**编译原理第四阶段实验报告**

**Use-Definition Chaining**

**计24 2012011335 柯均洁**

# 实验总述

在实验框架下完成了数据流分析中的**引用-定值链（Use-Definition Chaining）**的求解

首先求出了**到达-定值数据流**，然后在到达-定值数据流信息的基础上计算UD链的信息，其规则如下：

求变量A在位于基本块B中的引用点u的UD链

1. 如果在基本块中，变量A的引用点u之前有A的定值点，且距离u最近的定值点为点d（即A在d的定值可以到达u），那么A在点u的UD链就是{d}
2. 如果在基本块中，变量A的引用点u之前没有A的定值点，那么，in[B]中A的所有定值点均到达u，它们就是A在u的UD链

# UD链实现细节

## （1）到达-定值数据流计算

### 1. 在Tac类中增加一个globalNum域

原来的Tac中只保存了其所在的BasicBlock编号，而计算到达定值数据流需要知道每条语句在整个Functy中的编号，于是在Tac类中增加globalNum域来记录该语句的全局编号。

在FlowGraph的markBasicBlocks函数中增加对TacList中语句的全局编号的计算，从0开始编号，逐条语句加1

### 2. 增加DefRefPoint类

为了记录定值和引用点的信息，在BasicBlock.java中增加了DefRefPoint类，类中保存了被定值或者引用的变量Temp，已经该变量被定值或者引用的语句编号

重载了两种比较器，一个是先按被引用或定值的变量排序的TEMP\_COMPARATOR，另外一个是先按语句编号进行排序的INDEX\_COMPARATOR

### gen[B] kill[B] out[B] in[B]

由于在计算中不仅需要知道这些集合的语句编号，还需要知道这些语句编号所具体对应的变量。因此将这四个集合保存为TreeSet<DefRefPoint>。按照TEMP\_COMPARATOR排序。

### gen[B]计算

在BasicBlock中的computeDefAndLiveUse函数中同时计算gen[B]。出现变量的定值的时候，动态地刷新每个变量最新被定值的位置。当所有语句处理完，就会得到B中被定值的变量以及它所被定值的位置。

### kill[B]计算

在FlowGraph中增加了calKill函数。在得到所有的gen[B]之后，就可以计算出所有变量的有效定值点集合genAll，每个基本块的kill[B]是gen[B]集合中变量所对应的有效定值点几何除去gen[B]

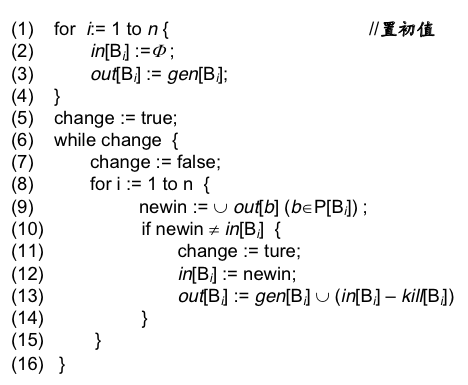
### prev[B]计算

在到达定值数据流计算中需要用到基本块的前驱。因此在BasicBlock中增加Set<Integer> prev，保存该基本块的直接前驱。

在FlowGraph中增加了calPrev函数，利用BasicBlock原有的后继保存域next[2]来计算prev

### out[B]和in[B]计算

在FlowGraph中增加了analyzeArriveDef函数来计算到达-定值数据流，首先调用calPrev()和calKill()。然后按照算法流程计算out[B]和in[B]



## （2）UD链计算

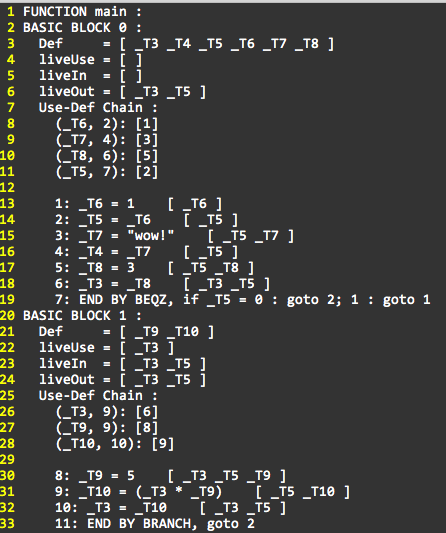
在BasicBlock中增加了calUseDefChain函数

* 为了知道在每个引用点之前是否有块内的定值，需要动态记录每个变量的块内最新定值点保存为一名叫newGenPointInBlock 的hashtable。
* 将UD链保存为名叫udChain的Map<DefRefPoint, List<Integer>>。也就是说每条ud链都是一个变量的引用点，对应一条全局语句标号的链表。
* 对每个引用点，按照算法计算其UD链。

# UD链输出

为了单独输出filename.ud，在BasicBlock中增加了printUDChainTo函数，但是就要单独为输出ud文件单独在Option中增加一个Level，看到实验要求中不能改变Option，所以最终决定在原来活跃变量数据流输出信息的基础上增加，将udChain的信息输出到原来的.dout文件中，稍微改变了原有的格式。

## 输出样例



* 在原有的liveOut后面加入了Use-Def Chain的输出，依次是每个引用点以及对应的UD链
* UD链的信息含义为：

（变量，变量引用点的语句标号）：[ud链的语句标号]

* 在原有的tac语句输出中增加了语句标号