Homework 3
Posted Monday February 14, Due Monday February 28
50 points

Now that you are (more) familiar with Soot and the Class analysis framework, you will build a more complex class analysis, XTA. Add a new package `analysis.XTA` and add your XTA implementation in `public class XTAAnalysis extends Analysis` in this package. You may add analogous drivers to the ones in RTA to test locally. Follow directory structure as Submitty pulls your `analysis/XTA/XTAAnalysis.java` to test.

Below is a rehash of the XTA constraints we discussed in class.

1. Allocation:
   1: for each new A in m s.t. m ∈ ReachableMethods do
   2: \{A\} ⊆ S_m
   3: end for

2. Virtual call:
   1: for each x = y.n(z) in m s.t. m ∈ ReachableMethods do
   2: for each C in S_m ∩ SubTypes(StaticType(y)) do
   3: n'(this, p, ret) = resolve(C, n) // adds target to ReachableMethods
   4: \{n'\} ⊆ ReachableMethods
   5: \{C\} ⊆ S_{n'} // adds receiver class to S_{n'}
   6: S_m ∩ SubTypes(StaticType(p)) ⊆ S_{n'} // adds to S_{n'} due to arguments
   7: S_{n'} ∩ SubTypes(StaticType(ret)) ⊆ S_m // adds to S_{m} from S_{n'} due to return
   8: end for
   9: end for

We have simplified the presentation showing each method having exactly one formal parameter p. Of course, a method may have 0 or more parameters and you will need to handle the general case. Note that the class analysis framework passes all actual arguments to the analysis, including ones that are of primitive type. You will have to do some extra work to filter out parameters of primitive type. Soot API methods `getParameterType` and `getReturnType` in `SootMethod` may be of use.

3. Field Read:
   1: for each x = y.f in m s.t. m ∈ ReachableMethods do
   2: S_f ⊆ S_m
   3: end for

4. Field Write:
   1: for each x.f = y in m s.t. m ∈ ReachableMethods do
   2: S_m ∩ SubTypes(StaticType(f)) ⊆ S_f
   3: end for

In addition, you must handle direct calls, static fields and array reads/writes.

Direct calls to static methods are straightforward. Direct calls to instance methods (i.e., these are methods such as constructors that have a receiver `this`) use the following constraint to pass the type of the receiver to the callee n:

\[ S_m \cap SubTypes(StaticType(this)) \subseteq S_n \]
Static field reads, local = static_field, and writes, static_field = local, are abstracted as assignStmt in the class analysis framework, where respectively, the right-hand-side or left-hand-side node is of kind STATIC_FIELD.

Make sure you handle arrays. Ignoring arrays renders the XTA analysis unsound. Consider

```java
void m(X[] a) {
    X x = a[0];
    x.n();
}
```

and note that in general, the array argument may have been written anywhere in the program.

Finally, as with RTA, display your result in showResult. Specifically, display all reachable methods $m$ in alphabetical order with all classes in $S_m$ in alphabetical order. For example, the expected output for p2 is the following

Reachable methods:

```java
<A: int add(A)>
=== A
=== B
<A: void <init>()>
=== A
=== B
<A: void m()>
=== A
=== B
<A: void main(java.lang.String[])
<A: void sm()>
=== A
=== B
<B: void <init>()>
=== B
```

When you are done, push into your hw02 repository and click Grade my Repository in Submitty. Submitty pulls your analysis/XTA/XTAAnalysis.java to test. Make sure you commit/push the entire XTA directory.