

# PA4实验报告

王焱 计73 2017050024

## 一、实验概述

本阶段是基于数据流分析的中间代码优化，基本任务是实现死代码消除。

## 二、实现方法

### 1.寻找死代码并作出标记

实验框架已有LivenessAnalyzer类，其中在analyzeLivenessForEachLocIn函数中已有后向遍历求解每一个location的livein，liveout的过程，死代码消除的思路就是在这个过程添加判断，如果写的变量不在liveout中，即后面没有用到，就给这条语句打上unused的标记。另外，如果写的变量为空，则这条TAC代码不能删掉，有可能为参数的声明语句等。

```
private void analyzeLivenessForEachLocIn(BasicBlock<I> bb) {
    var liveOut = new TreeSet<>(bb.liveOut);
    var it = bb.backwardIterator();
    while (it.hasNext()) {
        var loc = it.next();
        loc.liveOut = new TreeSet<>(liveOut);
        if(!loc.instr.getWritten().isEmpty() &&
liveOut.removeAll(loc.instr.getWritten())==false){
            loc.instr.unused = true;
        }
        liveOut.addAll(loc.instr.getRead());
        loc.liveIn = new TreeSet<>(liveOut);
    }
}
```

### 2.删除死代码

在Optimizer中的transform函数中，对TacProg类型的input进行遍历，为每一个函数使用CFGBuilder构造cfg,再用一个analyzer实例对其进行数据流分析。然后遍历分析后的指令序列，判断如果是unused的tac语句就删除。但是这里需要注意对call语句的特殊判断，形如 `int a = function(in param)` 的语句，如果变量 a 在之后未使用，依旧需要进行函数调用，只是不需要将返回值赋给a，所以修改函数的dst为空。对于多层的死代码，我采取的方法就是暴力进行多次优化。

```
public TacProg transform(TacProg input) {
    var analyzer = new LivenessAnalyzer<>();
    for(int optimizenum=0;optimizenum<10;optimizenum++){
        for(var func: input.funcs){
            var builder = new CFGBuilder<>();
            var cfg = builder.buildFrom(new ArrayList<>(func.getInstrSeq()));
            analyzer.accept(cfg);

            var size = func.getInstrSeq().size();
            var list = new ArrayList<>(func.getInstrSeq());
```

```

        for(int i=size-1;i>=0;i--){
            var inst =list.get(i);
            if(inst.unused){
                if(inst instanceof IndirectCall){
                    var call = (IndirectCall)inst;
                    call.dst = Optional.empty();
                }else if(inst instanceof DirectCall){
                    var call = (DirectCall)inst;
                    call.dst = Optional.empty();
                }else{
                    func.getInstrSeq().remove(i);
                }
            }
        }
    }
}
return input;
}

```

### 三、成果展示

```

test.decaf x  basic-basic.decaf x  basic-basic.tac x
1 // Computing Fibonacci Sequence
2
3 class Main {
4     static string test(string x){
5         return x;
6     }
7     // the main entry
8     static void main() {
9         int x = 2;
10        int xsz = 9999999;
11
12        string dsz = "hhhhhhhhhhhhhh";
13        string sz = test(dsz);
14
15        int y = 3;
16        bool a = true;
17        string s = "Hello THU";
18        Print(x,"\n");
19        Print(y,"\n");
20        Print(x+y,"\n");
21        Print(x*y,"\n");
22        Print(a,"\n");
23        Print(s,"\n");
24    }
25 }
26
27
test.tac x
10    *(_T1 + 0) = _T2
11    return _T1
12
13 FUNCTION<Main.test>:
14    return _T0
15
16 main:
17    _T1 = 2
18    _T0 = _T1
19    _T5 = "hhhhhhhhhhhhhh"
20    _T4 = _T5
21    parm _T4
22    call FUNCTION<Main.test>
23    _T9 = 3
24    _T8 = _T9
25    _T11 = 1
26    _T10 = _T11
27    _T13 = "Hello THU"
28    _T12 = _T13
29    parm _T0
30    call _PrintInt
31    _T14 = "\n"
32    parm _T14
33    call _PrintString
34    parm _T8
35    call _PrintInt
36    _T15 = "\n"
37    parm _T15
38    call _PrintString
39    _T16 = (_T0 + _T8)
40    parm _T16
41    call _PrintInt
42    _T17 = "\n"
43    parm _T17
44    call _PrintString
45    _T18 = (_T0 * _T8)
46    parm _T18
47    call _PrintInt

```

### 五、小结

在考完试的第一天下午，为了拯救凉凉的编译，我还是决定写一下PA4。这次PA感觉由于没有涉及到新特性简单了许多，凭借着复习时学到的知识，成了基本的死代码消除。实验中遇到的许多都是java语法特性方面的问题，比如list迭代器遍历时remove的问题等等，感觉复习了一周多已经不会写代码了orz