编译原理PA4实验报告

提前上学2018 李嘉图

2019 年 1 月 1 日

1 任务描述

在现有decaf框架的基础上，实现对于定值-引用链的求解。

2 实验操作

框架中已经给出了使用迭代算法求解活跃变量方程的方法，只需要稍加扩展就可以使得其支持定值-引用链的求解。不妨使用`Map<Temp, Set<Integer>>` 来表示`LiveOut`,`LiveUse`和`LiveIn`，表示一个变量及其对应的定值点/引用点；用`Set<Integer>` 来表示`Def`，这里对`Def`的定义做一些修改：所有在基本块内定值的变量都在`Def`中。而之所以使用`Set<Integer>` 存储是因为可以通过TAC的语句编号简单的判断出一个定值是不是在基本块内，因而为了节省空间不需要存其定值点。很明显，数据流方程可以表示为：

$$\begin{aligned} LiveOut[i] &= \bigcup_{i \to j} LiveIn[j] \\ LiveIn[i] &= LiveUse[i] \cup (LiveOut[i] - Def[i]) \end{aligned} \tag{1}$$

其中，$LiveOut[i] - Def[i]$ 被定义为：

$$\{(Var\ x, Position\ p) | (x,p) \in LiveOut[i] \wedge (x \notin Def[i] \vee p \notin i\} \tag{2}$$

2.1 修改几个集合的定义

在`dataflow/BasicBlock.java`中将`LiveUse`, `LiveIn`, `LiveOut`修改为`Map<Temp, Set<Integer>>`，并在构造函数初始化改为：

new TreeMap<Temp, Set<Integer>>(Temp.ID_COMPARATOR)

并将Def修改成Set<Integer>，将构造函数初始化改为：

new TreeSet<Temp>(Temp.ID_COMPARATOR)

在函数BasicBlock.computeDefAndLiveUse 中进行修改，对于在语句tac的定值变量temp，将其加入到def集合中；对于在语句tac使用了的未定值变量temp，将其位置加入到LiveUse[temp]中。

2.2 求出每个基本块的开始和结束位置

在dataflow/BasicBlock.java中加入变量beginId, endId，并在函数BasicBlock.allocateTacIds中计算开始和结束位置

2.3 解流图层次的数据流方程

修改函数FlowGraph.analyzeLiveness，使用如下的迭代算法求解数据流方程：

- 遍历每个基本块b
- 将其LiveOut更新为所有后继节点LiveIn的并，如果LiveOut发生了修改，重新计算LiveIn
- 如果存在一个基本块发生了改变，重复以上步骤

2.4 求基本块内的数据流信息

同样的，对基本块内的数据流信息做扩展，在tac/Tac中新增变量Map<Temp, Set<Integer>> liveOutMap，表示这句TAC语句之后，所有活跃变量的引用位置。修改函数BasicBlock.analyzeLiveness，使用如下算法分析数据流信息：

- 最后一条语句的liveOutMap是整个基本块的LiveOut
- 从后向前依次计算前一条语句的liveOutMap，即使用一个变量就加入liveOutMap，定值一个变量就从liveOutMap中删掉之。

2.5 求出DU链

容易发现，一个定值语句上的引用信息就是其`liveOutMap`，直接记录即可。

3 分析TAC序列和DU链

```
BASIC BLOCK 1 :
21      END BY BEQZ, if _T9 =
            0 : goto 7; 1 : goto 2
BASIC BLOCK 2 :
22      _T14 = 1 [ 23 ]
23      _T3 = _T14 [ 35 ]
24      END BY BEQZ, if _T9 =
            0 : goto 4; 1 : goto 3
BASIC BLOCK 3 :
25      call  _Main.f
26      END BY BRANCH, goto 4
BASIC BLOCK 4 :
27      _T15 = 1 [ 28 ]
28      _T16 = (_T4 + _T15) [ 29 ]
29      _T4 = _T16 [ 28 32 36 ]
30      END BY BEQZ, if _T9 =
            0 : goto 6; 1 : goto 5
BASIC BLOCK 5 :
31      _T17 = 4 [ 32 ]
32      _T18 = (_T4 - _T17) [ 33 ]
33      _T4 = _T18 [ 28 36 ]
34      END BY BRANCH, goto 6
BASIC BLOCK 6 :
35      _T5 = _T3 [ ]
36      _T6 = _T4 [ ]
37      END BY BRANCH, goto 1
```

不妨以`t0.decaf`中循环的一段来分析DU链的分析结果。以$T4$为例考虑变量的定值和引用情况，其定值点在29行和33行。对于第29行的定值点，其后要么经过BLOCK 5回来，要么跳过BLOCK 5，直接从BLOCK 6回来。那么其应当包含BLOCK 6, BLOCK 1-4，以及31行到33行，$T4$未被重新定制之前的所有引用，即$[28, 32, 36]$；对于33行的定值点，其无法到达32行的引用点（因为到达这个引用点必须经过BLOCK 4从而被重新定值），因此其引用位置是$[28, 36]$。

根据对照，求得的DU链和定义相符，算法运行正确。