编译原理（H）
习题总结

Homework 1 to 4

C1RECOGNIZER实验

Lab 1-1

LAB 1-1

- 一些错误：
  - 行注释跨行的处理
  - Equal是‘==’不是‘=’
  - 注释应该和WhiteSpace一样，skip而不是生成token
  - 非token的符号忘记标注fragment而作为token被输出
  - 正负号并不在number中，是unary operator
  - ‘main’不是关键字，学好C语言
  - Identifer/Indentifier等拼写错误

TYPICAL ERROR:
NO VIABLE ALTERNATIVE AT INPUT <EOF>

- ANTLR 4 old bug，当左递归非终结符作为起始符号时固定出现
- 保证起始符号不是包含左递归的非终结符，如以如下wrap作为起始即可避免：
  - wrap: leftRecur
  - leftRecur: leftRecur xxx | xxx

SYNTAX TREE DEFINITION

- assembly表示整个程序，对应于单一输入源文件
  - source_name字段留空；本意是表示输入文件名，后来因为只有单一文件输入而放置
- var_def_stmt_syntax同时是global_def_syntax和stmt_syntax
  - 多个变量的声明语句展开为多个var_def_stmt_syntax
  - 在不同上下文中，以不同类型放置于父节点的相应容器（ptr_list<?>）中

书面作业

Homework 1, 2, 3-1&2, 4-1&2

教材版本问题

- 以第三版为准，第二版和第三版的出入很多

HOMEWORK 1

- 基本没有问题

- 一些细节
  - 在Git仓库中提交编译生成的内容
  - 错误的Markdown格式
  - 不详细的说明

HOMEWORK 2

- 绝大多数人没有显式地表达出状态机和回滚。
- H1中的细节问题在H2中基本同时存在。
  - 每个人的特别问题所在在Google Doc文档中有列出

HOMEWORK 3-1&2

- 2.4 (c) C风格块注释的正规式表达，几个重点：
  - 注释内连续的多个“*”是合法的
  - 注释中“*”后不能出现“/”，否则会误配超过注释末尾的文本
  - “/*/”不是合法的

- 由于正规式基本可以简单改写成正则表达式，可以使用正则相关工具测试正规式，如
  - Online regex tester and debugger: https://regex101.com/

HOMEWORK 3-1&2

- 2.4 (e) 仅出现一次连续两个相同字符的数字串的正规式表达
- 类似课程中提到的“无连续两个相同字符的数字串”的逻辑，基于此扩展即可：
  - “无连续两个相同字符的数字串”记作no_dup，则
    - answer → no_dup no_dup
  - 这一正规式表达二义，但题目并未要求确定性。
- 另一种方式（非二义）：
  - answer → {no_dup_end_i no_dup_start_i} | no_dup

HOMEWORK 4-1&2

- 3.4 表达正规式语言的文法。
- 难点：(a) 证明此文法产生所有正规式。
- 两个方向：
  1. 产生的都是正规式（结论显然，没有明确写出的同学也没有判错）
  2. 正规式均能被此文法表达。两种证法：
     - 归纳法，对正规式长度或“层数”做归纳
     - 反证法，假设最短的不被此文法表达的正规式为R

HOMEWORK 4-1&2

- 3.8, 3.11, 3.12
- 同学们普遍能很好地理解题意并知晓如何解决问题。
- 但是!
  - 构造分析器=列出分析表，请不要写代码，看着太累，我不是人形编译器+CPU
  - 证明不是LL(1)文法只需要找出一点不满足条件的即可，不需要把所有的不满足条件的点都列出来

HOMEWORK 5-1&2

- 3.16, 3.20, 3.27
- 同样地没有出现很大的理解问题，但：
  - 证明不是SLR/LR需要至少列出部分分析表、指出冲突
    - 考试时最好列全
  - 证明是SLR/LR需要完整列出最终的分析表，不能偷懒！

HOMEWORK 6-1&2

- 3.22, 3.24, 3.30, 3.34, 3.37
- 依旧同样地没有出现很大的理解问题，但：
  - 同上，证明是或不是SLR/LR/LALR时不能偷懒
  - 注意新推出的状态是否和先前的相同
  - 细心！！！！！

HOMEWORK 6-1&2

- 3.30和3.34的语言相同；注意反例 aaababbb
  - 正解
    - S: a S | a T b S | %empty; T: a T b T | %empty;
  - 常见错解
    - S: a S | E S | %empty; E: a E b | E E | %empty;
- 3.37的DFA较为复杂，共11个状态，其中几对较为相似，易混淆
  - “列出所有可能”共两种可能；这实际上是GLR分析器的方法

习题课

2017-11-23

H7

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2， 5， 8， 10， 14。

(2)写出相应的语法制导定义

(3)写出相应的预测翻译器

(4)写出自下而上分析的栈操作代码

概念区分

- 语义规则和产生式相联系的两种方式
  - 语法制导定义
    - 将文法符号和某些属性相关联，并通过语义规则来描述如何计算属性的值，没有描述这些规则的计算时机
  - 语法制导的翻译方案
    - 在产生式的右部的适当位置，插入相应的语义动作，按照分析的进程，执行遇到的语义动作，从而明确了语法分析过程中属性的计算时机。

H7

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2，5， 8， 10， 14。

- a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。
- 方法一：
  - 继承属性 in：该文法符号推出的字符序列的前面已经有多少字符
  - 综合属性 out：该文法符号推出的字符序列的最后一个字符在序列中是第几个字符

S’ → { S.in = 0; } S
S → { L.in = S.in +1; } (L) { S.out = L.out + 1; }
S → a { S.out = S.in + 1; print (S.out); }
L → { L1.in = L.in; } L1, { S.in = L1.out + 1; } S {L.out = S.out; }
L → { S.in = L.in; } S { L.out = S.out; }

H7

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2，5， 8， 10， 14。

- a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。
- 方法二：
  - 继承属性 in：该文法符号推出的字符序列的前面已经有多少字符
  - 综合属性 total：该文法符号推出的字符序列所包含的字符总数

S’ → { S.in = 0; } S
S → { L.in = S.in +1; } (L) { S.total = L.total + 2; }
S → a { S.total = 1; print (S.in + 1); }
L → { L1.in = L.in; } L1, { S.in = L1.in + L1.total + 1; } S {L.total = L1.total +S.total + 1; }
L → { S.in = L.in; } S { L.total = S.total; }

H7

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

  写出相应的语法制导定义

- a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。
- 方法一的语法制导定义：
  - 继承属性 in：该文法符号推出的字符序列的前面已经有多少字符
  - 综合属性 out：该文法符号推出的字符序列的最后一个字符在序列中是第几个字符

产生式	语义规则
S’ → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → L1, S	L1.in = L.in; S.in = L1.out + 1; L.out = S.out;
L → S	S.in = L.in; L.out = S.out;

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

  写出相应的预测翻译器

H7

- 消除左递归

S’ → S
S → (L)
S → a
L → ST
T → ,ST | ε

产生式	语义规则
S’ → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → ST	S.in = L.in; T.in = S.out; L.out = T.out;
T → ,ST_1	S.in = T.in + 1; T_1.in = S.out; T.out = T_1.out
T → ε	T.in = T.out

H7

产生式	语义规则
S’ → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → ST	S.in = L.in; T.in = S.out; L.out = T.out;
T → ,ST_1	S.in = T.in + 1; T_1.in = S.out; T.out = T_1.out
T → ε	T.in = T.out

```
int S’(){
  return S(0);
}
```

```
int S(int b){
  int in, out;
  if(lookahead == ‘(’){
     in = b + 1;
     match(‘(’);
     out = L(in) + 1;
     match(‘)’)
  }else
  {
      match(‘a’);
      out = b + 1;
      print(out);
  }
   return out;
}
```

H7

产生式	语义规则
S’ → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → ST	S.in = L.in; T.in = S.out; L.out = T.out;
T → ,ST_1	S.in = T.in + 1; T_1.in = S.out; T.out = T_1.out
T → ε	T.in = T.out

```
int L(int b){
  int out;
  out = S(b);
  out = T(out);
  return out;
}
```

```
int T(int b)
{
   int out;
   if(lookahead == ‘,’){
      match(‘,’);
      out = S(b+1);
      out = T(out);
    }else{
      out = b;
    }
    return out;
}
```

H7

- 4.12(b) 文法如下：

  S’ → S

  S → (L)

  S → a

  L → ST

  T → ,ST | ε

  写出自下而上分析的栈操作代码

- 引入标记非终极符M,N,R,P

产生式	语义规则	栈操作代码
S’ → MS	S.in = M.out	Stack[top - 1] = Stack[top]
M → ε	M.out = 0	Stack[top + 1] = 0
S → (NL)	N.in = S.in + 1, L.in = N.out; S.out = L.out + 1;	Stack[top - 3] = Stack[top - 1] + 1
N → ε	N.out = N.in	Stack[top + 1] = Stack[top - 1] + 1
S → a	S.out = S.in + 1; print (S.out);	Stack[top] = Stack[top - 1] + 1
L → SRT	S.in = L.in; R.in = S.in; T.in = R.out, L.out = T.out;	Stack[top - 2] = Stack[top]
R → ε	R.out = R.in	Stack[top + 1] = Stack[top - 1]
T → ,SPT$_1$	S.in = T.in + 1; P.in = S.in; T$_1$.in = P.out; T$_1$.out = S.out;	Stack[top - 3] = Stack[top]
P → ε	P.out = P.in	Stack[top + 1] = Stack[top]
T → ε	T.out = T.in	Stack[top] = Stack[top - 1]

H8

- 5.5 假如有下列C的声明:

```
typedef struct{
        int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

H8

- 5.5 假如有下列C的声明:

```
typedef struct{
        int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

CELL foo[100];

array(Range ?, TypeOfElement ?)

array(0..99, TypeOfElement ?)

array(0..99, CELL)

array(0..99, record((int a) ×(int b)))

array(0..99, record((a × integer) ×(b × integer)))

H8

- 5.5 假如有下列C的声明:

```
typedef struct{
        int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

PCELL bar(x, y) int x; CELL y; {}

TypeOfParameters? -> TypeOfReturnValue?

(int × CELL) -> PCELL

(integer × record((a × integer) × (b × integer))) -> PCELL

(integer × record((a × integer) × (b × integer))) ->
pointer(record((a × integer) × (b × integer)))

H8

- 5.12 拓展5.3.3节的类型检查，使之能包含记录。有关记录部分的类型和记录域引用表达式的语法如下：

$$\mathrm{T} \rightarrow \boldsymbol{record}\ fields\ \mathbf{end}$$

$$fields \rightarrow fields; field \mid field$$

$$field \rightarrow \mathbf{id} : T$$

$$E \rightarrow E.\mathbf{id}$$

H8

- 5.12 拓展5.3.3节的类型检查，使之能包含记录。有关记录部分的类型和记录域引用表达式的语法如下：

$\mathrm{T} \rightarrow \ \boldsymbol{record}\ fields\ \mathbf{end}$	{T.type = record(fields.type)}
$fields \rightarrow \ fields; field$	{fields.type = fields.type × field.type}
$fields \rightarrow \ field$	{fields.type = field.type}
$field \rightarrow \mathbf{id} : T$	{field.type = id.name × T.type}
$E \rightarrow E_1 . \mathbf{id}$	{E.type = if(E1.type == record(t)) lookup(E1.type, id.name) else type_error;}

H8

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

用SPARC/Solaris C编译器编译下面的C程序时，错误信息如下：

type.c:24: warning: passing argument 4 of `qsort' from incompatible pointer type

请你对该程序略作修改，使得该警告错误能消失，并且不改变程序的结果。

```
#include <stdlib.h>

typedef struct{
            int    Ave;
            double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
     return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob) {
      return(1);
   }else{
      return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo,n,sizeof(HYPO),HypoCompare);
}
```

H8

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

```
#include <stdlib.h>

typedef struct{
        int    Ave;
        double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
     return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob) {
      return(1);
   }else{
      return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo, n, sizeof(HYPO), HypoCompare);
}
```

H8

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

方法一：修改HypoCompare函数形式参数的类型

```
#include <stdlib.h>

typedef struct{
        int    Ave;
        double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(const void *stHypo1, const void*stHypo2)
{
   if ((HYPO *)stHypo1->Prob>(HYPO *)stHypo2->Prob){
     return(-1);
   }else if ((HYPO *)stHypo1->Prob<(HYPO *)stHypo2->Prob) {
      return(1);
   }else{
      return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo, n, sizeof(HYPO), HypoCompare);
}
```

H8

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

方法二：强制修改qsort函数调用中第四个参数的类型

```
#include <stdlib.h>

typedef struct{
            int    Ave;
            double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
     return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob){
      return(1);
   }else{
      return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo, n, sizeof(HYPO), int (*)(const void *, const void *)
HypoCompare);
}
```

H9

- 5.16对下面的每对表达式，

  (a) $\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)$

  (b) $array\ (\beta_1) \rightarrow (pointer\ (\beta_1) \rightarrow \beta_2)$

  (c) $\gamma_1 \rightarrow \gamma_2$

找出(a) 和 (b) 、(b)和(c)最一般的合一代换：

H9

- (a)与(b)

S(α1) = array (β1)
S(α2) = pointer (β1)
S(β2) = array (β1)

- 5.16对下面的每对表达式，

(a) $\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)$

(b) *array* $(\beta_1) \rightarrow$ (*pointer* $(\beta_1) \rightarrow \beta_2$)

(c) $\gamma_1 \rightarrow \gamma_2$

找出(a) 和 (b) 、(b)和(c)最一般的合一代换：

H9

- (b)与(c)

- 5.16对下面的每对表达式，

(a) $\alpha_1 \to (\alpha_2 \to \alpha_1)$

(b) *array* $(\beta_1) \to$ (*pointer* $(\beta_1) \to \beta_2$)

(c) $\gamma_1 \to \gamma_2$

找出(a) 和 (b) 、(b)和(c)最一般的合一代换：

S(γ1) = array (β1)
S(γ2) = pointer (β1) → β2

H9

- 5.17效仿例5.5，推导下面map的多态类型：

map : $\forall\alpha.\ \forall\beta.\ (\ (\alpha \to \beta) \times list\ (\alpha)\ ) \to list\ (\beta)$

map的ML定义是

```
fun map ( f, l ) =
    if null (l ) then nil
    else cons (f (hd (l)), map (f, tl (l ) ) );
```

在这个函数体中，内部定义的标识符的类型是：

null : $\forall\alpha.\ list\ (\alpha) \to$ boolean ;

nil : $\forall\alpha.\ list\ (\alpha)$;

cons : $\forall\alpha.\ (\alpha \times list\ (\alpha)\ ) \to list\ (\alpha)$;

hd : $\forall\alpha.\ list\ (\alpha) \to \alpha$;

tl : $\forall\alpha.\ list\ (\alpha) \to list\ (\alpha)$;

H9

- 第一步：列出类型声明和要检查的表达式

f : α
l : β
if: $\forall\alpha.$ boolean $\times$ *list* (α) $\times$ *list* (α) $\rightarrow$ *list* (α)
null : $\forall\alpha.$ *list* (α) $\rightarrow$ boolean ;
nil : $\forall\alpha.$ *list* (α) ;
cons : $\forall\alpha.$ ($\alpha \times$ *list* (α)) $\rightarrow$ *list* (α) ;
hd : $\forall\alpha.$ *list* (α) $\rightarrow \alpha$;
tl : $\forall\alpha.$ *list* (α) $\rightarrow$ *list* (α) ;
match(
  map (f, l),
  if null (l) then nil
        else cons (f (hd (l)), map (f, tl (l)));
)

- 5.17效仿例5.5，推导下面map的多态类型：

map : $\forall\alpha.\ \forall\beta.$ (($\alpha \rightarrow \beta$) $\times$ *list* (α)) $\rightarrow$ *list* (β)

map的ML定义是

```
fun map ( f, l ) =
   if null (l ) then nil
   else cons (f (hd (l)), map (f, tl (l ) ) );
```

在这个函数体中，内部定义的标识符的类型是：

null : $\forall\alpha.$ *list* (α) $\rightarrow$ boolean ;
nil : $\forall\alpha.$ *list* (α) ;
cons : $\forall\alpha.$ ($\alpha \times$ *list* (α)) $\rightarrow$ *list* (α) ;
hd : $\forall\alpha.$ *list* (α) $\rightarrow \alpha$;
tl : $\forall\alpha.$ *list* (α) $\rightarrow$ *list* (α) ;

H9

- 第二步：代换推导

map : $\forall\alpha.\ \forall\beta.\ (\ (\alpha \to \beta) \times list\ (\alpha)\ ) \to list\ (\beta)$

fun map (f, l) =

  if null (l) then nil

  else cons (f (hd (l)), map (f, tl (l)));

null : $\forall\alpha.\ list\ (\alpha) \to$ boolean ;

  nil : $\forall\alpha.\ list\ (\alpha)$;

  cons : $\forall\alpha.\ (\alpha \times list\ (\alpha)\ ) \to list\ (\alpha)$;

  hd : $\forall\alpha.\ list\ (\alpha) \to \alpha$; tl : $\forall\alpha.\ list\ (\alpha) \to list\ (\alpha)$;

行	定型断言	代换	规则
1	f : α		(Exp Id)
2	l : β		(Exp Id)
3	map : γ		(Exp Id)
4	map (f , l) : δ	γ = (α × β) → δ	(Exp Funcall)
5	null : list (α0) → boolean		(Exp Id Fresh)
6	null (l) : boolean	β = list (α0)	(Exp Funcall + (2))
7	nil : list (α1)		(Exp Id Fresh)
8	l : list (α0)		由（2）可得
9	hd : list (α2) → α2		(Exp Id Fresh)

H9

行	定型断言	代换	规则
9	hd : list (α2) → α2		(Exp Id Fresh)
10	hd (l) : α0	α2 = α0	(Exp Funcall)
11	f (hd (l)) : α3	α = α0→α3	(Exp Id)
12	f : α0→α3		由（1）可得
13	tl : list (α4)→ list (α4)		(Exp Id Fresh)
14	tl (l) : list (α0)	α4 = α0	(Exp Funcall)
15	map : ((α0→α3) ×list(α0))→ δ		由（3）可得
16	map (f , tl (l)) : δ		(Exp Funcall)
17	cons : α5 × list(α5) → list(α5)		(Exp Id Fresh)
18	cons (…) : list (α3)	α5 = α3 , δ=list(α3)	(Exp Funcall)
19	if : boolean × list(α6) × list(α6) →list(α6)		(Exp Id Fresh)
20	if (…) : list (α1)	α6 = α1 , α3 = α1	(Exp Funcall)
21	match : α7 × α7 → α7		(Exp Id Fresh)
22	match (…) : list (α1)	α7 = list (α1)	(Exp Funcall)

map : $\forall\alpha.\ \forall\beta.\ (\ (\alpha \to \beta) \times list\ (\alpha)\ ) \to list\ (\beta)$

```
fun map ( f, l ) =
    if null (l ) then nil
    else cons (f (hd (l)), map (f, tl (l ) ) );
```

null : $\forall\alpha.\ list\ (\alpha) \to$ boolean ;

nil : $\forall\alpha.\ list\ (\alpha)$;

cons : $\forall\alpha.\ (\alpha \times list\ (\alpha)) \to list\ (\alpha)$;

hd : $\forall\alpha.\ list\ (\alpha) \to \alpha$;

tl : $\forall\alpha.\ list\ (\alpha) \to list\ (\alpha)$;

至此有**map : ((α0→α1) × list(α0))→list(α1)**

所以**map : ∀ *α* . ∀ *β* . ((*α* → *β*) × *list* (*α*)) → *list* (*β*)**

H9

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

H9

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z
- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex

H9

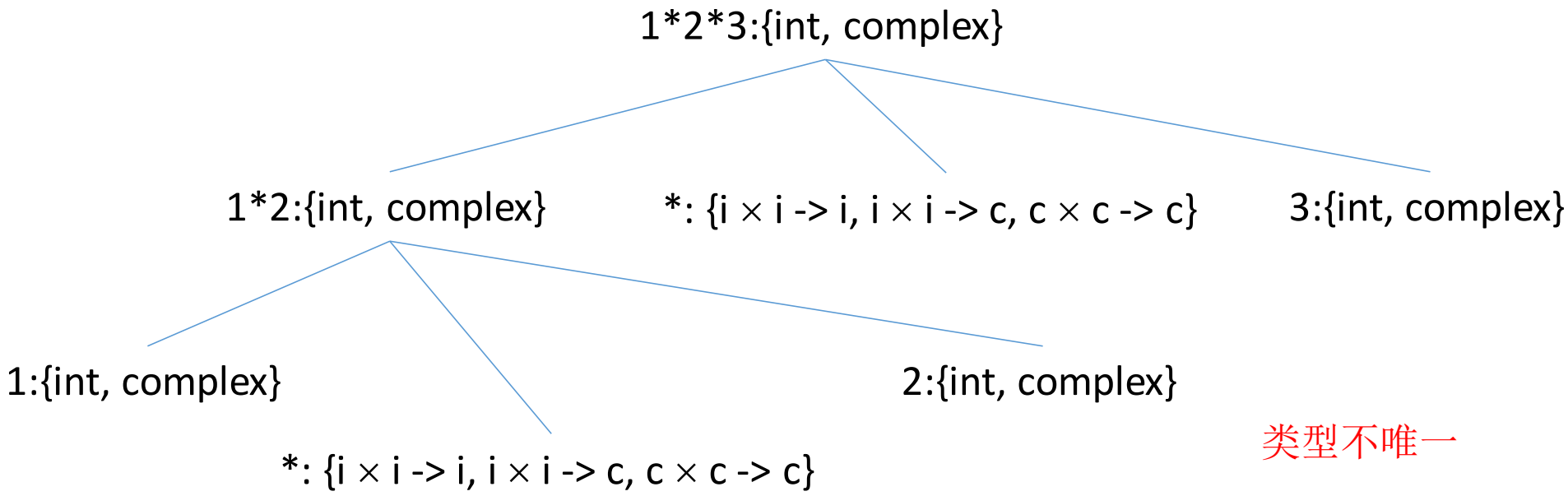
- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



类型不唯一

H9

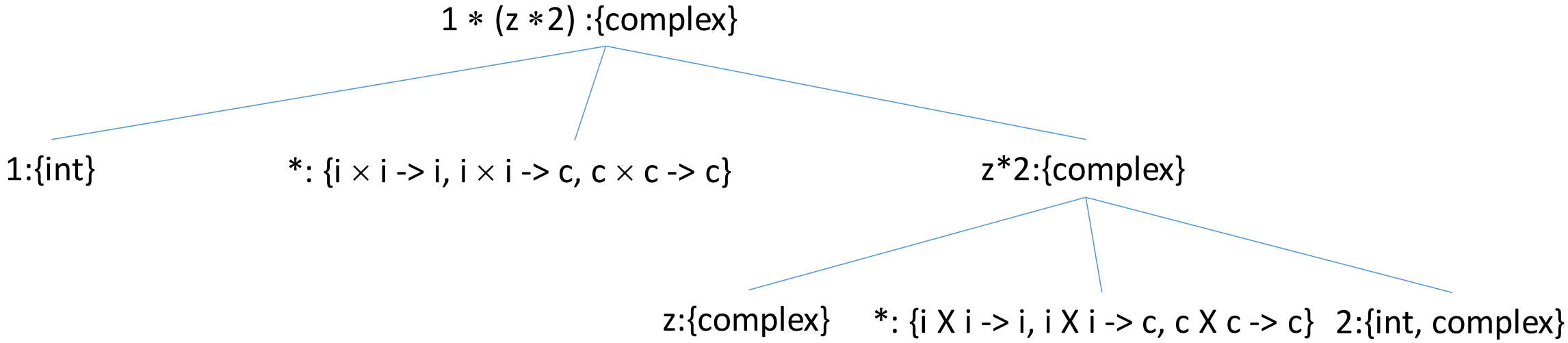
- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



类型唯一

H9

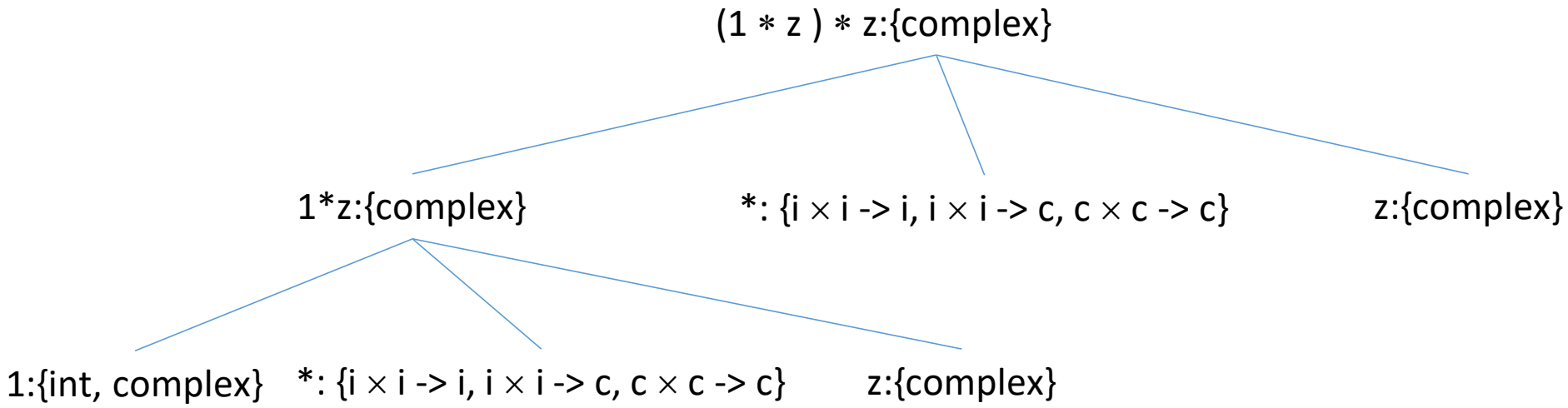
- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1*2*3

  (b) 1 * (z *2)

  (c) (1 * z) * z

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



类型唯一

习题课

2017-12-21

H12

- 8.5 在早先的SPARC/SunOS系统上，经某编译器编译后，下面程序的运行结果是120。但是如果把第十行abs（1）改成1的话，则结果为1。试分析一下原因。

```
int fact(){
  static int i = 5;
  if(i == 0){
    return(1);
  }
  else{
    i = i – 1;
    return ((i + abs(1)) * fact());
  }
}

main(){
  printf(“factor of 5 = %d\n”, fact())
}
```

解答：有一些编译器基于寄存器分配优化的考虑，在计算次序的选择上优先考虑函数表达式。

H12

- 9.15 a.计算支配关系

D(1) = {1}

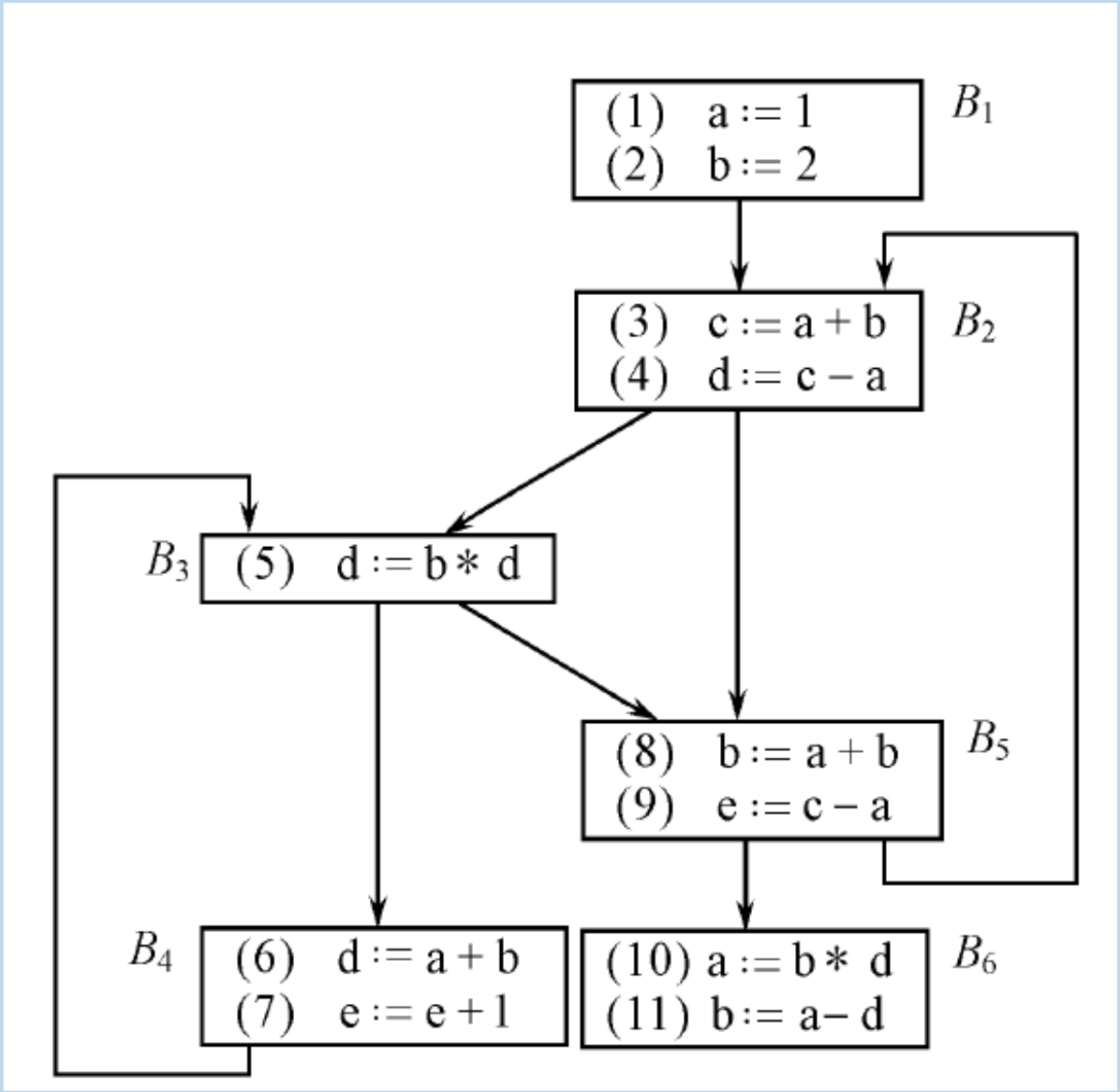
D(2) = {1,2}

D(3) = {1,2,3}

D(4) = {1,2,3,4}

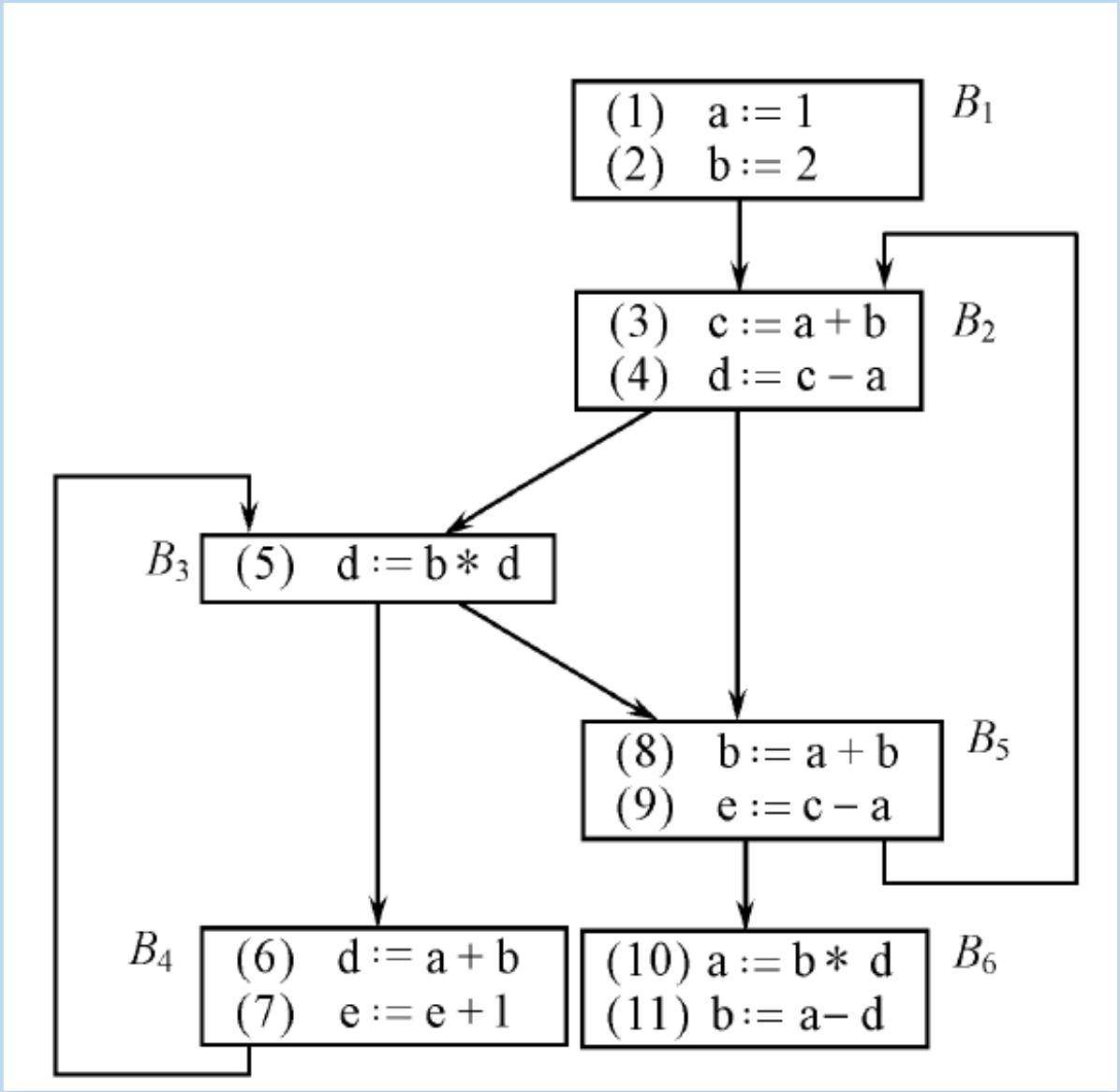
D(5) = {1,2,5}

D(6) = {1,2,5,6}



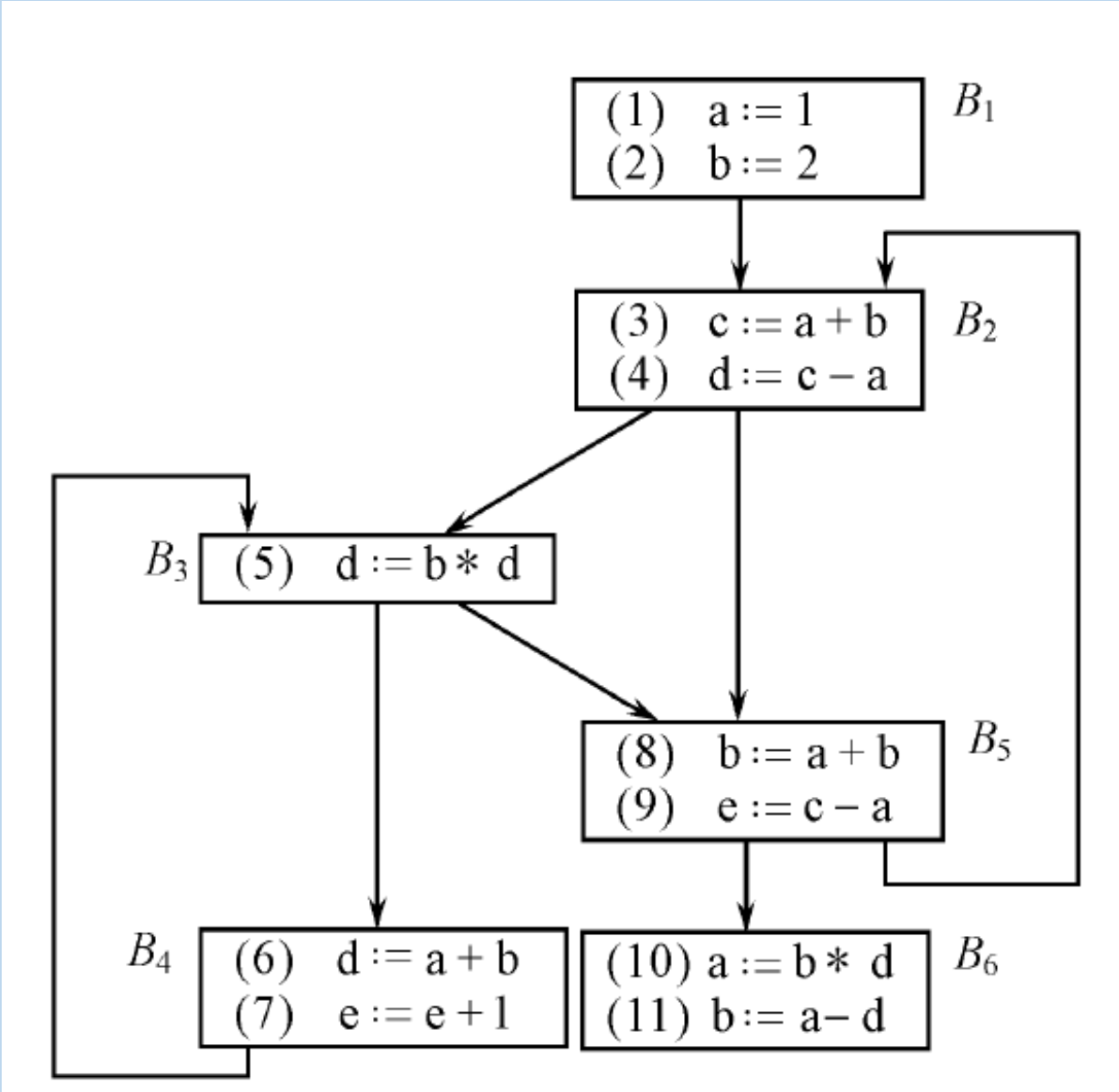
H12

- 9.15 b.找出一种深度优先排序
- {1,2,5,6,3,4}
- Or
- {1,2,3,4,5,6}



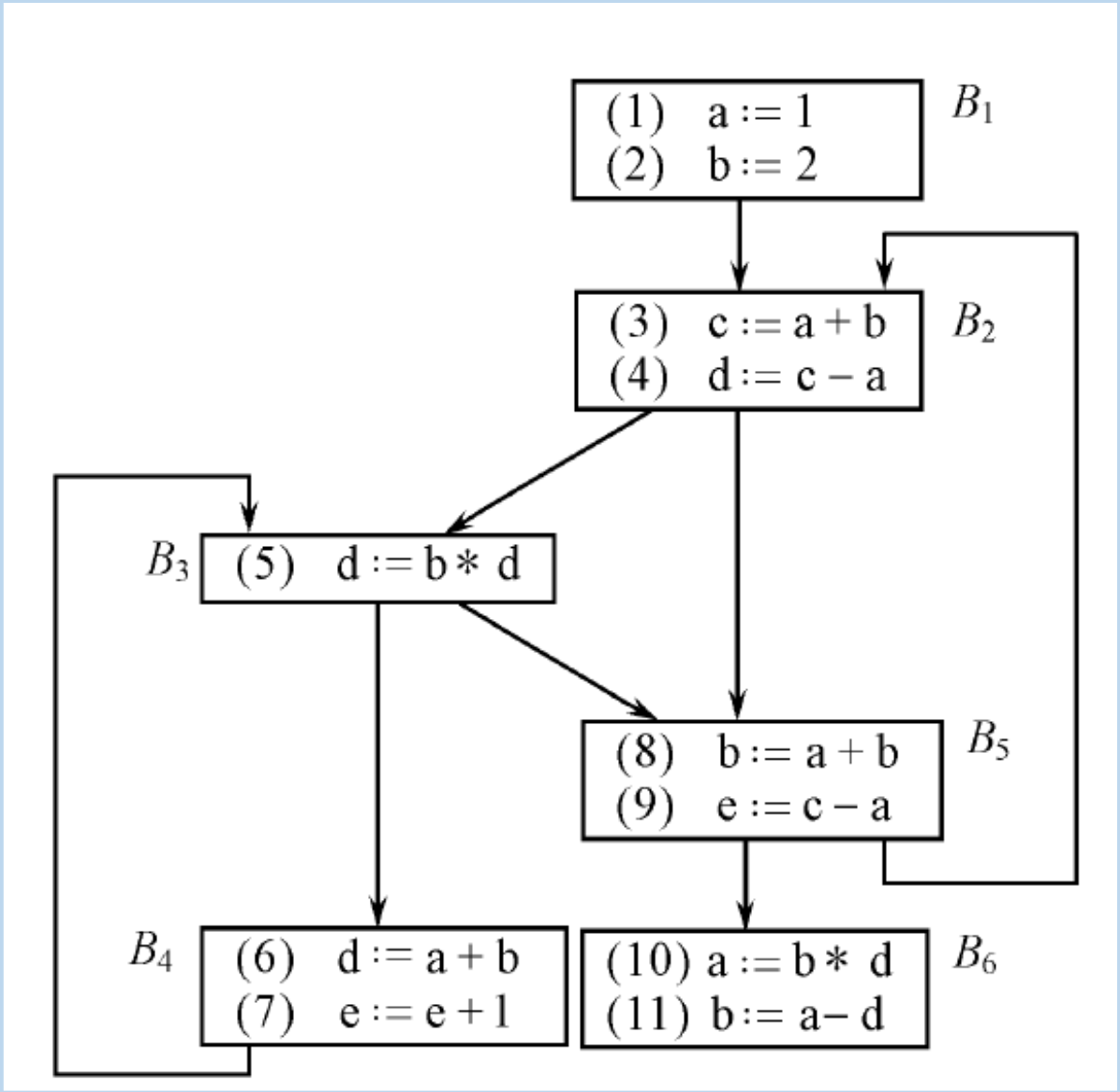
H12

- 9.15 c.对（b）的结果，标明前进边，后撤边和交叉边
- 前进边：1->2; 2->5;2->3;5->6;3->4
- 后撤边：4->3; 5->2
- 交叉边：3->5



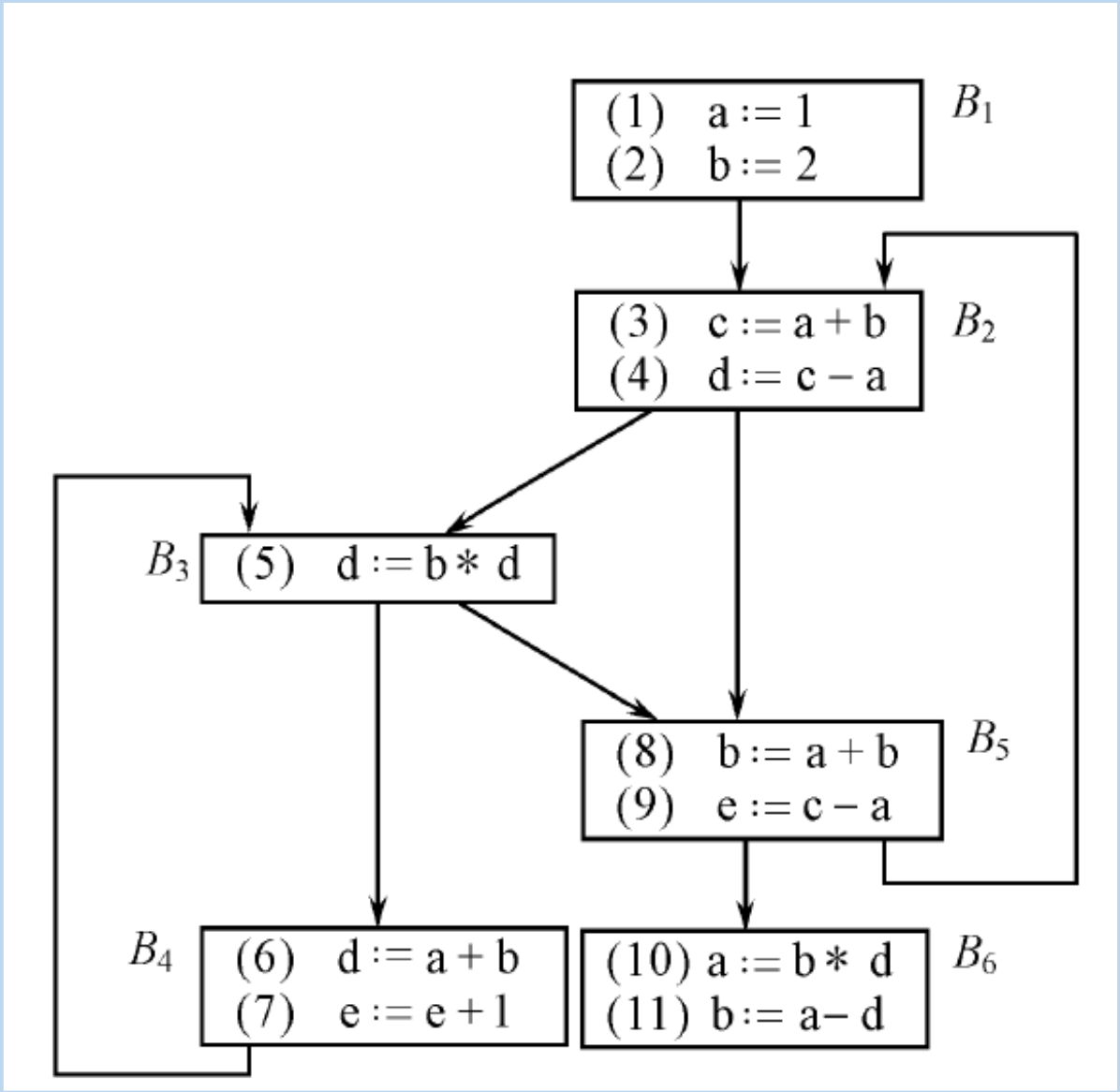
H12

- 9.15 d.该图是否可归约
- 后撤边：4->3; 5->2
- 判断他们是不是回边
- 显然是
- 所以可以归约



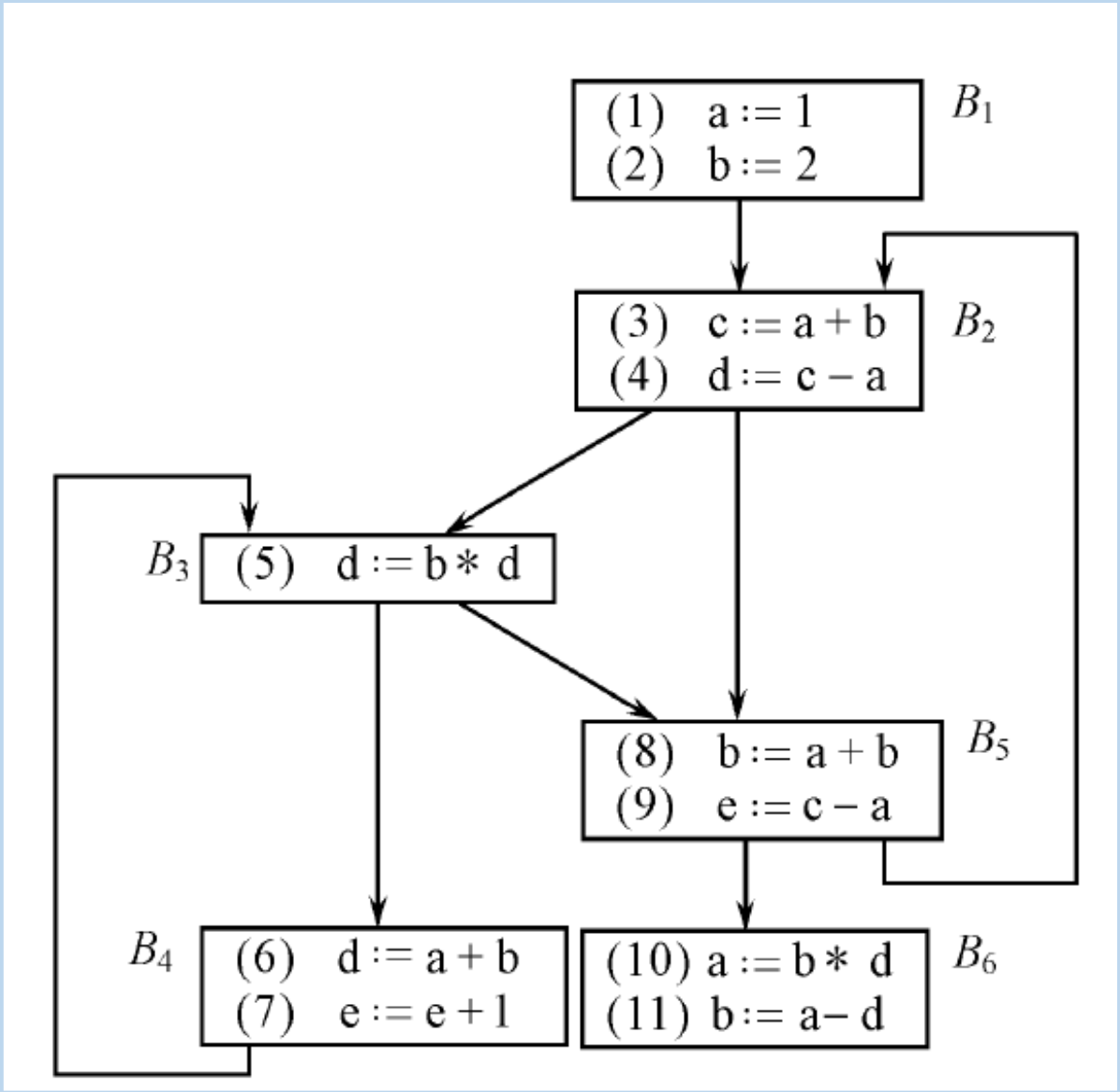
H12

- 9.15 e.计算该流图的深度
- 深度为1
- 看无环路径上有几条后撤边



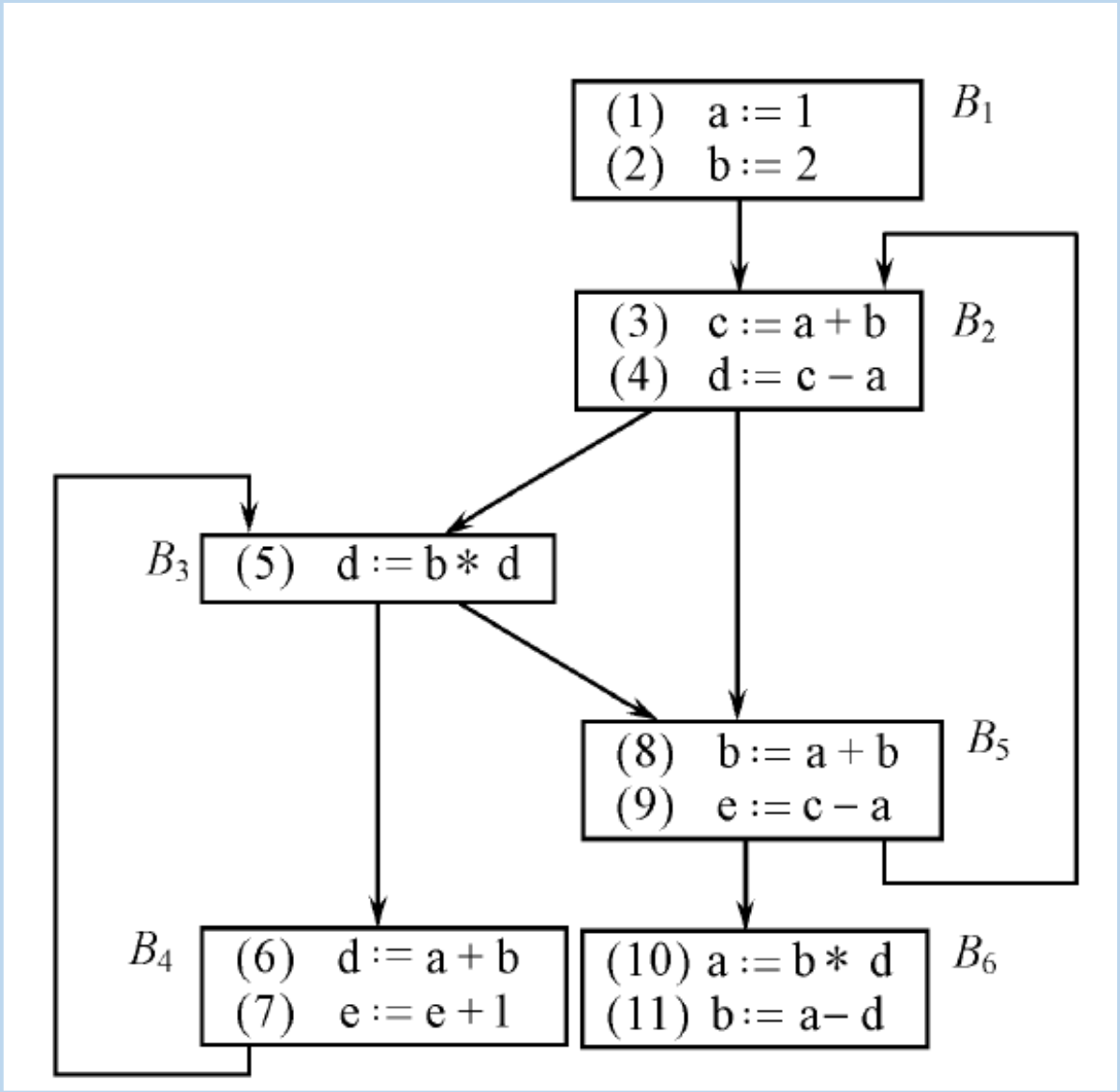
H12

- 9.15 f.找出该图的自然循环
- 针对回边：
- 4->3： {3,4}
- 5->2：{2,3,4,5}



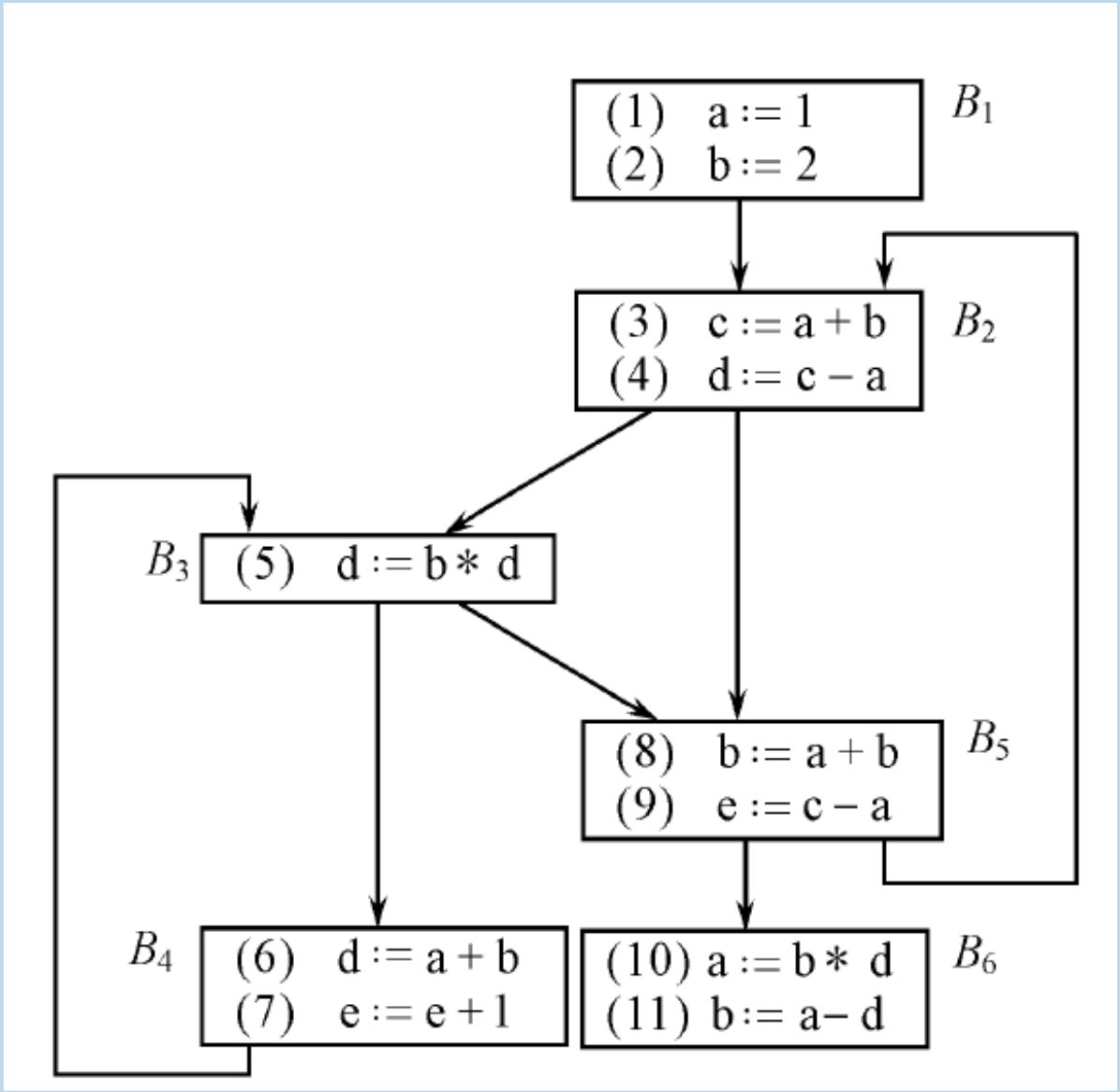
H12

- 9.1 a. 识别该流图的循环
- 针对回边：
- 4->3： {3,4}
- 5->2：{2,3,4,5}



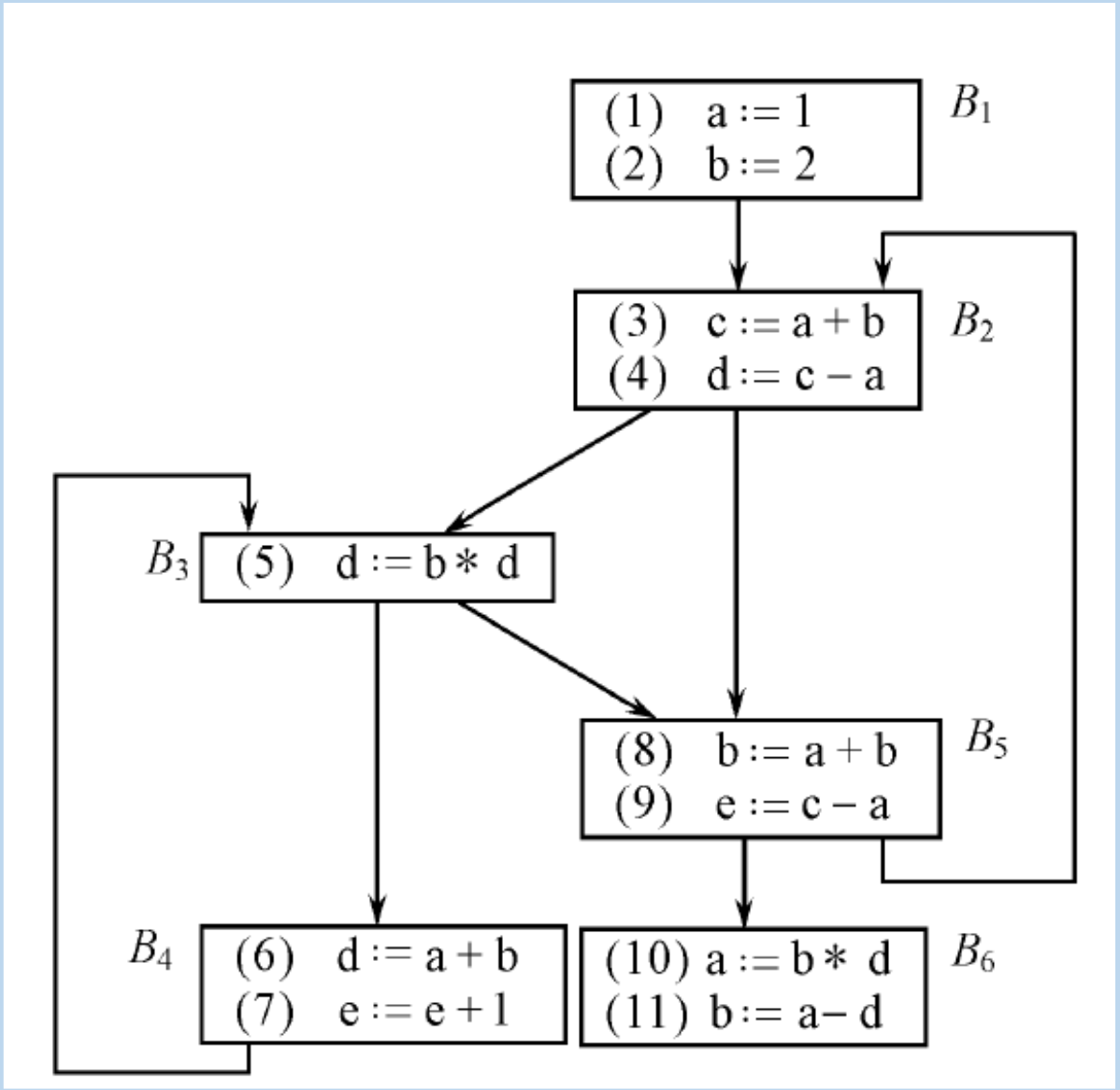
H12

- 9.1 b. 块B1中的语句(1)和(2)都是复写语句，并且它们给ab的赋值都是常量。可以对a和b的哪些引用实施复写传播并将这些引用替换成对常量的引用？
- a值在2,3,4,5中未被修改，所以可以使用复写，而b不可以



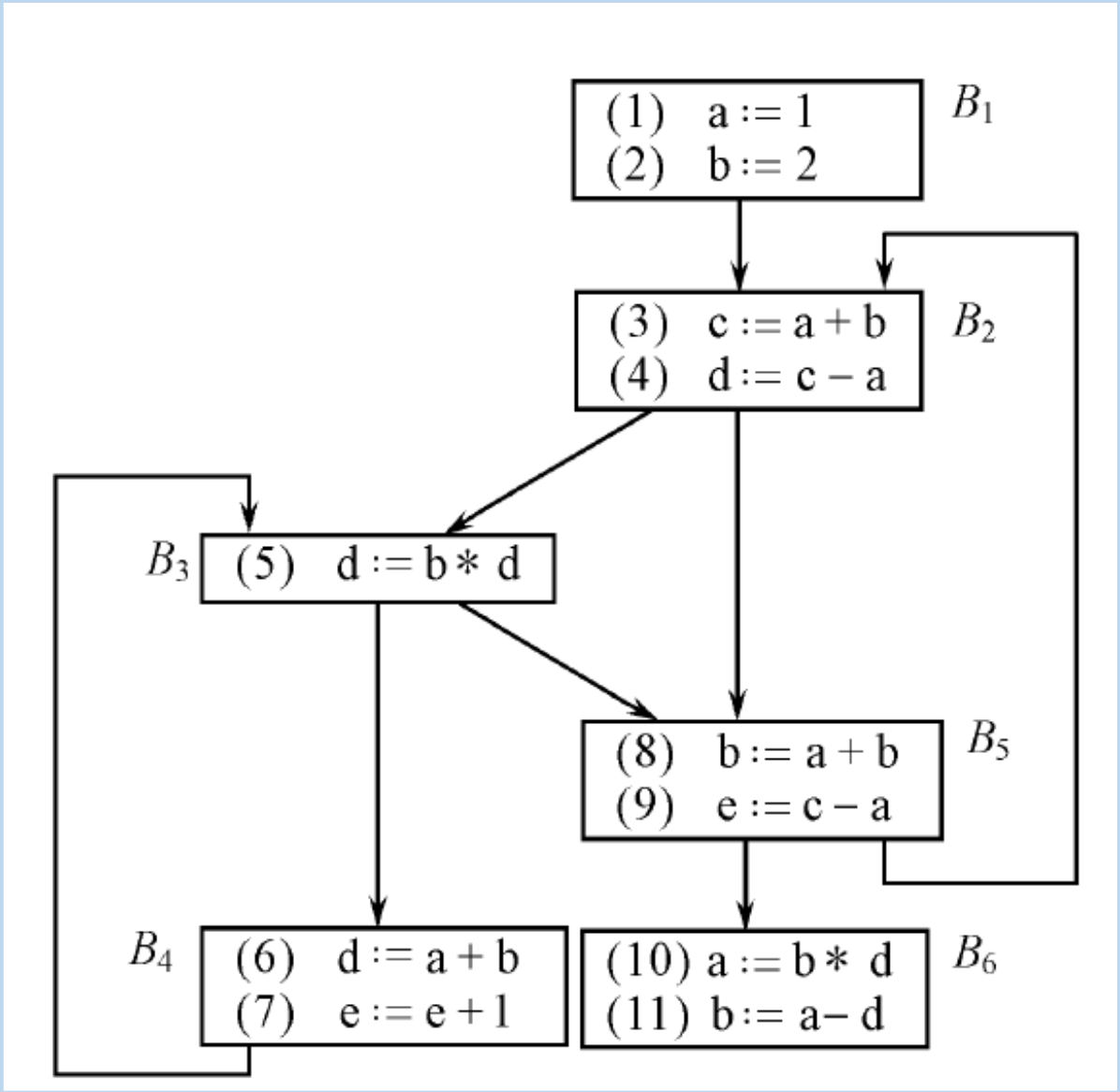
H12

- 9.1 c. 识别每个循环的全局公共子表达式。
- {3,4}： 无
- {2,3,4,5}： a+b 和 c-a



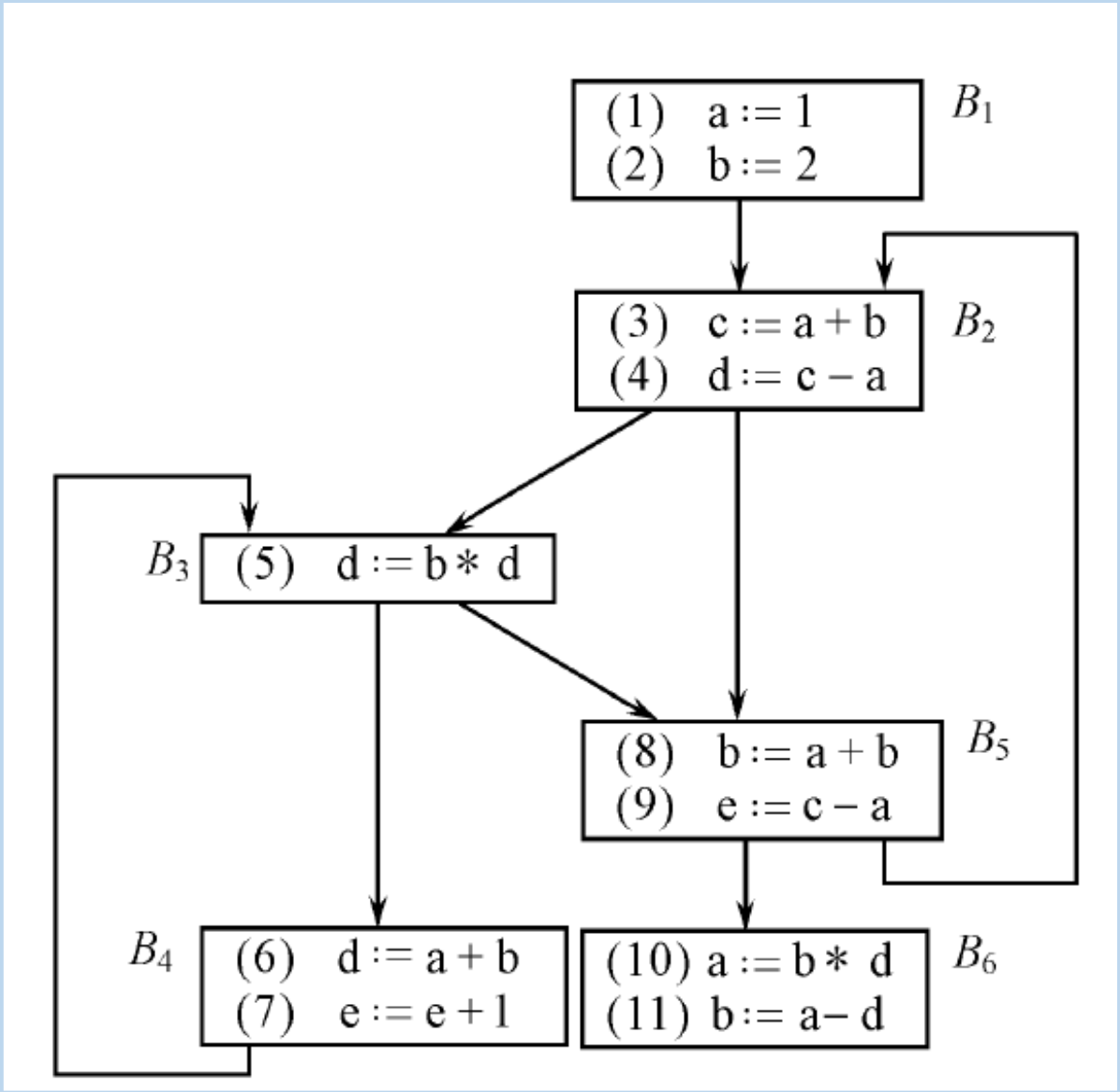
H12

- 9.1 d. 识别每个循环的归纳变量
- 归纳变量在循环的每一次迭代中增加固定的值
- {3,4}：e
- {2,3,4,5}：b，e，c



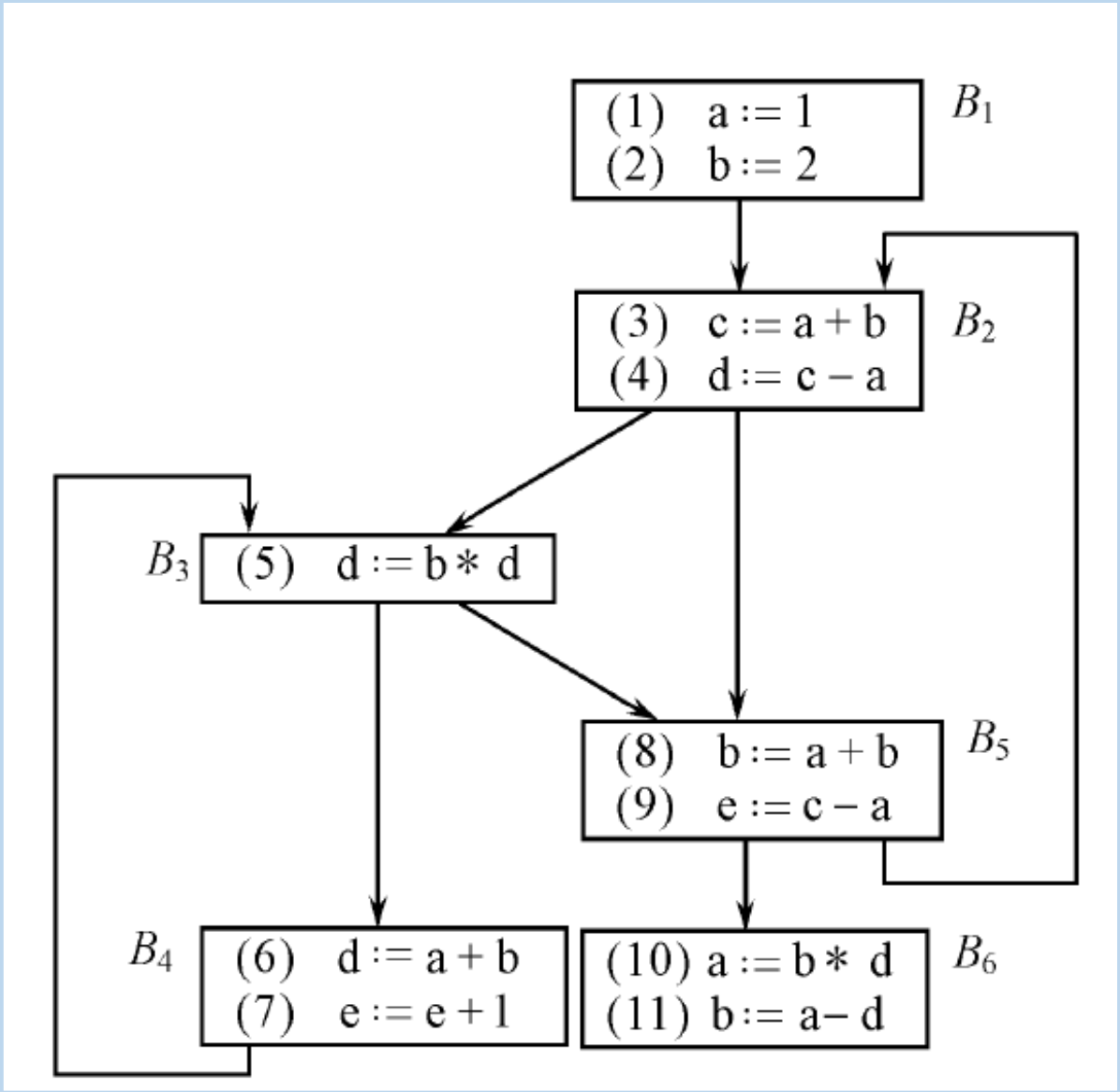
H12

- 9.1 e.识别每一个循环的不变计算
- {3,4}：a+b
- {2,3,4,5}：无



H12

- 9.3 a.为到达-定值分析，计算每个块的gen,kill,IN和OUT集合
- GEN[B1] = {d1,d2}
- KILL[B1] = {d8,d10,d11}
- GEN[B2] = {d3,d4}
- KILL[B2] = {d5,d6}
- GEN[B3] = {d5}
- KILL[B3] = {d4,d6}
- GEN[B4] = {d6,d7}
- KILL[B4] = {d4,d5,d9}
- GEN[B5] = {d8, d9}
- KILL[B5] = {d2,d11,d7}
- GEN[B6] = {d10,d11}
- KILL[B6] = {d1,d2,d8}



块	初始	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}		
B2	Ø				
B3	Ø				
B4	Ø				
B5	Ø				
B6	Ø				

块	初始	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}		
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}		
B3	Ø				
B4	Ø				
B5	Ø				
B6	Ø				

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}		
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}		
B3	Ø	{d1,d2,d3,d4}	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}		
B4	Ø				
B5	Ø				
B6	Ø				

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}		
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}		
B3	Ø	{d1,d2,d3,d4}	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}		
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}		
B5	Ø				
B6	Ø				

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}		
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}		
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}		
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}		
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}		
B6	Ø				

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}		
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}		
B3	Ø	{d1,d2,d3,d4}	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}		
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}		
B5	Ø	{d1,d2,d3,d4} U {d1,d2,d3,d5} = {d1,d2,d3,d4,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}		
B6	Ø	{d1,d3,d4,d5,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}		

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}	Ø	{d1,d2}
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}	{d1,d2} U {d1,d3,d4,d5,d8,d9} = {d1,d2,d3,d4,d5,d8,d9}	{d3,d4} + ({d1,d2,d3,d4,d5,d8,d9}- {d5,d6}) = {d1,d2,d3,d4,d6,d8,d9}
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}		
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}		
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}		
B6	Ø	{d1,d3,d4,d5 ,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}		

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}	Ø	{d1,d2}
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}	{d1,d2} U {d1,d3,d4,d5,d8,d9} = {d1,d2,d3,d4,d5,d8,d9}	{d3,d4} + ({d1,d2,d3,d4,d5,d8,d9}- {d5,d6}) = {d1,d2,d3,d4,d6,d8,d9}
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d6,d7} = {d1,d2,d3,d4,d6,d7,d8, d9}	{d5} + ({d1,d2,d3,d4,d6,d7,d8,d9} - {d4,d6}) = {d1,d2,d3,d5,d7,d8,d9}
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}		
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}		
B6	Ø	{d1,d3,d4,d5 ,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}		

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}	Ø	{d1,d2}
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}	{d1,d2} U {d1,d3,d4,d5,d8,d9} = {d1,d2,d3,d4,d5,d8,d9}	{d3,d4} + ({d1,d2,d3,d4,d5,d8,d9}- {d5,d6}) = {d1,d2,d3,d4,d6,d8,d9}
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d6,d7} = {d1,d2,d3,d4,d6,d7,d8, d9}	{d5} + ({d1,d2,d3,d4,d6,d7,d8,d9} - {d4,d6}) = {d1,d2,d3,d5,d7,d8,d9}
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}	{d1,d2,d3,d5,d7,d8,d9}	{d6,d7} + {d1,d2,d3,d5,d7,d8,d9} - {d4,d5,d9}) = {d1,d2,d3,d6,d7,d8}
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}		
B6	Ø	{d1,d3,d4,d5 ,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}		

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}	Ø	{d1,d2}
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}	{d1,d2} U {d1,d3,d4,d5,d8,d9} = {d1,d2,d3,d4,d5,d8,d9}	{d3,d4} + ({d1,d2,d3,d4,d5,d8,d9}- {d5,d6}) = {d1,d2,d3,d4,d6,d8,d9}
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d6,d7} = {d1,d2,d3,d4,d6,d7,d8, d9}	{d5} + ({d1,d2,d3,d4,d6,d7,d8,d9} - {d4,d6}) = {d1,d2,d3,d5,d7,d8,d9}
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}	{d1,d2,d3,d5,d7,d8,d9}	{d6,d7} + {d1,d2,d3,d5,d7,d8,d9} - {d4,d5,d9}) = {d1,d2,d3,d6,d7,d8}
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d5,d7,d8,d9} ={d1,d2,d3,d4,d5,d6,d 7,d8,d9}	{d8, d9} + ({d1,d2,d3,d4,d5,d6,d7,d8,d9}- {d2,d11,d7}) = {d1,d3,d4,d5,d6,d8,d9}
B6	Ø	{d1,d3,d4,d5 ,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}		

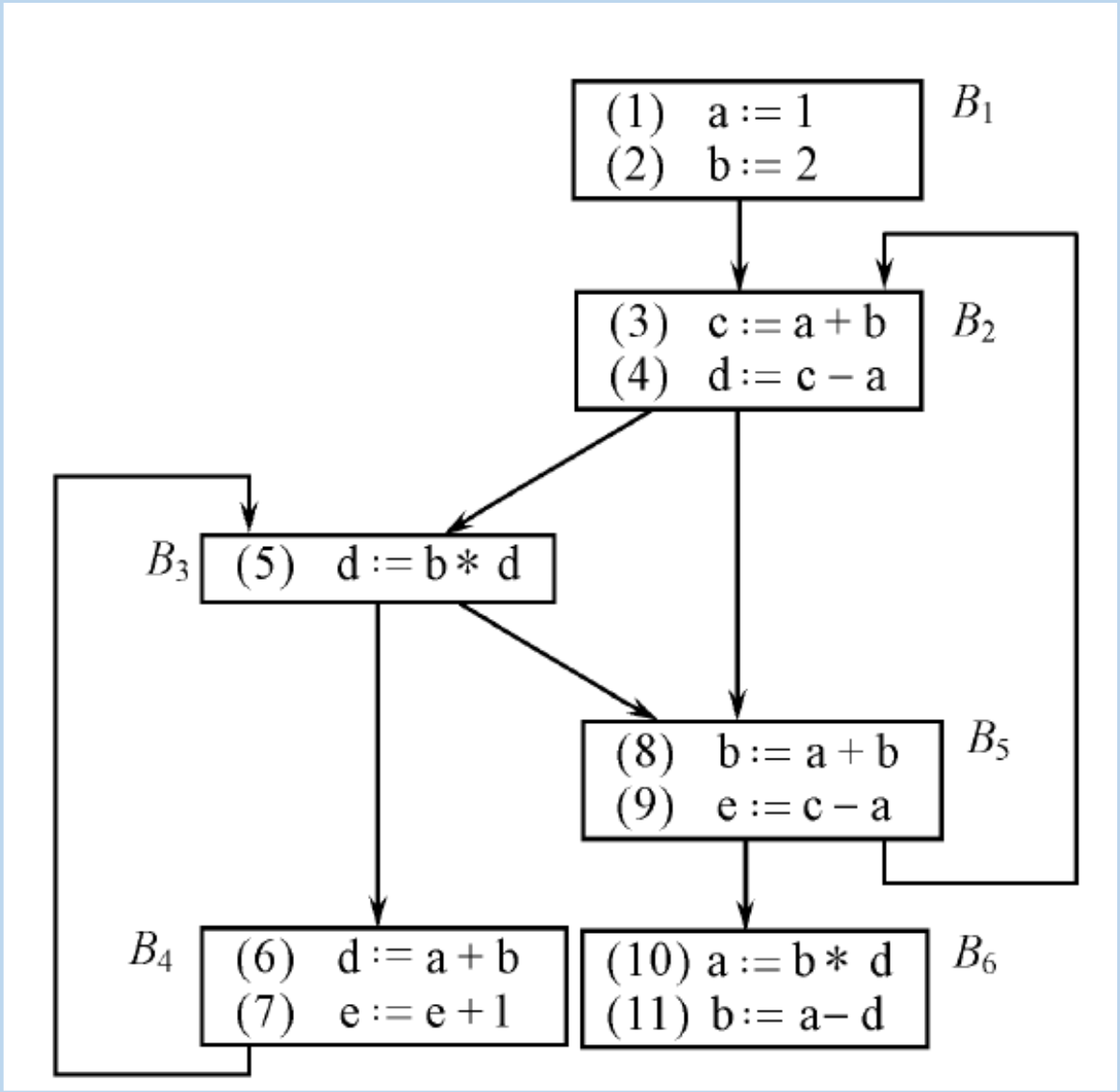
块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}	Ø	{d1,d2}
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}	{d1,d2} U {d1,d3,d4,d5,d8,d9} = {d1,d2,d3,d4,d5,d8,d9}	{d3,d4} + ({d1,d2,d3,d4,d5,d8,d9}- {d5,d6}) = {d1,d2,d3,d4,d6,d8,d9}
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d6,d7} = {d1,d2,d3,d4,d6,d7,d8, d9}	{d5} + ({d1,d2,d3,d4,d6,d7,d8,d9} - {d4,d6}) = {d1,d2,d3,d5,d7,d8,d9}
B4	Ø	{d1,d2,d3,d5}	{d6,d7} + ({d1,d2,d3,d5} - {d4,d5,d9}) = {d1,d2,d3,d6,d7}	{d1,d2,d3,d5,d7,d8,d9}	{d6,d7} + {d1,d2,d3,d5,d7,d8,d9} - {d4,d5,d9}) = {d1,d2,d3,d6,d7,d8}
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	{d8, d9} + ({d1,d2,d3,d4,d5} -{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d5,d7,d8,d9} ={d1,d2,d3,d4,d5,d6,d 7,d8,d9}	{d8, d9} + ({d1,d2,d3,d4,d5,d6,d7,d8,d9}- {d2,d11,d7}) = {d1,d3,d4,d5,d6,d8,d9}
B6	Ø	{d1,d3,d4,d5 ,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}	{d1,d3,d4,d5,d6,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d6,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d6,d9,d10,d11}

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	Ø	Ø	{d1,d2} U (Ø - {d8,d10,d11}) = {d1,d2}	Ø	{d1,d2}
B2	Ø	{d1,d2}	{d3,d4} + ({d1,d2} - {d5,d6}) = {d1,d2,d3,d4}	{d1,d2} U {d1,d3,d4,d5,d8,d9} = {d1,d2,d3,d4,d5,d8,d9}	{d3,d4} + ({d1,d2,d3,d4,d5,d8,d9}- {d5,d6}) = {d1,d2,d3,d4,d6,d8,d9}
B3	Ø	{d1,d2,d3,d4 }	{d5} + ({d1,d2,d3,d4} - {d4,d6}) = {d1,d2,d3,d5}	{d1,d2,d3,d4,d6,d8,d9} U {d1,d2,d3,d6,d7} = {d1,d2,d3,d4,d6,d7,d8, d9}	{d5} + ({d1,d2,d3,d4,d6,d7,d8,d9} - {d4,d6}) = {d1,d2,d3,d5,d7,d8,d9}
B4	Ø	{d1,d2,d3,d5}			7} + {d1,d2,d3,d5,d7,d8,d9} d5,d9}) = {d1,d2,d3,d6,d7,d8}
B5	Ø	{d1,d2,d3,d4 } U {d1,d2,d3,d5} = {d1,d2,d3,d4 ,d5}	-{d2,d11,d7}) = {d1,d3,d4,d5,d8,d9}	U {d1,d2,d3,d5,d7,d8,d9} ={d1,d2,d3,d4,d5,d6,d 7,d8,d9}	9} + ({d1,d2,d3,d4,d5,d6,d7,d8,d9}- {d2,d11,d7}) = {d1,d3,d4,d5,d6,d8,d9}
B6	Ø	{d1,d3,d4,d5 ,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d9,d10,d11}	{d1,d3,d4,d5,d6,d8,d9}	{d10,d11} + ({d1,d3,d4,d5,d6,d8,d9} - {d1,d2,d8}) = {d3,d4,d5,d6,d9,d10,d11}

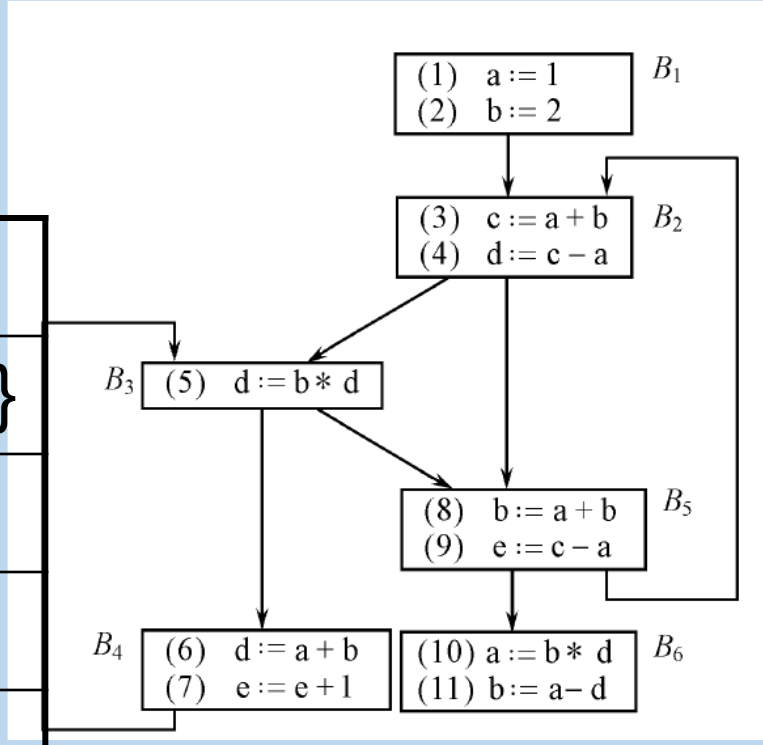
继续迭代直到out没有变化，此处略去，由于时间关系，有可能计算有误，做一个免责声明。

H12

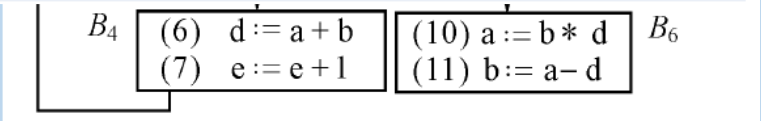
- 9.3 b.为可用表达式分析，计算每个块的e_gen,e_kill,IN和OUT集合



基本块	e_gen	e_kill
B_1	{1,2}	{ a+b, c-a, b*d, a-d}
B_2	{ a+b, c-a}	{ b*d, c-a, a-d}
B_3	Ø	{ b*d, a-d }
B_4	{ a+b}	{ b*d, a-d }
B_5	{c-a}	{ a+b, b*d, e+1}
B_6	{a-d}	{1,2,a+b}
全部表达式U = {1,2,a+b,c-a,b*d,e+1,a-d}		



块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	U	Ø	{1,2} U (Ø - { a+b, c-a, b*d, a-d}) = {1,2}		
B2	U				
B3	U				
B4	U				
B5	U				
B6	U				



块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	U	Ø	{1,2} U (Ø - { a+b, c-a, b*d, a-d}) = {1,2}		
B2	U	U	{a+b, c-a} U (U-{ b*d, c-a, a-d}) = {1,2,a+b,c-a,e+1}		
B3	U				
B4	U				
B5	U				
B6	U				

块	OUT[B]	IN[B]_1	OUT[B]_1	IN[B]_2	OUT[B]_2
B1	U	Ø	{1,2} U (Ø - { a+b, c-a, b*d, a-d}) = {1,2}		
B2	U	U	{a+b, c-a} U (U-{ b*d, c-a, a-d}) = {1,2,a+b,c-a,e+1}		
B3	U				
B4	U				
B5	U				
B6	U				

H12

- 9.3 c.为活跃变量分析，计算每个块的def,use,IN和OUT集合

- 迭代计算

  OUT[B] = ∪ IN[S], S∈Succ(B)

  IN[B] = USE[B] ∪ (OUT[B]-DEF[B])

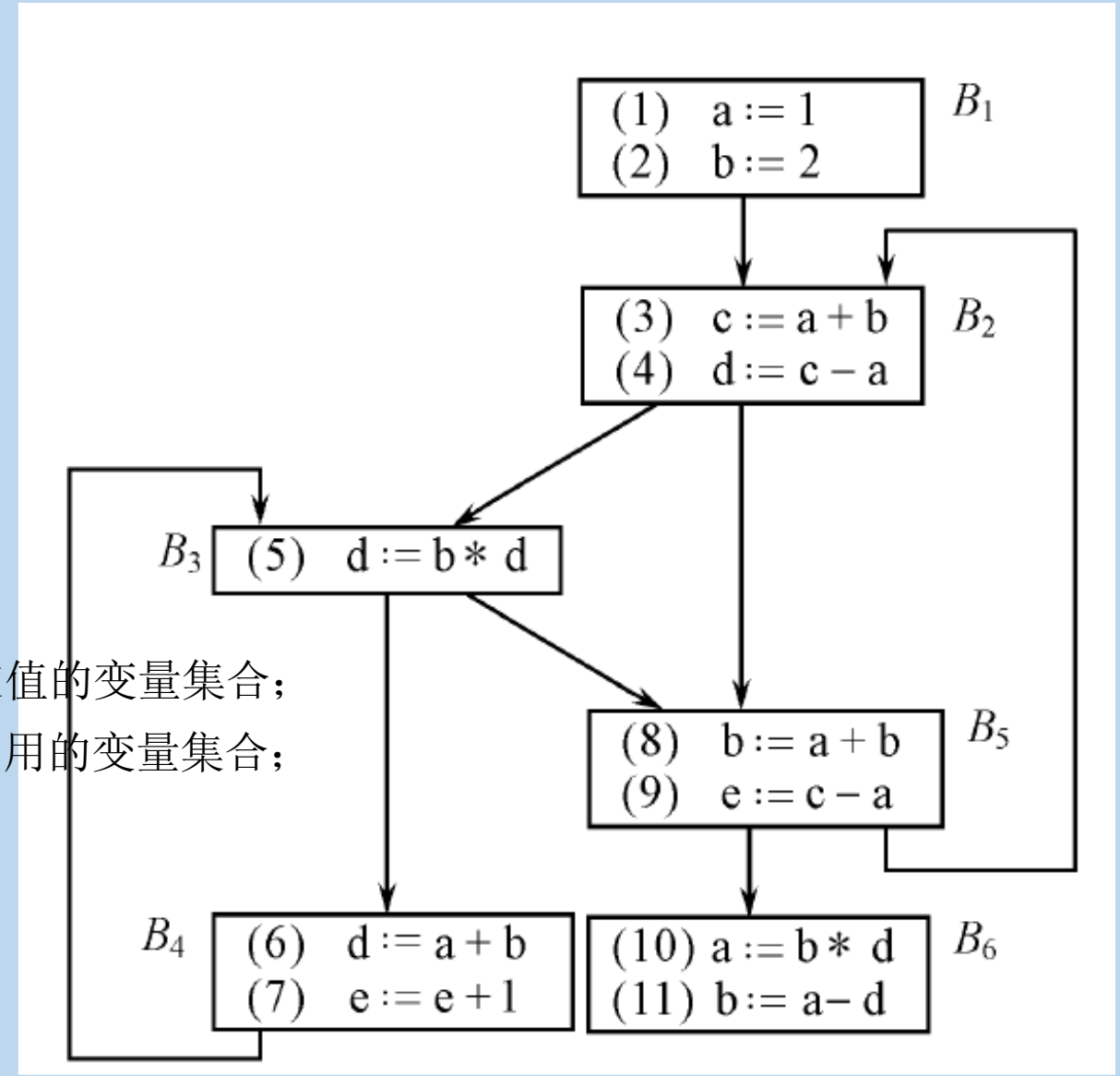
  USE[B]—基本块B中有引用且该引用前无定值的变量集合；

  DEF[B]—基本块B中有定值且该定值前无引用的变量集合；

- 计算次序

  — 结点深度优先序的逆序（向后流）：

  — B6 → B5 → B4 → B3 → B2 → B1



H12

- 9.3 c.为活跃变量分析，计算每个块的def,use,IN和OUT集合

- 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] = { }

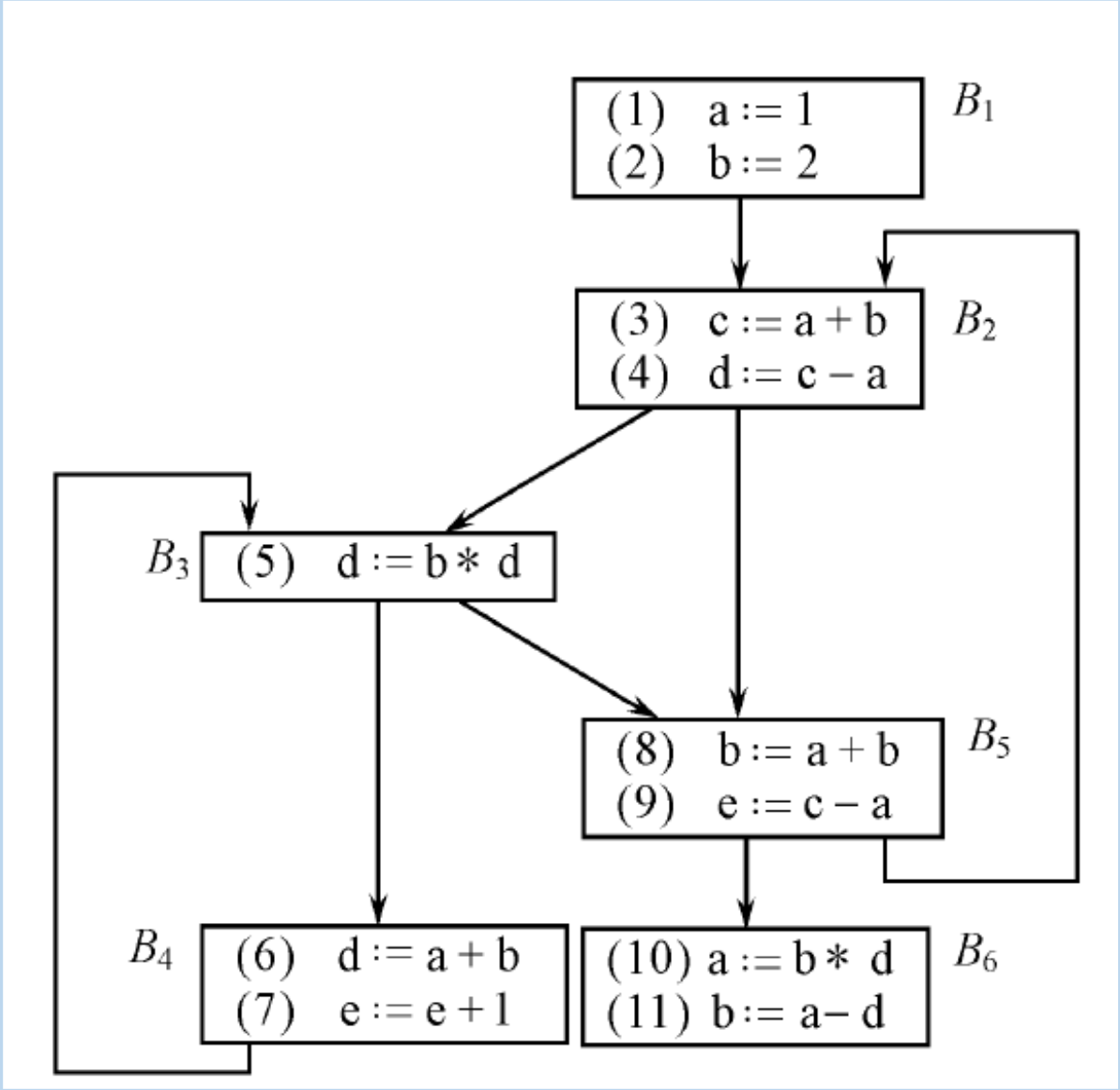
  USE[B4] = { a, b, e } ; DEF[B4] = { d }

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

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

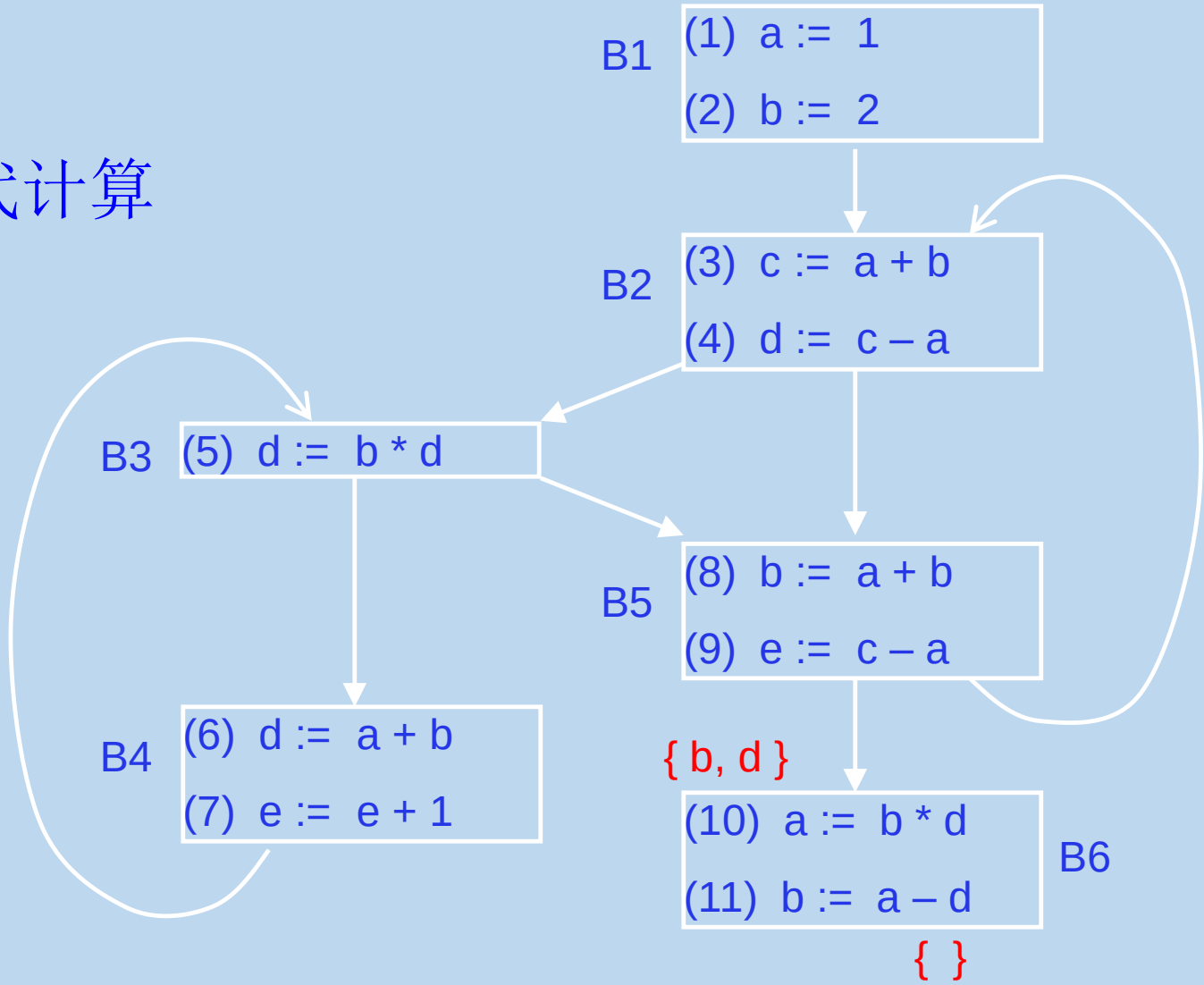
- 初始值，all B, IN[B] = { },

  OUT[B6]={ }//出口块



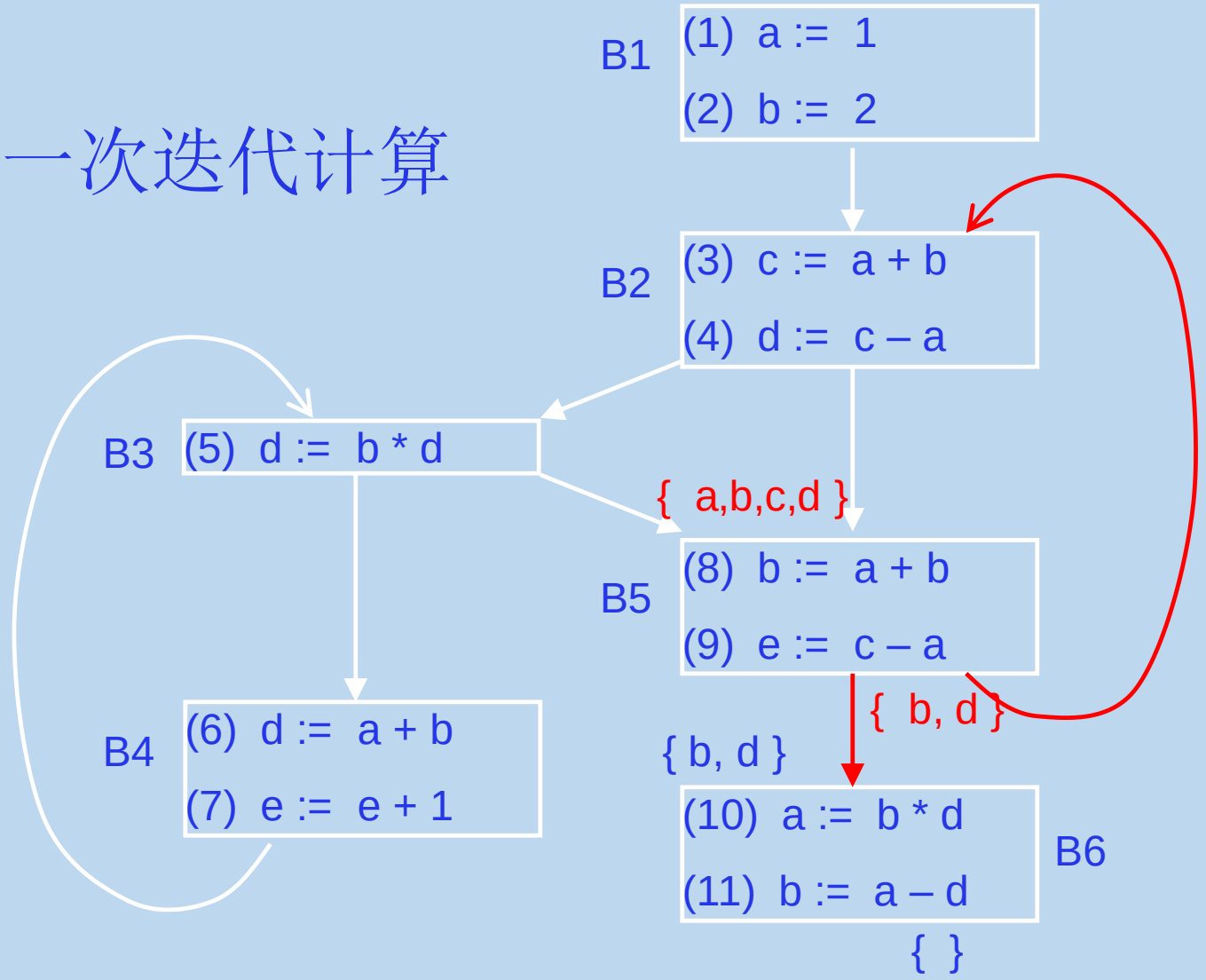
- 基本块出口活跃变量

- 第一次迭代计算



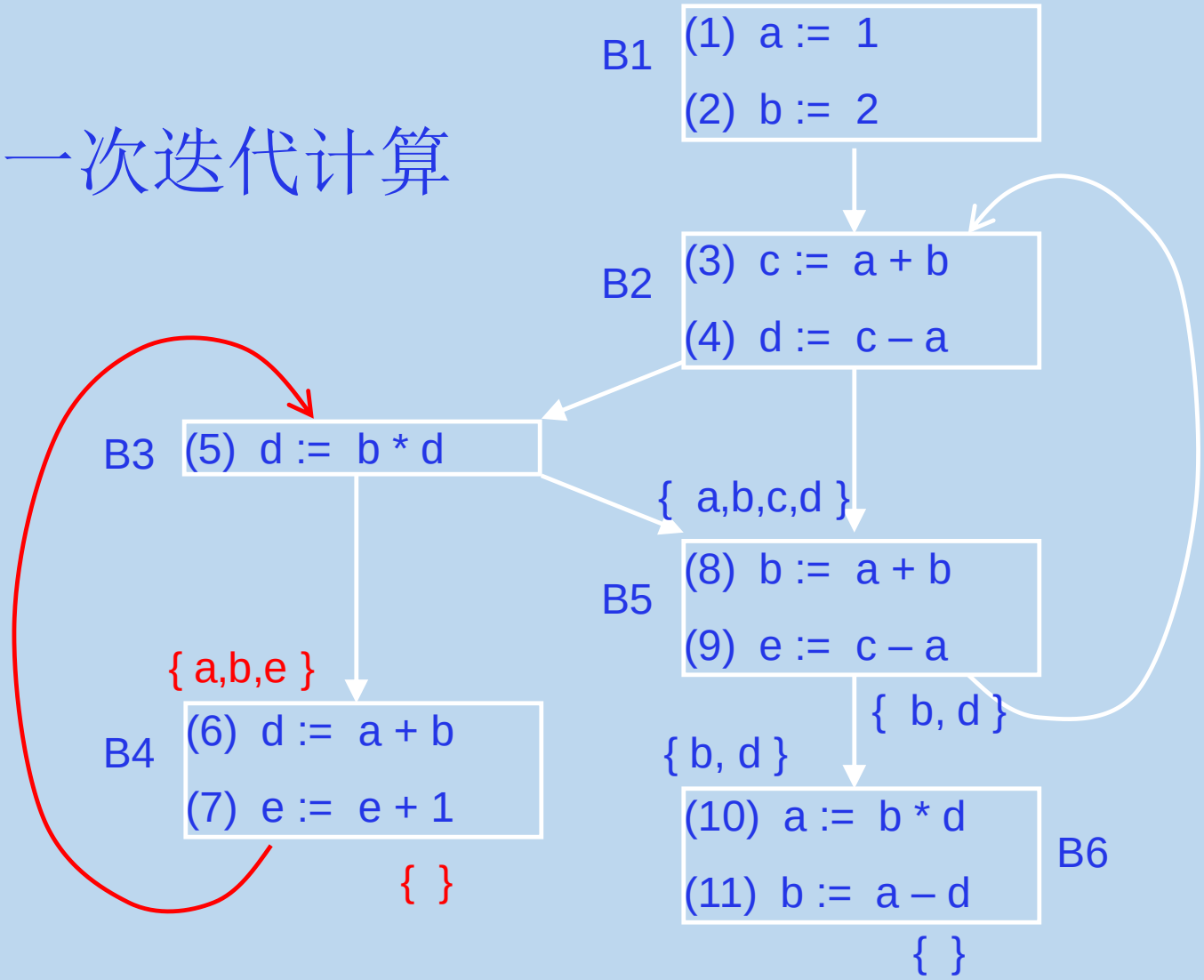
•基本块出口活跃变量

- 第一次迭代计算



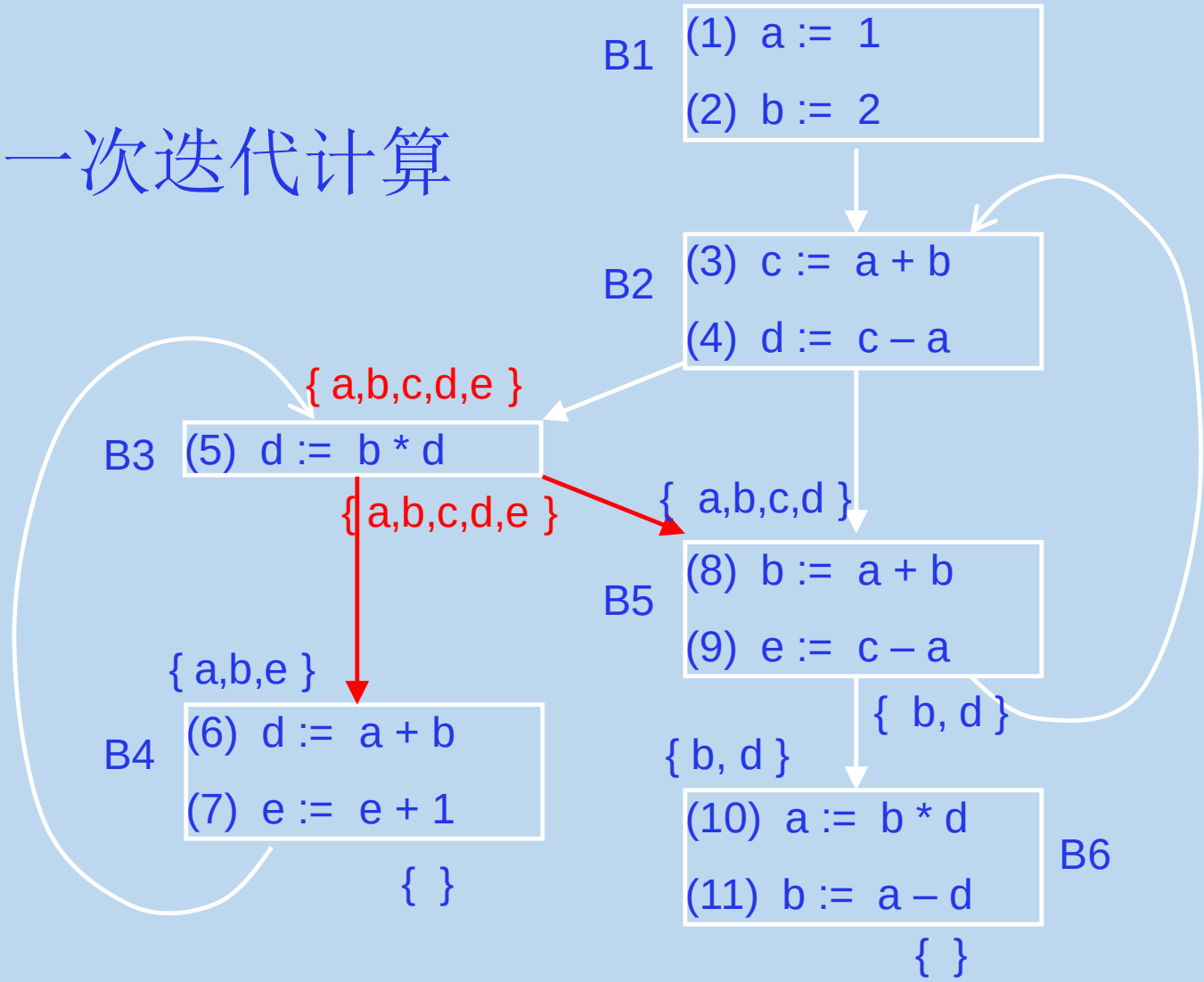
•基本块出口活跃变量

- 第一次迭代计算



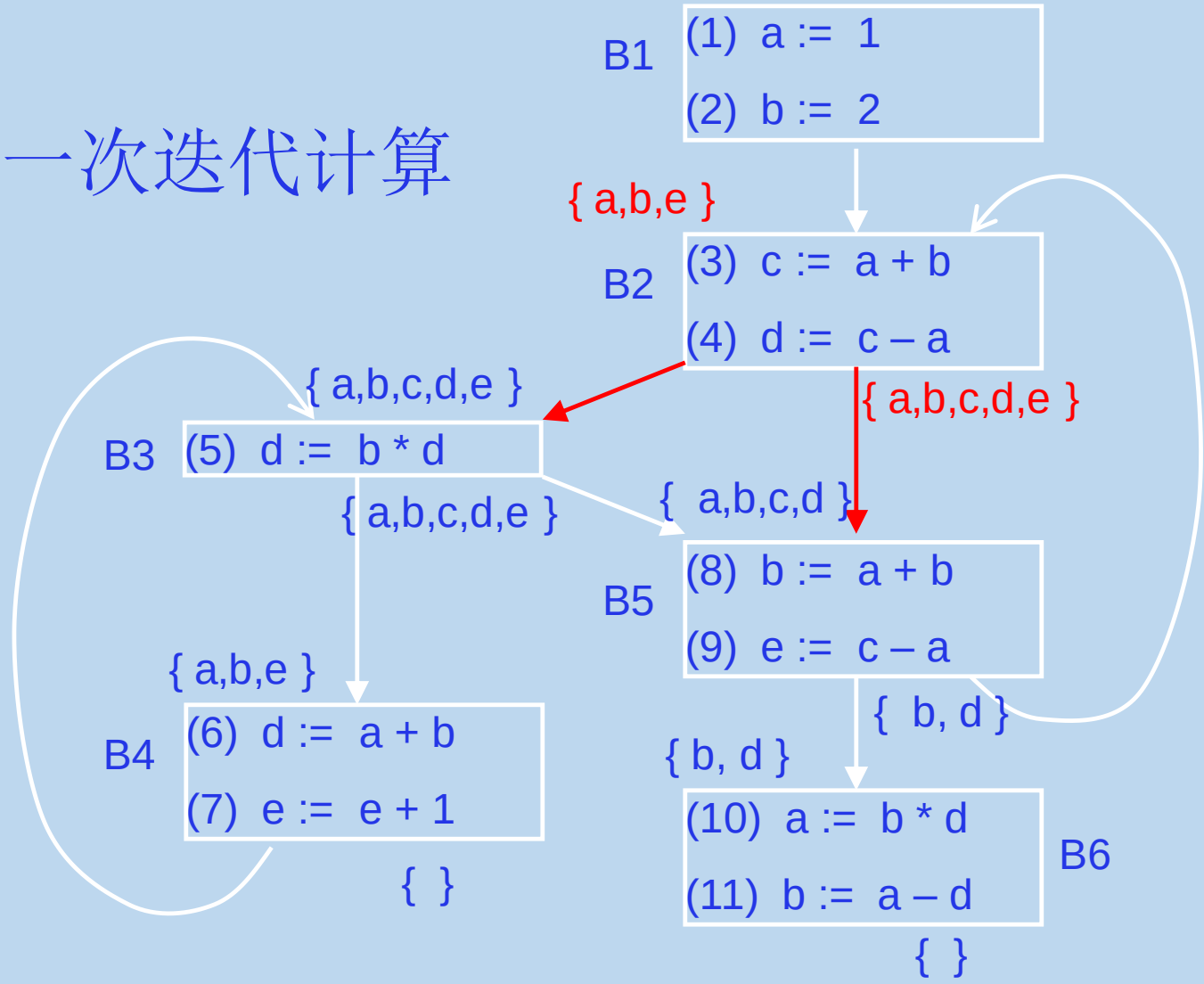
•基本块出口活跃变量

- 第一次迭代计算



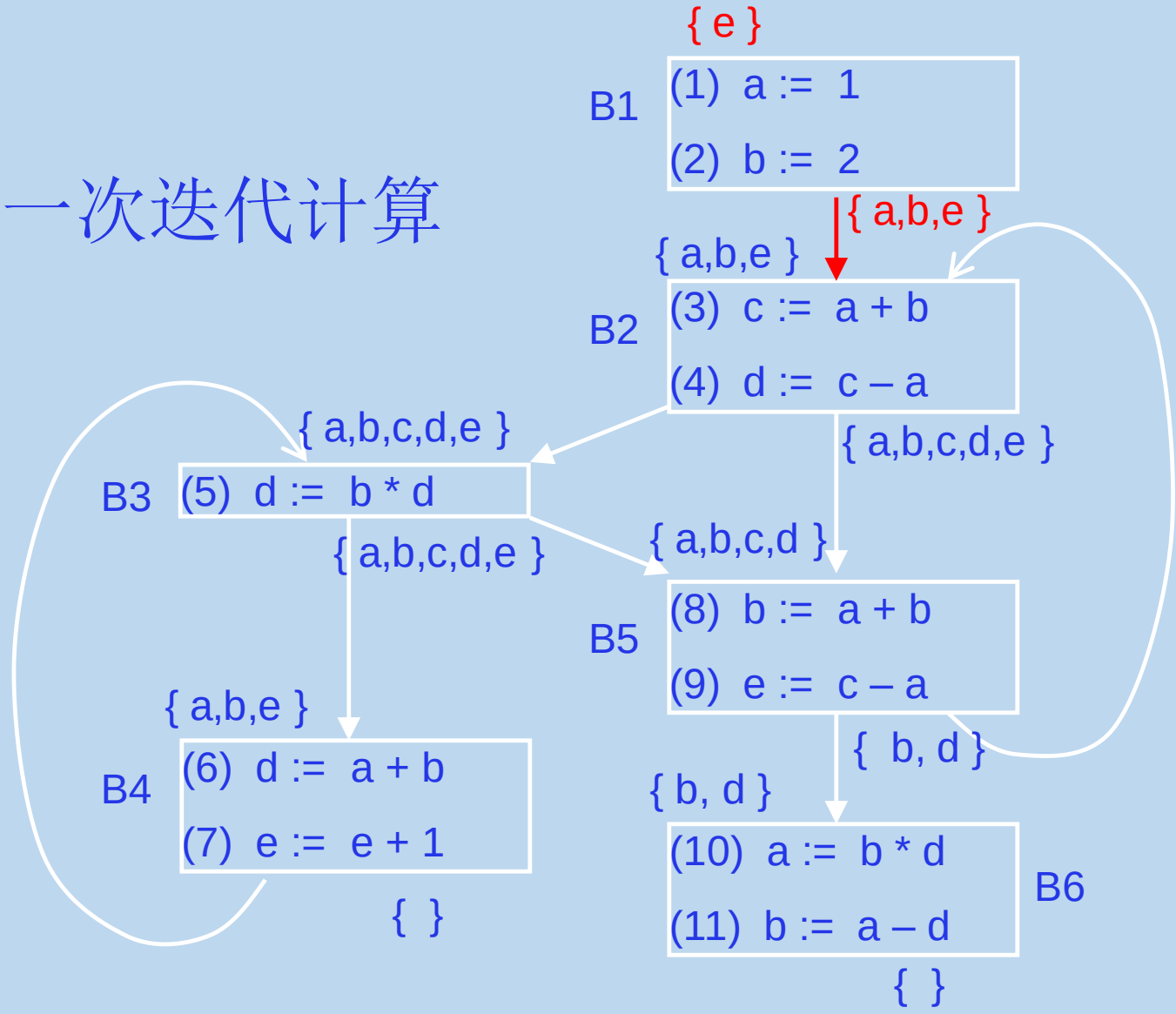
•基本块出口活跃变量

- 第一次迭代计算



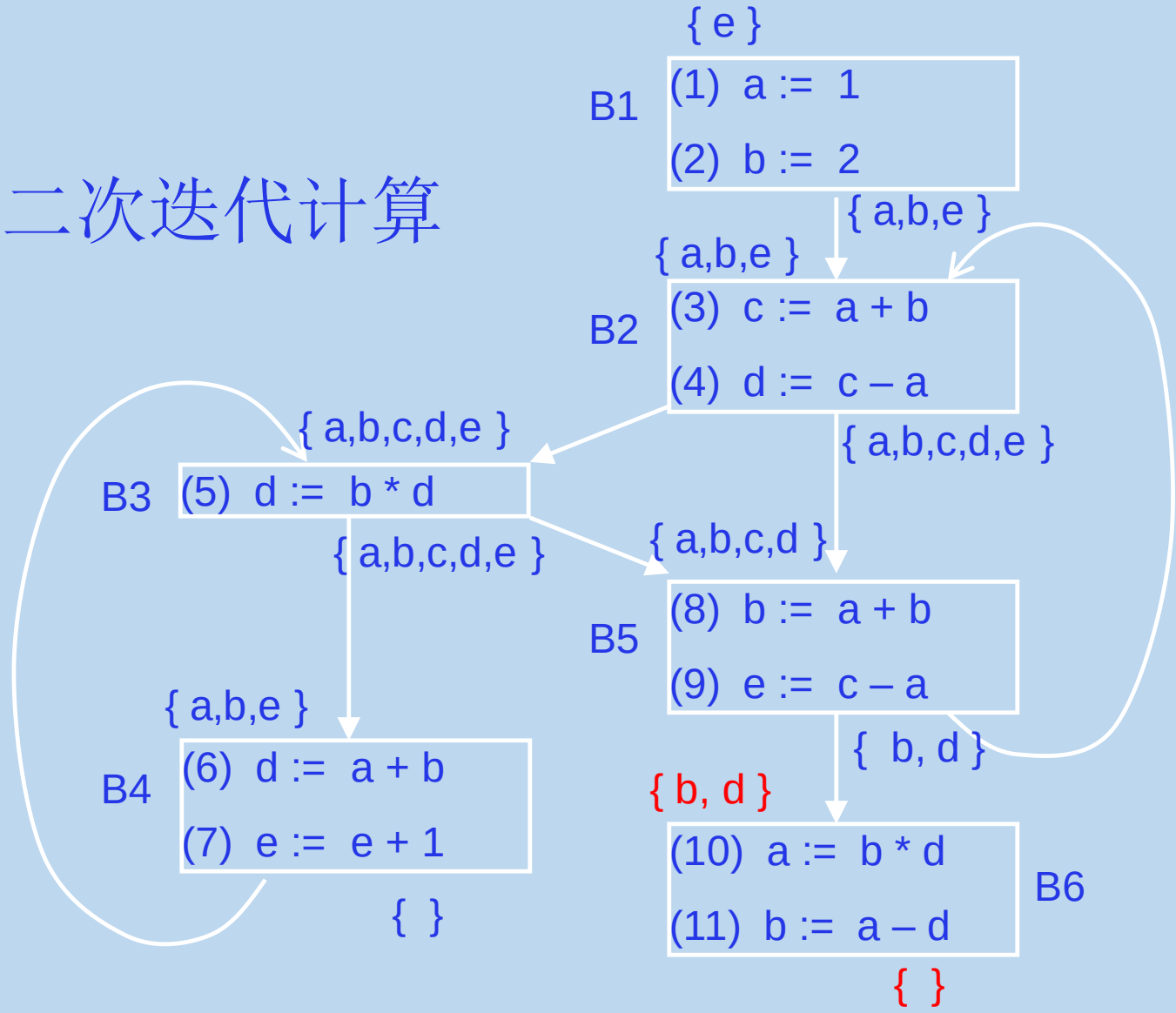
•基本块出口活跃变量

- 第一次迭代计算



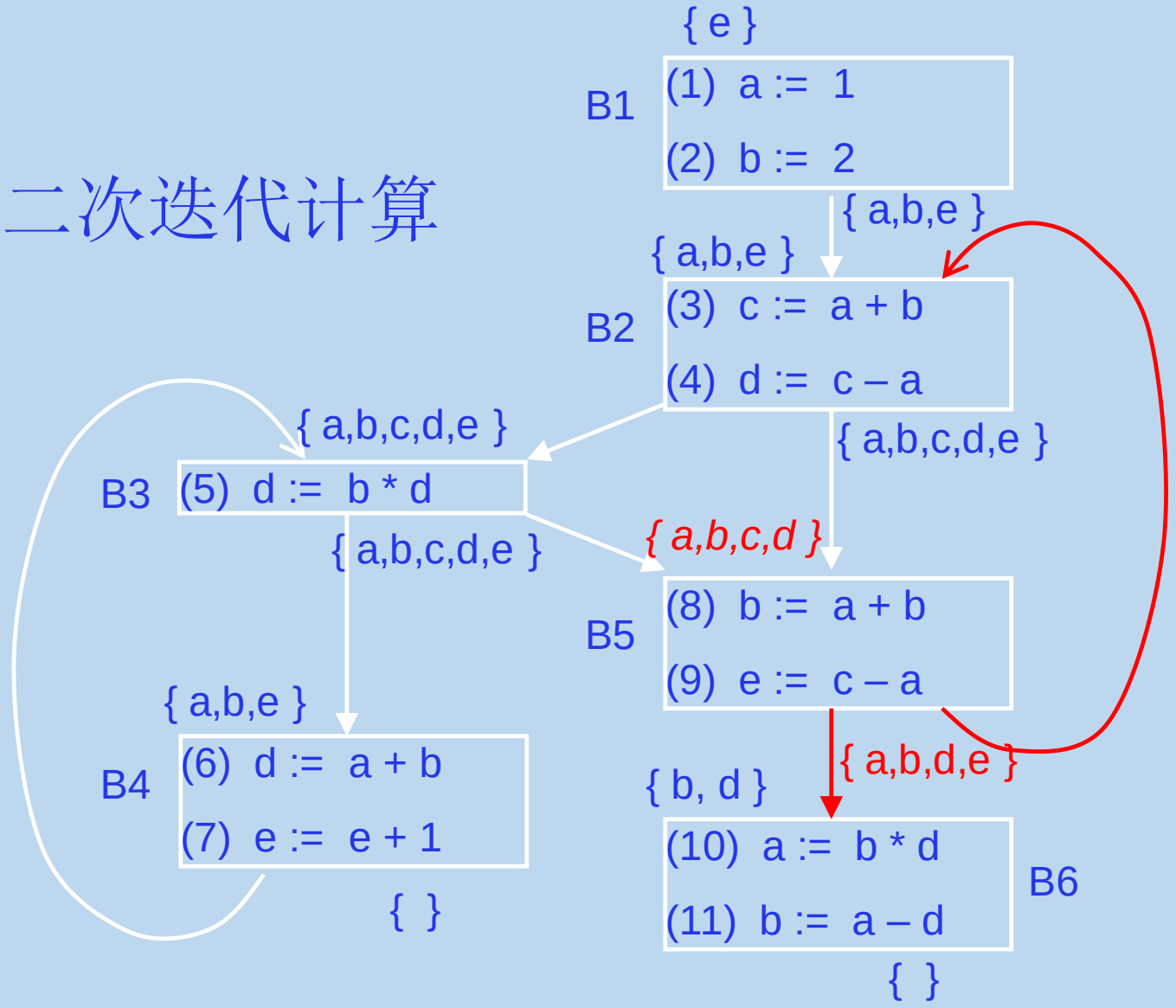
•基本块出口活跃变量

- 第二次迭代计算



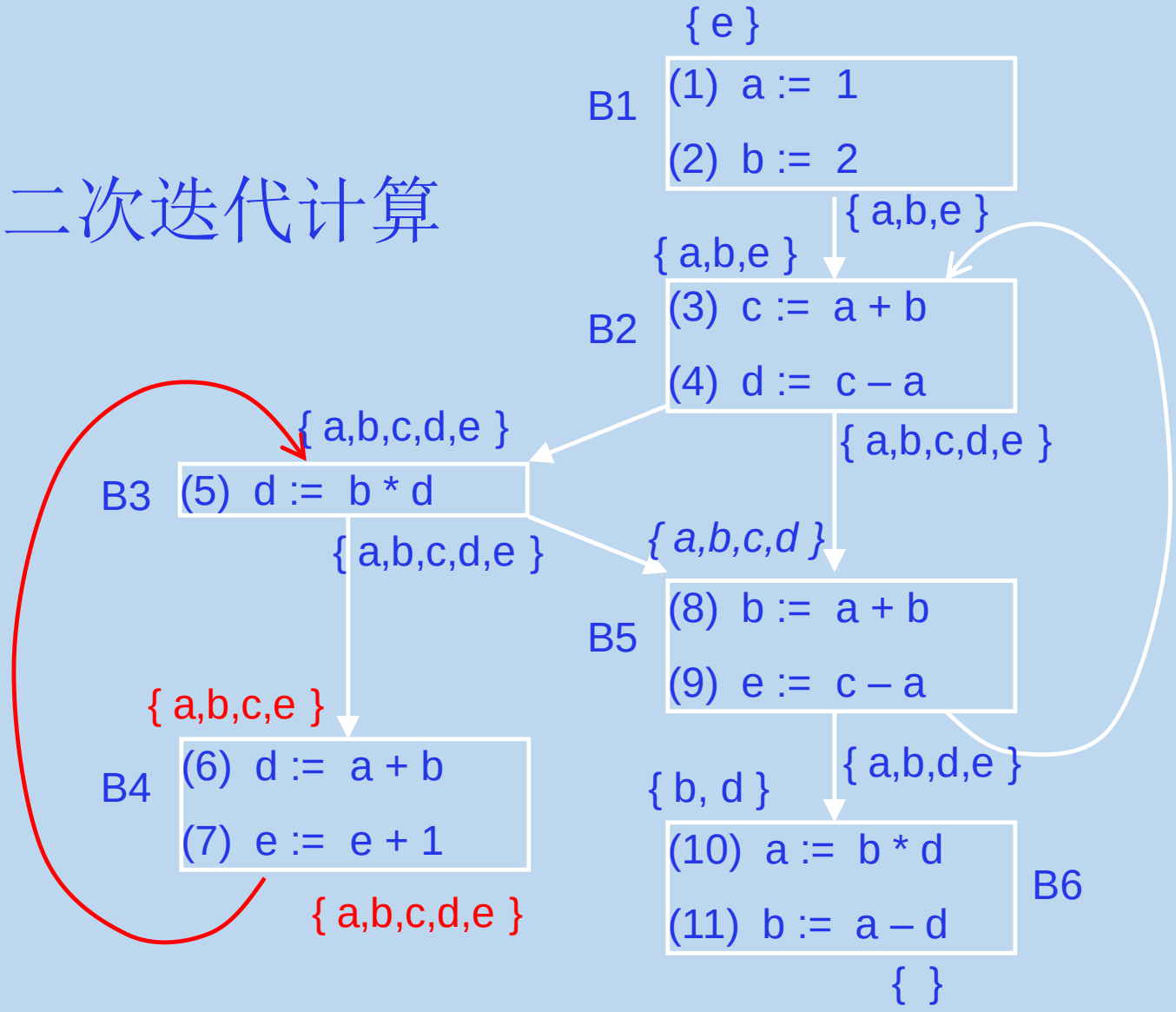
•基本块出口活跃变量

- 第二次迭代计算



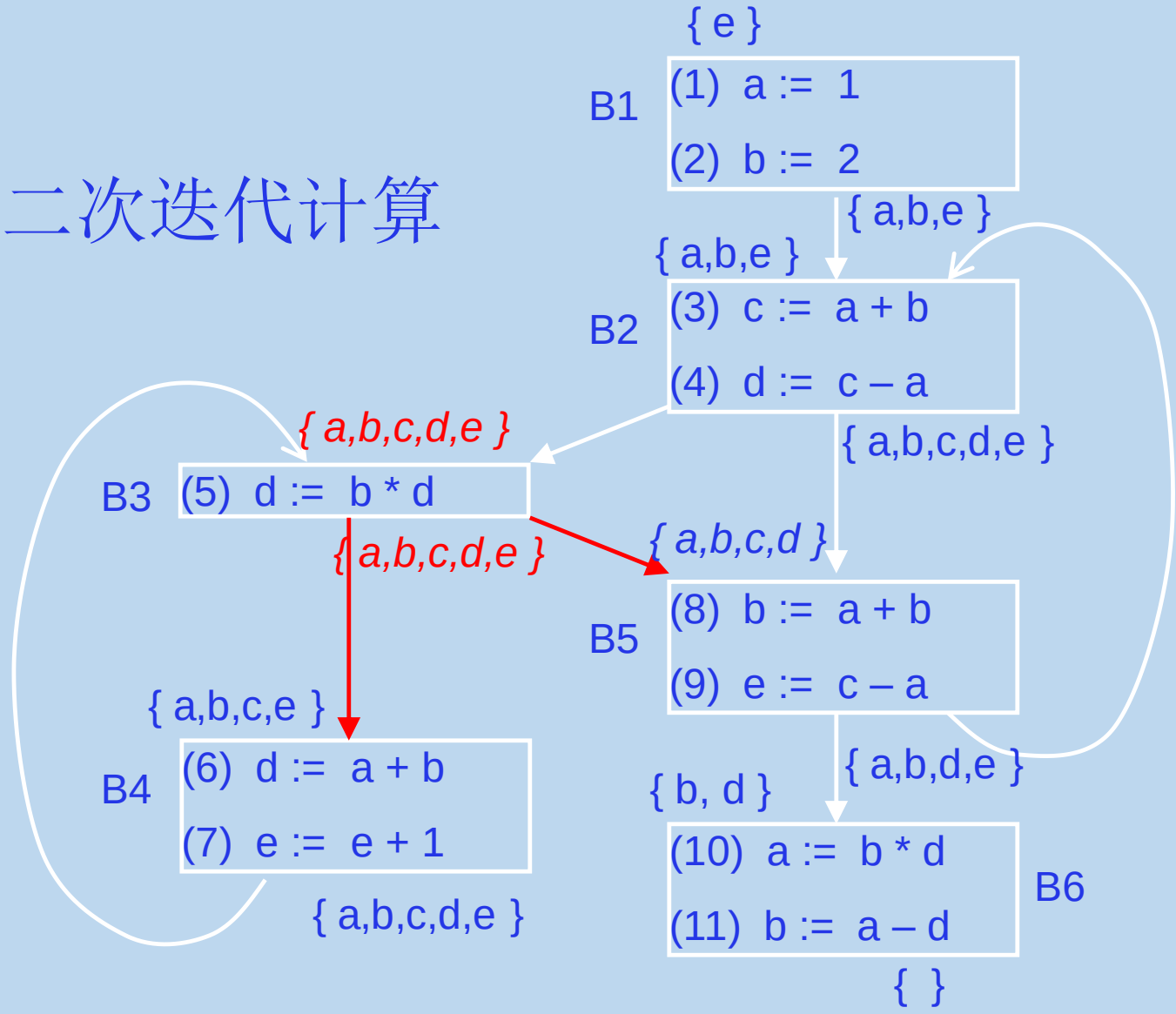
•基本块出口活跃变量

- 第二次迭代计算



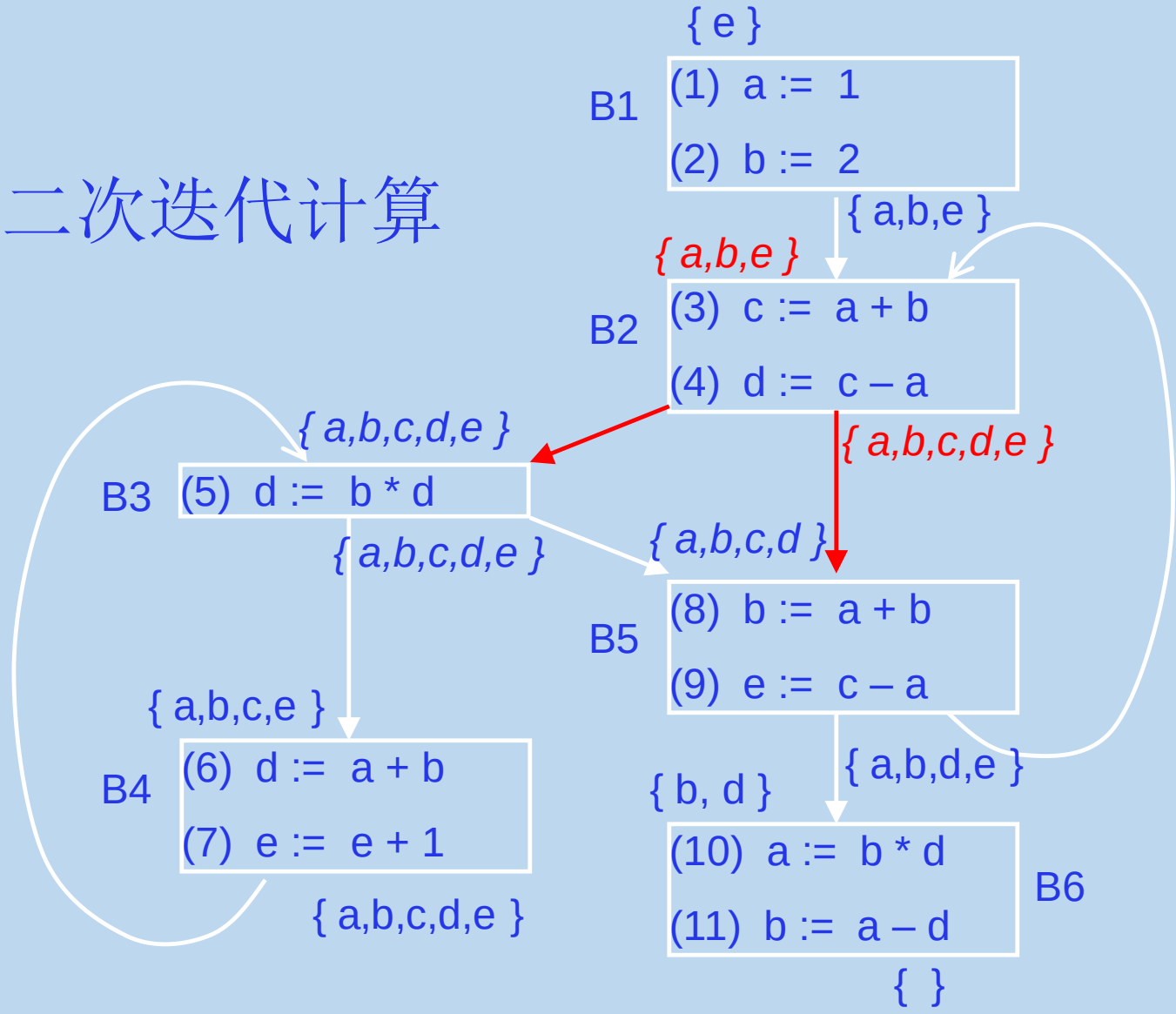
•基本块出口活跃变量

- 第二次迭代计算



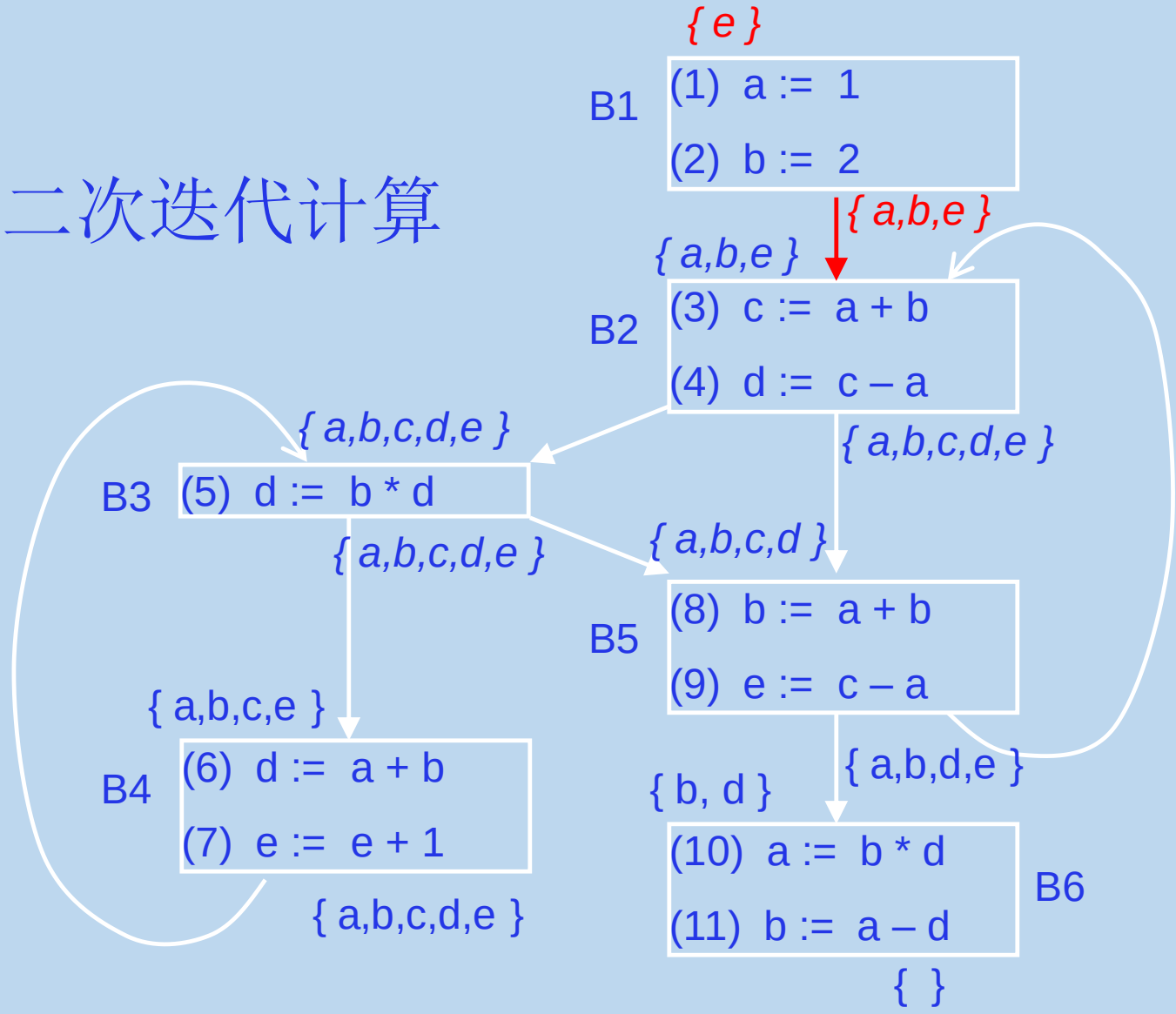
•基本块出口活跃变量

- 第二次迭代计算



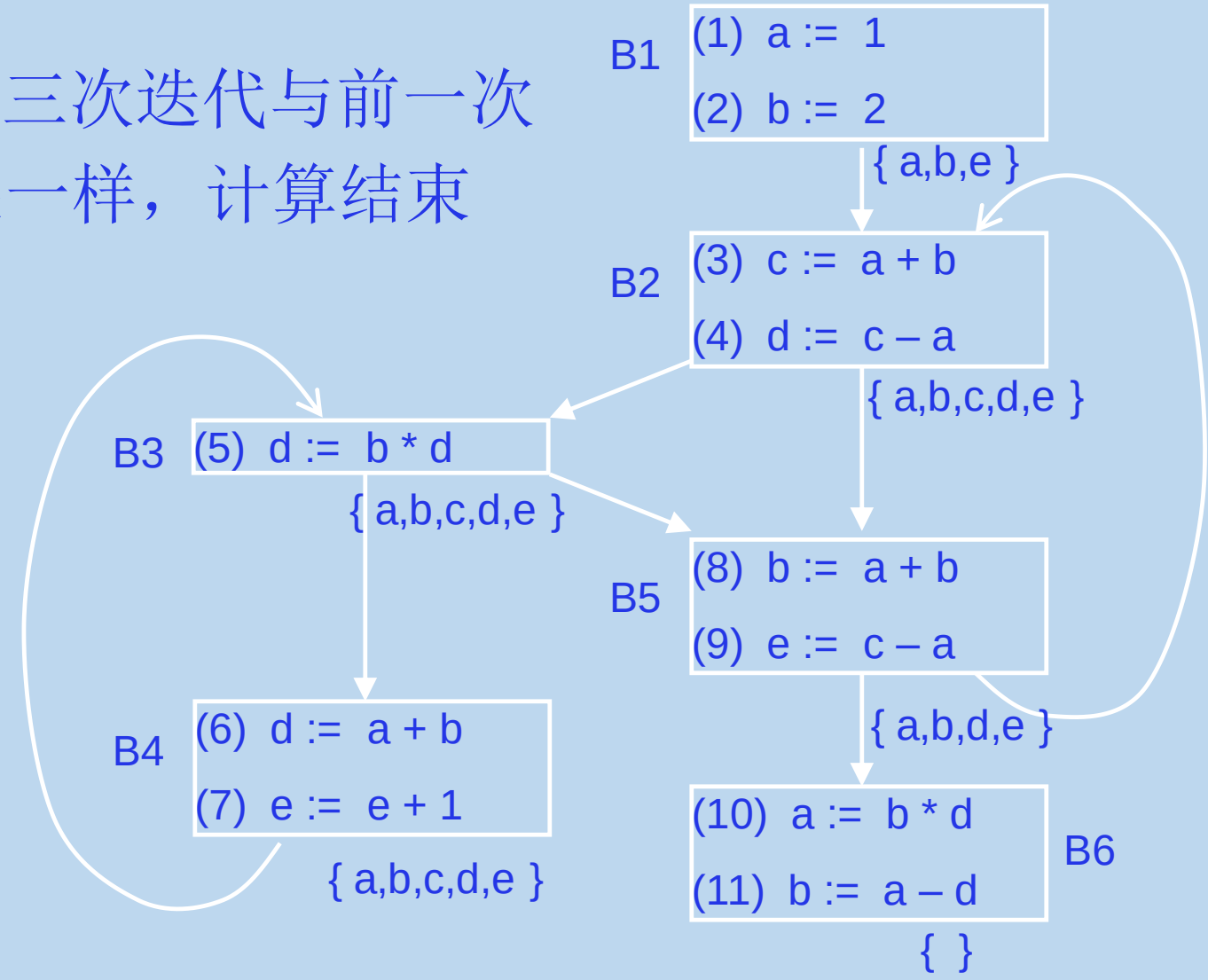
•基本块出口活跃变量

- 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



期中考试第二题

- 考虑如下的语言，其中的串可以划分成k>=0个子串，每个字串是以b为中心、2m个a组成的轴对称串(m>=1)；不同的子串可以有不同的m值。例如，语言包括串ε，aba，aabaaaba，abaaabaaaabaa。
  - (1) 为该语言设计一个LL(1)文法，并证明
  - $\mathrm{S} \rightarrow \mathrm{LS} \mid \varepsilon$
  - $\mathrm{L} \rightarrow aPa$
  - $\mathrm{P} \rightarrow aPa \mid b$

期中考试第二题

- 考虑如下的语言，其中的串可以划分成k>=0个子串，每个字串是以b为中心、2m个a组成的轴对称串(m>=1)；不同的子串可以有不同的m值。例如，语言包括串ε，aba，aabaaaba，abaaabaaaabaa。
  - (1) 为该语言设计一个LL(1)文法，并证明
  - S → LS | ε
  - L → aPa
  - P → $aPa \mid b$

  - First(LS) = {a}, $First(\varepsilon) = \{\varepsilon\}$, *无交集*
  - Follow(S) = {$}, $Follow(S)$与First(LS)无交集
  - First(aPa) = $\{a\}, First(b) = \{b\}$, 无交集

期中考试第二题

- 考虑如下的语言，其中的串可以划分成k>=0个子串，每个字串是以b为中心、2m个a组成的轴对称串(m>=1)；不同的子串可以有不同的m值。例如，语言包括串ε，aba，aabaaaba，abaaabaaaabaa。
  - (2) 为该语言设计一个二义的文法，并证明
  - S $\rightarrow$ SLS | ε
  - L $\rightarrow aPa$
  - P $\rightarrow aPa \mid b$

  - 对abaabaaba可以找出两个不同的最左推导
  - S ⇒ SLS ⇒ SLSLS ⇒ SLSLSLS ⇒ LSLSLS ⇒ aPaSLSLS ⇒ abaSLSLS ⇒ abaLSLS
  - S ⇒ SLS ⇒ SLSLS ⇒ LSLS ⇒ aPaSLS ⇒ abaSLS ⇒ abaSLSLS ⇒ abaLSLS

期中考试第二题

- 考虑如下的语言，其中的串可以划分成k>=0个子串，每个字串是以b为中心、2m个a组成的轴对称串(m>=1)；不同的子串可以有不同的m值。例如，语言包括串ε，aba，aabaaaba，abaaabaaaabaa。
  - (3) 为该语言设计一个非二义且非LR(1)的文法，并说明
  - S $\rightarrow MSL \mid \varepsilon$
  - M $\rightarrow \varepsilon$
  - L $\rightarrow aPa$
  - P $\rightarrow aPa \mid b$