**import** java.io.File;

**import** java.util.Iterator;

**import** soot.Body;

**import** soot.Local;

**import** soot.Scene;

**import** soot.SootClass;

**import** soot.SootMethod;

**import** soot.Unit;

**import** soot.ValueBox;

**import** soot.jimple.Stmt;

**import** soot.options.Options;

**import** soot.toolkits.graph.BriefUnitGraph;

**import** soot.toolkits.graph.DirectedGraph;

**import** soot.toolkits.graph.UnitGraph;

**import** soot.toolkits.scalar.ArraySparseSet;

**import** soot.toolkits.scalar.BackwardFlowAnalysis;

**import** soot.toolkits.scalar.FlowSet;

**public** **class** Project {

**public** **static** **void** main(String[] args) {

Options.*v*()

.set\_soot\_classpath(System.*getProperty*("user.dir") + File.***separator*** + "bin" + File.***pathSeparator***

+ System.*getenv*("JAVA\_HOME") + File.***separator*** + "jre/lib/rt.jar" + File.***pathSeparator***

+ System.*getProperty*("user.dir") + File.***separator*** + "soot-4.0.0-jar-with-dependencies.jar"

+ File.***pathSeparator*** + System.*getenv*("JAVA\_HOME") + File.***separator*** + "jre/lib/jce.jar");

Options.*v*().set\_whole\_program(**true**);

Options.*v*().set\_ignore\_resolving\_levels(**true**);

Scene.*v*().loadNecessaryClasses();

SootClass cl = Scene.*v*().loadClassAndSupport("Project");

cl.setApplicationClass();

SootMethod method = cl.getMethodByName("testFunction");

Body body = method.retrieveActiveBody();

UnitGraph graph = **new** BriefUnitGraph(body);

Iterator<Unit> iterator = graph.iterator();

BackwardFlowAnalysis<Unit, FlowSet<Local>> flowAnalysis = **new** MyBackwardFlowAnalysis<>(graph);

**while** (iterator.hasNext()) {

Unit unit = iterator.next();

Stmt stmt = (Stmt) unit;

String entry = "";

String exit = "";

FlowSet<Local> set = flowAnalysis.getFlowBefore(unit);

**for** (Local local : set) {

entry += local.getName() + " ";

}

set = flowAnalysis.getFlowAfter(unit);

**for** (Local local : set) {

exit += local.getName() + " ";

}

System.***out***.println("Statement: " + stmt);

System.***out***.println("Entry value: " + entry);

System.***out***.println("Exit value: " + exit);

System.***out***.println();

}

}

**public** **void** testFunction() {

String result = "unknown";

**int** a = 34;

**int** b = 15;

**double** d = 4.5;

**if** (a < 25) {

result = "value 1";

} **else** **if** (a >= 25 && a < 45) {

result = "value 2";

b /= 2;

} **else** {

result = "value 3";

d \*= 1.2;

**if** (d < 2.5) {

result = "value 4";

} **else** **if** (d >= 2.5 && d < 3.1) {

result = "value 5";

} **else** {

result = "value 6";

}

}

**while** (d > 0) {

b += 2;

d += 0.3;

}

result += d;

result += b;

System.***out***.println(result);

}

**static** **class** MyBackwardFlowAnalysis<T, E> **extends** BackwardFlowAnalysis<Unit, FlowSet<Local>> {

**public** MyBackwardFlowAnalysis(DirectedGraph<Unit> g) {

**super**(g);

doAnalysis();

}

@Override

**protected** FlowSet<Local> newInitialFlow() {

**return** **new** ArraySparseSet<Local>();

}

@Override

**protected** **void** merge(FlowSet<Local> arg0, FlowSet<Local> arg1, FlowSet<Local> arg2) {

arg0.union(arg1, arg2);

}

@Override

**protected** **void** copy(FlowSet<Local> arg0, FlowSet<Local> arg1) {

arg0.copy(arg1);

}

@Override

**protected** **void** flowThrough(FlowSet<Local> arg0, Unit arg1, FlowSet<Local> arg2) {

FlowSet<Local> flowSet = (FlowSet<Local>) arg0, outputSet = (FlowSet<Local>) arg2;

Unit unit = (Unit) arg1;

FlowSet<Local> finishedSet = **new** ArraySparseSet<>();

Iterator<ValueBox> valueIterator = unit.getDefBoxes().iterator();

**while** (valueIterator.hasNext()) {

ValueBox valueBox = (ValueBox) valueIterator.next();

**if** (valueBox.getValue() **instanceof** Local) {

Iterator<Local> localIterator = flowSet.iterator();

**while** (localIterator.hasNext()) {

Local local = (Local) localIterator.next();

**if** (local.equivTo(valueBox.getValue())) {

finishedSet.add((Local) valueBox.getValue());

}

}

}

}

flowSet.difference(finishedSet, outputSet);

Iterator<ValueBox> valueBoxIterator = unit.getUseBoxes().iterator();

**while** (valueBoxIterator.hasNext()) {

ValueBox vb = (ValueBox) valueBoxIterator.next();

**if** (vb.getValue() **instanceof** Local) {

outputSet.add((Local) vb.getValue());

}

}

}

}

}