Reflection Test Using Violist

Sean

Department of Information Science
Jinan University

April 27, 2019

Experiment Introduction

We use Violist, the string analyzer, to analyze some test cases about Java reflection. We conduct this analysis on 12 test cases.

- 3 for basic program structure: Sequential, Branch and Loop
- 3 for basic structure with inter-procedural
- 3 for Infinite Test and Recursive Test
- 3 for realistic test case

Table: Experiment Environment

Analyzer	JDK	Description
Violist_old	openjdk1.8	new Violist can only analyze Android

Experiment Result

Table: Experiment Result

Test Case	SSI	FSAI
Sequential	✓	✓
Branch	✓	✓
Loop	*N*	*N*
Sequential Inter	✓	✓
Branch Inter	✓	✓
Loop Inter	✓	✓
Infinite Test1	⊙	✓
Infinite Test2	×	×
Recursive	⊙	⊙
External classname	✓	✓
directions classname	*N*	*N*
OWASP	✓	✓

" ⊙ " denotes partly correct, *N* denotes No Result.

Notice

Strangely, the analysis result is unstable. We execute in many times. It happens that sometimes the test case which can be analyzed in last run, cannot be analyzed this run, vice versa. The detail analysis result will be described next.

Basic Structure Test – Sequential

Test Code:

```
1    public void test(){
2        String className = "java.lang.Object1";
3        // analysis point 1: "java.lang.Object1"
4        Logger.reportString(className,
             "Sequential1");
5        doReflect(className);
6        className = "java.lang.Object2";
7        // analysis point 2: "java.lang.Object2"
8        Logger.reportString(className,
             "Sequential12");
9        doReflect(className);
10   }
```

SSI Result: {"java.lang.Object1"}, {"java.lang.Object2"} ✓
FASI Result: same as SSI ✓

Basic Structure Test – Branch

Test Code:

```
1     public void test(){
2         String className = null;
3         if(Math.random() > 0.5) {
4             className = "java.lang.Object";
5         } else {
6             className = "java.lang.Class";
7         }
8         // analysis point
9         Logger.reportString(className, "Branch");
10        doReflect(className);
11    }
```

SSI Result: {"java.lang.Object","java.lang.Class"} ✓
FASI Result: same as SSI ✓

Basic Structure Test – Loop

Test Code:

```
1     public void test(){
2         String className;
3         String[] names = {"java.lang.Loop0",
              "java.lang.Loop1", "java.lang.Loop2",
              "java.lang.Loop3"};
4         for(int i=0; i<4; i++) {
5             className = names[i];
6             // analysis point
7             Logger.reportString(className, "Loop");
8             doReflect(className);
9         }
10    }
```

SSI Result:

FASI Result:

Sequential with Inter-Procedural

```
1     public String getClassName1() {
2         return "java.lang.Object1";}
3     public String getClassName2() {
4         return "java.lang.Object2";}
5     public void test(){
6         String className = getClassName1();
7         Logger.reportString(className,
              "SequentialInter1");
8         doReflect(className);
9         className = getClassName2();
10        Logger.reportString(className,
              "Sequential1Inter2");
11        doReflect(className);}
```

SSI Result: {"java.lang.Object1"}, {"java.lang.Object2"} ✓
FASI Result: same as SSI ✓

Branch with Inter-Procedural

```
1     public String getClassName() {
2         String className = null;
3         if(Math.random() > 0.5) {
4             className = "java.lang.Object";
5         } else {
6             className = "java.lang.Class";
7         }
8         return className;}
9     public void test(){
10        String className = getClassName();
11        Logger.reportString(className,
              "BranchInter");
12        doReflect(className);
13    }
```

SSI Result: {"java.lang.Object","java.lang.Class"} ✓
FASI Result: same as SSI ✓

Loop with Inter-Procedural

```
1    public String getClassName(int idx) {
2        String[] names = {"java.lang.Loop0",
             "java.lang.Loop1", "java.lang.Loop2",
             "java.lang.Loop3"};
3        return names[idx];
4    }
5    public void test(){
6        String className = null;
7        for(int i=0; i<4; i++) {
8            className = getClassName(i);
9            Logger.reportString(className,
                 "LoopInter");
10           doReflect(className);
11       }
12   }
```

SSI Result: {"null", "..Loop0", "..Loop1", "..Loop2", "..Loop3"} ✓
FASI Result: same as SSI ✓

Infinite Test1

```
1    public void test(){
2        String className = "";
3        for(int i=1; i < Integer.MAX_VALUE; i++) {
4            className += 'x';
5        }
6        Logger.reportString(className,
             "InfiniteTest");
7        doReflect(className);
8    }
```

SSI Result: {"", "x", "xx", "xxx"} ⊙
FASI Result: Recognize Regular Expression: x* ✓

Infinite Test2

```
1    public void test(){
2        String className = "";
3        for(int i=1; i < Integer.MAX_VALUE; i++) {
4            className += ('x'+i);
5        }
6        Logger.reportString(className,
             "InfiniteTest");
7        doReflect(className);
8    }
```

SSI Result: {} ×
FASI Result: {}×

Recursive Test

```
1     public String recur(int deep) {
2         if(deep > 0) {
3             return recur(--deep) + "recur ";
4         } else {
5             return "recur ";
6         }
7     }
8     public void test(){
9         String className = recur(5);
10        Logger.reportString(className,
              "RecursiveTest");
11        doReflect(className);
12    }
```

SSI Result: {"recur "} ⊙
FASI Result: same as SSI ⊙

Realistic Test 1: class name from external

```
1     public String getClassName() {
2         return getClassNameFromExternal();
3     }
4     private String getClassNameFromExternal() {
5         return "java.lang.Object";
6     }
7     public void test(){
8         String className = getClassName();
9         Logger.reportString(className,
              "ReflectNameFromExternal");
10        doReflect(className);
11    }
```

SSI Result: {"java.lang.Object"} ✓
FASI Result: same as SSI ✓

Realistic Test 2: class name from file directions

```
1    public String getClassName() {
2        String[] dirs = new String[]{"java", "lang",
             "Object"};
3        String className = null;
4        for (String dir : dirs) {
5            className += dir;
6            className += ".";
7        }
8        return className.substring(0,
             className.length()-1);}
9    public void test(){
10       String className = getClassName();
11       Logger.reportString(className,
             "ReflectNameFromDirs");
12       doReflect(className);}
```

SSI Result:

FASI Result:

Realistic Test 3: OWASP

```
1     public void doAction(String className){
2         Class c = Class.forName(className);
3         // expect result: "Command1", "CommandEvil"
4         Logger.reportString(className, "OWASP");
5         Worker w = (Worker)c.newInstance();
6         w.doAction();
7     }
8     public static void main(String[] args){
9         OWASP demo = new OWASP();
10        demo.doAction("Command1");
11        demo.doAction("CommandEvil");
12    }
```

SSI Result: {"Command1","CommandEvil"} ✓
FASI Result: same as SSI ✓