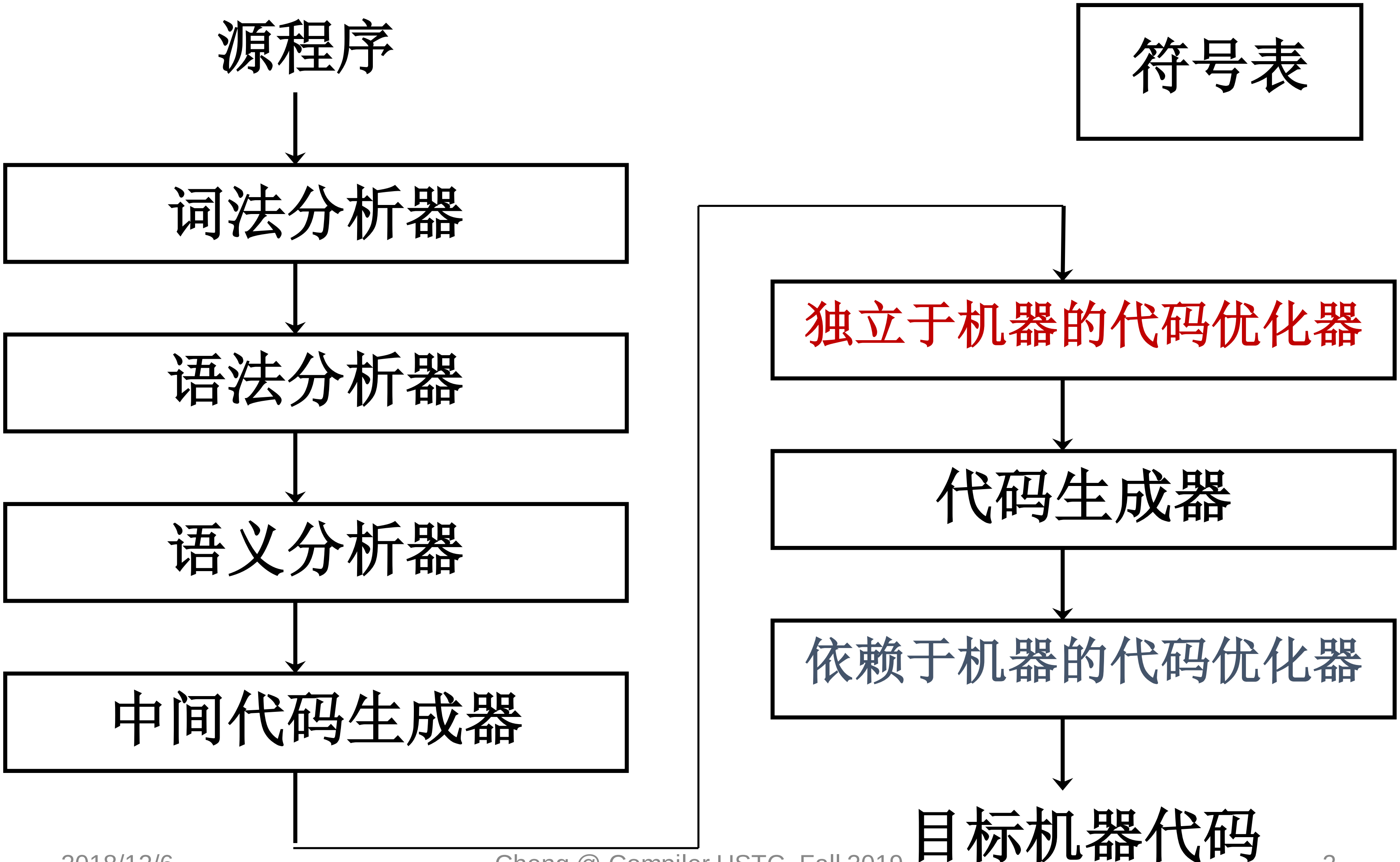
《编译原理与技术》
独立于机器的优化 II

计算机科学与技术学院
李 诚

25/11/2019

独立于机器的优化


源程序
词法分析器
语法分析器
语义分析器
中间代码生成器
符号表
独立于机器的代码优化器
代码生成器
依赖于机器的代码优化器
目标机器代码

优化的实现方式

- 局部视角-基本块的优化
  - DAG表示
- 全局视角-跨基本块的优化
  - 数据流分析

❑基本块DAG的构造方式

- 每个变量有一个对应的DAG结点表示其初值
- 每条语句s都对应一个内部结点N
  - 结点N的标号是s中的运算符
  - 有一组变量被关联到N，表示s是在此基本块中最晚对这些变量定值的语句
  - N的子结点是基本块中在s之前，最后一个对s所使用的某个运算分量进行定值的语句对应的结点。如果某个运算分量在基本块中在s之前没有被定值，则这个分量对应的子结点就是其初始值对应的结点，用下标0区分

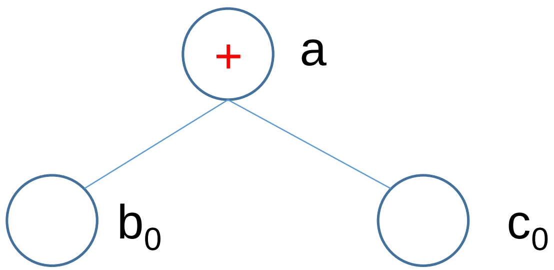
考虑如下基本块：

a = b + c

b = a – d

c = b + c

d = a - d



❑**每个变量有一个对应的DAG结点表示其初值**

❑**每条语句s都对应一个内部结点N**

❖结点N的标号是s中的运算符

❖有一组变量被关联到N，表示s是在此基本块中最晚对这些变量定值的语句

❖N的子结点是基本块中在s之前，最后一个对s所使用的某个运算分量进行定值的语句对应的结点。如果某个运算分量在基本块中在s之前没有被定值，则这个分量对应的子结点就是其初始值对应的结点，用下标0区分

基本块的DAG(有向无环图)表示

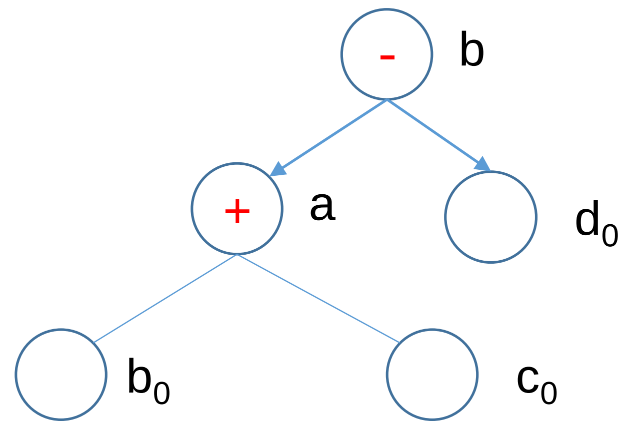
考虑如下基本块：

a = b + c

b = a – d

c = b + c

d = a - d



- **每个变量有一个对应的DAG结点表示其初值**
- **每条语句s都对应一个内部结点N**
  - 结点N的标号是s中的运算符
  - 有一组变量被关联到N，表示s是在此基本块中最晚对这些变量定值的语句
  - N的子结点是基本块中在s之前，最后一个对s所使用的某个运算分量进行定值的语句对应的结点。如果某个运算分量在基本块中在s之前没有被定值，则这个分量对应的子结点就是其初始值对应的结点，用下标0区分

基本块的DAG(有向无环图)表示

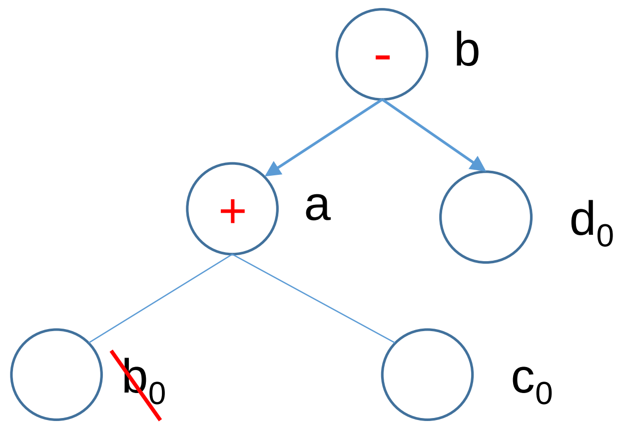
考虑如下基本块：

a = b + c

b = a – d

c = b + c

d = a - d



- ❑**每个变量有一个对应的DAG结点表示其初值**
- ❑**每条语句s都对应一个内部结点N**
  - ❖结点N的标号是s中的运算符
  - ❖有一 的语句
  - ❖N的 量进行
    定值 有被定
    值，则这个分量对应的子结点就是其初始值对应的结点，用下标0区分

注1：在为语句x= y+z构造结点N的时候，如果x已经被关联到某结点M上，那么需要从M的关联变量中删除变量x

基本块的DAG(有向无环图)表示

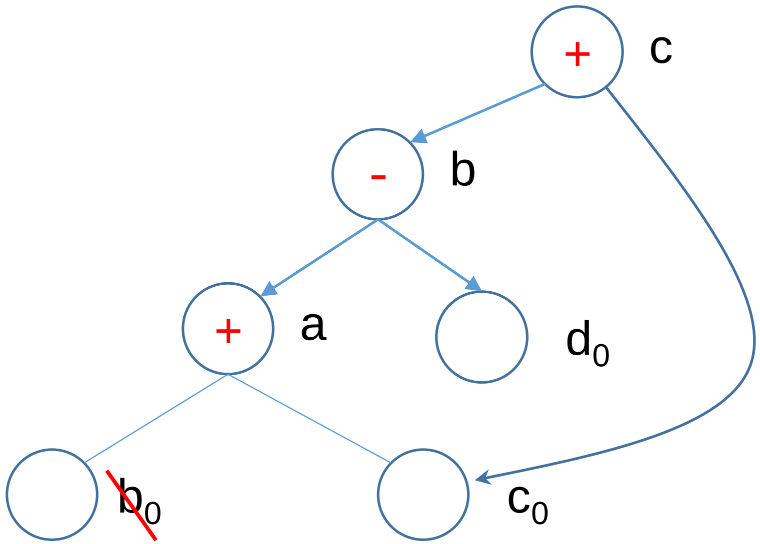
考虑如下基本块：

a = b + c

b = a – d

c = b + c

d = a - d



- ❑**每个变量有一个对应的DAG结点表示其初值**
- ❑**每条语句s都对应一个内部结点N**
  - ❖结点N的标号是s中的运算符
  - ❖有一组变量被关联到N，表示s是在此基本块中最晚对这些变量定值的语句
  - ❖N的子结点是基本块中在s之前，最后一个对s所使用的某个运算分量进行定值的语句对应的结点。如果某个运算分量在基本块中在s之前没有被定值，则这个分量对应的子结点就是其初始值对应的结点，用下标0区分

基本块的DAG(有向无环图)表示

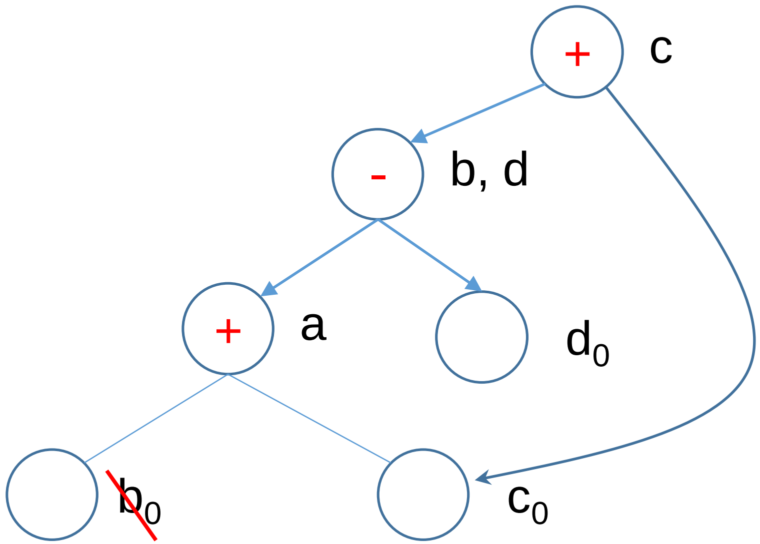
考虑如下基本块：

a = b + c

b = a – d

c = b + c

d = a - d



- ❑每个变量有一个对应的DAG结点表示**其初值**
- ❑每条语句s都对应一个**内部结点N**
  - ❖结点N的**标号**是s中的运算符
  - ❖有 … 语句
  - ❖N的 … 行定 … 定值，

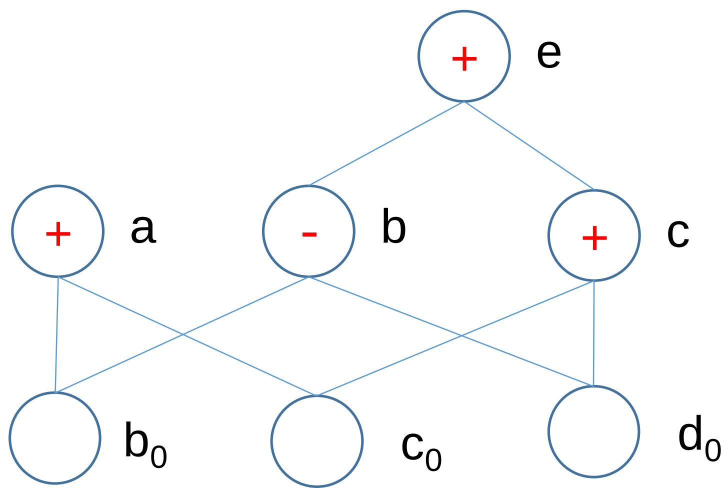
注2：对于x=y+z语句，可对y+z进行值编码，并在该基本快DAG中查询是否已经存在一个结点表示y+z，如果有，就不需要在DAG中加入新的结点，而是在已存在的结点的关联变量中增加x

DAG与删除死代码

- 活跃变量是指其值可能会在以后被使用的变量
- 在DAG上删除死代码，可进行如下操作
  - 1.删除所有没有关联活跃变量的根结点
  - 2.重复1操作

考虑如下基本块：

a = b + c
b = b – d
c = c + d
e = b + c



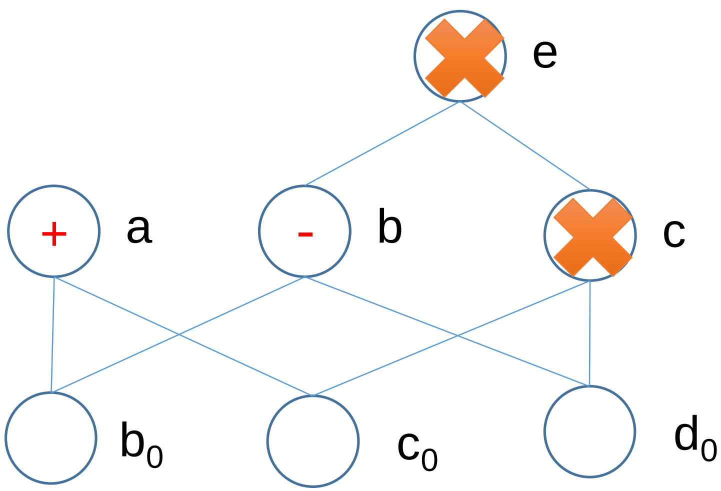
假设a和b是活跃变量，但c和e不是

DAG与删除死代码

- 活跃变量是指其值可能会在以后被使用的变量
- 在DAG上删除死代码，可进行如下操作
  - 1.删除所有没有关联活跃变量的根结点
  - 2.重复1操作

考虑如下基本块：

a = b + c
b = b – d
c = c + d
e = b + c



假设a和b是活跃变量，但c和e不是

数组引用的表示

中国科学技术大学 University of Science and Technology of China

考虑如下基本块：

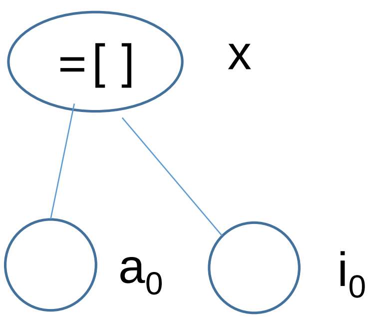
x = a[i]

a[j] = y

z = a[i]

在构造DAG时，如何避免将a[i]误判为公共子表达式

- **对于形如x=a[i]的三地址指令创建一个运算符为=[ ]的结点**
- **该结点的子结点为a和i**
- **该结点的关联变量是x**



数组引用的表示

考虑如下基本块：

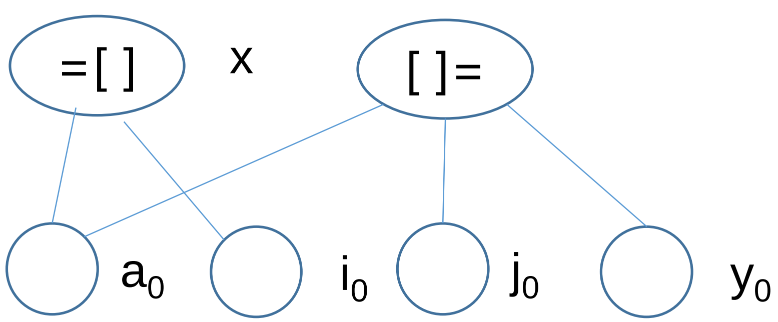
x = a[i]

a[j] = y

z = a[i]

在构造DAG时，如何避免将a[i]误判为公共子表达式

- 对于形如a[j]=y的三地址指令创建一个运算符为[ ]=的结点
- 该结点的子结点为a, j和y
- 该结点没有关联变量

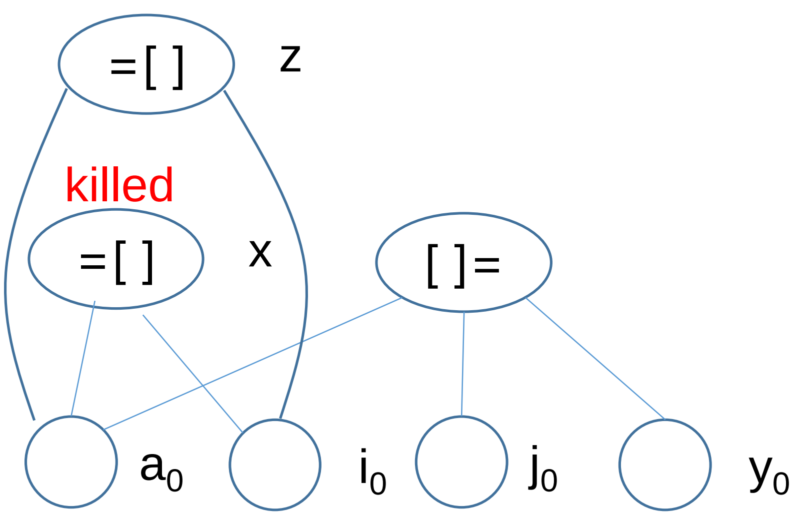


数组引用的表示

考虑如下基本块：

x = a[i]
a[j] = y
z = a[i]

在构造DAG时，如何避免将a[i]误判为公共子表达式



- 对于形如a[j]=y的三地址指令创建一个运算符为[ ]=的结点
- 该结点的子结点为a, j和y
- 该结点没有关联变量
- 该结点将杀死所有已经建立的、其值依赖于a的结点
- 被杀死的结点不能再关联定值变量，也就不能成为公共子表达式

从DAG到基本块的重组

中国科学技术大学 University of Science and Technology of China

- 当完成基本块优化后，就可以根据优化得到的DAG生成新的等价的三地址代码
- 对每个具有若干关联定值变量的结点，构造一个三地址指令来计算其中某个变量的值
  - 倾向于把计算得到的结果赋给一个在基本块出口处活跃的变量(如果没有全局活跃变量的信息作为依据，就要假设所有变量在基本块出口处活跃，不包含编译器为处理表达式而生成的临时变量)

从DAG到基本块的重组-例

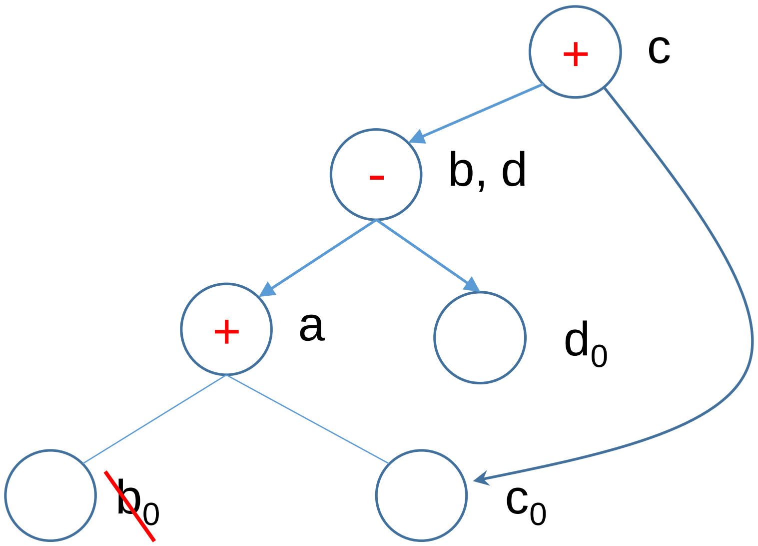
考虑如下基本块：

a = b + c

b = a – d

c = b + c

d = a - d



假设b在基本块出口处不活跃

a = b + c

d = a – d

c = d + c

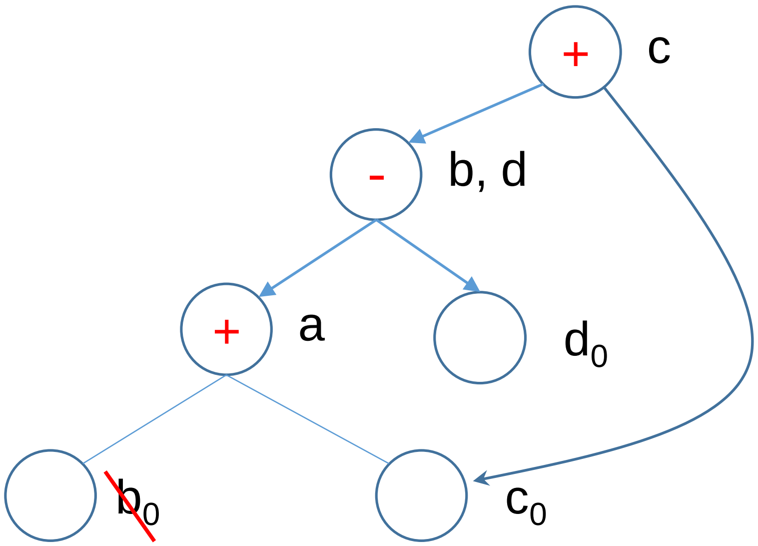
从DAG到基本块的重组

- 当完成基本块优化后，就可以根据优化得到的DAG生成新的等价的三地址代码
- 如果结点有多个关联的活跃变量，就必须引入复制语句，为每个变量赋予正确的值

从DAG到基本块的重组-例

考虑如下基本块:

a = b + c
b = a – d
c = b + c
d = a - d



假设b和d在基本块出口处都活跃

a = b + c
d = a – d
b = d
c = d + c

优化的实现方式

- 局部视角-基本块的优化
  - DAG
- **全局视角-跨基本块的优化**
  - 数据流分析

数据流分析

- **Data-flow analysis**
  - 一组用来获取程序执行路径上的数据流信息的技术
- **数据流分析应用**
  - 到达-定值分析(Reaching-Definition Analysis)
  - 活跃变量分析(Live-Variable Analysis)
  - 可用表达式分析(Available-Expression Analysis)
- **在每一种数据流分析应用中，都会把每个程序点和一个数据流值关联起来**

数据流抽象

❑流图上的点(程序点)

- ❖基本块中，两个相邻的语句之间为程序的一个点
- ❖基本块的开始点和结束点

❑流图上的路径

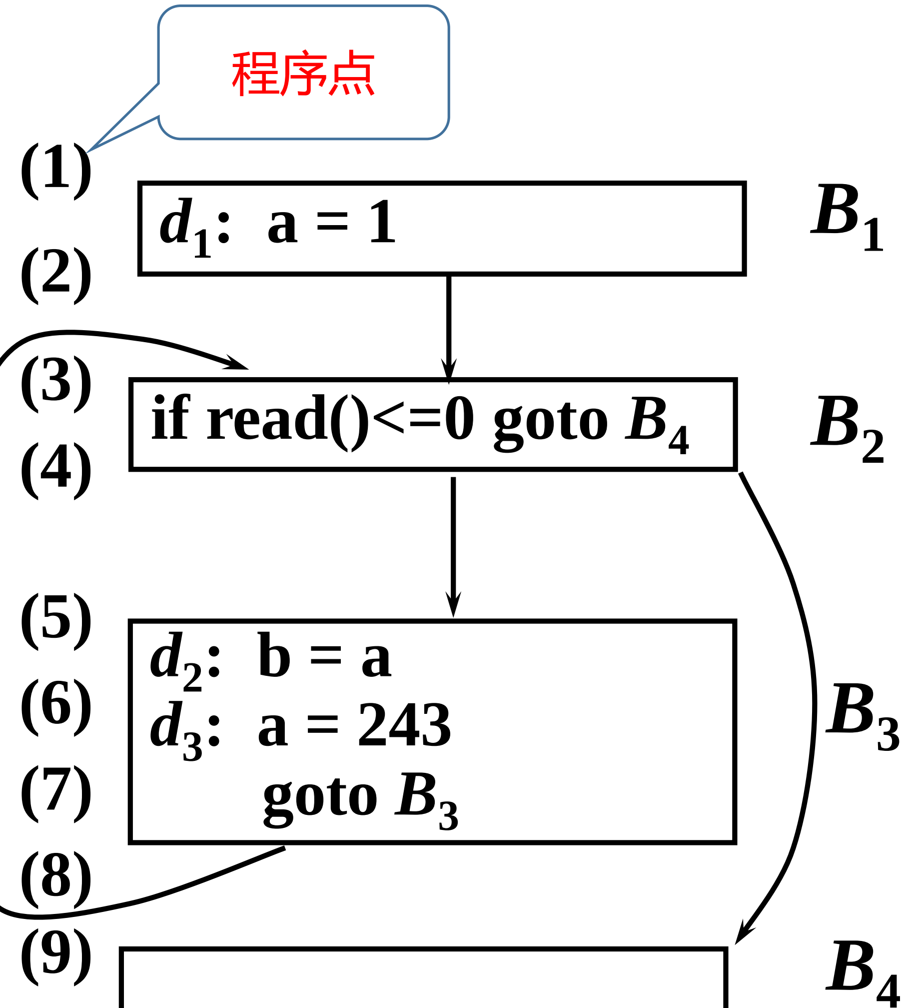
- ❖点序列$p_1, p_2, \ldots, p_n$，对1和n - 1间的每个i，满足

(1) p_i是先于一个语句的点，p_{i+1}是同一块中位于该语句后的点，或者

(2) p_i是某块的结束点，p_{i+1}是后继块的开始点

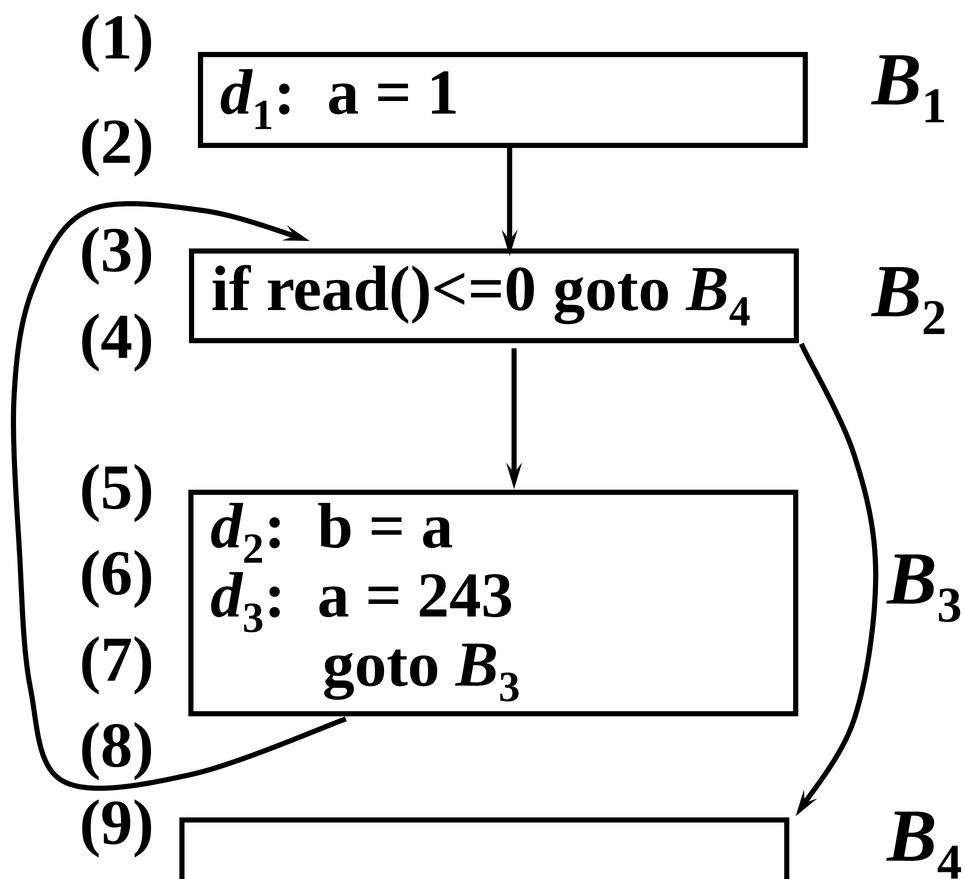
❑流图上路径实例

- (1, 2, 3, 4, 9)
- (1, 2, 3, 4, 5, 6, 7, 8, 3, 4, 9)
- (1, 2, 3, 4, 5, 6, 7, 8,
  3, 4, 5, 6, 7, 8, 3, 4, 9)
- (1, 2, 3, 4, 5, 6, 7, 8,
  3, 4, 5, 6, 7, 8,
  3, 4, 5, 6, 7, 8, …)
  路径长度无限
- 路径数无限



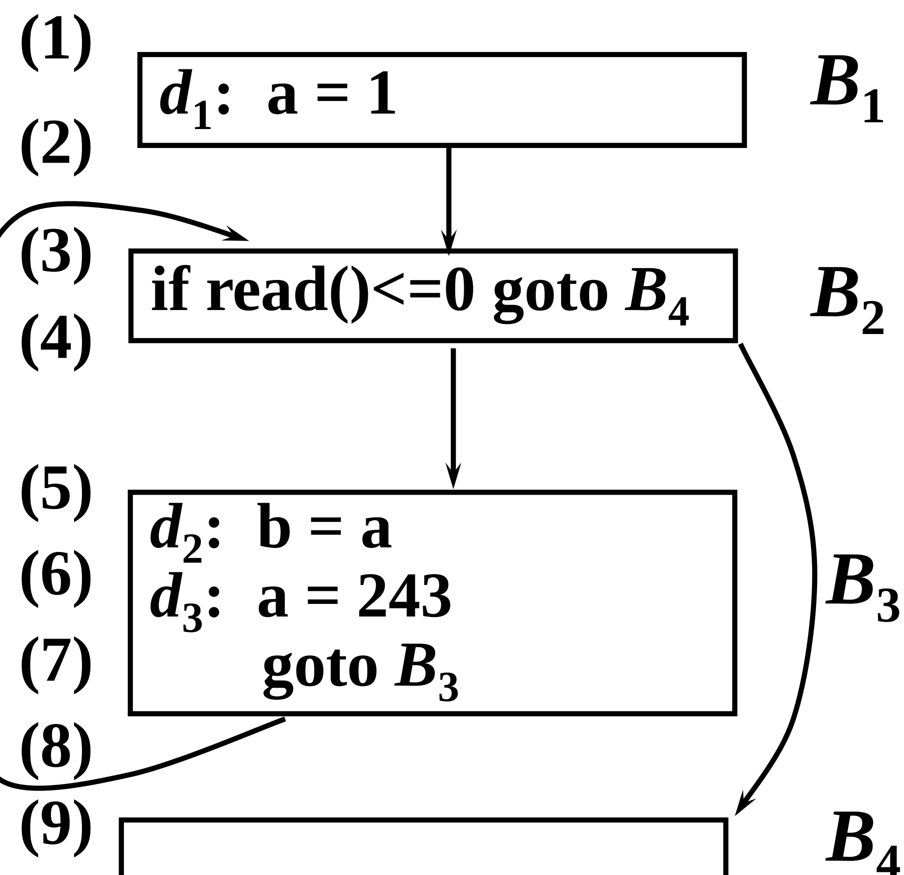
数据流分析介绍

- 分析程序的行为时，必须在其流图上考虑**所有的执行路径**（在调用或返回语句被执行时，还需要考虑执行路径在多个流图之间的跳转）
  - 通常，从流图得到的程序**执行路径数无限**，且执行路径长度没有有限的上界



数据流分析介绍

- 分析程序的行为时，必须在其流图上考虑**所有的执行路径**（在调用或返回语句被执行时，还需要考虑执行路径在多个流图之间的跳转）
  - 每个程序点的**不同状态数也可能无限**
  - 程序状态：存储单元到值的映射



数据流分析模式

- 数据流值代表在程序点能观测到的所有可能程序状态集合的一个抽象
- 对于一个语句*s*
  - *s*之前的程序点对应的数据流值用IN[*s*]表示
  - s之后的程序点对应的数据流值用OUT[*s*]表示

❑传递函数(transfer function) f

- ❖语句前后两点的数据流值受该语句的语义约束
- ❖若沿执行路径正向传播，则 $\mathrm{OUT}[s] = f_s(\mathrm{IN}[s])$
- ❖若沿执行路径逆向传播，则 $\mathrm{IN}[s] = f_s(\mathrm{OUT}[s])$

若基本块 B 由语句 $s_1, s_2, \ldots, s_n$ 依次组成，则

- ❖$\mathrm{IN}[s_{i+1}] = \mathrm{OUT}[s_i],\ i = 1, 2, \ldots, n-1$

考虑的是在语句执行后输入输出之间的变化关系

基本块上的数据流模式

- **IN[*B*]：紧靠基本块B之前的数据流值**
  - $\mathrm{IN}[B] = \mathrm{IN}[s_1]$
- **OUT[*B*]：紧靠基本块B之后的数据流值**
  - $\mathrm{OUT}[B] = \mathrm{OUT}[s_n]$
- **f_B :基本块B的传递函数**
  - 前向数据流： $\mathrm{OUT}[B] = f_B\,(\mathrm{IN}[B])$
    - $f_B = f_n \circ \ldots \circ f_2 \circ f_1$
  - 逆向数据流： $\mathrm{IN}[B] = f_B\,(\mathrm{OUT}[B])$
    - $f_B = f_1 \circ \ldots \circ f_{n-1} \circ f_n$

控制流约束

- 正向传播

$$\mathrm{IN}[B] = \cup_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$$

- 逆向传播

$$\mathrm{OUT}[B] = \cup_{S\text{是}B\text{的后继}} \mathrm{IN}[S]$$

约束方程组的解通常不是唯一的

- 求解的目标是要找到满足这两组约束（控制流约束和迁移约束）的最“精确”解

考虑的是在其他语句或块对于输入的影响和本次执行的输出对其他语句和块的影响

三种有用的分析

- 到达-定值
- 活跃变量
- 可用表达式

到达-定值分析

- 到达一个程序点的所有定值(gen)
- 定值的注销(kill)
  - 在一条执行路径上，对x的赋值注销先前对x的所有赋值
- 别名给到达-定值的计算带来困难，因此，本章其余部分仅考虑变量无别名的情况

❑定值与引用

d： x := y + z // 语句d 是变量x的一个定值点

ud链，即引用—定值链。
流图中有路径d---->u，
且该路径上x在d处的定值没有被杀死。

u： w := x + v // 语句u 是变量x的一个引用点

❑变量x在d点的定值到达u点

到达-定值分析的用途

- 循环不变计算的检测
  - 如果循环中含有赋值x=y+z，而y和z所有可能的定值都在循环外，那么y+z就是循环不变计算
- 常量合并
  - 如果对变量x的某次使用只有一个定值到达，且该定值把一个常量赋给x，则可以用该常量替换x
- 错误检测
  - 判定变量x在p点上是否未经定值就被引用

到达-定值分析

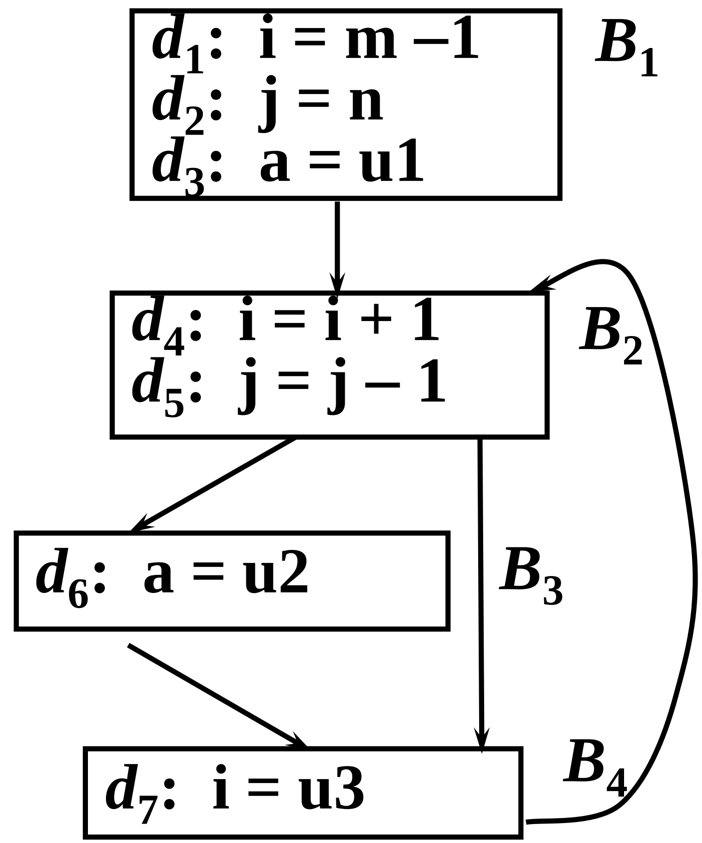
- *gen*和*kill*分别表示一个基本块生成和注销的定值

$gen\,[B_1] = \{d_1, d_2, d_3\}$
$kill\,[B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\,[B_2] = \{d_4, d_5\}$
$kill\,[B_2] = \{d_1, d_2, d_7\}$

$gen\,[B_3] = \{d_6\}$
$kill\,[B_3] = \{d_3\}$

$gen\,[B_4] = \{d_7\}$
$kill\,[B_4] = \{d_1, d_4\}$



❑基本块的*gen*和*kill*是怎样计算的

- ❖对三地址指令 *d*: u = v + w，它的状态传递函数是

  $f_d(x) = gen_d \cup (x - kill_d)$

- ❖若：$f_1(x) = gen_1 \cup (x - kill_1)$，$f_2(x) = gen_2 \cup (x - kill_2)$

  则：$f_2(f_1(x)) = gen_2 \cup (gen_1 \cup (x - kill_1) - kill_2)$

  $= (gen_2 \cup (gen_1 - kill_2)) \cup (x - (kill_1 \cup kill_2))$

- ❖若基本块*B*有*n*条三地址指令

  $kill_B = kill_1 \cup kill_2 \cup \ldots \cup kill_n$

  $gen_B = gen_n \cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup \ldots \cup (gen_1 - kill_2 - kill_3 - \ldots - kill_n)$

到达-定值分析

❑到达-定值的数据流等式

- gen_B：B中能到达B的结束点的定值语句
- $kill_B$：整个程序中决不会到达B结束点的定值
- IN[B]：能到达B的开始点的定值集合
- OUT[B]：能到达B的结束点的定值集合

两组等式（根据*gen*和*kill*定义IN和OUT）

- $\mathrm{IN}[B] = \cup_{P\text{是}B\text{的前驱}} \mathrm{OUT}[P]$
- $\mathrm{OUT}[B] = gen_B \cup (\mathrm{IN}[B] - kill_B)$
- $\mathrm{OUT}[\mathrm{ENTRY}] = \varnothing$

❑到达-定值方程组的迭代求解，最终到达不动点

// **正向数据流分析**

引入两个虚拟块：ENTRY、EXIT

(1) OUT[ENTRY] = Ø;

(2) for (除了ENTRY以外的每个块B) OUT[B] = Ø;

(3) while (任何一个OUT出现变化){

(4) for (除了ENTRY以外的每个块B) {

(5) IN[B] = $\cup_{\text{P是B的前驱}}$ OUT[P];

(6) OUT[B] = $gen_B \cup (IN[B] - kill_B)$;

(7) }}

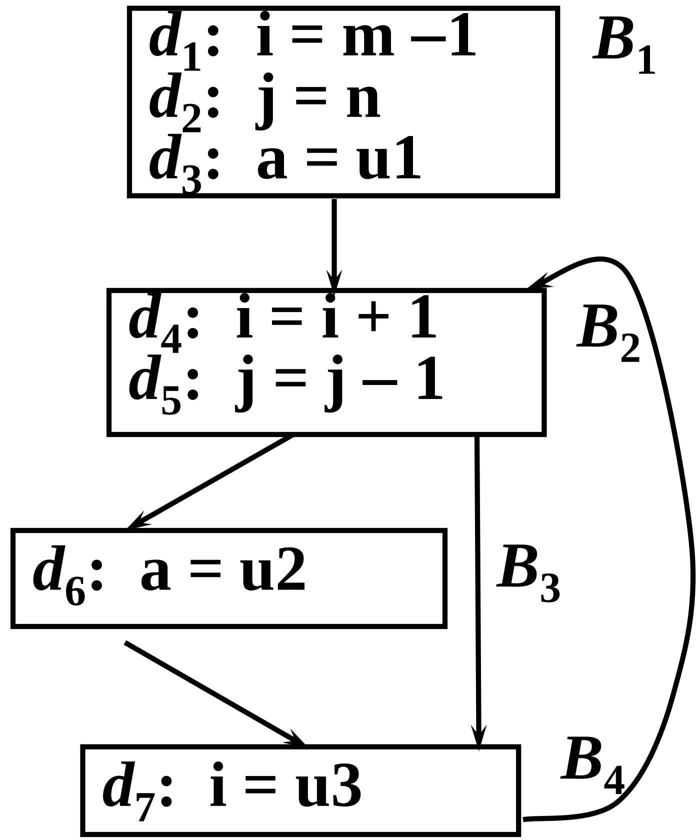
向量求解：集合并操作使用逻辑或，集合相减使用后者求补再逻辑与

到达-定值分析

$\text{IN}[B] = \cup_{P是B的前驱} \text{OUT}[P]$

$\text{OUT}[B] = gen_B \cup (\text{IN}[B] - kill_B)$

	IN [B]	OUT [B]
B_1		000 0000
B_2		000 0000
B_3		000 0000
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$
$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$
$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$
$kill\ [B_3] = \{d_3\}$

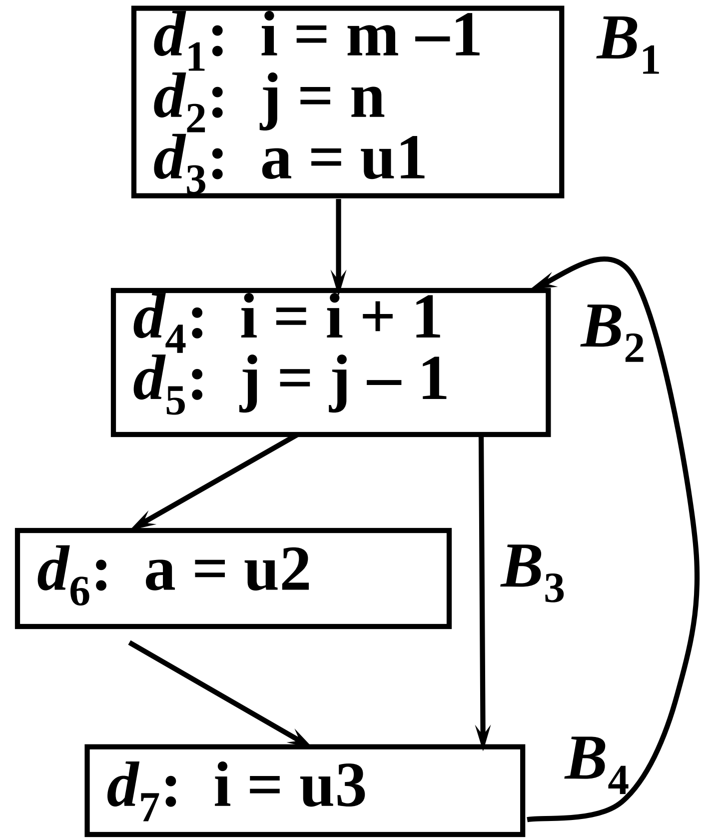
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	000 0000
B_2		000 0000
B_3		000 0000
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$

$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$

$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$

$kill\ [B_3] = \{d_3\}$

$gen\ [B_4] = \{d_7\}$

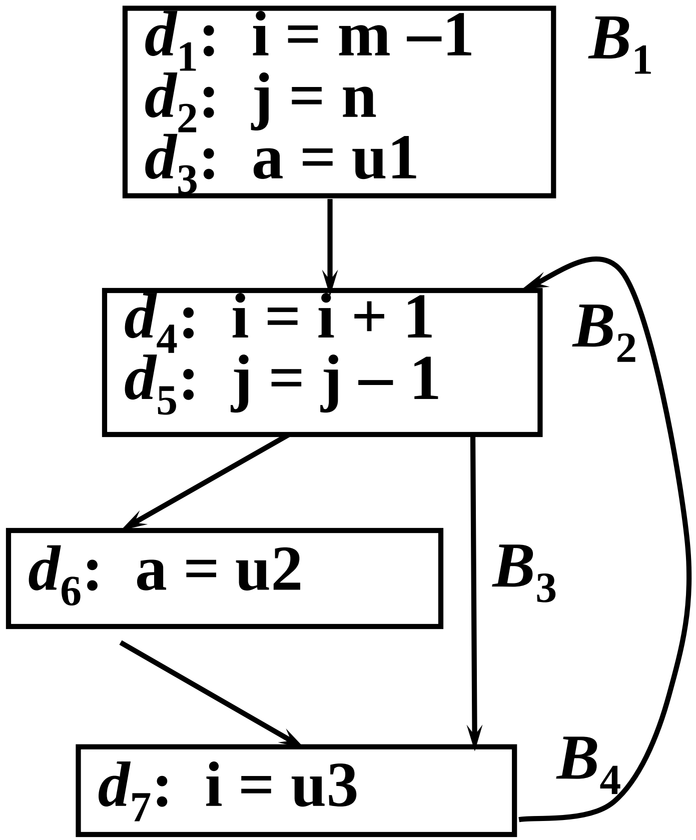
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P\text{是}B\text{的前驱}} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2		000 0000
B_3		000 0000
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$
$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$
$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$
$kill\ [B_3] = \{d_3\}$

$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

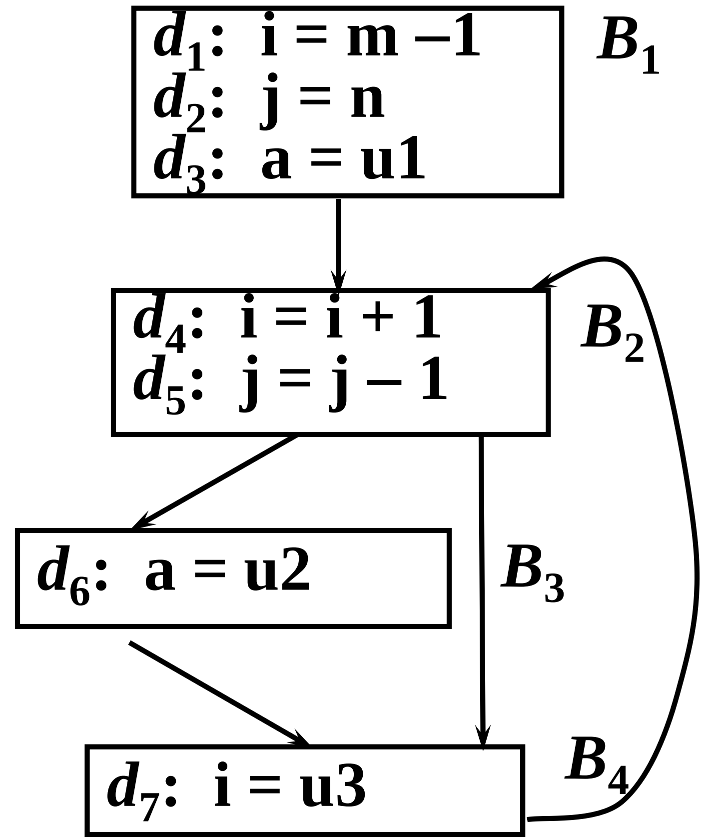
到达-定值分析

中国科学技术大学 University of Science and Technology of China

$IN[B] = \cup_{P\text{是}B\text{的前驱}} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0000	000 0000
B_3		000 0000
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$

$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$

$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$

$kill\ [B_3] = \{d_3\}$

$gen\ [B_4] = \{d_7\}$

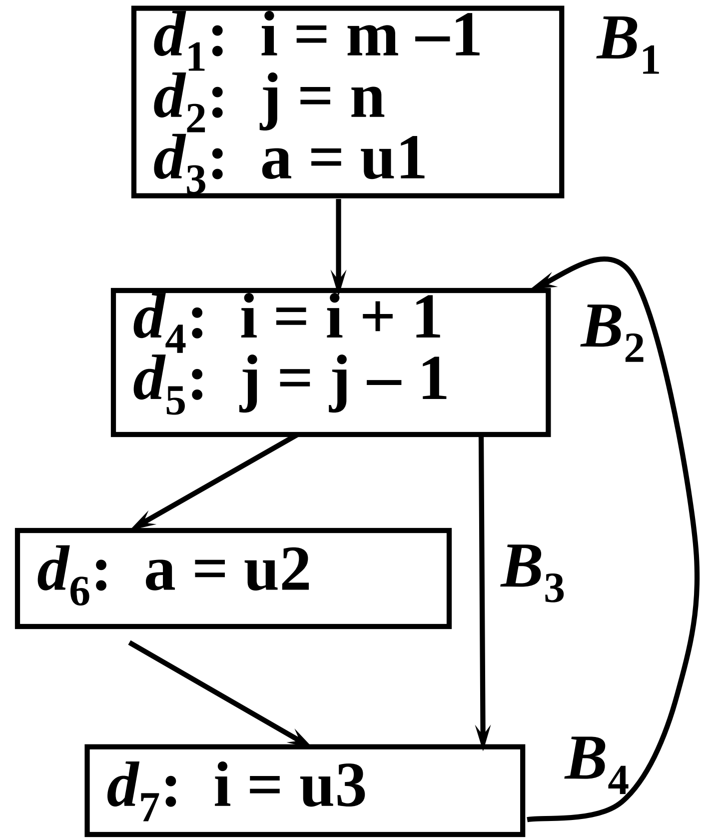
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0000	001 1100
B_3		000 0000
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$

$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$

$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$

$kill\ [B_3] = \{d_3\}$

$gen\ [B_4] = \{d_7\}$

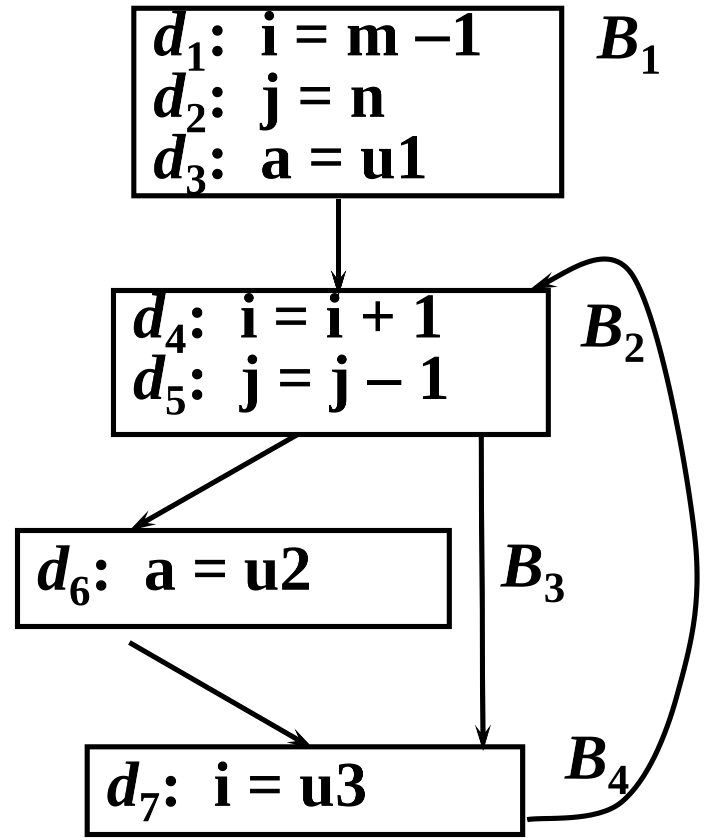
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0000	001 1100
B_3	001 1100	000 0000
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$
$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$
$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$
$kill\ [B_3] = \{d_3\}$

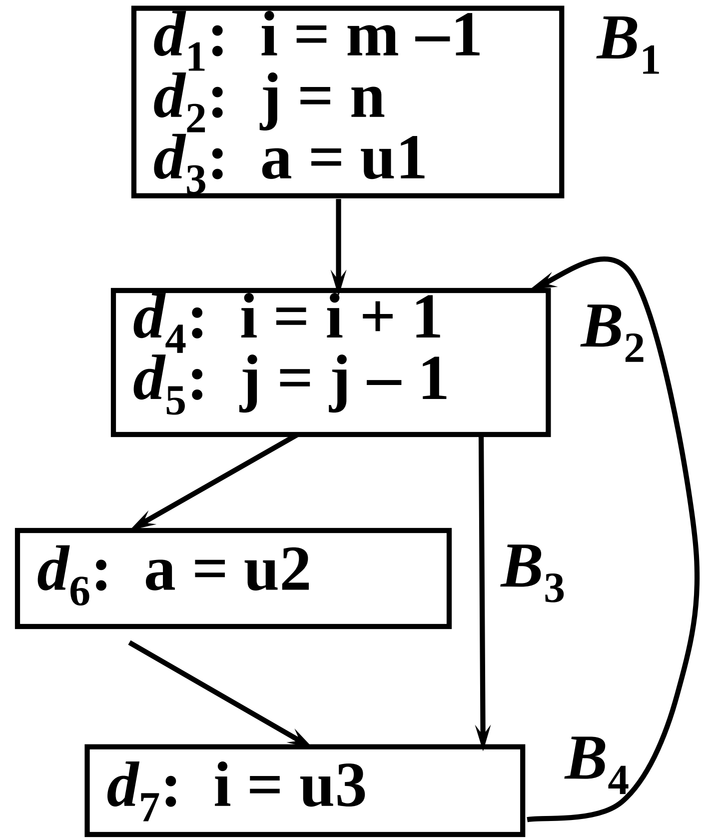
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0000	001 1100
B_3	001 1100	000 1110
B_4		000 0000



$gen\ [B_1] = \{d_1, d_2, d_3\}$
$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$
$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$
$kill\ [B_3] = \{d_3\}$

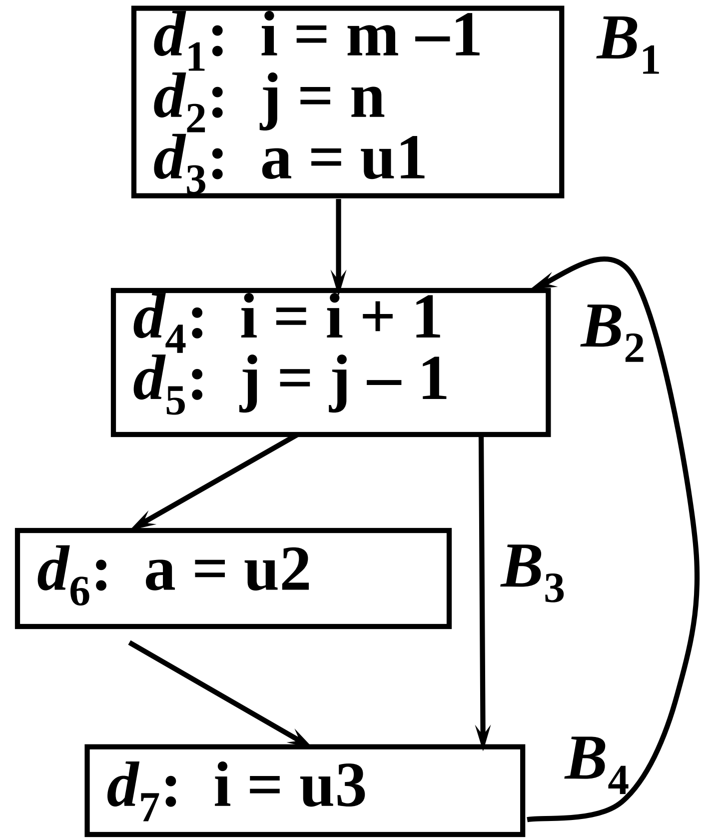
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0000	001 1100
B_3	001 1100	000 1110
B_4	001 1110	000 0000



$gen[B_1] = \{d_1, d_2, d_3\}$

$kill[B_1]=\{d_4, d_5, d_6, d_7\}$

$gen[B_2] = \{d_4, d_5\}$

$kill[B_2] = \{d_1, d_2, d_7\}$

$gen[B_3] = \{d_6\}$

$kill[B_3] = \{d_3\}$

$gen[B_4] = \{d_7\}$

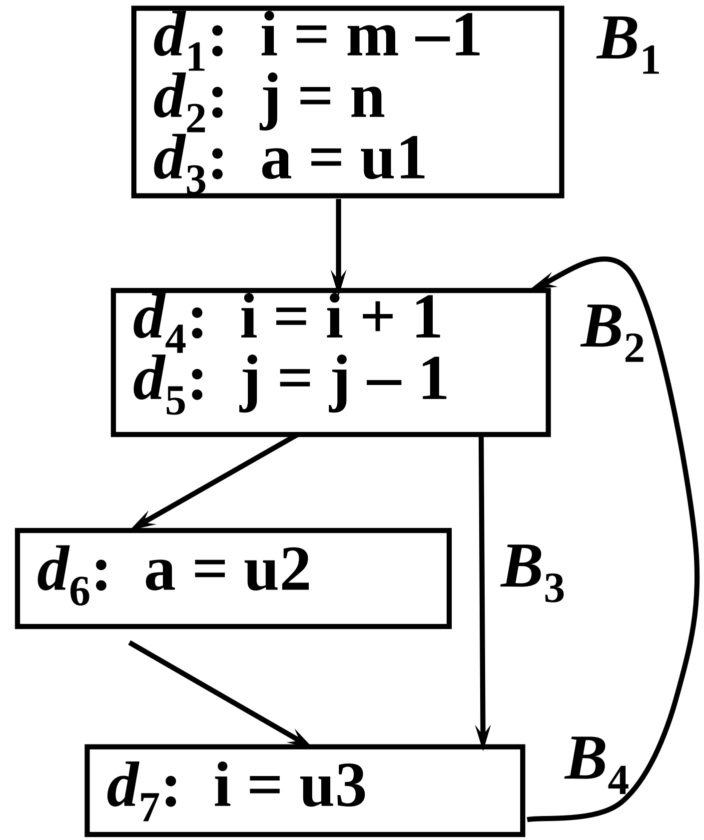
$kill[B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0000	001 1100
B_3	001 1100	000 1110
B_4	001 1110	001 0111



$gen\ [B_1] = \{d_1, d_2, d_3\}$
$kill\ [B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$
$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$
$kill\ [B_3] = \{d_3\}$

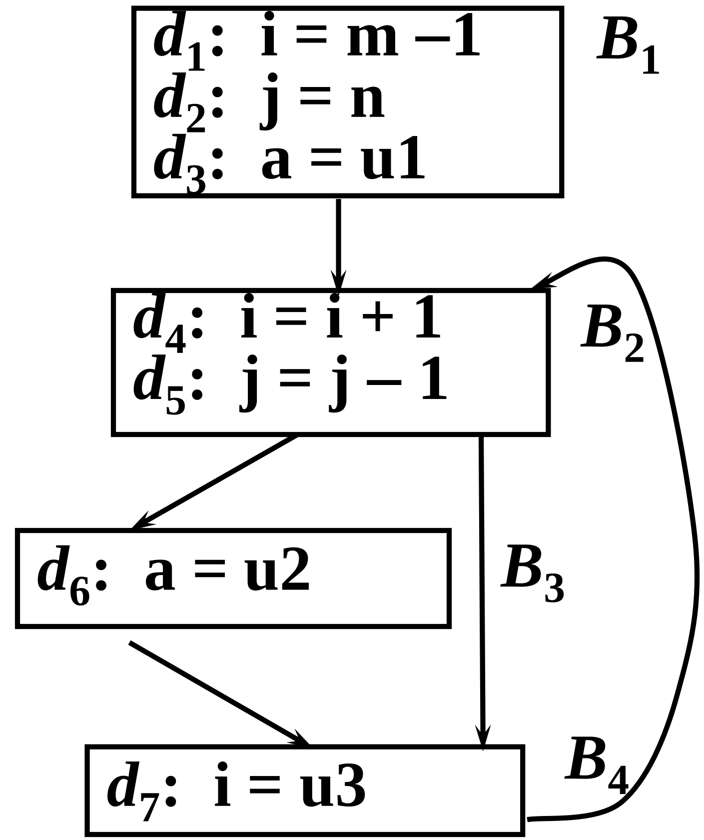
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0111	001 1100
B_3	001 1100	000 1110
B_4	001 1110	001 0111



$gen\,[B_1] = \{d_1, d_2, d_3\}$

$kill\,[B_1]=\{d_4, d_5, d_6, d_7\}$

$gen\,[B_2] = \{d_4, d_5\}$

$kill\,[B_2] = \{d_1, d_2, d_7\}$

$gen\,[B_3] = \{d_6\}$

$kill\,[B_3] = \{d_3\}$

$gen\,[B_4] = \{d_7\}$

$kill\,[B_4] = \{d_1, d_4\}$

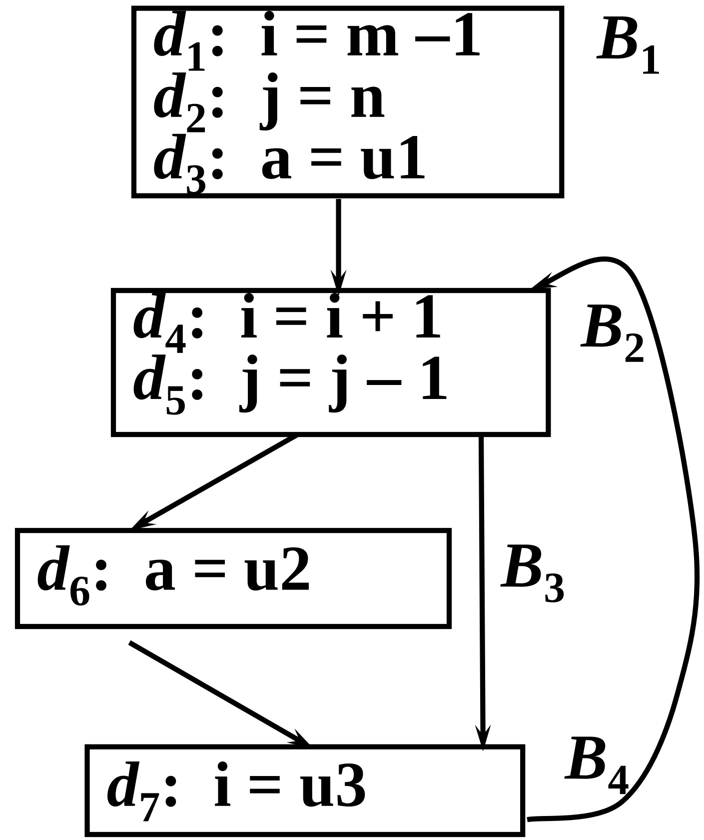
到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

$OUT[B] = gen_B \cup (IN[B] - kill_B)$

	IN [B]	OUT [B]
B_1	000 0000	111 0000
B_2	111 0111	001 1110
B_3	001 1100	000 1110
B_4	001 1110	001 0111

不再继续演示迭代计算



$gen\ [B_1] = \{d_1, d_2, d_3\}$
$kill\ [B_1] = \{d_4, d_5, d_6, d_7\}$

$gen\ [B_2] = \{d_4, d_5\}$
$kill\ [B_2] = \{d_1, d_2, d_7\}$

$gen\ [B_3] = \{d_6\}$
$kill\ [B_3] = \{d_3\}$

$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析非向量计算方法

- 迭代计算
  - 计算次序，深度优先序，即 B1 -> B2 -> B3 -> B4
  - 初始值：for all B: IN[B] = ∅; OUT[B] = GEN[B]
  - 第一次迭代：

IN[B1] = ∅; // B1 无前驱结点

OUT[B1] = GEN[B1] ∪(IN[B1]-KILL[B1]) = GEN[B1] = { d1, d2, d3 }

IN[B2] = OUT[B1] ∪OUT[B4] = {d1, d2, d3 } ∪{ d7 } = { d1, d2, d3, d7 }

OUT[B2] = GEN[B2] ∪(IN[B2]-KILL[B2]) = { d4, d5 } ∪{ d3 } = { d3, d4, d5 }

IN[B3] = OUT[B2] = { d3, d4, d5 }

OUT[B3] = { d6 } ∪ ({ d3, d4, d5 } – { d3 }) = { d4, d5, d6 }

IN[B4] = OUT[B3] ∪ OUT[B2] = { d3, d4, d5, d6 }

OUT[B4] = { d7 } ∪ ({ d3, d4, d5, d6 } – { d1, d4 }) = { d3, d5, d6, d7 }

到达-定值分析非向量计算方法

一第二次迭代

IN[B1] = ∅; // B1 无前驱结点

OUT[B1] = GEN[B1] ∪ (IN[B1]-KILL[B1]) = GEN[B1] = { d1, d2, d3 }

IN[B2] = OUT[B1] ∪ OUT[B4] = { d1,d2,d3 } ∪ { d3, d5, d6, d7 } = {d1, d2, d3, d5, d6, d7}

OUT[B2] = GEN[B2] ∪ (IN[B2]-KILL[B2]) = { d4, d5 } ∪ { d3, d5, d6 } = { d3, d4, d5, d6 }

IN[B3] = OUT[B2] = { d3, d4, d5, d6 }

OUT[B3] = { d6 } ∪ ({ d3, d4, d5, d6 } – { d3 }) = { d4, d5, d6 }

IN[B4] = OUT[B3] ∪ OUT[B2] = { d3, d4, d5, d6 }

OUT[B4] = { d7 } ∪ ({ d3, d4, d5, d6 } – { d1, d4 }) = { d3, d5, d6, d7 }

经过三次迭代后，IN[B]和OUT[B] 不再变化。

到达-定值分析

- 到达-定值数据流等式是正向的方程

  $$\text{OUT}[B] = gen[B] \cup (\text{IN}[B] - kill[B])$$

  $$\text{IN}[B] = \cup_{P\text{是}B\text{的前驱}} \text{OUT}[P]$$

  某些数据流等式是反向的

- 到达-定值数据流等式的合流运算是求并集

  $$\text{IN}[B] = \cup_{P\text{是}B\text{的前驱}} \text{OUT}[P]$$

  某些数据流等式的合流运算是求交集

- 对到达-定值数据流方程，迭代求它的最小解

  某些数据流方程可能需要求最大解

三种有用的分析

- ❑到达-定值
- ❑活跃变量
- ❑可用表达式

活跃变量分析的应用

- 删除无用赋值
- 为基本块分配寄存器
  - 如果所有寄存器都被占用，且还需要申请一个寄存器，则应该考虑使用已经存放死亡值的寄存器
  - 如果一个值在基本块结尾处是死的，就不必在结尾处保存这个值了

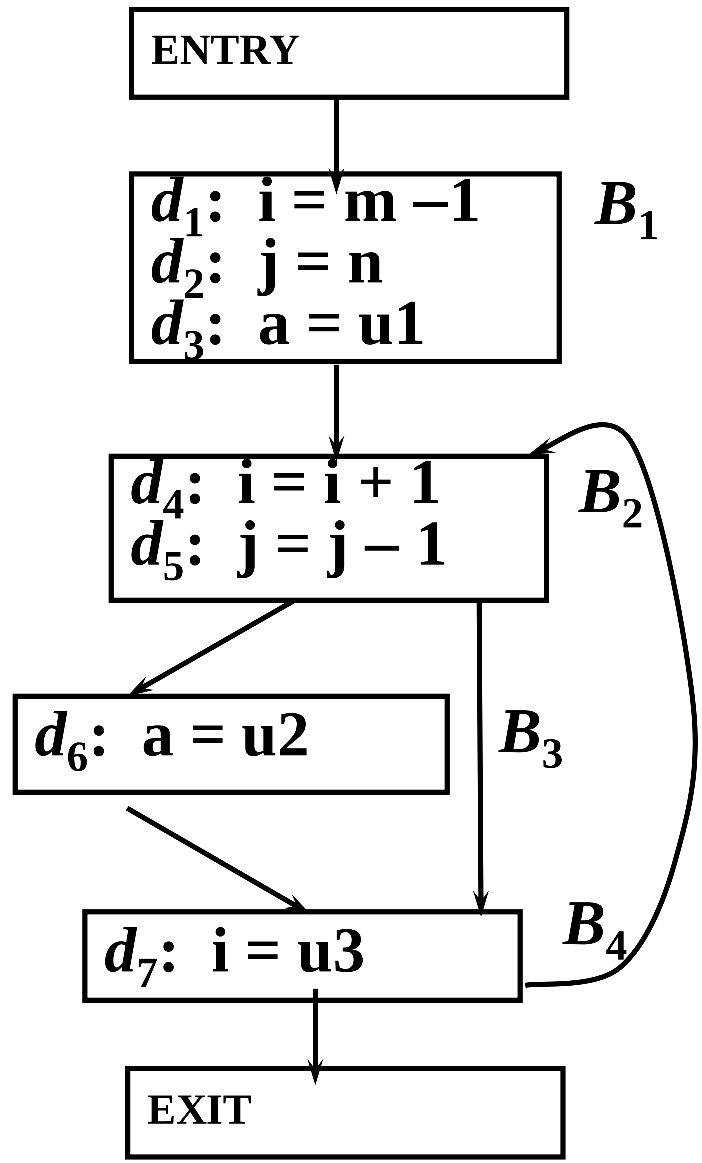
❑定义

- ❖ x的值在*p*点开始的某条执行路径上被引用，则说x在*p*点活跃，否则称x在*p*点已经死亡
- ❖ IN[*B*]：块*B*开始点的活跃变量集合
- ❖ OUT[*B*]：块*B*结束点的活跃变量集合
- ❖ use_B：块*B*中有引用，且在引用前在B中没有被定值的变量集
- ❖ def_B：块B中有定值，且该定值前在B中没有被引用的变量集

活跃变量分析

❑例

- $use[B_1]$ = { m, n, u1 }
- $def[B_1]$ = {i, j, a}
- $use[B_2]$ = { i, j }
- $def[B_2]$ = {}
- $use[B_3]$ = { u2}
- $def[B_3]$ = {a}
- $use[B_4]$ = { u3 }
- $def[B_4]$ = {i}



活跃变量数据流等式

- IN $[B] = use_B \cup$ (OUT $[B] - def_B$)
- OUT$[B] = \cup_{S是B的后继}$ IN $[S]$
- IN [EXIT] = $\varnothing$

和到达-定值等式之间的联系与区别

- 都以集合并算符作为它们的汇合算符
- 信息流动方向相反，IN和OUT的作用相互交换
- *use*和*def*分别取代*gen*和*kill*
- 仍然需要最小解

活跃变量的迭代计算算法

输入：流图G，其中每个基本块B的use和def都已计算

输出：IN[B]和OUT[B]

IN[EXIT] = Ø;

for (除了EXIT以外的每个块B) IN[B] = Ø;

while (某个IN值出现变化) {

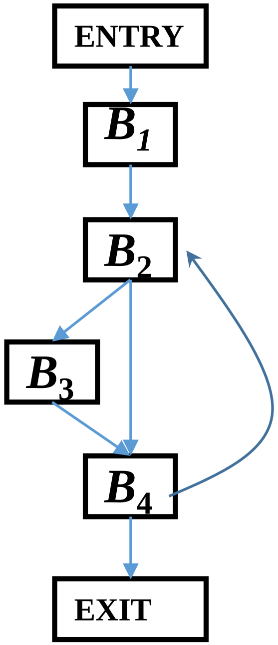
for (除了EXIT以外的每个块B) {

$$\text{OUT}[B] = \cup_{S\text{是}B\text{的后继}} \text{IN}[S]$$

$$\text{IN}[B] = use_B \cup (\text{OUT}[B] - def_B);$$

}}

活跃变量举例



use$[B_1]$ = { m, n, u1 }
def$[B_1]$ = {i, j, a}
use$[B_2]$ = { i, j }
def$[B_2]$ = {}
use$[B_3]$ = { u2}
def$[B_3]$ = {a}
use$[B_4]$ = { u3 }
def$[B_4]$ = {i}

IN[EXIT] = Ø;

for (除了EXIT以外的每个块B) IN[B] = Ø;

while (某个IN值出现变化) {

for (除了EXIT以外的每个块B) {

$OUT[B] = \cup_{S是B的后继} IN[S]$

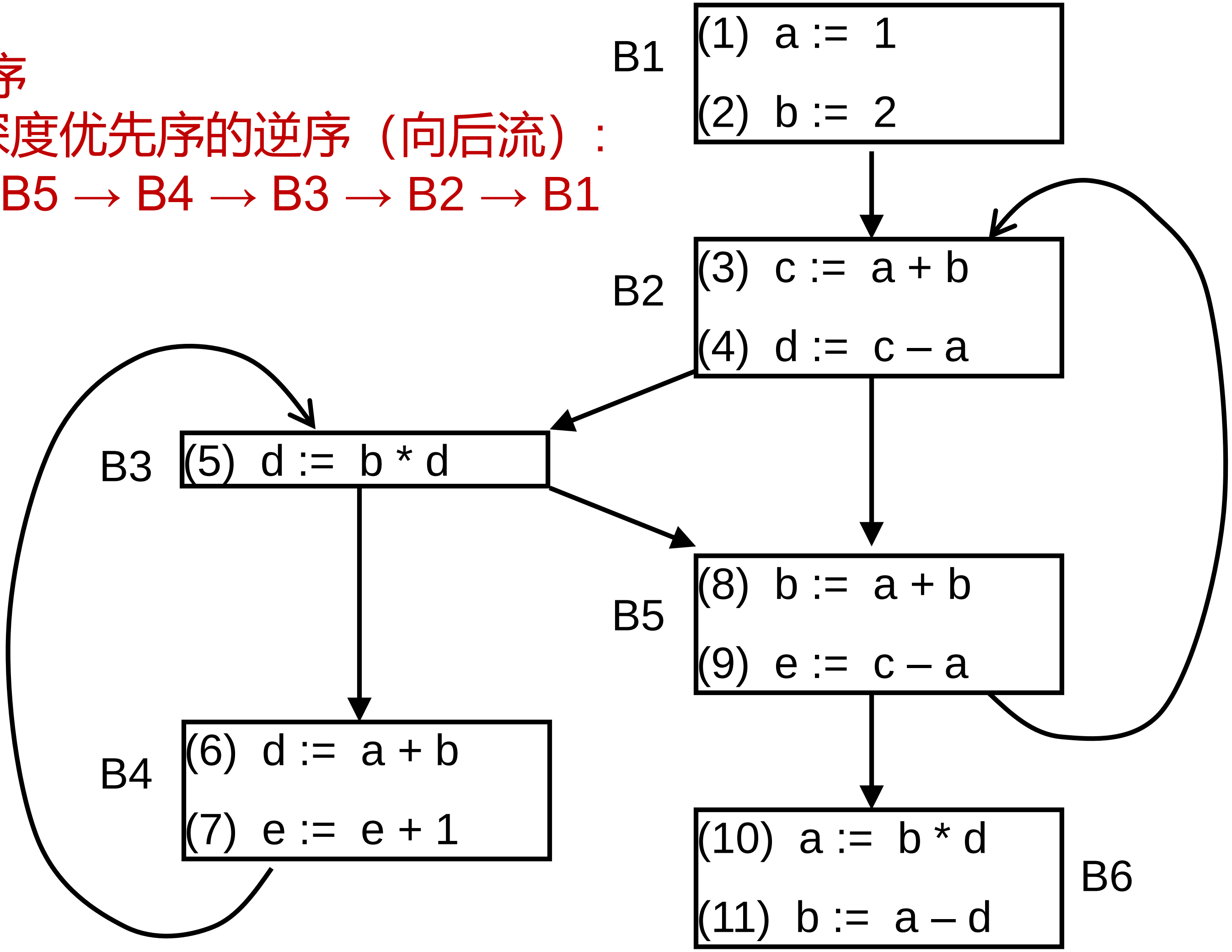
$IN[B] = use_B \cup (OUT[B] - def_B)$;

}}

	OUT[B][1]	IN[B][1]	OUT[B][2]	IN[B][2]	OUT[B][3]	IN[B][3]
B_4		u3	i, j, u2, u3	j, u2, u3	i, j, u2, u3	j, u2, u3
B_3	u3	u2, u3	j, u2, u3	j, u2, u3	j, u2, u3	j, u2, u3
B_2	u2, u3	i, j, u2, u3	j, u2, u3	i, j, u2, u3	j, u2, u3	i, j, u2, u3
B_1	i, j, u2, u3	m, n, u1, u2, u3	i, j, u2, u3	m, n, u1, u2, u3	i, j, u2, u3	m, n, u1, u2, u3

活跃变量分析-举例2

计算次序
* 结点深度优先序的逆序（向后流）：
* B6 → B5 → B4 → B3 → B2 → B1



基本块出口活跃变量

- 各基本块USE和DEF如下，

  USE[B1] = { } ; DEF[B1] = { a, b }

  USE[B2] = { a, b } ; DEF[B2] = { c, d }

  USE[B3] = { b, d } ; DEF[B3] = { }

  USE[B4] = { a, b, e } ; DEF[B4] = { d }

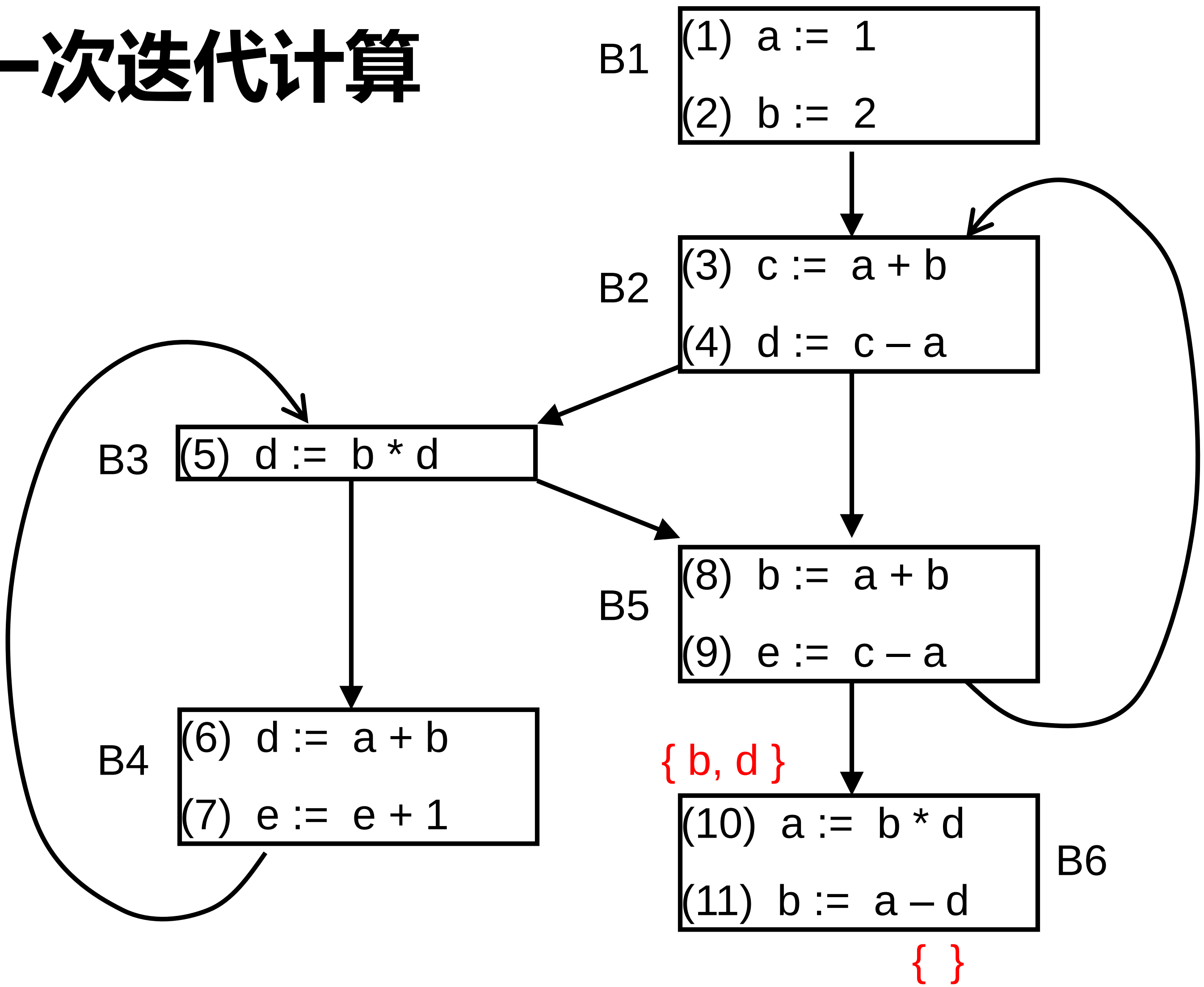
  USE[B5] = { a, b, c } ; DEF[B5] = { e }

  USE[B6] = { b, d } ; DEF[B6] = { a }

- 初始值，all B, IN[B] = { },

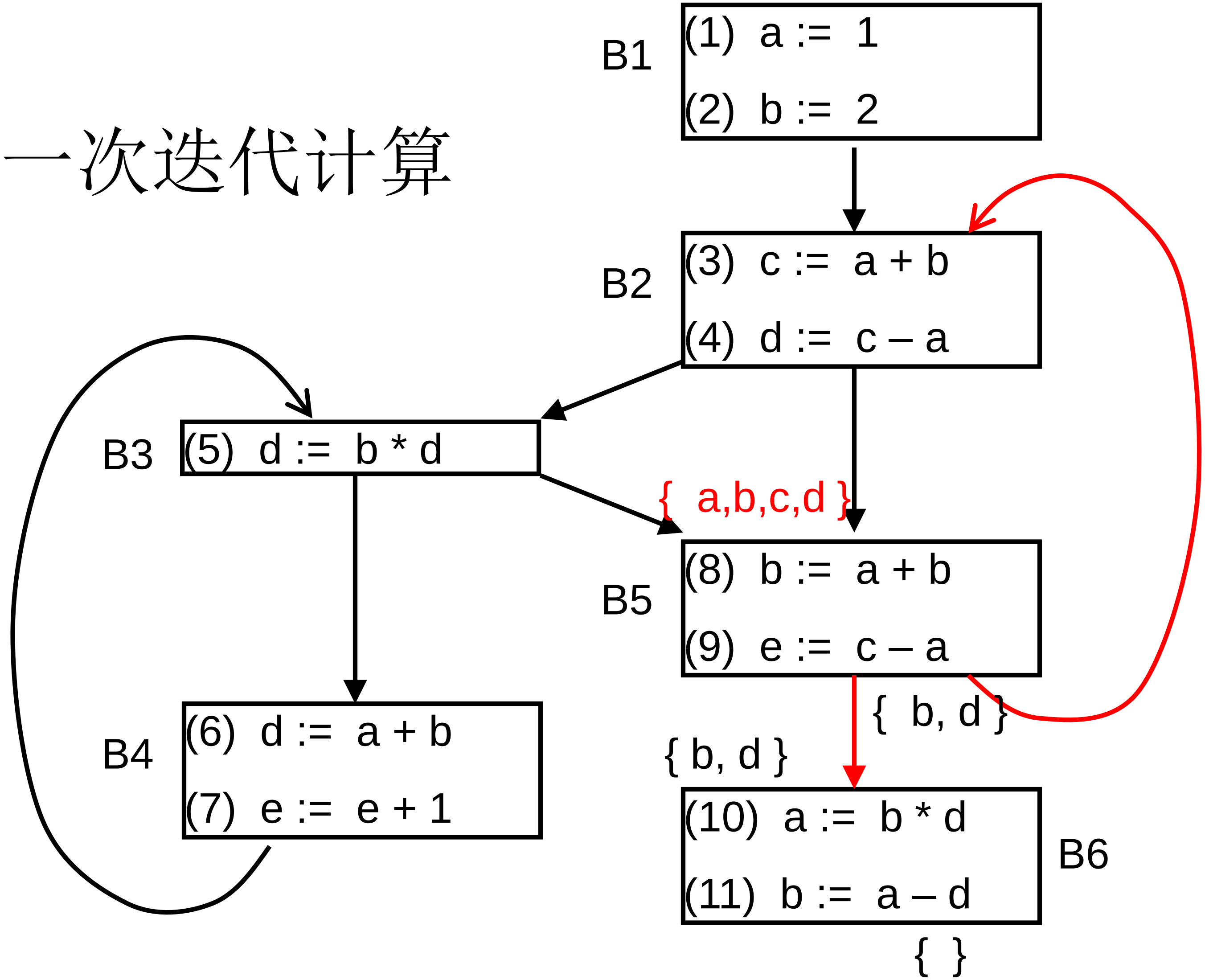
  OUT[B6]={ }//出口块

❑第一次迭代计算


B1
(1) a := 1
(2) b := 2
B2
(3) c := a + b
(4) d := c – a
B3
(5) d := b * d
B4
(6) d := a + b
(7) e := e + 1
B5
(8) b := a + b
(9) e := c – a
{ b, d }
(10) a := b * d
(11) b := a – d
B6
{ }

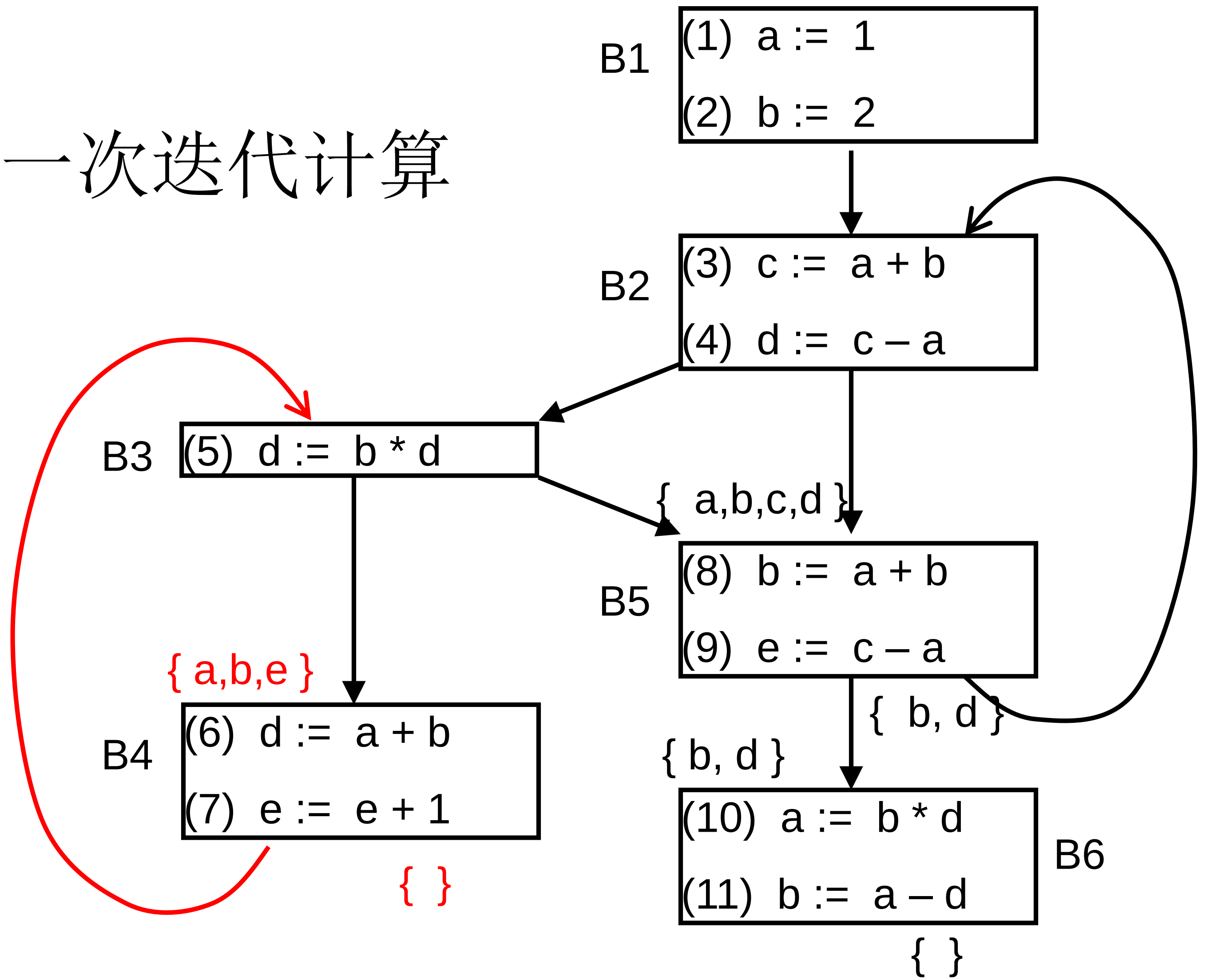
基本块出口活跃变量

- 第一次迭代计算



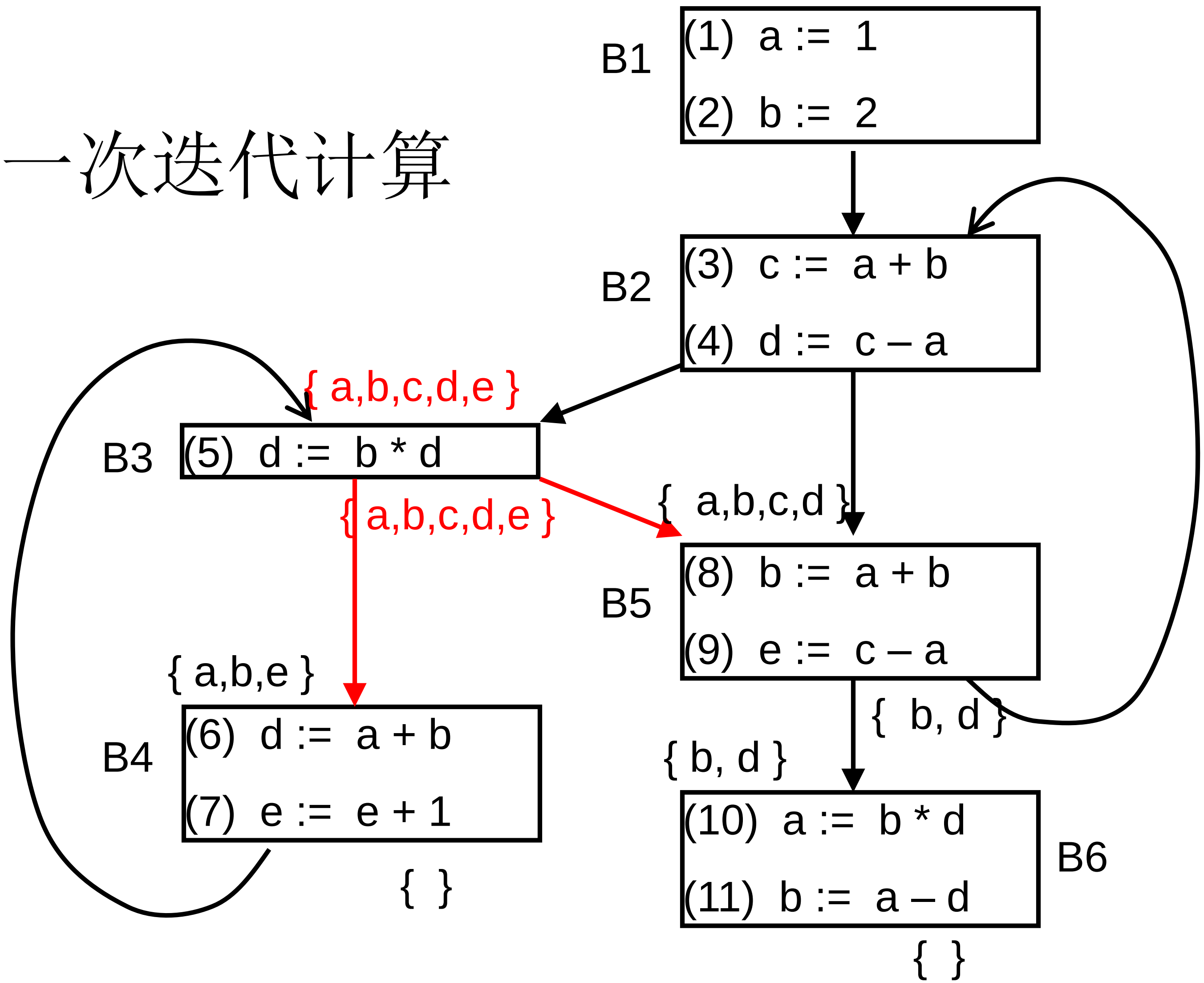
基本块出口活跃变量

- 第一次迭代计算

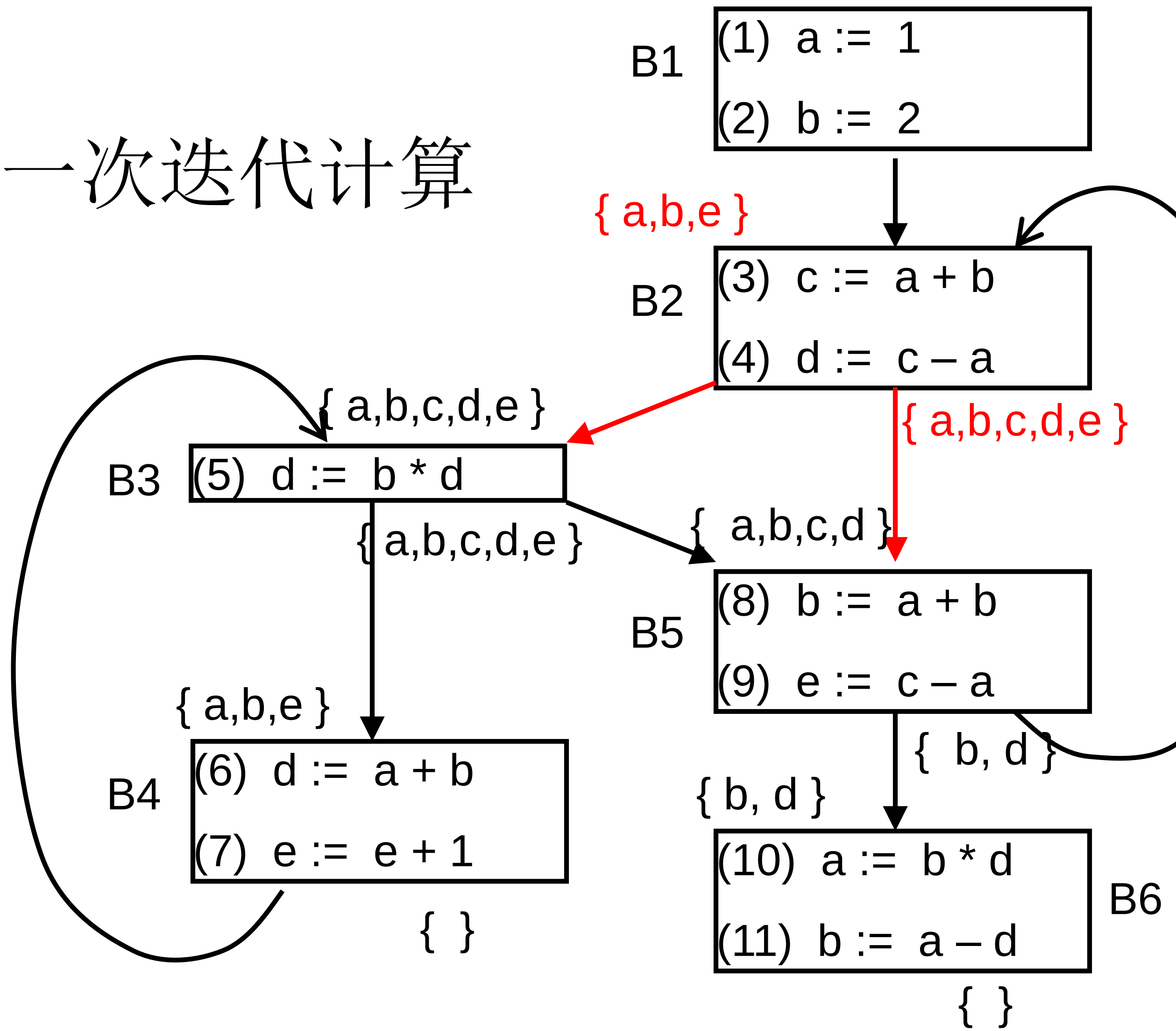


基本块出口活跃变量

- 第一次迭代计算



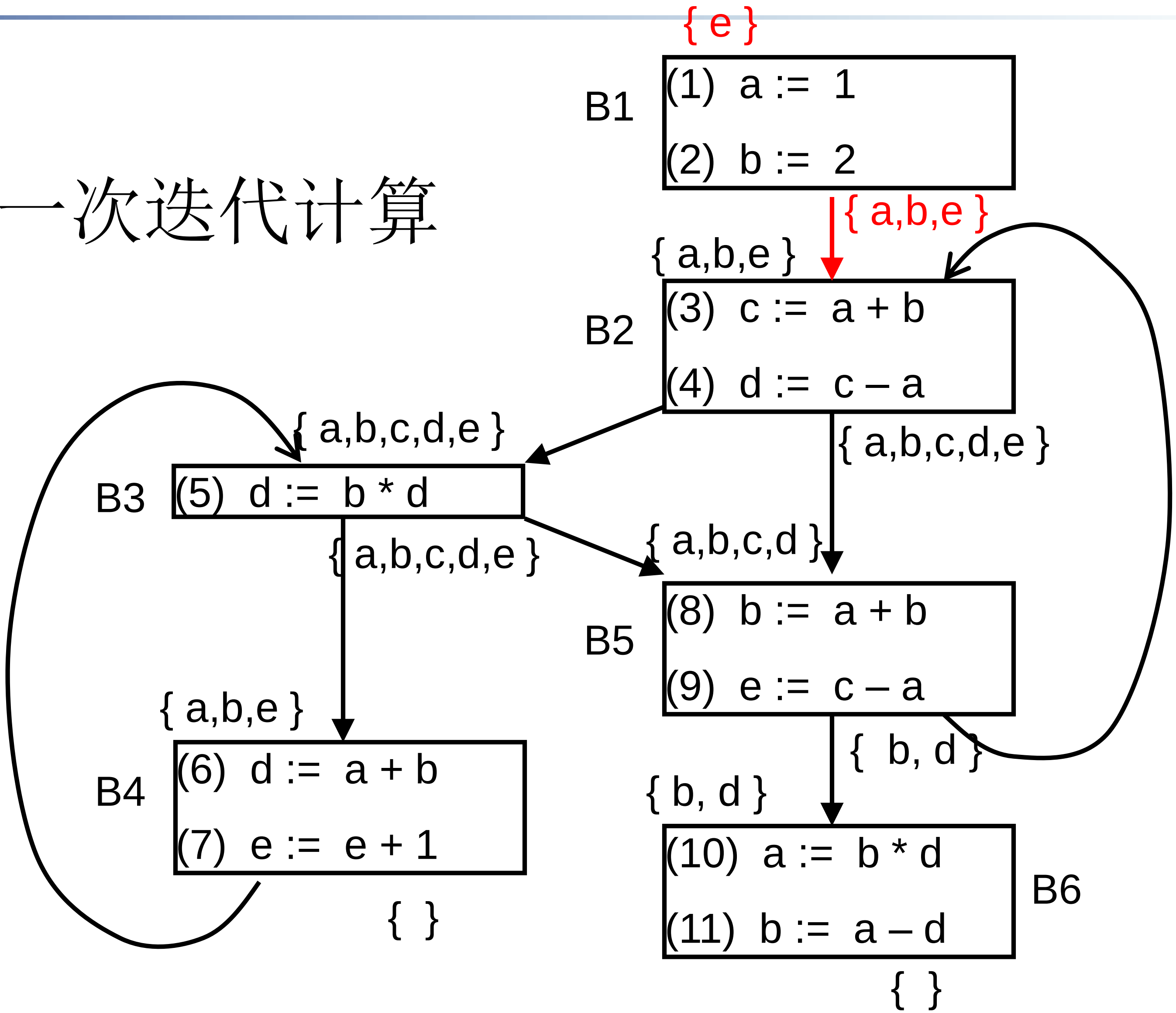
- 第一次迭代计算



B1
(1) a := 1
(2) b := 2
{ a,b,e }
B2
(3) c := a + b
(4) d := c – a
{ a,b,c,d,e }
{ a,b,c,d,e }
B3
(5) d := b * d
{ a,b,c,d,e }
{ a,b,c,d }
B5
(8) b := a + b
(9) e := c – a
{ a,b,e }
B4
(6) d := a + b
(7) e := e + 1
{ }
{ b, d }
{ b, d }
(10) a := b * d
(11) b := a – d
B6
{ }

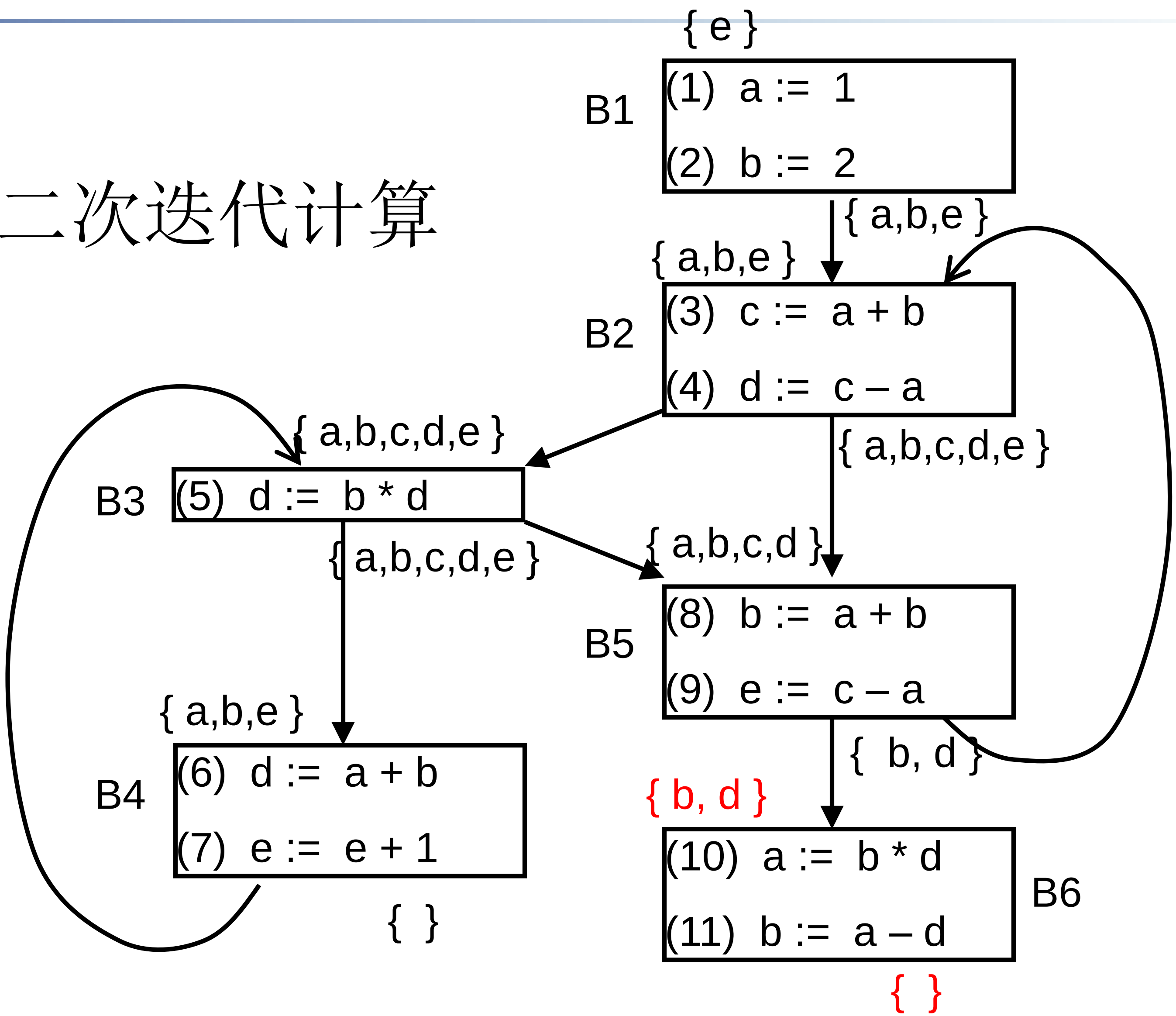
基本块出口活跃变量

- 第一次迭代计算



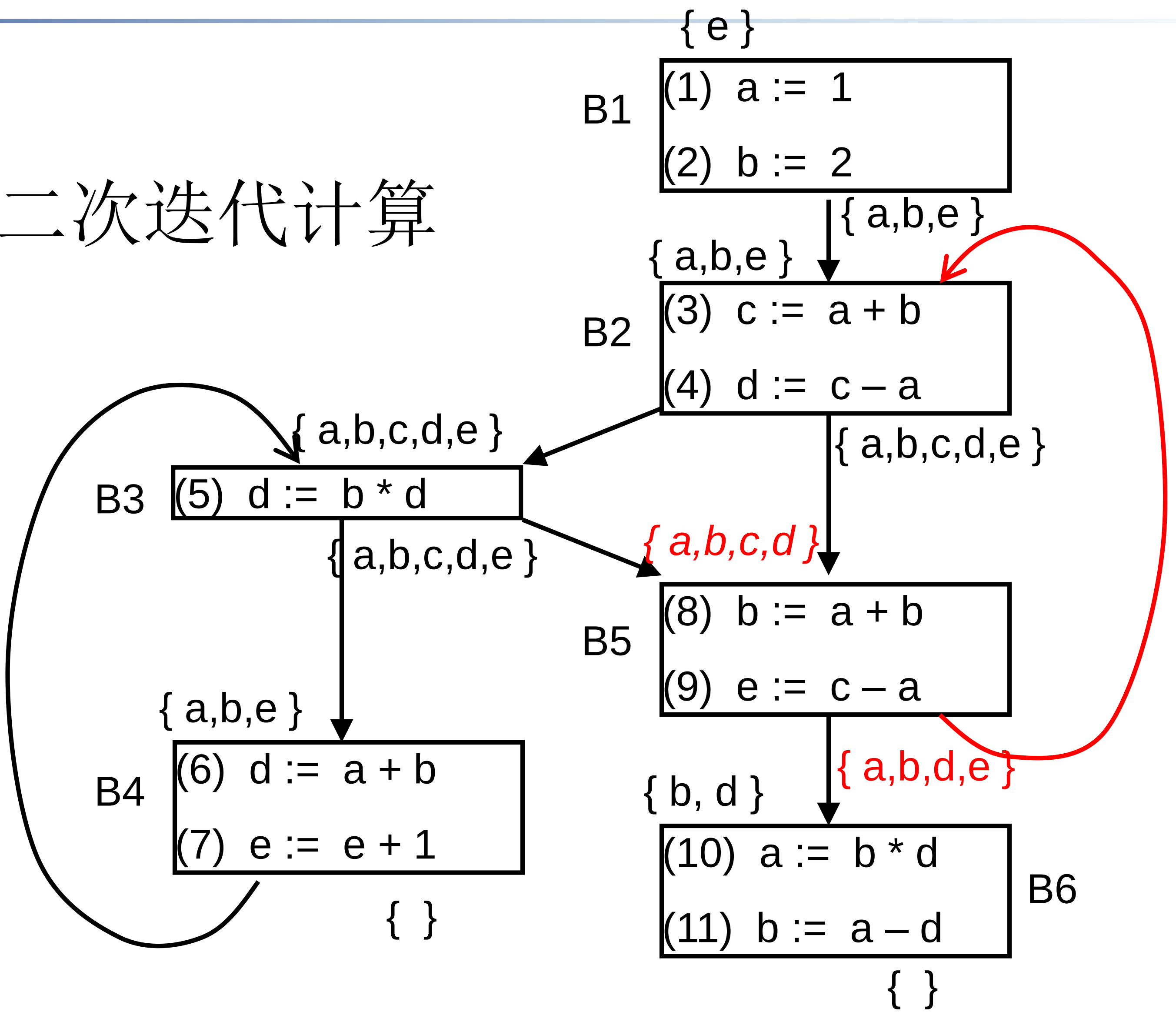
基本块出口活跃变量

- 第二次迭代计算



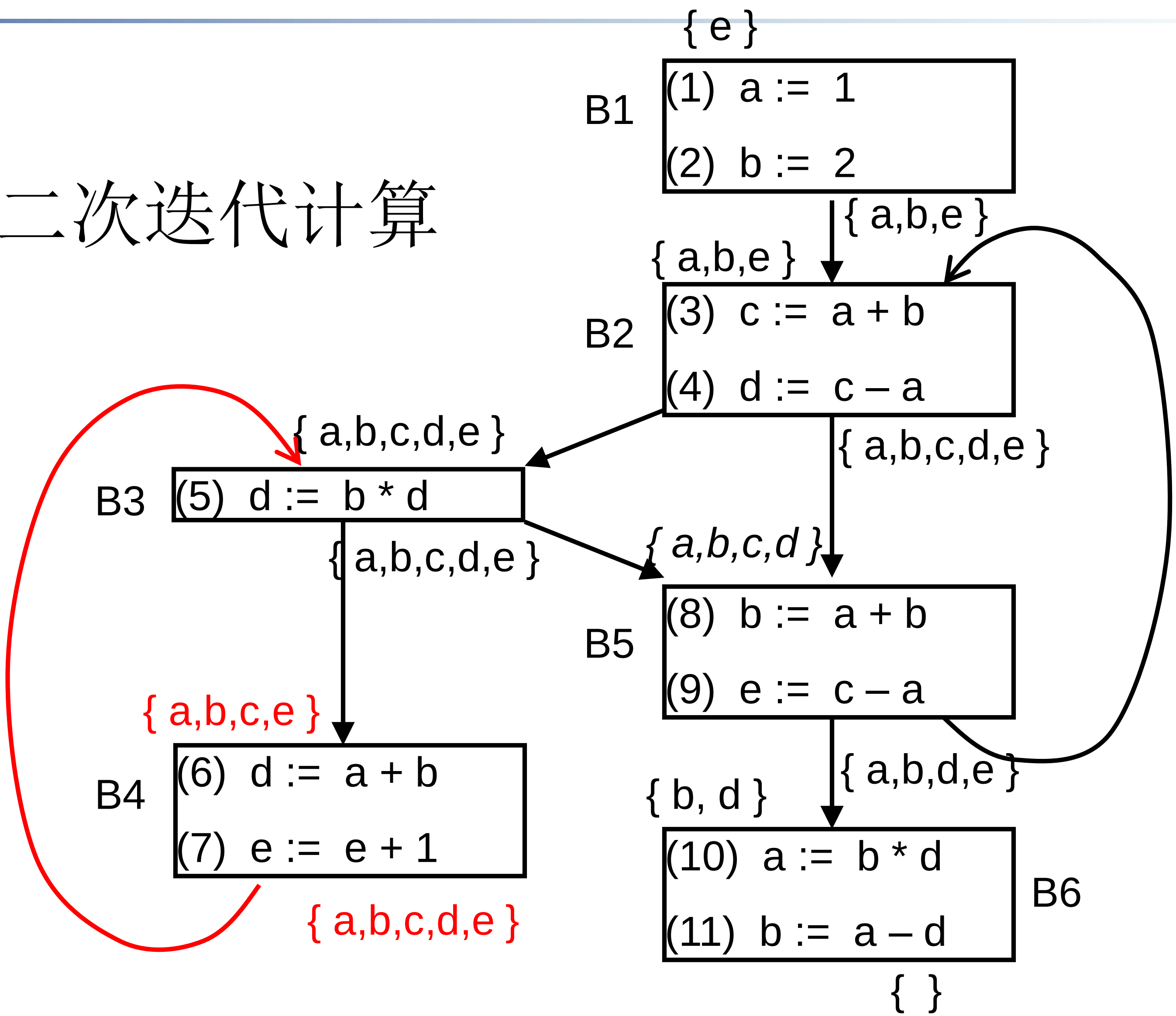
基本块出口活跃变量

- 第二次迭代计算



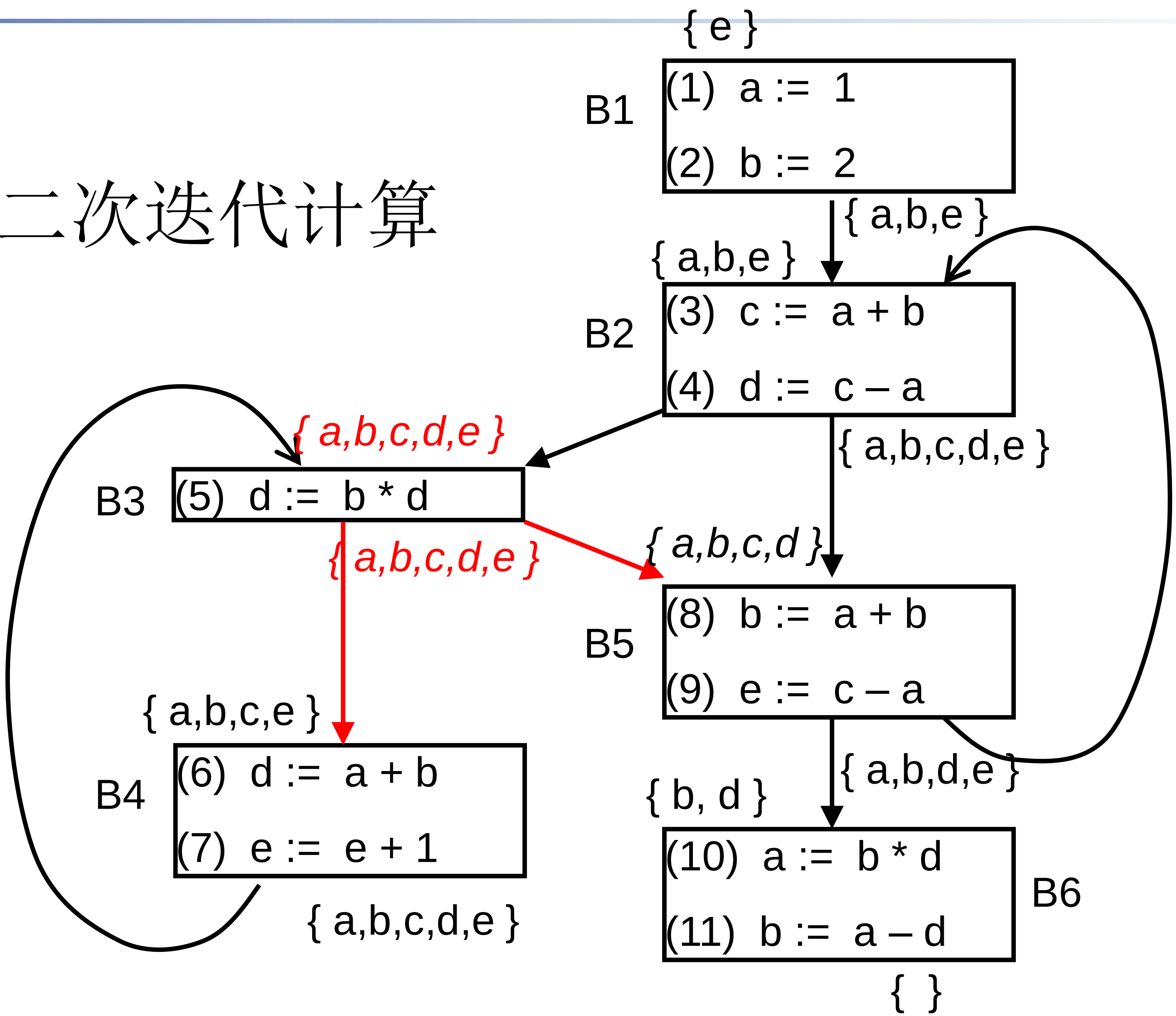
基本块出口活跃变量

- 第二次迭代计算



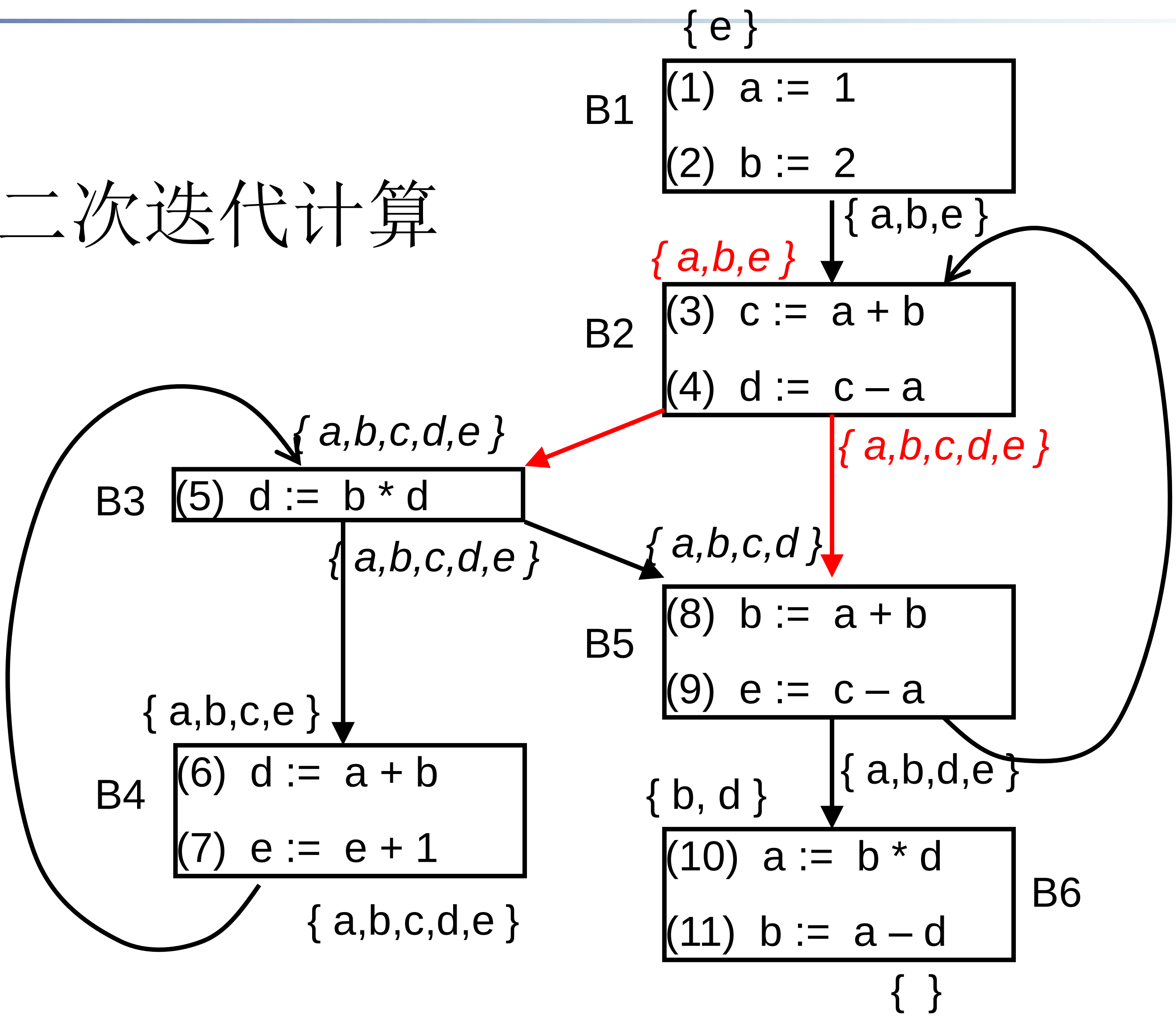
基本块出口活跃变量

- 第二次迭代计算



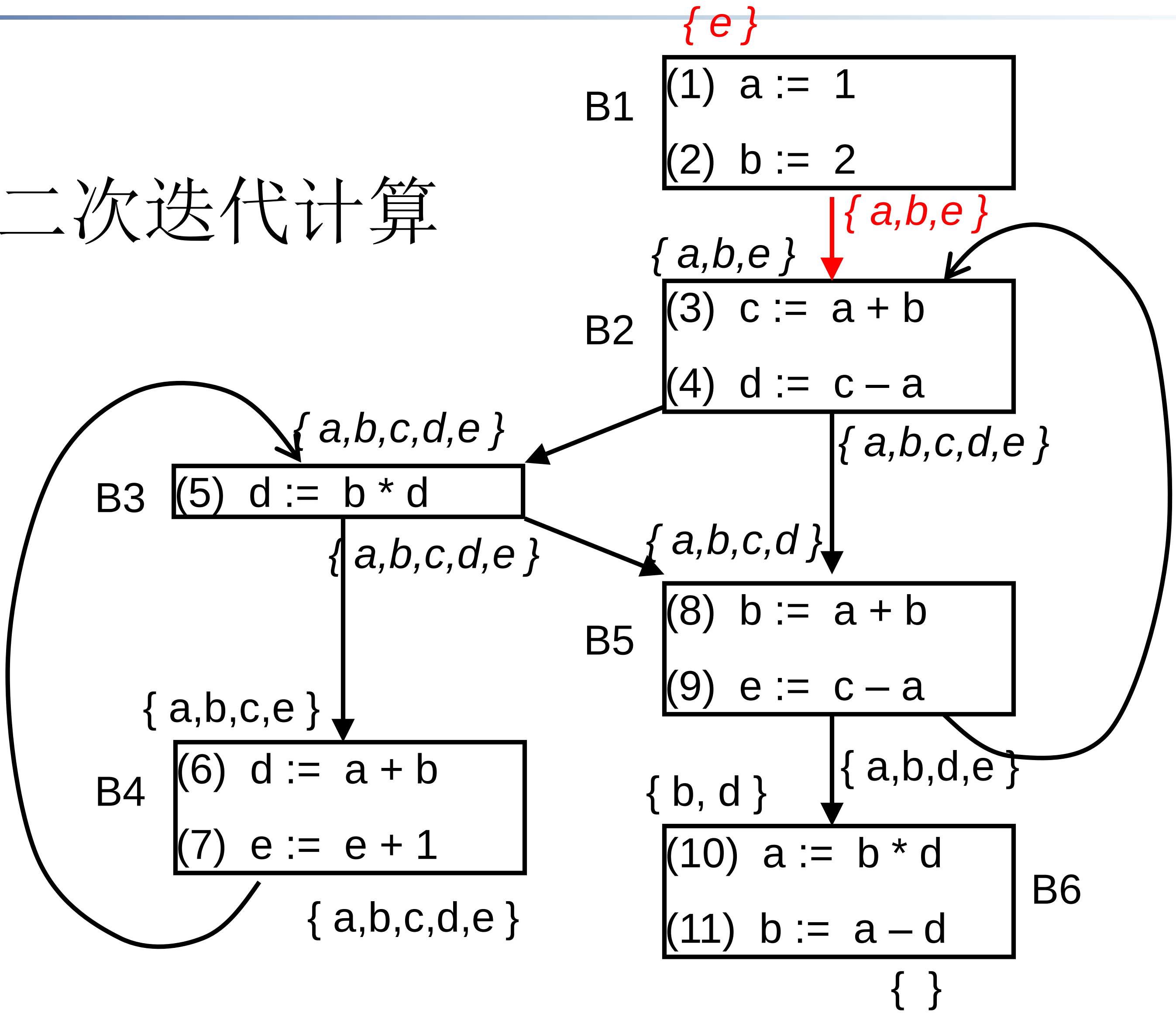
基本块出口活跃变量

- 第二次迭代计算



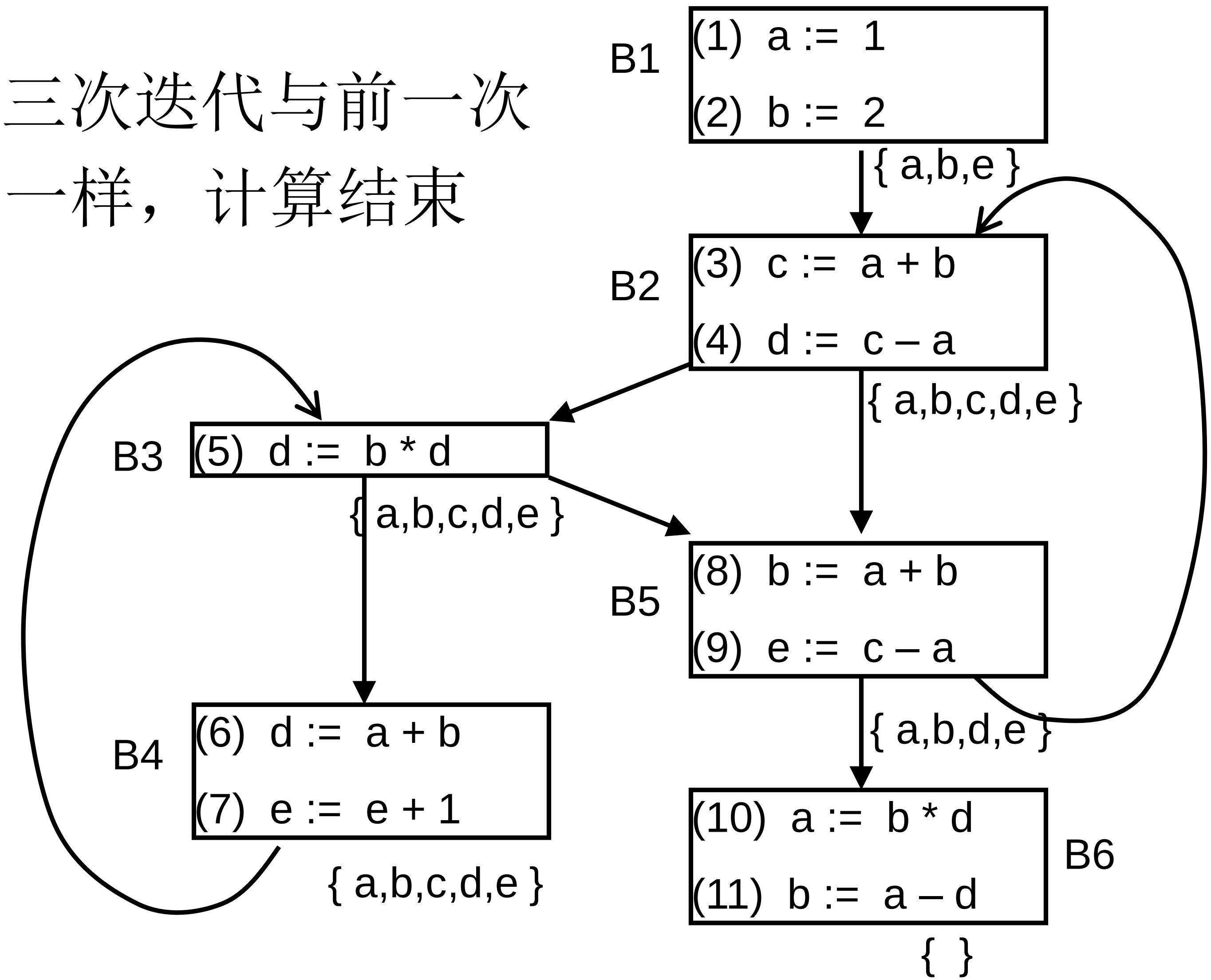
基本块出口活跃变量

- 第二次迭代计算



基本块出口活跃变量

- 第三次迭代与前一次结果一样，计算结束



三种有用的分析

中国科学技术大学 University of Science and Technology of China

- 到达-定值
- 活跃变量
- **可用表达式**

可用表达式

❑定义

- ❖ 若到点*p*的每条执行路径都计算x op y，并且计算后没有对x或y赋值，那么称x op y在点*p*可用
- ❖e_gen_B：块*B*产生的可用表达式集合
- ❖e_kill_B：块*B*注销的可用表达式集合
- ❖IN [*B*]：块*B*入口的可用表达式集合
- ❖OUT [*B*]：块*B*出口的可用表达式集合

可用表达式

x = y + z

.

.

.

p

y + z 在*p*点可用

x = y + z

.

y = …

.

p

y + z 在*p*点不可用

x = y + z

.

z = …

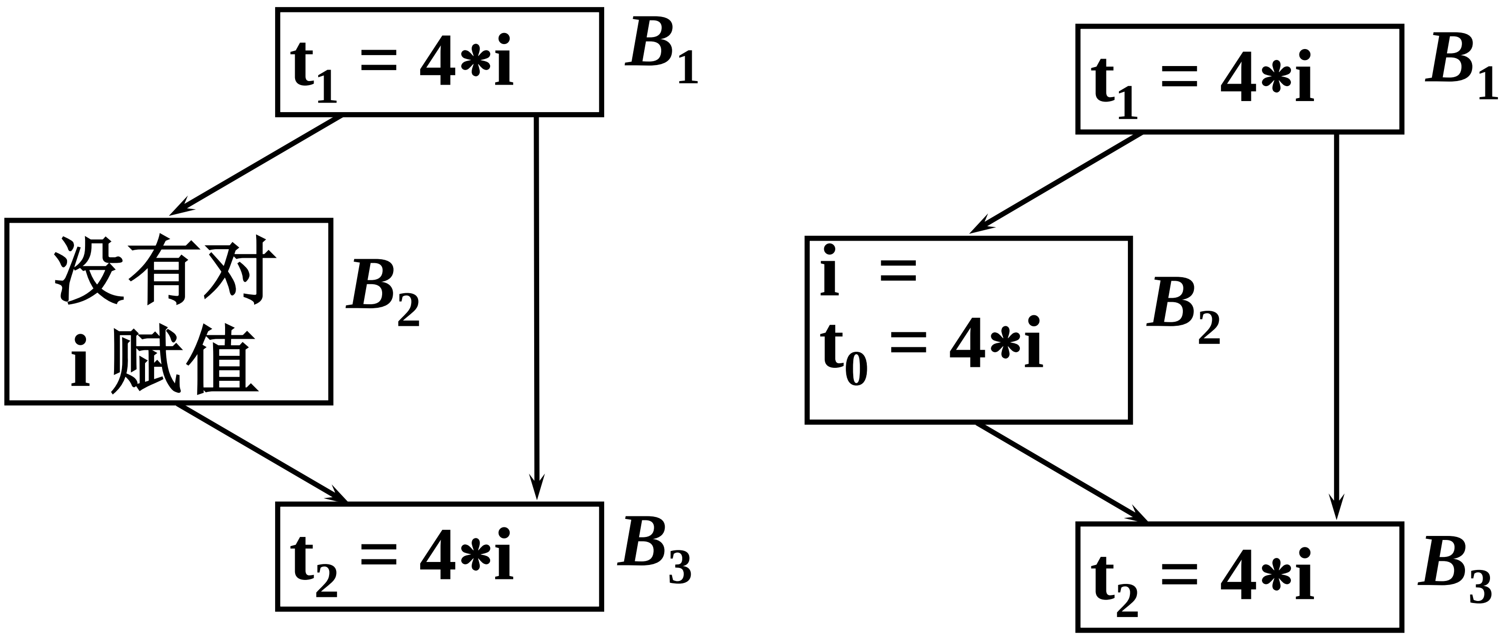
.

p

y + z 在*p*点不可用

消除全局公共子表达式

❖例：下面两种情况下，4*i在B_3的入口都可用



可用表达式

❑数据流等式

- ❖ $\text{OUT}[B] = e\_gen_B \cup (\text{IN}[B] - e\_kill_B)$
- ❖ $\text{IN}[B] = \cap_{P\text{是}B\text{的前驱}} \text{OUT}[P]$
- ❖ $\text{IN}[\text{ENTRY}] = \varnothing$

❑同先前的主要区别

- ❖ 使用∩而不是∪作为这里数据流等式的汇合算符
- ❖ 求最大解而不是最小解

可用表达式

- 基本块生成的表达式：

  基本块中语句d：$x = y + z$的前、后点分别为点p与点q。设在点p处可用表达式集合为S（基本块入口点处S为空集），那么经过语句d之后，在点q处可用表达式集合如下构成：

  (1) $S = S \cup \{ y+z \}$

  (2) $S = S - \{$ S 中所有涉及变量x的表达式 $\}$

  注意，步骤(1)和(2)不可颠倒，x可能就是y或z。

  如此处理完基本块中所有语句后，可以得到基本块生成的可用表达式集合S；

- 基本块杀死的表达式：所有其他类似y+z的表达式，基本块中对y或z定值，但基本块没有**生成**y+z。

示例：基本块生成的表达式

语句	可用表达式
	Ø
a = b + c	{ b + c }
b = a – d	{ a – d } // b+c被杀死
c = b + c	{ a – d } // b+c被杀死
d = a – d	Ø // a – d 被杀死

- **传递方程：**

  $IN[B] = \cap_{P是B的前驱基本块}(OUT[P])$

  $OUT[B] = e\_gen_B \cup (IN[B] - e\_kill_B)$

- **边界值：** $OUT[ENTRY] = ø$； **程序开始，无可用表达式！**
- **迭代算法：**

(1) OUT[ENTRY] = ø

(2) for(除ENTRY之外的每个基本块B) OUT[B]= *U*

(3) while(某个OUT值发生变化) {

(4) for(除ENTRY之外的每个基本块B){

(5) $IN[B] = \cap_{P是B的前驱基本块}(OUT[P])$

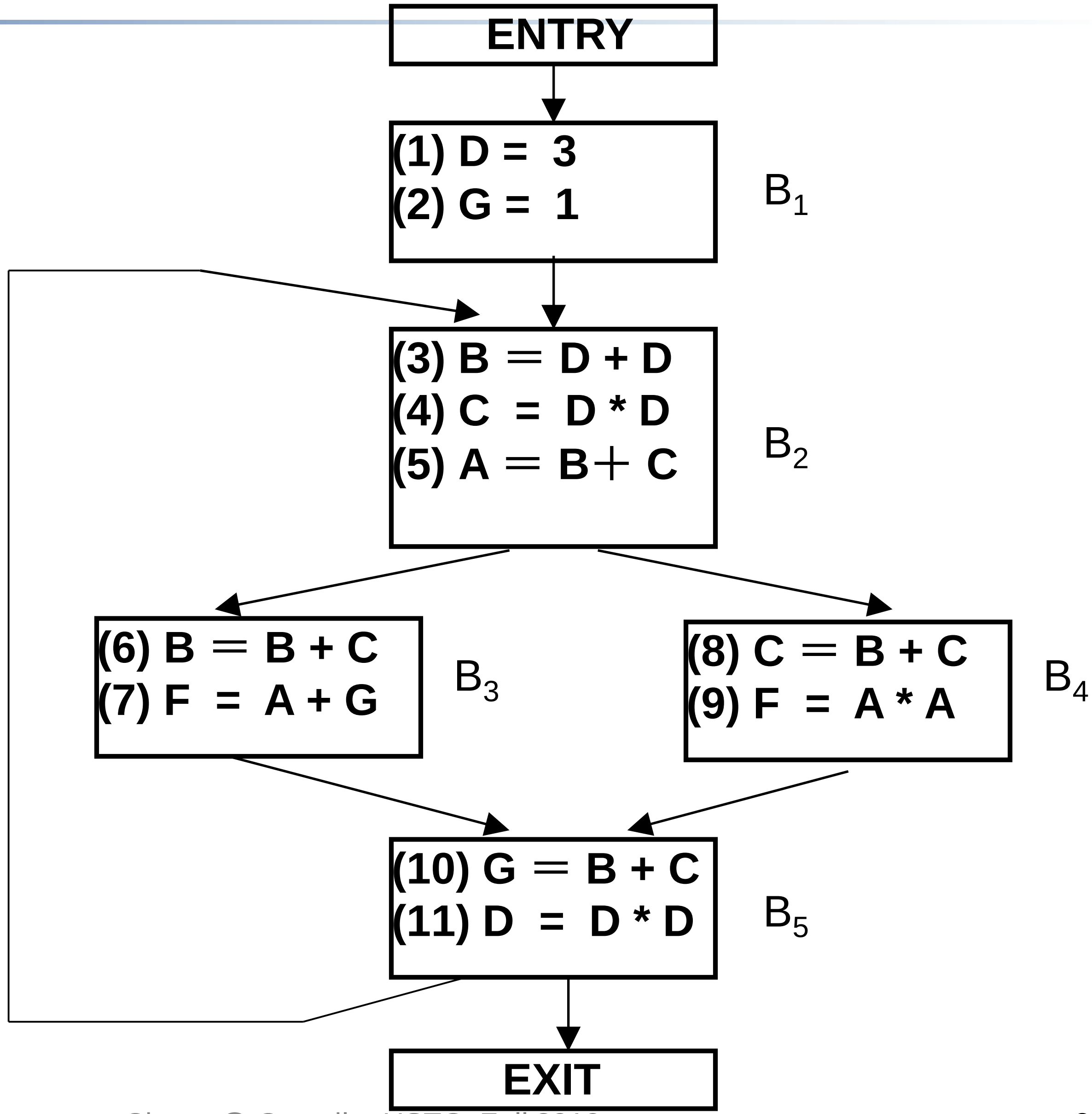
(6) $OUT[B] = e\_gen_B \cup (IN[B] - e\_kill_B)$

} // end-of-for

} // end-of-while

U是全体表达式集合

示例：可用表达式


ENTRY
(1) D = 3
(2) G = 1
B1
(3) B = D + D
(4) C = D * D
(5) A = B + C
B2
(6) B = B + C
(7) F = A + G
B3
(8) C = B + C
(9) F = A * A
B4
(10) G = B + C
(11) D = D * D
B5
EXIT

示例：可用表达式

基本块	前驱	后继
ENTRY	—	B_1
B_1	ENRTY	B_2
B_2	B_1 B_5	B_3 B_4
B_3	B_2	B_5
B_4	B_2	B_5
B_5	B_3 B_4	B_2 EXIT
EXIT	B_5	—

示例：可用表达式

基本块	e_gen	e_kill
ENTRY	Ø	Ø
B_1	{3, 1}	{ D+D, D*D, A+G }
B_2	{ D+D, D*D, B+C }	{ A*A, A+G }
B_3	{ A+G }	{ B+C }
B_4	{ A * A }	{ B+C }
B_5	{ B+C }	{ A+G, D*D, D+D }
EXIT	Ø	Ø
全部表达式*U*={ 3, 1, D+D, D*D, B+C, A+G, A*A }		

示例：可用表达式

- B2块的e_kill集合不包含B+C，因为虽然B和C的赋值改变了B+C的值，但是最后一个语句再次计算了B+C，这样B+C又成为可用表达式。生命力顽强，没有被kill掉。
- 从另一个视角来看，即便是e_kill中包含了B+C，OUT集合计算的时候也会被e_gen中的B+C覆盖掉。

		e_kill
E		Ø
		{ D+D, D*D, A+G }
	}	{ A*A, A+G }
B_3	{ A+G }	{ B+C }
B_4	{ A * A }	{ B+C }
B_5	{ B+C }	{ A+G, D*D, D+D }
EXIT	Ø	Ø
全部表达式*U*={ 3, 1, D+D, D*D, B+C, A+G, A*A }		

示例：可用表达式

- 可用表达式的迭代计算
  - 深度优先序，即 B1 -> B2 -> B3 -> B4 -> B5 -> EXIT
  - 边界值：OUT[ENTRY] = ∅;
  - 初始化：for all NON-ENTRY B: OUT[B] = U ;
- 第一次迭代：（all NON-ENTRY B）

(1) IN[B1] = OUT[ENTRY] = ∅; // B1 前驱仅为ENTRY

OUT[B1] = e_GEN[B1] ∪ (IN[B1] – e_KILL[B1])

= e_GEN[B1] = { 3, 1 } //变化

(2) IN[B2] = OUT[B1] ∩ OUT[B5]

= { 3, 1 } ∩ U = { 3, 1 }

OUT[B2] = e_GEN[B2] ∪ (IN[B2] – KILL[B2])

= { D+D, D*D, B+C } ∪ ({ 3, 1 } – {A*A, A+G })

= {3, 1, D+D, D*D, B+C } //变化

示例：可用表达式

□第一次迭代：（all NON-ENTRY B）

(3) IN[B3] = OUT[B2]

= {3, 1, D+D, D*D, B+C }

OUT[B3] = e_gen[B3] ∪ (IN[B3] – e_kill[B3])

= { A+G } ∪ ({ 3, 1, D+D, D*D, B+C } – { B+C })

= { 3, 1, D+D, D*D, A+G } //变化

(4) IN[B4] = OUT[B2]

= {3, 1, D+D, D*D, B+C }

OUT[B4] = e_gen[B4] ∪ (IN[B4] – e_kill[B4])

= { A * A } ∪({ 3, 1, D+D, D*D, B+C } – { B+C })

= { 3, 1, D+D, D*D, A * A } //变化

示例：可用表达式

❑第一次迭代：（all NON-ENTRY B）

(5) IN[B5] = OUT[B3] ∩ OUT[B4]

= { 3, 1, D+D, D*D, A+G } ∩ { 3, 1, D+D, D*D, A * A }

= { 3, 1, D+D, D*D }

OUT[B5] = e_gen[B5] ∪ (IN[B5] – e_kill[B5])

= {B+C}∪({3,1,D+D, D*D} – {A+G, D*D, D+D})

= { 3, 1, B+C } //变化

(6) IN[EXIT] = OUT[B5] = { 3, 1, B+C }

OUT[EXIT] = e_GEN[EXIT] ∪

(IN[EXIT] –e_KILL[EXIT])

= ∅ ∪ ({ 3, 1, B+C } – ∅)

= { 3, 1, B+C } //变化

□第二次迭代：(all NON-ENTRY B)

(1) IN[B1] = OUT[ENTRY] = ∅;

OUT[B1] = e_GEN[B1] ∪ (IN[B1] – e_KILL[B1])

= e_GEN[B1] = { 3, 1 } // 不变

(2) IN[B2] = OUT[B1] ∩ OUT[B5]

= { 3, 1 } ∩ { 3, 1, B+C } = { 3, 1 } // 不变

OUT[B2] = e_GEN[B2] ∪ (IN[B2] – e_KILL[B2])

= { D+D, D*D, B+C } ∪

({ 3, 1 } – {A*A, A+G })

= {3, 1, D+D, D*D, B+C } // 不变

□第二次迭代：（all NON-ENTRY B）

(3) IN[B3] = OUT[B2]

= {3, 1, D+D, D*D, B+C } //不变

OUT[B3] = e_gen[B3] ∪ (IN[B3] – e_kill[B3])

= { A+G } ∪ ({ 3, 1, D+D, D*D, B+C } – { B+C })

= { 3, 1, D+D, D*D, A+G } //不变

(4) IN[B4] = OUT[B2]

= {3, 1, D+D, D*D, B+C } //不变

OUT[B4] = e_gen[B4] ∪ (IN[B4] – e_kill[B4])

= { A * A } ∪({ 3, 1, D+D, D*D, B+C } – { B+C })

= { 3, 1, D+D, D*D, A * A } //不变

示例：可用表达式

□第二次迭代：（all NON-ENTRY B）

(5) IN[B5] = OUT[B3] ∩ OUT[B4]

= { 3, 1, D+D, D*D, A+G } ∩ { 3, 1, D+D, D*D, A * A }

= { 3, 1, D+D, D*D } //不变

OUT[B5] = e_gen[B5] ∪ (IN[B5] – e_kill[B5])

= {B+C}∪({3,1,D+D, D*D} – {A+G, D*D, D+D})

= { 3, 1, B+C } //不变

(6) IN[EXIT] = OUT[B5] = { 3, 1, B+C } //不变

OUT[EXIT] = e_GEN[EXIT] ∪

(IN[EXIT] –e_KILL[EXIT])

= ∅ ∪ ({ 3, 1, B+C } – ∅)

= { 3, 1, B+C } //不变

中国科学技术大学
University of Science and Technology of China

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
独立于机器的优化 II

The end!