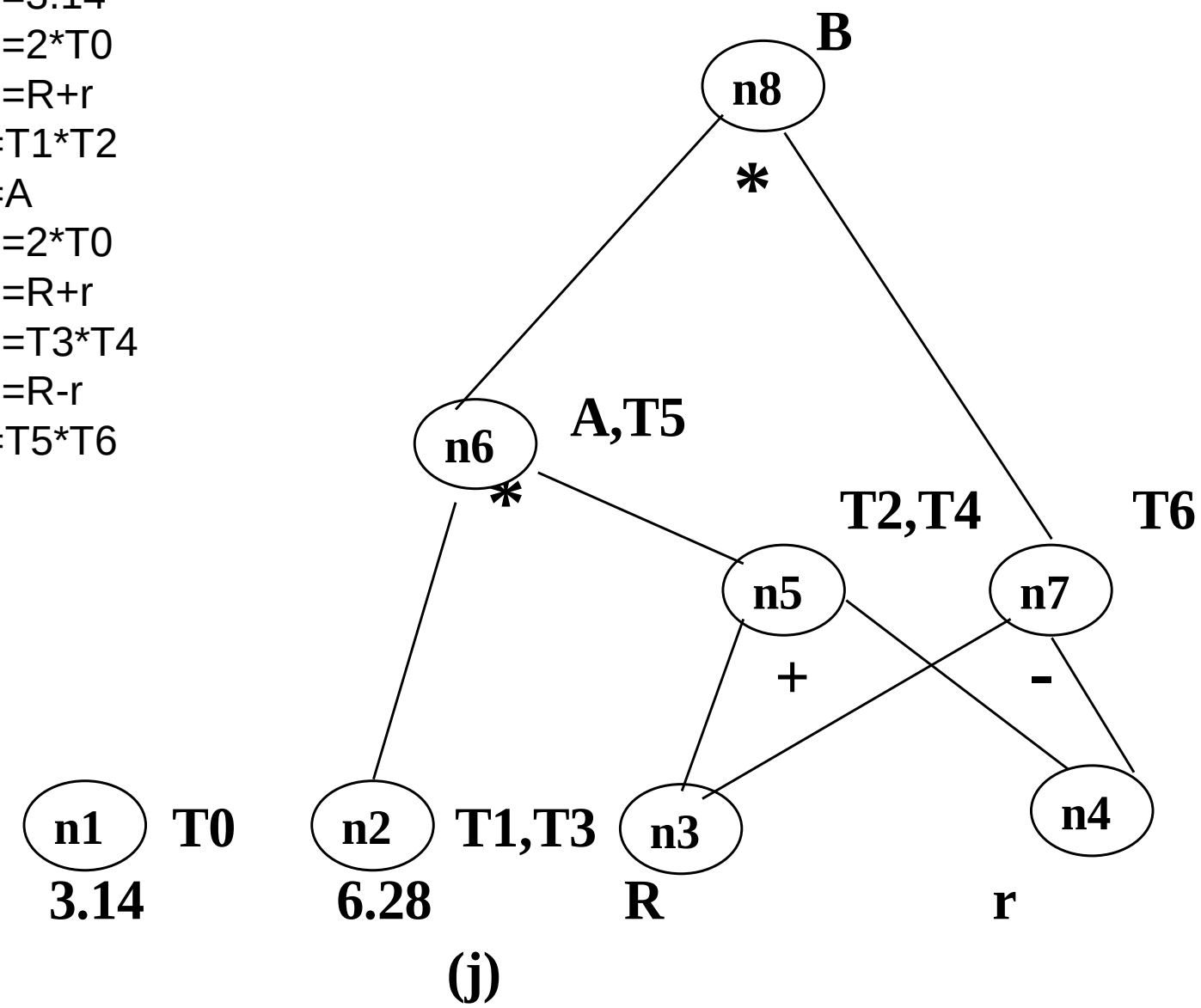
(h)

(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



(i)

(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



(j)

从DAG重建四元式序列算法

- 按照DAG图中的各个节点生成的次序，对于每个节点作如下处理：
  - 若是叶子节点，且附加标识符为空，不生成四元式。
  - 若是叶子节点，标记为x，附加标识符为z，生成（=， x，， z）
  - 若是内部节点，附加标识符为z，根据其标记op和子节点数目，生成下列2种形式的四元式。
    - 有两个子节点，生成（op，x，y，z）
    - 只有一个子节点，生成（op，x，_，z）
  - 若是内部节点，且无附加标识符，则添加一个局部于本基本块的临时性附加标识符，按照上一情况生成。
  - 如果节点的标识符包含多个附加标识符z1,z2,…,zk时：
    - 若是叶子节点，标记为z，生成一系列四元式

      =　　z　　z1

      =　　z　　z2

      … …　…

      =　　z　　zn
    - 不是叶子节点，生成四元式序列：

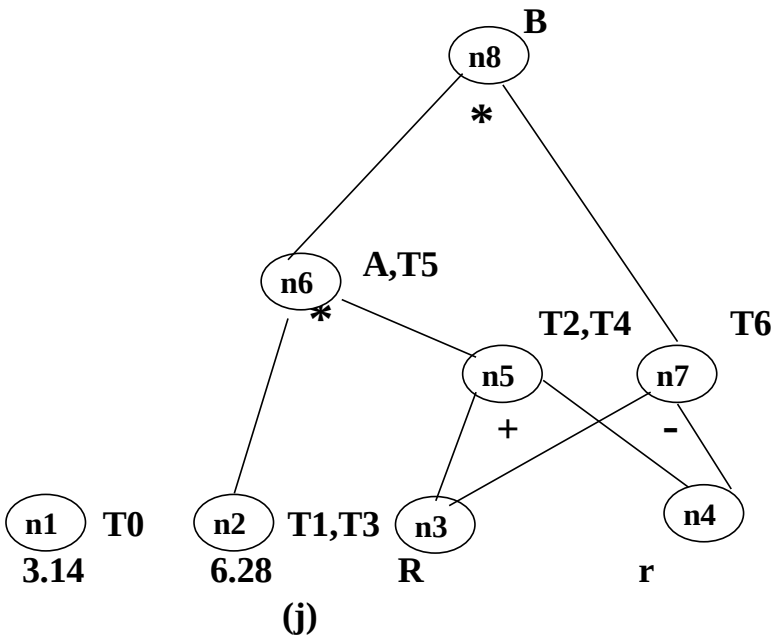
      =　　z1　　z2

      … …　…

      =　　z　　zn

从DAG重建四元式序列算法

T0:=3.14
T1:=2*T0
T2:=R+r
A:=T1*T2
B:=A
T3:=2*T0
T4:=R+r
T5:=T3*T4
T6:=R-r
B:=T5*T6



- T0:=3.14
- T1:=6.28
- T3:=6.28
- T2:=R+r
- T4:=T2
- A:=6.28*T2
- T5:=A
- T6:=R-r
- B:=A*T6

第9章 优化

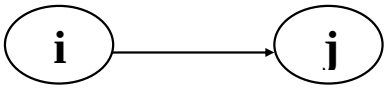
- 9.1 概述
- 9.2 局部优化
- 9.3 控制流分析和循环优化
- 9.4 数据流分析

9.3 控制流分析和循环优化

- 控制流分析
  - 确定程序流图及其结构
- 控制流程图
  - 找出程序中的循环
  - 具有唯一首结点的有向图
    - 首结点：从它开始到控制流程图中任何结点都有一条通路 的结点
  - 用三元组$G=(N,E,n_0)$表示
    - N：结点集，基本块集
    - E：有向边集 ，控制流方向
    - n_0：首结点，包含第一个语句的基本块
- 循环：程序中可能反复执行的代码序列

控制流分析和循环查找

有向边



①基本块 j 在程序的位置紧跟在 i 后,且 i 的出口语句不是转移或停语句

②i 的出口是 goto(S)或

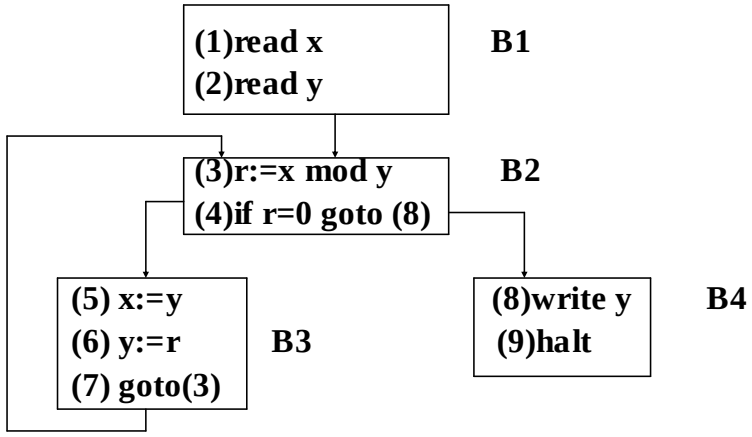
if goto(S),而(S)

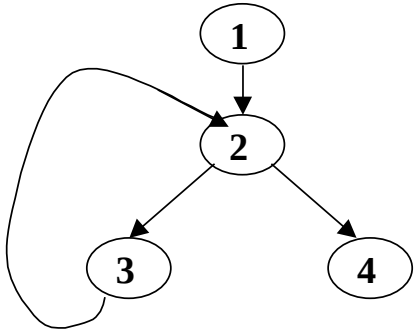
是 j 的入口语句

控制流分析和循环查找

- 例：

*(1)read x
(2)read y
*(3)r:=x mod y
(4)if r=0 goto (8)
*(5)x:=y
(6)y:=r
(7)goto(3)
*(8)write y
(9)halt





循环优化

- 三种技术
  - 代码外提
  - 强度削弱
  - 删除归纳变量

循环优化

- 代码外提
  - 目的：减少循环中代码数目
  - 方法：把循环中的不变运算放到循环前面（前置结点），书P288图
  - 例
    - 计算半径为r的从10度到360度的扇形的面积：
      - for(n=1; n<36; n++)
      - {S:=10/360*pi*r*r*n; printf(“Area is %f”, S); }
    - 显然，表达式10/360*pi*r*r中的各个量在循环过程中不改变。可以修改程序如下：
      - C= 10/360*pi*r*r;
      - for(n=1; n<36; n++)
      - {S:=C*n; printf(“Area is %f”, S); }
    - 修改后的程序中，C的值只需要被计算一次，而原来的程序需要计算36次。
  - 查找循环中不变运算的算法
  - 代码外提算法

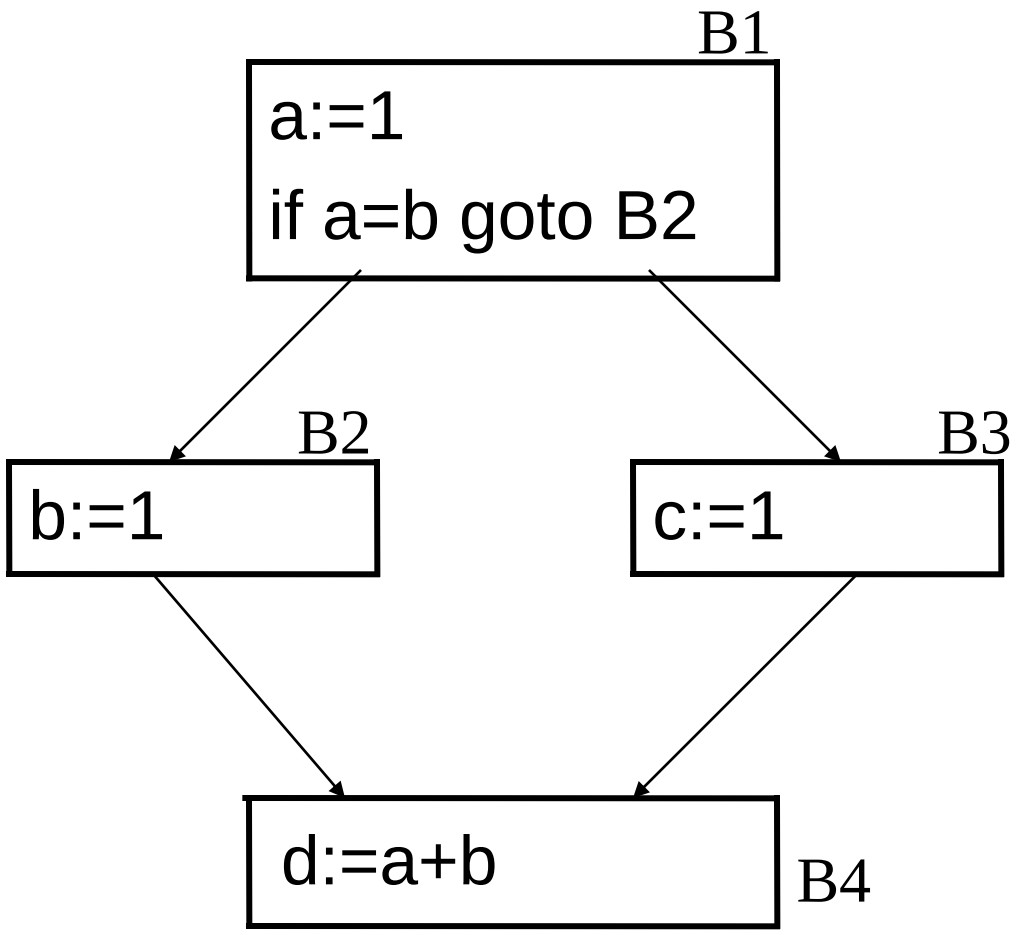
第9章 优化

- 9.1 概述
- 9.2 局部优化
- 9.3 控制流分析和循环优化
- 9.4 数据流分析

活跃变量的数据流方程

- 例如

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



活跃变量的数据流方程

- 提取Def(在B中定值的变量集合)和LiveUse (B中被定值之前要引用变量的集合)集合

基本块	Def	LiveUse
B_1	{a}	{b}
B_2	{b}	Ø
B_3	{c}	Ø
B_4	{d}	{a, b}