

# A Compositional Static Deadlock Detector for Android Code Revisions

---

Nikos Gorogiannis (Facebook London)

Joint work with James Brotherston, Paul Brunet, Max Kanovich (UCL)

# Contents

---

- Motivation & Problem Statement
- An Analysis for Deadlocks
- Adaptation to Java & Implementation
- Impact & Future Work

# Problem Statement

---

Find **deadlocks** introduced by revisions, during **code review** (in <15min), on app code in the **10s of MLoC**, running on 1000s of revisions/day.

- Deadlock analyses are whole-program.
- A deadlock involves two traces.
  - Often only one trace is affected by a revision.
- We can't afford to analyse the whole program here.

# Approach

---

- ***Partial-program*** analysis of modified files in revision.
- Compositional summarisation of each method.
  - Sequential analysis of ***lock behaviour***.
- Concurrency check:
  - What methods may run in parallel to Foo?  
Use locks acquired by Foo to find these methods.
  - Collect static information on thread identity.

# An Analysis for Deadlocks

# Abstract Language

---

$\mathcal{L}$ : set of global lock names for recursive/reentrant locks

$$C := \text{skip} \mid p() \mid \text{acq}(\ell) \mid \text{rel}(\ell) \mid C; C \\ \mid \text{if}(\ast) \text{ then } C \text{ else } C \mid \text{while}(\ast) \text{ do } C$$

**Non-deterministic** control; no recursion.

Top-level programs must be **balanced** wrt locking (**synchronized**).

**Op. semantics** via tracking lock states  $L : \mathcal{L} \rightarrow \mathbb{N}$ .

**Deadlock** = absence of transitions to a next state.

# Critical Pairs

---

$(X, y) \in \text{Crit}(C)$

where  $X$  is a set of locks and  $y$  is a lock such that  $y \notin X$

Intuitively (definition in the paper):

$C$  **acquires** lock  $y$  (which it does not hold)  
while it holds precisely the locks in  $X$

**Thm.**  $C_1 \parallel C_2$  deadlocks iff there are critical pairs  
 $(X_1, \ell_1) \in \text{Crit}(C_1)$  and  $(X_2, \ell_2) \in \text{Crit}(C_2)$  such that

$$X_1 \cap X_2 = \emptyset \text{ and } \ell_1 \in X_2 \text{ and } \ell_2 \in X_1$$

# A Static Analysis for Critical Pairs

---

Abstract States:

$$\alpha = \langle L, Z \rangle \quad \begin{array}{l} Z \subseteq 2^{\mathcal{L}} \times \mathcal{L} \text{ is a set of critical pairs} \\ L : \mathcal{L} \rightarrow \mathbb{N} \text{ is a thread-local lock state} \end{array}$$

**Prop.** For any balanced  $C$ :  $\llbracket C \rrbracket \alpha_{\perp} = \langle \emptyset, \text{Crit}(C) \rangle$

**Thm.** Checking  $P = C_1 \parallel \dots \parallel C_n$  for deadlock can be done in exponential time in  $|P|$  and is in **NP**.



# Example Deadlock

---

`a . foo(b) || b . bar(a)` deadlocks:

Critical Pairs

```
class A {  
  public synchronized void foo(B b) { b.foo(); }  
  public synchronized void bar() {}  
}
```

$\{(\emptyset, a), (\{a\}, b)\}$   
 $\{(\emptyset, a)\}$

```
class B {  
  public synchronized void bar(A a) { a.bar(); }  
  public synchronized void foo() {}  
}
```

$\{(\emptyset, b), (\{b\}, a)\}$   
 $\{(\emptyset, b)\}$

Pairs  $(\{a\}, b)$  and  $(\{b\}, a)$  satisfy the deadlock conditions:

- $\{a\} \cap \{b\} = \emptyset$
- $b \in \{b\}$  and  $a \in \{a\}$

Adaptation to Java & Implementation

# Implementation and Adaptations

---

- Implemented in ***Infer*** (open source, OCaml, ~3kLoC).
- Locks represented as ***access paths*** (this *.f.g.h*).
- ***Thread identity***: main-thread, worker, both, neither.
  - Android lifecycle, annotations, assertions.
  - + Class hierarchy + back-propagation over calls.

# Analysis applied to Code Revisions

---

```
class A {  
    public synchronized void foo(B b) { b.foo(); } (modification)  
    public synchronized void bar() {}  
}  
  
class B {  
    public synchronized void bar(A a) { a.bar(); }  
    public synchronized void foo() {}  
}
```

1.  $a . \text{foo}(b)$  is analysed; it has the pair  $(\{a : A\}, b : B)$ .
2. Since  $a . \text{foo}(b)$  takes a lock in class  $B$ , analyse all methods in  $B$  (which may run in parallel with  $a . \text{foo}(b)$ ).
3. Does any pair of  $a . \text{foo}(b)$  satisfy the deadlock conditions against any pair from methods in  $B$ ?

Impact & Future Work

# Impact

---

- Analysed **>100k of revisions** in >2 years.
- Issued **>500 reports**, with long traces.
- Fix rate is **>50%**.
- In last 100 days,
  - Infer analysis runtime on average=**~200sec**.
  - #methods/revision analysed on average=**~5k**.

# Future Work

---

- Is the problem NP-complete?
- Which adaptations admit further study?
  - Treatment of access paths.
- Can we modestly enlarge the set of dependencies?
  - Eg, by precomputing the locks used by classes.

# Thanks!

<https://fbinfer.com>

Infer

Docs Support Blog Twitter Facebook GitHub Search

## A tool to detect bugs in Java and C/C++/Objective-C code before it ships

Infer is a static analysis tool - if you give Infer some Java or C/C++/Objective-C code it produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped to users, and help prevent crashes or poor performance.

Get Started Learn More

☆ Star 12,722

### Android and Java

Infer checks for null pointer exceptions, resource leaks, annotation reachability, missing lock guards, and concurrency race conditions in Android and Java code.

### C, C++, and iOS/Objective-C

Infer checks for null pointer dereferences, memory leaks, coding conventions and unavailable API's.

### Infer in Action

```
class Infer {
  String mayReturnNull(int i) {
    if (i > 0) {
      return "Hello, Infer!";
    }
    return null;
  }
}
```

Recorded with

### Try Infer

code() Infer Java Tutorial Hello.java Pointers.java

```
1 /**
2  * The Infer "Hello World" Java example
3  *
4  * Click the "Analyze" button to run the example
5  * Learn more about Infer at http://fbinfer.com
6  */
7
8
9
```

This will display the output.

Input to your program (press Enter to send) Send

User: #anonymous (sign in to save your changes) Role: Project user codeboard.io