

第八章 代码优化

数据流分析

哈尔滨工业大学 陈鄞



数据流分析(data-flow analysis)

➤ 数据流分析

➤ 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

数据流分析模式

➤ 语句的数据流模式

➤ $IN[s]$: 语句 s 之前的数据流值

$OUT[s]$: 语句 s 之后的数据流值

➤ f_s : 语句 s 的传递函数(transfer function)

➤ 一个赋值语句 s 之前和之后的数据流值的关系

➤ 传递函数的两种风格

➤ 信息沿执行路径前向传播(前向数据流问题)

$$OUT[s] = f_s(IN[s])$$

➤ 信息沿执行路径逆向传播(逆向数据流问题)

$$IN[s] = f_s(OUT[s])$$

数据流分析模式

- 语句的数据流模式
 - $IN[s]$: 语句 s 之前的数据流值
 - $OUT[s]$: 语句 s 之后的数据流值
 - f_s : 语句 s 的传递函数(transfer function)
 - 一个赋值语句 s 之前和之后的数据流值的关系
 - 基本块中相邻两个语句之间的数据流值的关系
 - 设基本块 B 由语句 s_1, s_2, \dots, s_n 顺序组成, 则

$$IN[s_{i+1}] = OUT[s_i] \quad i=1, 2, \dots, n-1$$

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
- $OUT[B]$: 紧随基本块 B 之后的数据流值
- 设基本块 B 由语句 s_1, s_2, \dots, s_n 顺序组成, 则

- $IN[B] = IN[s_1]$

- $OUT[B] = OUT[s_n]$

- f_B : 基本块 B 的传递函数

- 前向数据流问题: $OUT[B] = f_B(IN[B])$

$$f_B = f_{s_n} \cdots f_{s_2} f_{s_1}$$

- 逆向数据流问题: $IN[B] = f_B(OUT[B])$

$$f_B = f_{s_1} f_{s_2} \cdots f_{s_n}$$

$$OUT[B]$$

$$= OUT[s_n]$$

$$= f_{s_n}(IN[s_n])$$

$$= f_{s_n}(OUT[s_{n-1}])$$

$$= f_{s_n} f_{s_{n-1}}(IN[s_{n-1}])$$

$$= f_{s_n} f_{s_{n-1}}(OUT[s_{n-2}])$$

.....

$$= f_{s_n} f_{s_{n-1}} \cdots f_{s_2}(OUT[s_1])$$

$$= f_{s_n} f_{s_{n-1}} \cdots f_{s_2} f_{s_1}(IN[s_1])$$

$$= f_{s_n} f_{s_{n-1}} \cdots f_{s_2} f_{s_1}(IN[B])$$



第八章 代码优化

数据流分析

哈尔滨工业大学 陈鄞



→ 以下未看，不知是否重点



第八章 代码优化

到达定值分析

哈尔滨工业大学 陈鄞



到达定值分析

➤ 定值 (Definition)

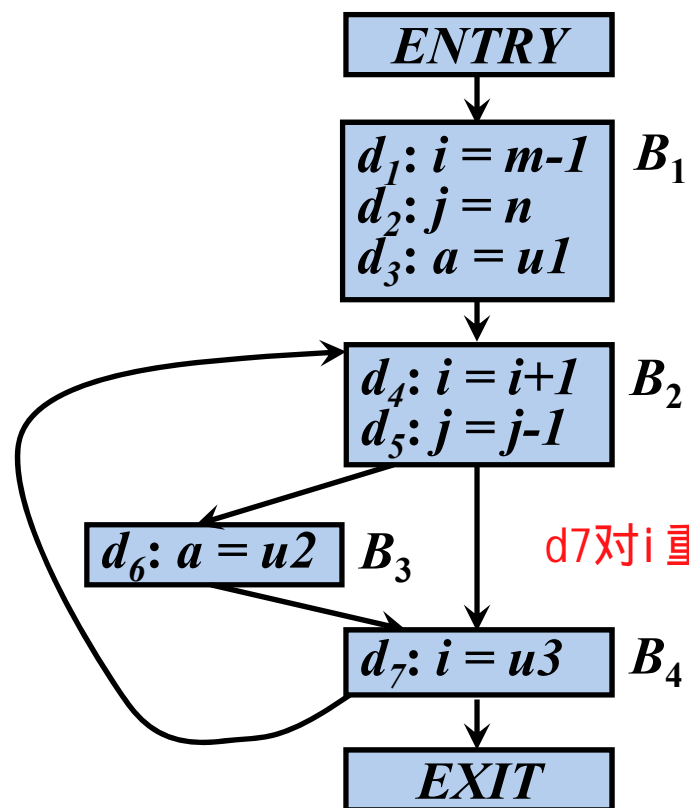
➤ 变量 x 的定值是(可能)将一个值赋给 x 的语句

➤ 到达定值 (Reaching Definition)

➤ 如果存在一条从紧跟在定值 d 后面的点到达某一程序点 p 的路径, 而且在此路径上 d 没有被“杀死” (如果在此路径上有对变量 x 的其它定值 d' , 则称变量 x 被这个定值 d' “杀死”了), 则称定值 d 到达程序点 p

➤ 直观地讲, 如果某个变量 x 的一个定值 d 到达点 p , 在点 p 处使用的 x 的值可能就是由 d 最后赋予的

例：可以到达各基本块的入口处的定值



假设每个控制流图都有两个空基本块, 分别是表示流图的开始点的 $ENTRY$ 结点和结束点的 $EXIT$ 结点 (所有离开该图的控制流都流向它)

$IN[B]$	B_2	B_3	B_4
d_1	✓	×	×
d_2	✓	×	×
d_3	✓	✓	✓
d_4	×	✓	✓
d_5	✓	✓	✓
d_6	✓	✓	✓
d_7	✓	×	×

d_7 对 i 重新定值

到达定值分析的主要用途

➤ 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面(包括 y 或 z 是常数的特殊情况)，那么 $y+z$ 就是循环不变计算

到达定值分析的主要用途

- 循环不变计算的检测
- 常量合并
 - 如果对变量 x 的某次使用只有一个定值可以到达，并且该定值把一个常量赋给 x ，那么可以简单地把 x 替换为该常量

到达定值分析的主要用途

- 循环不变计算的检测
- 常量合并
- 判定变量 x 在 p 点上是否未经定值就被引用

“生成”与“杀死”定值

这里，“+” 代表一个一般性的二元运算符

➤ 定值 d : $u = v + w$

➤ 该语句“生成”了一个对变量 u 的定值 d ，并“杀死”了程序中其它对 u 的定值

到达定值的传递函数

➤ f_d : 定值 d : $u = v + w$ 的传递函数

➤ $f_d(x) = gen_d \cup (x - kill_d)$ —— 生成-杀死形式

➤ gen_d : 由语句 d 生成的定值的集合 $gen_d = \{d\}$

➤ $kill_d$: 由语句 d 杀死的定值的集合 (程序中所有其它对 u 的定值)

到达定值的传递函数

➤ f_d : 定值 d : $u = v + w$ 的传递函数

➤ $f_d(x) = gen_d \cup (x - kill_d)$

➤ f_B : 基本块 B 的传递函数

➤ $f_B(x) = gen_B \cup (x - kill_B)$

➤ $kill_B = kill_1 \cup kill_2 \cup \dots \cup kill_n$

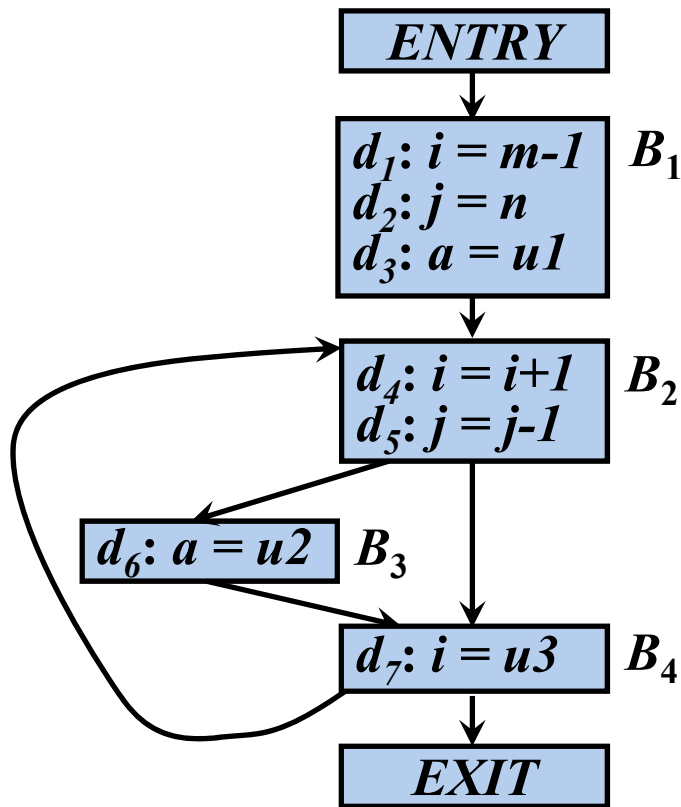
➤ 被基本块 B 中各个语句杀死的定值的集合

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

➤ 基本块中没有被块中各语句“杀死”的定值的集合

例：各基本块 B 的 gen_B 和 $kill_B$

eg



➤ $gen_{B1} = \{ d_1, d_2, d_3 \}$

➤ $kill_{B1} = \{ d_4, d_5, d_6, d_7 \}$

➤ $gen_{B2} = \{ d_4, d_5 \}$

➤ $kill_{B2} = \{ d_1, d_2, d_7 \}$

➤ $gen_{B3} = \{ d_6 \}$

➤ $kill_{B3} = \{ d_3 \}$

➤ $gen_{B4} = \{ d_7 \}$

➤ $kill_{B4} = \{ d_1, d_4 \}$

到达定值的数据流方程

➤ $IN[B]$: 到达流图中基本块 B 的入口处的定值的集合

$OUT[B]$: 到达流图中基本块 B 的出口处的定值的集合

➤ 方程

➤ $OUT[ENTRY] = \Phi$

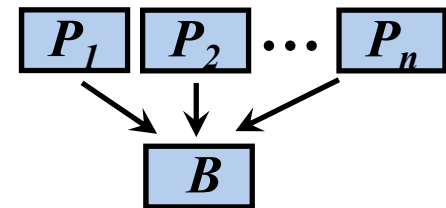
➤ $OUT[B] = f_B(IN[B]) \quad (B \neq ENTRY)$


➤ $f_B(x) = gen_B \cup (x - kill_B)$

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

➤ $IN[B] = \bigcup_{P \text{ 是 } B \text{ 的一个前驱}} OUT[P] \quad (B \neq ENTRY)$

gen_B 和 $kill_B$ 的值可以直接从流图计算出来, 因此在方程中作为已知量






第八章 代码优化

到达定值分析

哈尔滨工业大学 陈鄞





第八章 代码优化

到达定值方程的计算

哈尔滨工业大学 陈鄞



计算到达定值的迭代算法

➤ 输入:

➤ 流图 G , 其中每个基本块 B 的 gen_B 和 $kill_B$ 都已计算出来

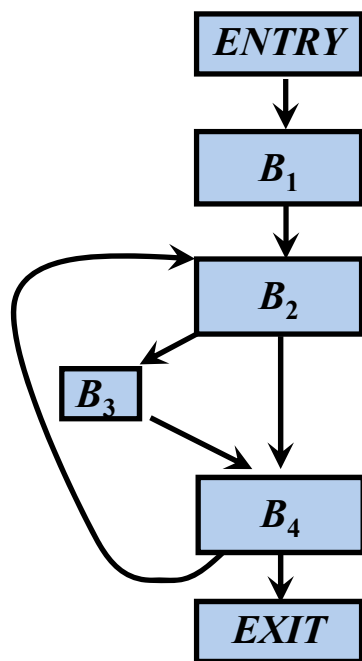
➤ 输出:

➤ $IN[B]$ 和 $OUT[B]$

➤ 方法:

```
 $OUT[ENTRY] = \Phi;$   
for (除 $ENTRY$ 之外的每个基本块 $B$ )  $OUT[B] = \Phi;$   
while (某个 $OUT$ 值发生了改变)  
    for (除 $ENTRY$ 之外的每个基本块 $B$ ) {  
         $IN[B] = \bigcup_{P \text{ 是 } B \text{ 的一个前驱}} OUT[P];$   
         $OUT[B] = gen_B \cup (IN[B] - kill_B)$   
    }
```


例



$gen_{B1} = \{d_1, d_2, d_3\}$
 $kill_{B1} = \{d_4, d_5, d_6, d_7\}$
 $gen_{B2} = \{d_4, d_5\}$
 $kill_{B2} = \{d_1, d_2, d_7\}$
 $gen_{B3} = \{d_6\}$
 $kill_{B3} = \{d_3\}$
 $gen_{B4} = \{d_7\}$
 $kill_{B4} = \{d_1, d_4\}$

$OUT[ENTRY] = \Phi;$
 for (除ENTRY之外的每个基本块B) $OUT[B] = \Phi;$
 while (某个OUT值发生了改变)
 for (除ENTRY之外的每个基本块B) {
 $IN[B] = \bigcup_{P \text{ 是 } B \text{ 的一个前驱}} OUT[P];$
 $OUT[B] = gen_B \cup (IN[B] - kill_B)$
 }

gen置1, kill置0

迭代次数

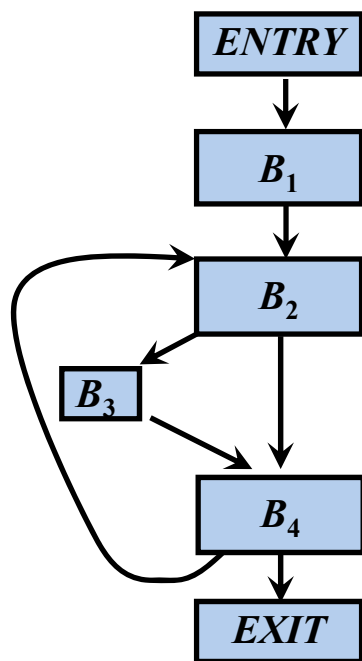
B	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$	$IN[B]^3$	$OUT[B]^3$
B_1	000 0000	000 0000	111 0000	000 0000	111 0000	000 0000	111 0000
B_2	000 0000	111 0000	001 1100	111 0111	001 1110	111 0111	001 1110
B_3	000 0000	001 1100	000 1110	001 1110	000 1110	001 1110	000 1110
B_4	000 0000	001 1110	001 0111	001 1110	001 0111	001 1110	001 0111
EXIT	000 0000	001 0111	001 0111	001 0111	001 0111	001 0111	001 0111

0 在第一位不出现

1 在第三位出现

7 : 37

例



$gen_{B1} = \{d_1, d_2, d_3\}$
 $kill_{B1} = \{d_4, d_5, d_6, d_7\}$
 $gen_{B2} = \{d_4, d_5\}$
 $kill_{B2} = \{d_1, d_2, d_7\}$
 $gen_{B3} = \{d_6\}$
 $kill_{B3} = \{d_3\}$
 $gen_{B4} = \{d_7\}$
 $kill_{B4} = \{d_1, d_4\}$

$IN[B]$	B_2	B_3	B_4
d_1	✓	✗	✗
d_2	✓	✗	✗
d_3	✓	✓	✓
d_4	✗	✓	✓
d_5	✓	✓	✓
d_6	✓	✓	✓
d_7	✓	✗	✗

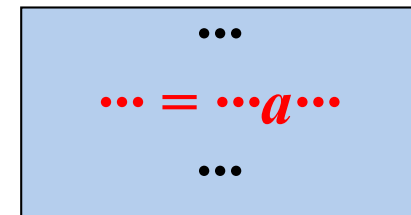
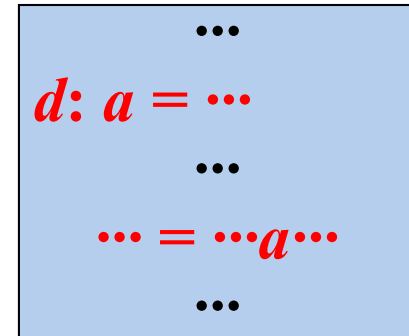
B	$OUT[B]^0$	$IN[B]^1$	$OUT[B]^1$	$IN[B]^2$	$OUT[B]^2$	$IN[B]^3$	$OUT[B]^3$
B_1	000 0000	000 0000	111 0000	000 0000	111 0000	000 0000	111 0000
B_2	000 0000	111 0000	001 1100	111 0111	001 1110	111 0111	001 1110
B_3	000 0000	001 1100	000 1110	001 1110	000 1110	001 1110	000 1110
B_4	000 0000	001 1110	001 0111	001 1110	001 0111	001 1110	001 0111
$EXIT$	000 0000	001 0111	001 0111	001 0111	001 0111	001 0111	001 0111


引用-定值链 (*Use-Definition Chains*)

➤ **引用-定值链**(简称 ud 链) 是一个列表, 对于变量的每一次引用, 到达该引用的所有定值都在该列表中

➤ 如果块 B 中变量 a 的引用之前有 a 的定值, 那么只有 a 的最后一次定值会在该引用的 ud 链中

➤ 如果块 B 中变量 a 的引用之前没有 a 的定值, 那么 a 的这次引用的 ud 链就是 $IN[B]$ 中 a 的定值的集合





第八章 代码优化

到达定值方程的计算

哈尔滨工业大学 陈鄞

