Scalable Static Analysis Using Facebook Infer

Atomicity Violation — Deadlock — Worst-Case Cost

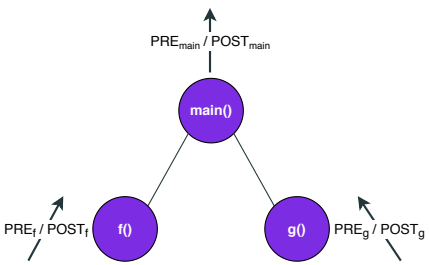
Dominik Harmim xharmi00@stud.fit.vutbr.cz
Vladimír Marcin xmarci10@stud.fit.vutbr.cz
Ondřej Pavela xpavel34@stud.fit.vutbr.cz

27th April 2019

VeriFIT – Adam Rogalewicz, Tomáš Fiedor, Tomáš Vojnar

Facebook Infer

- Open-source static analysis framework for interprocedural analyses.
  - Based on Abstract Interpretation.
  - Checks, e.g., for buffer overflows, thread-safety, null-dereferencing or memory leaks.
- Follows principles of compositionality.
  - Computes function summaries.
  - Incremental $\Longrightarrow$ highly scalable.
- Supports Java, C, C++ and Objective-C.



ATOMER: Atomicity Violations Analyser

- Atomicity violations for **sequences of functions**.
- Sequences executed **once atomically** should (probably) be executed **always atomically**.
- Targets **C/C++** programs that use **Pthreads locks**.

ATOMER: Two Phases of Analysis

1. Detection of atomic sequences.

```
void g(int *array) {
    pthread_mutex_lock(&lock);
    int i = index_of(array, 42);
    if (i >= 0) set(array, i, 3);
    pthread_mutex_unlock(&lock);
}
```

$summary_g$: **{(index_of set)}**

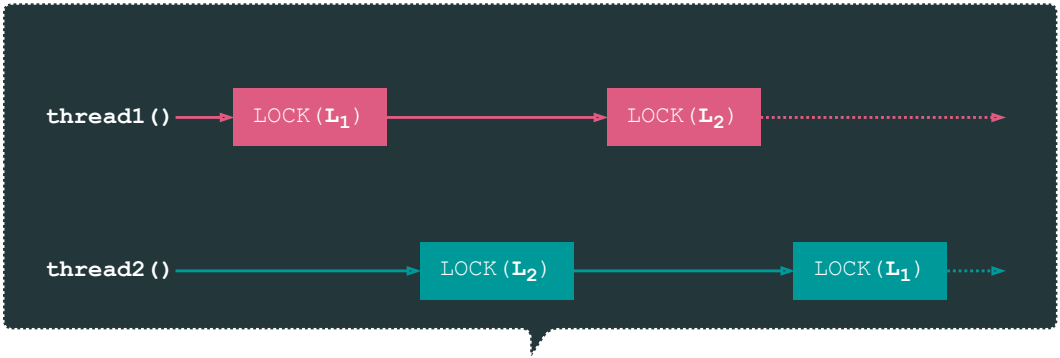
2. Detection of violations of the atomic sequences.

```
void h(int *array) {
    int i = index_of(array, 66);
    if (i >= 0) set(array, i, 0);
}
```

ATOMICITY VIOLATION!

L2D2: Low Level Deadlock Detector

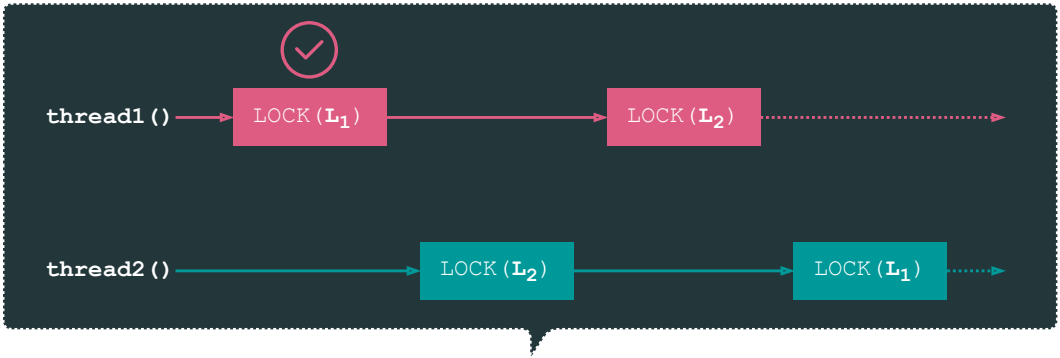
- Deadlock = one of the most common concurrency bugs.



TEXTBOOK DEADLOCK

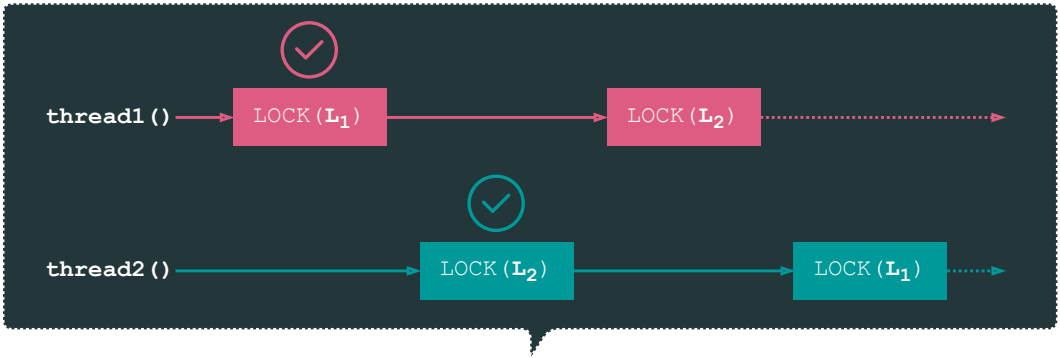
L2D2: Low Level Deadlock Detector

- Deadlock = one of the most common concurrency bugs.



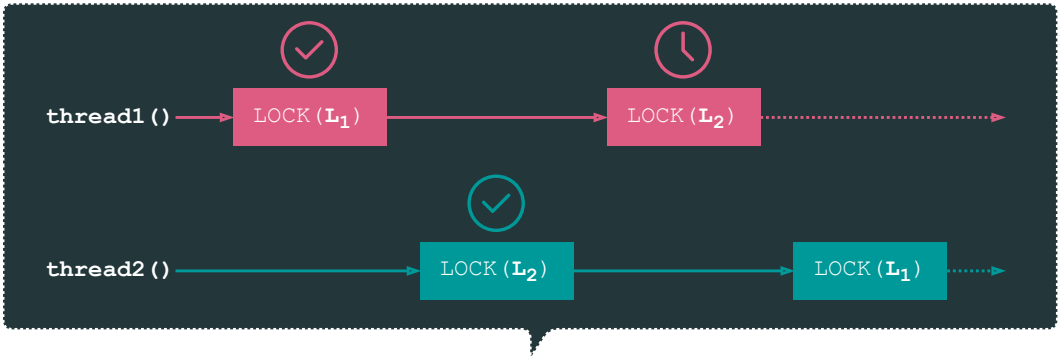
TEXTBOOK DEADLOCK

- Deadlock = one of the most common concurrency bugs.



L2D2: Low Level Deadlock Detector

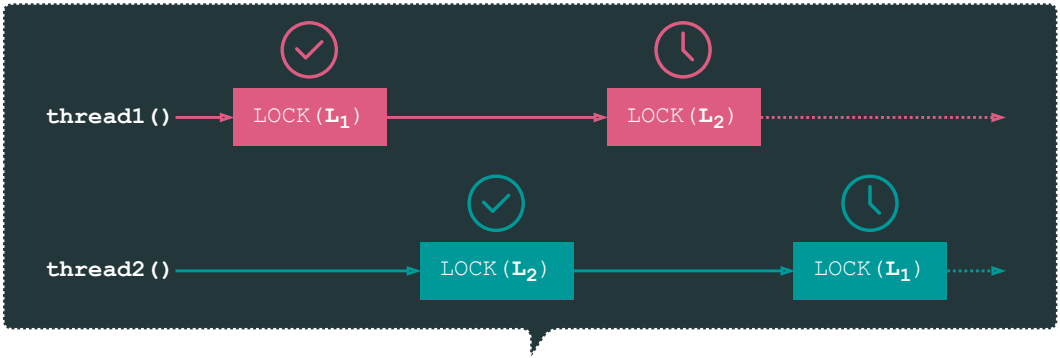
- Deadlock = one of the most common concurrency bugs.



TEXTBOOK DEADLOCK

L2D2: Low Level Deadlock Detector

- Deadlock = one of the most common concurrency bugs.



TEXTBOOK DEADLOCK

L2D2: Summarising

- Pass 1: summary construction

  { Locked, Unlocked }

  `foo()`

  { Lockset, Unlockset, PossiblyLocked, Dependencies }

- Pass 2: solving Dependencies

  ```
  Lock(L1);              Lock(L2);
  Lock(L2);→L1->L2       Lock(L1);→L2->L1
  ```

  - Compute transitive closure & flag cycles

    L1→L2→L1: thread1 acquires L1, thread2 acquires L2 ⇒ DEADLOCK

L2D2: Experimental evaluation

- 11.4 MLOC derived from Debian GNU/Linux.
- 100 % deadlock detection rate.
- Roughly 11 % false positives rate.
- Less than 1 % of the time of CPROVER.

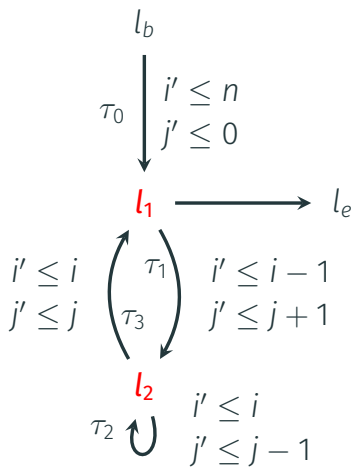
	L2D2	CPROVER
Deadlocks	8	8
False Positives	104	114
No Deadlocks	810	292
Failed Cases	80	588
Total	1002	1002

Motivation: stack example

```
    int i = n; (processed elements)
    int j = 0; (current stack size)
l1: while (i > 0) {
        i--;
        j++; (push)
l2:     while (j > 0 && *[1])
            j--; (pop)
    }
```

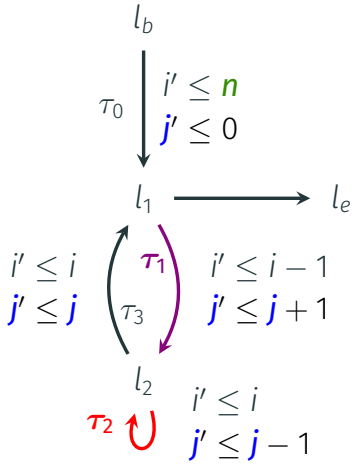
Naive approach ($\mathcal{O}(n^2)$) vs Precise analysis ($\mathcal{O}(n)$)

[1]Asterisk denotes non-determinism.



- Complexity of loop l_2 is equal to $TB(\tau_2) = n$

$TB(\tau_2)$	$\rightarrow \texttt{Incr}(j) + TB(\tau_0) \times \max(VB(0) + 0, 0)$ $\rightarrow n + 1 \times 0 = n$
$\texttt{Incr}(j)$	$\rightarrow TB(\tau_1) \times 1 = n \times 1 = n$
$TB(\tau_1)$	$\rightarrow \texttt{Incr}(i) + TB(\tau_0) \times \max(VB(n) + 0, 0)$ $\rightarrow 0 + 1 \times \max(n + 0, 0) = n$
$VB(n)$	$\rightarrow n$ (formal parameter)



LOOPER: Experimental evaluation

- Recast of the Loopus tool in Infer.
- Supports amortized complexity analysis.
- Based on recursive transition and variable bounds computation.
- Current prototype is intra-procedural only.

	real bound	our Looper	Infer Cost
#1	n	$2n$	n^2
#2	$2n$	$2n$	$5n$
#3	$4n$	$5n$	∞
#4	$*n^2$	$*n^2$	∞
#5	$2n$	$2n$	$12n$
#6	$*n$	$*n$	∞
#7	$2n$	$2n$	∞
#8	$2n$	$2n$	∞