

# 编译原理PA4实验报告

张钰晖 计55 2015011372 yuhui-zh15@mails.tsinghua.edu.cn

## <重要说明>

由于本次实验没有采用Lecture 12的方法进行，故math.du与标准输出不一致。

根据同学的反映，框架中遇到return语句的时候这个框架并没有给next赋值，导致next的值为0，它会错误地认为下一个block是0。

```
private void gatherBasicBlocks(Tac start) {
    BasicBlock current = null;
    Tac nextStart = null;

    while (start != null && start.bbNum < 0) {
        start = start.next;
    }

    for (; start != null; start = nextStart) {
        int bbNum = start.bbNum;
        while (start != null && start.opc == Tac.Kind.MARK) {
            start = start.next;
        }

        if (start == null) {
            current = new BasicBlock();
            current.bbNum = bbNum;
            current.tacList = null;
            current.endKind = BasicBlock.EndKind.BY_RETURN;
            nextStart = null;
        } else {
            start.prev = null;
            Tac end = start;
            while (end.next != null && end.next.bbNum == start.bbNum) {
                end = end.next;
            }
            nextStart = end.next;
            current = new BasicBlock();
            current.bbNum = bbNum;
            current.tacList = start;
            switch (end.opc) {
                case RETURN:
                    current.endKind = BasicBlock.EndKind.BY_RETURN;
                    current.var = end.op0;
                    end = end.prev;
                    break;
                case BRANCH:
                    current.endKind = BasicBlock.EndKind.BY_BRANCH;
                    current.next[0] = current.next[1] = end.label.where.bbNum;
                    end = end.prev;
                    break;
            }
        }
    }
}
```

这会导致math.du的第81行错误，不应该有36。

```
5
6 FUNCTION _Math.log :
7 BASIC BLOCK 0 :
8 35  _T25 = 1 [ 36 ]
9 36  _T26 = (_T3 < _T25) [ 37 ]
10 37  END BY BEQZ, if _T26 =
11      0 : goto 2; 1 : goto 1
12 BASIC BLOCK 1 :
```

```

2 BASIC BLOCK 1 :
3 38  _T27 = 1 [ 39 ]
4 39  _T28 = - _T27 [ 40 ]
5 40  END BY RETURN, result = _T28
6 BASIC BLOCK 2 :
7 41  _T30 = 0 [ 42 ]
8 42  _T29 = _T30 [ 48 54 ]
9 43  END BY BRANCH, goto 3
10 BASIC BLOCK 3 :
11 44  _T31 = 1 [ 45 ]
12 45  _T32 = (_T3 > _T31) [ 46 ]
13 46  END BY BEQZ, if _T32 =
14      0 : goto 5; 1 : goto 4
15 BASIC BLOCK 4 :
16 47  _T33 = 1 [ 48 ]
17 48  _T34 = (_T29 + _T33) [ 49 ]
18 49  _T29 = _T34 [ 48 54 ]
19 50  _T35 = 2 [ 51 ]
20 51  _T36 = (_T3 / _T35) [ 52 ]
21 52  _T3 = _T36 [ 36 45 51 ]
22 53  END BY BRANCH, goto 3
23 BASIC BLOCK 5 :
24 54  END BY RETURN, result = _T29

```

由于笔者的算法依赖这部分框架程度较低，不需要框架此部分代码，故输出正确，没有输出36，导致和错误的答案输出不一致，特此说明，希望您理解加上此部分分数，谢谢！

## 任务描述

PA1阶段，我们完成了词法分析、语法分析，生成了抽象语法树(AST)。

PA2阶段，我们要基于PA1的抽象语法树，实现构造符号表、静态语义检查。

PA3阶段，我们在PA2的基础上，实现语法制导的中间代码翻译。

PA4阶段，我们在PA3的基础上，进行数据流分析，实现DU链求解。

## 文件说明

在本阶段，以下文件非常重要，主要需要修改以下文件。

文件名	含义	说明
dataflow/BasicBlock	基本块定义	实现DU链求解功能
dataflow/FlowGraph	控制流图定义	实现DU链求解功能

## 实验说明

DU链即定值—引用链(Definition-Use Chaining)。

假设在程序中某点 p 定义了变量 A 的值，从 p 存在一条到达 A 的某个引用点 s 的路径，且该路径上不存在 A 的其他定值点，则把所有此类引用点 s 的全体称为 A 在定值点 p 的定值-引用链，简称 DU 链。

DU链反映了定义变量被使用情况，是数据流分析环节的重要一环。

## 实验算法

本次实验没有采用Lecture 12的方法进行，故在此详细说明本算法。如果不能较好的理解这个算法，可以参考下面详细的Java代码。

对每个块的每个定义Tac语句，递归搜索定义变量被使用情况，直至遇到其被重复定义的Tac语句，这一段区间内使用的该变量的Tac语句即为DU链。

分析DU链伪代码如下

```
1 def 分析DU链：
2     for 基本块 in 流图：
3         for Tac语句 in 块Tac语句：
4             if Tac语句 为定义类型：
5                 获得 Tac语句 定义的变量，清空搜索状态，递归跟踪该变量使用情况
```

在递归的过程中，若基本块已经被搜索或者该变量被重复定义，则递归返回。在递归的过程中，需要根据块结束类型，获得其连接的所有块，进行不同形式的递归。同时递归的过程中需要对递归深度为0进行特殊处理，因为递归深度为0时，需要对当前定义语句的下一条语句开始搜索，而其他时刻都从块第一条语句开始搜索。

递归搜索伪代码如下

```

1 def 递归搜索：
2     if 基本块 已被搜索，返回
3     if 递归深度 为0，起始搜索位置为定义语句下一条语句
4     else 起始搜索位置为 基本块 第一条语句，标记块已搜索
5     for Tac语句 in 块Tac语句：
6         if Tac语句 使用了定义变量，记录该 Tac语句 位置
7         if Tac语句 重新定义了之前变量，标记重定义，退出循环
8     if 没有重定义：
9         根据返回类型，查看 基本块 返回语句是否使用了定义变量，递归搜索 基本块 所有的
    连接块
10    返回

```

获得Tac语句定义变量和判断Tac语句是否使用了定义变量较为简单，不再赘述，可以参考代码。

由于标记了每个块是否被搜索，每个块至多被搜索一次，故单条语句搜索算法复杂度为 $O(n)$ ，不会出现指数型递归的情况。总体算法复杂度上界为 $O(\text{Tac语句数量平方})$ ，事实上，由于并非所有Tac语句都为定义语句，同时由于每个块能递归的连接块远远小于程序总块数（因为一条Tac语句的递归区间一般仅在一个函数中），故实际算法复杂度应该低得多。

## 实验实现

本次实验没有按照Lecture 12的方法进行，故在此列出详细的Java代码。

由于框架已经提供好了很多方便的接口，例如Tac的id，BasicBlock的id等，故本次实验修改较为简单，主要是使用接口。

- 修改类dataflow/BasicBlock.java

修改DUChain为public变量，新增searched变量用于记录基本块是否已经被搜索。

```

1 public Map<Pair, Set<Integer>> DUChain;
2 public boolean searched;

```

- 修改类dataflow/FlowGraph.java

新增DU链分析函数analyzeDuChain，新增递归搜索DU链函数dfsSearch，新增获取定义函数getDef，新增判断TAC语句是否使用定义的变量isUse。

```

1 public FlowGraph(Functy func) {
2     ...
3     analyzeDuChain();
4 }
5
6 public void analyzeDuChain(){
7     for (BasicBlock bb : bbs) {
8         Tac taclist = bb.tacList;
9         for (Tac t = taclist; t != null; t = t.next){
10             Temp def = getDef(t);
11             if (def == null) continue;
12             Pair pair = new Pair(t.id, t.op0);

```

```

13         Set<Integer> locations = new TreeSet<Integer>();
14         for (BasicBlock blk : bbs) blk.searched = false;
15         dfsSearch(locations, t, bb, 0);
16         bb.DUChain.put(pair, locations);
17     }
18 }
19 }
20
21 private void dfsSearch(Set<Integer> locations, Tac tac, BasicBlock bb,
22 int depth) {
23     if (bb.searched) return;
24     Tac taclist;
25     if (depth == 0) { taclist = tac.next; }
26     else { taclist = bb.tacList; bb.searched = true; }
27     Temp def = getDef(tac);
28     boolean isRedef = false;
29     for (Tac t = taclist; t != null; t = t.next) {
30         if (isUse(def, t)) {
31             locations.add(t.id);
32         }
33         Temp redef = getDef(t);
34         if (def == redef) {
35             isRedef = true;
36             break;
37         }
38     }
39     if (!isRedef) {
40         switch (bb.endKind) {
41             case BY_BRANCH:
42                 dfsSearch(locations, tac, getBlock(bb.next[0]), depth +
43 1);
44                 break;
45             case BY_BEQZ:
46             case BY_BNEZ:
47                 if (bb.var == def) {
48                     locations.add(bb.endId);
49                 }
50                 dfsSearch(locations, tac, getBlock(bb.next[0]), depth +
51 1);
52                 dfsSearch(locations, tac, getBlock(bb.next[1]), depth +
53 1);
54                 break;
55             case BY_RETURN:
56                 if (bb.var == def) {
57                     locations.add(bb.endId);
58                 }
59                 break;
60         }
61     }
62 }

```

```

58 }
59
60 public boolean isUse (Temp def, Tac tac) {
61     switch (tac.opc) {
62         case ADD:
63         case SUB:
64         case MUL:
65         case DIV:
66         case MOD:
67         case LAND:
68         case LOR:
69         case GTR:
70         case GEQ:
71         case EQU:
72         case NEQ:
73         case LEQ:
74         case LES:
75             /* use op1 and op2, def op0 */
76             if (tac.op1 == def || tac.op2 == def) return true;
77             break;
78         case NEG:
79         case LNOT:
80         case ASSIGN:
81         case INDIRECT_CALL:
82         case LOAD:
83             /* use op1, def op0 */
84             if (tac.op1 == def) return true;
85             break;
86         case LOAD_VTBL:
87         case DIRECT_CALL:
88         case RETURN:
89         case LOAD_STR_CONST:
90         case LOAD_IMM4:
91             /* def op0 */
92             break;
93         case STORE:
94             /* use op0 and op1*/
95             if (tac.op0 == def || tac.op1 == def) return true;
96             break;
97         case BEQZ:
98         case BNEZ:
99         case PARM:
100             /* use op0 */
101             if (tac.op0 == def) return true;
102             break;
103         default:
104             /* BRANCH MEMO MARK PARM*/
105             break;
106     }

```

```

107     return false;
108 }
109
110 public Temp getDef (Tac tac) {
111     Temp def = null;
112     switch (tac.opc) {
113         case ADD:
114         case SUB:
115         case MUL:
116         case DIV:
117         case MOD:
118         case LAND:
119         case LOR:
120         case GTR:
121         case GEQ:
122         case EQU:
123         case NEQ:
124         case LEQ:
125         case LES:
126             /* use op1 and op2, def op0 */
127         case NEG:
128         case LNOT:
129         case ASSIGN:
130         case INDIRECT_CALL:
131         case LOAD:
132             /* use op1, def op0 */
133         case LOAD_VTBL:
134         case DIRECT_CALL:
135         case RETURN:
136         case LOAD_STR_CONST:
137         case LOAD_IMM4:
138             /* def op0 */
139             def = tac.op0;
140             break;
141         case STORE:
142             /* use op0 and op1*/
143         case BEQZ:
144         case BNEZ:
145         case PARM:
146             /* use op0 */
147             break;
148         default:
149             /* BRANCH MEMO MARK PARM*/
150             break;
151     }
152     return def;
153 }

```

## 思考题

### 1. 说明本阶段的工作，尤其是在现有框架下实现DU链的求解。

解答：前面已经充分通过了文字+伪代码+Java代码描述，如果您依旧没有理解，可以邮件或微信联系我。

### 2. 以 TestCases/S4/t0.decaf(对应于讲义第 12 讲 2.2 节图 4)为例，分析输出的 TAC 序列与 DU 链信息，并验证它与讲义中 2.4.2 节给出的结果是一致的。

t0.decaf:

```
1  class Main {
2      static void main() {
3          f();
4      }
5
6      static void f() {
7          int i;
8          int j;
9          int a;
10         int b;
11         a = 0;
12         b = 1;
13
14         bool flag;
15         flag = false;
16
17         i = 2;
18         j = i + 1;
19
20         while (flag) {
21             i = 1;
22             if (flag)
23                 f();
24
25             j = j + 1;
26             if (flag)
27                 j = j - 4;
28             a = i;
29             b = j;
30         }
31     }
32 }
```

t0.du:

```
1  FUNCTION _Main_New :
2  BASIC BLOCK 0 :
```



```

3  1  _T0 = 4 [ 2 ]
4  2  parm _T0
5  3  _T1 = call _Alloc [ 5 6 ]
6  4  _T2 = VTBL <_Main> [ 5 ]
7  5  *(_T1 + 0) = _T2
8  6  END BY RETURN, result = _T1
9
10 FUNCTION main :
11 BASIC BLOCK 0 :
12 7  call _Main.f
13 8  END BY RETURN, void result
14
15 FUNCTION _Main.f :
16 BASIC BLOCK 0 :
17 9  _T7 = 0 [ 10 ]
18 10 _T5 = _T7 [ ]
19 11 _T8 = 1 [ 12 ]
20 12 _T6 = _T8 [ ]
21 13 _T10 = 0 [ 14 ]
22 14 _T9 = _T10 [ 21 24 30 ]
23 15 _T11 = 2 [ 16 ]
24 16 _T3 = _T11 [ 18 ]
25 17 _T12 = 1 [ 18 ]
26 18 _T13 = (_T3 + _T12) [ 19 ]
27 19 _T4 = _T13 [ 28 ]
28 20 END BY BRANCH, goto 1
29 BASIC BLOCK 1 :
30 21 END BY BEQZ, if _T9 =
31      0 : goto 7; 1 : goto 2
32 BASIC BLOCK 2 :
33 22 _T14 = 1 [ 23 ]
34 23 _T3 = _T14 [ 35 ]
35 24 END BY BEQZ, if _T9 =
36      0 : goto 4; 1 : goto 3
37 BASIC BLOCK 3 :
38 25 call _Main.f
39 26 END BY BRANCH, goto 4
40 BASIC BLOCK 4 :
41 27 _T15 = 1 [ 28 ]
42 28 _T16 = (_T4 + _T15) [ 29 ]
43 29 _T4 = _T16 [ 28 32 36 ]
44 30 END BY BEQZ, if _T9 =
45      0 : goto 6; 1 : goto 5
46 BASIC BLOCK 5 :
47 31 _T17 = 4 [ 32 ]
48 32 _T18 = (_T4 - _T17) [ 33 ]
49 33 _T4 = _T18 [ 28 36 ]
50 34 END BY BRANCH, goto 6
51 BASIC BLOCK 6 :

```

```

52 35  _T5 = _T3 [ ]
53 36  _T6 = _T4 [ ]
54 37  END BY BRANCH, goto 1
55 BASIC BLOCK 7 :
56 38  END BY RETURN, void result

```

讲义图：

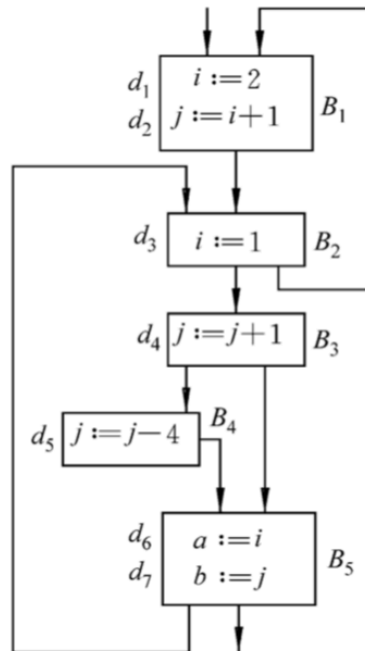


图 4 某个流图

讲义结果：

- 1 i 在定值点 d1 的 DU 链为 {d2},
- 2 j 在定值点 d2 的 DU 链为 {d4},
- 3 i 在定值点 d3 的 DU 链为 {d6},
- 4 j 在定值点 d4 的 DU 链为 {d4, d5, d7},
- 5 j 在定值点 d5 的 DU 链为 {d4, d7}

分析验证：

由输出结果可以很容易找到Main.main和Main.f，Main.f后半部分与讲义一致，可以看出Main.f被分成了8个基本块，同时可以看出T3是i，T4是j，T7是a，T8是b。

由以下验证结果可知其与讲义中 2.4.2 节给出的结果是一致的。

- 1 以下可验证 i 在定值点 d1 的 DU 链为 {d2}
- 2 16 \_T3 = \_T11 [ 18 ] \*\*
- 3 17 \_T12 = 1 [ 18 ]
- 4 18 \_T13 = (\_T3 + \_T12) [ 19 ]
- 5 以下可验证 j 在定值点 d2 的 DU 链为 {d4}
- 6 18 \_T13 = (\_T3 + \_T12) [ 19 ]
- 7 19 \_T4 = \_T13 [ 28 ] \*\*

```

8  27  _T15 = 1 [ 28 ]
9  28  _T16 = (_T4 + _T15) [ 29 ]
10 以下可验证 i 在定值点 d3 的 DU 链为 {d6}
11 22  _T14 = 1 [ 23 ]
12 23  _T3 = _T14 [ 35 ] **
13 35  _T5 = _T3 [ ]
14 以下可验证 j 在定值点 d4 的 DU 链为 {d4,d5,d7}
15 27  _T15 = 1 [ 28 ]
16 28  _T16 = (_T4 + _T15) [ 29 ]
17 29  _T4 = _T16 [ 28 32 36 ] **
18 31  _T17 = 4 [ 32 ]
19 32  _T18 = (_T4 - _T17) [ 33 ]
20 36  _T6 = _T4 [ ]
21 以下可验证 j 在定值点 d5 的 DU 链为 {d4,d7}
22 32  _T18 = (_T4 - _T17) [ 33 ]
23 33  _T4 = _T18 [ 28 36 ] **
24 27  _T15 = 1 [ 28 ]
25 28  _T16 = (_T4 + _T15) [ 29 ]
26 36  _T6 = _T4 [ ]

```

## 技巧心得

本次作业难度相对PA3有所缓和，通过以下方法可以加速编程。

### 1. 仔细阅读代码框架

由于框架已经提供好了很多方便的接口，例如Tac的id，BasicBlock的id等，故本次实验修改较为简单，主要是使用接口。如果不理解这些接口自己实现的话，将花掉大量时间，代码量也急剧扩大。

### 2. 仔细阅读测试样例及正确输出

当充分理解测试样例和正确输出后，不仅能加深巩固对DU链的理解，更是加速编程，调出bug的必要保证。

## 总结

本次实验PA4相对实验PA3难度略有下降，主要可能是因为框架已经提供好了很多方便的接口，并且一个明显的特点就是这次实现代码量并不算很大，或许是因为笔者的算法和课件略有不同。但实现时但必须非常充分理解整个框架的结构。在实现PA4过程中，通过以上两个技巧，加速了编程，在实现的过程中，充分的锻炼了笔者的编程能力，对DU链的相关概念理解有了质的提高，笔者在实践之中真正感受到了编译的神奇之处。