第九章　代码优化

廖力
东南大学

第一节 概述

一、代码优化的目的

- 提高目标代码的运行效率。
  - 注：效率是指目标代码运行时间较短，占有空间较少。

二、代码优化的实质

- 代码优化实际上是对代码进行等价变换，由一组代码变成运行结果相同的另一组代码。

第一节 概述

三、优化级别

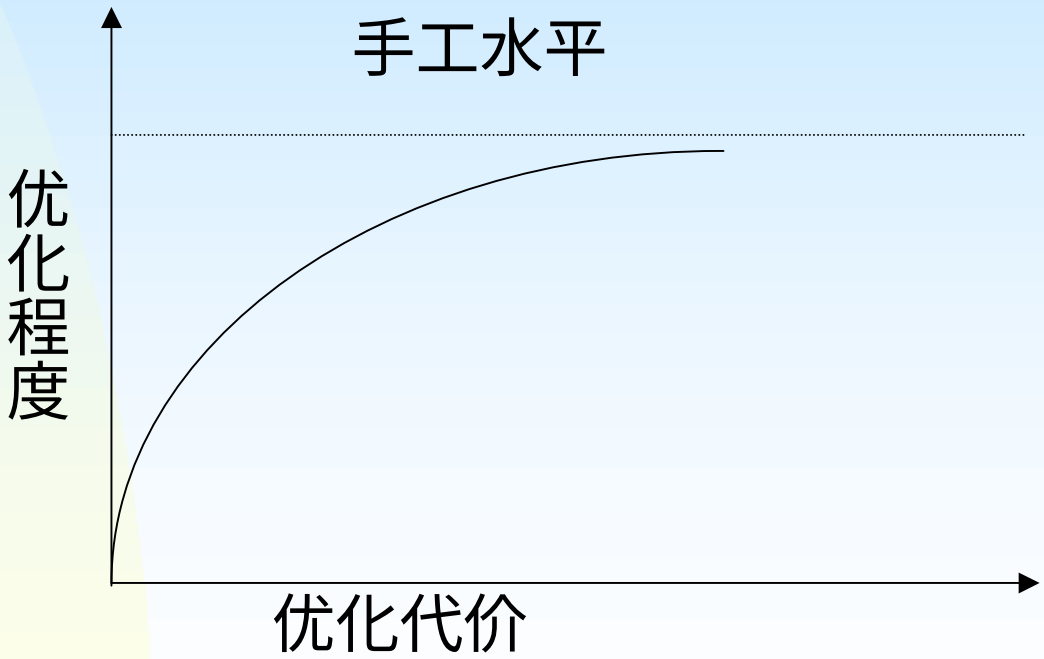
- 源程序级：语言和算法级
- 中间代码级：
  - 是主要优化内容，与计算机硬件无关。
  - 注：本章内容主要是中间代码优化。
- 目标代码级

四、优化代价

- 指优化算法的复杂性，即编程难易度及算法时间复杂度。

第九章　代码优化

- 五、优化程度和优化代价的关系：



9.1　优化概述

一、中间代码优化分类

- 1、局部优化
- 1）含义：局部优化是针对一段顺序执行语句序列的优化。
- 2）主要方法
  - 合并已知量
  - 删除公共子表达式
  - 变量传播与无用赋值删除

- 例如：下面程序用来求圆环内外圆周之和及圆环正反两面面积之和。对其优化：
- Pi:=3.14
- A:＝2*Pi*(R+r);
- B:=A;
- B:=2* Pi*(R+r)*(R-r)
- 由语法制导翻译生成右边的四元式序列

(1)Pi:=3.14
(2)T_1:=2*Pi
(3)T_2:=R+r
(4)A:=T_1*T2
(5)B:=A
(6)T3:=2*Pi
(7)T4:=R+r
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6

9.1 优化概述

- 1、局部优化
- 1)合并已知量
  - 对于编译时已知的常量，不必生成目标代码到运行时才计算，而是可以直接对它计值，并用计算结果代替表达式的计算代码。

(1)Pi:=3.14
(2)T_1:=2*Pi
(3)T2:=R+r
(4)A:=T_1*T2
(5)B:=A
(6)T3:=2*Pi
(7)T4:=R+r
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6

(1)Pi:=3.14
(2) T_1 :=6.28
(3)T2:=R+r
(4)A:= T_1 *T2
(5)B:=A
(6)T3:=6.28
(7)T4:=R+r
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6

9.1 优化概述

- 1、局部优化
- 2)删除公共子表达式
  - 对于两个相同的表达式计算，若其计算结果相同，则没有必要重复地生成两条运算指令。

(1)Pi:=3.14
(2) T_1 :=6.28
(3)T2:=R+r
(4)A:= T_1 *T2
(5)B:=A
(6)T3:=6.28
(7)T4:=R+r
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6

(1)Pi:=3.14
(2) T_1 :=6.28
(3)T2:=R+r
(4)A:= T_1 *T2
(5)B:=A
(6)T3:= T_1
(7)T4:=T2
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6

9.1 优化概述

- 1、局部优化
- 3)变量传播与无用赋值删除

(1)Pi:=3.14
(2)T1:=6.28
(3)T2:=R+r
(4)A:=T1*T2
(5)B:=A
(6)T3:=T1
(7)T4:=T2
(8)T5:=T3*T4
(9)T6:=R-r
(10)B:=T5*T6

(1)Pi:=3.14
(2) T_1 :=6.28
(3)T2:=R+r
(4)A:= T_1 *T2
(5)B:=A
(6)T3:= T_1
(7)T4:=T2
(8)T5:= T_1 *T2
(9)T6:=R-r
(10)B:=T5*T6

(1)Pi:=3.14
(2) T_1 :=6.28
(3)T2:=R+r
(4)A:= T_1 *T2
(8)T5:= T_1 *T2
(9)T6:=R-r
(10)B:=T5*T6

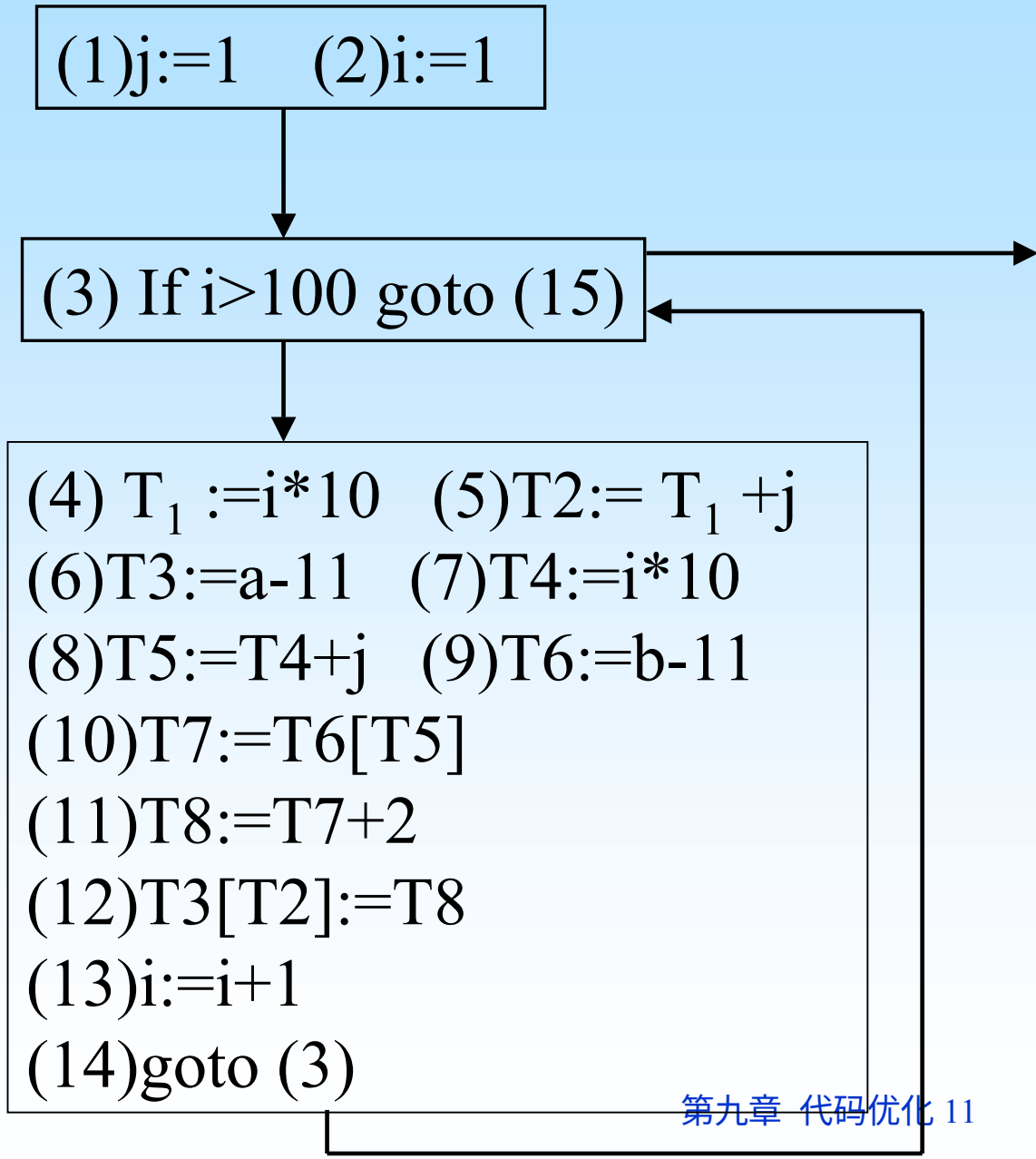
9.1 优化概述

一、中间代码优化分类

2、循环优化简介

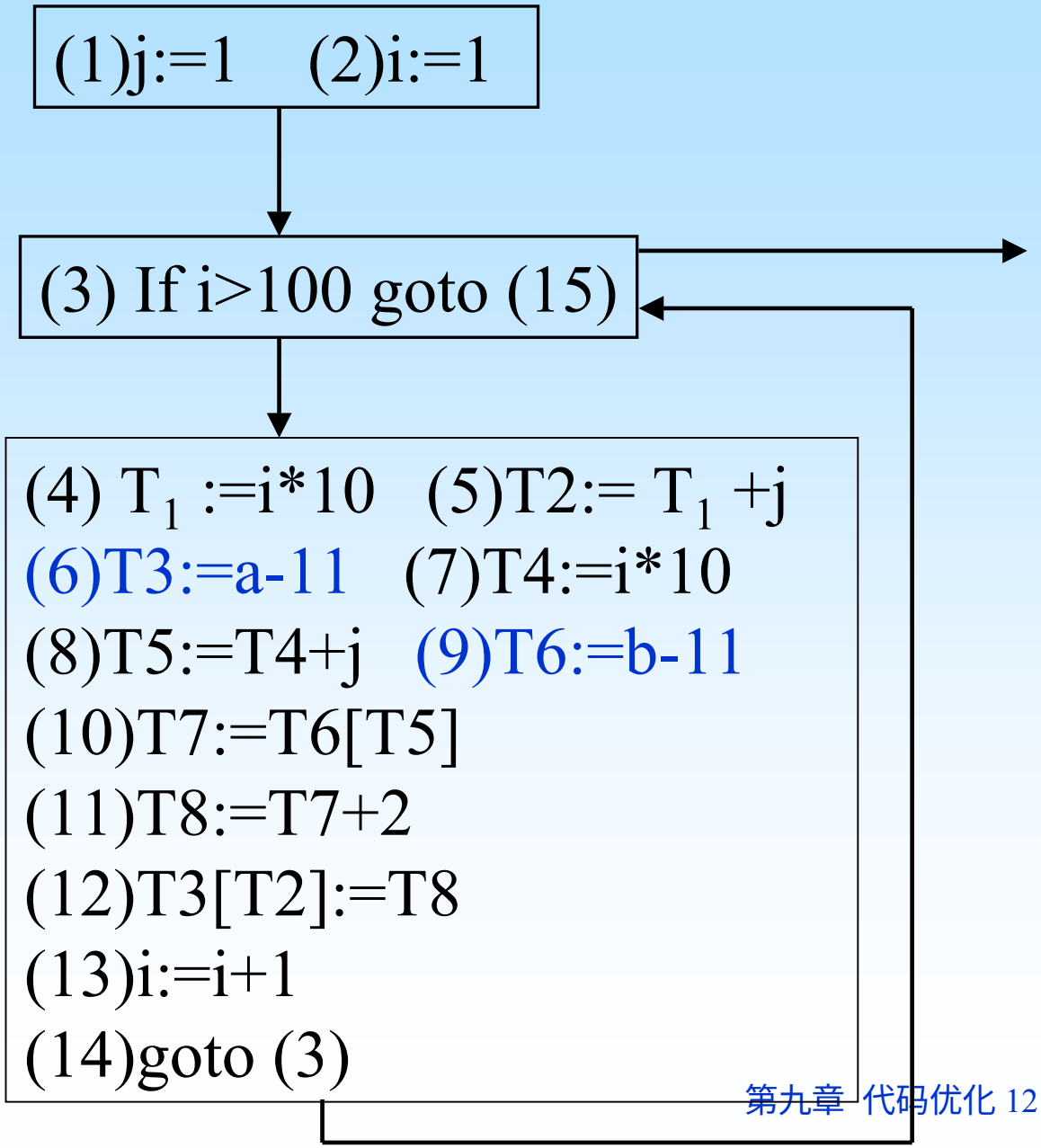
- 主要方法
  - 循环不变运算外提（代码外提）
  - 降低运算强度
  - 变换循环控制变量并删除其自增赋值式

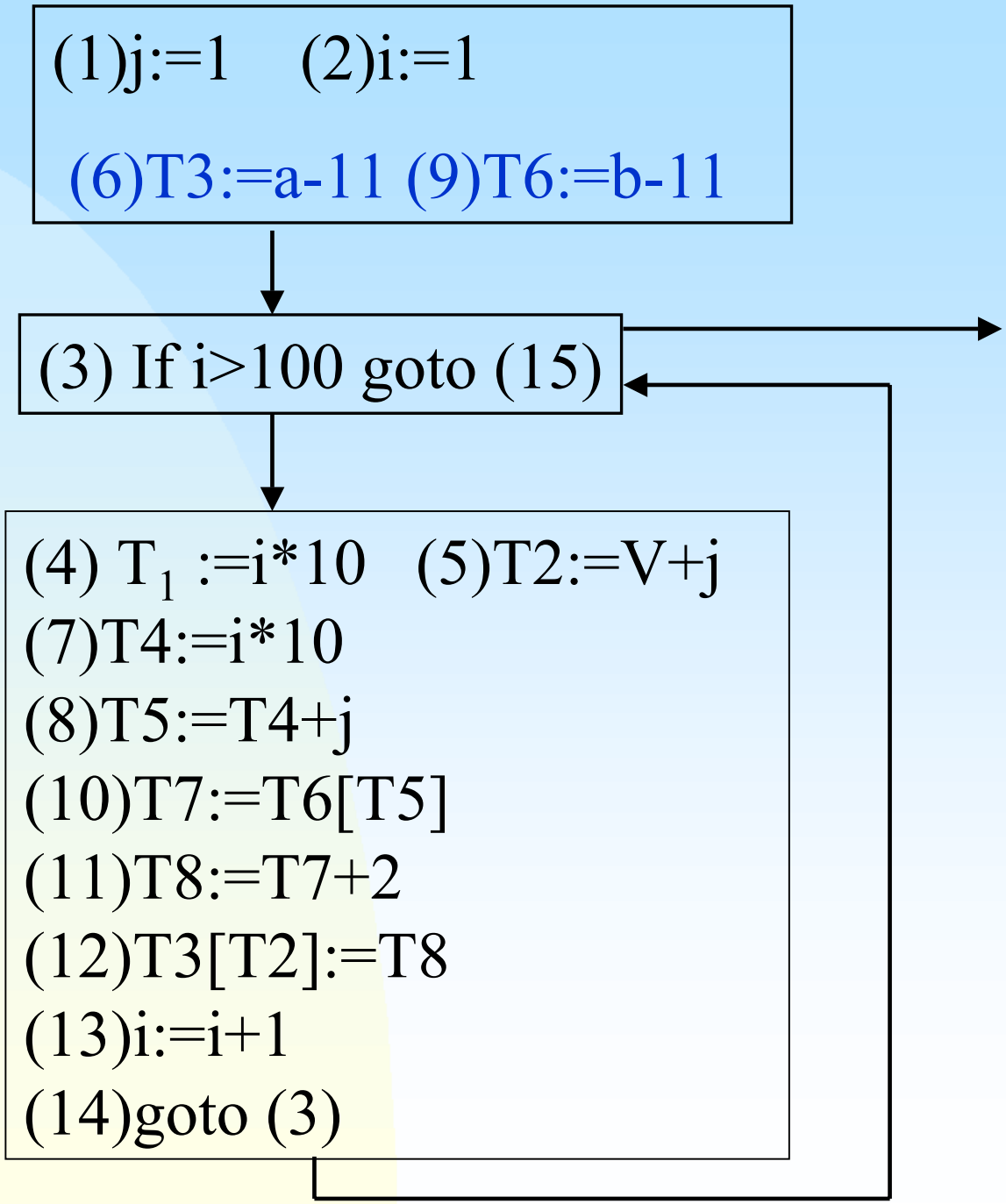
- 例如：有如下一段源程序
- j:=1;
- for i=1 to 100 do
- A[i,j]:=B[i,j] +2
- 其中数组说明为
  - A,B:array[1:100, 1:10]
  - 假定每个数组元素占1字编址
- 得其四元式序列



9.1 优化概述

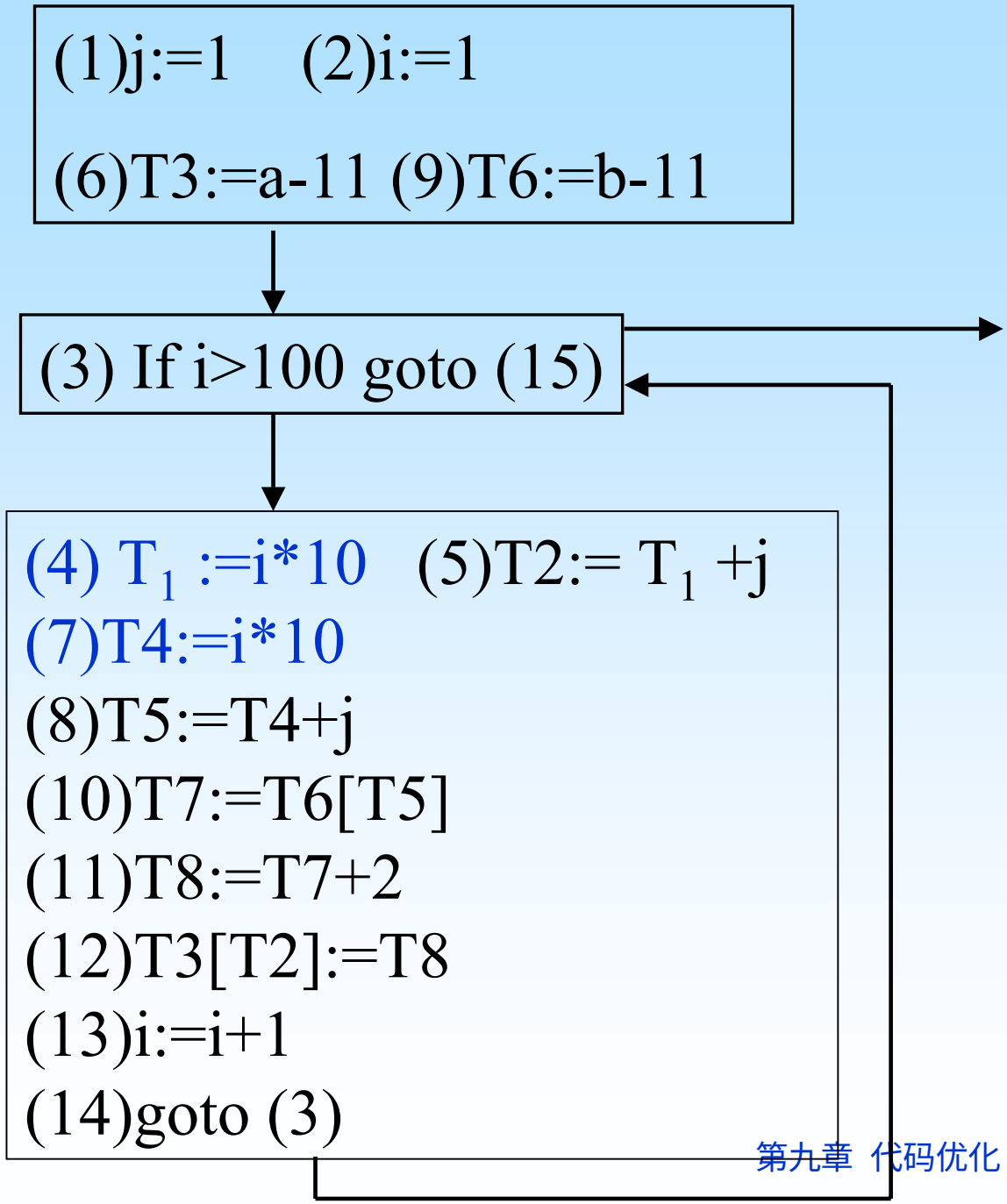
- 2、循环优化简介
- 1)循环不变运算外提（代码外提）
  - 将循环内不变运算提到循环外，可以使它只计算一次。节省时间。

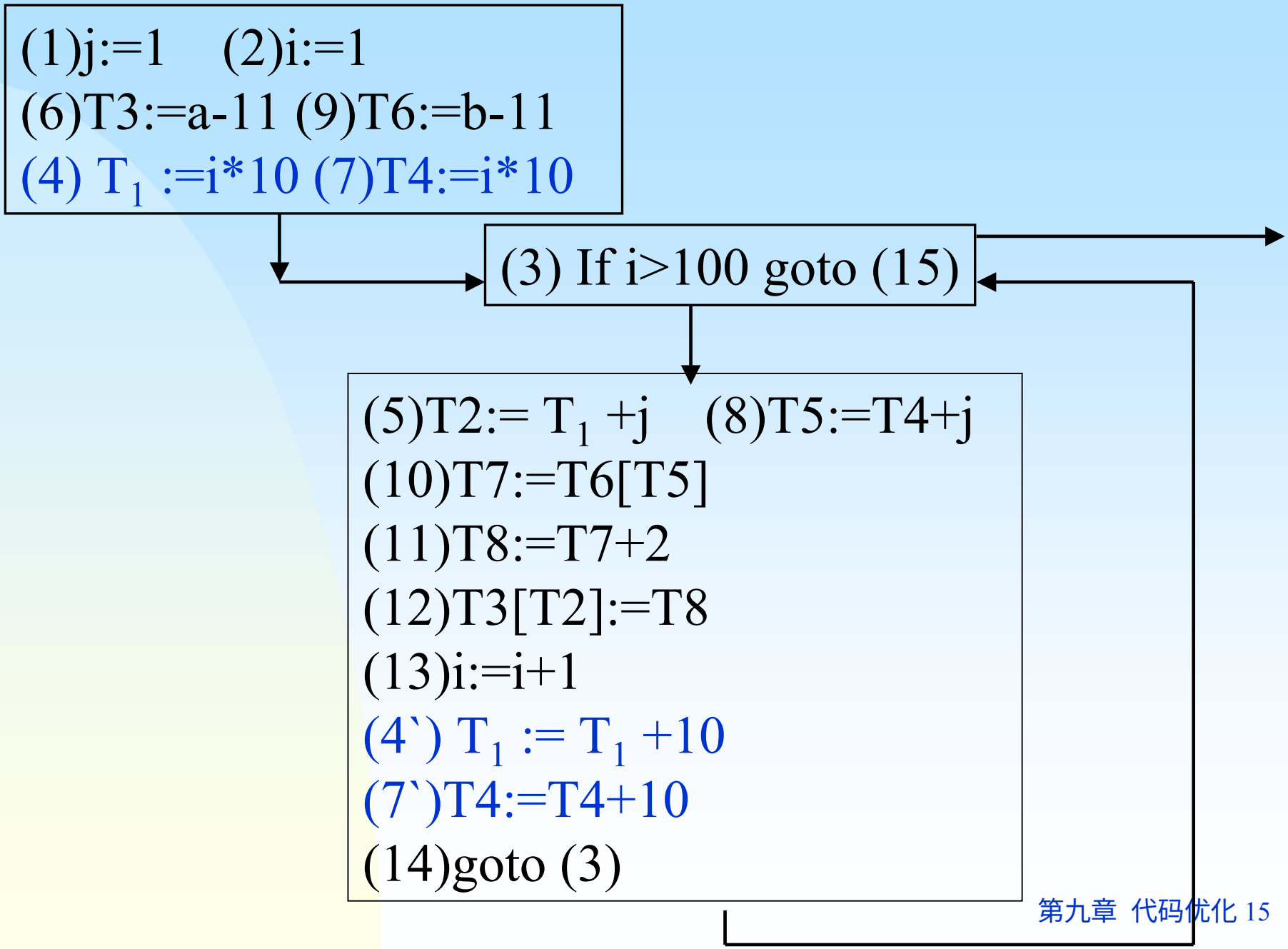



(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(3) If i>100 goto (15)
(4) T1 :=i*10 (5)T2:=V+j
(7)T4:=i*10
(8)T5:=T4+j
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(13)i:=i+1
(14)goto (3)

9.1 优化概述

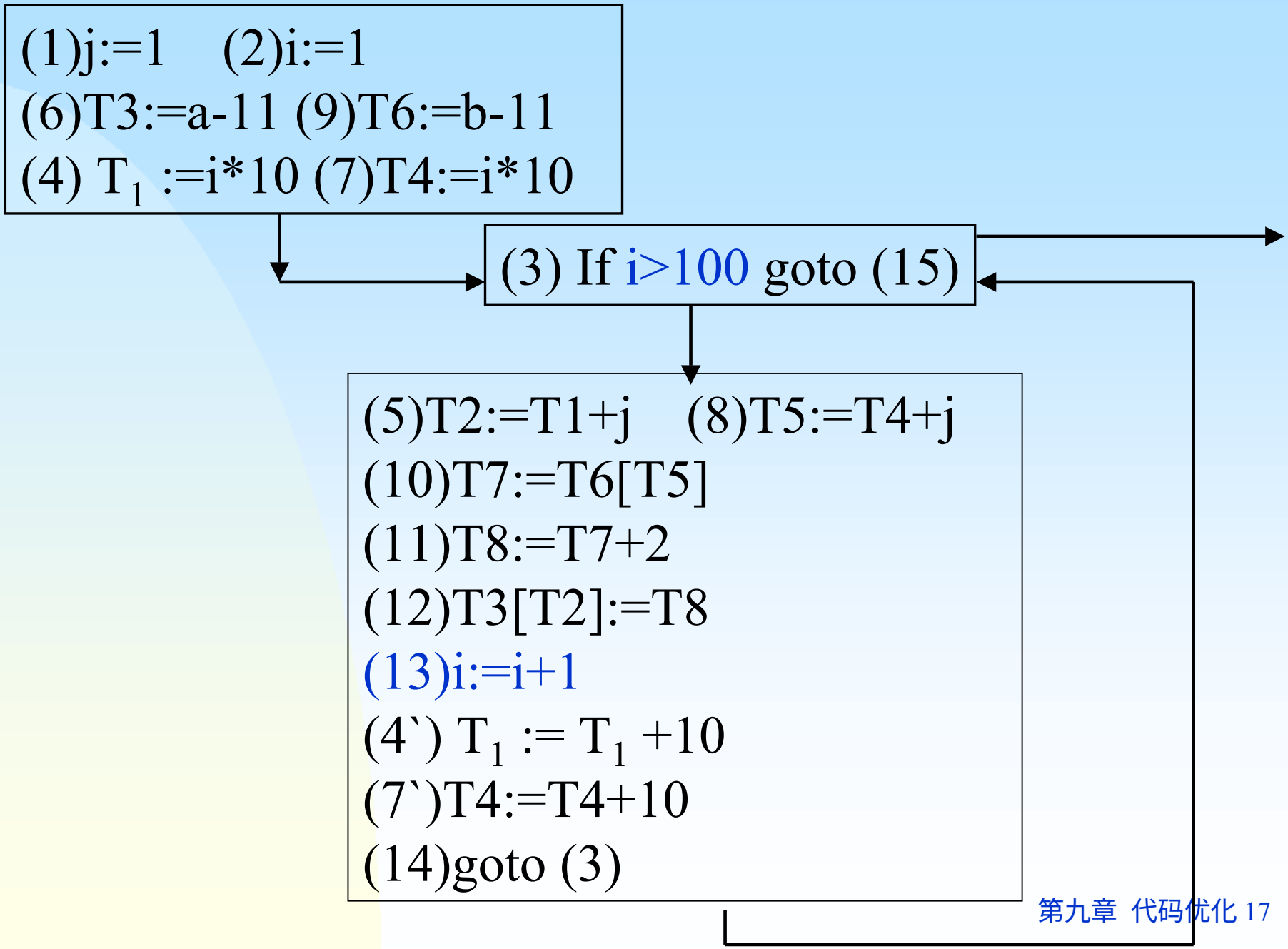
- 2、循环优化
- 2)降低运算强度
  - 由于乘法运算比加法运算要多花时间，将循环内得乘法运算转换为加法运算，可节省大量时间。

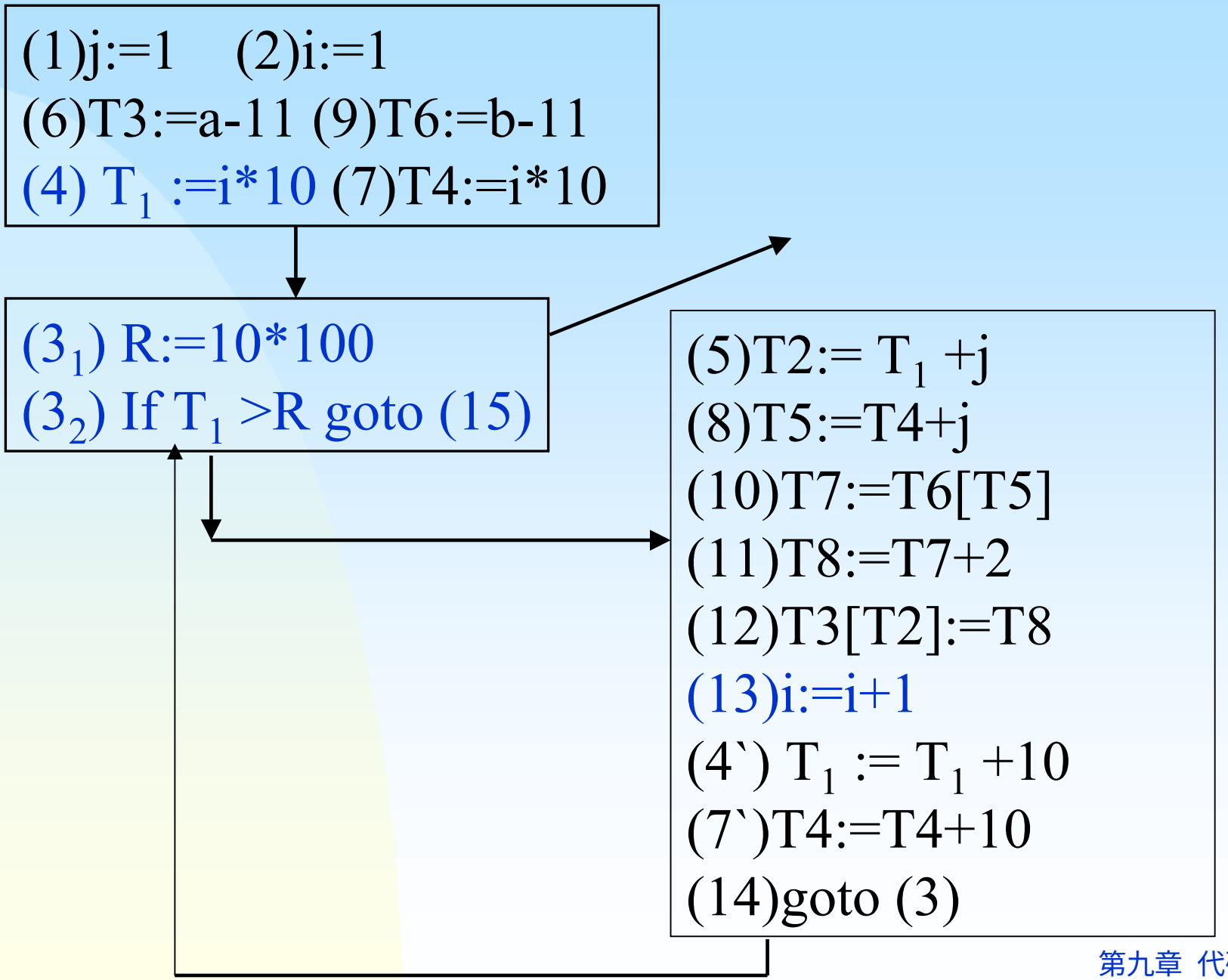


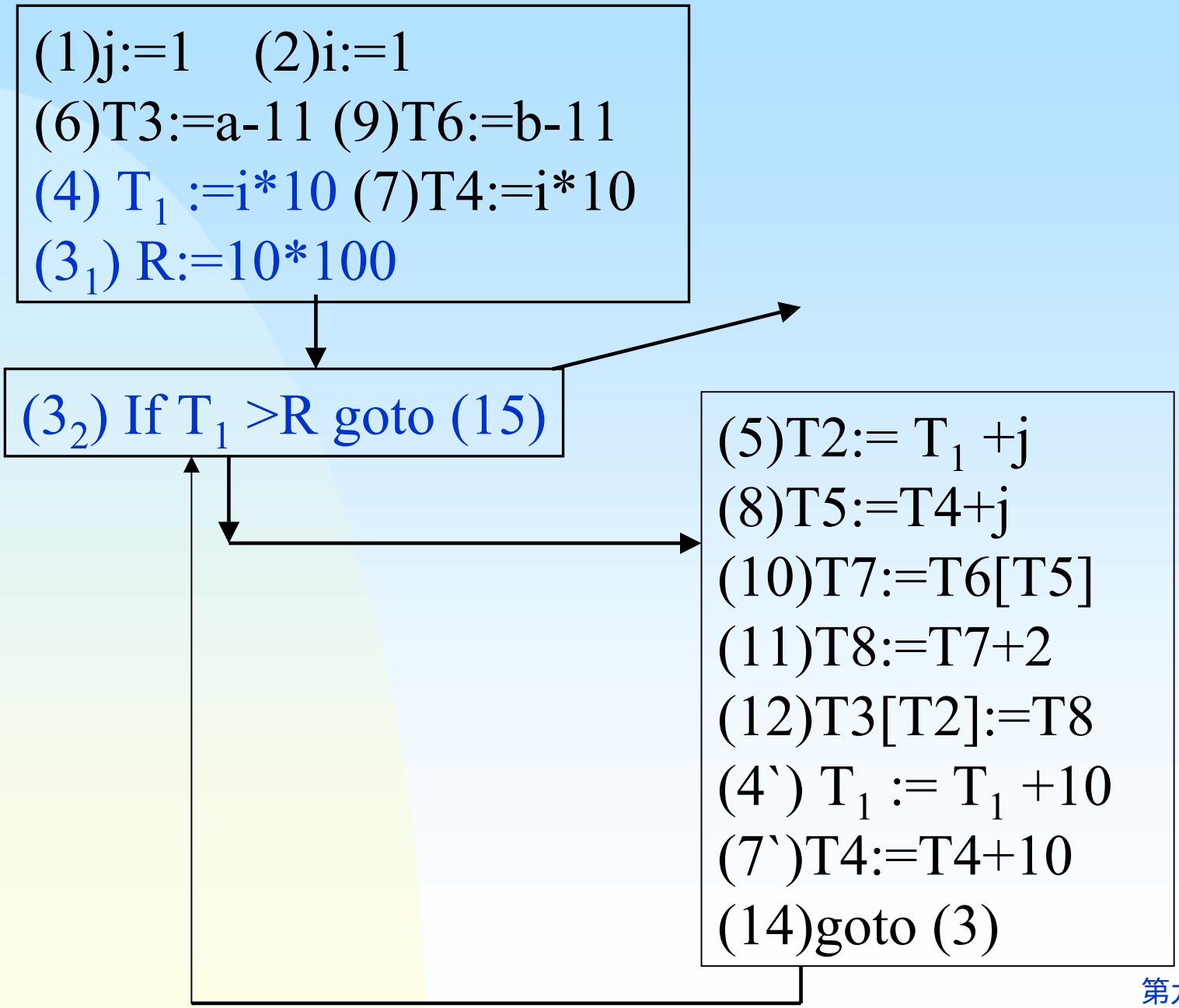

(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=i*10 (7)T4:=i*10
(3) If i>100 goto (15)
(5)T2:= T1 +j (8)T5:=T4+j
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(13)i:=i+1
(4`) T1 := T1 +10
(7`)T4:=T4+10
(14)goto (3)

9.1 优化概述

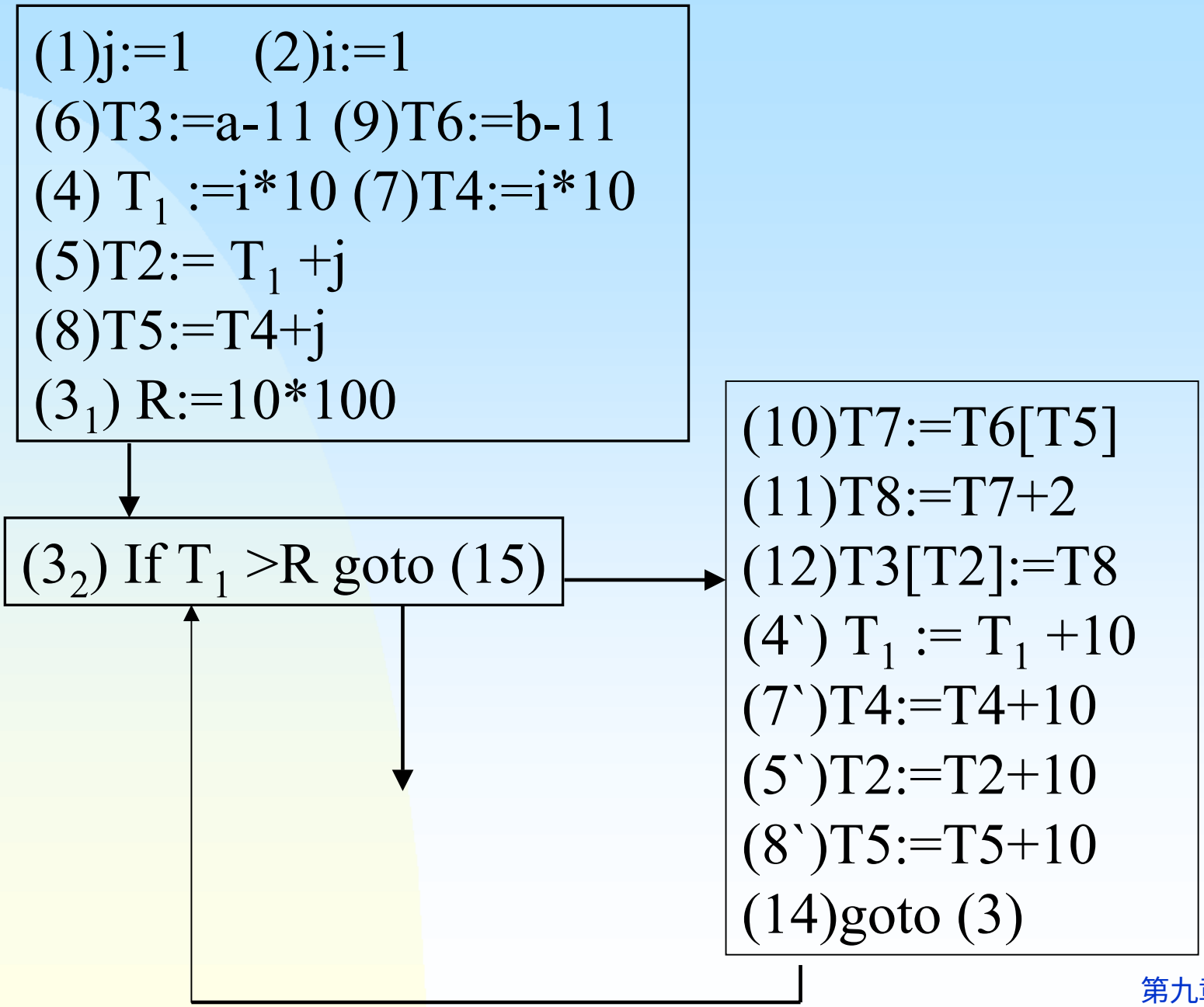
- 2、循环优化
- 3)变换循环控制变量i并删除自增赋值式
  - 循环控制变量用于：
    - 控制循环的进行
    - 计算其他变量的值


(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=i*10 (7)T4:=i*10
(3) If i>100 goto (15)
(5)T2:=T1+j (8)T5:=T4+j
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(13)i:=i+1
(4`) T1 := T1 +10
(7`)T4:=T4+10
(14)goto (3)


(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=i*10 (7)T4:=i*10
(31) R:=10*100
(32) If T1 >R goto (15)
(5)T2:= T1 +j
(8)T5:=T4+j
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(13)i:=i+1
(4`) T1 := T1 +10
(7`)T4:=T4+10
(14)goto (3)


(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=i*10 (7)T4:=i*10
(31) R:=10*100
(32) If T1 >R goto (15)
(5)T2:= T1 +j
(8)T5:=T4+j
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(4`) T1 := T1 +10
(7`)T4:=T4+10
(14)goto (3)

- 将(5)T2:= T_1 +j转换为自增式
- 因为： T2:= T_1 +j即T_2^{i+1}= $T_1^{\ i+1}$ +j
  - 所以： T_2^{i+1}= $T_1^{\ i+1}$ +j
  - = $T_1^{\ i}$ +10+j
  - = T_2^{i} +10
- 故：(5)T2:= T_1 +j 转换为 (5`)T2:=T2+10 ;
  - 同理(8)T5:=T4+j转换为(8`)T5:=T5+10。
  - 它们需要在循环前计算一次T2:= T_1 +j和T5:=T4+j,故得到

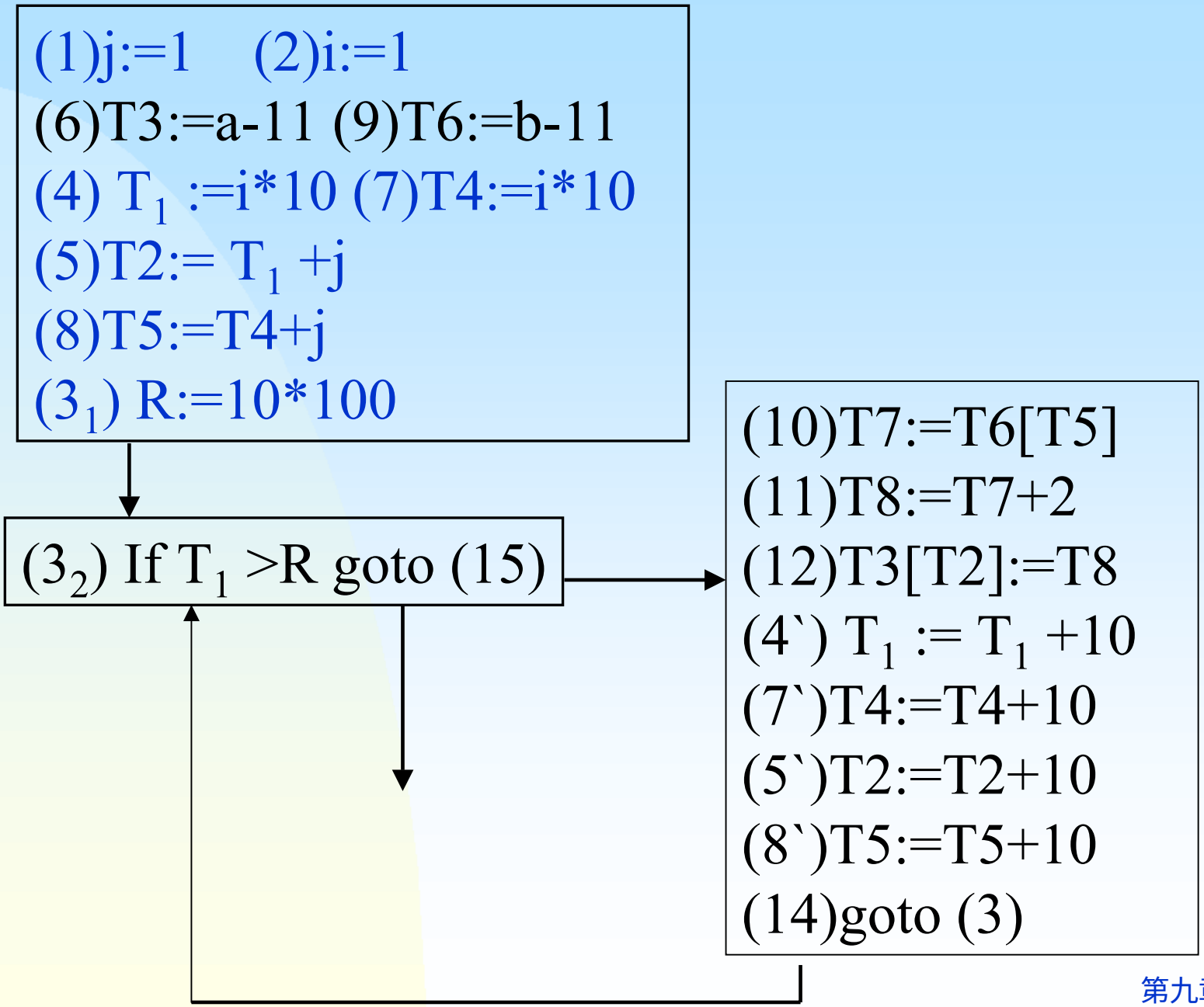

(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=i*10 (7)T4:=i*10
(5)T2:= T1 +j
(8)T5:=T4+j
(31) R:=10*100
(32) If T1 >R goto (15)
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(4`) T1 := T1 +10
(7`)T4:=T4+10
(5`)T2:=T2+10
(8`)T5:=T5+10
(14)goto (3)

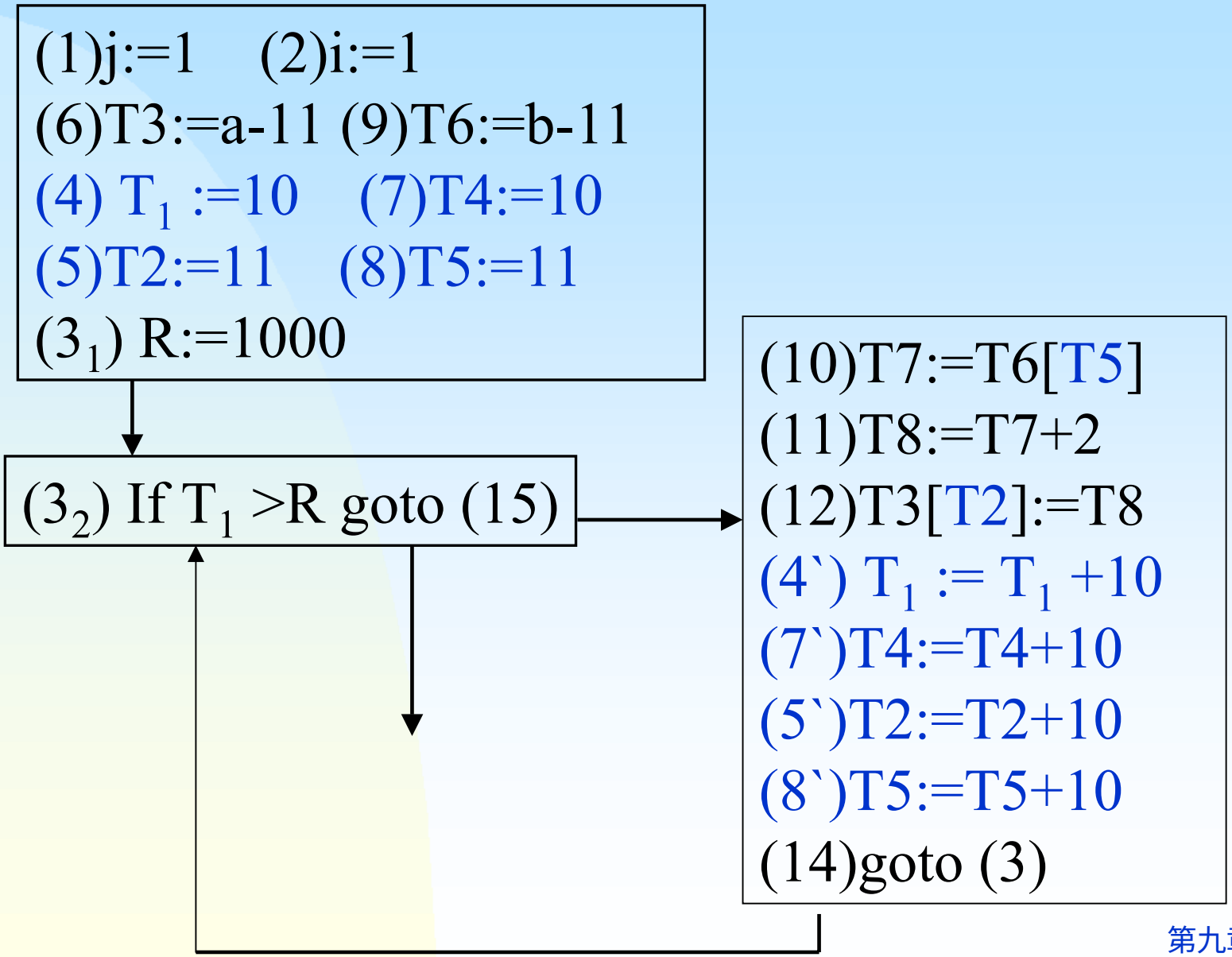
9.1 优化概述

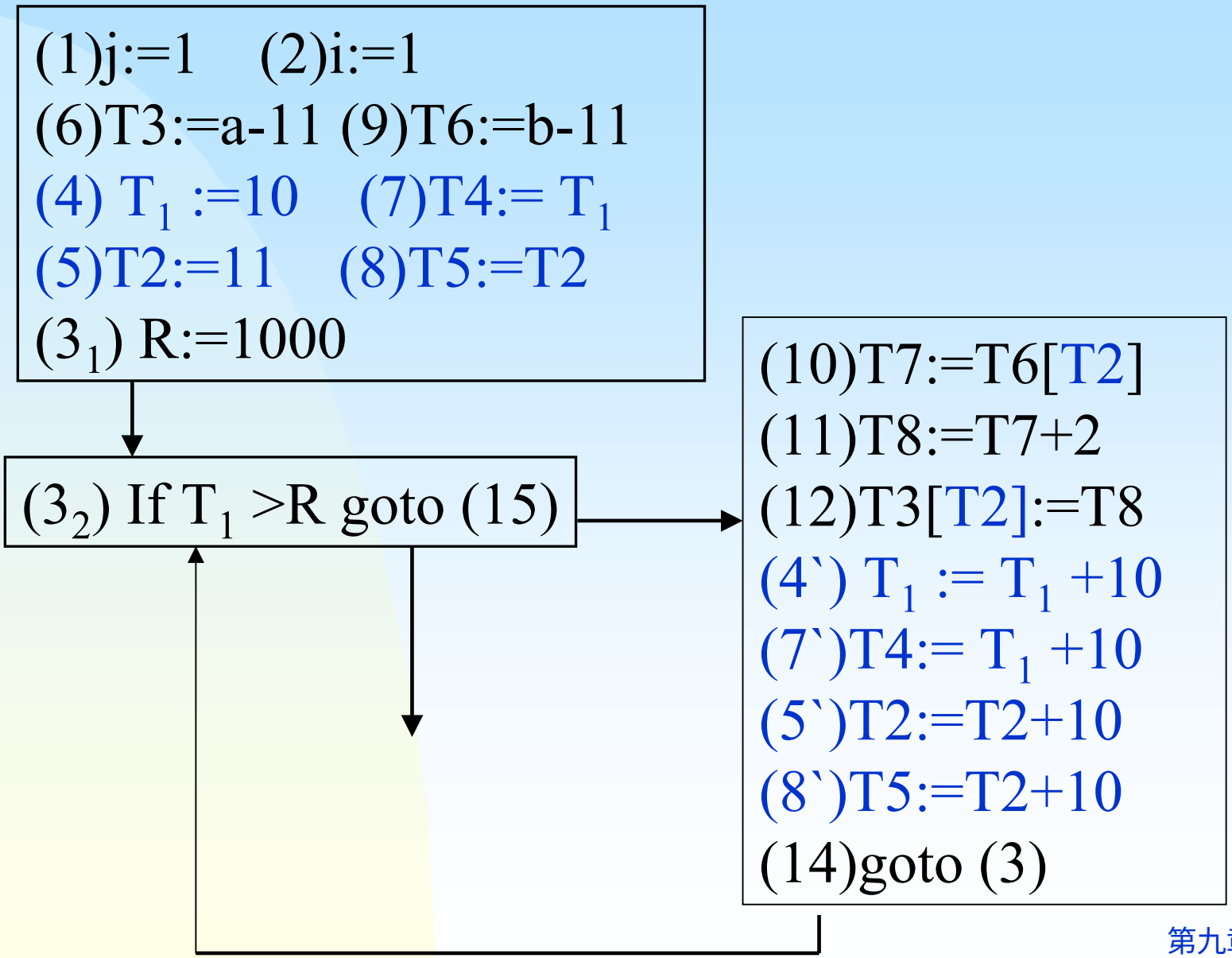
一、中间代码优化分类

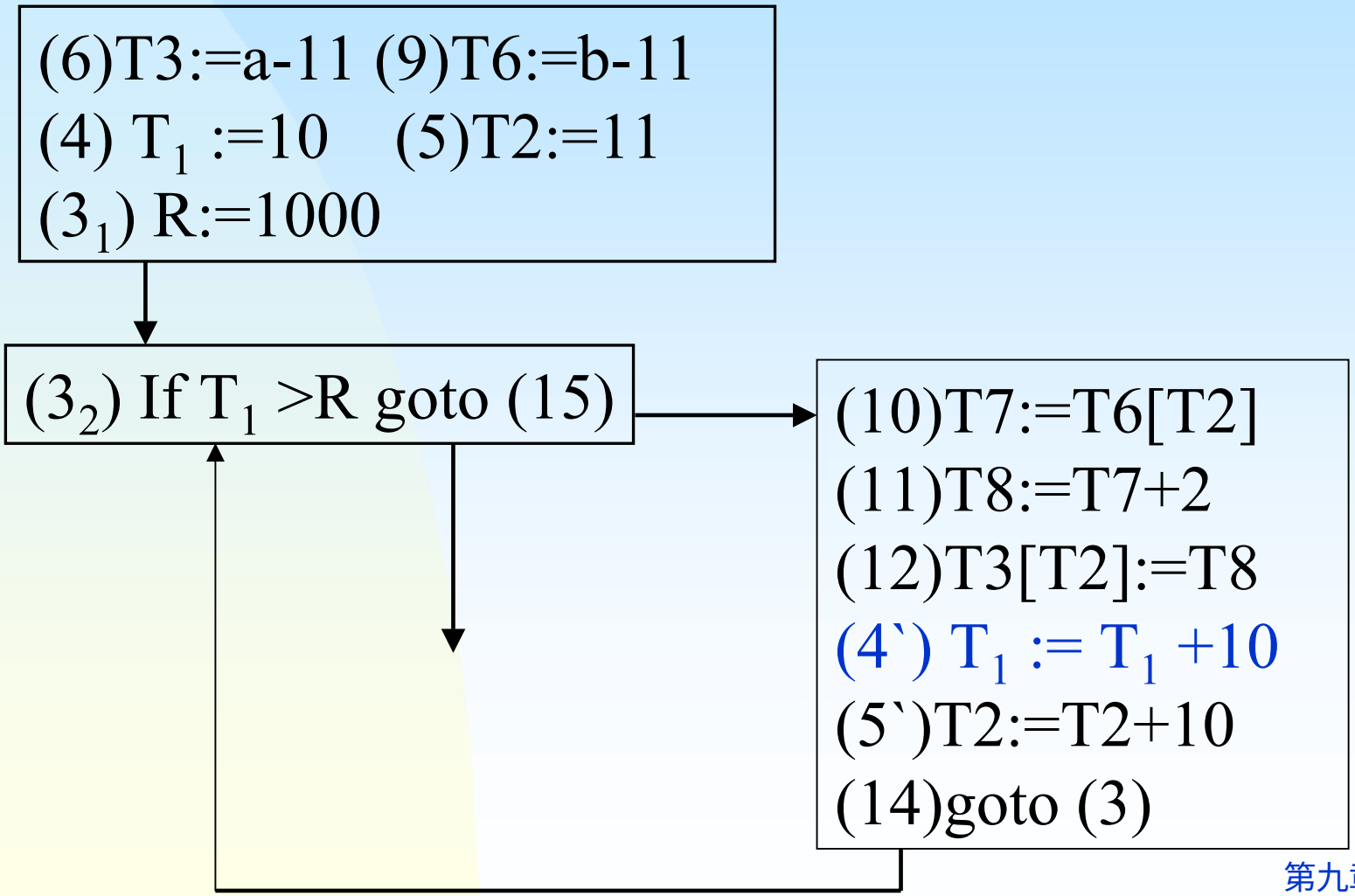
3、全局优化简介

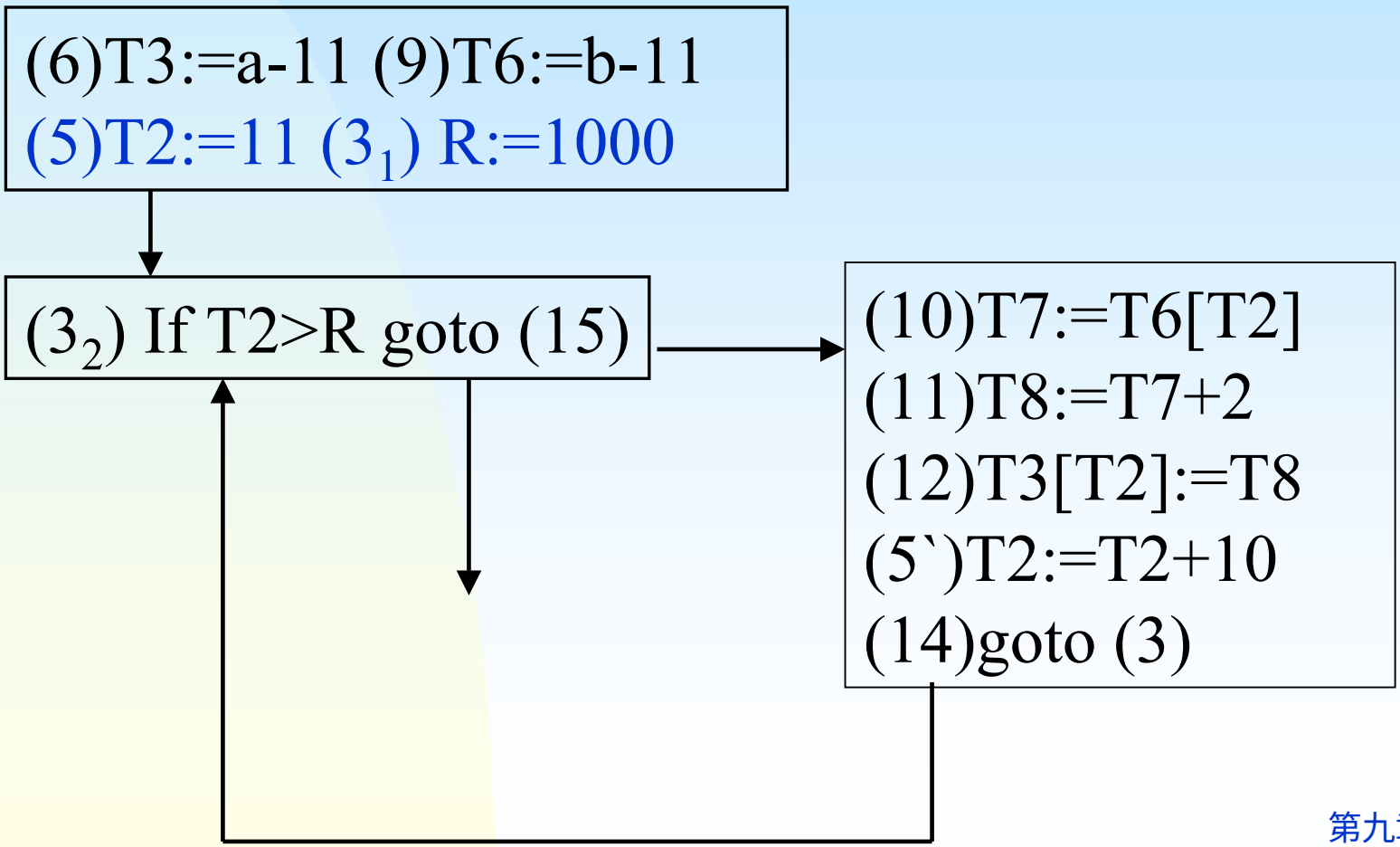
- 经过循环优化后，再做局部优化。进行合并已知量、删除公共子表达式和变量传播等操作。
- 再合并整个程序中的已知量，删除不同块内的公共子表达式等。


(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=i*10 (7)T4:=i*10
(5)T2:= T1 +j
(8)T5:=T4+j
(31) R:=10*100
(32) If T1 >R goto (15)
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(4`) T1 := T1 +10
(7`)T4:=T4+10
(5`)T2:=T2+10
(8`)T5:=T5+10
(14)goto (3)


(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=10 (7)T4:=10
(5)T2:=11 (8)T5:=11
(31) R:=1000
(32) If T1 >R goto (15)
(10)T7:=T6[T5]
(11)T8:=T7+2
(12)T3[T2]:=T8
(4`) T1 := T1 +10
(7`)T4:=T4+10
(5`)T2:=T2+10
(8`)T5:=T5+10
(14)goto (3)


(1)j:=1 (2)i:=1
(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=10 (7)T4:= T1
(5)T2:=11 (8)T5:=T2
(31) R:=1000
(32) If T1 >R goto (15)
(10)T7:=T6[T2]
(11)T8:=T7+2
(12)T3[T2]:=T8
(4`) T1 := T1 +10
(7`)T4:= T1 +10
(5`)T2:=T2+10
(8`)T5:=T2+10
(14)goto (3)


(6)T3:=a-11 (9)T6:=b-11
(4) T1 :=10 (5)T2:=11
(31) R:=1000
(32) If T1 >R goto (15)
(10)T7:=T6[T2]
(11)T8:=T7+2
(12)T3[T2]:=T8
(4`) T1 := T1 +10
(5`)T2:=T2+10
(14)goto (3)


(6)T3:=a-11 (9)T6:=b-11
(5)T2:=11 (3$_1$) R:=1000
(3$_2$) If T2>R goto (15)
(10)T7:=T6[T2]
(11)T8:=T7+2
(12)T3[T2]:=T8
(5`)T2:=T2+10
(14)goto (3)

9.2 局部优化

一、基本块

- 1、定义
  - 基本块是程序中一段顺序执行的语句序列，只有一个入口，一个出口。执行时只能从其入口进入，从出口退出。
  - 注：一个给定的程序，可以划分为一系列的基本块，优化在各基本块中分别进行。
- 局部优化：局限于基本块范围内的优化称为基本块内的优化。

9.2 局部优化

一、基本块

2、为四元式程序划分基本块

- 算法步骤：
  - 1）求出四元式程序中各基本块的入口语句，即
    - 程序的第一个语句，or
    - 能由条件转移语句和无条件转移语句转移到达的语句，or
    - 紧跟在条件转移语句后面的语句。
    - 注：若条件语句由真转移和假转移组成，则跟在转移语句后面的语句是指假转移语句后面的一条语句。

9.2 局部优化

- 算法步骤:
  - 2）为每个入口语句构造其所属的基本块。
    - 由该入口语句到下一入口语句（不包括该入口语句）、或到一转移语句（包括转移语句）、或到停(halt)语句（包括该语句）之间的语句序列组成。
  - 3）凡是没有纳入到任何一个基本块中的语句，都是程序控制流程所无法到达的语句，即不会被执行到的语句，可将其删除。

- 例如：

```
begin
    read X;
    read Y;
    while (X mod Y<>0) do
      begin
        T:=X mod Y;
        X:=Y;
        Y:=T
      end;
   write Y
end
```

(1) Read X

(2) Read Y

(3) T_1 :=X mod Y

(4) If T_1 <>0 goto (6)

(5)goto (10)

(6)T:=X mod Y

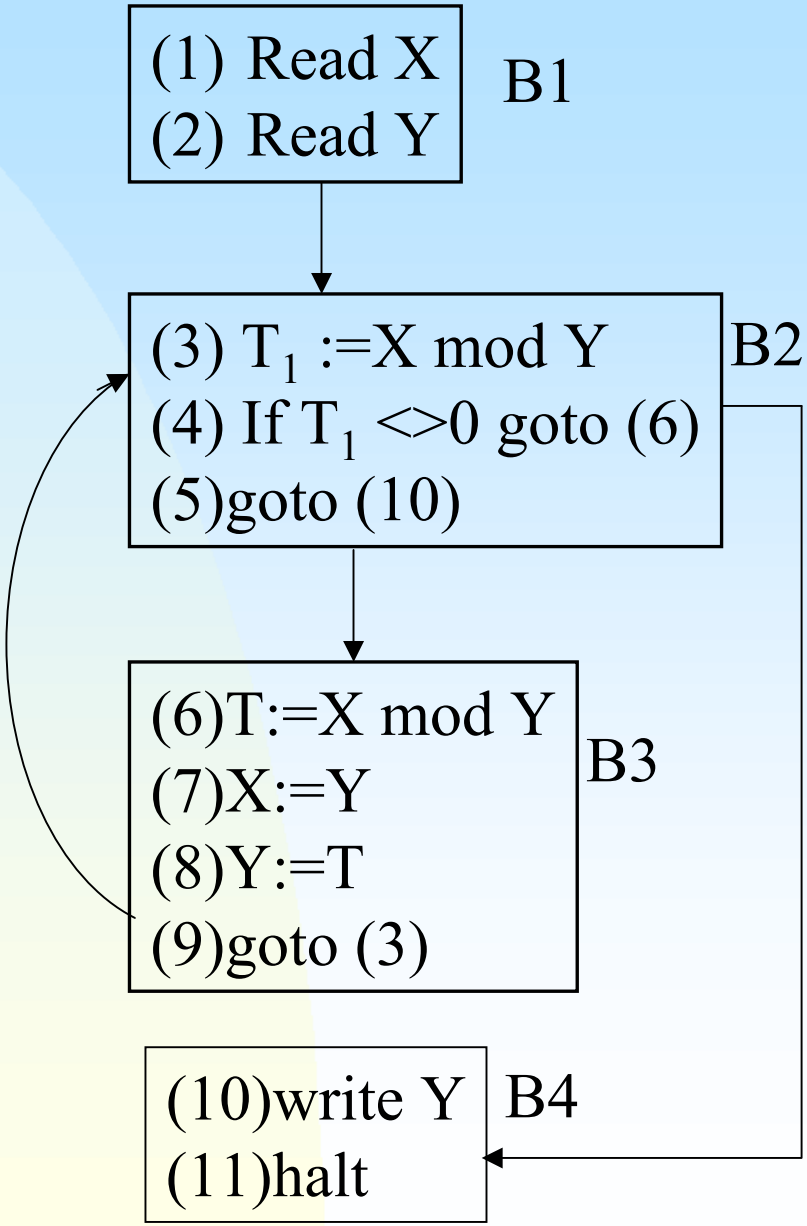
(7)X:=Y

(8)Y:=T

(9)goto (3)

(10)write Y

(11)halt


(1) Read X
(2) Read Y
B1
(3) T1 :=X mod Y
(4) If T1 <>0 goto (6)
(5)goto (10)
B2
(6)T:=X mod Y
(7)X:=Y
(8)Y:=T
(9)goto (3)
B3
(10)write Y
(11)halt
B4

9.2 局部优化

一、基本块

3、块内优化

- 合并已知量
- 删除公共子表达式
- 删除无用赋值
- 注：仅在一个基本块中，是否是无用赋值很难判断。因为：
  - 无用赋值有以下几种：
    - 对某变量A赋值后，在程序中没有引用；
    - 对某变量A赋值后，在引用它之前又重新赋值了；
    - 对某变量A进行自增赋值，且它仅仅被用在自增运算中。
  - 注：对上面第一和第三种情况，应进行全局分析。

9.2 局部优化

二、基本块的DAG表示

1、无环路有向图（Directed Acyclic Graph)定义

- 若结点n_i有弧指向n_j，则n_i是n_j的父结点， n_j是n_i的子结点；
- 若$n_1\ n_2\ \ldots\ldots\ n_k$间存在有向弧$n_1 \rightarrow n_2 \rightarrow \ldots\ldots \rightarrow n_k$,则称$n_1$到$n_k$之间存在一条通路，若$n_1 = n_k$则该通路称为环路；
- 若有向图种任一通路都不是环路，则称该图为无环路有向图。

9.2 局部优化

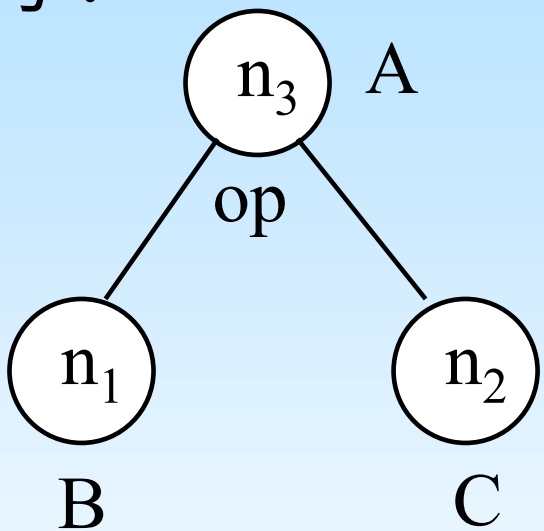
二、基本块的DAG表示

2、作为优化工具的DAG带有标记或附加信息

- 1）图的叶子结点（没有后继的结点）以一标识符（变量名）或常数作为标记。即，该结点代表该变量或常数的值。
- 若叶结点代表变量a的地址，则标记为：addr(a)。
  - 注:通常把叶结点上作为标记的标识符加上下标0，以表示它是该变量的初值。
- 2）图的内部结点（即：有后继的结点）以一运算符作为标记。即，该结点代表应用该运算符对其后继结点所代表的值进行运算的结果。
- 3）图中各个结点上可能附加一个或多个标识符，表示这些标识符具有该结点所代表的值，这简称附标。

9.2 局部优化

- 例如：A:=B op C 即四元式(op,B,C,A)的DAG图为：

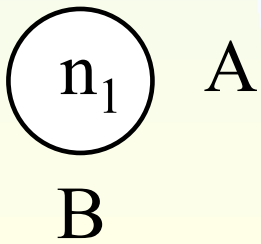


- 注：图中的有向弧都省去箭头。
  - 结点n_i是构造DAG过程中给予各结点的编号;
  - 各结点下面的符号（运算符、变量、常数）是各结点的标记;
  - 各结点右边的标识符是结点的附标。

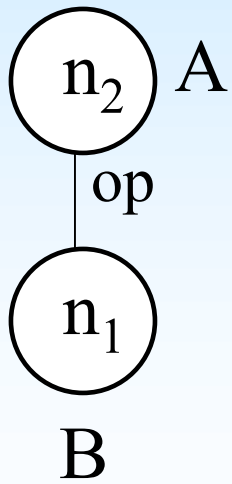
9.2 局部优化

3、DAG图类型

- 一个基本块可以用一个DAG来表示，一般，按四元式对应结点的后继个数分为四种类型：
- 0型：
  - 例如:A:=B即四元式(:=,B,_,A)
  - 其DAG结点为：



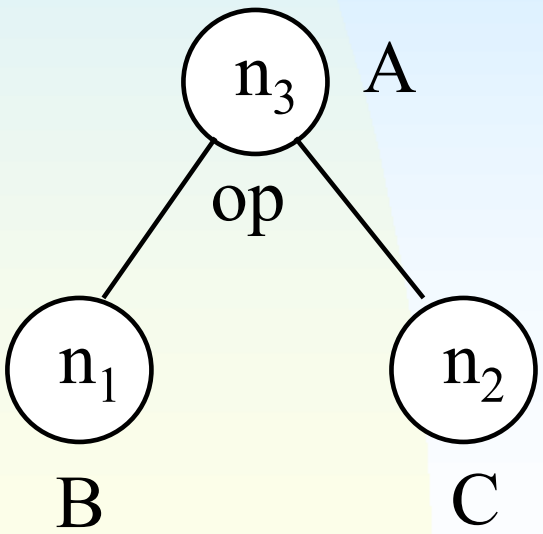
- 1型：
  - 例如：A:=op B即四元式(op,B,_,A)
  - 其DAG结点为：



9.2 局部优化

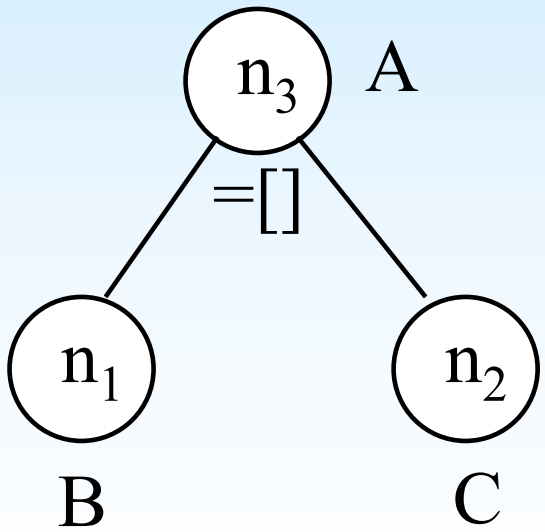
3、DAG图类型

- 2型：
  - 例如：A:=B op C
    即四元式(op,B,C,A)
  - 其DAG结点为：



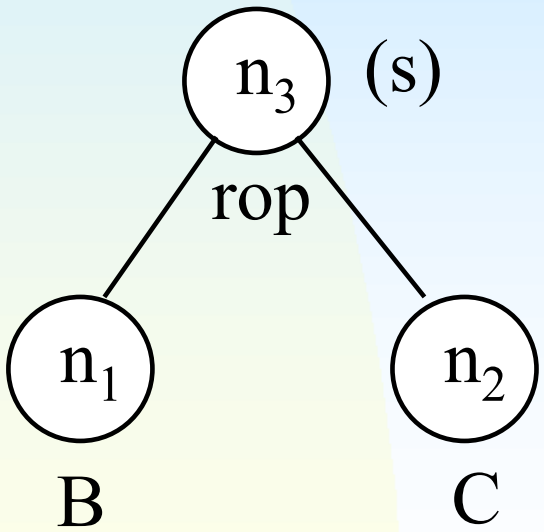
–又如：A:=B[C]即四元式(=[],B[C],_,A)

–其DAG结点为：

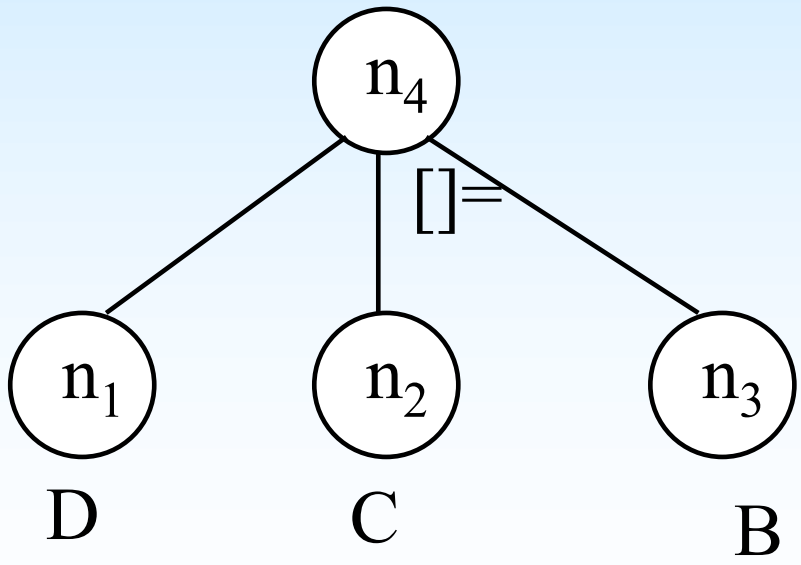


9.2 局部优化

- 2型
  - 又如：if B rop C goto (s)
    即四元式(jrop,B,C,(s))
  - 其DAG结点为：



- 3型
  - 如：D[C]:=B即四元式([]=,B,_,D[C])
  - 其DAG结点为：



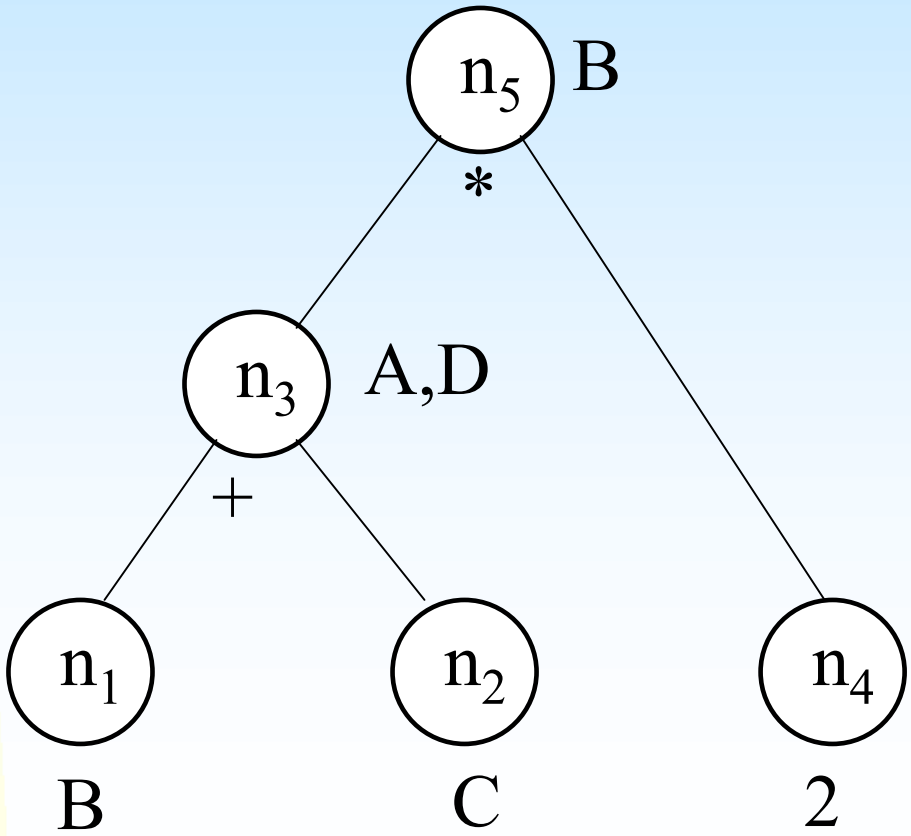
9.2 局部优化

4、DAG结点的数据表示法

- 设DAG各结点信息用一结点表示，其结点包含三项信息：标记、子结点链表、附标
  - 标记：叶结点登记变量或常量的符号表入口地址；内部结点登记操作符；
  - 子结点链表：子结点分长子和次子用链表连接；
  - 附标：填变量的符号表入口地址，多个附标可以拉链。
- 除了结点表之外，为了方便查找，再建一张元表，用来登记四元式除了操作符以外的其他三元的名称和在结点表中的序号。

9.2 局部优化

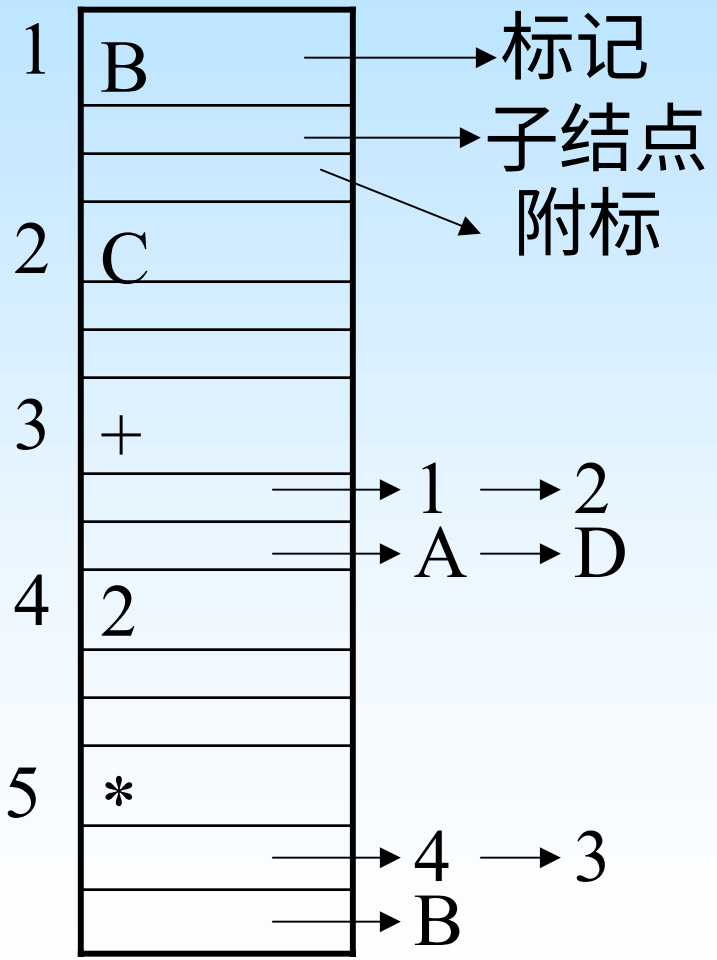
- 例如：四元式序列如下：
- A:=B+C
- B:=2*A
- D:=A
- DAG图如下:



9.2 局部优化

元表

Name	Node
B	5
C	2
A	3
2	4
D	3



9.2 局部优化

5、建立结点表算法（仅考虑0、1、2型）

- 1）将元表和结点表置空
- 2）对基本块内每个四元式(op,B,C,A)进行如下操作：
  - 查元表中有无结点B，有则返回其序号，无则建立该结点并返回序号；
  - 若四元式类型为0型，不做任何操作；
  - 若四元式类型为1型：　（即：A:=OP B)
  - 若四元式类型为2型：　（即：A:=B OP C)
  - 若元表中无结点A，则，把A附加在当前结点上并填元表；否则，从node(A)结点表的附标中删去标记A，把A附加在当前结点上并填元表（注：作为叶结点标记的A不删除）。

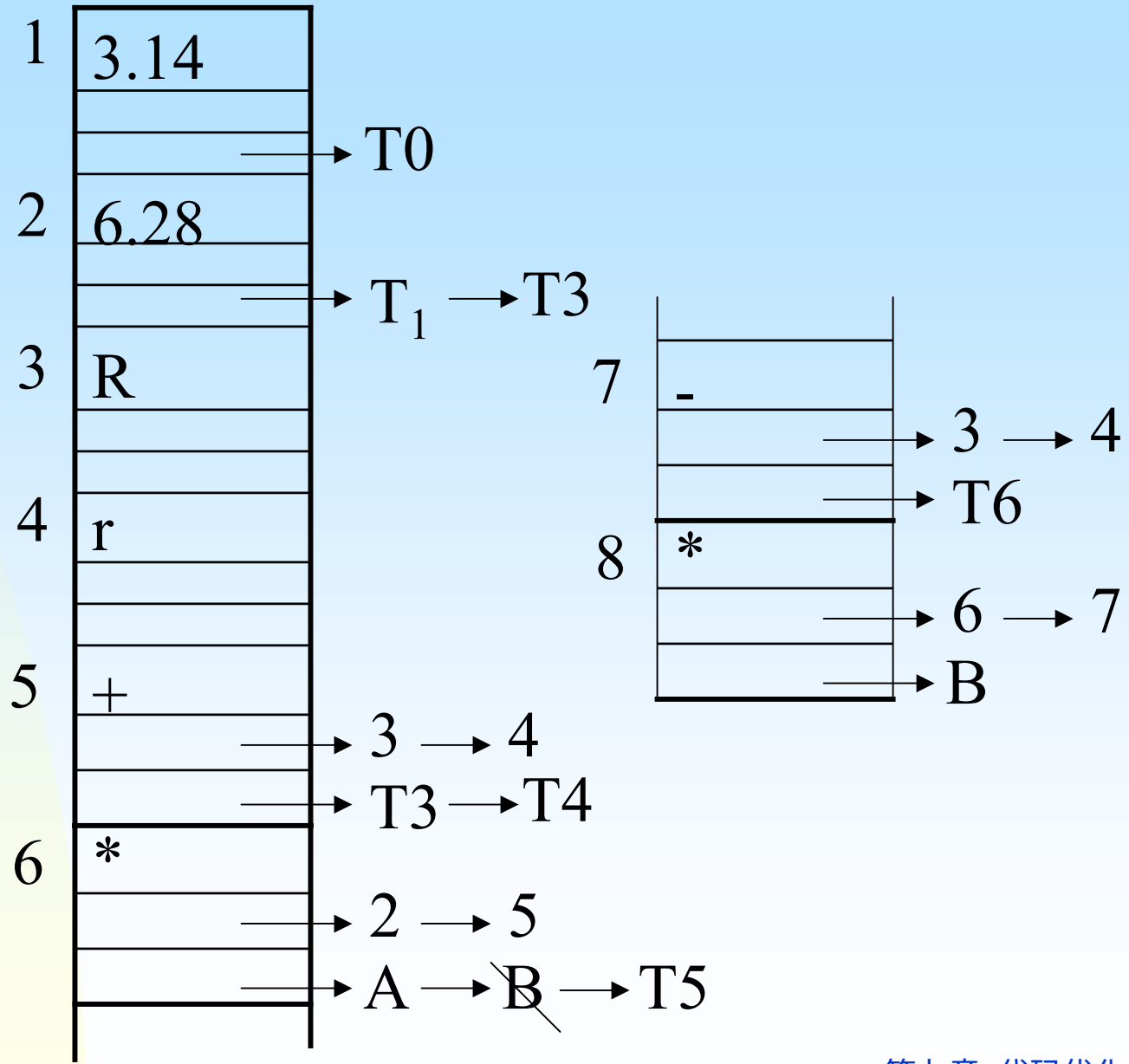
- 1型： A:=OP B
- IF node(B).标记＝常量 THEN /*叶*/
- {执行OP B 得值P /*合并已知量*/ 若node(B)是当前四元式建立的，就删去。
- 查有无node(P)结点，有就返回结点序号，没有就建立并返回其序号。}
- ELSE
- {查结点表是否有一结点，其后继为node(B)，标记为OP，有就返回结点序号，没有就建立并返回其序号。};

- 2型： A:=B OP C
- 查元表中有无结点node(C) ,有就返回结点序号，没有就建立并返回其序号;
- IF (node(B).标记＝常量) AND(node(C).标记＝常量)
- {执行B OP C 得值P /*合并已知量*/ 若node(B), node(C)是当前四元式建立的，就删去;
- 查有无结点node(P) ,有就返回结点序号，没有就建立并返回其序号;}
- ELSE {查结点表是否有一结点，其左后继为node(B)右后继为node(C)，标记为OP，有就返回结点序号，没有就建立并返回其序号。};

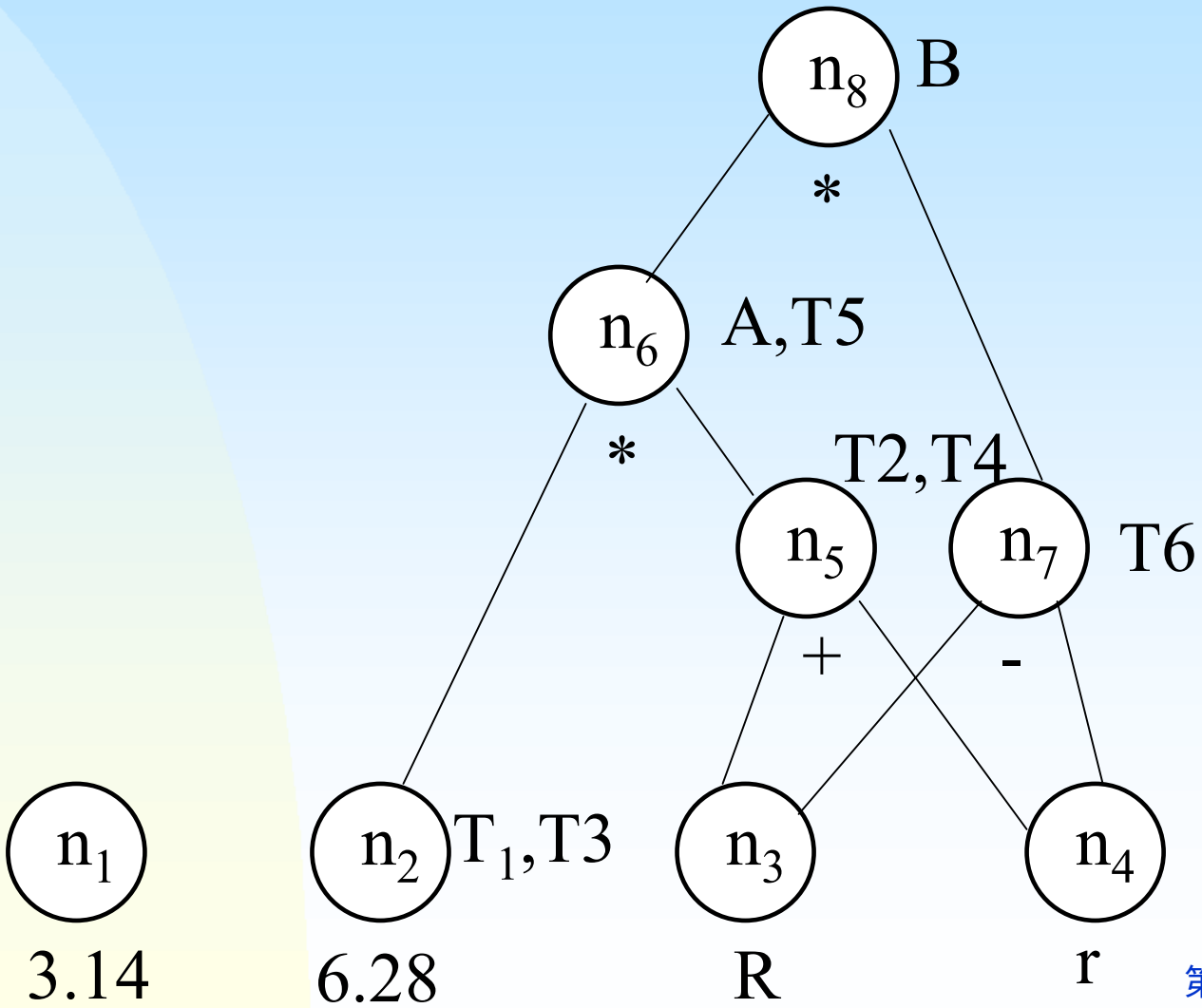
9.2 局部优化

- 注：由于对数组元素赋值的情况特殊，
  所以此算法暂不考虑；在本算法中，建立结点即填写结点表和元表；删除结点就是从结点表和元表中删去相应项。
- 例如：为下列四元式程序段构造DAG.
- (1)T0:=3.14　　(2) T_1 :=2*T0　　(3)T2:=R+r
- (4)A:= T_1 *T2　　(5)B:=A　　(6)T3:=2*T0
- (7)T4:=R+r　　(8)T5:=T3*T4　　(9)T6:=R-r
- (10)B:=T5*T6

Name	Node
3.14	1
T0	1
2 6.28	2 2
T_1	2
R	3
r	4
T2	5
A	6
B	6 8
2 T3	7 2
T4	5
T5	6
T6	7



- 得相应得DAG图为：



9.2 局部优化

三、DAG在基本块优化中作用

1、 DAG图可以做到以下三种优化

- 合并已知量：对任何一个四元式，若参与运算的对象都是编译时已知的常量，则合并已知量，并用合并后算出的常量生成一个叶结点。若参与运算的已知量的叶结点是当前四元式建立的结点，可删除。
- 删除无用赋值：某变量被赋值后，随后又重新赋值，那么算法删除对变量的前一个赋值，即：删除该变量的前一个附标。
- 检查公共子表达式：对具有公共子表达式的所有四元式，算法只产生一个计算该表达式值的内部结点，而把那些被赋值的变量附加到该结点上。

9.2 局部优化

三、DAG在基本块优化中作用

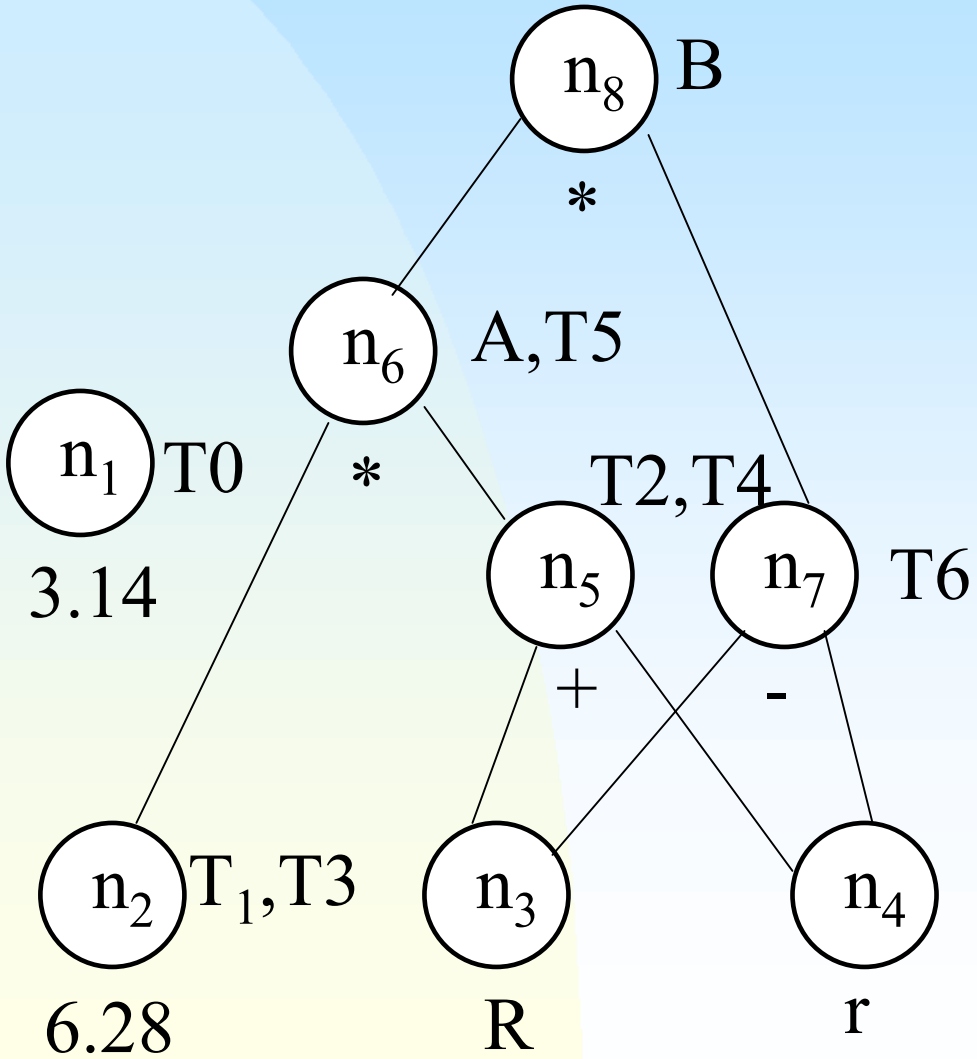
2、利用DAG图还原成四元式序列

- 1)若为叶结点，无附标，则不生成四元式；
- 2)若为叶结点，标记为B,附标为A，则生成A:=B四元式；
- 3)若为中间结点，有附标，则根据其标记op生成：
  - A:=B op C
  - A:=op B
  - A:=B[C]
  - If B rop C goto(s)
- 4)若为中间结点，无附标，则为此结点添加一个此基本块的局部临时附标SA，然后转3)生成相应的四元式；

9.2 局部优化

2、利用DAG图还原成四元式序列

- 5）若结点有多个附标A1,A2,…An，分两种情况考虑：
  - 若结点是叶结点，标记为B，则生成
    A1:=B,A2:=B,…An:=B
  - 若结点是内部结点，则除了第一附标A1外，其他附标生成A2:=A1,A3:=A1,…An:=A1.



- 得四元式序列为：
- 1)T0:=3.14
- 2) T_1 :=6.28
- 3)T3:=6.28
- 4)T2:=R+r
- 5)T4:=T2
- 6)A:=6.28*T2
- 7)T5:=A
- 8)T6:=R-r
- 9)B:=A*T6

9.2 局部优化

- 由DAG图得四元式序列为：
- 1)T0:=3.14
- 2) T_1 :=6.28
- 3)T3:=6.28
- 4)T2:=R+r
- 5)T4:+T2
- 6)A:=6.28*T2
- 7)T5:=A
- 8)T6:=R-r
- 9)B:=A*T6

- 原序列为：
- 1)T0:=3.14
- 2) T_1 :=2*T0
- 3)T3:= 2*T0
- 4)T2:=R+r
- 5)T4:= R+r
- 6)A:= T_1 *T2
- 7)B:=A
- 8)T5:= T3*T4
- 9)T6:=R-r
- 10)B:=T5*T6

9.2 局部优化

- 注：除了应用DAG进行这三种优化外，
  还可以得到其他一些优化信息：
  - 在基本块外被定值并在基本块内被引用得所有标识符，即作为叶结点得那些标识符；
  - 在基本块内被定值且该值能在基本块后面被引用得所有标识符，即DAG各结点上的附标。

3、进一步优化

- 1）根据这些信息，及有关变量在基本块后面被引用的情况，可以进一步删除四元式序列中的无用赋值。

9.2 局部优化

3、进一步优化

- 情况1：若DAG中某结点上附加的标识符，在该基本块后面不会被引用，则不生成对该标识符赋值的四元式。
- 情况2：若某结点上不附有任何标识符或者附加的标识符在基本块后面不会被引用，而且它没有父结点，即基本块内和基本块后都不会引用该结点的值，则不生成计算该结点的值的四元式。
- 情况3：若有两个相邻的四元式A:=C op D和B:=A，其中第一个四元式计算出来的A值仅仅在第二个四元式中引用，则重写四元式时写为：B:=C op D.

9.2 局部优化

- 经如此优化后DAG可重写为：
- 1)S1:=R+r
- 2)A:=6.28*S1
- 3)S2:=R-r
- 4)B:=A*S2
- 注：此处DAG重写是按照原来构造DAG结点的顺序依次进行的。实际上还可以采用其他顺序。

•第一次优化的四元式序列为：

1)T0:=3.14

2) T_1 :=6.28

3)T3:=6.28

4)T2:=R+r

5)T4:=T2

6)A:=6.28*T2

7)T5:=A

8)T6:=R-r

9)B:=A*T6

9.2 局部优化

3、进一步优化

2）DAG结点的启发式排序（P256）

- （1）目的：生成更有效的目标代码。
  - 例如：由DAG生成两种四元式序列

1)S1:=R+r		1`)S1:=R－r
2)A:=6.28*S1	和	2`)S2:=R+r
3)S2:=R-r		3`)A:=6.28*S2
4)B:=A*S2		4`)B:=A*S1

  - 生成两组目标代码：

9.2 局部优化

3、进一步优化

2）DAG结点的启发式排序

（2）启发式思想

- 计算X:=A*B-C*D的右边时，有两种计算顺序
  - 从左向右
  - 从右向左：使得每一计算量总是紧跟在其左运算对象之后计算，使得目标代码较优。

（3）启发式排序算法

- 设DAG中有N各内部结点，T是一维数组，存放排序结果的结点序号。

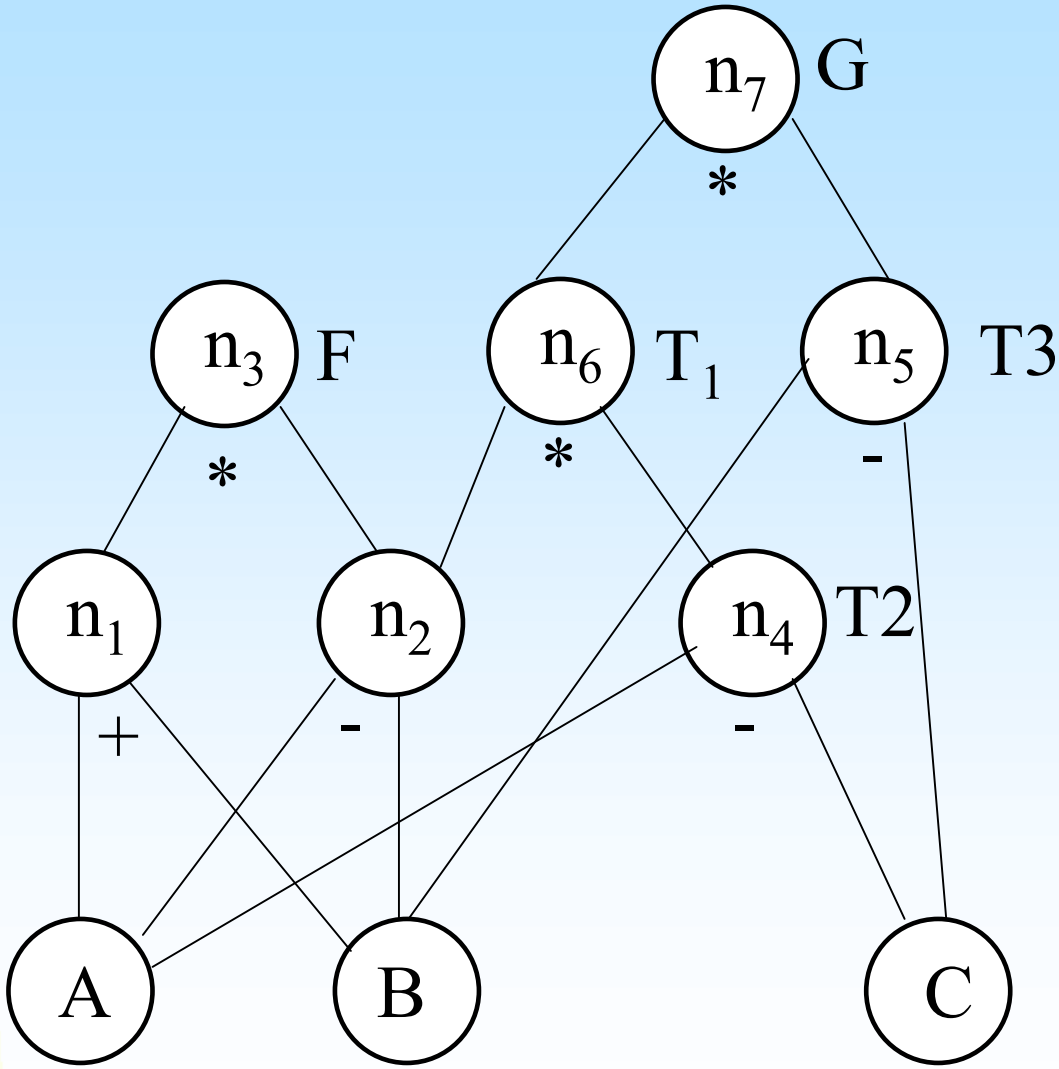
- PROCEDURE Heuristic-Ordering
- { FOR k:=1 TO N DO T[k]:=null;
- i:=N;
- WHILE 存在未列入T的内部结点DO
- {选择一个未列入T的，其父结点均已列入T，或无父结点的结点n;
- T[i]:=n; i:=i-1;
- WHILE n的最左子结点m不为叶结点且其父结点均已列入T中 DO
- {T[i]:=m; i:=i-1; n:=m}
- }
- }
- 则，最后在数组T中的元素序列即为所求的结点顺序。

9.2 局部优化

- 注：在上述的算法中未给叶结点排序，因为：
  - 不需要生成计算叶结点的四元式，如果计算内部结点值时要引用叶结点值，则直接引用它的标记；
  - 若叶结点上附有其它标识符，则需要生成用叶结点的标记对该标识符的赋值指令，但这类指令的次序是任意的。

9.2 局部优化

- 例如：考查下面的四元式序列：
- T_1 :=A+B
- T2:=A-B
- F:= T_1 *T2
- T_1 :=A-B
- T2:=A-C
- T3:=B-C
- T_1 := T_1 *T2
- G:= T_1 *T3

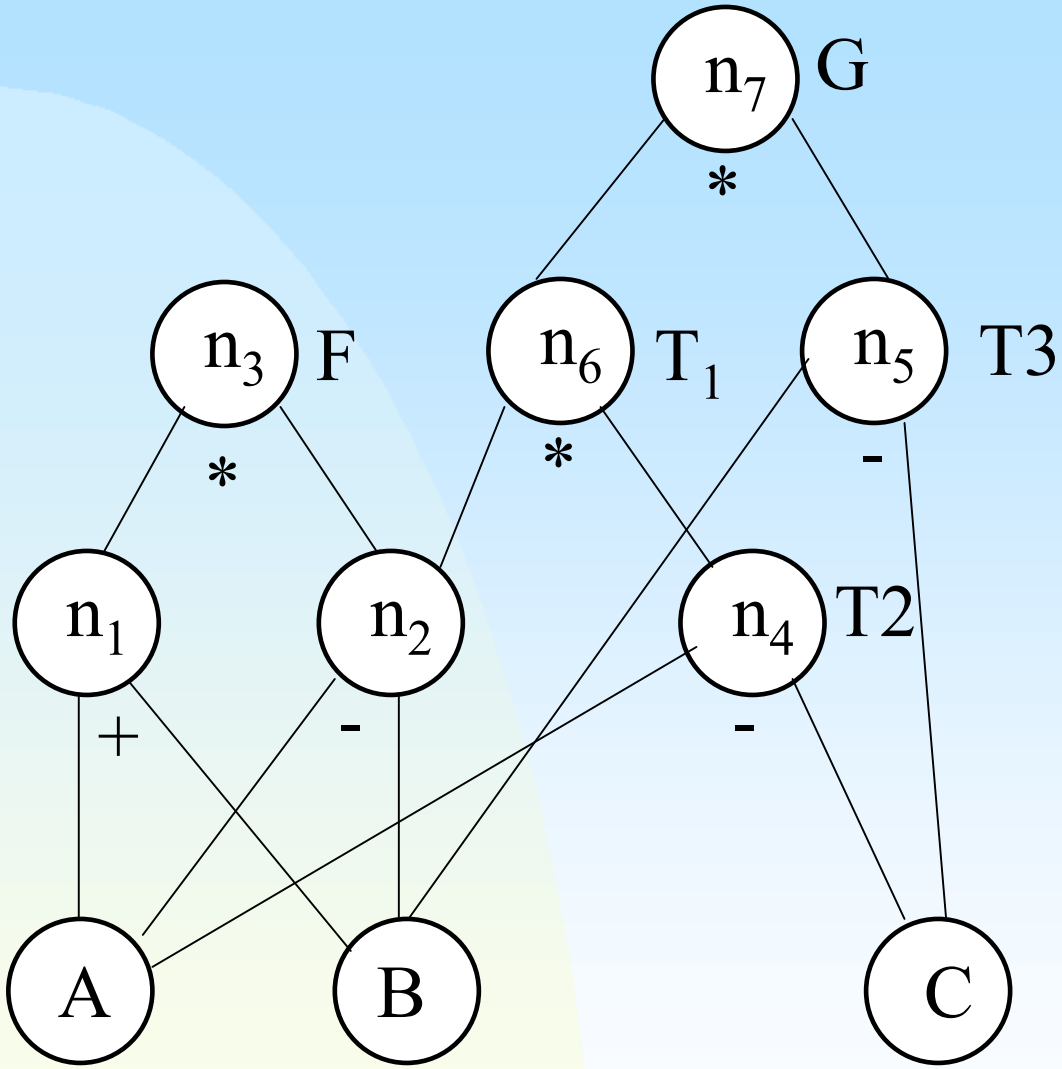


9.2 局部优化

- 对其进行启发式排序得
- I 7 6 5 4 3 2 1
- 1、选n3进行排序： n3 n1 n7 n6 n2 n4 n5

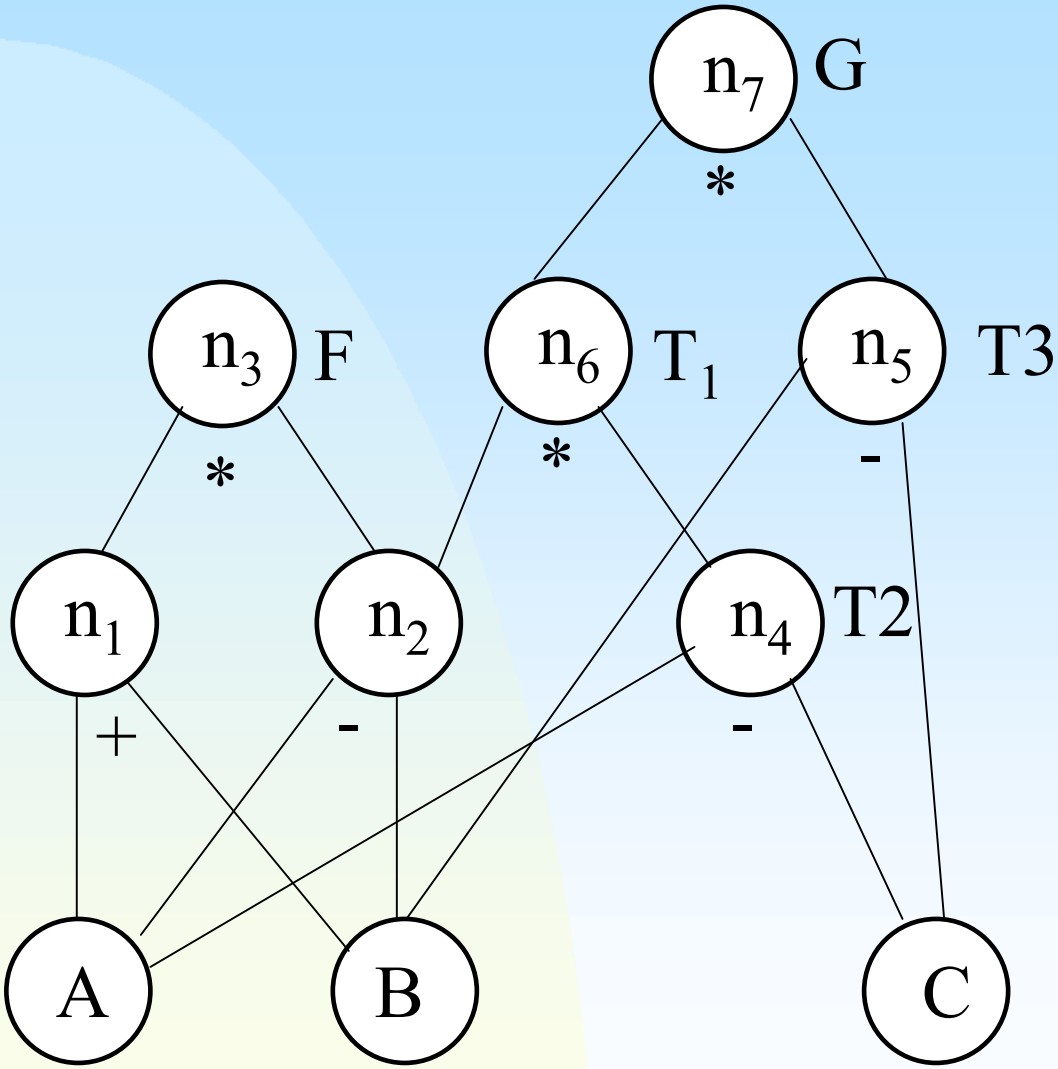
- 2、选n7进行排序： n7 n6 n3 n1 n2 n4 n5

- 箭头表示排好的序。可以按顺序生成目标代码。



- 四元式序列
- T3:=B-C
- T2:= A-C
- S1:=A-B
- T_1 := S1*T2
- G:= T_1 *T3
- S2:=A+B
- F:=S2*S1

n3 n1 n7 n6 n2 n4 n5



n7 n6 n3 n1 n2 n4 n5

- 四元式序列
- T3:=B-C
- T2:= A-C
- S1:=A-B
- S2:=A+B
- F:=S2*S1
- T_1 := S1*T2
- G:= T_1 *T3

9.2 局部优化

- 注：使用启发式排序得到的排序结果不唯一，不同排序产生的目标代码效率也有轻微差异，使用启发式排序可以得到较为满意的结果，但不一定是最佳结果。

9.2 局部优化

四、DAG构造算法的讨论

1、结点或标识符的注销

- 隐式赋值
  - 对数组元素的赋值
  - 间接赋值
  - 两个或更多的变量共享同一单元
- 例如：考查源程序段：
  - (1)X:=A[i]
  - (2)B:=X+2
  - (3)A[j]:=Y
  - (4)Z:=A[i]
  - 在这段程序执行后，X是否等于Z?

9.2 局部优化

四、DAG构造算法的讨论

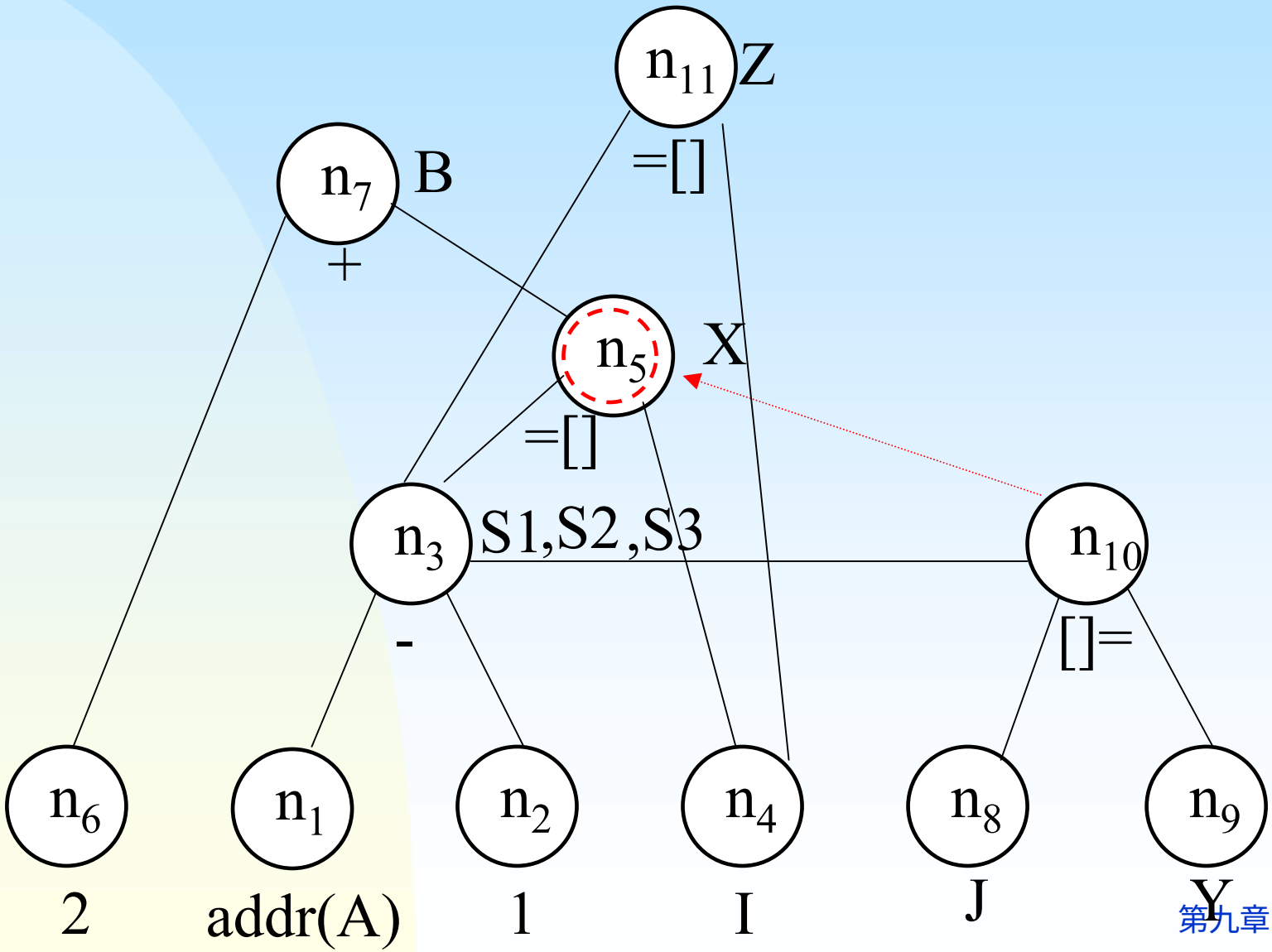
- 对隐式赋值的处理
  - 若出现上述这些隐式赋值，则与这些赋值有关变量的结点将不再被选作公共子表达式使用，不能再在这些结点上增加附标，即：这些结点被注销了。
  - 注：原来在该结点上的附标与对该结点的引用仍然有效。
- 1）对数组元素的赋值
- 构造DAG算法中若遇到对数组元素赋值的四元式，可以把与该数组首地址相关的祖先结点标记为“=[]”的都予以注销（作一个记号），则被注销的结点就再也不能加附标，也不再被引用。而原先它的附标和对它的引用仍有效。

9.2 局部优化

四、DAG构造算法的讨论

- 首先：生成四元式序列如下：
  - (1)S1:=addr(A)-1
  - (2)X:=S1[i]
  - (3)B:=X+2
  - (4)S2:=addr(A)-1
  - (5)S2[j]:=Y
  - (6)S3:=addr(A)-1
  - (7)Z:=S3[i]
- 注：由于（5）式是对数组元素A[j]赋值，它可能会改变数组元素A[i]的值，所以把所有与A数组首地址相关的祖先结点标记为=[]的都予以注销。即（2）式结点被注销。

- 构造过程如下：



9.2 局部优化

四、DAG构造算法的讨论

- 由DAG图还原成四元式序列如下：
  - (1)S1:=addr(A)-1
  - (4)S2:=S1
  - (6)S3:=S1
  - (2)X:=S1[i]
  - (3)B:=2+X
  - (5)S1[j]:=Y
  - (7)Z:=S1[i]

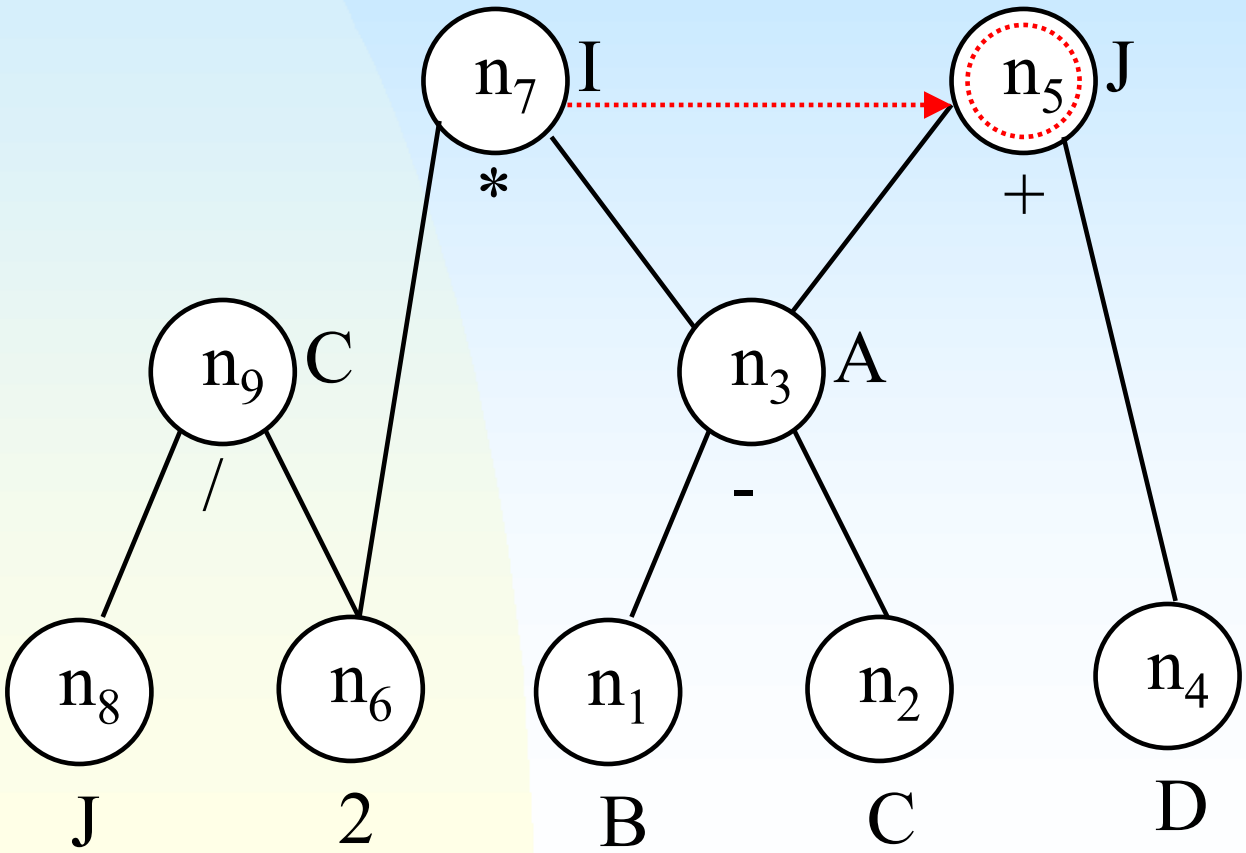
- 原四元式序列如下：
  - (1)S1:=addr(A)-1
  - (2)X:=S1[i]
  - (3)B:=X+2
  - (4)S2:=addr(A)-1
  - (5)S2[j]:=Y
  - (6)S3:=addr(A)-1
  - (7)Z:=S3[i]

四、DAG构造算法的讨论

2）、共享单元的隐式赋值

- 若X,Y是等价变量，对X赋值就会改变Y的值，于是Y不再等于它原来结点的值，所以原来的结点node(Y)要注销，不能再被引用，若要引用则需重新构造一个以它为标记的叶结点。
- 例如：设I,J是等价变量，四元式序列为：
  - A:=B-C
  - J:=A+D
  - I:=A*2
  - C:=J/2

- 其DAG图的构造过程如下



–还原成四元式为

–A:=B-C

–J:=A+D

–I:=A*2

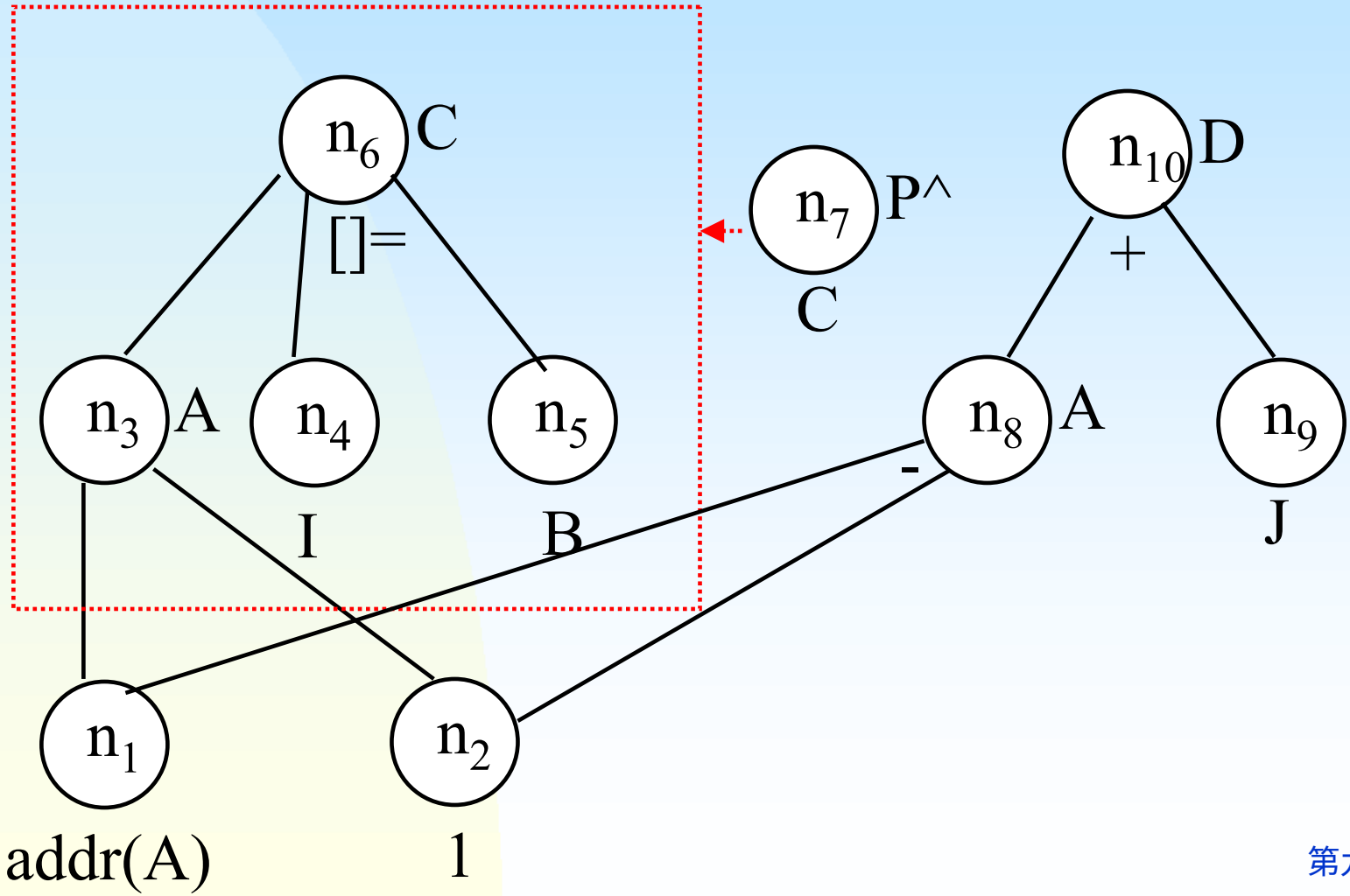
–C:=J/2

四、DAG构造算法的讨论

3）间接赋值

- 若有语句P^:=W，其中P是指向某变量的地址的，若P不知指向哪个变量，则此赋值将注销所有的标识符，包括叶结点上的标识符，只留下常量结点。
- 注：把DAG中所有结点上的标识符都注销，意味着DAG中所有结点都被注销。
- 例如：如下程序语句
- A[I]:=B
- P^:=C
- D:=A[J]

- 其DAG图的构造过程如下



9.2 局部优化
四、DAG构造算法的讨论

2、重写四元式要遵守的顺序

- 存在隐式赋值时，该赋值式是个界线，与该赋值相关的标识符在界线前的赋值和引用与界线后的赋值和引用不能对调。

9.3 控制流程分析和循环查找算法

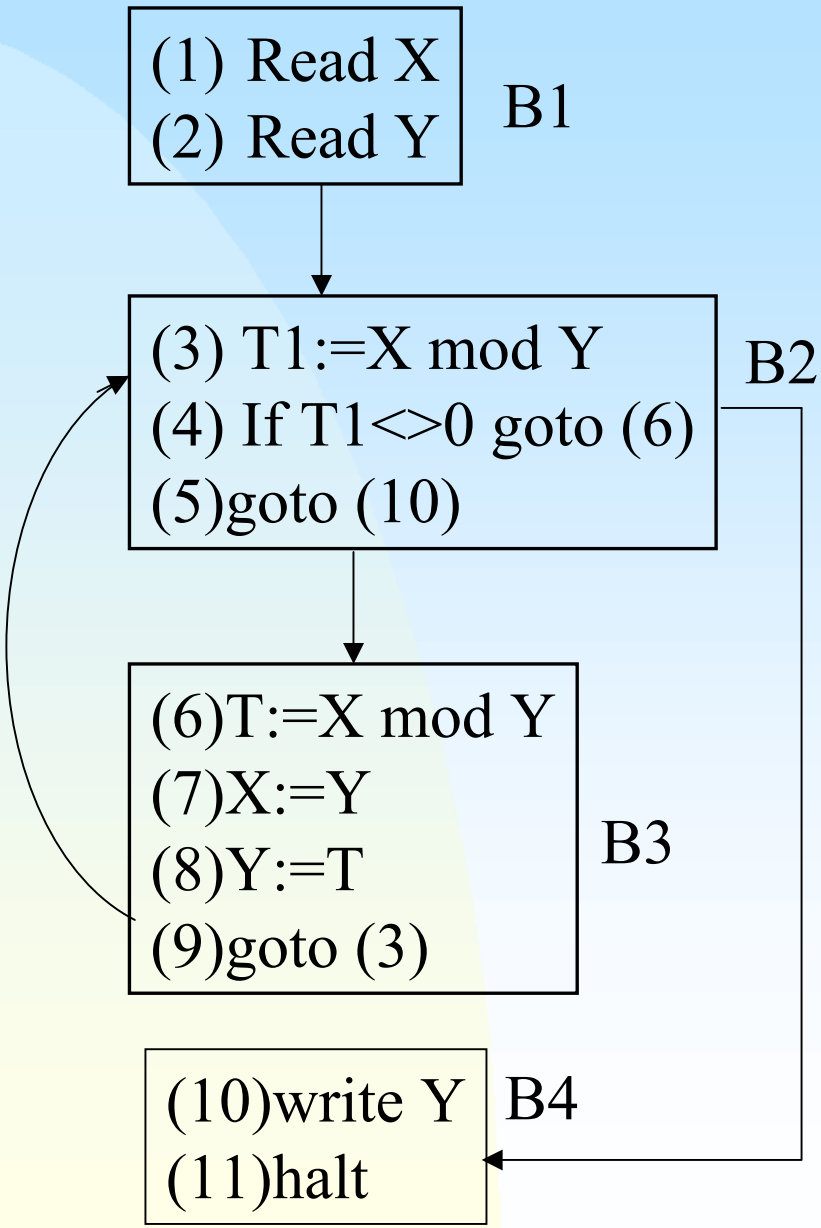
- 找出程序中的循环
  - 循环语句
  - 条件转移和无条件转移构成循环
- 找程序中的循环的方法
  - 程序数据流程分析
  - 程序控制流程分析是数据流程分析的基础

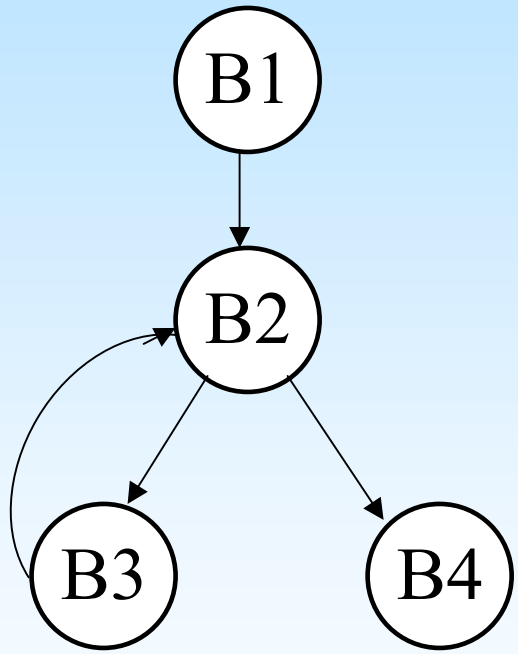
9.3 控制流程分析和循环查找算法

一、程序流图与必经结点集

1、程序流图

- 定义：以基本块作为结点，控制程序流向作为有向弧，画出的图。
- 特点：具有唯一首结点的有向图
  - 首结点：即包含程序第一个语句的基本块。


(1) Read X
(2) Read Y
B1
(3) T1:=X mod Y
(4) If T1<>0 goto (6)
(5)goto (10)
B2
(6)T:=X mod Y
(7)X:=Y
(8)Y:=T
(9)goto (3)
B3
(10)write Y
(11)halt
B4


B1
B2
B3
B4

程序流图

9.3 控制流程分析和循环查找算法

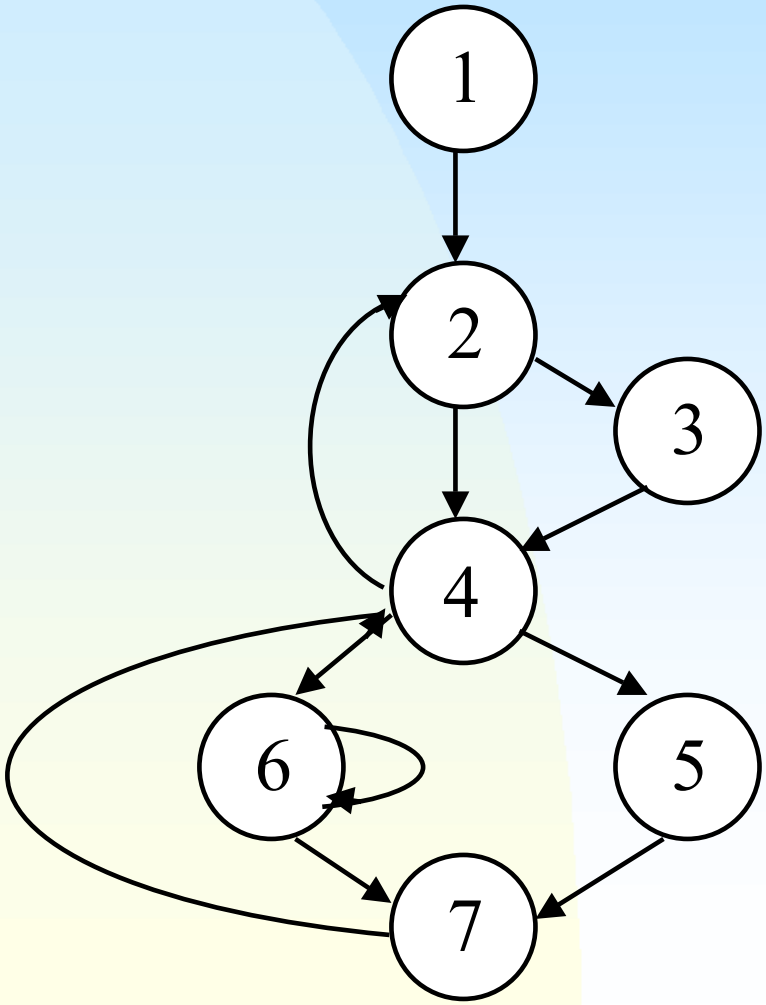
一、程序流图与必经结点集

2、必经结点集

- 定义：若从首结点出发到达nj结点的各条通路都必须经过结点ni，称ni为nj的必经结点，记做ni dom nj.
  - 注：dom是dominate的缩写。
  - nj的全部必经结点的集合称为nj的必经结点集。记做D(nj)。

3、求必经结点集的算法

- 设结点n的父结点是P1,P2,…Pk，则
  - $D(n)=(D(P1)\cap D(P2)\cap \ldots \cap D(Pk))\cup\{n\}$



各结点的必经结点集为：

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}

9.3 控制流程分析和循环查找算法

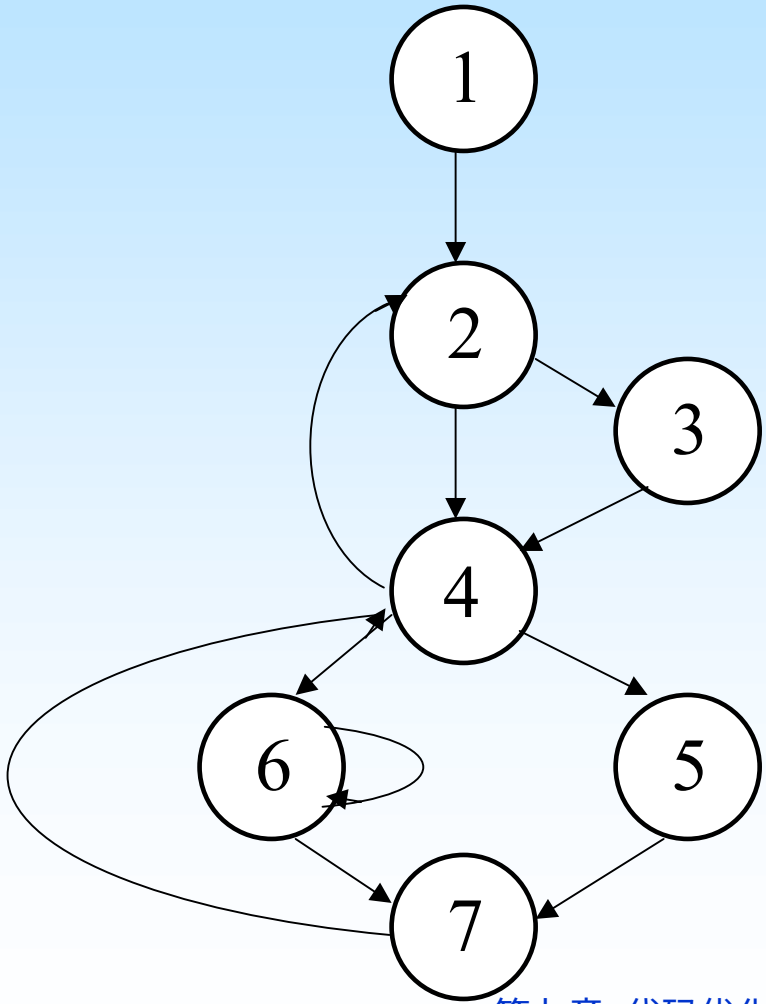
一、程序流图与必经结点集

- 求必经结点集算法：
  - 设全部结点集合为N，首结点为n0。
- PROCEDURE Dominators
- {D(n0)={n0};
- FOR n∈N-{N0} DO D(n):=N;
- CHANGE:=TRUE;
- WHILE CHANGE DO
- {CHANGE:=FALSE;
- FOR n ∈N-{N0} DO
- {NEWD:= $\bigcap_{p \in p(n)} D(P) \cup \{n\}$;
-
- IF D(n)<>NEWD THEN

9.3 控制流程分析和循环查找算法

- CHANGE:=TRUE;
- D(n):=NEWD
- }}}

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



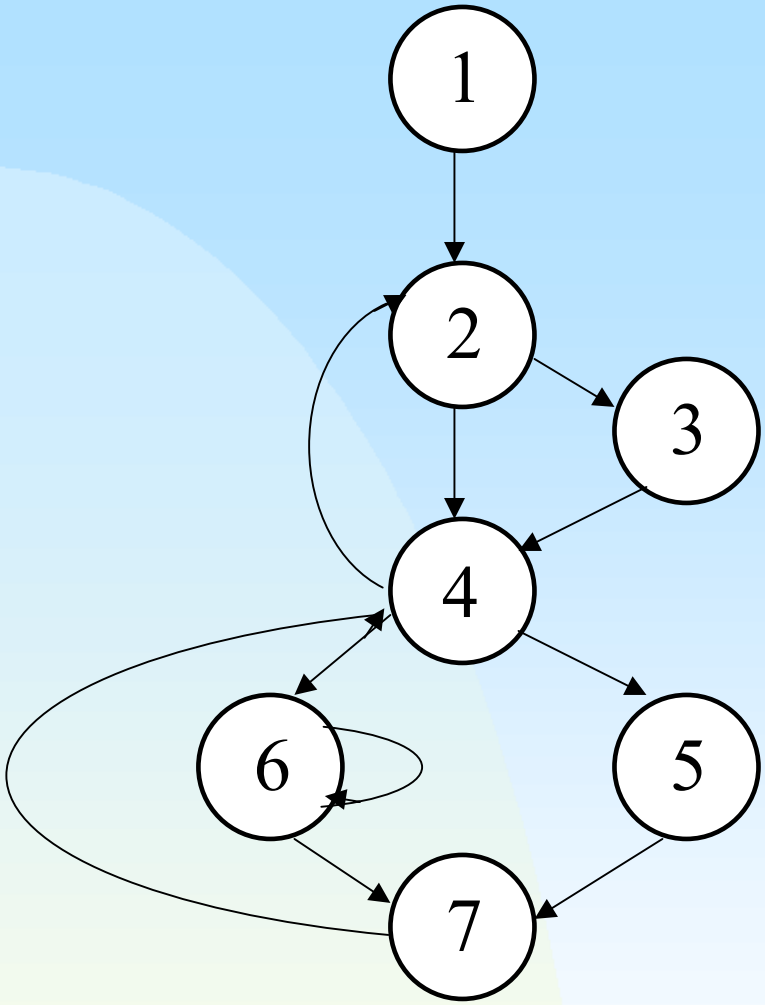
- 算法初始化：
  - $D(1)=\{1\}$
  - $D(2\sim7)=\{1,2,3,4,5,6,7\}$
- 进行第一次迭代：
  - $D(2)= D(1)\cap D(4)\cup\{2\}=\{1,2\}$
  - $D(3)=D(2)\cup\{3\}=\{1,2,3\}$
  - $D(4)=D(2)\cap D(3)\cap D(7)\cup\{4\}=\{1,2,4\}$
  - $D(5)= D(4)\cup\{5\}=\{1,2,4,5\}$
  - $D(6)= D(4)\cup\{6\}=\{1,2,4,6\}$
  - $D(7)= D(5)\cap D(6)\cup\{7\}=\{1,2,4,7\}$
- 进行第二次迭代结果相同，故得到全部必经结点集。

- 注：此算法中没有规定计算结点的顺序，若顺序选的不好，效率可能很低。所以使用算法之前先将流图中各结点排序，此序为深度为主排序。

二、深度为主排序

1、深度为主排序算法

- 对给定流图，从首结点出发沿某条路径尽量前进，直至访问不到新结点时，才退回到其前驱结点；然后再由前驱结点沿另一通路（若有）尽量前进，…重复此过程，直到退回到首结点且再也访问不到新结点为止。
  - 这种尽量往通路深处访问新结点的过程就是按深度为主查找。
  - 查找过程中所经过的结点的逆序就是深度为主次序。



例如：求左图的深度为主次序

注意：没有规定查找路径的话，可以按任意路径查找，得到的排序结果可能不同，随便按哪个结果计算必经结点集均可。

深度为主查找：1 2 4 6 7 6 4 5 4 2 3 2 1

深度为主排序：(7)(6) (5)(4) (3)(2)(1)

9.3 控制流程分析和循环查找算法

三、查找循环算法

1、回边

- 定义：设a→b是流图中一条有向边，若b dom a ，则a→b是流图中一条回边。
- 注：我们可以应用必经结点集来求出流图中的回边。
- 例如：求出下图的所有回边：

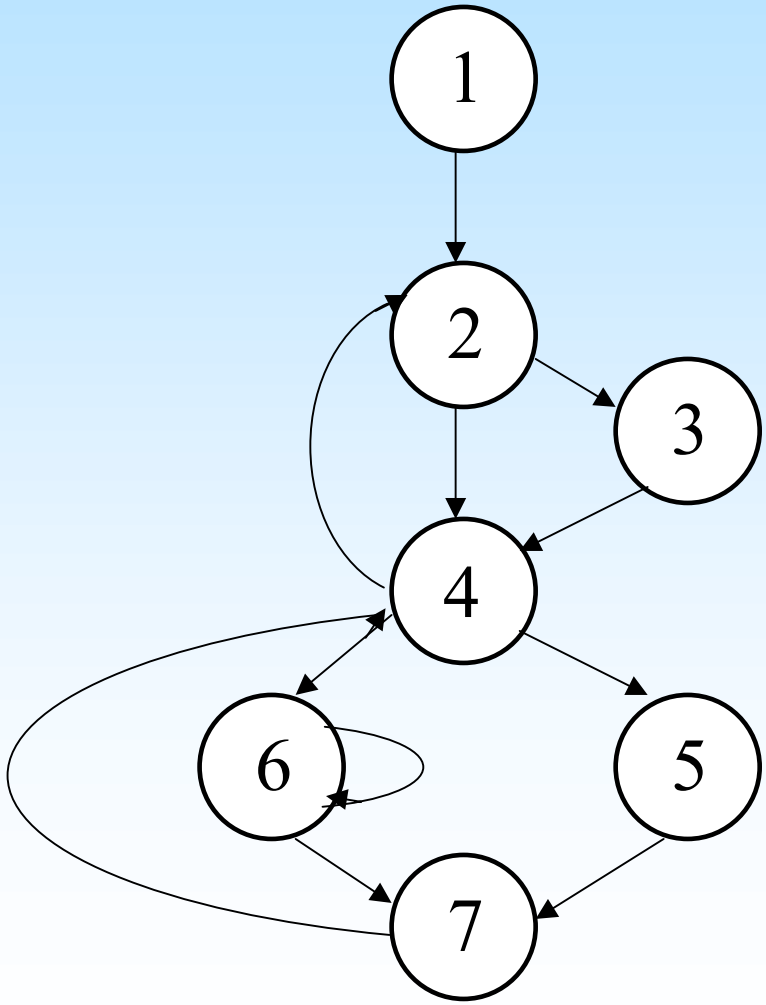
9.3 控制流程分析和循环查找算法

三、查找循环算法

解：首先，求出必经结点集：

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}

- 有6 dom 6 ,4 dom 7 ,2 dom 4
- 故：6→6，7→4，4→2都是流图的回边。



9.3 控制流程分析和循环查找算法

三、查找循环算法

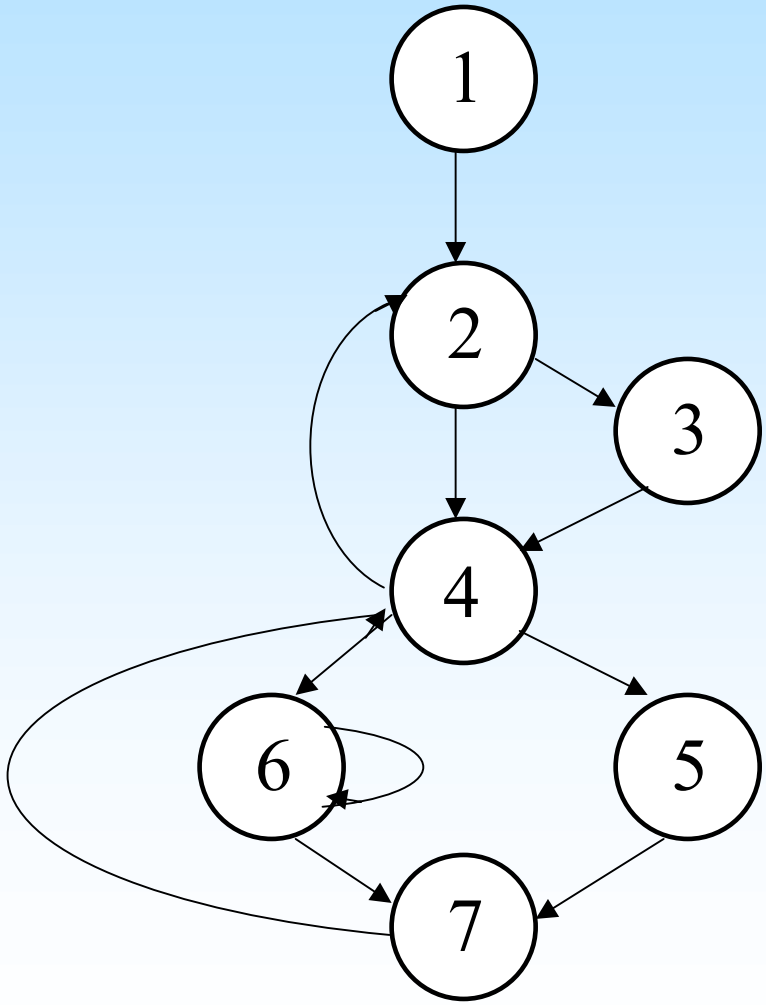
2、利用回边找循环

- 1）每条回边可构成一个循环。
  - 注：有可能两条不同的回边构成相同的循环。
- 2）设n→d是回边，则该回边构成的循环包括下列结点：
  - n,d及不经过d能到达n的所有结点。

9.3 控制流程分析和循环查找算法

三、查找循环算法

- 回边对应的循环结点：
  - 6→6
  - 循环结点：{6}
  - 7→4
  - 循环结点：{7,4,5,6}
  - 4→2
  - 循环结点：{4,2,3,7,5,6}



9.3 控制流程分析和循环查找算法

三、查找循环算法

3、查找循环算法

- 1）找出回边n→d；
- 2）则n,d必定属于n→d回边组成的循环L中，即：L:={n,d};
- 3）若 n<>d且n的父结点n`不在L中，则将它添至L中，即L:=L∪{n`};
- 4）对3）求出的父结点n重复执行3），直至不再有新结点加入L为止。

9.3 控制流程分析和循环查找算法

三、查找循环算法

4、定理

- 对于循环必定满足流图强连通，而且只有唯一入口点。
- 强连通：循环中任意两个结点都有通路，单结点也有一弧指向自身。
  - 它保证了循环内各结点的代码都可反复执行。
- 唯一入口点：要到达循环内任意结点必须经过此入口点。
  - 它保证了循环优化时，可将代码外提到唯一入口点前的前置结点中。

9.4 数据流分析

一、作用

- 为了进行局部优化、循环优化和全局优化，需要分析程序中所有变量的定值和引用之间的关系、基本块出口的活跃变量信息等，这就需要进行数据流分析。

二、方法

  - 变量到达一定值数据流方程和变量引用一定值链
  - 活跃变量数据流方程

9.4 数据流分析

三、到达－定值数据流方程

1、相关概念

- 点：
  - 程序中某一四元式的位置（序号或地址）；
- 定值点：
  - 对某变量赋值或输入值的四元式的位置；
- 引用点：
  - 引用某变量的四元式的位置；
- 变量A的到达与定值：
  - 若d点是变量A的一个定值点，u点是A的一个引用点，存在一条从d到u的通路，并在此通路上没有对A的其他定值点，则称d点对A的定值能达到u点。

9.4 数据流分析

三、到达一定值数据流方程

1、相关概念

- 引用一定值链：
  - 假定在程序中某点u引用了变量A，则把能到达u的A的所有定值点的全体，称为A在引用点u的引用一定值链（ud链）。

9.4 数据流分析

三、到达－定值数据流方程

2、到达－定值数据流方程求解

1)基本量

- IN[B]:能到达基本块B入口点的各个变量的定值点集合。
- OUT[B]:能到达基本块B出口点的各变量全部定值点集合。
- GEN[B]:基本块B中定值并能到达B出口点的所有定值点集合。
- KILL[B]：因B中定值而注销所有与它相关变量的定值点集合。

9.4 数据流分析

三、到达－定值数据流方程

2、到达－定值数据流方程求解

2）基本方程

- OUT[B]=IN[B]-KILL[B]∪GEN[B]
- IN[B]= $\bigcup_{P \in P[B]} OUT[P]$
- 注：P[B]代表B的所有前驱基本块集合。
- GEN[B]：基本块B中定值并能到达B出口点的所有定值点集合，即是基本块内DAG图上的附标，所以它是已知量；
- KILL[B]：因B中定值而注销所有与它相关变量的定值点集合。即，若Ai在B中定值，则所有以前的Ai定值点都注销，故它也是已知量；

9.4 数据流分析

一、到达－定值数据流方程

2、到达－定值数据流方程求解

注：所以上式是关于变量IN[B]和OUT[B]的线性联立方程组，即到达－定值数据流方程。

- 流图中若有n个基本块，则有2n个方程。

3）方程求解算法

```
PROCEDURE Reading-Definitions
    {  FOR i:=1 TO n DO
        { IN[Bi]:= Φ；OUT[Bi]:=GEN[Bi]  }
      CHANGE:=TRUE;
      WHILE CHANGE DO
```

{CHANGE:=FALSE;
FOR i:=1 TO n DO
{NEWIN:= $\bigcup_{P \in P[Bi]}$ OUT[P]

IF NEWIN<>IN[Bi] THEN
{CHANGE:=TRUE;
IN[Bi]:=NEWIN;
OUT[Bi]:=IN[Bi]-KILL[Bi]∪GEN[Bi]
}
}}}

注：对这n个基本块采取深度为主排序的正序计算，其迭代的收敛比较快。

9.4 数据流分析

四、引用－定值链（ud链）

1、定义：

- 到达基本块内某点u的变量A的定值点集合称变量A在u点的ud链。

2、构成规则：

- 1）在基本块B中，变量A在引用点u前有定值点d，并且d点的定值能到达u，那么A在u点的ud链＝{d};
- 2）若在基本块B中，在u点之前没有对A的定值点，则变量A在u点的ud链＝{IN[B]中有关A的集合}。
- 注：这里，链是指将各集合中各定值点连在一起构成链，便于检索。

9.4 数据流分析
五、活跃变量及数据流方程

1、活跃变量定义

- 程序中某变量A和某点P，存在一条从P开始的路径，在该路径上A在定值前被引用或仅有引用，则A在P点是活跃的变量。
- 注：对基本块而言，若能求得哪些变量在出口点是活跃的，哪些不是活跃的，则它将为块内优化提供有用信息，如，将DAG中那些基本块出口为不活跃的变量的附标删除；在循环优化中也为循环代码外提提供有用信息等。

9.4 数据流分析

五、活跃变量及数据流方程

2、活跃变量数据流方程

1)基本量

- L·IN[B]：
  - 块B入口点活跃变量集合；
- L·OUT[B]：
  - 块B出口点活跃变量集合；
- L·USE[B]：
  - 块B中引用的，但引用前未曾在B中定值的变量集合；即：DAG中叶结点的标识符；
- L·DEF[B]：
  - 块B中定值的，但在定值前未曾在B中引用过的变量集合；

9.4 数据流分析

三、活跃变量及数据流方程

2、活跃变量数据流方程

2）基本方程

$$L\cdot IN[B] = L\cdot OUT[B] - L\cdot DEF[B] \cup L\cdot USE[B]$$

$$L\cdot OUT[B] = \bigcup_{S \in S[B]} L.IN[S]$$

（S[B]代表B的后继基本块集合）

- L·DEF[B]和L·USE[B]可以在各基本块中直接求得，所以这个方程组是关于变量L·IN[B]和L·OUT[B]的线性联立方程组。
- 流图中若有n个基本块，则有2n个方程。

3）求解活跃变量数据流方程的算法

```
PROCEDURE Live-Variables
    {   FOR i:=1 TO n DO
          IN[Bi]:=空集；
        CHANGE:=TRUE;
        WHILE CHANGE DO
          {CHANGE:=FALSE;
            FOR i:=n TO 1 BY -1 DO
              {OUT[Bi]:= ∪ IN[S];
                          S∈S[Bi]
                NEWIN:= OUT[Bi]-DEF[Bi]∪USE[Bi];
                IF IN[Bi]<>NEWIN THEN
                  {CHANGE:=TRUE;
                   IN[Bi]:=NEWIN;
                  }}}}
```

9.5 循环优化

一、代码外提

- 1、循环不变量定义：形如(s)A:=B op C四元式若
  - B,C为常量，or
  - 到达(s)点得B,C定值点都在循环外（可通过查ud链得知）；
  - 到达(s)点得B,C定值点虽在循环内，但只有一个定值点且已被标为循环不变运算;
- 则(s)为循环不变运算，A,B,C为循环不变量。
- 注：循环不变运算可以外提到循环外，以便提高目标代码的运行速度。

9.5 循环优化

一、代码外提

2、前置结点

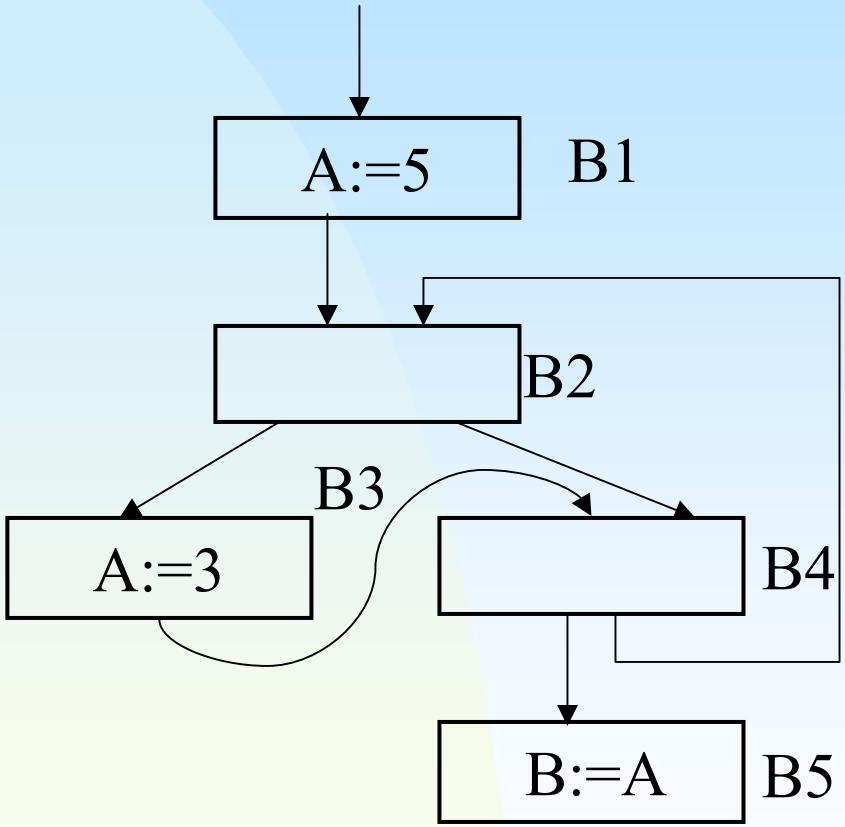
- 实行代码外提时，在循环入口结点前面建立一个新结点（基本块），即前置结点。
- 特点：以循环入口结点为其唯一后继。
- 前置结点是唯一的，循环中外提的代码将全部放到前置结点中。

3、循环不变运算外提条件：

- 1）该不变运算所在结点（基本块）必须是循环出口结点的必经结点或者该不变运算所定值的变量在循环出口之后是不活跃的；

9.5 循环优化

- 例如：B3中的循环不变运算，是否可以外提？

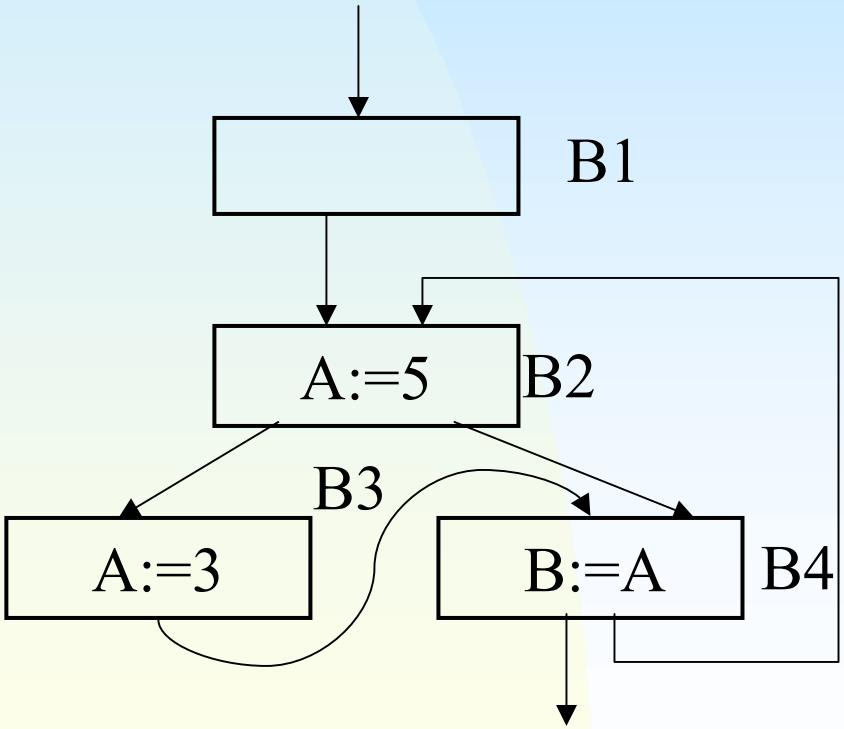


假定程序两次执行途径分别为：
B1,B2,B4,B5,出口后B=5
B1,B2,B3,B4,B5,出口后B＝3；
若A:=3外提，则上述两条途径执行结果均为B=3。
注意：其原因就是B3不是B4的必经结点，而且A在出口是活跃的。

9.5 循环优化

3、循环不变运算外提条件：

- 2）循环内不变运算所定值的变量只有唯一一个定值点。
- 例如：下图中，B2中的循环不变运算能否外提？

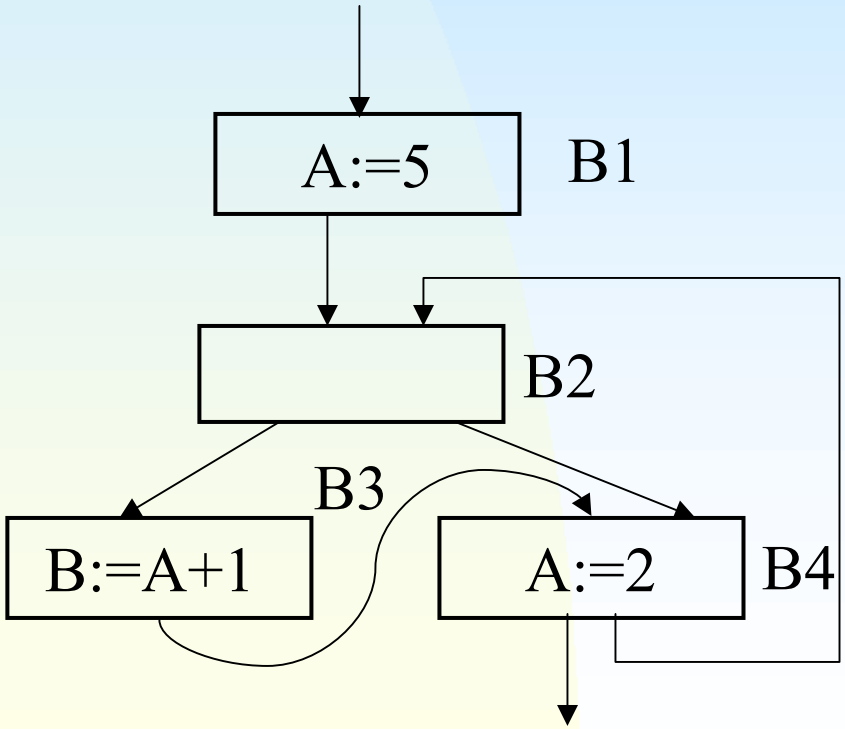


假定程序外提前执行途径为：
B1,B2,B3,B4,B2,B4,出口
后B=5;
外提后执行相同的执行途径出口后B＝3；
注意：A有两个定值点。

9.5 循环优化

3、循环不变运算外提条件：

- 3）外提循环不变运算(s)A:=B op C时循环内所有A的引用点必须而且仅是(s)所能到达的。
- 例如：下图中，B4中的循环不变运算能否外提？



假定程序外提前执行途径为：
B1,B2,B3,出口后B=6;
外提后执行相同的执行途径出口后B＝3；
注意：原因是B3当中引用的A值不仅B4中A的定值可到达，B1中A的定值也能到达。

9.5 循环优化

4、代码外提算法思想

寻找符合循环不变量外提条件的不变运算，进行外提。并外提到前置结点。

9.5 循环优化

二、强度减弱与归纳变量删除

1、归纳变量定义：

- 循环中变量I只有唯一的形如(s)I:=I+C的赋值，其中C为循环不变量，则称I为循环中基本归纳变量。
- 若变量J与基本归纳变量I可归纳为线性关系J:=C1*I+C2,其中C1,C2为循环不变量，则称J是和I同族的归纳变量。
- 注：循环控制变量是基本归纳变量的特例。

9.5 循环优化

二、强度减弱与归纳变量删除

2、强度减弱与归纳变量删除算法

a)利用循环不变运算信息，找出循环中基本归纳变量X；

b)找出所有归纳变量A，并指出A与X的线性关系；

c)强度减弱，将归纳变量算式化为自增赋值算式，并按原式计算初值，将自增赋值算式置于原基本归纳变量自增赋值算式之后；

d)变换循环控制变量；

e)删除对X的自增赋值四元式。

小结

a)程序基本块的划分
　寻找入口语句
b)基本块的DAG表示
c)根据DAG进行块内优化
　合并已知量、删除公共子表达式、删除无用赋值、重组程序语句顺序(一种启发式排序)
d)程序流图与必经结点集及循环查找
e)到达-定值数据流方程及其含义解释
f)活跃变量数据流方程及其含义解释
g)循环不变量代码外提