

IDEs-Friendly Interprocedural Analyser

Ilya Nozhkin
Saint Petersburg State University
St. Petersburg, Russia
nozhkin.ii@gmail.com

Semyon Grigorev
Saint Petersburg State University
St. Petersburg, Russia
semen.grigorev@jetbrains.com

ABSTRACT

TODO: ABSTRACT

ACM Reference Format:

Ilya Nozhkin and Semyon Grigorev. 2019. IDEs-Friendly Interprocedural Analyser. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 1 page. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Static analyses are important part of modern development tools. Generally speaking, they take care of verifying correctness of some program's behaviour freeing a programmer from this duty. By used scope of program, an analyze can be classified as intraprocedural or interprocedural, i.e. as those which make decisions based on only one current procedure or based on the whole program respectively. And interprocedural analyses, in theory, can be more precise due to amount of available information.

```
class A {
    [Source] int Source;
}

class B {
    [Filter] int Filter(int d);
}

class C {
    [Sink] void Sink(int d);
}

class D {
    void Process(A a) {
        int d = Read(a);
        int f = B.Filter(d);
        Consume(d);
        AnotherConsume(f);
    }

    int Read(A a) {
        return a.Source;
    }

    void Consume(int d) {
        C c = new C();
        c.Sink(d);
    }

    void AnotherConsume(int d) { ... }
}
```

Figure 1: Sample code

For example, let's consider the listing 1. It is known that method *Sink* is vulnerable to invalid arguments. The method *Filter* validates its argument and ensures that if some data is returned then it is definitely valid. And the field *Source* is

known as potentially tainted. So, the problem is to find out whether data from *Source* reaches *Sink* bypassing *Filter*.

Generally speaking, this problem is a special case of label-flow analysis, so there are several approaches to solving such problems. One of them is CFL-reachability. (TODO: ADV: performance, DISADV: expressive power, non-obvious structure, CITATIONS) Another is abstract interpretation. (TODO: vice versa) We propose to combine these two approaches to achieve acceptable performance and expressive power. I.e. the program is translated into a graph as it is in CFL-r, but constraints that specify what paths is needed to be accepted are set by pushdown automaton which transition relation simulates the semantics of original program. Let's take a closer look at these two components that define an analysis in conjunction.

2 ANALYSIS DEFINITION

2.1 Graph extraction

The graph that is explored during analysis is an aggregate of control-flow graphs of each method. The one that corresponds to our example is shown at figure (TODO: REF2)

TODO: REF2: GRAPH

It has the following structure. Each node represents a position in a program. Each edge contains an operation that reflects the one from the source code.

2.2 PDA construction

3 SOLUTION

4 EVALUATION

TODO: EVALUATION

5 CONCLUSION

TODO: CONCLUSION

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>