Graspan

A Big Data System for Big Code Analysis

Kai Wang, Aftab Hussain, Zhiqiang Zuo, Guoqing Xu, Ardalan Amiri Sani
Department of Computer Science, University of California, Irvine

The Work at a Glance

We address the **scalability** problem of **inter-procedural static analysis** for **bug detection** in **big code**.
We built **Graspan**, a graph processing system that helped us uncover **85** new Null pointer bugs in **Linux 4.4.0.**

Problem

Why Interprocedural Analysis?

NULL BUG CHECKER [1]

```
void function1() {
  int * ptr1;
  if (<condition>) {
    ptr1 = fn_explct_ret_null();
  }
  else {
    ptr1 = fnA();
  }
  if (ptr1!=NULL) {   Absence of this check is a bug.
    int b = *ptr1;
  }
}
```

Pattern Checker can detect that **ptr1** can be **NULL**.

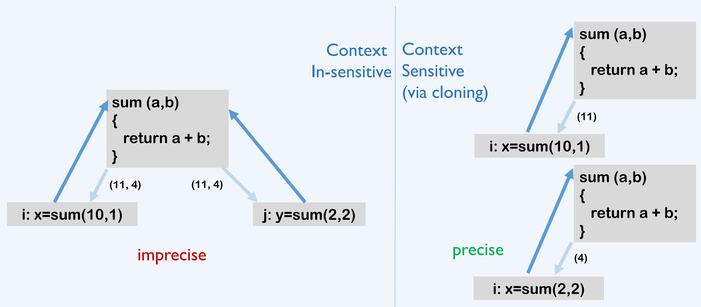
```
void function1() {
  int * ptr1;
  if (<condition>) {
    ptr1 = function2();
  }
  else {
    ptr1 = fnA();
  }
  int b = *ptr1;
}
```

```
int * function2() {
  int * ptr2 = NULL;
  int * ptr3;
  if (<condition>) {
    ptr3 = ptr2;
  }
  else {
    ptr3 = fnB();
  }
  return ptr3;
}
```

Pattern Checker cannot detect that **ptr1** can be **NULL**, we need interprocedural analysis, e.g. dataflow analysis.

The Scalability Problem

The scalability problem of interprocedural analysis stems from producing distinct solutions for different **calling contexts**.



No. of calling contexts grows **exponentially** with program size.
A moderate-sized program can have **10^{14} distinct contexts**. [2]

Non-Parallelizability

Most existing techniques frequently involve decision making based on information they discover dynamically.

Implementation Difficulty

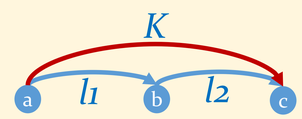
To address scalability, practitioners approximate their analyses, however, implementing them is complicated.
In Sridharan`s and Bodik`s [4] work, **more than 75%** of their entire code was dedicated to tuning the analysis.

Graspan

Graphs and Program Analysis?

Thomas Reps et al. [3] showed most interprocedural analyses, like **dataflow analysis** and **pointer analysis**, can be transformed to a **graph reachability problem**.

Queries can be solved using dynamic transitive computation.



$K \rightarrow l_1\ l_2$

Quick Note:

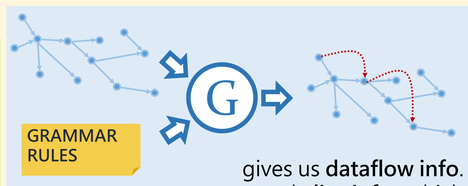
Dataflow Analysis: Helps us find how data flows in a program across statements

Pointer Analysis: Helps us find aliases, i.e., variables that point to the same memory location.

Our Approach

Find **more bugs** on **big code** In **your machine**
Performance-wise
Scalable
Parallelizable
Easy to Implement

Benefits



gives us **dataflow info.** and **alias info.**, which can be fed into checkers to find more bugs.

How it works

Challenges

Duplicate Edges
Overburdens memory. How to terminate edge addition process?

Repartitioning
How do we partition the graph, and then after computation, how do we repartition oversized partitions?

Our Design

Preprocessing
Partition the graphs.

Edge-Pair Centric Computation
Analyze pairs of edges, and perform DTC computation.

Post-Processing
Repartition oversized partitions according to a threshold.

Future Ambitions

Extend system support for path-sensitive analysis, constraint-based analyses by encoding constraints into edge values.

Results

Our Analysis

We implemented **fully context-sensitive** dataflow analysis and pointer analysis on these programs:

Program	Version	#LOC
Linux	4.4.0-rc5	16M
PostgreSQL	8.3.9	700K
Apache httpd	2.2.18	300K

Research Q&As

Can Interprocedural Analysis improve checkers?
85 new bugs in Linux 4.4.0.

Is Graspan Efficient and Scalable?
Computations took ~2 to less than 12 hrs, largest graph generated had 1.1B edges.

Graspan implementation v/s old implementation?
Use an API to provide a grammar.

Graspan v/s existing graph systems?
GraphChi crashed in 133 secs with 65M edges added.

Example Bug (missed by original checker)

```
void*probe_kthread_data(
    task_struct *task){
 void *data = NULL;
 probe_kernel_read(&data);

 /*data will be dereferenced
 after return.*/
 return data;
}

long probe_kernel_read
 (void *dst){
 if(...)
   return -EFAULT;
 return
 __probe_kernel_read(dst);
}
```

References

[1] Nicolas Palix, Gaël Thomas, Suman Saha, Christophe Calvès, Julia Lawall, and Gilles Muller, **Faults in linux: ten years later,** ASPLOS '11
[2] John Whaley and Monica S. Lam, **Cloning-based context-sensitive pointer alias analysis using binary decision diagrams**, PLDI '04
[3] Thomas Reps, Susan Horwitz, and Mooly Sagiv, **Precise interprocedural dataflow analysis via graph reachability**, POPL '95
[4] Manu Sridharan and Rastislav Bodík, **Refinement-based context-sensitive points-to analysis for Java,** PLDI '06

Acknowledgements

This work was done under the supervision of Guoqing Xu, University of California, Irvine. Other contributors of this work are Kai Wang, Zhiqiang Zuo, and Ardalan Amiri Sani. The work was submitted to OSDI 2016.