Decaf PA4 Report

王琛 计65 2016011360

一、本阶段工作

本阶段的修改主要都在`BasicBlock.java`和`FlowGraph.java`中，代码量并不大，且基本都可以参考框架中已有的函数。

1. DU链的求解思路

DU链是定值-引用链，如果变量A在p点被定值，该值被之后的某店s引用，那么这类引用点构成的集合即为DU链。可以通过扩充活跃变量信息来求解DU链。

原先已有的活跃变量数据流方程为：

$$LiveIn[B] = LiveUse[B] \cup (LiveOut[B] - Def[B])$$

$$LiveOut[B] = \cup(LiveIn[b])(b \in succ[B])$$

其中B是一个基本块，而$LiveOut[B]$给出了离开基本块时哪些变量在B的后继中被使用。数据流方程的求解算法课件已经给出。而求出基本块的数据流信息后，可以进一步求出每个TAC语句的数据流信息，因为每个语句也可以看成一个基本块。因为只有一个语句，$LiveOut(B) = LiveIn(succ[B])$，每个TAC的数据流求解较为容易，只需要倒序遍历每个块，根据语句类型修改前一句的$LiveOut$集合。

既然$LiveOut[B]$给出的是哪些变量在B的后继中被使用，那么当B是单条定值语句时，就可以通过简单的修改求出DU链。在原来的$LiveOut$基础上，不仅记录哪些变量会被使用，而且记录下它们在哪一行被使用。这样，我们就可以通过遍历该定值语句修改过的$LiveOut$，找出这条语句中被定值变量接下来被引用的位置。这些位置的集合就是DU链。

因此，总体思路为：

- 将$LiveIn, LiveUse, LiveOut, Def$集合元素修改为(s, A)的形式，其中A为变量，s是某一点的标号。
- 求解每个基本块的数据流方程
- 求解每条TAC语句的数据流方程
- 对每条定值语句，通过其$LiveOut$求出被定值变量的DU链

2. 具体实现

`BasicBlock.java`：

- 添加$liveInDU, liveOutDU, liveUseDU, defDU$，类型均为`public Set<Pair>`，表示修改定义后的各个集合。
- 增加`computeDefAndLiveUseDU`函数，求解重新定义后的`Def`和`LiveUse`集合，基本方法和修改前的相同。但是有两点区别，一是在增加元素时获得该语句的位置信息；二是为每个`Temp`

类增加一个`lastDefBB`，表示该变量上次被定义的基本块，如果某条语句为定值语句，定值变量的`lastDefBB`则修改为本基本块。而在使用某个变量时，只有在上次定义不是在本基本块中才加入`liveOut`集合。

- 添加`analyzeLivenessDU`函数，类似`analyzeLiveness`函数，倒序遍历整个基本块，求解每条TAC语句的数据流信息（为TAC类添加`liveOutDU`变量）。
- 添加`isDefTac`，用于判断某条TAC语句是否为定值语句
- 添加`computeDUChain`，计算这个基本块中每个定值语句的DU链。遍历整个基本块，使用`isDefTac`判断是否为定值语句。如果为定值语句，则被定值变量为op0，再遍历该tac的`liveOut`集合，如果该元素的变量等于op0，则将其对应的行号加入op0的DU链集合中。

`FlowGraph.java`：

- 添加`analyzeDUChain`函数，算法基本和`analyzeLiveness`相同，。区别在于使用的是新定义的`liveInDU`等集合，用于计算新定义的数据流信息。另外，由于`DefDU`的定义为(s,A)的集合，s不是B中的点，s引用变量A的值，A再B中重新定值。但是，实际求解时，记录s的位置比较麻烦。所以，我实现时并没有记录s的位置，而是在求`liveOutDU - DefDU`时，只要判断`liveOutDU`中引用点的范围不在当前此基本块范围内，这就已经说明了引用s时不是在此基本块中。
- 在`FlowGraph`函数中，增加如下语句，用于调用前面所增加的函数。

```
analyzeDUChain();
for (BasicBlock bb : bbs) {
    bb.analyzeLivenessDU();
    bb.computeDUChain();
}
```

二、验证TAC序列和DU链信息

由于课件中的流图和程序生成的流图稍有不同，因此比较结果时按语句进行比较。例如，课件中i在定值点d1(i:=2)的DU链为{d2(j:=i+1)}，而程序中i:=2对应`16 _T3 = _T11`，DU链为{18}，其中18对应的TAC为`_T13 = (_T3 + _T12)`，即j:=i+1，因此程序结果和课件中计算结果一致。课件中j在d4(j:=j+1)的DU链为{d4(j:=j+1), d5(j:=j-4), d7(b:=j)}，而程序中j:=j+1对应29行，求得DU链为{28, 32, 36}，为上面的三条语句，因此也是符合的。经验证，其他定值点的DU链程序运算结果和课件均一致。

下面给出了t0.decaf完整的DU链信息以及修改过的数据流集合，并且给出了相应的TAC对应的decaf语句：

```
FUNCTION _Main_New :
BASIC BLOCK 0 :
Def     = [ (_T0, 1) (_T1, 3) (_T2, 4) ]
liveUse = [ ]
liveIn  = [ ]
```

```
liveOut = [ ]
1   _T0 = 4 [ 2 ]
2   parm _T0
3   _T1 =  call _Alloc [ 5 6 ]
4   _T2 = VTBL <_Main> [ 5 ]
5   *(_T1 + 0) = _T2
6   END BY RETURN, result = _T1

FUNCTION main :
Def     = [ ]
liveUse = [ ]
liveIn  = [ ]
liveOut = [ ]
BASIC BLOCK 0 :
7   call _Main.f
8   END BY RETURN, void result

FUNCTION _Main.f :
BASIC BLOCK 0 :
Def     = [ (_T3, 16) (_T4, 19) (_T5, 10) (_T6, 12) (_T7, 9) (_T8, 11)
(_T9, 14) (_T10, 13) (_T11, 15) (_T12, 17) (_T13, 18) ]
liveUse = [ ]
liveIn  = [ ]
liveOut = [ (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
9   _T7 = 0 [ 10 ]
10  _T5 = _T7 [ ]   # a = 0;
11  _T8 = 1 [ 12 ]
12  _T6 = _T8 [ ] # b = 1;
13  _T10 = 0 [ 14 ]
14  _T9 = _T10 [ 21 24 30 ] # flag = false;
15  _T11 = 2 [ 16 ]
16  _T3 = _T11 [ 18 ] # i = 2;
17  _T12 = 1 [ 18 ]
18  _T13 = (_T3 + _T12) [ 19 ]
19  _T4 = _T13 [ 28 ] # j = i + 1;
20  END BY BRANCH, goto 1
BASIC BLOCK 1 :
Def     = [ ]
liveUse = [ (_T9, 21) ]
liveIn  = [ (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
liveOut = [ (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
21  END BY BEQZ, if _T9 =
        0 : goto 7; 1 : goto 2
BASIC BLOCK 2 :
Def     = [ (_T3, 23) (_T14, 22) ]
liveUse = [ (_T9, 24) ]
liveIn  = [ (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
liveOut = [ (_T3, 35) (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
22  _T14 = 1 [ 23 ]
```

```
23  _T3 = _T14 [ 35 ] # i = 1;
24  END BY BEQZ, if _T9 =
        0 : goto 4; 1 : goto 3
BASIC BLOCK 3 :
Def     = [ ]
liveUse = [ ]
liveIn  = [ (_T3, 35) (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ] liveOut = [
(_T3, 35) (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
25  call _Main.f
26  END BY BRANCH, goto 4
BASIC BLOCK 4 :
Def     = [ (_T4, 29) (_T15, 27) (_T16, 28) ]
liveUse = [ (_T4, 28) (_T9, 30) ]
liveIn  = [ (_T3, 35) (_T4, 28) (_T9, 21) (_T9, 24) (_T9, 30) ]
liveOut = [ (_T3, 35) (_T4, 28) (_T4, 32) (_T4, 36) (_T9, 21) (_T9, 24)
(_T9, 30) ]
27  _T15 = 1 [ 28 ]
28  _T16 = (_T4 + _T15) [ 29 ]
29  _T4 = _T16 [ 28 32 36 ] #j = j + 1;
30  END BY BEQZ, if _T9 =
        0 : goto 6; 1 : goto 5
BASIC BLOCK 5 :
Def     = [ (_T4, 33) (_T17, 31) (_T18, 32) ]
liveUse = [ (_T4, 32) ]
liveIn  = [ (_T3, 35) (_T4, 32) (_T9, 21) (_T9, 24) (_T9, 30) ]
liveOut = [ (_T3, 35) (_T4, 28) (_T4, 36) (_T9, 21) (_T9, 24) (_T9, 30) ]
31  _T17 = 4 [ 32 ]
32  _T18 = (_T4 - _T17) [ 33 ] # j = j - 4;
33  _T4 = _T18 [ 28 36 ]
34  END BY BRANCH, goto 6
BASIC BLOCK 6 :
Def     = [ (_T5, 35) (_T6, 36) ]
liveUse = [ (_T3, 35) (_T4, 36) ]
liveIn  = [ (_T3, 35) (_T4, 28) (_T4, 36) (_T9, 21) (_T9, 24) (_T9, 30) ]
35  _T5 = _T3 [ ] # a = i;
36  _T6 = _T4 [ ] # b = j;
37  END BY BRANCH, goto 1
BASIC BLOCK 7 :
Def     = [ ]
liveUse = [ ]
liveIn  = [ ]
liveOut = [ ]
38  END BY RETURN, void result
```

Acknowledgement

本次实验中，刘应天同学和李文凯同学曾给我提供思路，在此感谢他们的帮助！