



第八章 代码优化

活跃变量分析

哈尔滨工业大学 陈鄞

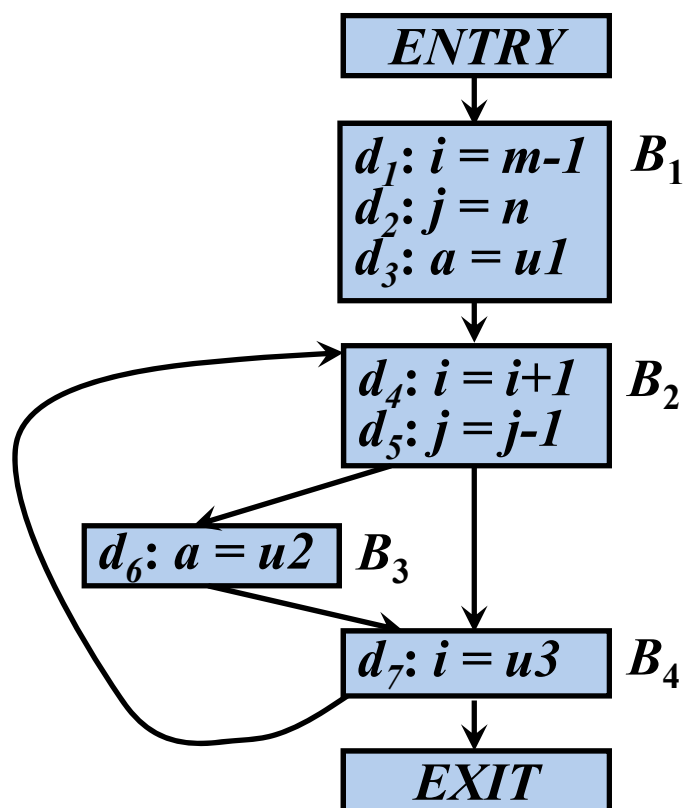


活跃变量分析

➤ 活跃变量

- 对于变量 x 和程序点 p ，如果在流图中沿着从 p 开始的某条路径会引用变量 x 在 p 点的值，则称变量 x 在点 p 是活跃(*live*)的，否则称变量 x 在点 p 不活跃(*dead*)

例：各基本块的出口处的活跃变量



$OUT[B]$	B_1	B_2	B_3	B_4
a	×	×	×	×
i	√	×	×	√
j	√	√	√	√
m	×	×	×	×
n	×	×	×	×
$u1$	×	×	×	×
$u2$	√	√	√	√
$u3$	√	√	√	√

活跃变量信息的主要用途

➤ 删除无用赋值

- **无用赋值**：如果 x 在点 p 的定值在基本块内所有后继点都不被引用，且 x 在基本块出口之后又是**不活跃**的，那么 x 在点 p 的定值就是无用的

➤ 为基本块分配寄存器

- 如果**所有寄存器都被占用**，并且还需要申请一个寄存器，则应该考虑使用已经存放了死亡值的寄存器，因为这个值不需要保存到内存
- 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**

活跃变量的传递函数

➤ 逆向数据流问题

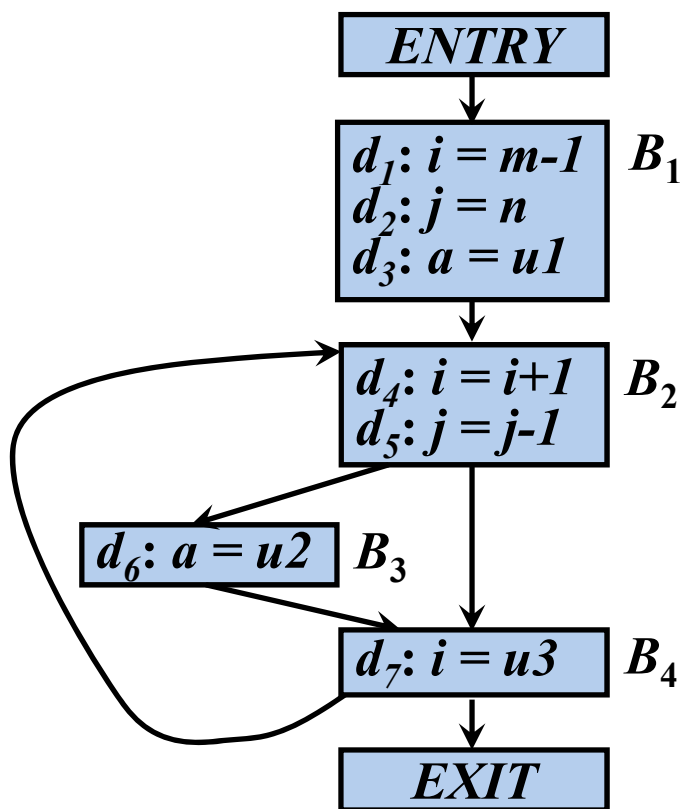
➤ $IN[B] = f_B(OUT[B])$

➤ $f_B(x) = use_B \cup (x - def_B)$

➤ def_B : 在基本块 B 中定值, 但是定值前在 B 中没有被引用的变量的集合

➤ use_B : 在基本块 B 中引用, 但是引用前在 B 中没有被定值的变量集合

例：各基本块 B 的 use_B 和 def_B



➤ $use_{B1} = \{ m, n, u1 \}$

➤ $def_{B1} = \{ i, j, a \}$

➤ $use_{B2} = \{ i, j \}$

➤ $def_{B2} = \Phi$

➤ $use_{B3} = \{ u2 \}$

➤ $def_{B3} = \{ a \}$

➤ $use_{B4} = \{ u3 \}$

➤ $def_{B4} = \{ i \}$

活跃变量数据流方程

- $IN[B]$: 在基本块 B 的入口处的活跃变量集合
- $OUT[B]$: 在基本块 B 的出口处的活跃变量集合

- 方程

- $IN[EXIT] = \Phi$

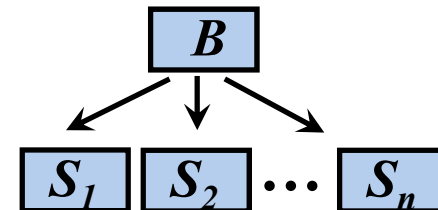
- $IN[B] = f_B(OUT[B]) \quad (B \neq EXIT)$

- $f_B(x) = use_B \cup (x - def_B)$

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

- $OUT[B] = \bigcup_{S \text{ 是 } B \text{ 的一个后继}} IN[S] \quad (B \neq EXIT)$

use_B 和 def_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

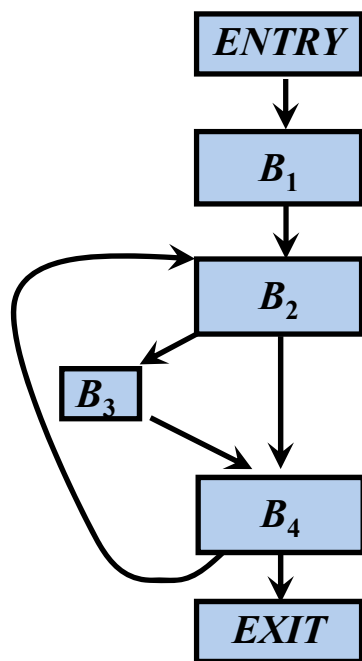


计算活跃变量的迭代算法

- 输入：流图 G ，其中每个基本块 B 的 use_B 和 def_B 都已计算出来
- 输出： $IN[B]$ 和 $OUT[B]$
- 方法：

```
 $IN[EXIT] = \Phi;$   
for (除  $EXIT$  之外的每个基本块  $B$ )  $IN[B] = \Phi;$   
while (某个  $IN$  值发生了改变)  
    for (除  $EXIT$  之外的每个基本块  $B$ ) {  
         $OUT[B] = \bigcup_{S \text{ 是 } B \text{ 的一个后继}} IN[S];$   
         $IN[B] = use_B \cup (OUT[B] - def_B);$   
    }
```

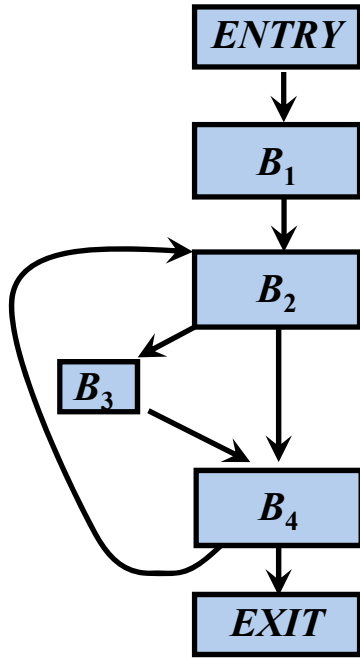

例



$use_{B1} = \{ m, n, u1 \}$
 $def_{B1} = \{ i, j, a \}$
 $use_{B2} = \{ i, j \}$
 $def_{B2} = \Phi$
 $use_{B3} = \{ u2 \}$
 $def_{B3} = \{ a \}$
 $use_{B4} = \{ u3 \}$
 $def_{B4} = \{ i \}$

$IN[EXIT] = \Phi;$
for (除EXIT之外的每个基本块B) $IN[B] = \Phi;$
while (某个IN值发生了改变)
 for (除EXIT之外的每个基本块B) {
 $OUT[B] = \bigcup_{S \text{ 是 } B \text{ 的一个后继}} IN[S];$
 $IN[B] = use_B \cup (OUT[B] - def_B);$
 }
 }

	$OUT[B]^1$	$IN[B]^1$	$OUT[B]^2$	$IN[B]^2$	$OUT[B]^3$	$IN[B]^3$
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	$i, j, u2, u3$	$m, n, u1, u2, u3$	$i, j, u2, u3$	$m, n, u1, u2, u3$	$i, j, u2, u3$	$m, n, u1, u2, u3$



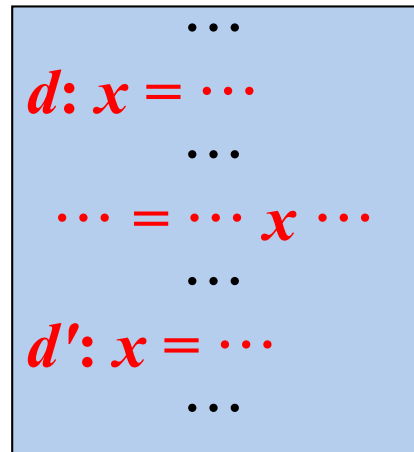
$use_{B1} = \{ m, n, u1 \}$
 $def_{B1} = \{ i, j, a \}$
 $use_{B2} = \{ i, j \}$
 $def_{B2} = \Phi$
 $use_{B3} = \{ u2 \}$
 $def_{B3} = \{ a \}$
 $use_{B4} = \{ u3 \}$
 $def_{B4} = \{ i \}$

$OUT[B]$	B_1	B_2	B_3	B_4
a	×	×	×	×
i	√	×	×	√
j	√	√	√	√
m	×	×	×	×
n	×	×	×	×
u_1	×	×	×	×
u_2	√	√	√	√
u_3	√	√	√	√

	$OUT[B]^1$	$IN[B]^1$	$OUT[B]^2$	$IN[B]^2$	$OUT[B]^2$	$IN[B]^2$
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	$i, j, u2, u3$	$m, n, u1, u2, u3$	$i, j, u2, u3$	$m, n, u1, u2, u3$	$i, j, u2, u3$	$m, n, u1, u2, u3$

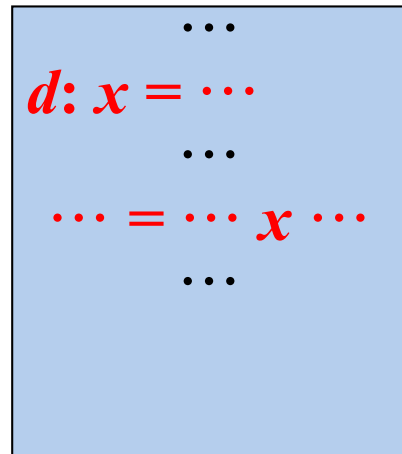
定值-引用链 (Definition-Use Chains)


- 定值-引用链：设变量 x 有一个定值 d ，该定值所有能够到达的引用 u 的集合称为 x 在 d 处的**定值-引用链**，简称**du链**
- 如果在求解**活跃变量**数据流方程中的 $OUT[B]$ 时，将 $OUT[B]$ 表示成**从 B 的末尾处能够到达的引用的集合**，那么，可以直接利用这些信息计算基本块 B 中每个变量 x 在其定值处的**du链**
- 如果 B 中 x 的定值 d 之后有 x 的第一个定值 d' ，则 d 和 d' 之间 x 的所有引用构成 d 的**du链**



定值-引用链 (Definition-Use Chains)

- 定值-引用链：设变量 x 有一个定值 d ，该定值所有能够到达的引用 u 的集合称为 x 在 d 处的**定值-引用链**，简称**du链**
- 如果在求解**活跃变量**数据流方程中的 $OUT[B]$ 时，将 $OUT[B]$ 表示成**从 B 的末尾处能够到达的引用的集合**，那么，可以直接利用这些信息计算基本块 B 中每个变量 x 在其定值处的**du链**
- 如果 B 中 x 的定值 d 之后有 x 的第一个定值 d' ，
则 d 和 d' 之间 x 的所有引用构成 d 的**du链**
- 如果 B 中 x 的定值 d 之后没有 x 的新的定值，
则 B 中 d 之后 x 的所有**引用**以及 $OUT[B]$ 中 x 的所有**引用**构成 d 的**du链**






第八章 代码优化

活跃变量分析

哈尔滨工业大学 陈鄞





第八章 代码优化

可用表达式分析

哈尔滨工业大学 陈鄞



可用表达式分析

➤ 可用表达式

➤ 如果从流图的首节点到达程序点 p 的每条路径都对表达式 $x \text{ op } y$ 进行计算，并且从最后一个这样的计算到点 p 之间没有再次对 x 或 y 定值，那么表达式 $x \text{ op } y$ 在点 p 是可用 (available) 的

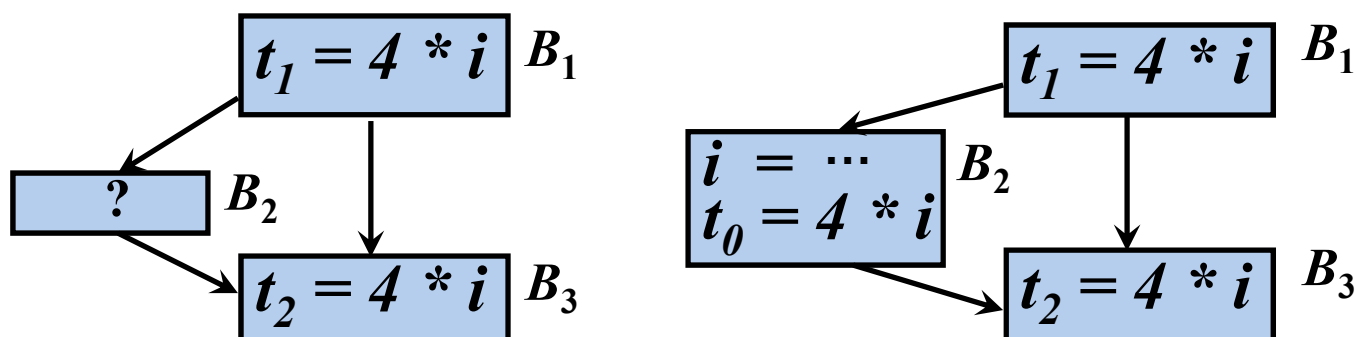
➤ 表达式可用的直观意义

➤ 在点 p 上， $x \text{ op } y$ 已经在之前被计算过，不需要重新计算

可用表达式信息的主要用途

➤ 消除全局公共子表达式

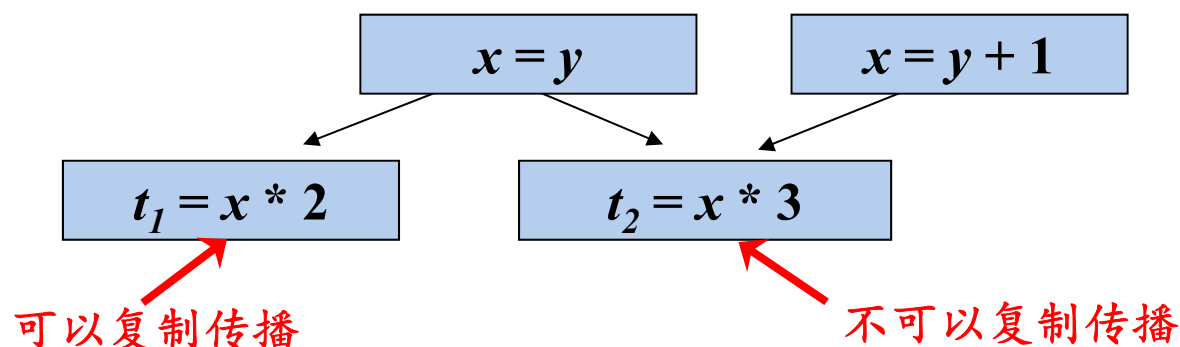
➤ 例



可用表达式信息的主要用途

- 消除全局公共子表达式
- 进行复制传播

➤ 例



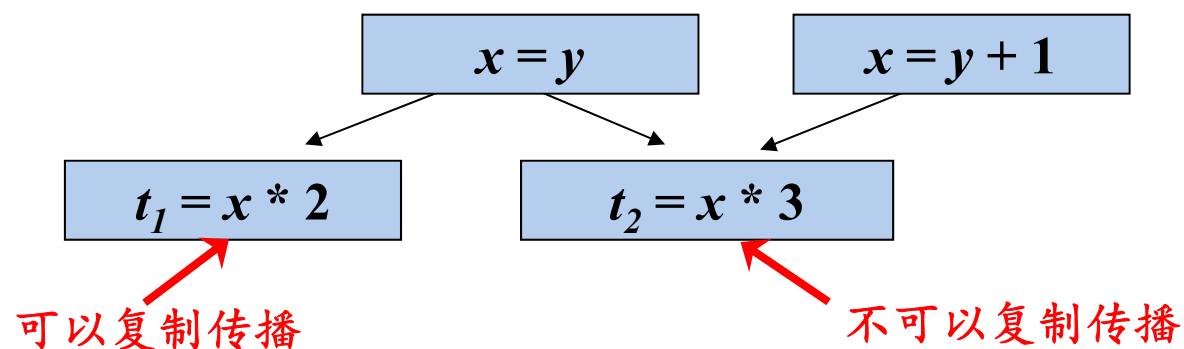
在 x 的引用点 u 可以用 y 代替 x 的条件：复制语句 $x = y$ 在引用点 u 处可用

从流图的首节点到达 u 的每条路径都存在复制语句 $x = y$ ，并且从最后一条复制语句 $x = y$ 到点 u 之间没有再次对 x 或 y 定值

可用表达式信息的主要用途

- 消除全局公共子表达式
- 进行复制传播

➤ 例



可用表达式的传递函数

- 对于可用表达式数据流模式而言，如果基本块 B 对 x 或者 y 进行了(或可能进行)定值，且以后没有重新计算 $x \text{ op } y$ ，则称 B 杀死表达式 $x \text{ op } y$ 。如果基本块 B 对 $x \text{ op } y$ 进行计算，并且之后没有重新定值 x 或 y ，则称 B 生成表达式 $x \text{ op } y$
- $f_B(x) = e_gen_B \cup (x - e_kill_B)$
 - e_gen_B ：基本块 B 所生成的可用表达式的集合
 - e_kill_B ：基本块 B 所杀死的 U 中的可用表达式的集合
 - U ：所有出现在程序中一个或多个语句的右部的表达式的全集

e_gen_B 的计算

- 初始化: $e_gen_B = \Phi$
 - 顺序扫描基本块的每个语句: $z = x \text{ op } y$
 - 把 $x \text{ op } y$ 加入 e_gen_B
 - 从 e_gen_B 中删除和 z 相关的表达式
- } 顺序不能颠倒

语句	可用表达式
..... $a := b+c$	\emptyset
..... $b := a-d$	$\{ b+c \}$
..... $c := b+c$	$\{ a-d \}$
..... $d := a-d$	$\{ a-d \}$
.....	\emptyset

e_kill_B 的计算

- 初始化: $e_kill_B = \Phi$
- 顺序扫描基本块的每个语句: $z = x \text{ op } y$
 - 从 e_kill_B 中删除表达式 $x \text{ op } y$
 - 把所有和 z 相关的表达式加入到 e_kill_B 中

可用表达式的数据流方程

➤ $IN[B]$: 在 B 的入口处可用的 U 中的表达式集合

$OUT[B]$: 在 B 的出口处可用的 U 中的表达式集合

➤ 方程

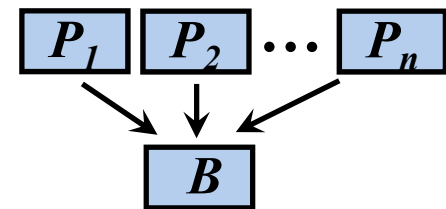
➤ $OUT[ENTRY] = \Phi$

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

➤ $f_B(x) = e_gen_B \cup (x - e_kill_B)$

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

e_gen_B 和 e_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



计算可用表达式的迭代算法

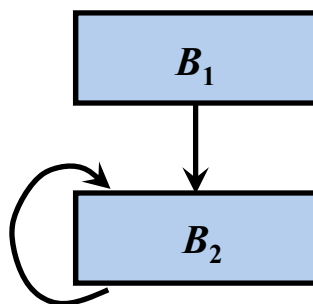
- 输入：流图G，其中每个基本块 B 的 e_gen_B 和 e_kill_B 都已计算出来
- 输出： $IN[B]$ 和 $OUT[B]$
- 方法：

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

为什么将 $OUT[B]$ 集合初始化为 U ?

➤ 将 OUT 集合初始化为 Φ 局限性太大

➤ 例



➤ 如果 $OUT[B_2]^0 = \Phi$

那么 $IN[B_2]^1 = OUT[B_1]^1 \cap OUT[B_2]^0 = \Phi$

➤ 如果 $OUT[B_2]^0 = U$

那么 $IN[B_2]^1 = OUT[B_1]^1 \cap OUT[B_2]^0 = OUT[B_1]$



第八章 代码优化

可用表达式分析

哈尔滨工业大学 陈鄞

