第三次习题课-H10-2

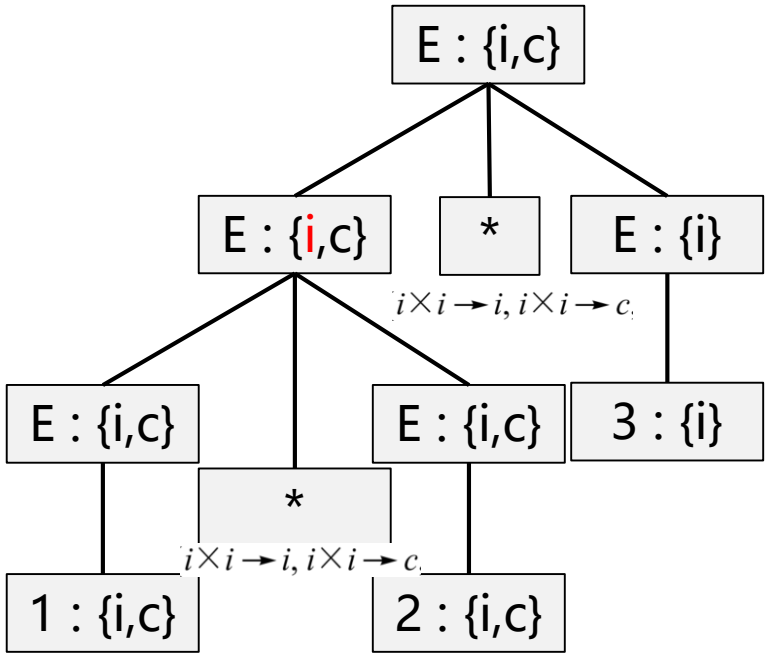
多态重载和中间表示

刘硕
zkdliushuo@mail.ustc.edu.cn

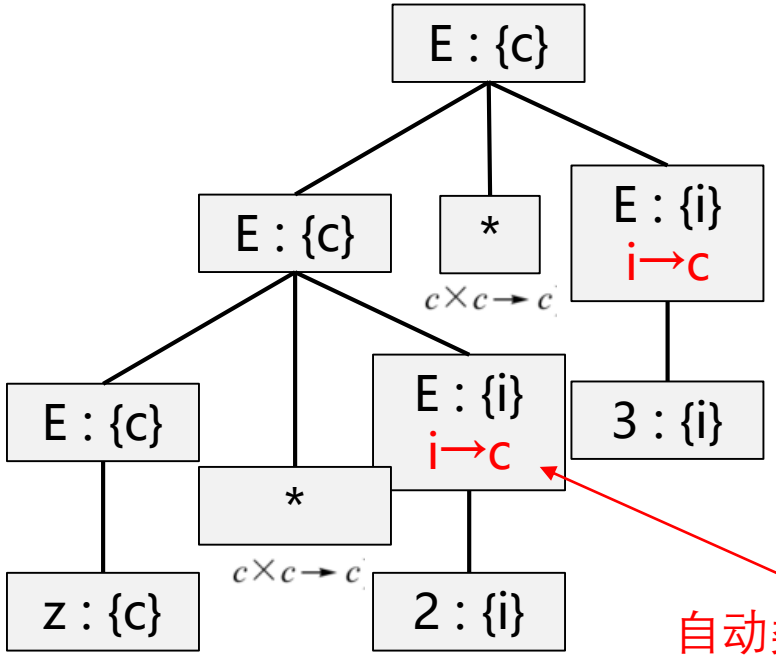
题一

- 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）

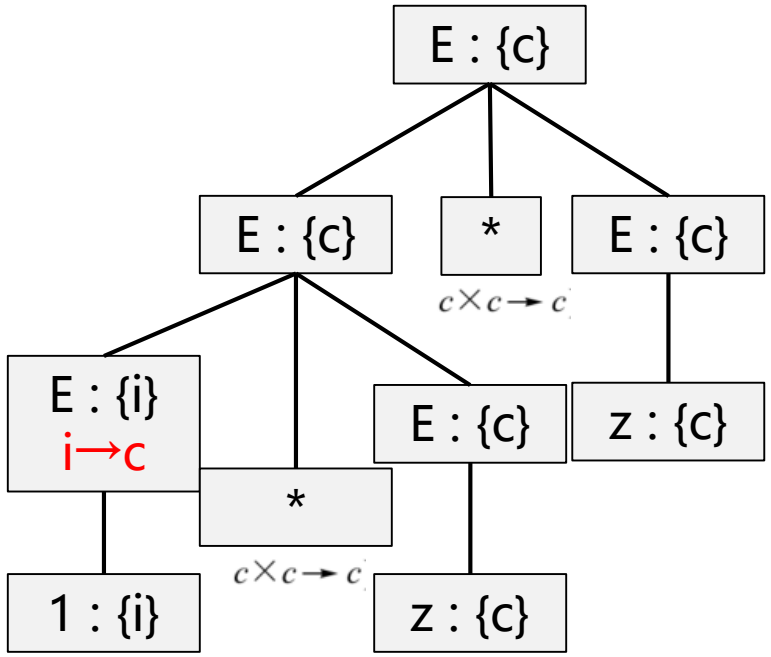
(a) 1 * 2 * 3



(b) 1 * (z * 2)

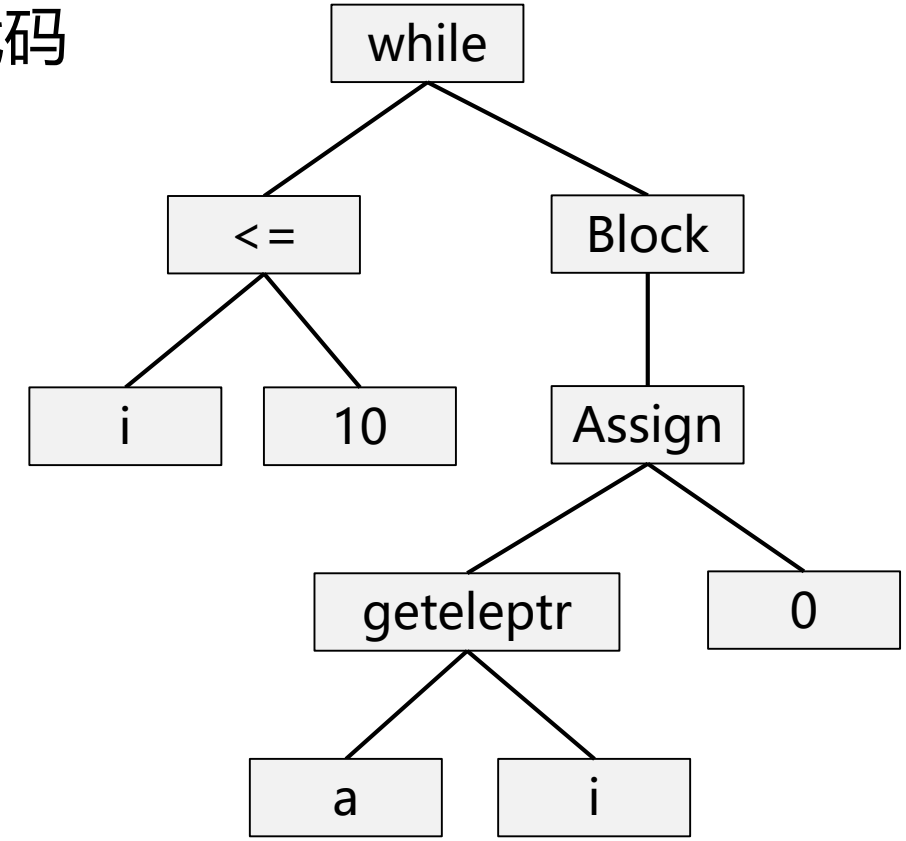


(c) (1 * z) * z



题二

- 把右侧C程序的可执行语句翻译成：
  - 语法树
  - 后缀表示
  - 三地址代码




```
main () {
    int i;
    int a[10];
    while (i < = 10 )
        a[i] = 0;
}
```

假设产生式为：

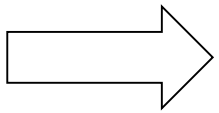
Stmt → **while** Cond Block

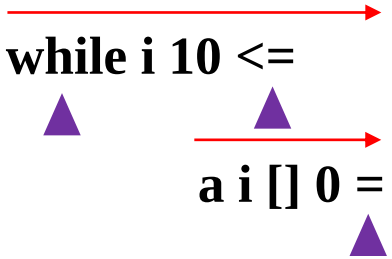
Block → Stmt+

题二

- 把右侧C程序的可执行语句翻译成：
  - 语法树
  - 后缀表示
  - 三地址代码

```
while (i < = 10 )
    a[i] = 0;
```






```
main () {
    int i;
    int a[10];
    while (i < = 10 )
        a[i] = 0;
}
```

假设产生式为：
Stmt → **while** Cond Block
Block → Stmt+

1. 自左向右扫描，遇到while。接下来处理Cond语句

2. Cond: 此时，栈中有符号i和10。遇到relop，退栈两个符号。遇到换行符，压入Cond，完成。

3. Stmt: 此时，栈中有符号a[i]和0，遇到assign，退栈两个符号，压入一个stmt。完成。结束。

思考：如果不用换行符，栈还能描述while控制语句的计算吗？

题二

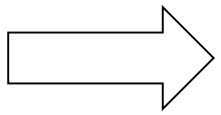
- 把右侧C程序的可执行语句翻译成：
  - 语法树
  - 后缀表示
  - 三地址代码

重点：while语句转换成

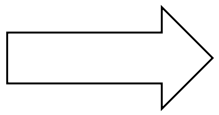
L: if Cond goto L1
   goto L2
L1: Block goto L
L2: -

```
main () {
   int i;
   int a[10];
   while (i < = 10 )
      a[i] = 0;
}
```

```
while (i < = 10 )
    a[i] = 0;
```



```
L: if (i < = 10 )
{
   a[i] = 0;
   goto L
}
```



```
(1)  if i < = 10 goto (3)
(2)  goto (5)
(3)  a[i] = 0;
(4)  goto (1)
(5)  -
```

第三次习题课-H15-2

语法分析5&语法制导的翻译1

刘硕
zkdliushuo@mail.ustc.edu.cn

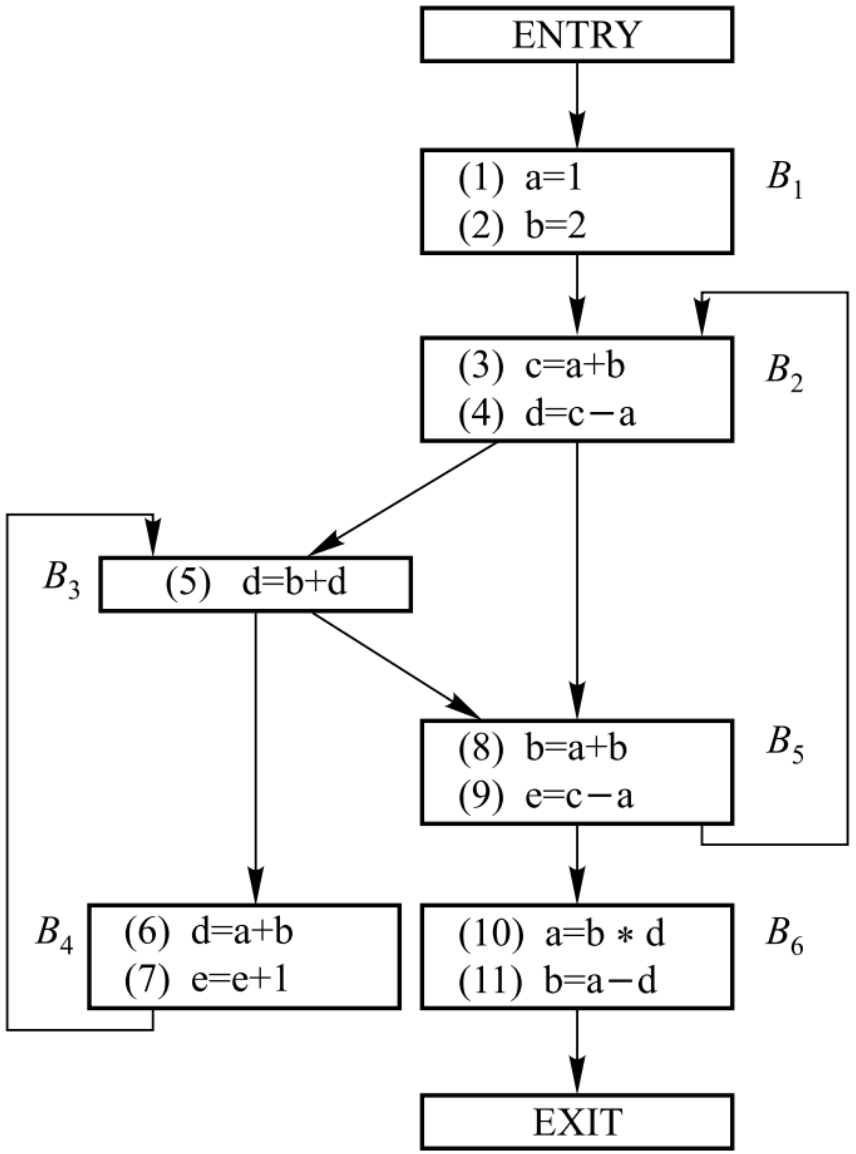
题一

- 对右侧流图，计算：
  - 为到达-定值分析，计算每个块的gen、kill、IN和OUT集合
  - gen_B：B中生成的且能到达B的结束点的定值
  - $kill_B$：B注销的定值
  - 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）

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

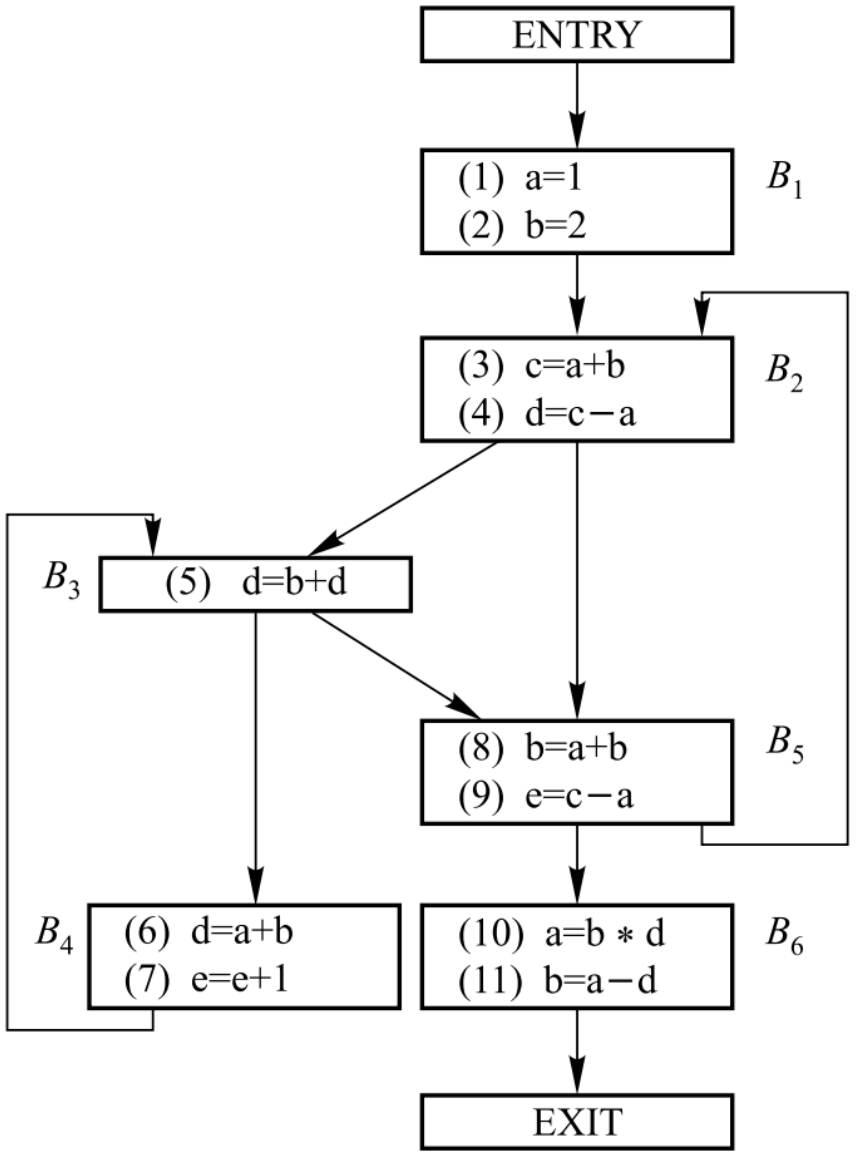
到达-定值方程组的迭代求解，最终到达不动点(MFP)



题一

- 对右侧流图，计算：
  - 为到达-定值分析，计算每个块的gen、kill、IN和OUT集合

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

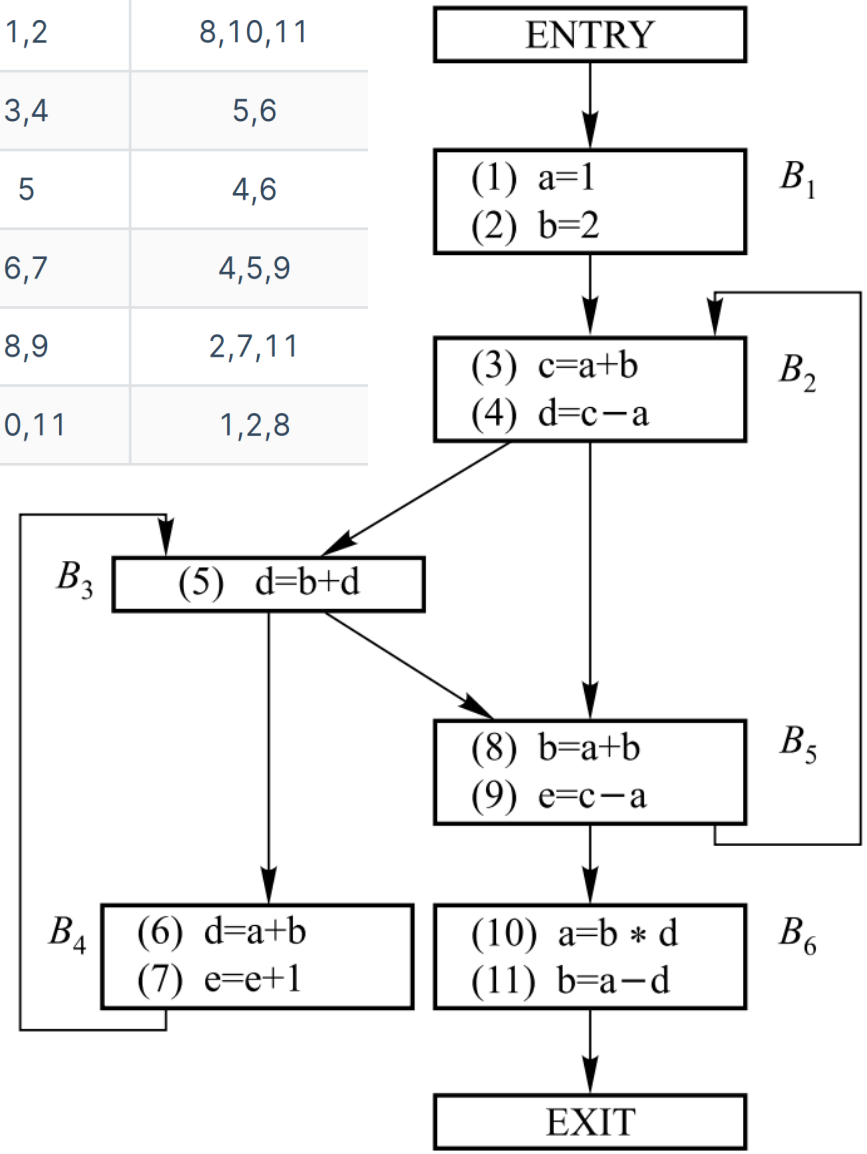


题一

- 到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	00000000000
B2		00000000000
B3		00000000000
B4		00000000000
B5		00000000000
B6		00000000000

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

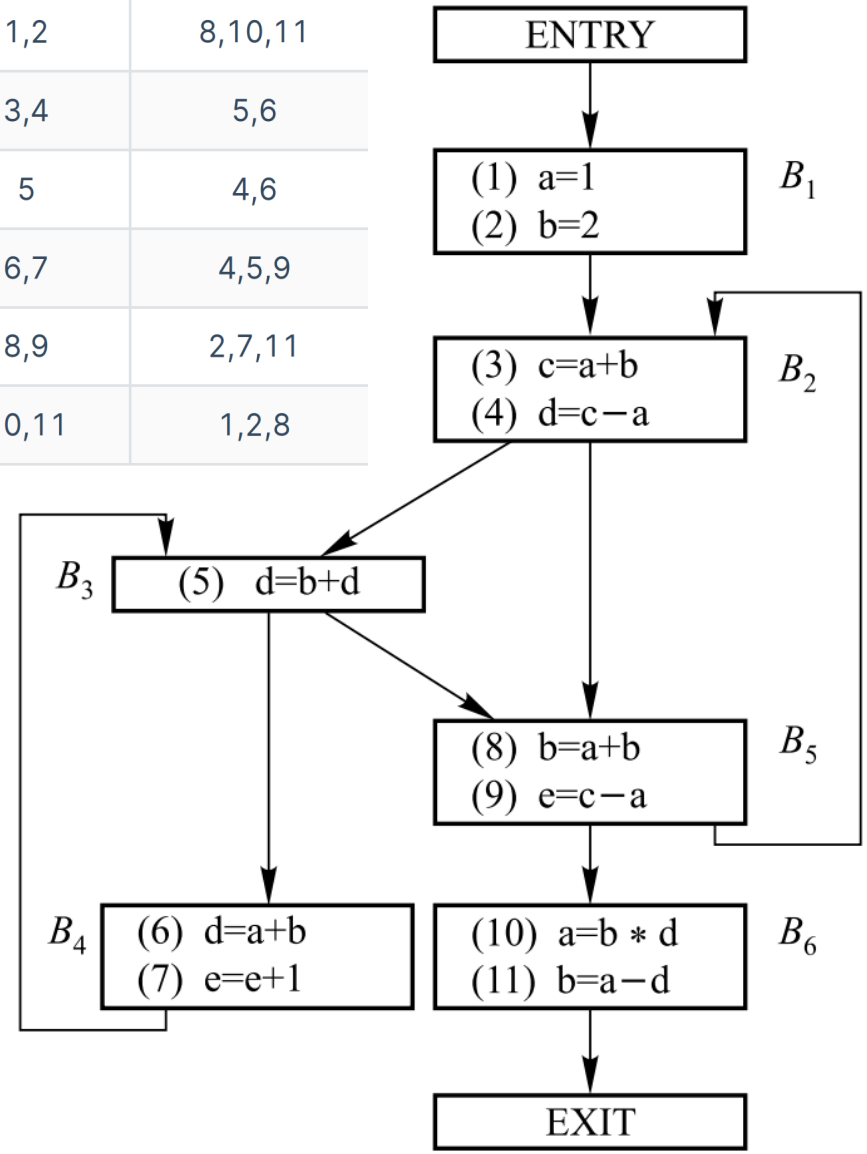


题一

到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11000000000	00000000000
B3		00000000000
B4		00000000000
B5		00000000000
B6		00000000000

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

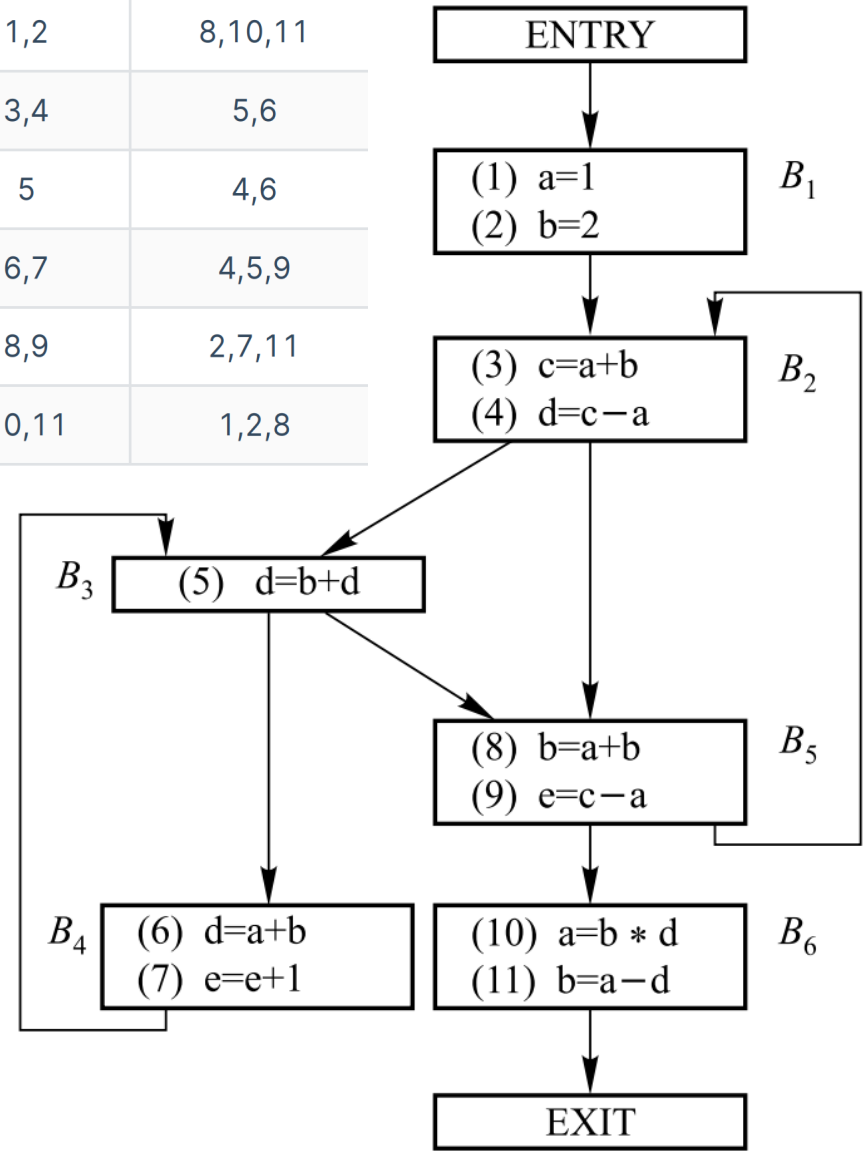


题一

到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11000000000	11110000000
B3	11110000000	00000000000
B4		00000000000
B5		00000000000
B6		00000000000

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

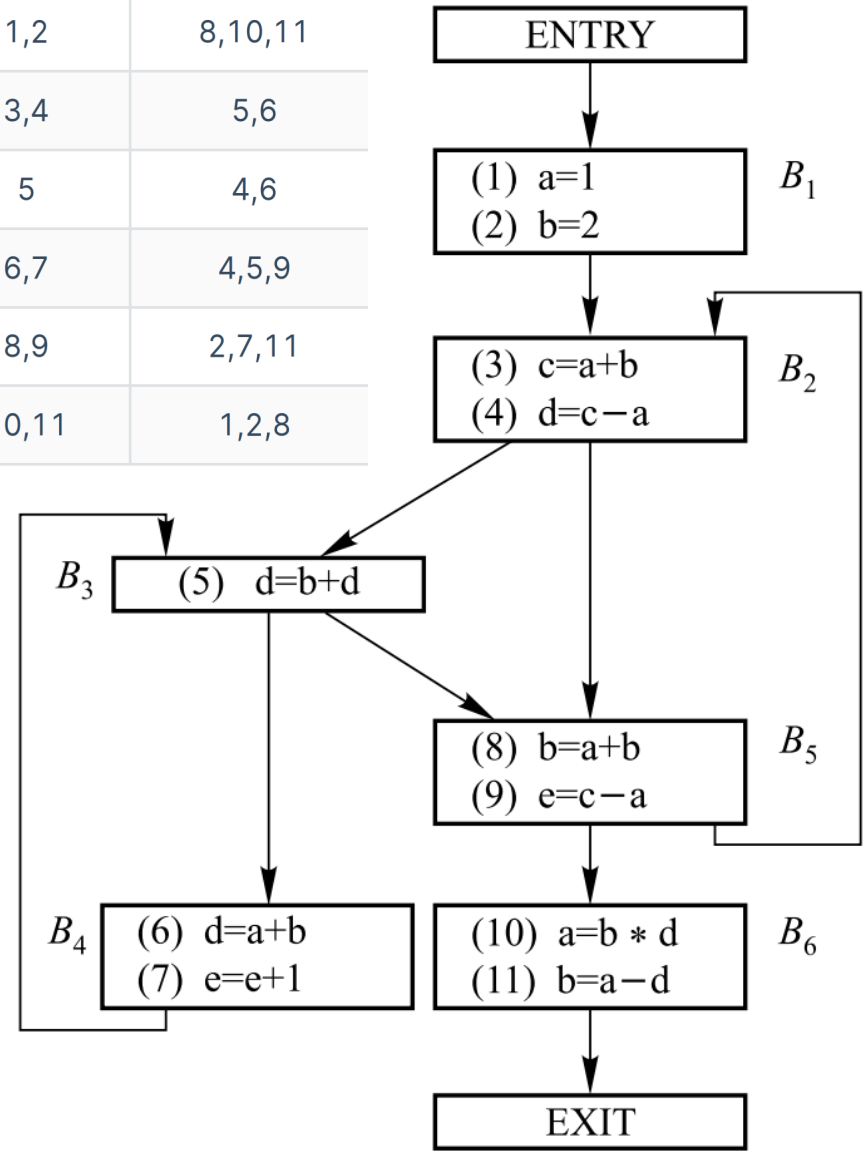


题一

到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11000000000	11110000000
B3	11110000000	11101000000
B4	11101000000	00000000000
B5		00000000000
B6		00000000000

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

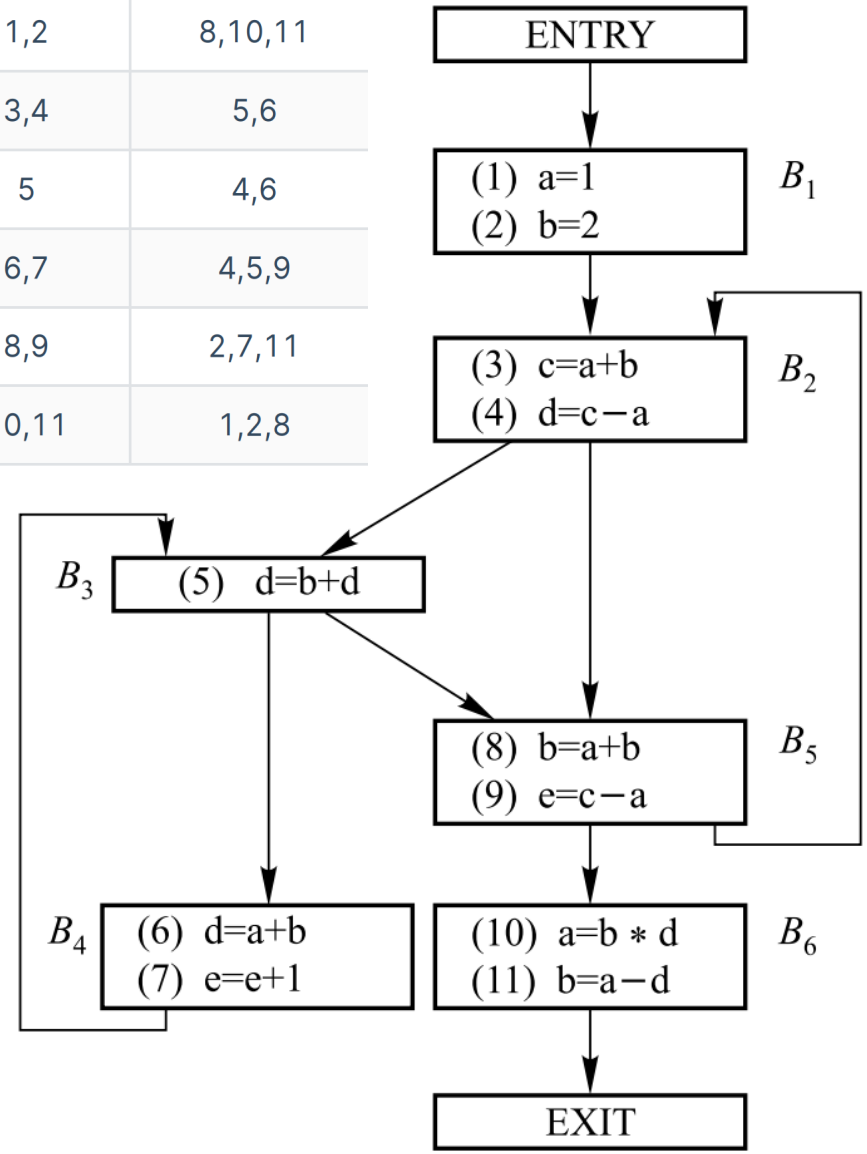


题一

到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11000000000	11110000000
B3	11110000000	11101000000
B4	11101000000	11100110000
B5	11111000000	00000000000
B6		00000000000

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

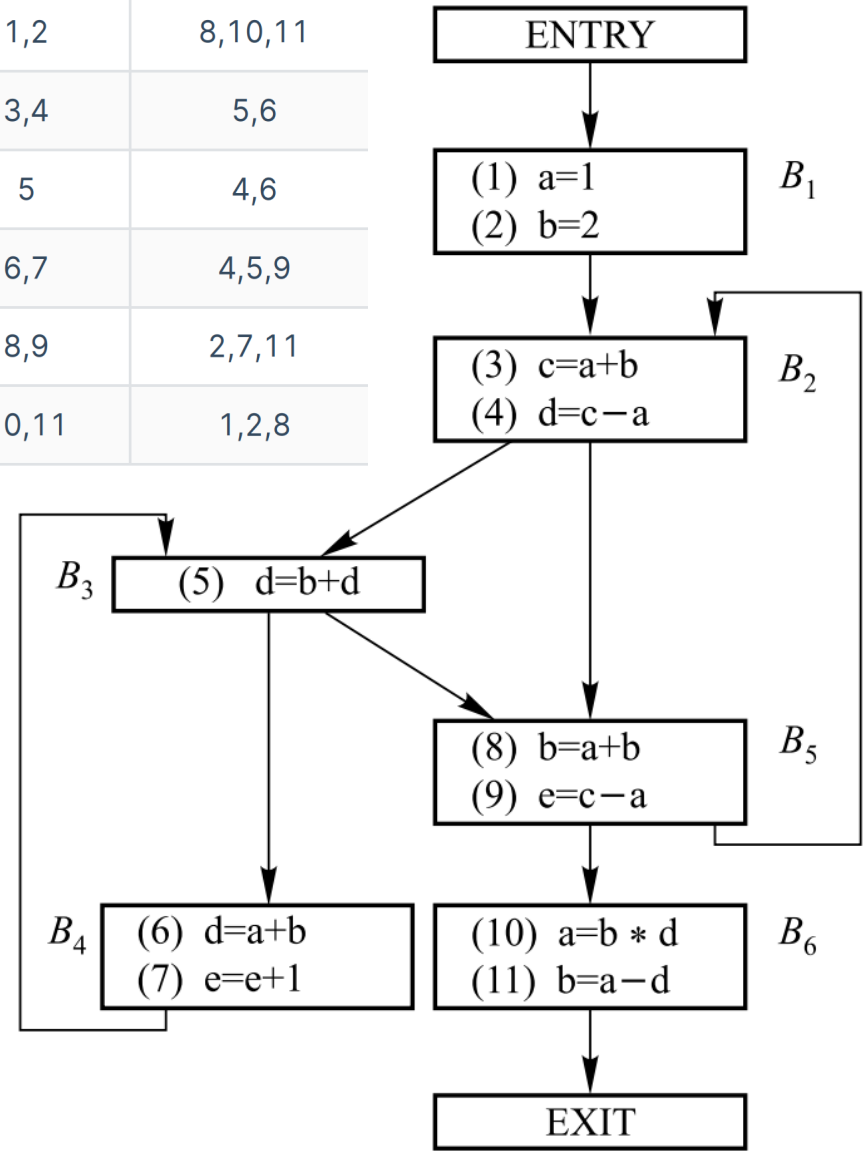


题一

到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11000000000	11110000000
B3	11110000000	11101000000
B4	11101000000	11100110000
B5	11111000000	10111001100
B6	10111001100	00000000000

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

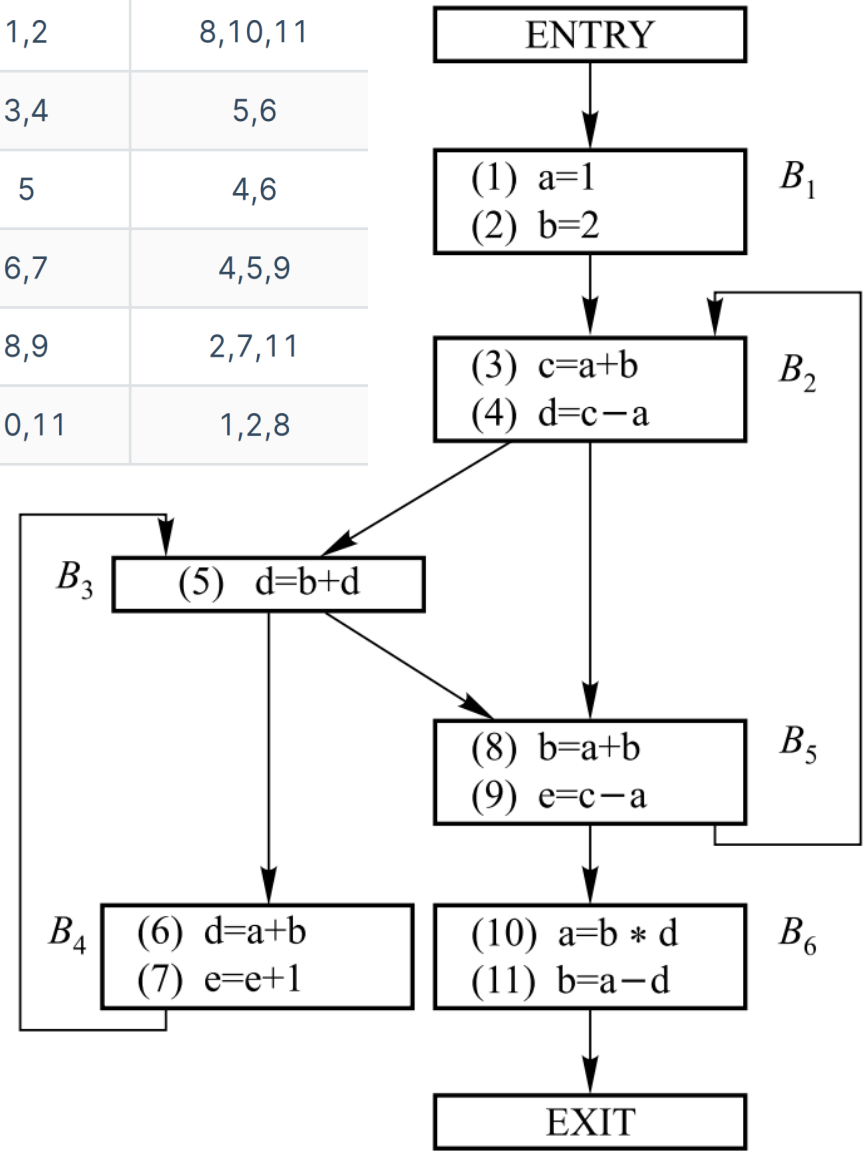


题一

到达-定值分析

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11000000000	11110000000
B3	11110000000	11101000000
B4	11101000000	11100110000
B5	11111000000	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



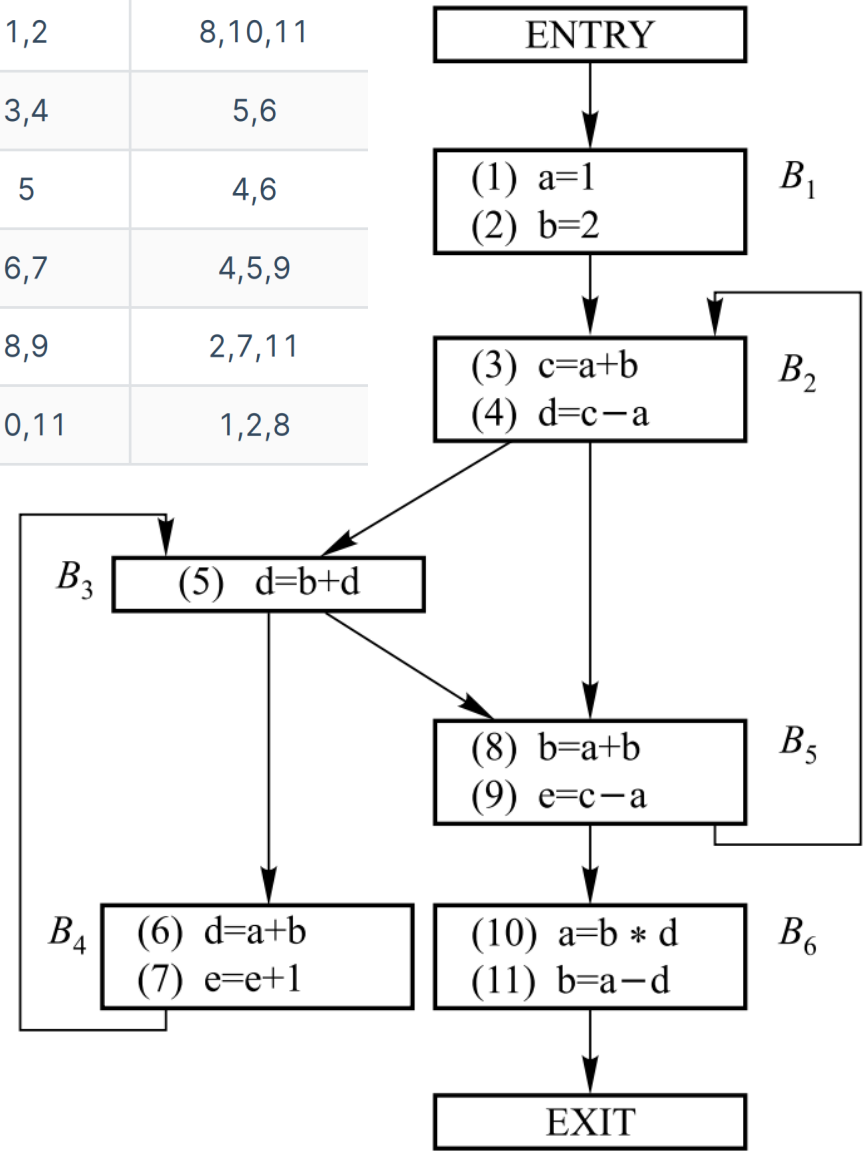
题一

- 到达-定值分析

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8

第二轮迭代

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110000000
B3	11110000000	11101000000
B4	11101000000	11100110000
B5	11111000000	10111001100
B6	10111001100	00111000111



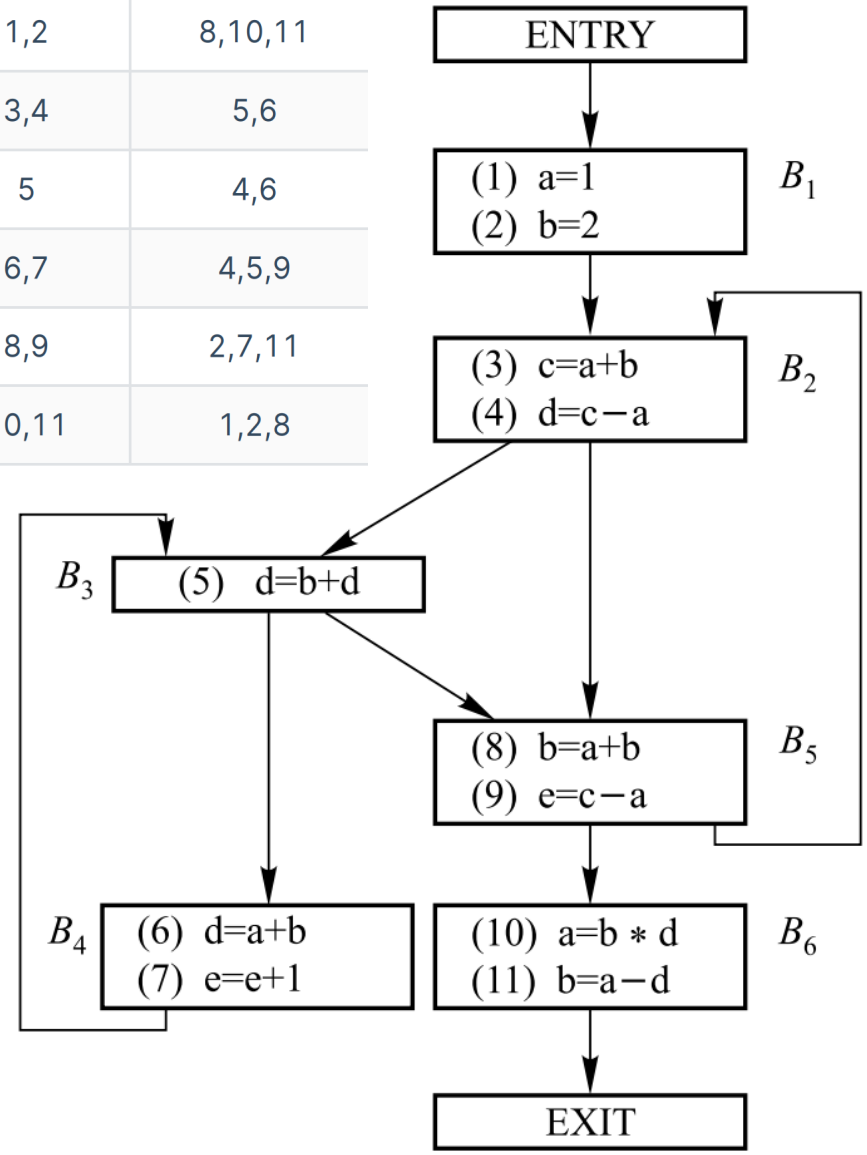
题一

- 到达-定值分析

第二轮迭代

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110001100
B3	11110111100	11101000000
B4	11101000000	11100110000
B5	11111000000	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



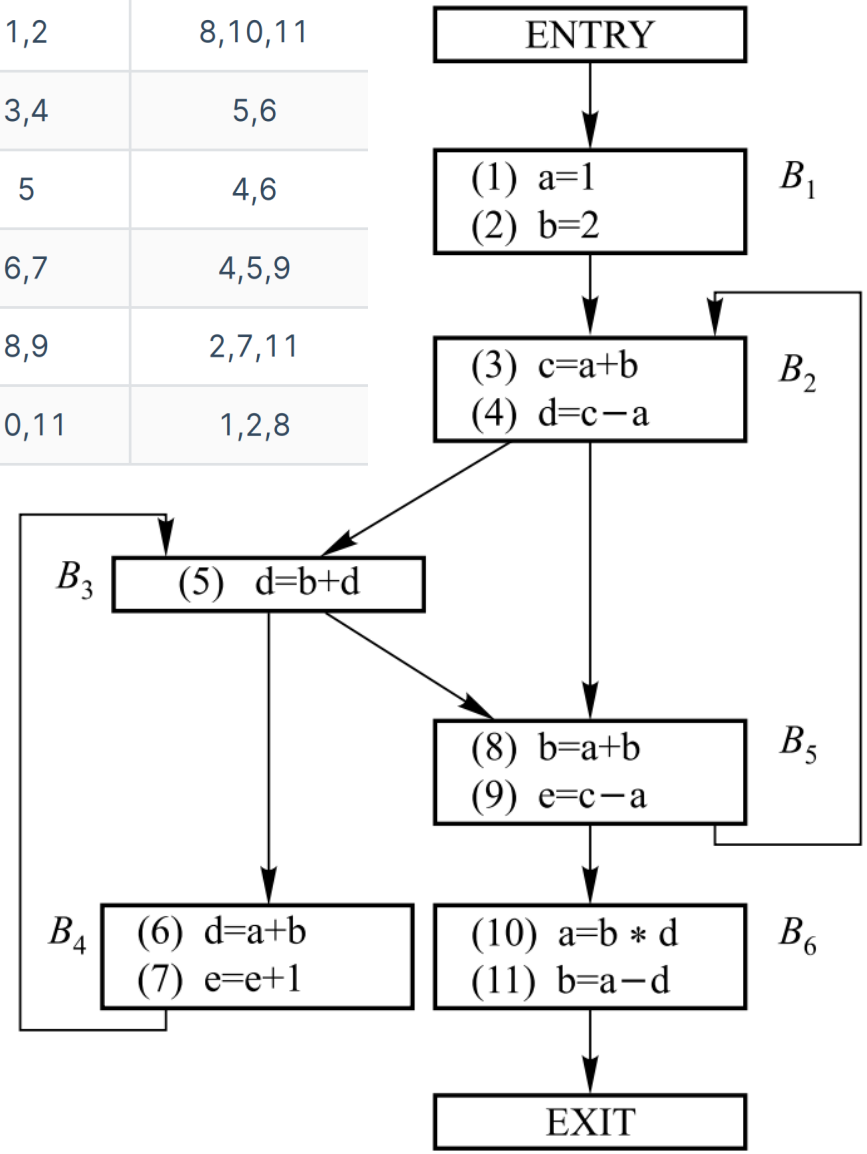
题一

到达-定值分析

第二轮迭代

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110001100
B3	11110111100	11101011100
B4	11101011100	11100110000
B5	11111000000	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



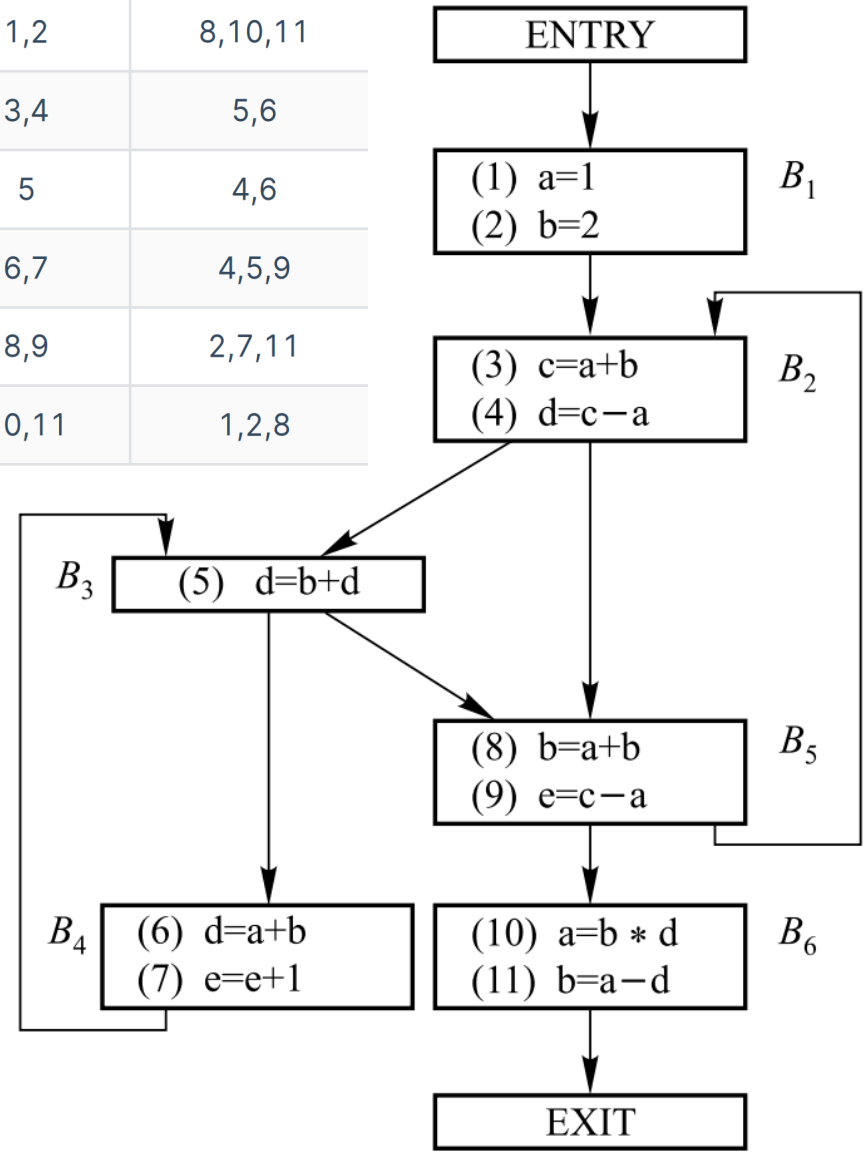
题一

- 到达-定值分析

第二轮迭代

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110001100
B3	11110111100	11101011100
B4	11101011100	11100111000
B5	11111011100	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



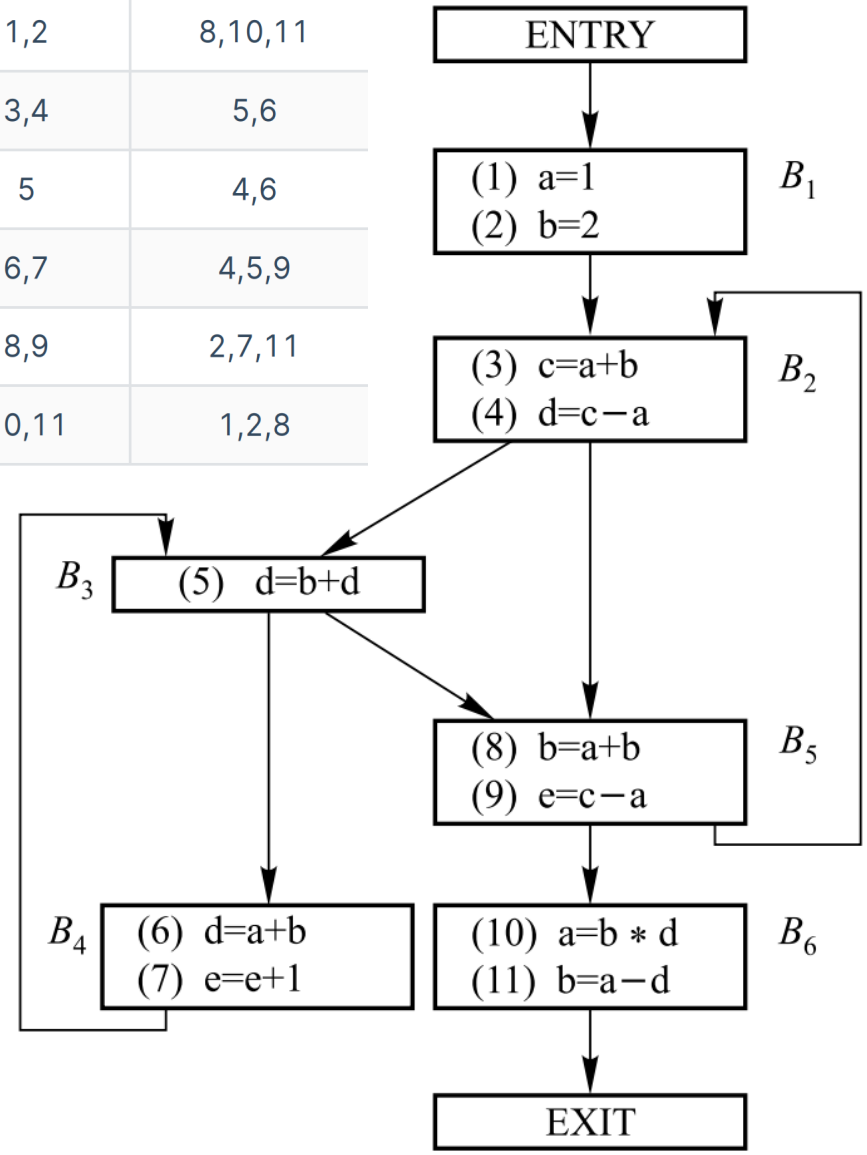
题一

- 到达-定值分析

第二轮迭代

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110001100
B3	11110111100	11101011100
B4	11101011100	11100111000
B5	11111011100	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



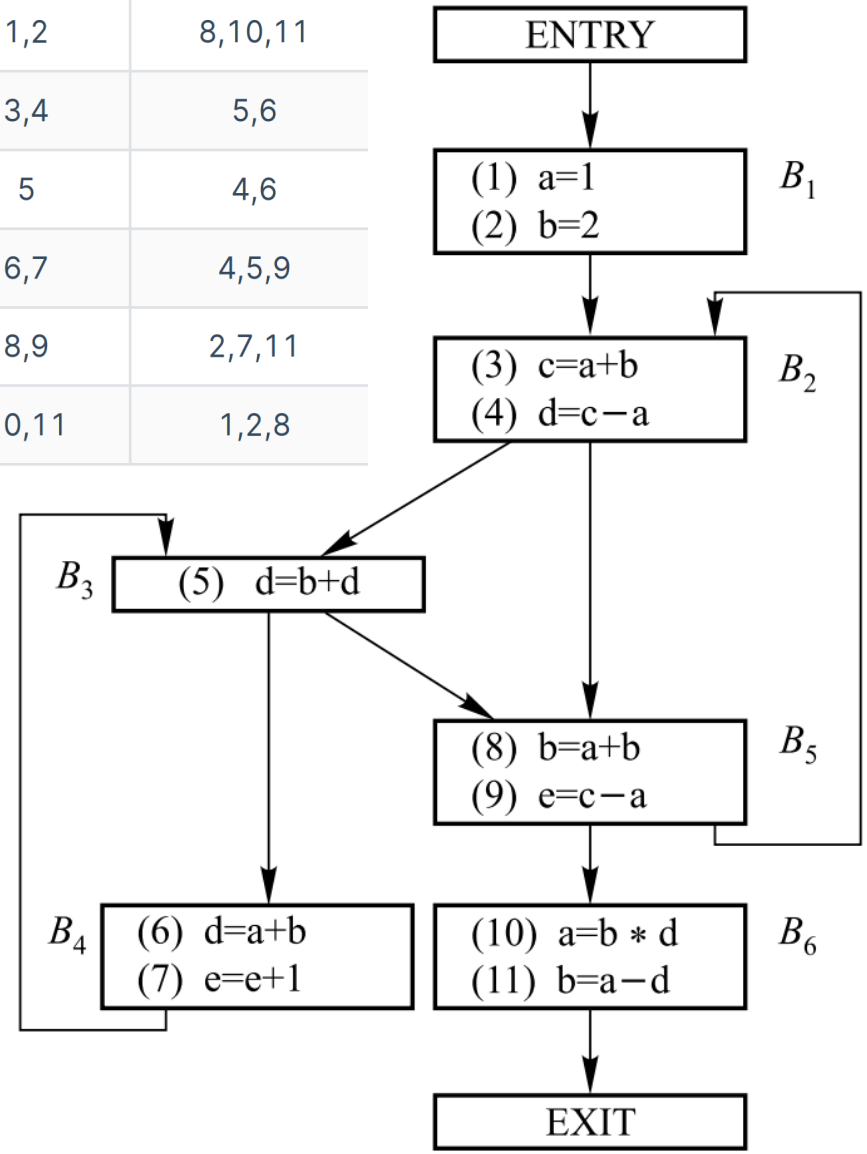
题一

- 到达-定值分析

第二轮迭代

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110001100
B3	11110111100	11101011100
B4	11101011100	11100111000
B5	11111011100	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



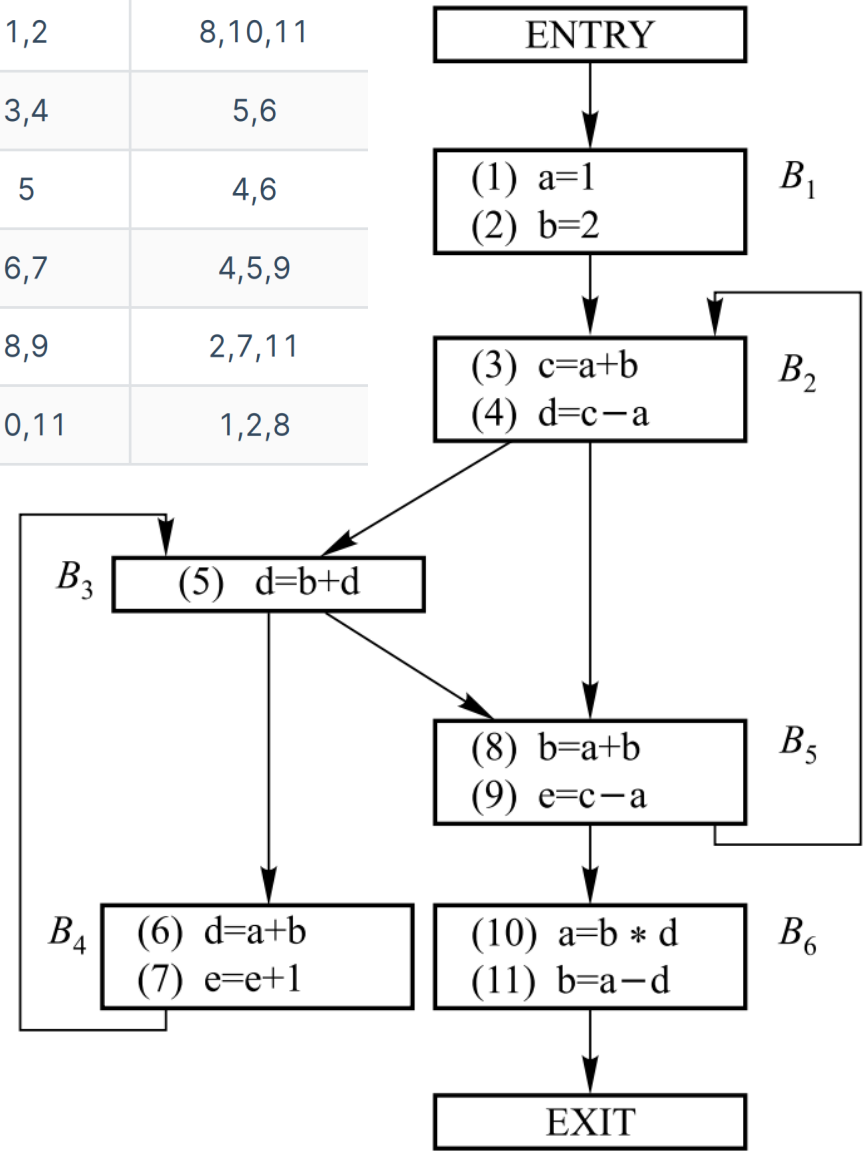
题一

- 到达-定值分析

第三轮迭代 都不变

Block	IN[B]	OUT[B]
B1	00000000000	11000000000
B2	11111001100	11110001100
B3	11110111100	11101011100
B4	11101011100	11100111000
B5	11111011100	10111001100
B6	10111001100	00111000111

	GEN	KILL
B1	1,2	8,10,11
B2	3,4	5,6
B3	5	4,6
B4	6,7	4,5,9
B5	8,9	2,7,11
B6	10,11	1,2,8



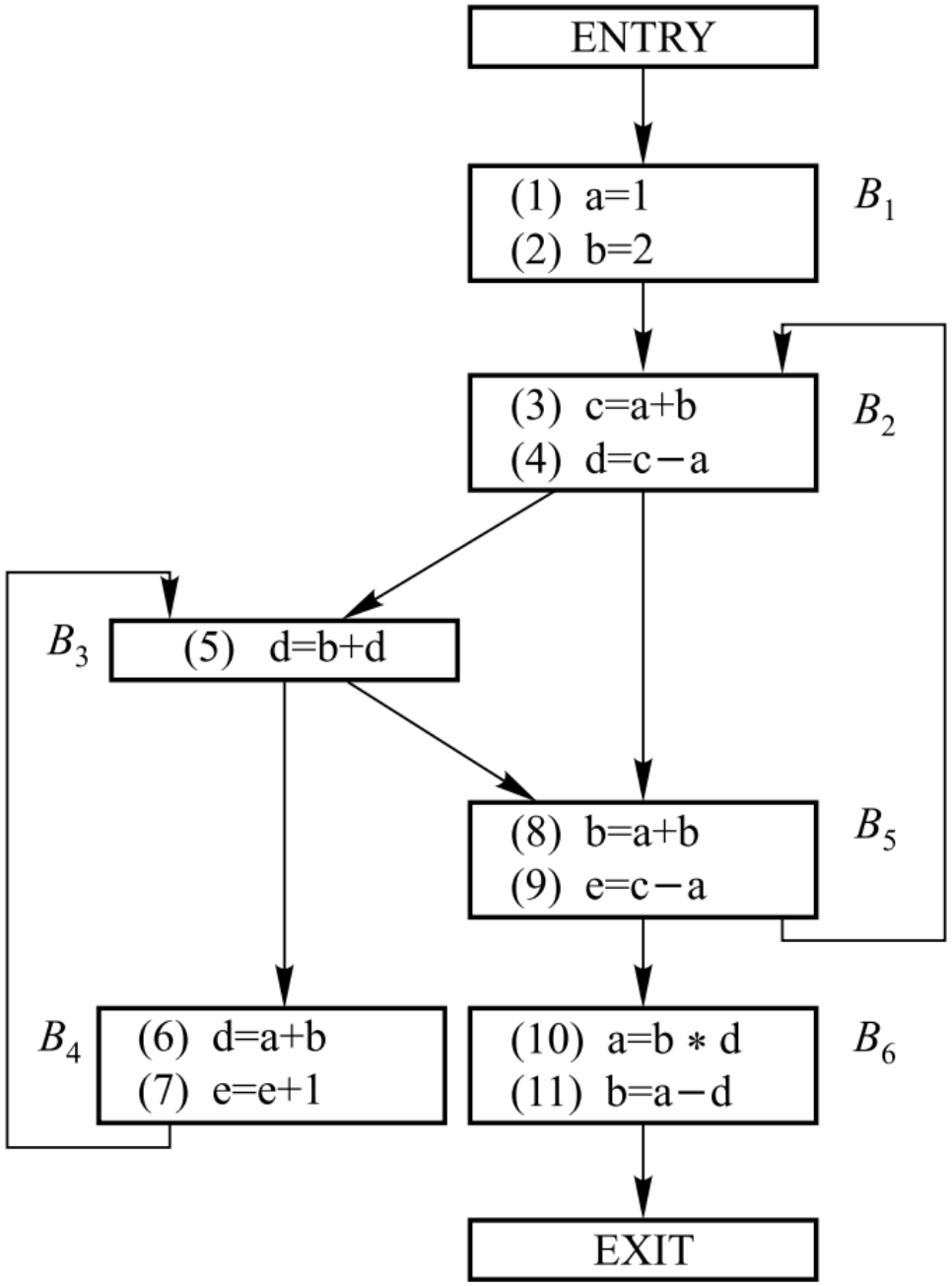
题一

- 对右侧流图，计算：
  - 可用表达式分析

所有表达式集合U:

$\{1,2,\mathrm{a}+\mathrm{b},\mathrm{c}-a,b*d,a-d,e+1,b+d\}$

Block	e_gen	e_kill
B1	{1, 2}	{a+b, c-a, b*d, a-d, b+d}
B2	{a+b, c-a}	{c-a, b*d, a-d, b+d}
B3	∅	{b*d, a-d, b+d}
B4	{a+b}	{b*d, a-d, b+d, e+1}
B5	{c-a}	{2, a+b, b*d, e+1, b+d}
B6	{a-d}	{1, 2, a+b, a-d, b*d, c-a, b+d}



题一

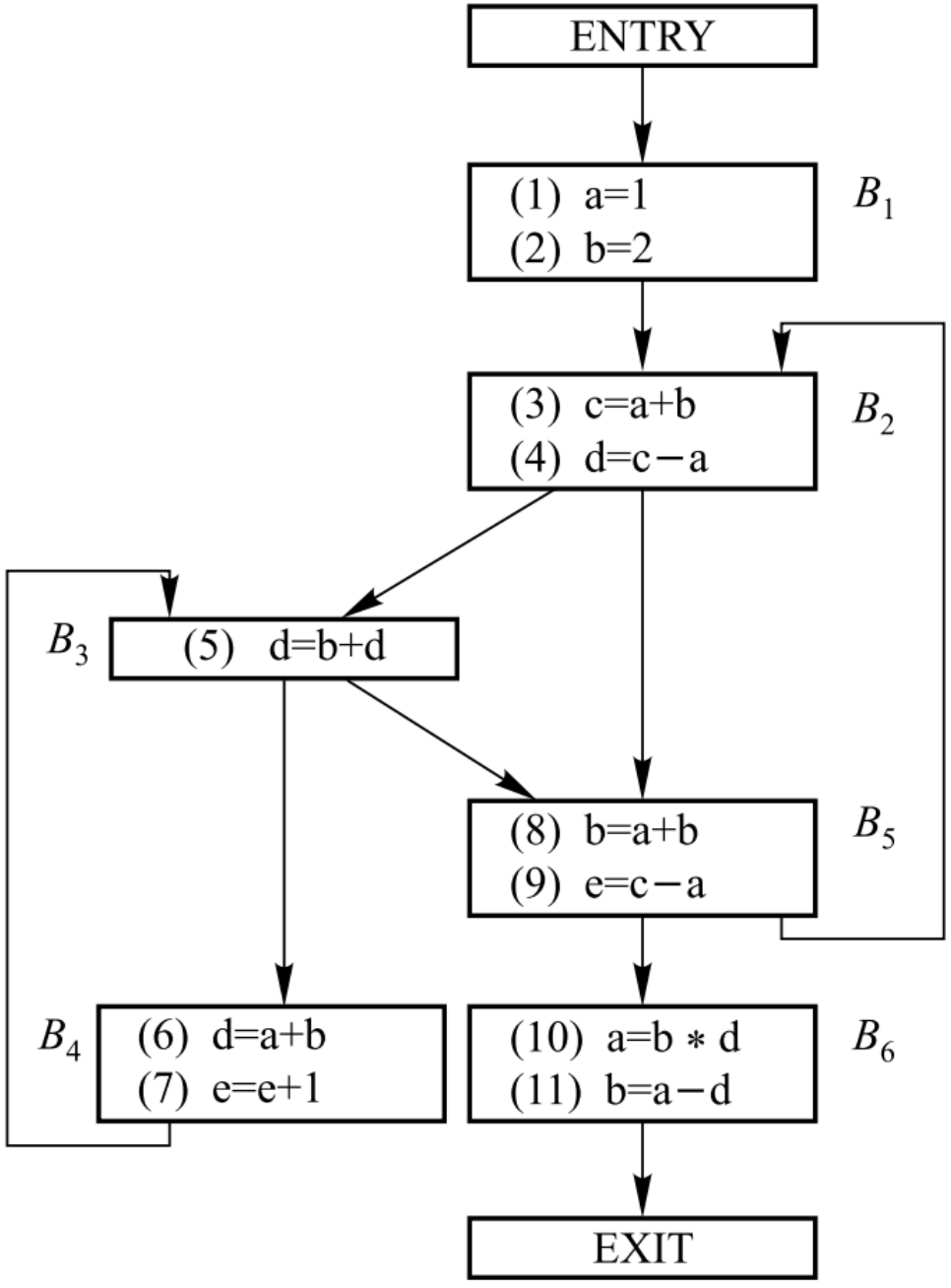
- 对右侧流图，计算：
  - 可用表达式分析

所有表达式集合U:

$\{1,2,\mathrm{a}+\mathrm{b},\mathrm{c}-a,b*d,a-d,e+1,b+d\}$

第一轮迭代

Block	IN(B)	OUT(B)
B1	∅	{1, 2}
B2	{1, 2}	{1, 2, a+b, c-a}
B3	{1, 2, a+b, c-a}	{1, 2, a+b, c-a}
B4	{1, 2, a+b, c-a}	{1, 2, a+b, c-a}
B5	{1, 2, a+b, c-a}	{1, c-a}
B6	{1, c-a}	{a-d}



题一

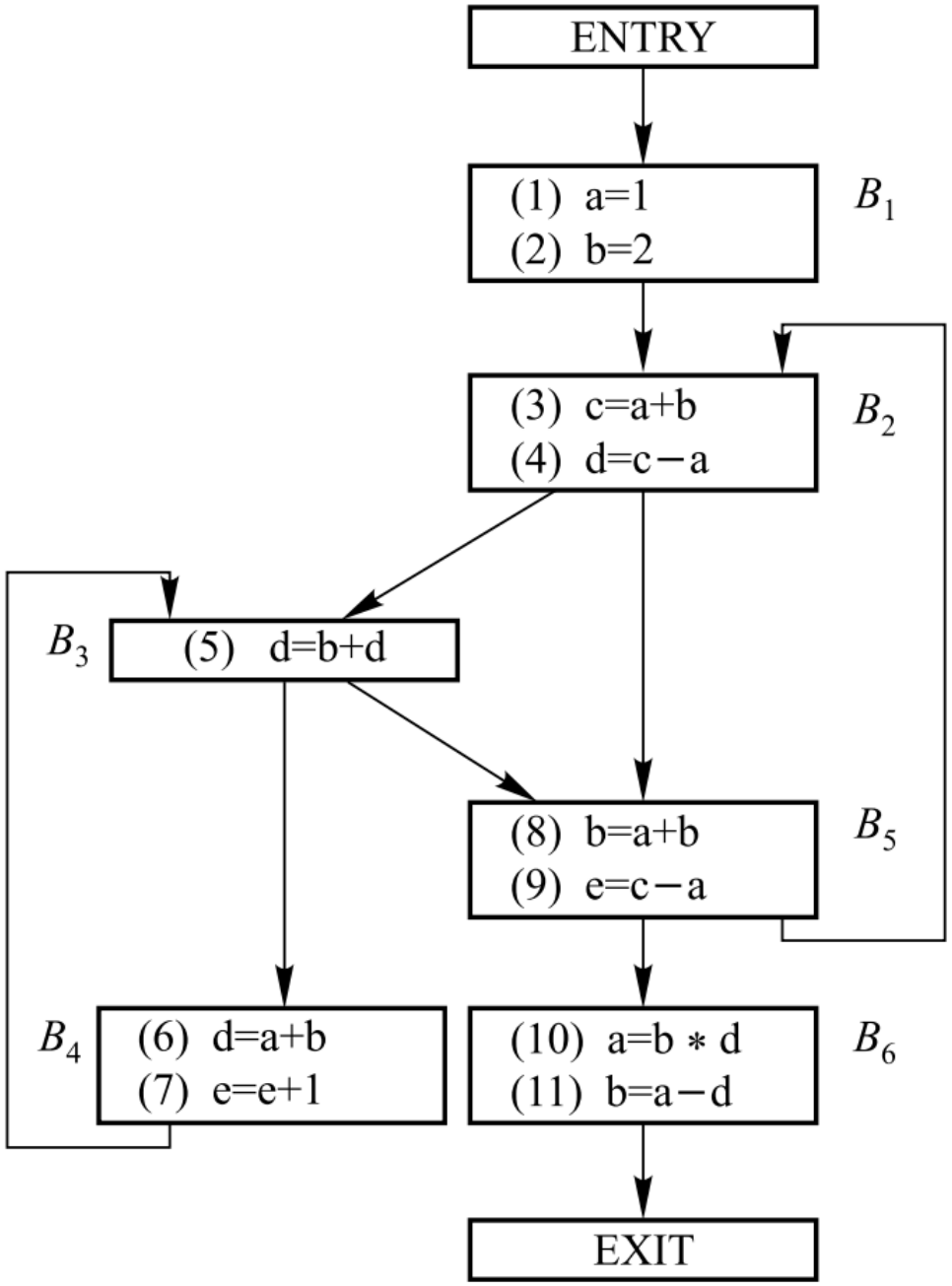
- 对右侧流图，计算：
  - 可用表达式分析

所有表达式集合U:

$\{1,2,\mathrm{a}+\mathrm{b},\mathrm{c}-a,b*d,a-d,e+1,b+d\}$

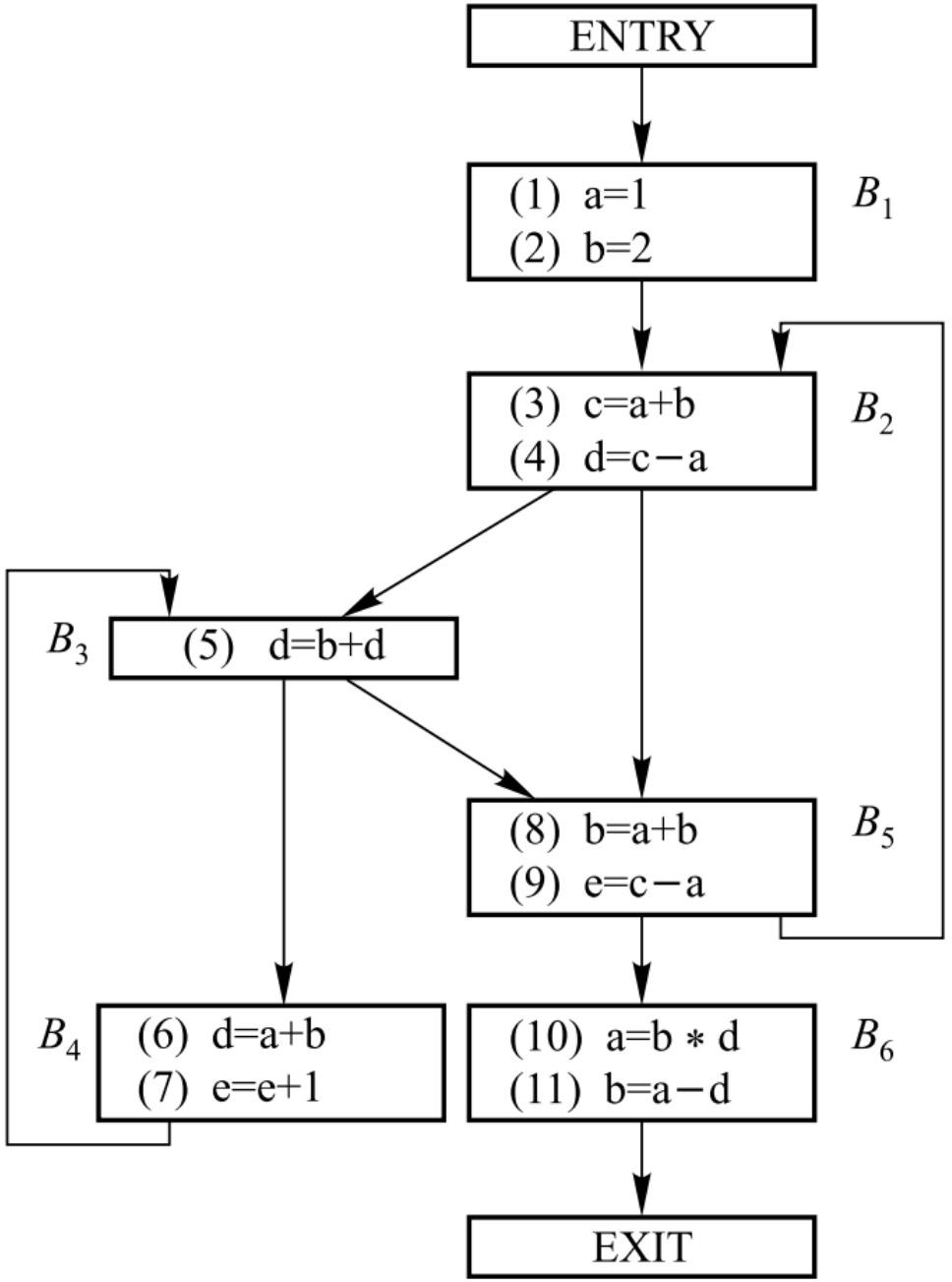
第二轮迭代

Block	IN(B)	OUT(B)
B1	∅	{1, 2}
B2	{1}	{1, a+b, c-a}
B3	{1, a+b, c-a}	{1, a+b, c-a}
B4	{1, a+b, c-a}	{1, a+b, c-a}
B5	{1, a+b, c-a}	{1, c-a}
B6	{1, c-a}	{a-d}



题一

- 对右侧流图，计算：
  - 可用表达式分析



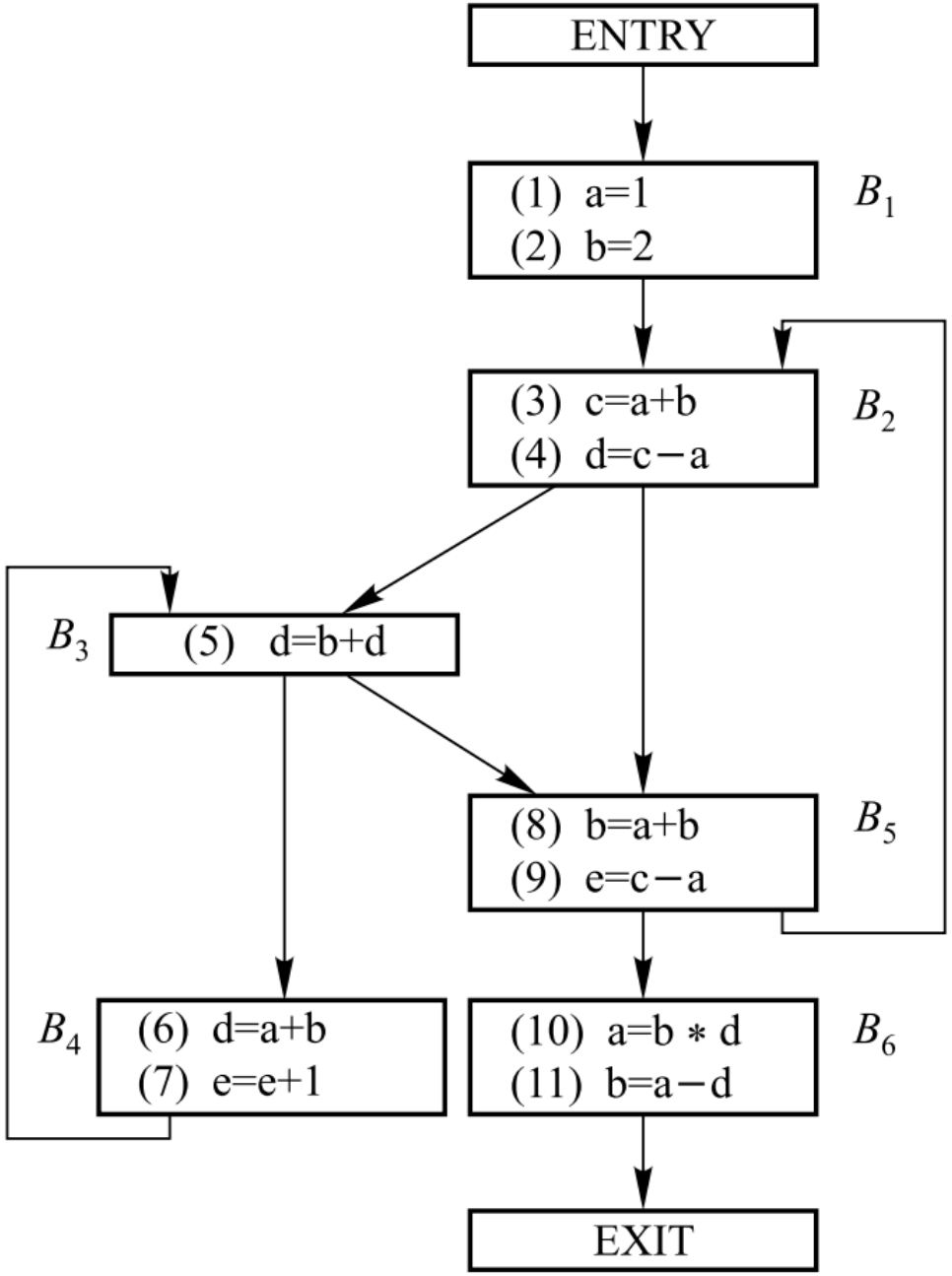
与到达-定值分析的区别：

1. 汇合符号为求交集，含义是在任何一个前驱路径上表达式都可用
2. 数据流值的定义域为表达式集合

题一

- 对右侧流图，计算：
  - 活跃变量分析

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}



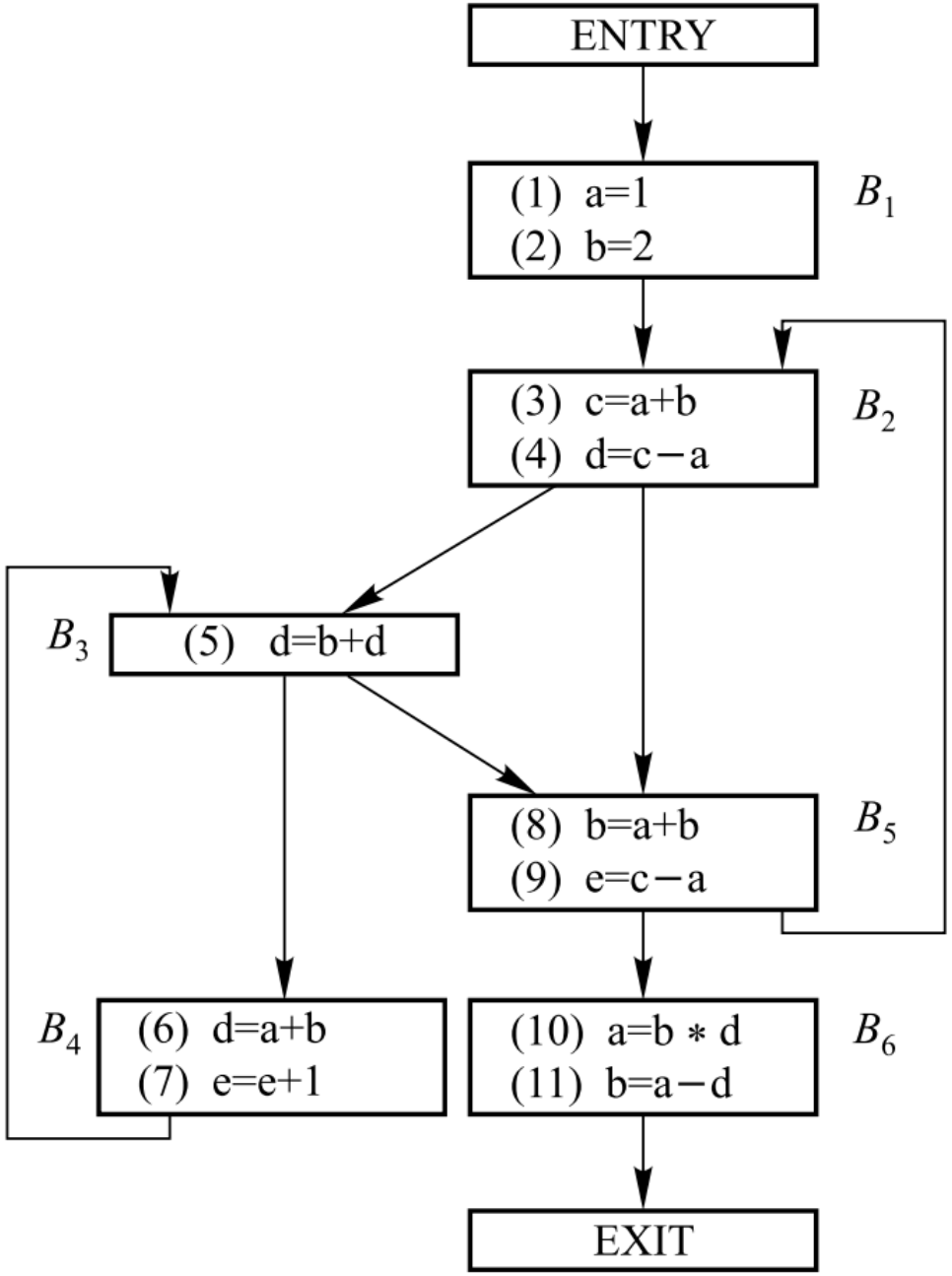
题一

- 对右侧流图，计算：
  - 活跃变量分析

第一轮迭代

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1		ϕ
B2		ϕ
B3		ϕ
B4		ϕ
B5		ϕ
B6	ϕ	{b, d}
EXIT		ϕ



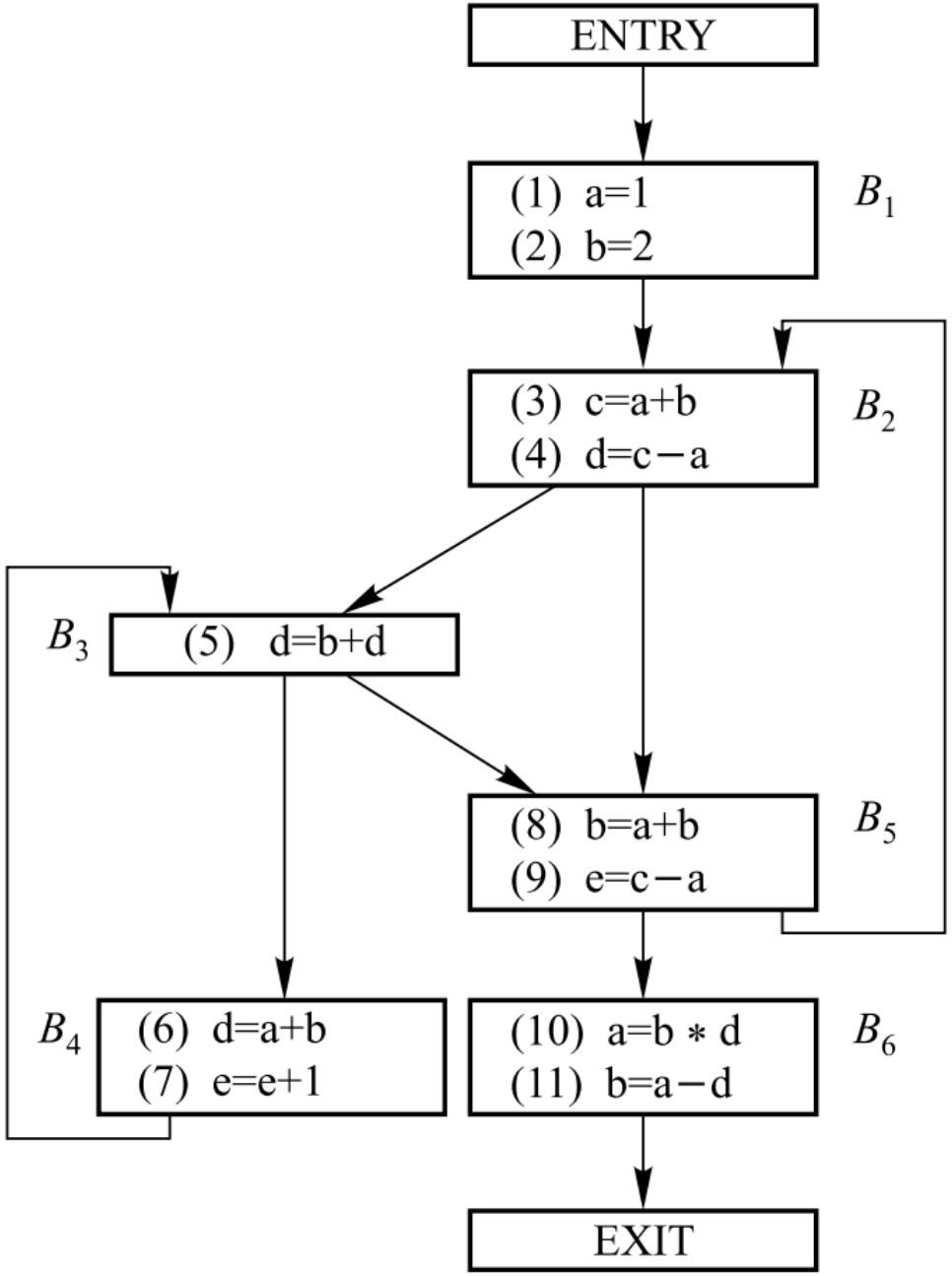
题一

- 对右侧流图，计算：
  - 活跃变量分析

第一轮迭代

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1		ϕ
B2		ϕ
B3		ϕ
B4		ϕ
B5	{b, d}	{a, b, c, d}
B6	ϕ	{b, d}
EXIT		ϕ



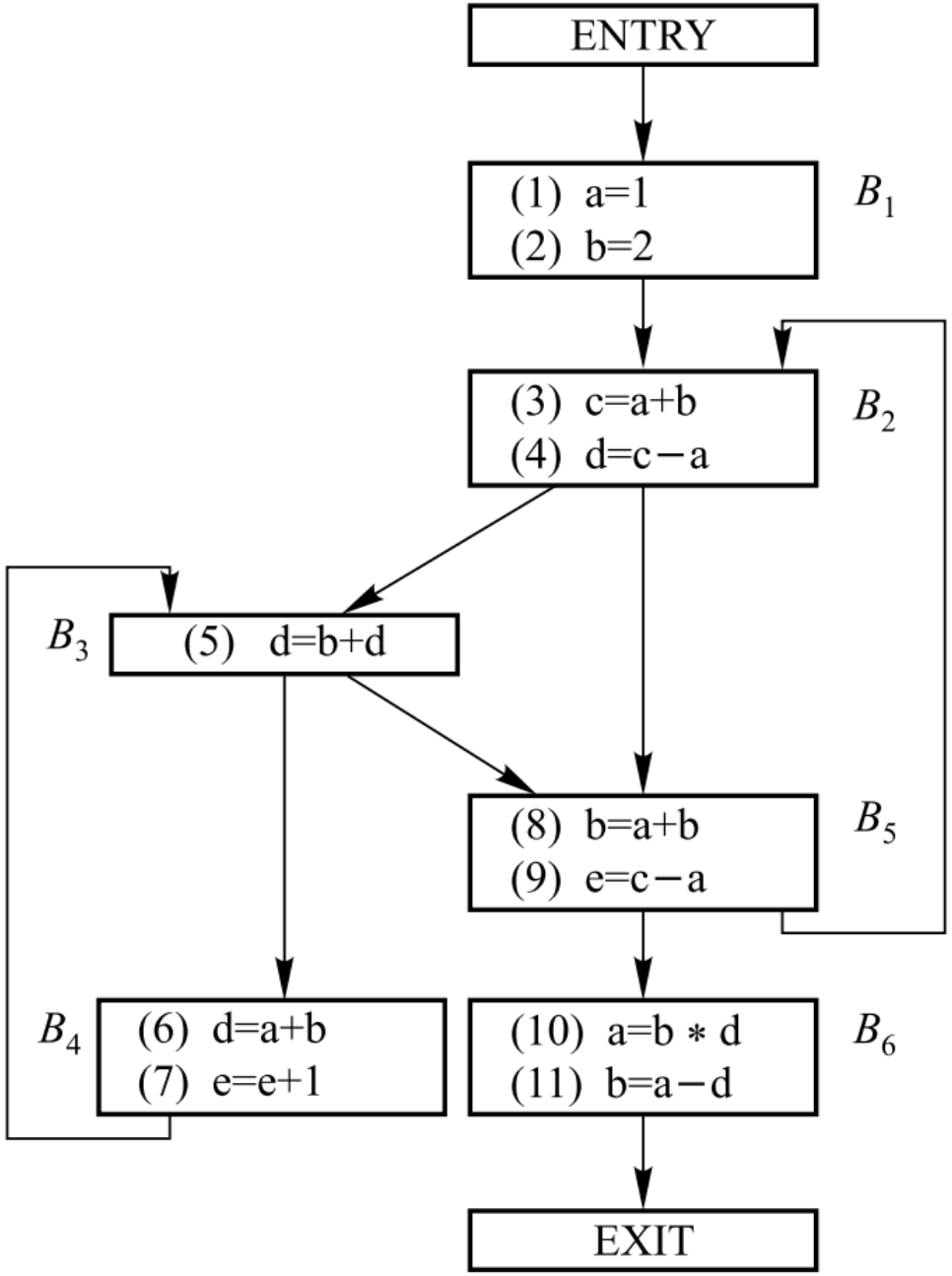
题一

- 对右侧流图，计算：
  - 活跃变量分析

第一轮迭代

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1		ϕ
B2		ϕ
B3	{a, b, c, d}	{a, b, c, d}
B4		ϕ
B5	{b, d}	{a, b, c, d}
B6	ϕ	{b, d}
EXIT		ϕ



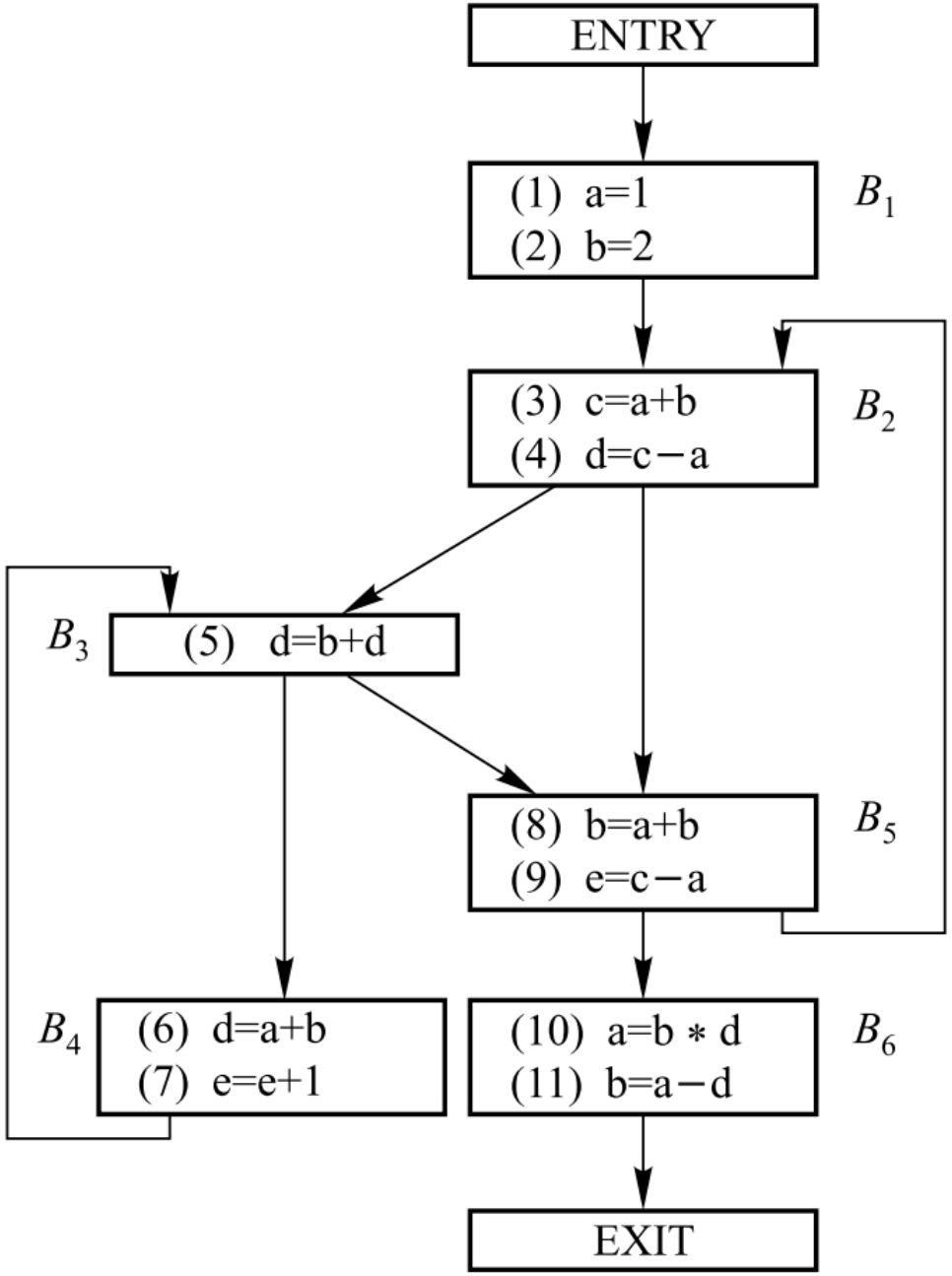
题一

- 对右侧流图，计算：
  - 活跃变量分析

第一轮迭代

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1		ϕ
B2		ϕ
B3	{a, b, c, d}	{a, b, c, d}
B4	{a, b, c, d}	{a, b, c, e}
B5	{b, d}	{a, b, c, d}
B6	ϕ	{b, d}
EXIT		ϕ



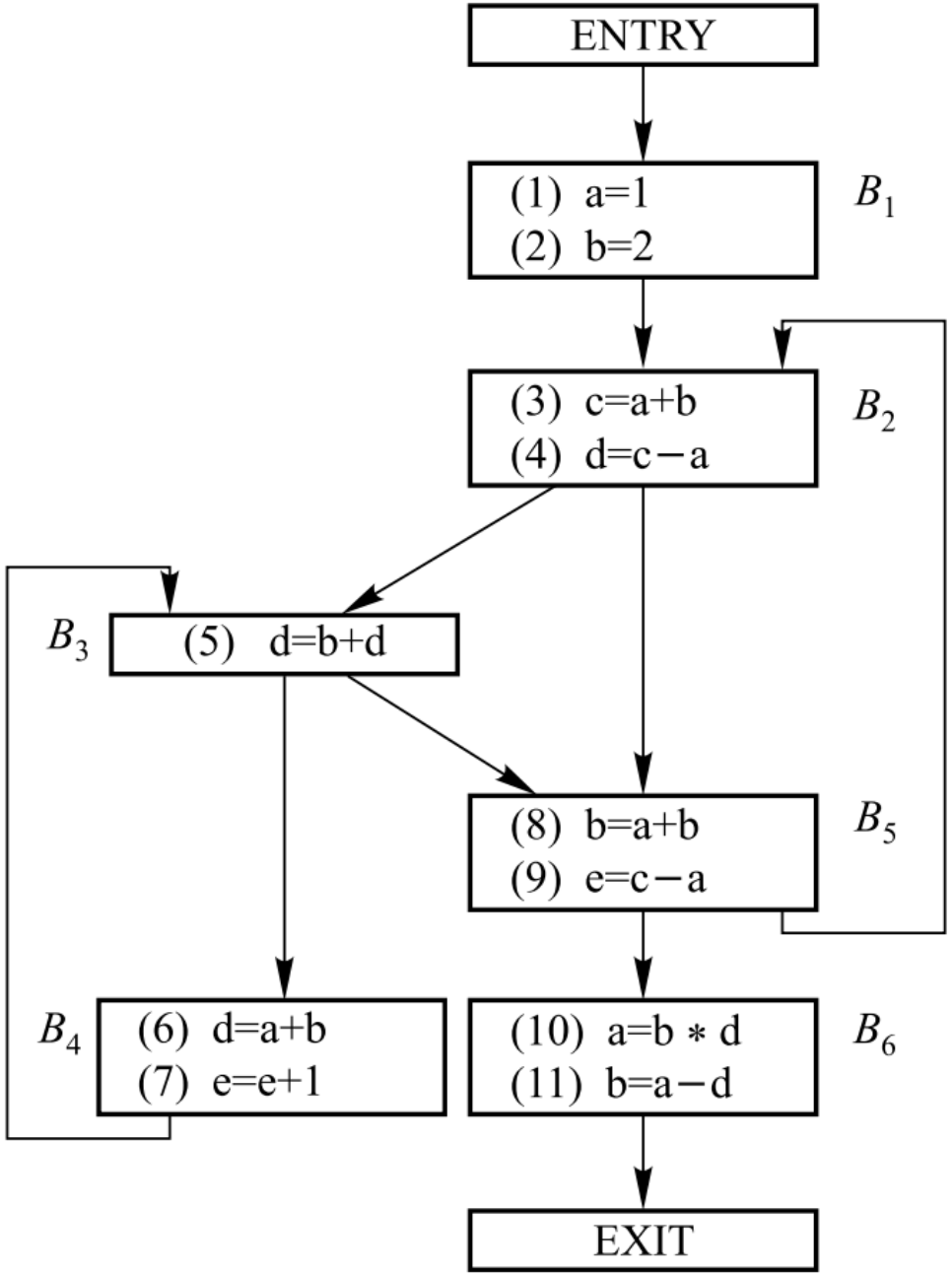
题一

- 对右侧流图，计算：
  - 活跃变量分析

第一轮迭代

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1		ϕ
B2	{a, b, c, d}	{a, b}
B3	{a, b, c, d}	{a, b, c, d}
B4	{a, b, c, d}	{a, b, c, e}
B5	{b, d}	{a, b, c, d}
B6	ϕ	{b, d}
EXIT		ϕ



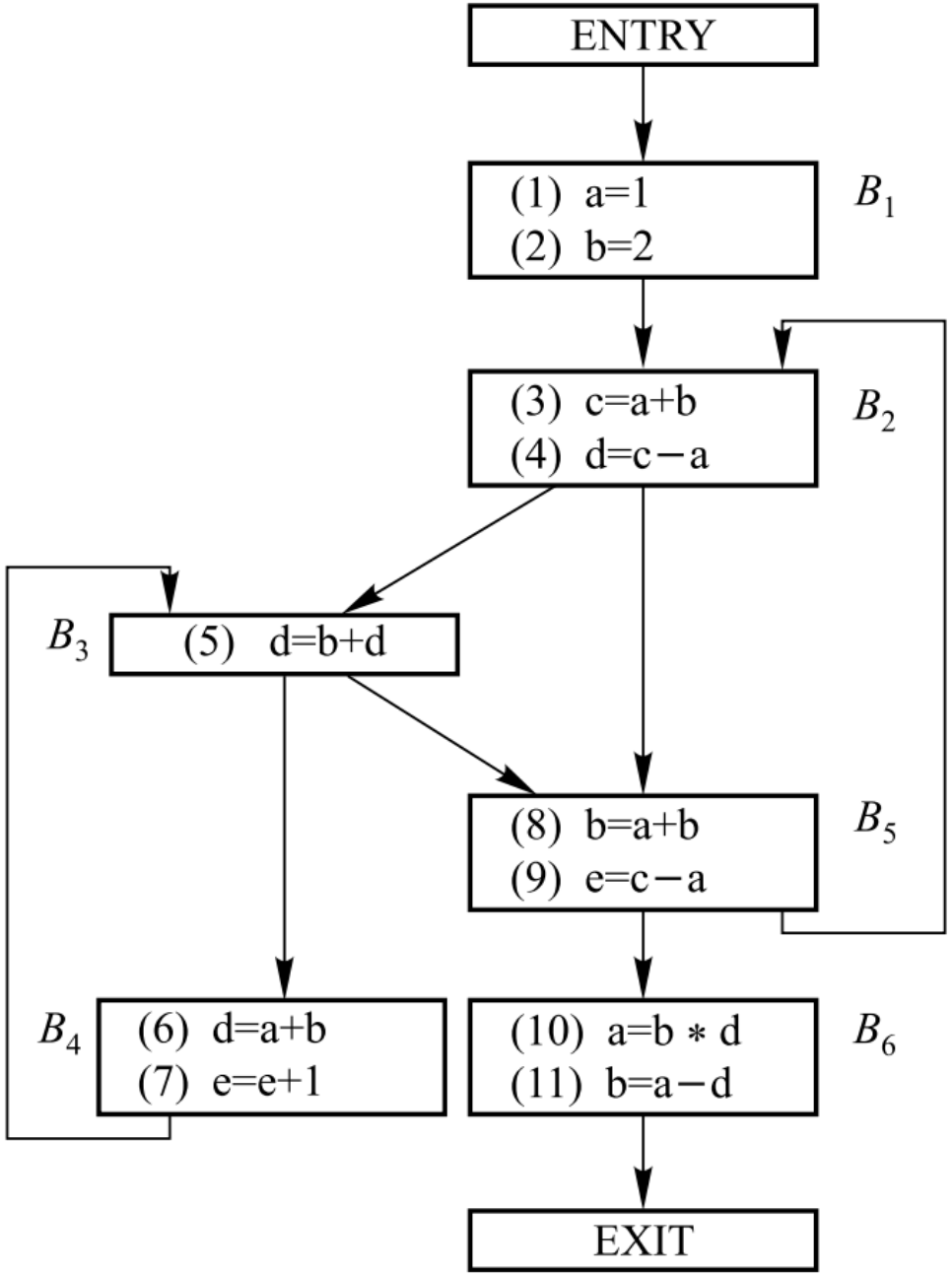
题一

- 对右侧流图，计算：
  - 活跃变量分析

第一轮迭代

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1	{a, b}	ϕ
B2	{a, b, c, d}	{a, b}
B3	{a, b, c, d}	{a, b, c, d}
B4	{a, b, c, d}	{a, b, c, e}
B5	{b, d}	{a, b, c, d}
B6	ϕ	{b, d}
EXIT		ϕ



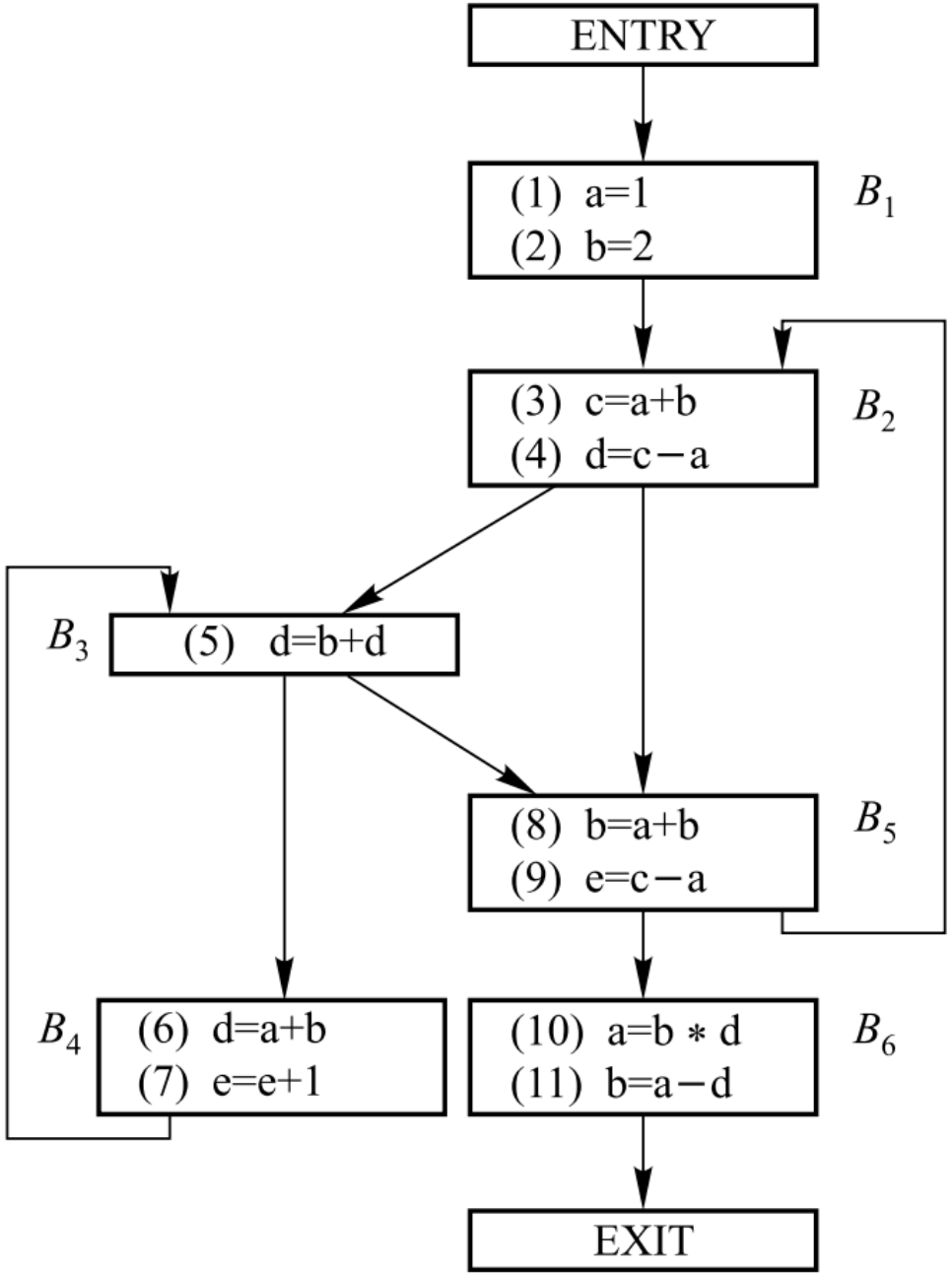
题一

- 对右侧流图，计算：
  - 活跃变量分析

最终结果

Block	use	def
B1	ϕ	{a, b}
B2	{a, b}	{c, d}
B3	{b, d}	{d}
B4	{a, b, e}	{d, e}
B5	{a, b, c}	{b, e}
B6	{b, d}	{a, b}

Block	OUT[B]	IN[B]
B1	{a, b, e}	{e}
B2	{a, b, c, d, e}	{a, b, e}
B3	{a, b, c, d, e}	{a, b, c, d, e}
B4	{a, b, c, d, e}	{a, b, c, e}
B5	{a, b, d, e}	{a, b, c, d}
B6	ϕ	{b, d}
EXIT		ϕ



题二

- 下面C程序分别经非优化编译和2级以上（含2级，如命令行选项 -O2）的优化编译后，生成的两个目标程序运行时的表现不同（例如，编译器是GCC: (GNU) 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)）。运行时的表现有何不同，并说明原因。

```
int f(int g( ) ) {
        return g(g);
}
int main() {
        f(f);
}
```

不开优化：递归调用，且无递归终止条件，最终因为栈满而OOM

开优化：f函数定义中的函数调用被转换成跳转到f函数标号的指令。由于不存在栈帧的增长，因此表现为死循环（而不是OOM）

题二

- **下面C程序分别经非优化编译和2级以上（含2级，如命令行选项 -O2）的优化编译后，生成的两个目标程序运行时的表现不同（例如，编译器是GCC: (GNU) 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)）。运行时的表现有何不同，并说明原因。**

优化前

```
f:
    ...
    subq    $16, %rsp
    movq    %rdi, -8(%rbp)
    movq    -8(%rbp), %rax
    movq    -8(%rbp), %rdx
    movq    %rax, %rdi
    movl    $0, %eax
    call    *%rdx
    leave
    ret
```

优化后

```
f:
    ...
    xorl    %eax, %eax
    jmp     *%rdi
    ...
```

```
int f(int g( ) ) {
        return g(g);
}
int main() {
        f(f);
}
```

第三次习题课-PW7

为SysYF语言生成LLVM IR

刘硕
zkdliushuo@mail.ustc.edu.cn

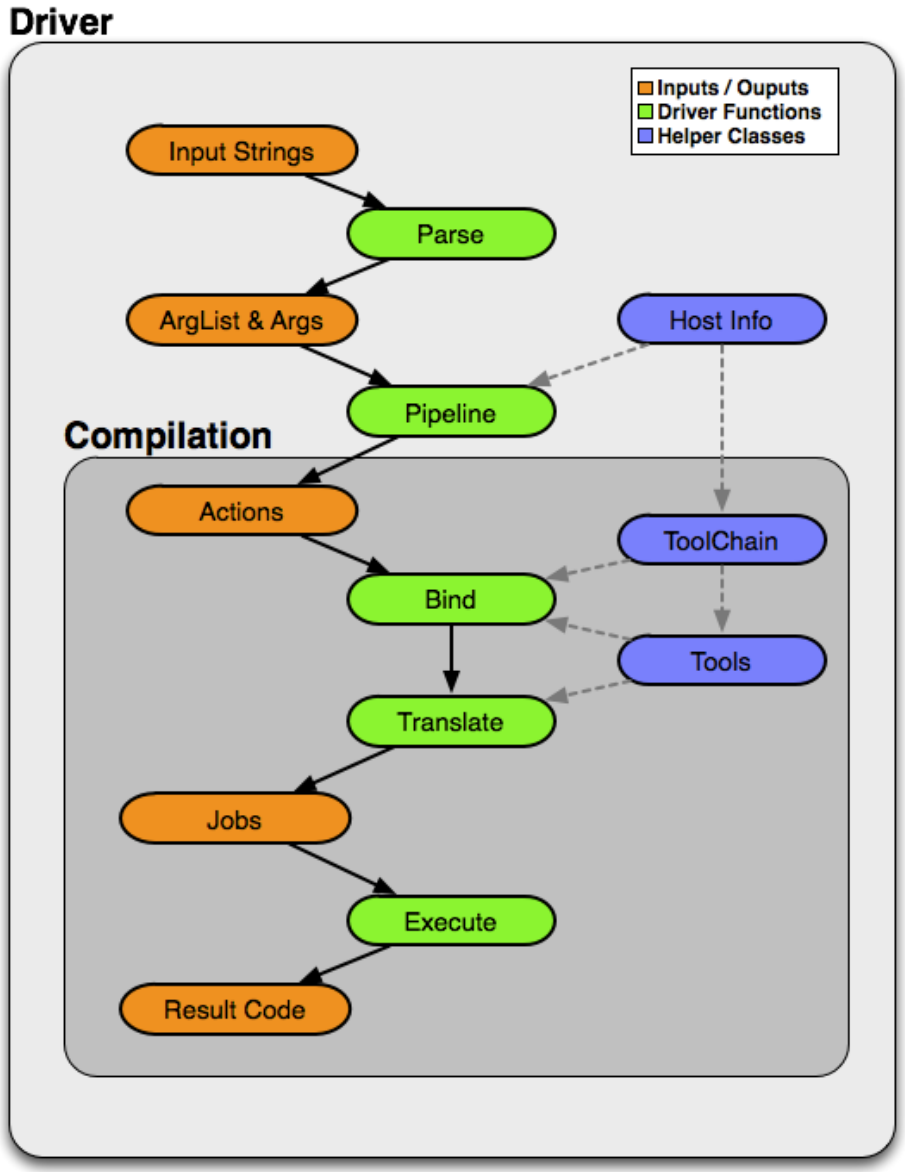
LLVM驱动程序的总体概念及主要流程

Clang Driver

- Clang Driver 提供了与GCC兼容的命令行接口，并提供了对LLVM编译器和相关工具的高效访问
  - **Parser 函数**
    - ✓ 将命令行字符串，解析成具体的参数对象

```
$ clang -### -Xarch_i386 -fomit-frame-pointer -Wa,-fast -Ifoo -I foo t.c

Option 0 - Name: "-Xarch_", Values: {"i386", "-fomit-frame-pointer"}
Option 1 - Name: "-Wa,", Values: {"-fast"}
Option 2 - Name: "-I", Values: {"foo"}
Option 3 - Name: "-I", Values: {"foo"}
Option 4 - Name: "<input>", Values: {"t.c"}
```

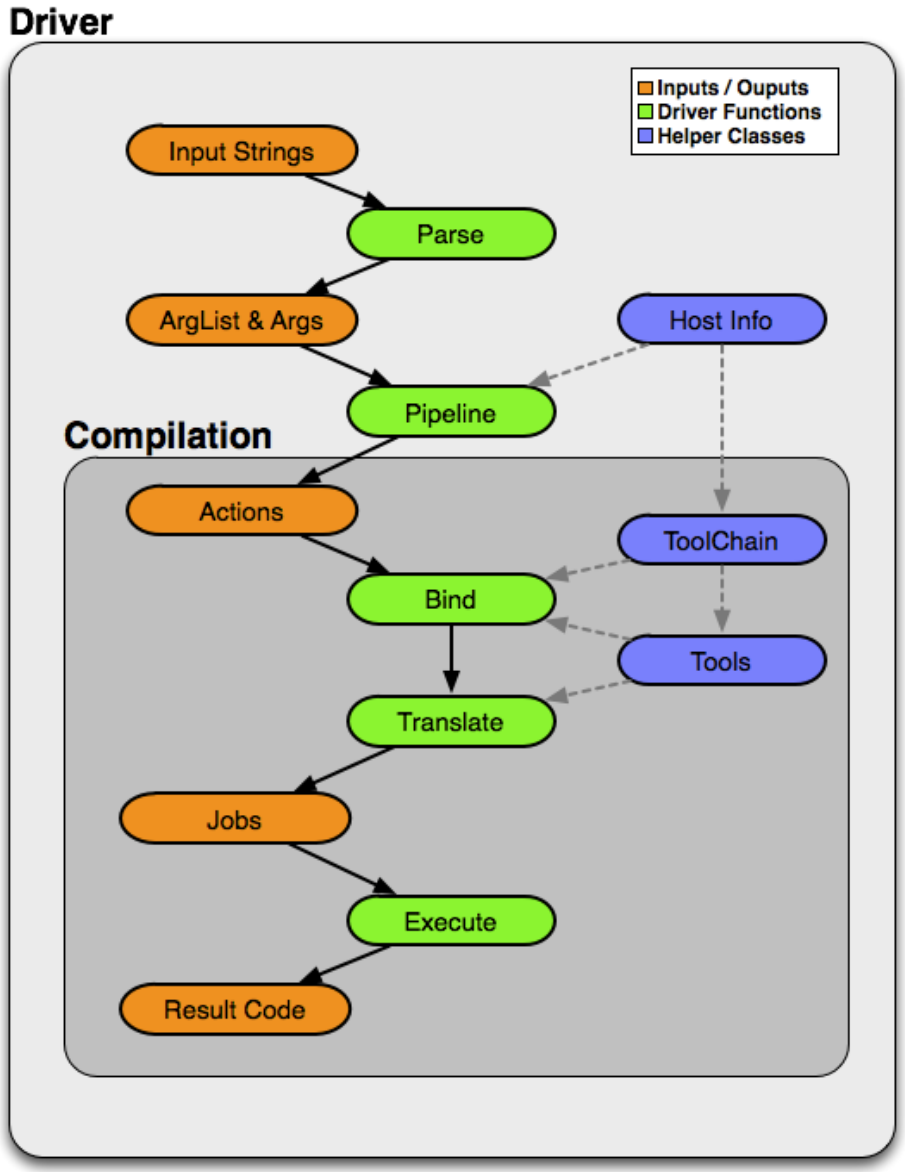


LLVM驱动程序的总体概念及主要流程

Clang Driver

- Clang Driver 提供了与GCC兼容的命令行接口，并提供了对LLVM编译器和相关工具的高效访问
  - **Pipeline 函数**
    - ✓ 根据解析的参数，构造后续编译所需要的子任务(action)

```
$ clang -ccc-print-phases -x c t.c -x assembler t.s
0: input, "t.c", c
1: preprocessor, {0}, cpp-output
2: compiler, {1}, assembler
3: assembler, {2}, object
4: input, "t.s", assembler
5: assembler, {4}, object
6: linker, {3, 5}, image
```



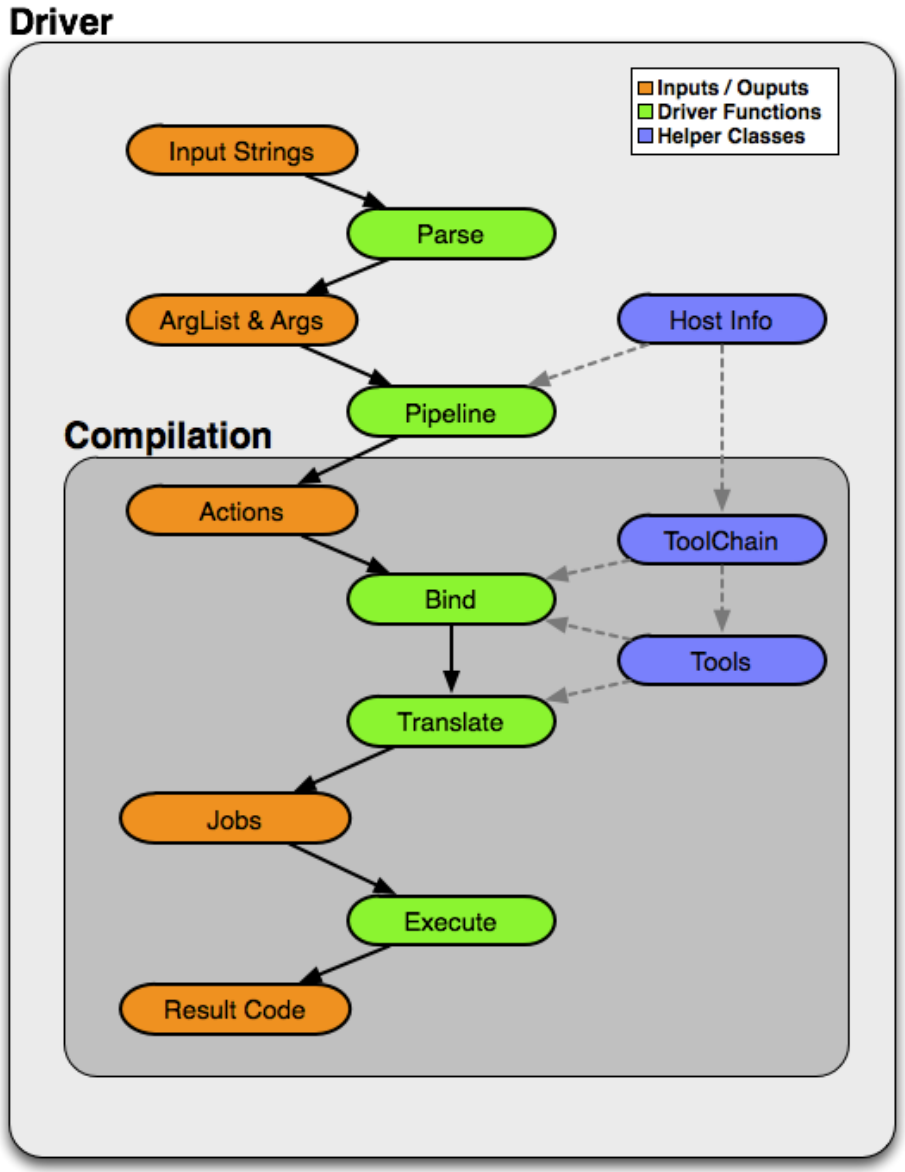
LLVM驱动程序的总体概念及主要流程

Clang Driver

- Clang Driver 提供了与GCC兼容的命令行接口，并提供了对LLVM编译器和相关工具的高效访问
  - **Bind 函数**
    - ✓ Driver自上而下匹配，将Actioins分配给分配给Tools
    - ✓ Driver根据选择的Tools决定如何连接

```
1  $ clang -ccc-print-bindings -arch i386 -arch ppc t0.c
2  # "i386-apple-darwin9" - "clang", inputs: ["t0.c"], output: "/tmp/cc-Sn4RKF.s"
3  # "i386-apple-darwin9" - "darwin::Assemble", inputs: ["/tmp/cc-Sn4RKF.s"], output:
```

  - **Translate 函数**
    - ✓ 根据Tool的期望，将gcc风格的命令行选项翻译成具体的Command



llvm-ustc-proj 驱动程序流程

CompilerInvocation(CI)
（1）管理运行编译器所必须的各个对象，
如 preprocessor、target information、AST context 等；
（2）提供创建和操作常用 Clang 对象的有用方法。

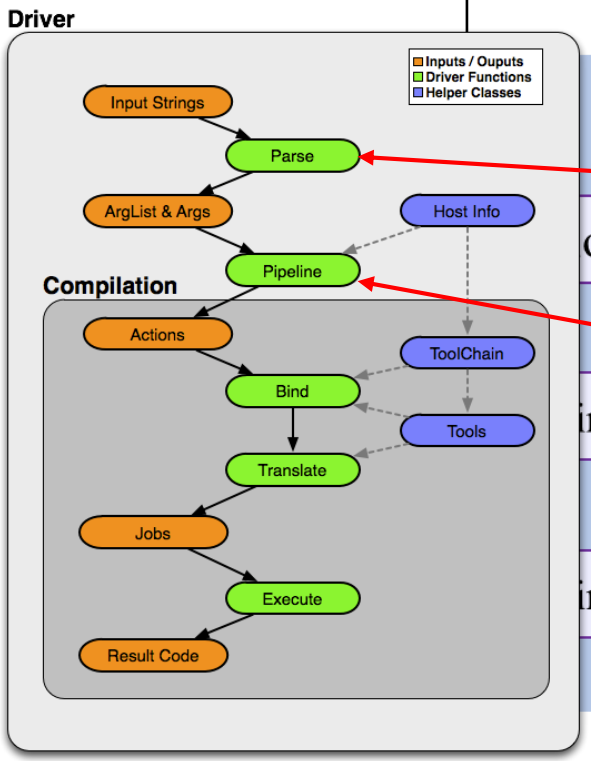
main

1. 创建诊断，用于warning等显示

2. 初始化Clang Driver

3. 调用cDriver得到单个Job，用于构造CI (CompilerInvocation)

4. 通过CI生成LLVM IR，执行AST Checker，在Module上执行Pass



mDriver

llvm::sys::findProgramByName

llvm::sys::getProcessTriple

Clang

clang::driver::Driver

mDriver

clang::driver::BuildCompilation (Args)

CompilerInvocation::CreateFromArgs(Jobs.begin(),…)

mDriver

CI::ExecuteAction(llvm::EmitLLVMOnlyAction)

llvm::EmitLLVMAction::takeModule

FunctionPassManager::Run

PassManager::Run

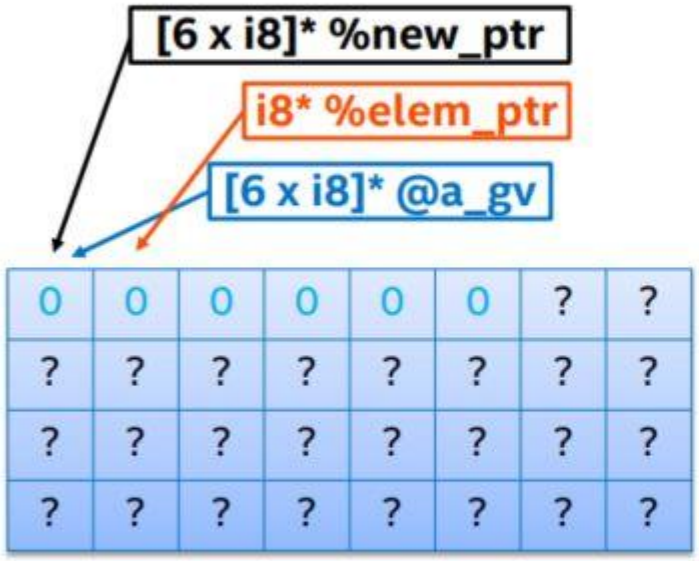
getelementptr 指令

格式：`<result> = getelementptr <type>, <type>* <ptrval> [, <type> <idx>]`

例子：

```
%2 = getelementptr [10 x i32], [10 x i32]* %1, i32 0, i32 %0
%2 = getelementptr i32, i32* %1 i32 %0
```

Manipulating pointers



"Offset by 0 **elements of the base type**"

```
@a_gv = global [6 x i8] zeroinitializer

%new_ptr = getelementptr [6 x i8], [6 x i8]* @a_gv, i32 0

%elem_pt = getelementptr [6 x i8], [6 x i8]* @a_gv, i32 0, i32 1
```

Get the 1st element from the current aggregate:

[6 x i8]

谢谢