续表

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

9.2.4 活跃变量

一些代码改进变换依赖从程序流图逆向计算得到的信息，现在考虑其中的**活跃变量分析**。在活跃变量分析中，对变量 x 和点 p，希望知道 x 的值是否可能在从点 p 开始的某个路径上被引用，如果可能被引用，就说 x 在点 p 是**活跃**的，否则称 x 在点 p 是**死亡**的。

活跃变量信息的一种重要应用就是基本块的寄存器分配，8.3 和 8.4 节对此已有简单的介绍。一个保存在寄存器中的值，如果在该块出口是死亡的，那么离开该块时这个值不必存储。还有，如果所有寄存器都被占用，此时需要释放一个寄存器，则首先选择保存死亡值的寄存器，因为这个值不必存储。

数据流等式的定义仍然直接根据 IN[B]和 OUT[B]，它们分别代表基本块 B 入口和出口的活跃变量集合。这些等式也是由先定义单个语句的迁移函数并把它们复合成基本块的迁移函数。对于单个语句 d:u=v+w 来说，它产生 v 和 w 的活跃性，注销 u 的活跃性，因此迁移函数也是产生-注销形式。逆向复合单个语句的迁移函数，得到基本块的迁移函数如下：

（1）use_B 是在块 B 中有引用并且在该引用前 B 中无定值的变量集合，

（2）def_B 是在块 B 中有定值的变量集合。

它们分别对应到达-定值中的 gen_B 和 $kill_B$。

例 9.9 图 9.10 的 B_2 对 i 重新定值前引用 i，对 j 也是这样。因此 $use_{B_2}=\{i,j\}$。另外 B_2 为 i 和 j 定值，因此 $def_{B_2}=\{i,j\}$。 □

根据这两个定义，use_B 中的任何变量在 B 的入口应该是活跃的，而在 def_B 中的变量在 B 的入口应该是死亡的，除非它也出现在 use_B 中。

将 *def* 和 *use* 联系到未知的 IN 和 OUT 的方程组定义如下：

$$\text{IN}[\text{EXIT}]=\varnothing$$

和对所有块 B（EXIT 除外）有

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

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

第一个等式描述了边界条件：没有任何变量在程序终点活跃。第二个等式说，如果一个变量

在块中定值前有引用,或者在该块出口活跃并且没有被该块定值,那么它在该块入口活跃。第三个等式指出,一个变量在块的出口活跃,当且仅当它在该块某个后继块入口活跃。

必须注意活跃性等式和到达-定值等式之间的联系。

(1) 它们都以集合并算符作为它们的汇合算符。因为在这两种数据流模式中,信息都是沿着路径传播,并且仅关心是否有一条具备所关心性质的路径存在,而不是关心某个性质是否在所有路径上都保持。

(2) 活跃性的信息流是逆控制流方向遍历,因为在这个问题中,要保证的是,变量 x 在点 p 之后可能被引用的信息要沿路径逆向传递到点 p 的前驱点,除非 x 正好在这两点之间被定值。

为求解该逆向问题,注意它和到达-定值问题的联系和区别。这里初始化 IN[EXIT]而不是 OUT[ENTRY],IN 和 OUT 集合的作用相互交换,*use* 和 *def* 分别代替了 *gen* 和 *kill*。同样,方程组的解也不一定唯一,所要的也是最小解。用于求这个最小解的算法本质是算法 9.1 的逆向版本。

算法 9.2 活跃变量的迭代求解。

输入 一个流图及各块 B 的 def_B 和 use_B。

输出 该流图中每个块 B(EXIT 除外)的 IN[B]和 OUT[B]。

方法 执行图 9.12 的程序。 □

```
IN[EXIT] = ∅;
for(除了 EXIT 以外的每个块 B)  IN[B] = ∅;
while(任何一个 IN 出现变化)
    for(除了 EXIT 以外的每个块 B){
        OUT[B] = ∪ S是B的后继 IN[S];
        IN[B] = use_B ∪ (OUT[B] - def_B);
    }
```

图 9.12 计算活跃变量的迭代算法

9.2.5 可用表达式

如果从流图起点到点 p 的每条路径上都计算 x+y,并且在到达 p 之前的最后一次计算 x+y 后,一直到 p 为止都没有对 x 或 y 赋值,那么称表达式 x+y 在点 p **可用**。

在可用表达式数据流分析模式中,对于单个语句 d:z=x+y 来说,它产生可用表达式 x+y,注销含 z 的可用表达式,因此迁移函数仍然是产生-注销形式。复合单个语句的迁移函数,得到基本块 B 的迁移函数如下。

(1) e_gen_B 是在块 B 中产生的可用表达式集合。

一个基本块产生可用表达式 x+y 是指它计算 x+y,并且随后没有对 x 或 y 的定值。

(2) e_kill_B 是在块 B 中注销的可用表达式集合。

一个基本块注销含 z 的可用表达式,若它有对 z 定值的语句。