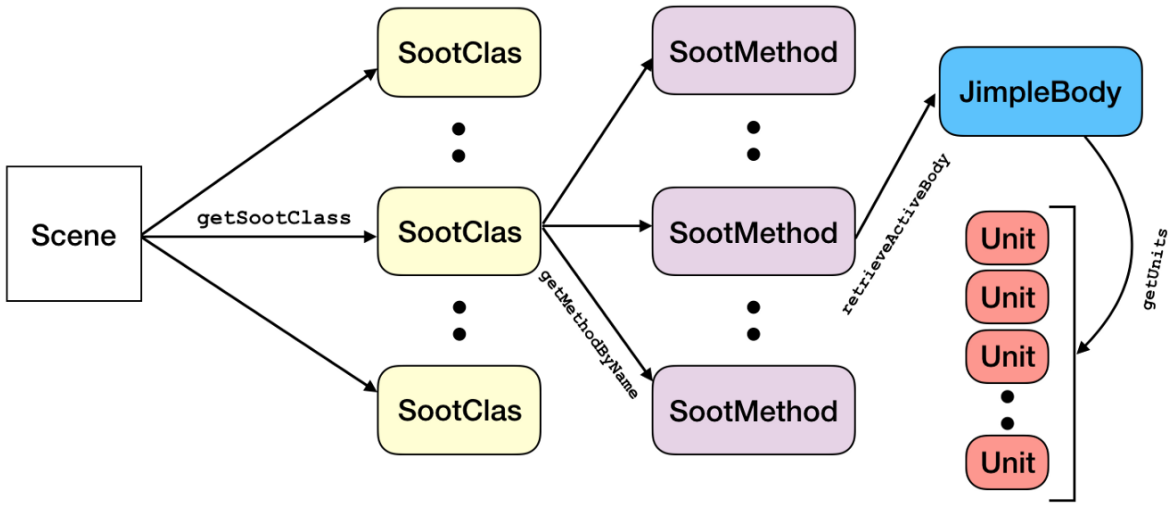
WICHITA STATE UNIVERSITY

CS 891
Detection of Warnings in Android Plugins using Soot

Soot FrameWork:

Soot is a Java optimization framework. It provides four intermediate representations for analyzing and transforming Java bytecode a streamlined representation of bytecode which is simple to manipulate. Jimple a typed 3-address intermediate representation suitable for optimization.



Static Program Analysis :

Static program analysis, in its simplest form, is a black box that inputs a program (code) and outputs some of the properties of the program. For example, let's say we are interested in finding all the branch statements in a method and call this analysis *BranchDetectorAnalysis*. To illustrate this example, I am going to use a trivial program.

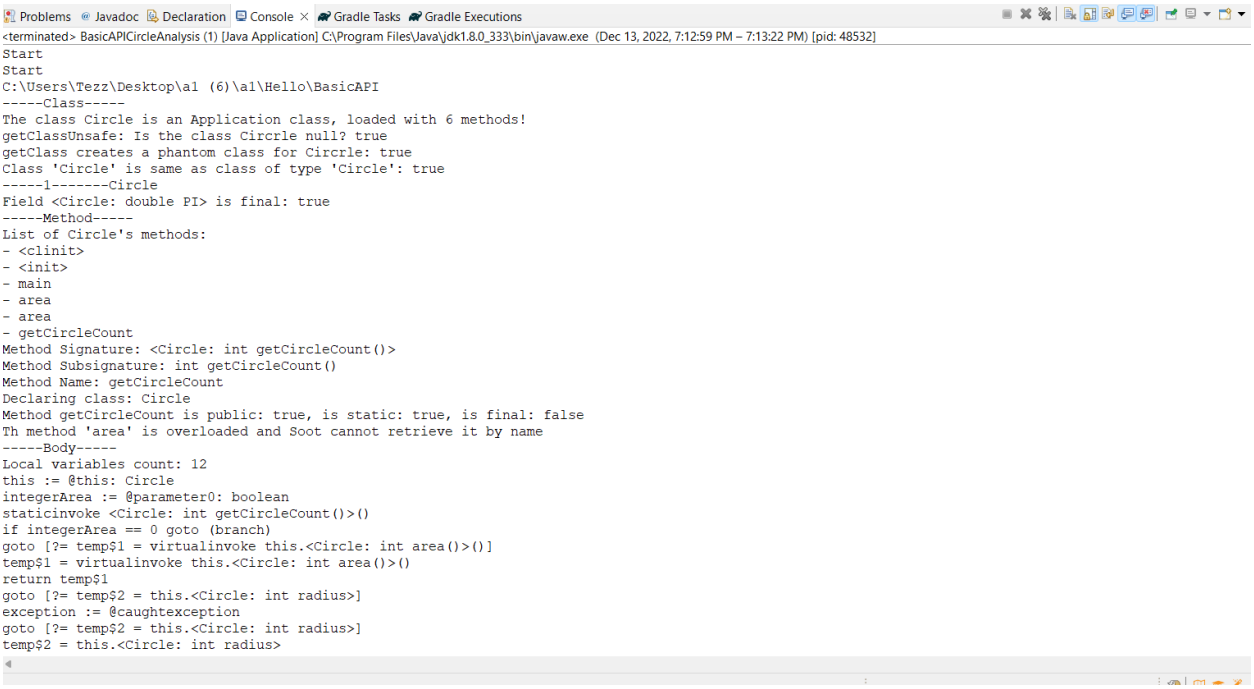
Psuedo Code:

```
public
class
FizzBuzz
{

                public void printFizzBuzz(int k){
                    if (k%15==0)
                        System.out.println("FizzBuzz");
                    else if (k%5==0)
                        System.out.println("Buzz");
                    else if (k%3==0)
                        System.out.println("Fizz");
                    else
                        System.out.println(k);
                }

                public static void main(String[] args){
                   FizzBuzz fb = new FizzBuzz();
                    for (int i=1; i<=100; i++)
                        fb.printFizzBuzz(i);
                }
            }
```

As a result, the input of *BranchDetectorAnalysis* is a Java method (`printFizzBuzz`) and the output will be the statements that branch the execution of the code (lines 4, 6, and 8). Note that line 10 is not considered a branch statement since its condition is implicitly determined in line 8.

PS: For this Analysis we have a sample code and used to explain here, attached with this report

Analysis Pattern:

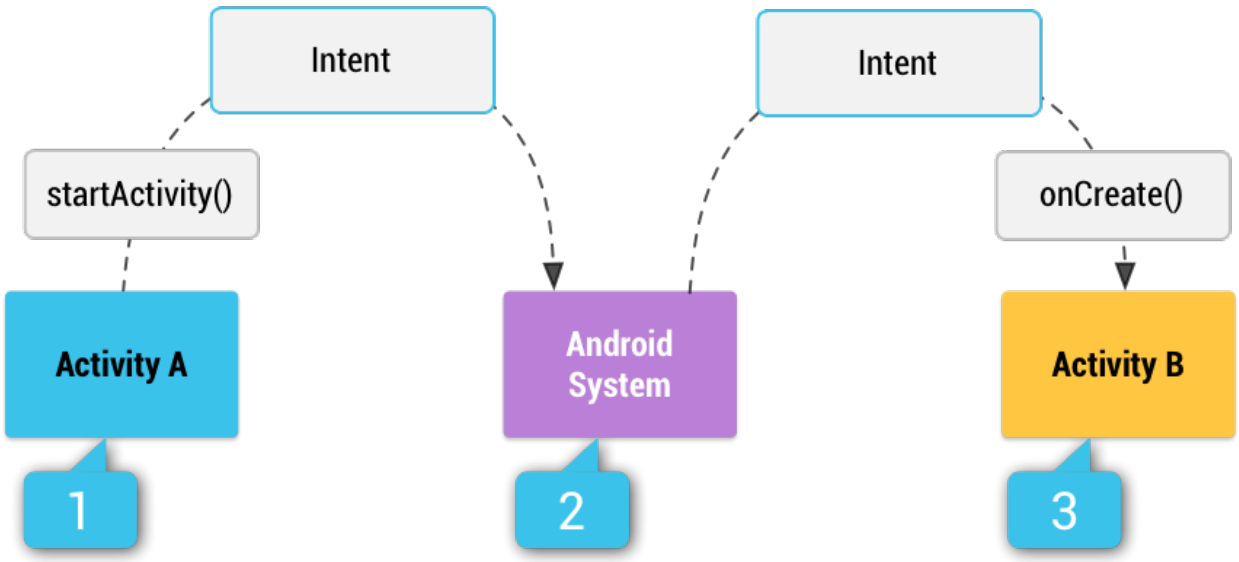


```
Problems  Javadoc  Declaration  Console  Gradle Tasks  Gradle Executions
<terminated> BasicAPICircleAnalysis (1) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (Dec 13, 2022, 7:12:59 PM – 7:13:22 PM) [pid: 48532]
Start
Start
C:\Users\Tezz\Desktop\a1 (6)\a1\Hello\BasicAPI
-----Class-----
The class Circle is an Application class, loaded with 6 methods!
getClassUnsafe: Is the class Circrle null? true
getClass creates a phantom class for Circrle: true
Class 'Circle' is same as class of type 'Circle': true
-----1-------Circle
Field <Circle: double PI> is final: true
-----Method-----
List of Circle's methods:
- <clinit>
- <init>
- main
- area
- area
- getCircleCount
Method Signature: <Circle: int getCircleCount()>
Method Subsignature: int getCircleCount()
Method Name: getCircleCount
Declaring class: Circle
Method getCircleCount is public: true, is static: true, is final: false
Th method 'area' is overloaded and Soot cannot retrieve it by name
-----Body-----
Local variables count: 12
this := @this: Circle
integerArea := @parameter0: boolean
staticinvoke <Circle: int getCircleCount()>()
if integerArea == 0 goto (branch)
goto [?= temp$1 = virtualinvoke this.<Circle: int area()>()]
temp$1 = virtualinvoke this.<Circle: int area()>()
return temp$1
goto [?= temp$2 = this.<Circle: int radius>]
exception := @caughtexception
goto [?= temp$2 = this.<Circle: int radius>]
temp$2 = this.<Circle: int radius>
```

```
<terminated> BasicAPICircleAnalysis (1) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (Dec 13, 2022, 7:12:59 PM – 7:13:22 PM) [pid: 48532]
temp$2 = this.<Circle: int radius>
temp$3 = (double) temp$2
temp$4 = 1.0 * temp$3
temp$5 = this.<Circle: int radius>
temp$6 = (double) temp$5
temp$7 = temp$4 * temp$6
temp$8 = temp$7 * 3.1415
temp$9 = (int) temp$8
return temp$9
(1): this := @this: Circle
(2): integerArea := @parameter0: boolean
(3): staticinvoke <Circle: int getCircleCount()>()
    This statement is the first non-identity statement!
    StaticInvokeExpr 'staticinvoke <Circle: int getCircleCount()>()' from class 'int'
(4): if integerArea == 0 goto (branch)
    (Before change) if condition 'integerArea == 0' is true goes to stmt 'goto [?= temp$2 = this.<Circle: int radius>]'
    (After change) if condition 'integerArea == 0' is true goes to stmt 'goto [?= temp$1 = virtualinvoke this.<Circle: int area()>()]'
(5): goto [?= temp$1 = virtualinvoke this.<Circle: int area()>()]
(6): temp$1 = virtualinvoke this.<Circle: int area()>()
    VirtualInvokeExpr 'virtualinvoke this.<Circle: int area()>()' from local 'this' with type Circle
    This statement invokes 'int area()' method
(7): return temp$1
(8): goto [?= temp$2 = this.<Circle: int radius>]
(9): exception := @caughtexception
(10): goto [?= temp$2 = this.<Circle: int radius>]
(11): temp$2 = this.<Circle: int radius>
    Field <Circle: int radius> is used through FieldRef 'this.<Circle: int radius>'. The base local of FieldRef has type 'Circle'
(12): temp$3 = (double) temp$2
(13): temp$4 = 1.0 * temp$3
(14): temp$5 = this.<Circle: int radius>
    Field <Circle: int radius> is used through FieldRef 'this.<Circle: int radius>'. The base local of FieldRef has type 'Circle'
(15): temp$6 = (double) temp$5
(16): temp$7 = temp$4 * temp$6
(17): temp$8 = temp$7 * 3.1415
(18): temp$9 = (int) temp$8
(19): return temp$9
Trap :
```

```
(4): if integerArea == 0 goto (branch)
    (Before change) if condition 'integerArea == 0' is true goes to stmt 'goto [?= temp$2 = this.<Circle: int radius>]'
    (After change) if condition 'integerArea == 0' is true goes to stmt 'goto [?= temp$1 = virtualinvoke this.<Circle: int area()>()]'
(5): goto [?= temp$1 = virtualinvoke this.<Circle: int area()>()]
(6): temp$1 = virtualinvoke this.<Circle: int area()>()
    VirtualInvokeExpr 'virtualinvoke this.<Circle: int area()>()' from local 'this' with type Circle
    This statement invokes 'int area()' method
(7): return temp$1
(8): goto [?= temp$2 = this.<Circle: int radius>]
(9): exception := @caughtexception
(10): goto [?= temp$2 = this.<Circle: int radius>]
(11): temp$2 = this.<Circle: int radius>
    Field <Circle: int radius> is used through FieldRef 'this.<Circle: int radius>'. The base local of FieldRef has type 'Circle'
(12): temp$3 = (double) temp$2
(13): temp$4 = 1.0 * temp$3
(14): temp$5 = this.<Circle: int radius>
    Field <Circle: int radius> is used through FieldRef 'this.<Circle: int radius>'. The base local of FieldRef has type 'Circle'
(15): temp$6 = (double) temp$5
(16): temp$7 = temp$4 * temp$6
(17): temp$8 = temp$7 * 3.1415
(18): temp$9 = (int) temp$8
(19): return temp$9
Trap :
begin  : staticinvoke <Circle: int getCircleCount()>()
end    : return temp$1
handler: exception := @caughtexception
Trap :
begin  : goto [?= temp$2 = this.<Circle: int radius>]
end    : exception := @caughtexception
handler: exception := @caughtexception
Body is validated! No inconsistency found.
-----CallGraph-----
Method '<Circle: int area(boolean)>' invokes method '<Circle: int area()>' through stmt 'temp$1 = virtualinvoke this.<Circle: int area()>()
Method '<Circle: int area(boolean)>' invokes method '<java.lang.Object: void <clinit>()>' through stmt 'staticinvoke <Circle: int getCircleCount()>()
Method '<Circle: int area(boolean)>' invokes method '<Circle: int getCircleCount()>' through stmt 'staticinvoke <Circle: int getCircleCount()>()
Method '<Circle: int area(boolean)>' invokes method '<Circle: void <clinit>()>' through stmt 'staticinvoke <Circle: int getCircleCount()>()
```

Intents and Intent Filters:



Intent types:

There are two types of intents:

- **Explicit intents** specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Implicit Intent :

An implicit intent specifies an action to be called by any app on the device that can perform the action. Using implicit intents is useful when you can't perform an action in your app, but most likely other apps can, and you want the user to choose which app to use.

For example, if you have content that you want users to share with others, create an intent with an ACTION_SEND action and add extras to specify the content to share. Calling startActivity() with this intent allows the user to choose which app to share content with.

Psuedo Code:

```
// Create the text message with a string.
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Try to invoke the intent.
try {
    startActivity(sendIntent);
} catch (ActivityNotFoundException e) {
    // Define what your app should do if no activity can handle the intent.
}
```

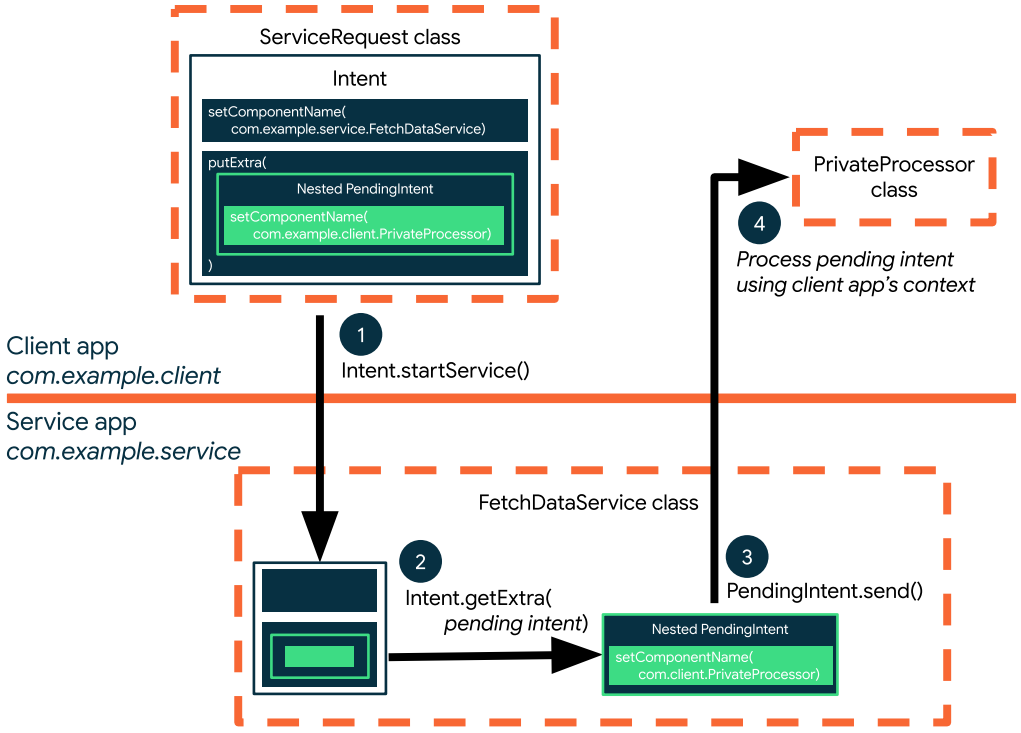
Explicit intent:

An explicit intent is an intent used to launch a specific app component Specific activities or services within the app. To create an explicit intent, define the component name of the intent object. All other intent properties are optional.

The Intent(Context, Class) constructor provides a Class object for your app context and your component. So this intent explicitly starts the DownloadService

Psuedo Code:

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

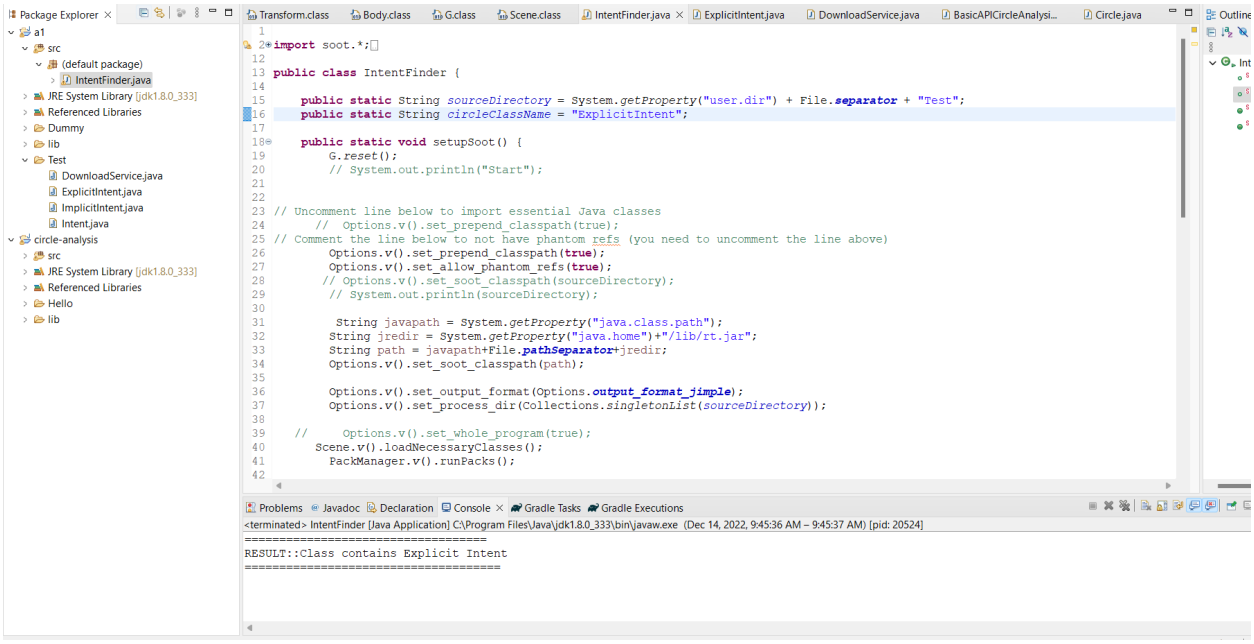


```
1
2  import soot.*;
12
13 public class IntentFinder {
14
15     public static String sourceDirectory = System.getProperty("user.dir") + File.separator + "Test";
16     public static String circleClassName = "ImplicitIntent";
17
18     public static void setupSoot() {
19         G.reset();
20         // System.out.println("Start");
21
22
23 // Uncomment line below to import essential Java classes
24       //  Options.v().set_prepend_classpath(true);
25 // Comment the line below to not have phantom refs (you need to uncomment the line above)
26         Options.v().set_prepend_classpath(true);
27         Options.v().set_allow_phantom_refs(true);
28        // Options.v().set_soot_classpath(sourceDirectory);
29         // System.out.println(sourceDirectory);
30
31          String javapath = System.getProperty("java.class.path");
32         String jredir = System.getProperty("java.home")+"/lib/rt.jar";
33         String path = javapath+File.pathSeparator+jredir;
34         Options.v().set_soot_classpath(path);
35
36         Options.v().set_output_format(Options.output_format_jimple);
37         Options.v().set_process_dir(Collections.singletonList(sourceDirectory));
38
39    //     Options.v().set_whole_program(true);
40       Scene.v().loadNecessaryClasses();
41         PackManager.v().runPacks();
42
```

Problems Javadoc Declaration Console Gradle Tasks Gradle Executions

```
<terminated> IntentFinder [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (Dec 13, 2022, 10:48:50 PM – 10:48:52 PM) [pid: 11792]
====================================
RESULT::Class contains Implicit Intent
====================================
```

Soot provided several Intermediate Representation (IR) of Java programs in order to make the static analysis more convenient. Jimple is default Intermediate Representation in Soot. Jimple is a statement based, typed and 3-addressed intermediate representation.In the source code, we are setting the soot class path with rt.jar (contains all of the compiled class files for the base Java Runtime environment, as well as the bootstrap classes) and java class path.



Classes are the pivotal data structures in analyzing code in Soot. We can access the classes using Scene and in the code we are passing the test class name and wrong class name to getSootClass. We can access the class type using getType() method. We are getting all methods information in reportSootMethodInfo and printing the method name, method signature, method sub signature, class information.
In reportSootFieldInfo, we are displaying the fields information of class.

Conclusion :
Intents are matched against intent filters not only to discover a target component to activate, but also to discover something about the set of components on the device. For example, the Home app populates the app launcher by finding all the activities with intent filters that specify the action and category. A match is only successful if the actions and categories in the Intent match against the filter.

References:

[1] Ali-Reza Adl-Tabatabai, Michal Cierniak, Guei-Yuan Lueh, Vishesh M. Parikh, and James M. Stichnoth. Fast and effective code generation in a just-in-time Java compiler. ACM SIGPLAN Notices, 33(5):280–290, May 1998.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. Compilers Principles, Techniques and Tools. Addison-Wesley, 1986.

[3] Geoff A. Cohen, Jeffrey S. Chase, and David L. Kaminsky. Automatic program transformation with JOIE. In Proceedings of the USENIX 1998 Annual Technical Conference, pages 167–178, Berkeley, USA, June 15–19 1998. USENIX Association.

[4] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark K. Wegman, and F. Kenneth Zadeck. An efficient method of computing static single assignment form. In 16th Annual ACM Symposium on Principles of Programming Languages, pages 25–35, 1989.

[5] Jeffrey Dean, Greg DeFouw, David Grove, Vassily Litvinov, and Craig Chambers. VORTEX: An optimizing compiler for object-oriented languages. In Proceedings OOPSLA ’96 Conference on Object-Oriented Programming Systems, Languages, and Applications, volume 31 of ACM SIGPLAN Notices, pages 83–100. ACM, October 1996.