学院：软件工程学院　专业：软件工程　姓名：王梓阳　学号：21317002　日期：2024 年 4 月 15 日

实验三 死代码检测

一、实验目标

1. 为 Java 实现死代码检测算法。

二、实验理论（20%）

1. 对于控制流不可达代码，可以使用 CFG 检测出来，即：遍历 CFG 并标记可达语句，遍历结束未标记的即为不可达的代码。
2. 对于分支不可达代码，Java 中 if 语句和 switch 语句皆会导致此类不可达代码的出现。当这两类语句的条件值是常数时，if 语句两个分支中的其中一个分支是不可达的。Switch 语句不符合条件的分支则是可能不可达。检测此类不可达代码时，需要预先进行常量传播分析，由此判断条件值是否为常量，再通过遍历 CFG 的方式标记代码。
3. 无用赋值：一个局部变量在一条语句中被赋值后却再没有被后面的语句读取即为无用赋值。检测无用赋值需要先进行活跃变量分析 ，对一个赋值语句如果复制号左侧变量是无用变量则这个语句标记为无用赋值。

三、实验实现（20%）

1. 对于 LiveVariableAnalysis.java 和 ConstantPropagation.java 等前面实验已经实现的内容，本实验中直接使用即可，不再重写，这里对其实现具体细节也不再赘述。
2. 在本次实验中，不是从死代码出发，而是记录所有可达的语句，没有被记录的语句都是不可达的。这里不能直接遍历控制流图中的 IR，而是使用队列记录所有即将被访问的语句，使用 bfs 进行遍历，对于每条语句根据其不同的类型进行不同处理。除了队列之外，还需要一个集合来判断某条语句是否被访问过，避免重复访问，防止死循环。具体每条代码的解释见下方代码注释。

```
@Override
public Set<Stmt> analyze(IR ir) {
    // obtain CFG
    CFG<Stmt> cfg = ir.getResult(CFGBuilder.ID);
    // obtain result of constant propagation
    DataflowResult<Stmt, CPFact> constants =
            ir.getResult(ConstantPropagation.ID);
    // obtain result of live variable analysis
```

```
        DataflowResult<Stmt, SetFact<Var>> liveVars =
                ir.getResult(LiveVariableAnalysis.ID);
        // keep statements (dead code) sorted in the resulting set
        Set<Stmt> deadCode = new TreeSet<>(Comparator.comparing(Stmt::g
etIndex));
        // TODO - finish me
        ArrayDeque<Stmt> st = new ArrayDeque<>();//使用队列
        Set<Stmt> re = new HashSet<>();
        Set<Stmt> reached = new HashSet<>();
        st.addLast(cfg.getEntry());// 第一个访问结点是方法的入口
        re.add(cfg.getExit());//方法的入口和出口肯定是可达的
        re.add(cfg.getEntry());
        while(!st.isEmpty()) {//bfs
            Stmt s = st.pollFirst();
            reached.add(s);//弹出结点记录为被访问
            if(s instanceof AssignStmt assignStmt) {// 无用赋值语句处理
                SetFact<Var> le=liveVars.getResult(assignStmt);
                LValue lValue = assignStmt.getLValue();
                RValue rValue = assignStmt.getRValue();
                boolean f=true;
                if(lValue instanceof Var) {
                    if (!le.contains((Var) lValue)) {//要判断左值是否能转
化成变量类型
                        if (hasNoSideEffect(rValue)) {
                            f = false;
                        }
                    }
                }
                if (f) {
                    re.add(assignStmt);
                }
                for (Stmt succ : cfg.getSuccsOf(assignStmt)) {//把后继结
点加入队列
                    if (!reached.contains(succ))
                        st.addLast(succ);
                }
            }
            else if(s instanceof If ifStmt) {//if 语句
                CPFact result = constants.getResult(ifStmt);// 获取常量
传播结果
                ConditionExp condition = ifStmt.getCondition();
                ConditionExp.Op operator = condition.getOperator();
                Value evaluate = ConstantPropagation.evaluate(condition,
 result);
```

```
                re.add(ifStmt);// 当前 if 语句可达
                if (evaluate.isConstant()) {//// 如果 if 条件表达式是个常数，
那么只可能到达一个分支
                    if (evaluate.getConstant() == 1) {
                        for (Edge<Stmt> edge : cfg.getOutEdgesOf(ifStmt)
) {
                            if (edge.getKind() == Edge.Kind.IF_TRUE) {
                                Stmt target = edge.getTarget();
                                if (!reached.contains(target))
                                    st.add(target);
                            }
                        }
                    } else {
                        for (Edge<Stmt> edge : cfg.getOutEdgesOf(s)) {
                            if (edge.getKind() == Edge.Kind.IF_FALSE) {
                                Stmt target = edge.getTarget();
                                if (!reached.contains(target))
                                    st.add(target);
                            }
                        }
                    }
                }
                else { // 如果 if 条件表达式不是个常数，那么两条分支都可能，按
照控制流执行
                    for (Stmt succ : cfg.getSuccsOf(s)) {
                        if (!reached.contains(succ))
                            st.addLast(succ);
                    }
                }
            }else if (s instanceof SwitchStmt switchStmt){// switch 语句
处理
                Var var = switchStmt.getVar();
                CPFact result = constants.getResult(switchStmt);
                re.add(switchStmt);// 当前 switch 语句可达
                if (result.get(var).isConstant()) {
                    int constant = result.get(var).getConstant();
                    boolean match = false;
                    for (Edge<Stmt> edge : cfg.getOutEdgesOf(switchStmt)
) {
                        if (edge.getKind() == Edge.Kind.SWITCH_CASE) {
                            int caseValue = edge.getCaseValue();
                            if (caseValue == constant) {
                                match = true;
```

```
                                if (!reached.contains(edge.getTarget())
)
                                    st.addLast(edge.getTarget());
                            }
                        }
                    }
                    if (!match) {// 如果不匹配，执行 default
                        Stmt defaultTarget = switchStmt.getDefaultTarge
t();
                        if (!reached.contains(defaultTarget))
                            st.addLast(defaultTarget);
                    }
                } else {// 如果 switch 表达式不是常数，每个 case 都可能执行，
按照控制流执行
                    for (Stmt succ : cfg.getSuccsOf(switchStmt)) {
                        if (!reached.contains(succ))
                            st.addLast(succ);
                    }
                }
            }else {// 其他类型语句，都可达，按照控制流执行
                re.add(s);
                for (Stmt succ : cfg.getSuccsOf(s)) {
                    if (!reached.contains(succ))
                        st.addLast(succ);
                }
            }
        }
        for (Stmt stmt : ir.getStmts()) {// 遍历当前方法的所有 IR，如果不可
达，那么就是死代码
            if (!re.contains(stmt)) {
                deadCode.add(stmt);
            }
        }
        // Your task is to recognize dead code in ir and add it to dead
Code
        return deadCode;
    }
```

四、实验结果（50%）

1. （测试结果截图）（25%）

ControlFlowUnreachable.java

```
------------------- <ControlFlowUnreachable: void <init>()> (deadcode) -------------------

------------------- <ControlFlowUnreachable: int foo()> (deadcode) -------------------
[2@L6] invokestatic <ControlFlowUnreachable: void dead()>();

------------------- <ControlFlowUnreachable: void dead()> (deadcode) -------------------
```

DeadAssignment.java

```
------------------- <DeadAssignment: void <init>()> (deadcode) -------------------

------------------- <DeadAssignment: void deadAssign()> (deadcode) -------------------
[2@L5] y = x + %intconst0;
[6@L8] a = x;

------------------- <DeadAssignment: void use(int)> (deadcode) -------------------
```

Loops.java

```
-------------------- <Loops: void <init>()> (deadcode) --------------------

-------------------- <Loops: void deadLoop()> (deadcode) --------------------
[5@L7] goto 9;
[9@L7] nop;
[10@L10] invokevirtual %this.<Loops: void dead()>();
[11@L10] return;

-------------------- <Loops: void dead()> (deadcode) --------------------

-------------------- <Loops: void use(int)> (deadcode) --------------------
```

UnreachableIfBranch.java

```
------------------- <UnreachableIfBranch: void <init>()> (deadcode) -------------------

------------------- <UnreachableIfBranch: int branch()> (deadcode) -------------------
[3@L7] goto 7;
[7@L7] nop;
[8@L10] z = 200;
```

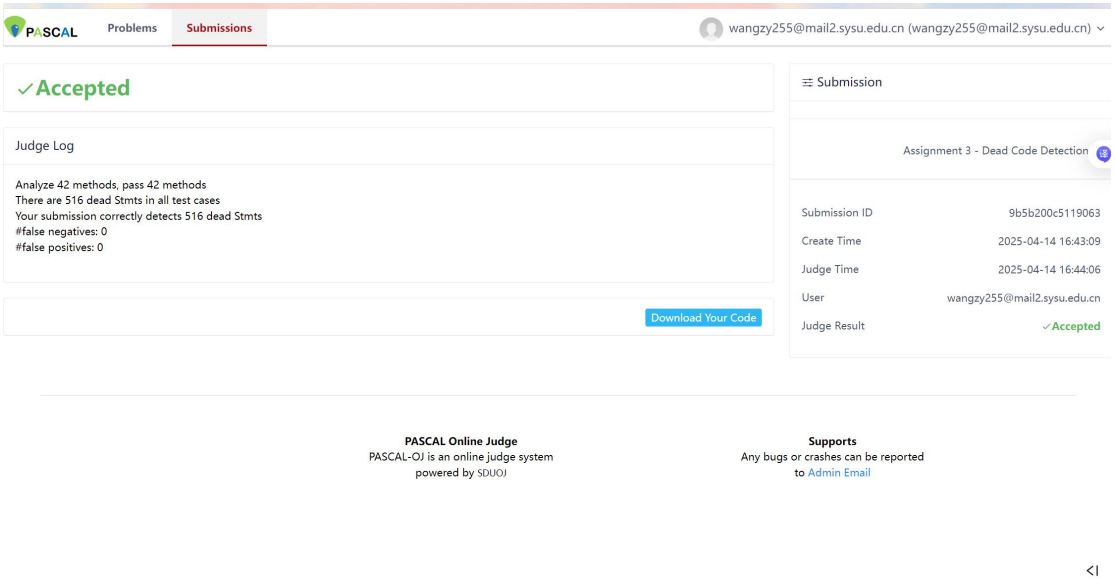
UnreachableSwitchBranch.java

```
------------------- <UnreachableSwitchBranch: void <init>()> (deadcode) -------------------

------------------- <UnreachableSwitchBranch: void lookupSwitch()> (deadcode) ---------------
[4@L6] nop;
[5@L8] %intconst1 = 2;
[6@L8] invokevirtual %this.<UnreachableSwitchBranch: void use(int)>(%intconst1);
[7@L9] goto 24;
[8@L9] nop;
[9@L11] %intconst2 = 4;
[10@L11] invokevirtual %this.<UnreachableSwitchBranch: void use(int)>(%intconst2);
[11@L12] goto 24;
[16@L15] nop;
[17@L17] %intconst4 = 666;
[18@L17] invokevirtual %this.<UnreachableSwitchBranch: void use(int)>(%intconst4);
[19@L18] goto 24;
[20@L6] goto 24;

------------------- <UnreachableSwitchBranch: void use(int)> (deadcode) -------------------
```

2. （测试平台的结果截图）**（25%）**



五、实验总结（10%）

本次实验实现了死代码检测算法，在实现过程中，我学会了如何使用广度优先搜索遍历控制流图，合理记录可达语句，最终识别出不可达代码、无用赋值及因常量判断而永远不会执行的分支代码。通过本实验，我加深了对程序静态分析的理解，掌握了分析算法的实际实现过程。该实验不仅提升了我的分析能力和代码实现能力，也为后续学习打下了坚实基础。