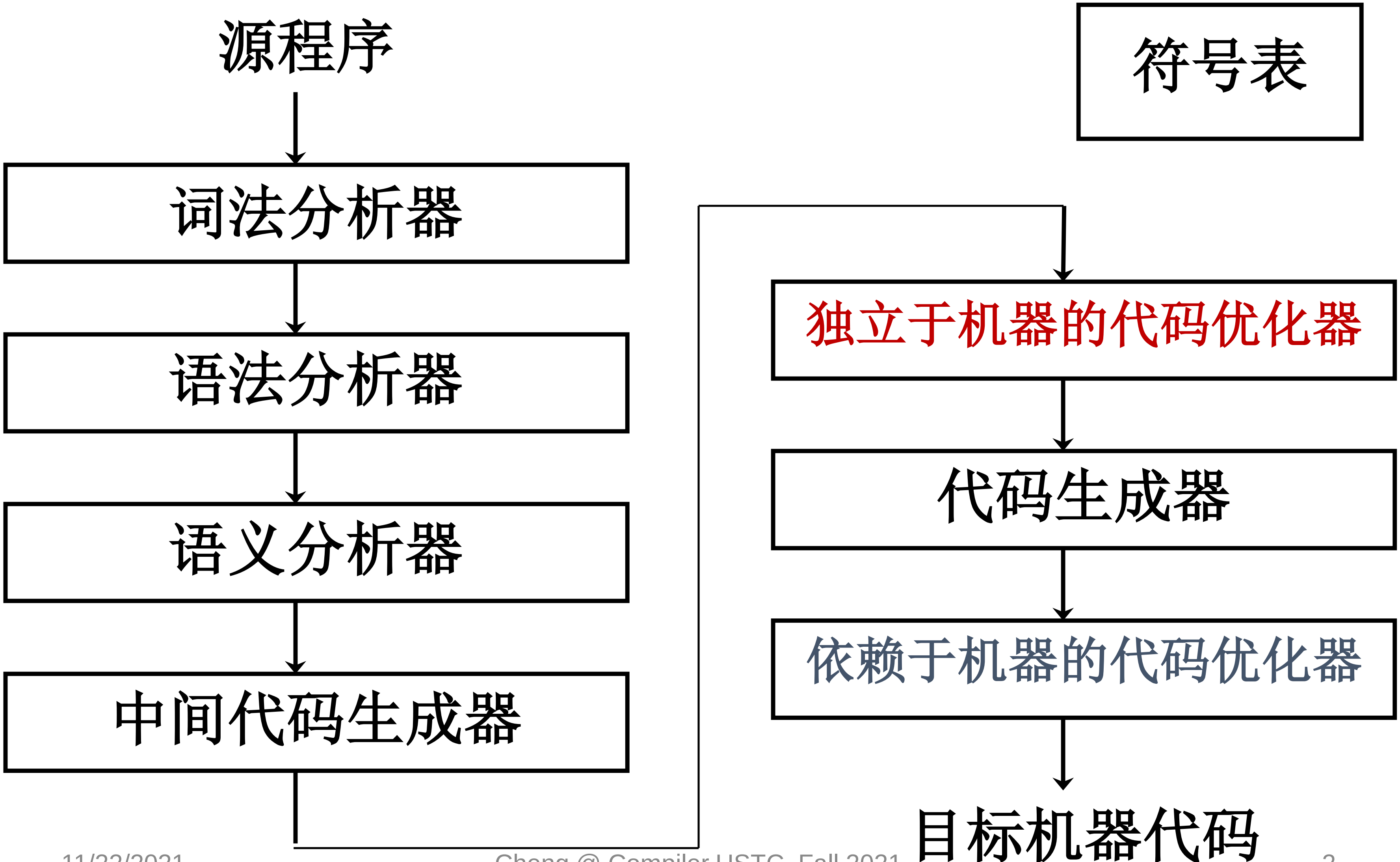
《编译原理与技术》
独立于机器的优化 II

计算机科学与技术学院
李 诚

2021-11-15

独立于机器的优化


源程序
符号表
词法分析器
语法分析器
语义分析器
中间代码生成器
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

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

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

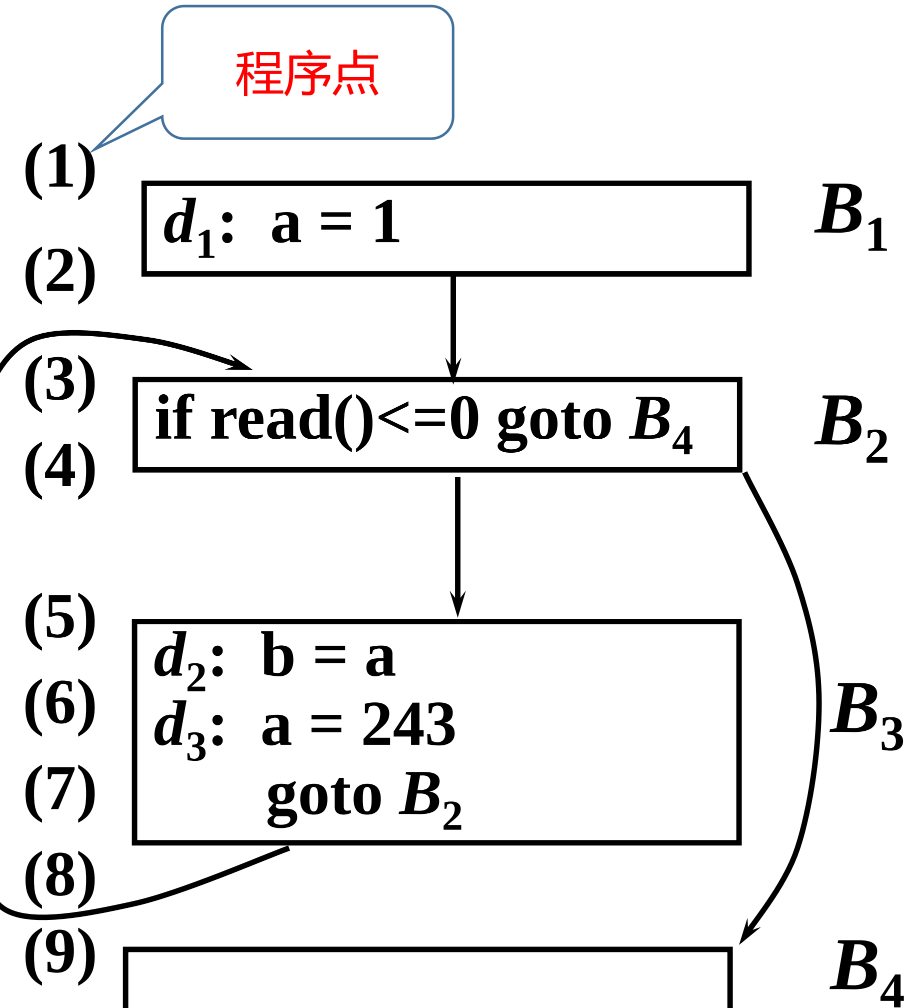
(1) p_i是先于一个语句的点，p_{i+1}是同一块中位于该语句后的点，或者

(2) p_i是某块的结束点，p_{i+1}是后继块的开始点

数据流抽象

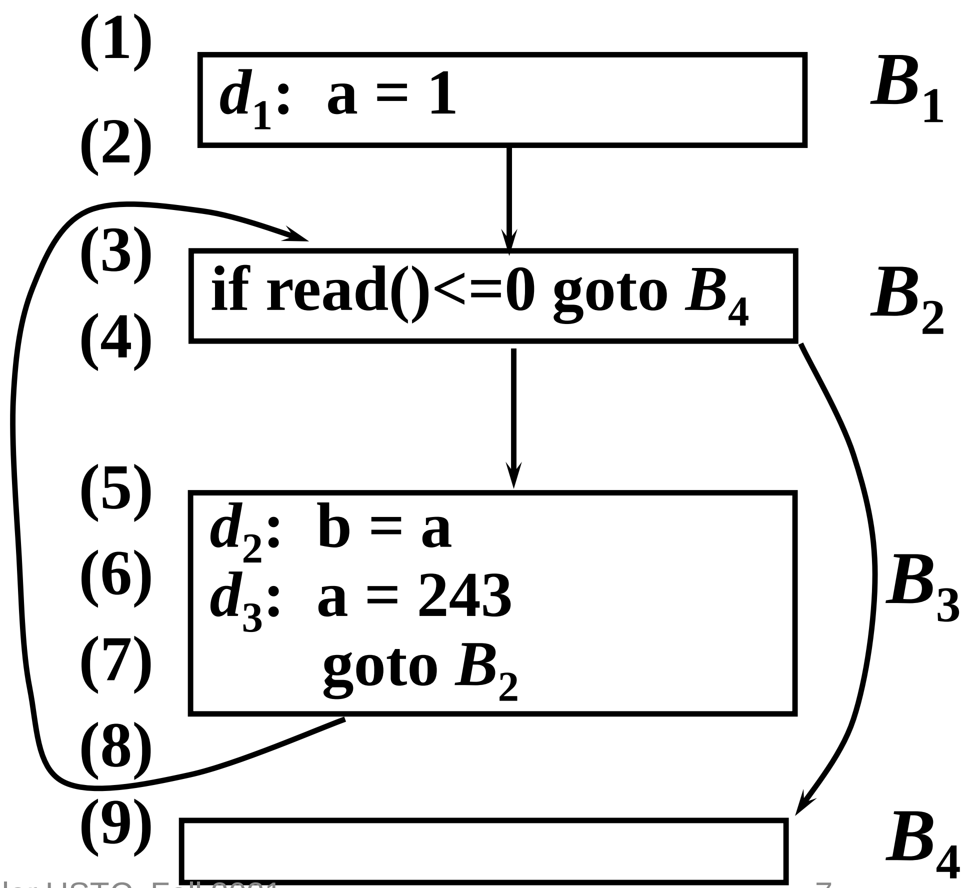
❑流图上路径实例

- (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**

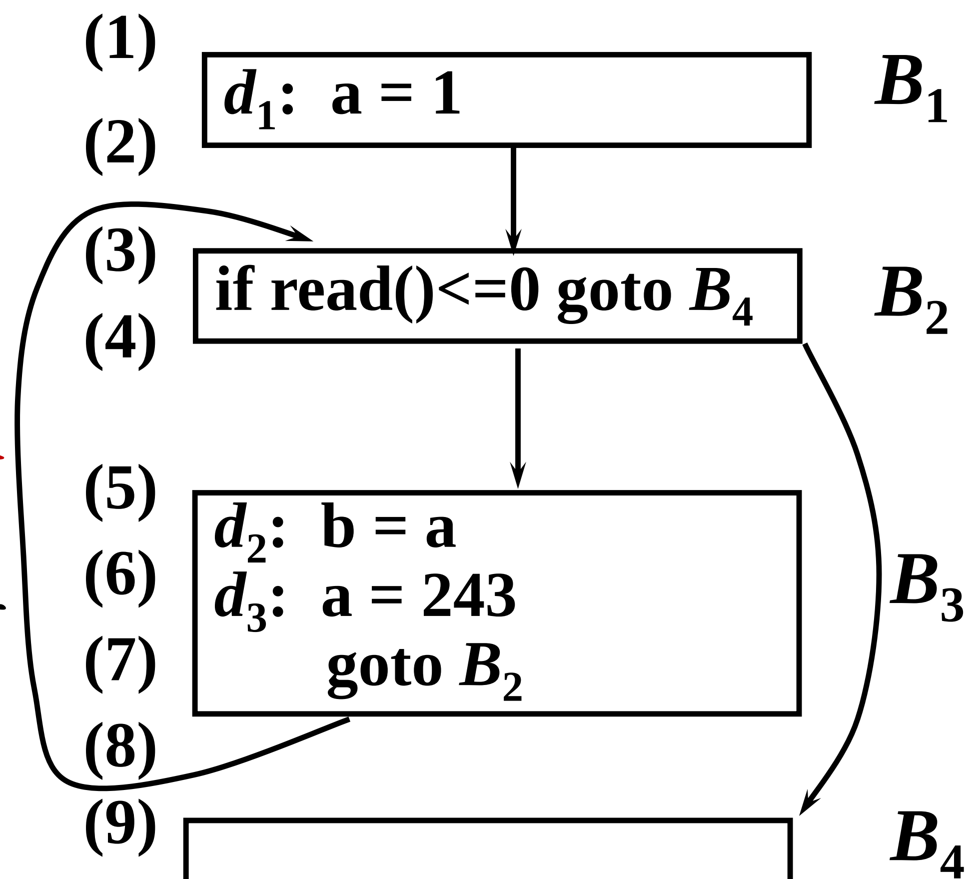


数据流分析介绍

- 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



- 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OUT[s] = f_s(IN[s])
- ❖若沿执行路径逆向传播，则IN[s] = f_s(OUT[s])

若基本块B由语句$s_1, s_2, \ldots, s_n$依次组成，则

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

- ❑可用表达式
- ❑到达-定值
- ❑活跃变量

可用表达式

$x = y + z$

.

.

.

p

$y + z$ 在 p 点

可用

$x = y + z$

.

$y = \ldots$

.

p

$y + z$ 在 p 点

不可用

$x = y + z$

.

$z = \ldots$

.

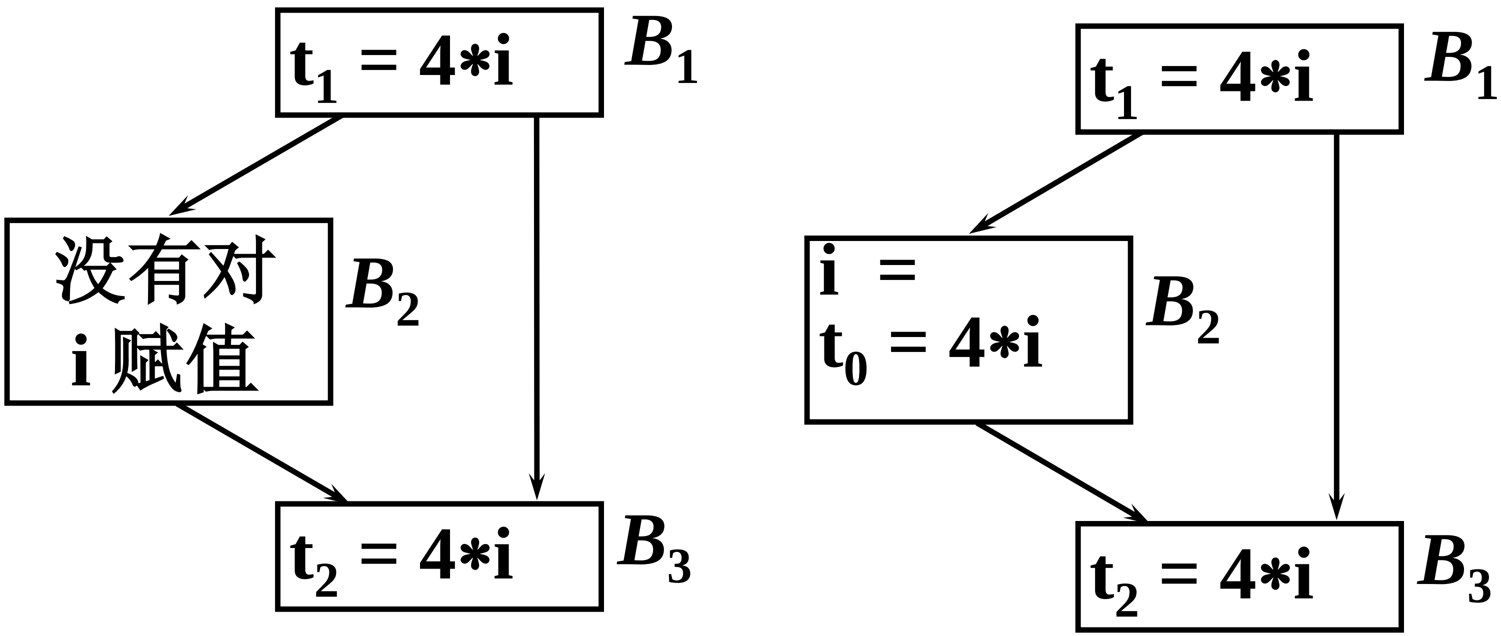
p

$y + z$ 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

University of Science and Technology of China

❑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 ∪ { 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

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
	∅
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	∅ // a – d 被杀死

❑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

❑数据流等式

- ❖ $\text{OUT}\ [B] = e\_gen_B \cup (\text{IN}\ [B] - e\_kill_B\ )$
- ❖ $\text{IN}\ [B] = \cap_{P\text{是}B\text{的前驱}} \text{OUT}\ [P]$
- ❖ $\text{OUT}\ [\text{ENTRY}] = \varnothing$
  - ➢在ENTRY的出口处没有可用表达式

❑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
- ❖只有当一个表达式在B的所有前驱的结尾处都可用，那么它才会在B的开头可用
- ❖ 求最大解而不是最小解

U是全体表达式集合

❑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){

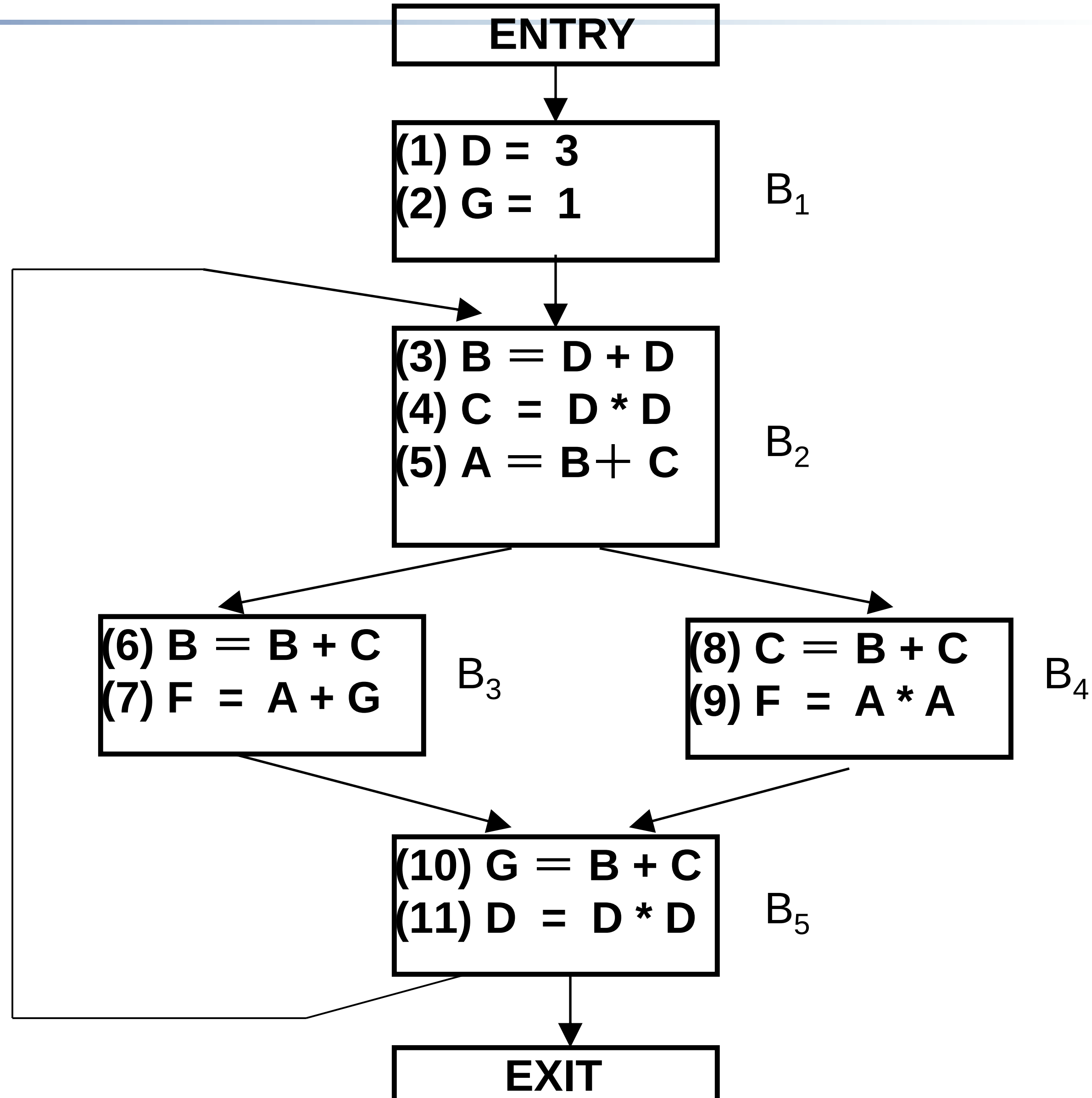
(5) IN[B] = ∩ $_{\text{P是B的前驱基本块}}$(OUT[P])

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

} // end-of-for

} // end-of-while

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		∅
		{ 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	∅	∅
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] – e_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

示例：可用表达式

❑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

三种有用的分析

- 可用表达式
- 到达-定值
- 活跃变量

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

到达-定值分析

□定值与引用

d :　　x := y + z　// 语句d 是变量x的一个定值点

ud链，即引用一定值链。
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

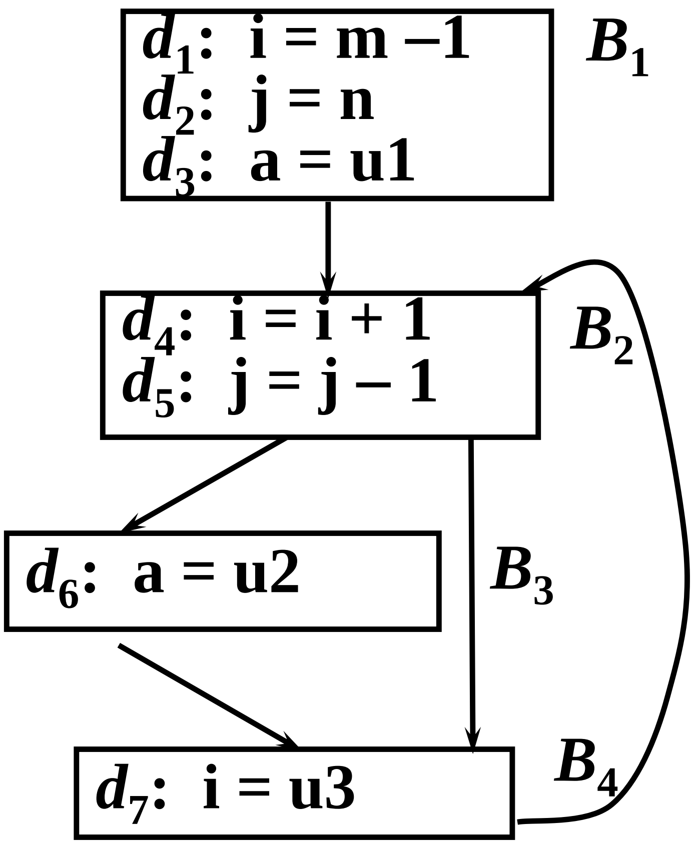
- *gen*和*kill*分别表示一个基本块生成和注销的定值

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

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

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

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



❑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
  $$f_B(x) = gen_B \cup (x - kill_B)$$
  $$kill_B = kill_1 \cup kill_2 \cup \ldots \cup kill_n$$
  $$gen_B = gen_n \cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup \ldots \cup (gen_1 - kill_2 - kill_3 - \ldots - kill_n)$$

❑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

到达-定值的迭代计算算法

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

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

(7) }}

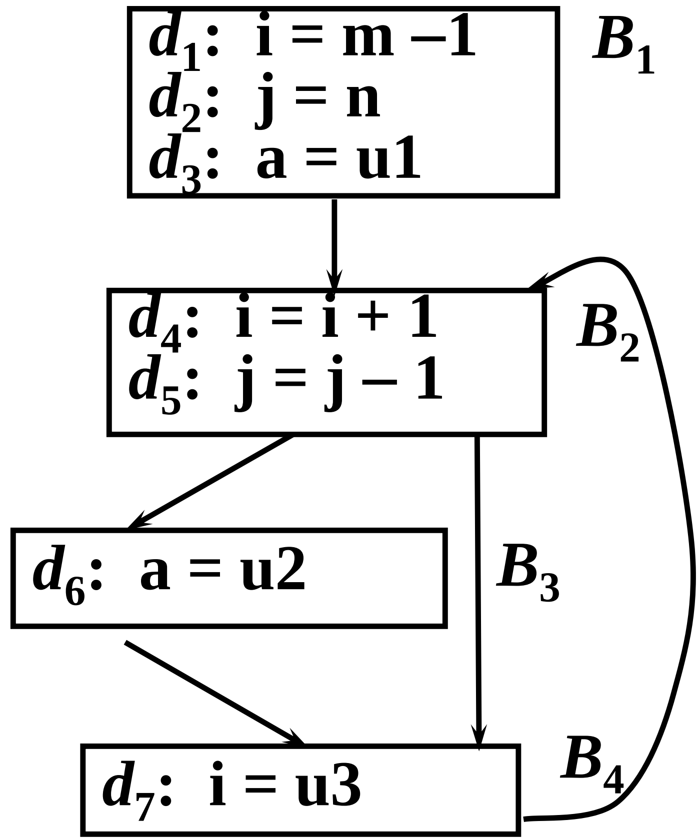
向量求解：集合并操作使用逻辑或，集合相减使用后者求补再逻辑与

到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

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

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



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

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

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

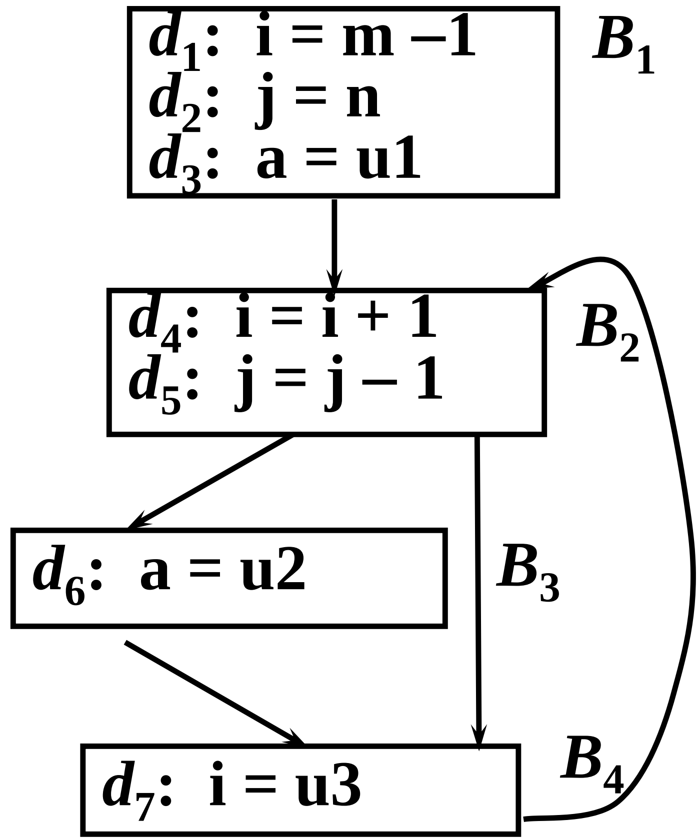
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

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



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

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

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

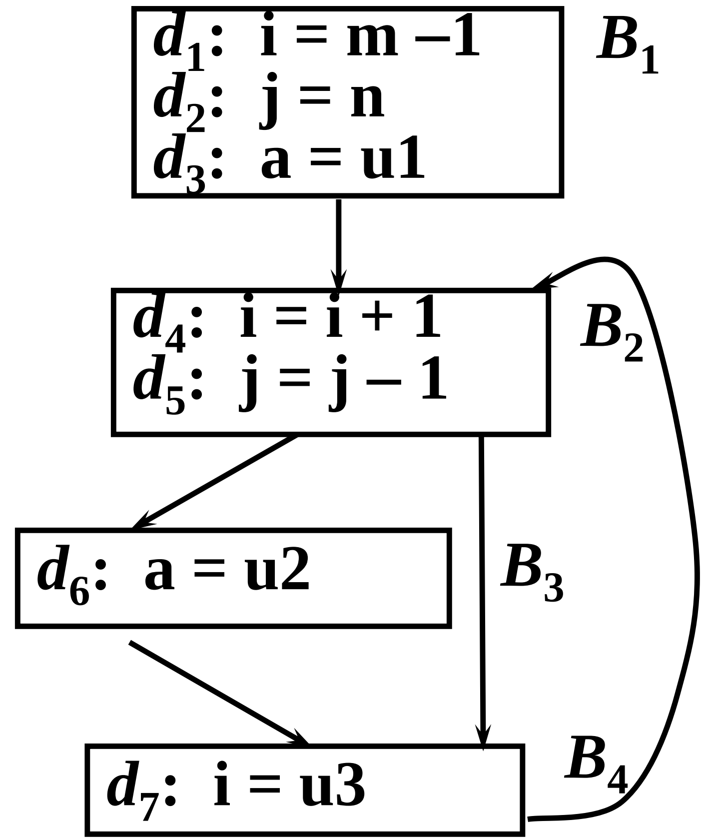
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

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		000 0000
B_3		000 0000
B_4		000 0000



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

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

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

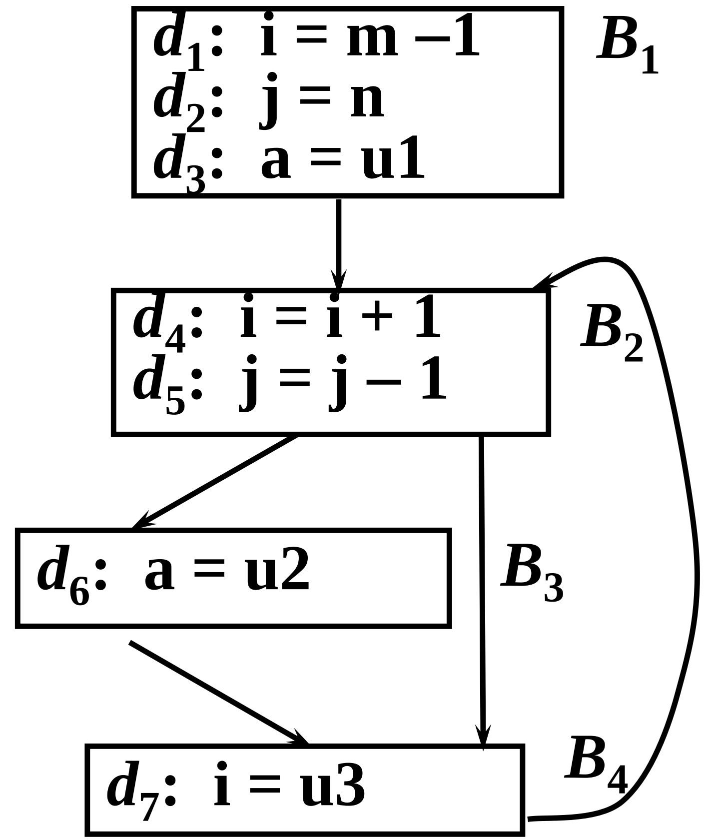
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

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	111 0000	000 0000
B_3		000 0000
B_4		000 0000



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

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

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

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

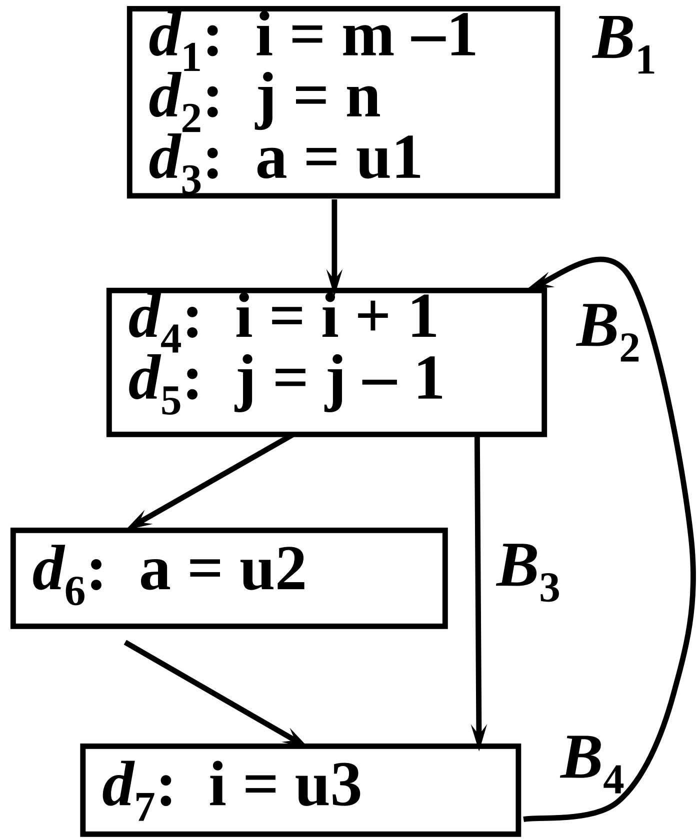
到达-定值分析

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

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

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

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



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

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

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

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

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

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

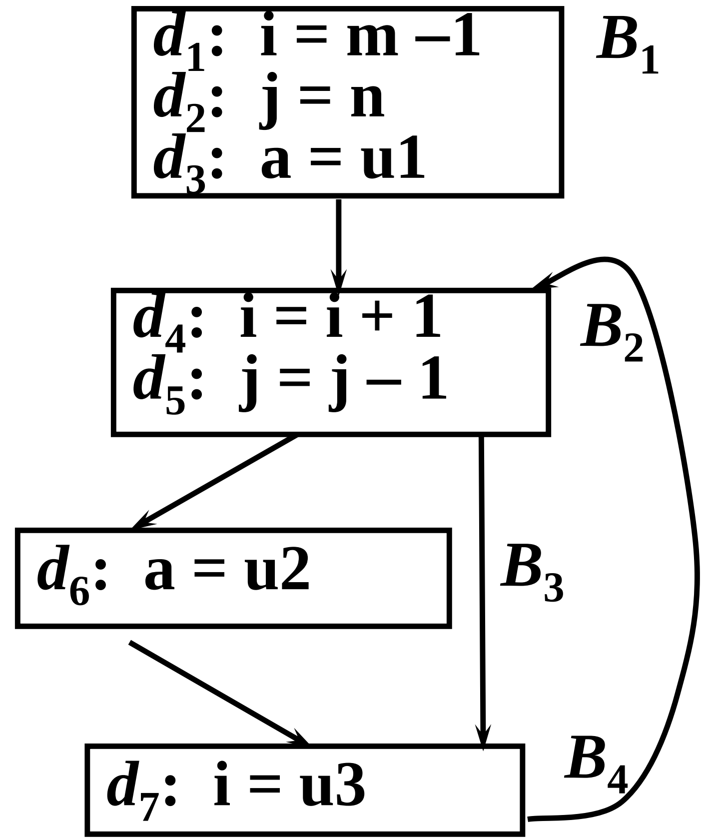
$gen\,[B_4] = \{d_7\}$

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

$IN[B] = \cup_{P是B的前驱} OUT[P]$

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

	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



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

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

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

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

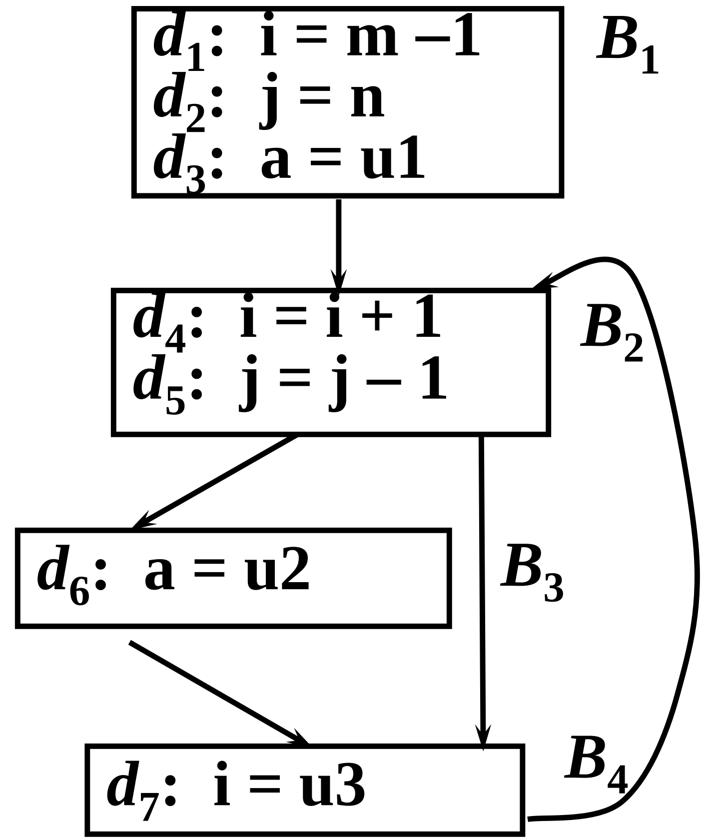
到达-定值分析

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

$IN[B] = \cup_{P是B的前驱} OUT[P]$

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

	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



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

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

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

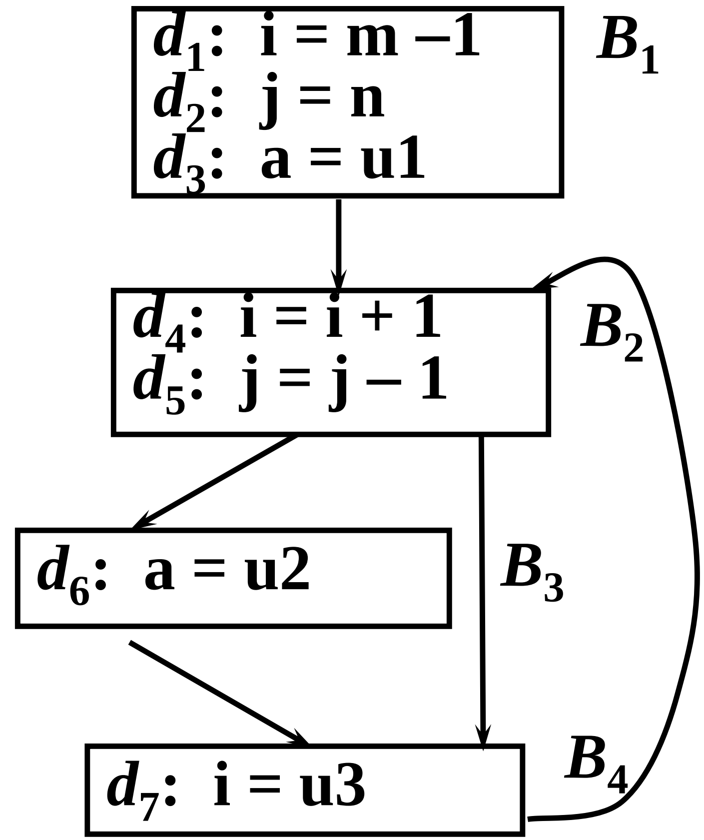
$gen\ [B_4] = \{d_7\}$
$kill\ [B_4] = \{d_1, d_4\}$

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



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

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

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

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

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

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

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

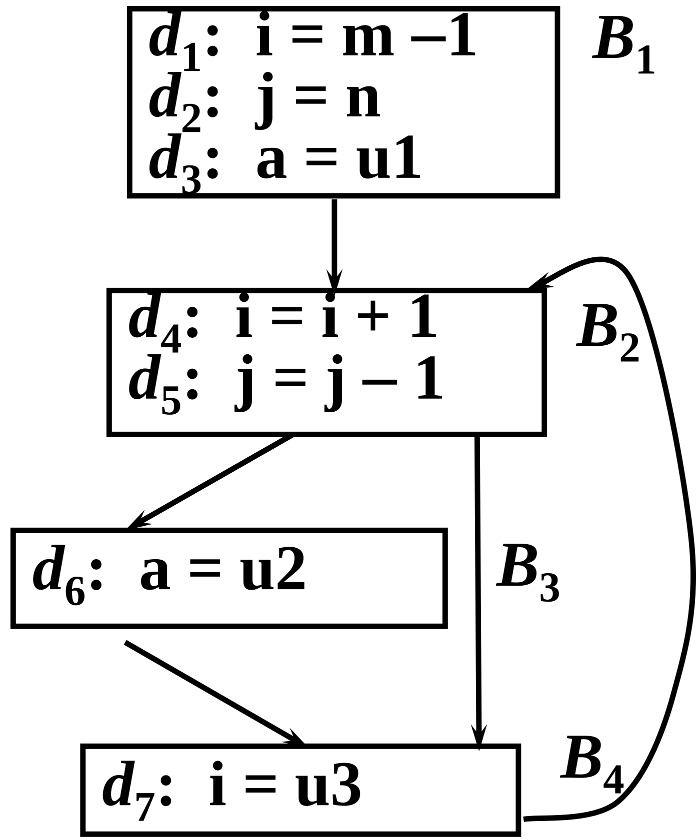
$kill\ [B_4] = \{d_1, d_4\}$

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



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

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

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

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

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

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

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

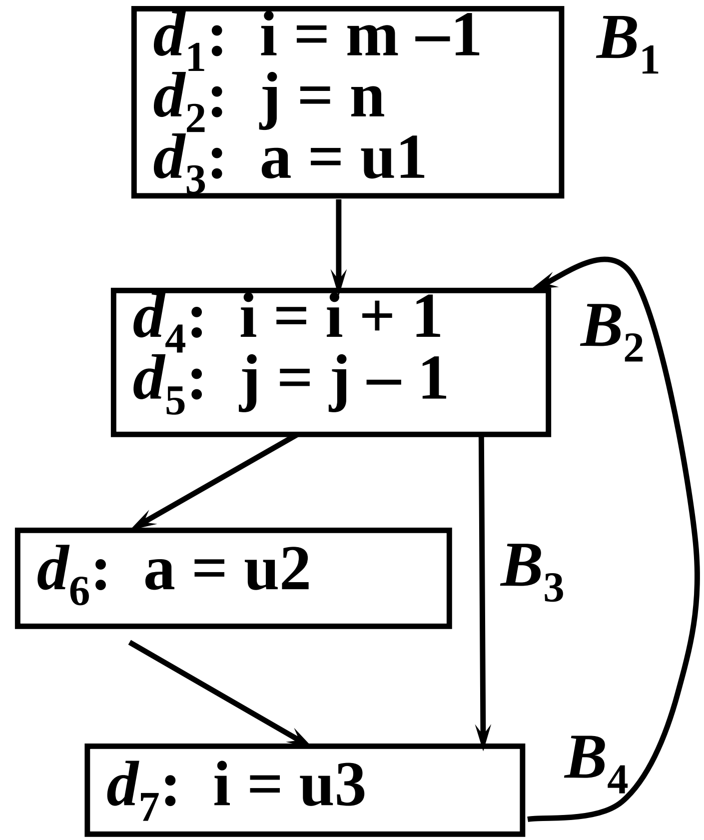
$kill\ [B_4] = \{d_1, d_4\}$

到达-定值分析

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

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

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



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

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

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

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

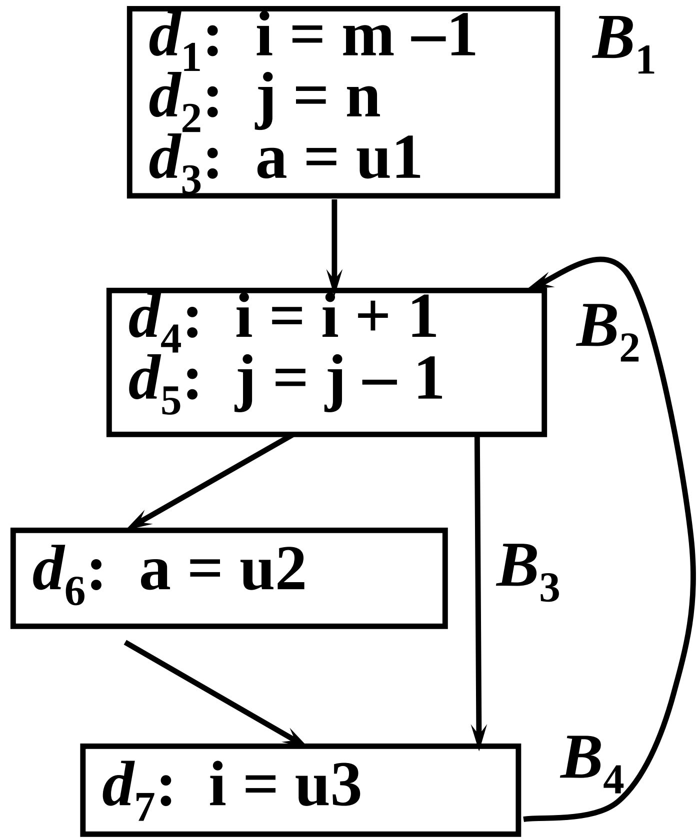
到达-定值分析

$IN[B] = \cup_{P是B的前驱} OUT[P]$

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

	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

—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。

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

- 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

- ❑可用表达式
- ❑到达-定值
- ❑活跃变量

活跃变量分析的应用

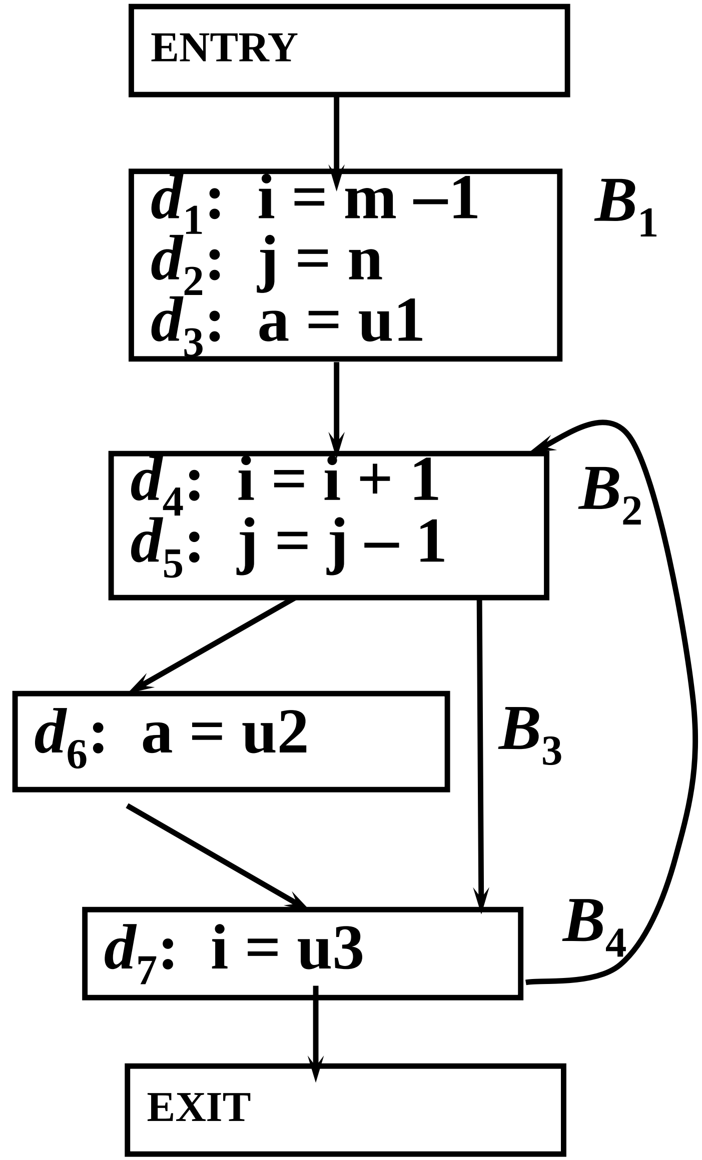
- 删除无用赋值
- 为基本块分配寄存器
  - 如果所有寄存器都被占用，且还需要申请一个寄存器，则应该考虑使用已经存放死亡值的寄存器
  - 如果一个值在基本块结尾处是死的，就不必在结尾处保存这个值了

□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

❑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 [EXIT] = ∅
  - ➢ 边界条件：程序出口处没有活跃变量
- ❖IN [*B*] = use_B ∪ (OUT [*B*] – def_B)
  - ➢入口处活跃：1）在B中重定值之前被使用；2)离开时活跃且没有在B中被定值
- ❖ OUT[*B*] = ∪ $_{S\text{是}B\text{的后继}}$ IN [*S*]

❑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) {

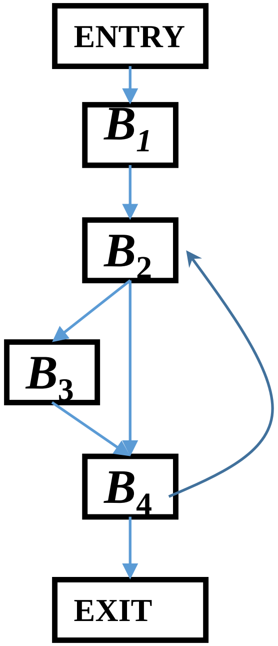
for (除了EXIT以外的每个块B) {

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

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

}}

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) {

$OUT[B] = \cup_{S是B的后继} IN[S]$

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

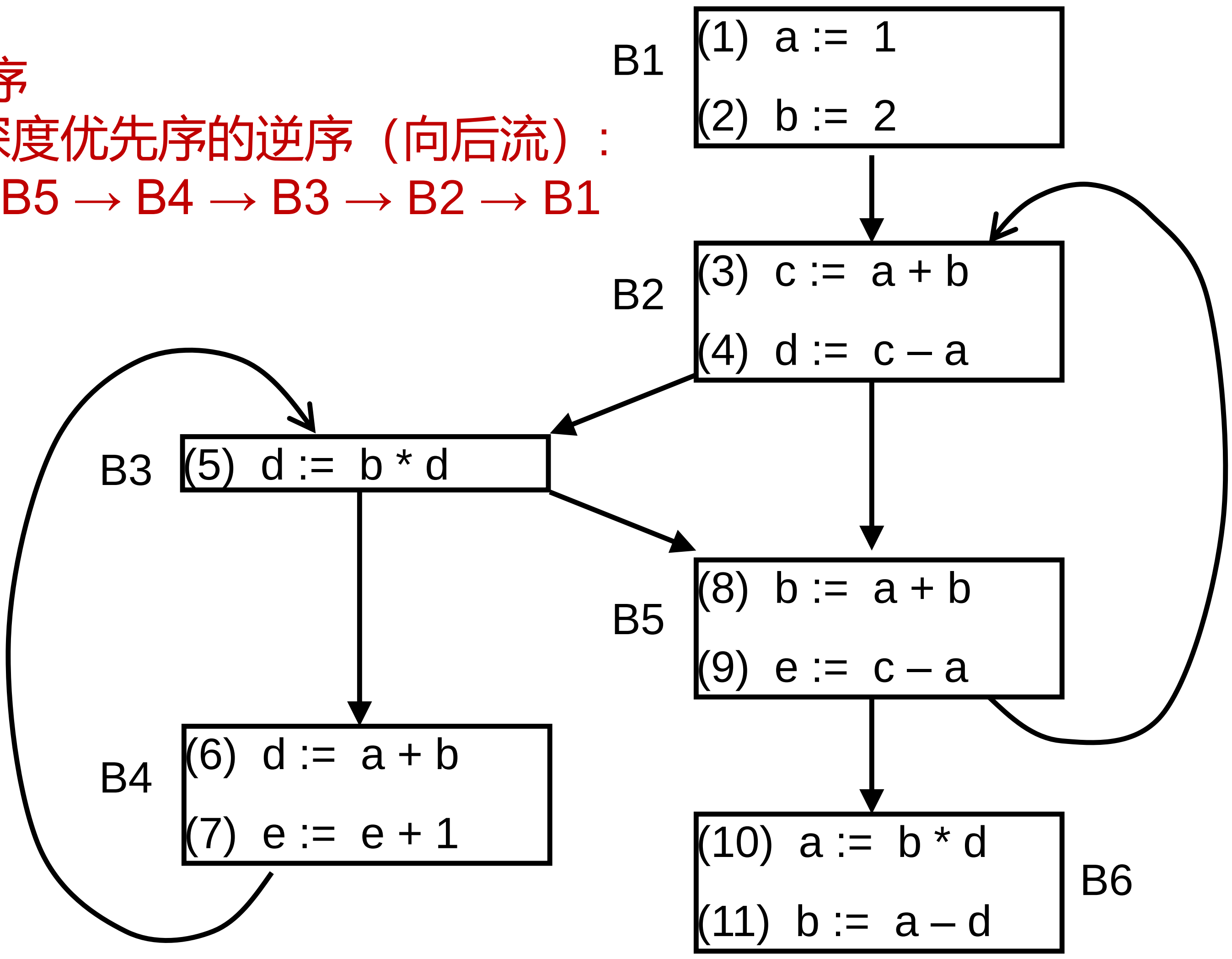
}}

	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 }

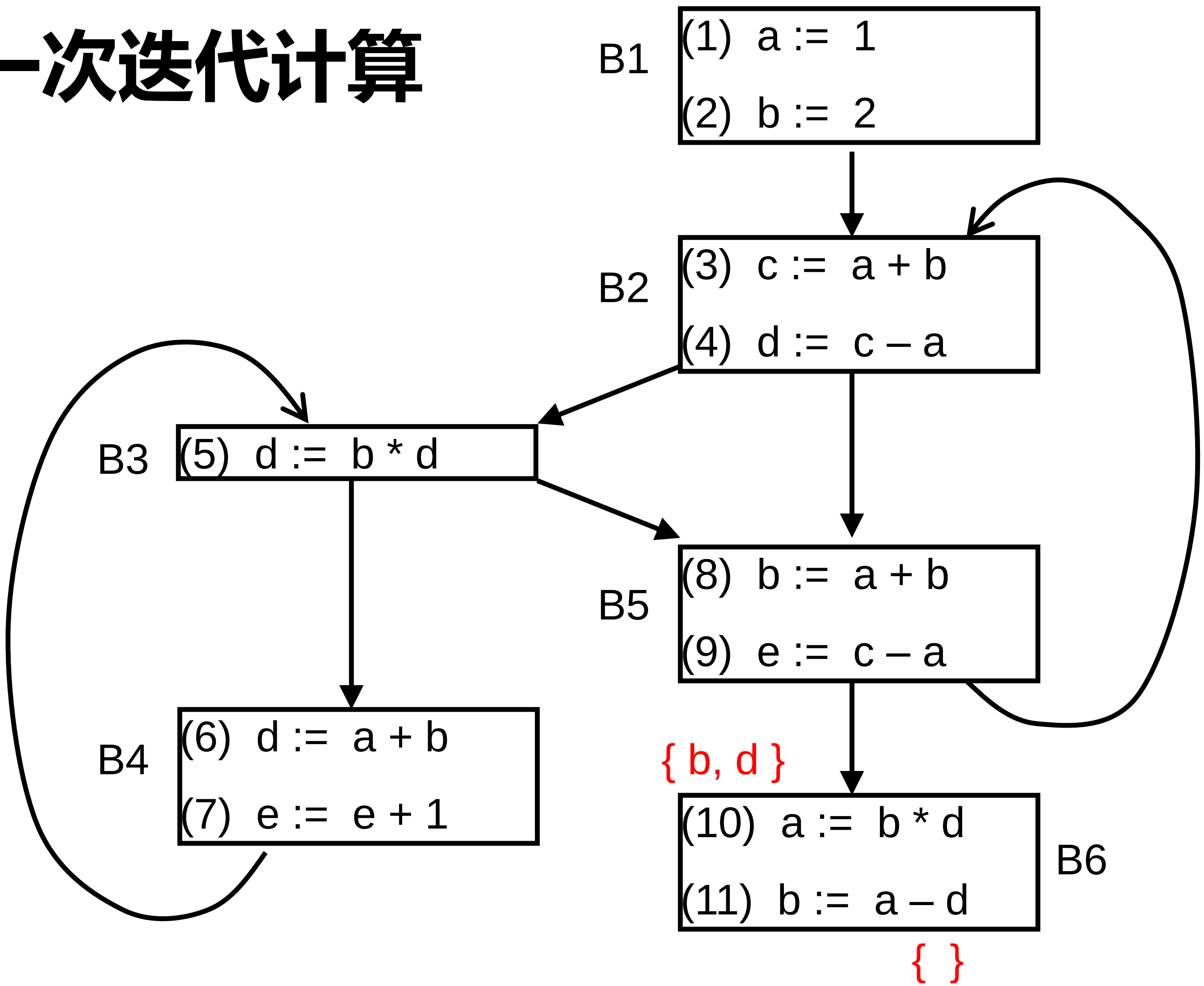
  USE[B6] = { b, d } ; DEF[B6] = { a }

- 初始值，all B, IN[B] = { },

  OUT[B6]={ }//出口块

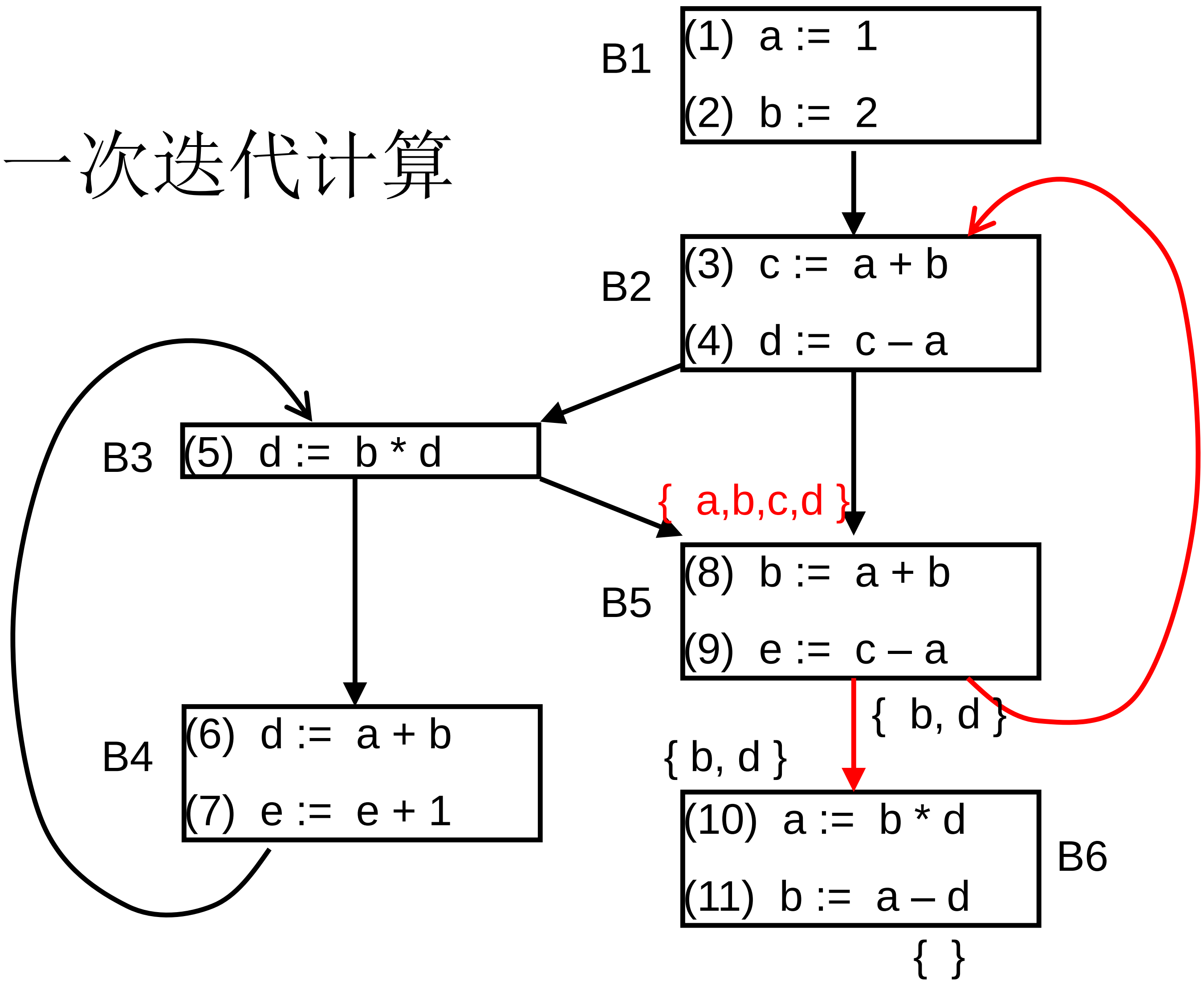
基本块出口活跃变量

□第一次迭代计算



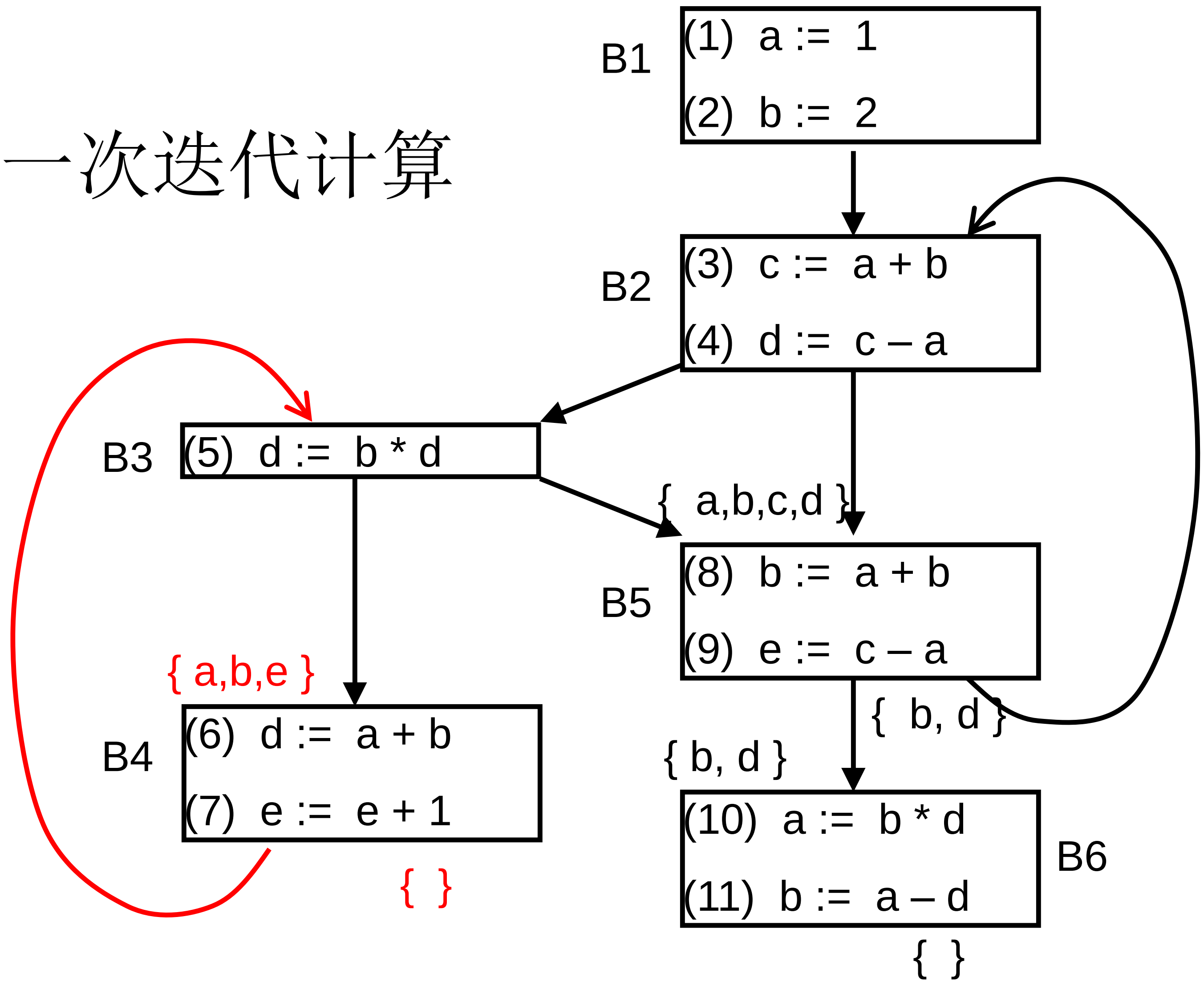
基本块出口活跃变量

- 第一次迭代计算



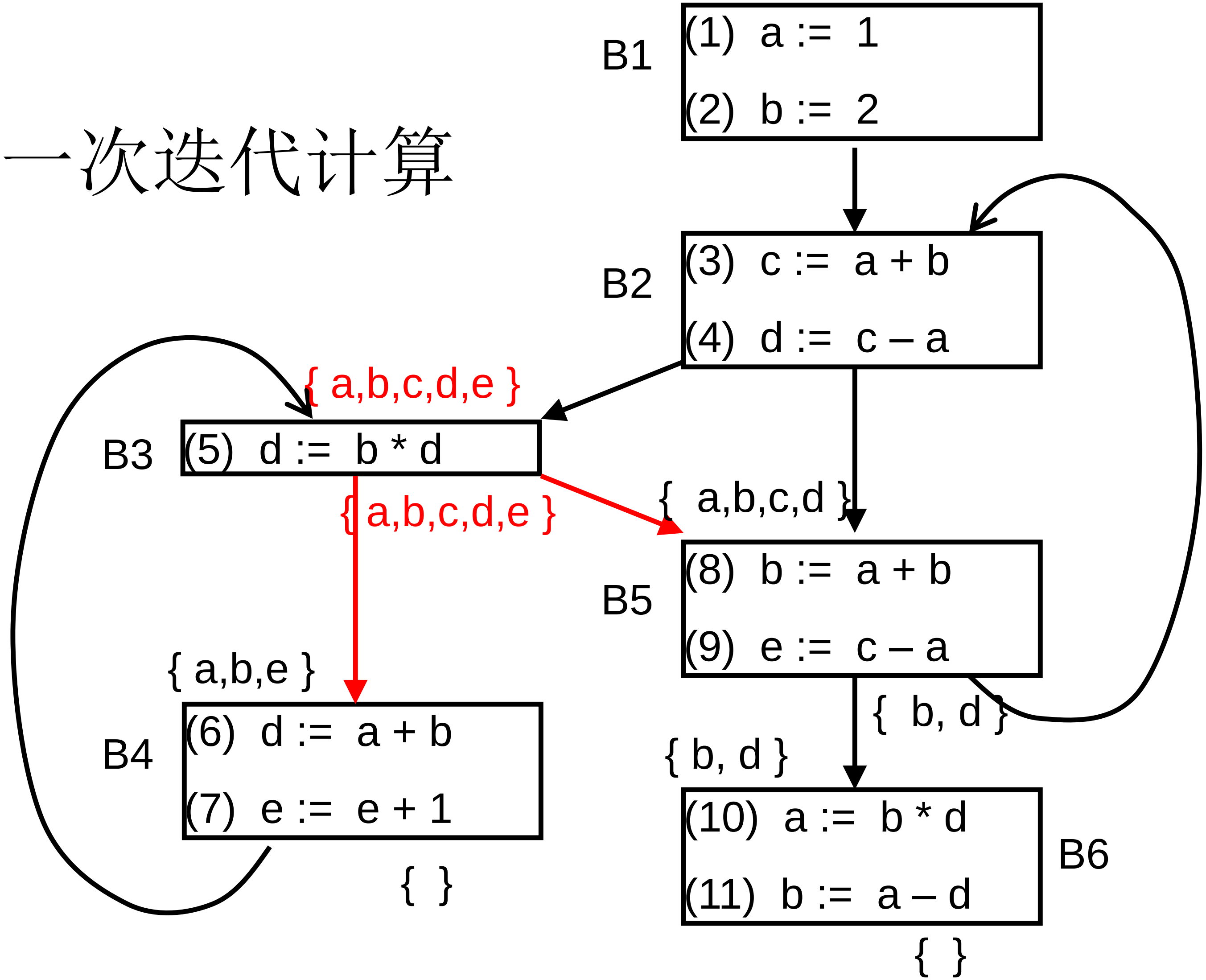
基本块出口活跃变量

- 第一次迭代计算



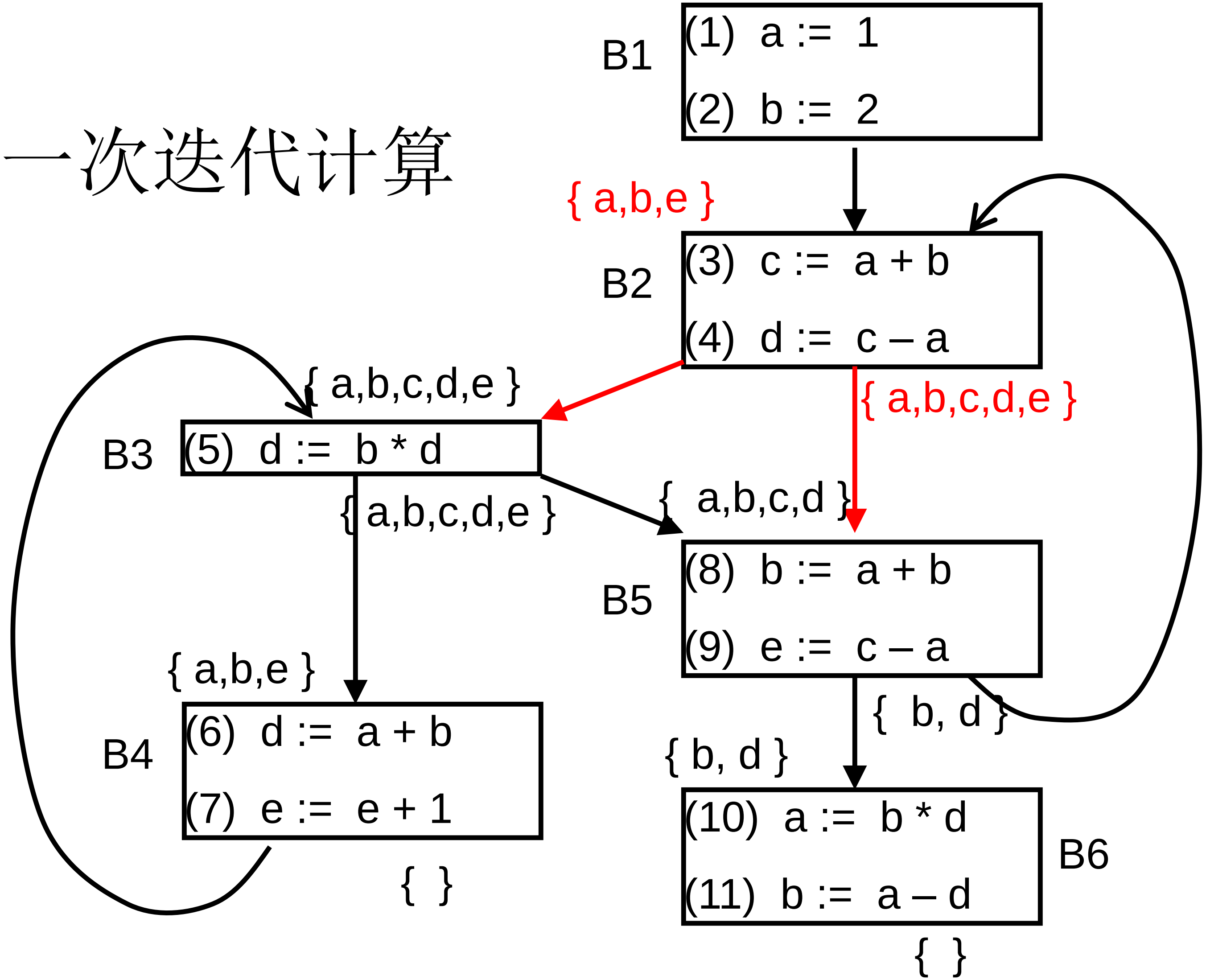
基本块出口活跃变量

- 第一次迭代计算



基本块出口活跃变量

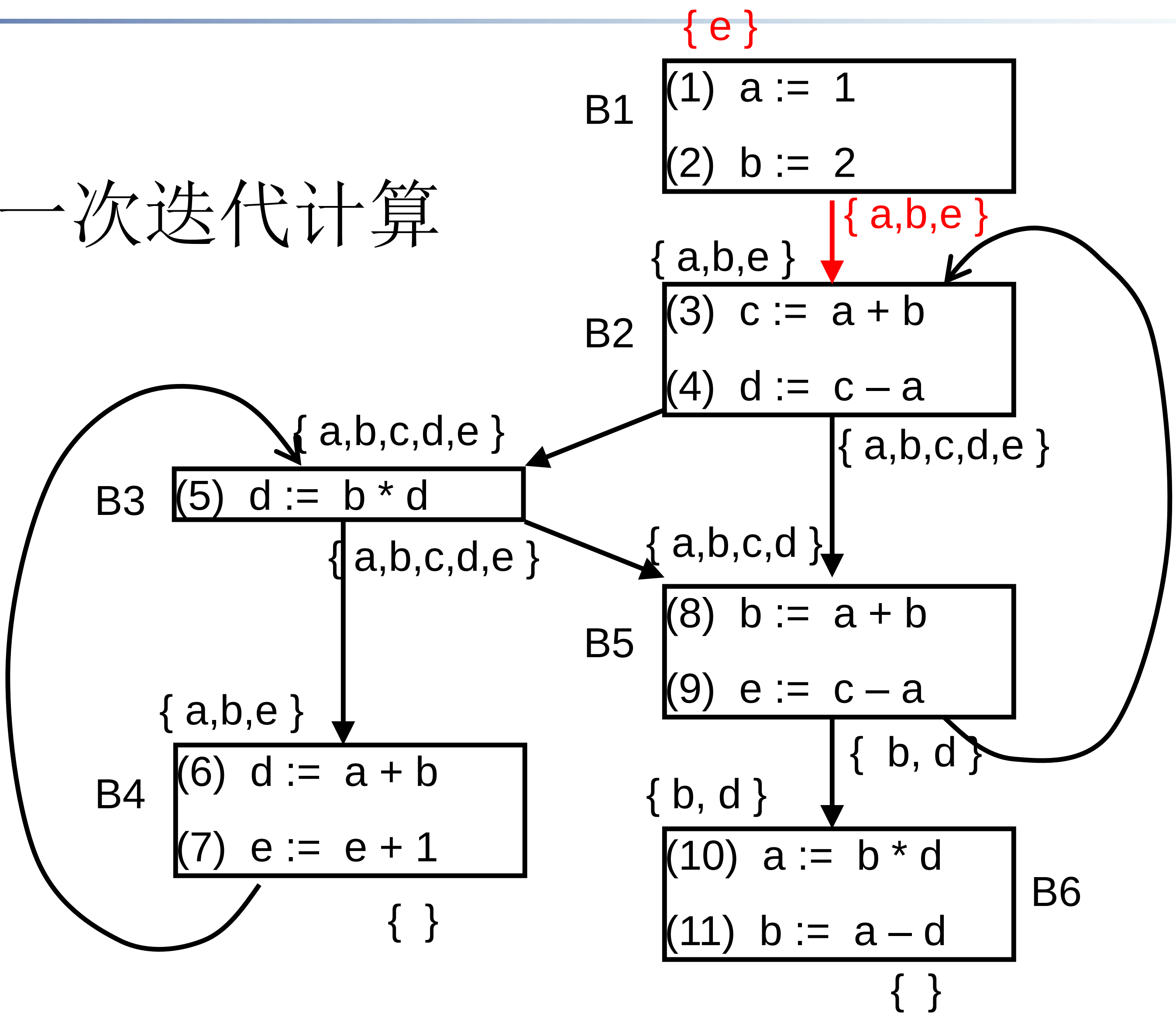
- 第一次迭代计算



基本块出口活跃变量

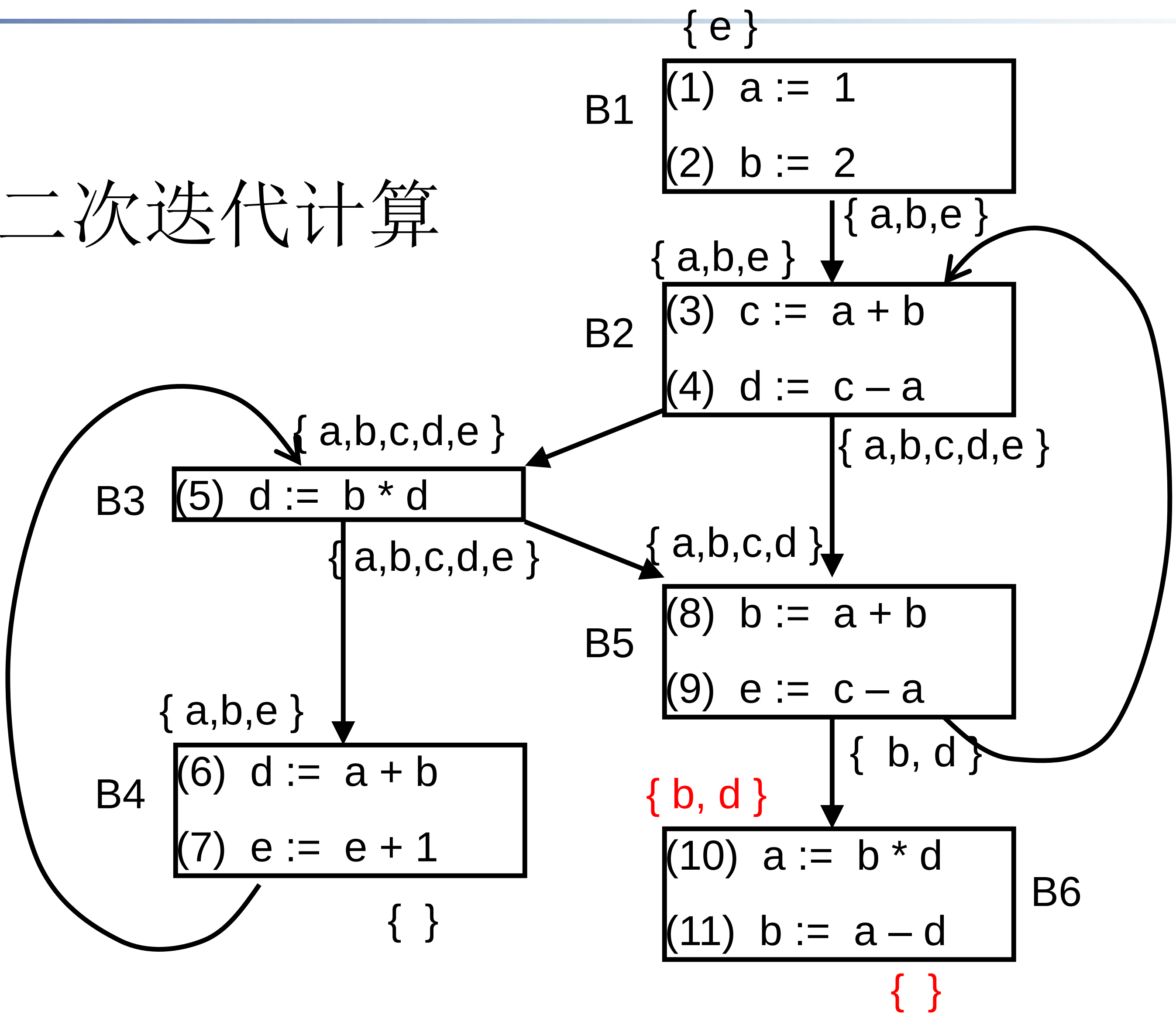
中国科学技术大学 University of Science and Technology of China

- 第一次迭代计算



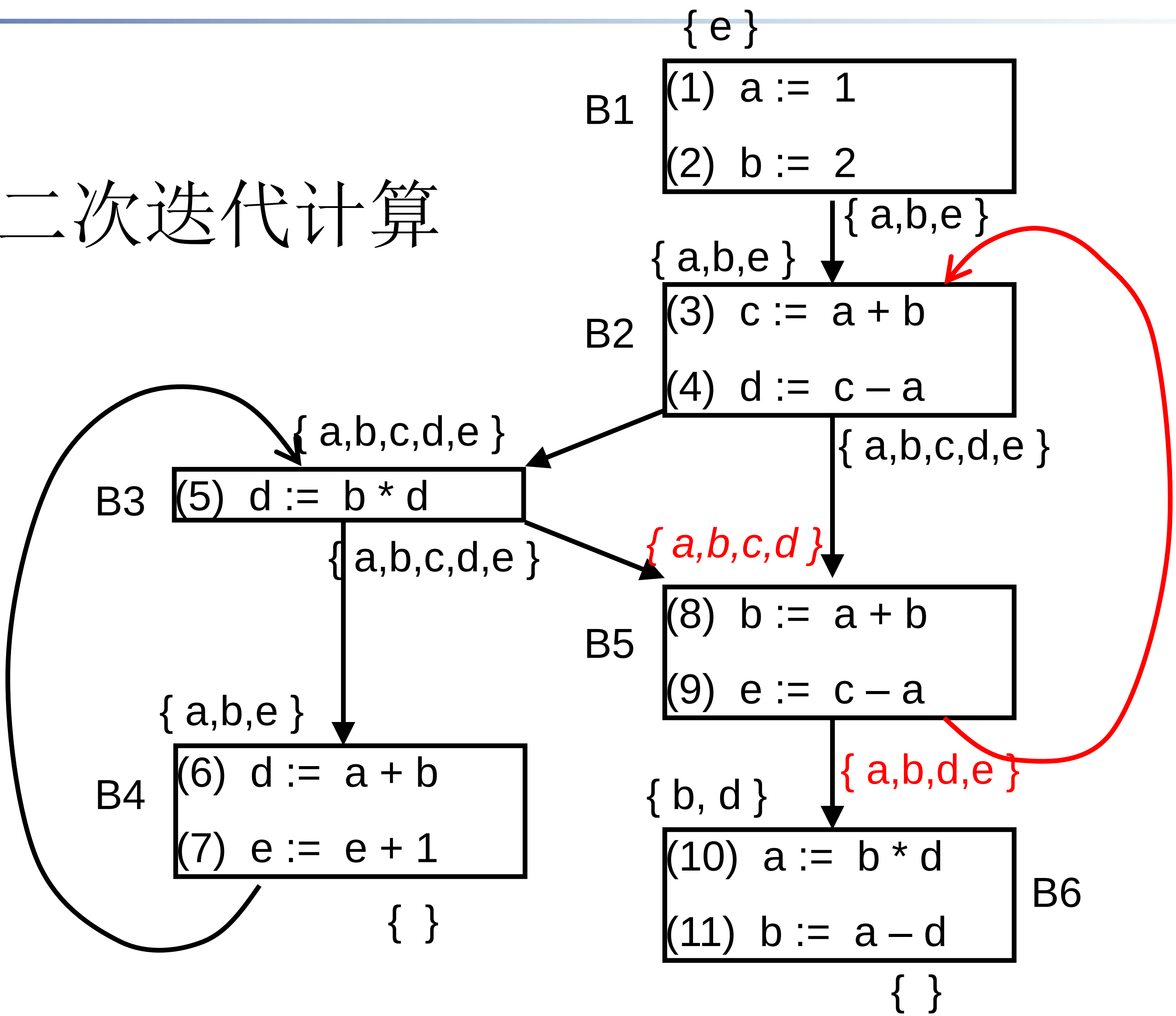
基本块出口活跃变量

- 第二次迭代计算



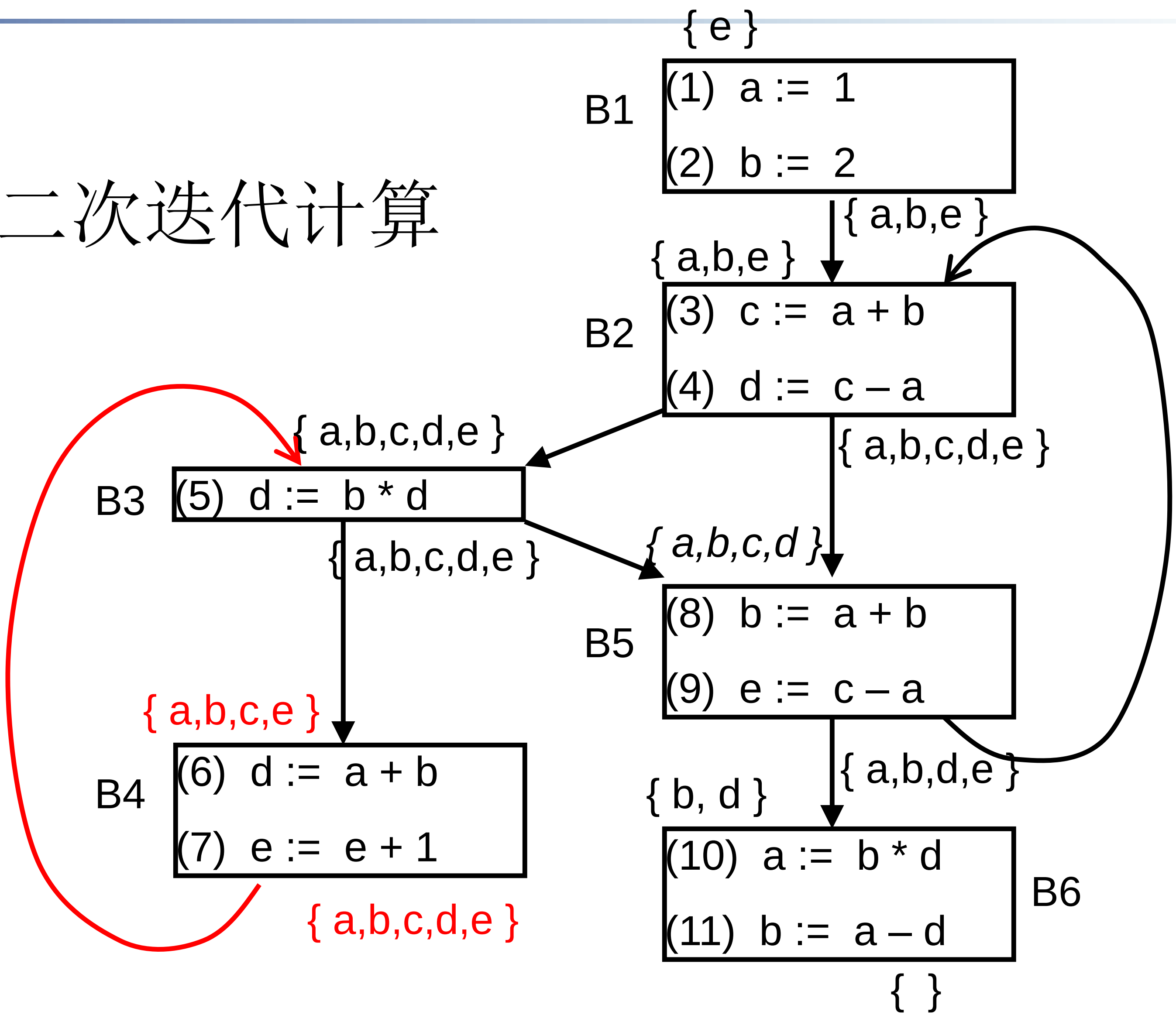
基本块出口活跃变量

- 第二次迭代计算



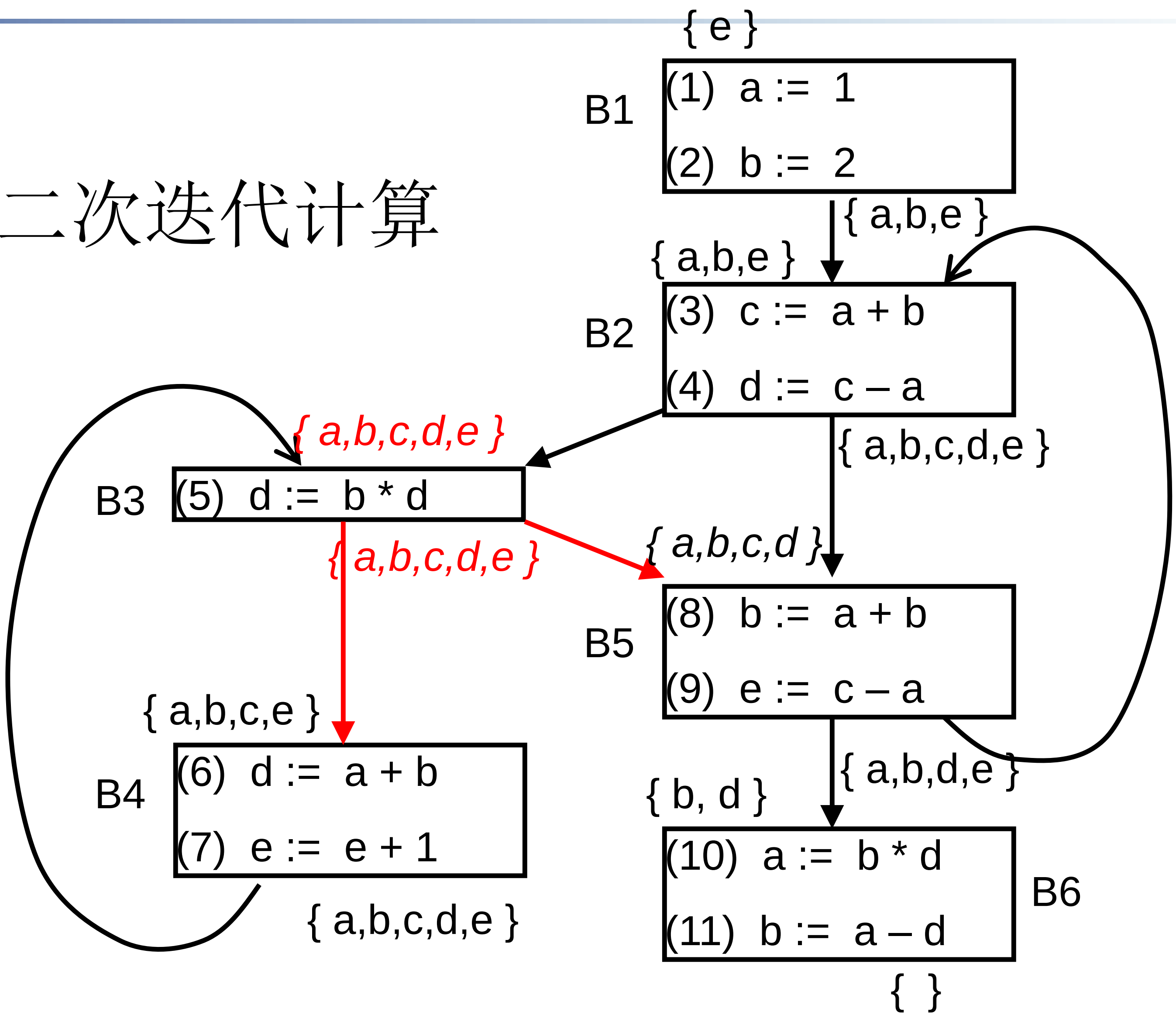
基本块出口活跃变量

- 第二次迭代计算



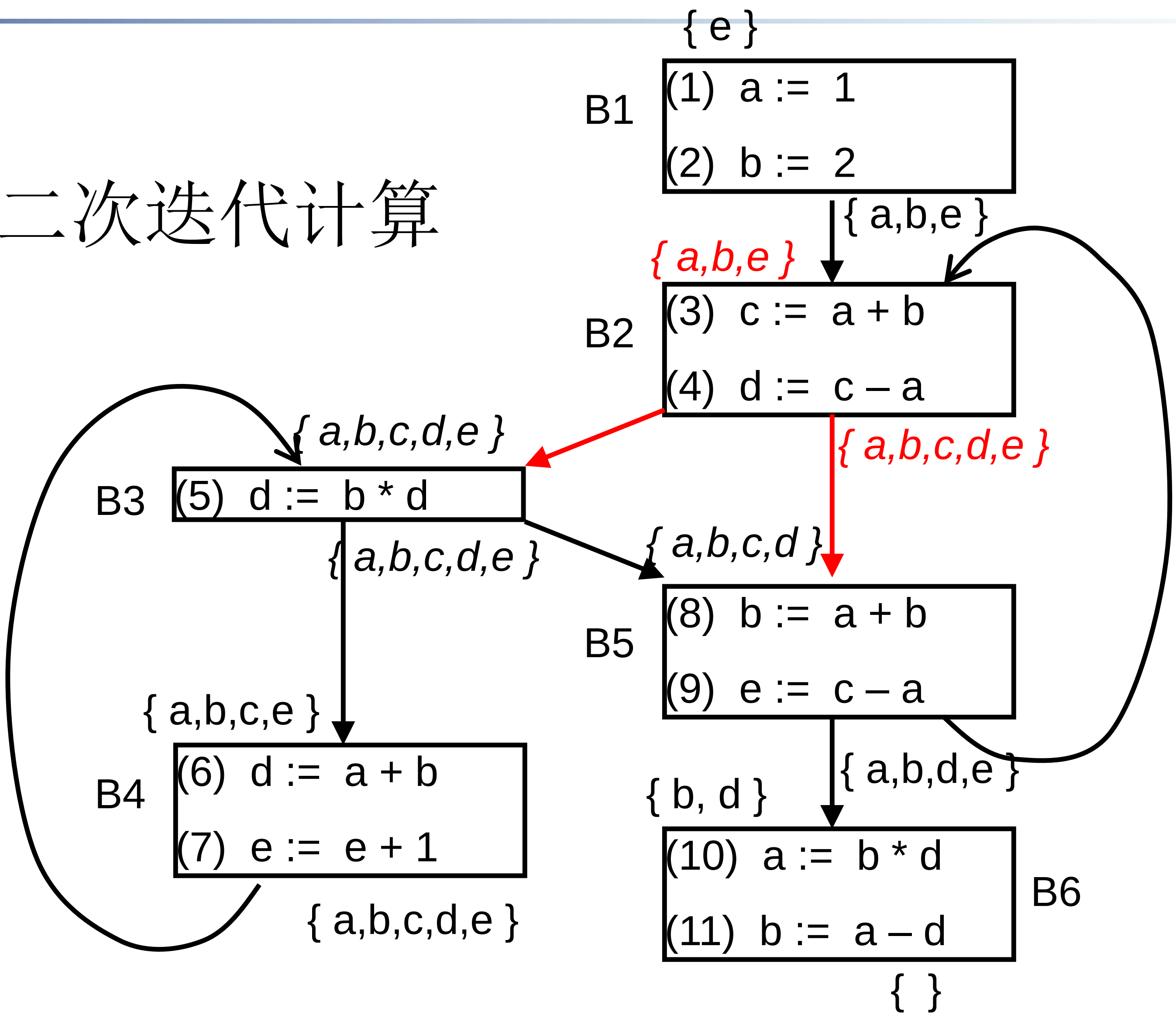
基本块出口活跃变量

- 第二次迭代计算



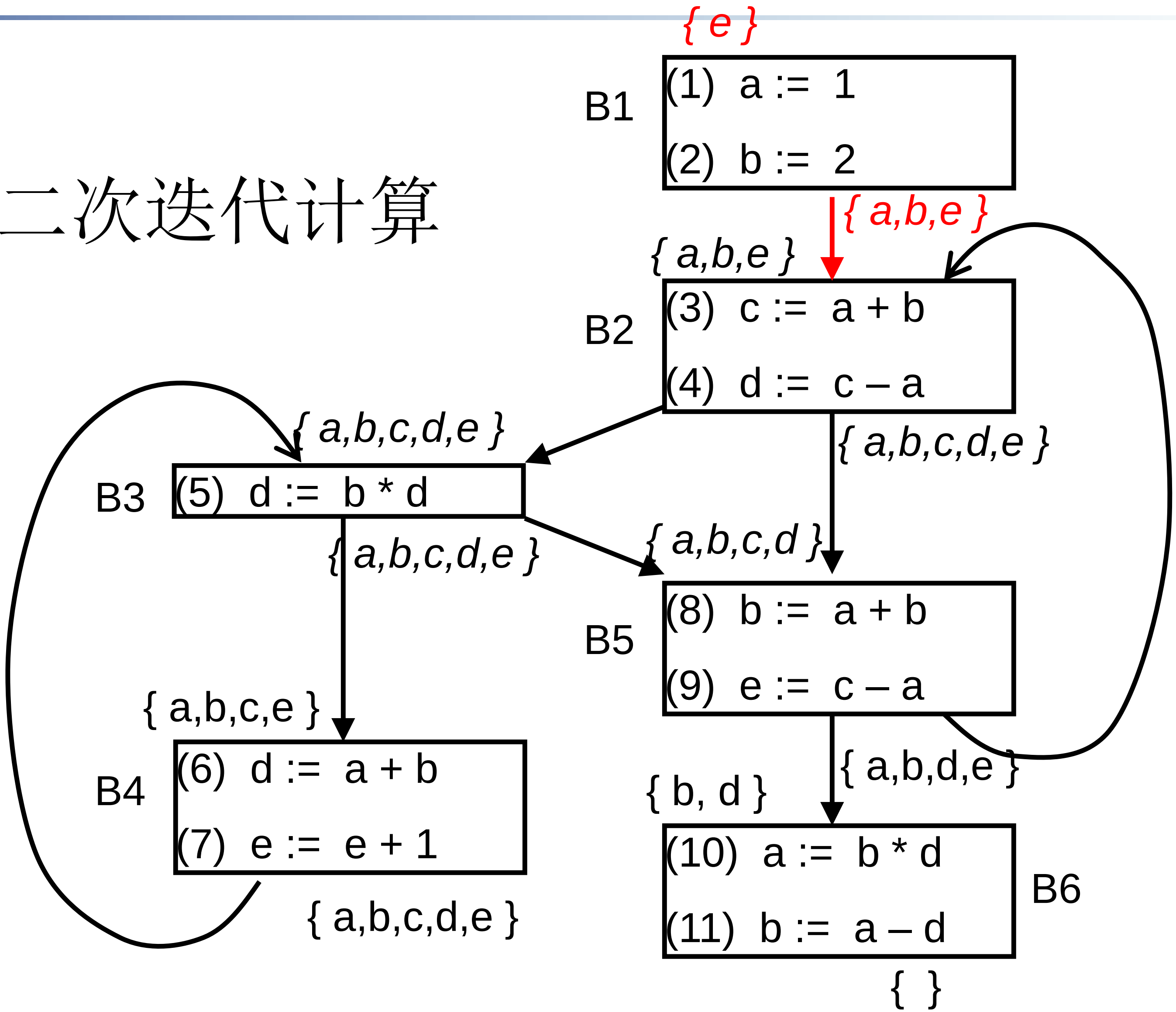
基本块出口活跃变量

- 第二次迭代计算

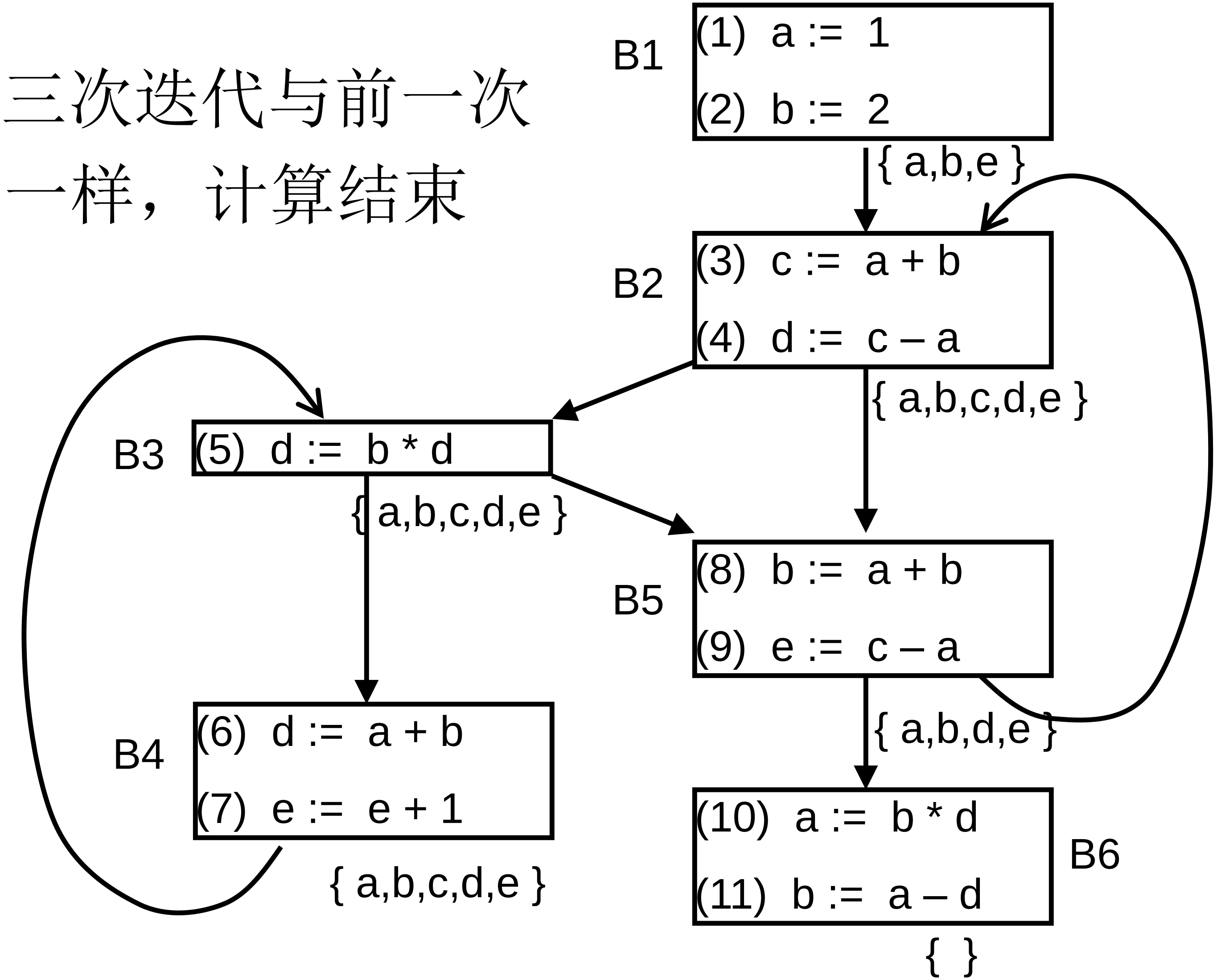


基本块出口活跃变量

- 第二次迭代计算



- 第三次迭代与前一次结果一样，计算结束



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!