Underapproximate Reasoning at Scale

HCSS 2023

Peter O'Hearn

Lacework and University College London

Tutorial: How to Cook a Static Analyzer

or, The Surprising Effectiveness of Substructural Proof Theory

Peter O'Hearn

Queen Mary, University of London

HCSS Conference, 21 May, 2009

Bi-Abductive symbolic execution

Pre: list(x) * list(y)
void foo(list_item *x,list_item *y)
Post: list(x)

```
node* p(list_item *y) {
  node *x, *z;
1   x=malloc(sizeof(list_item)); x->tail =  0;
2   z=malloc(sizeof(list_item)); z->tail =  0;
3   foo(x,y);
4   foo(x,z);
5   return x;
  }
```

list(y)

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

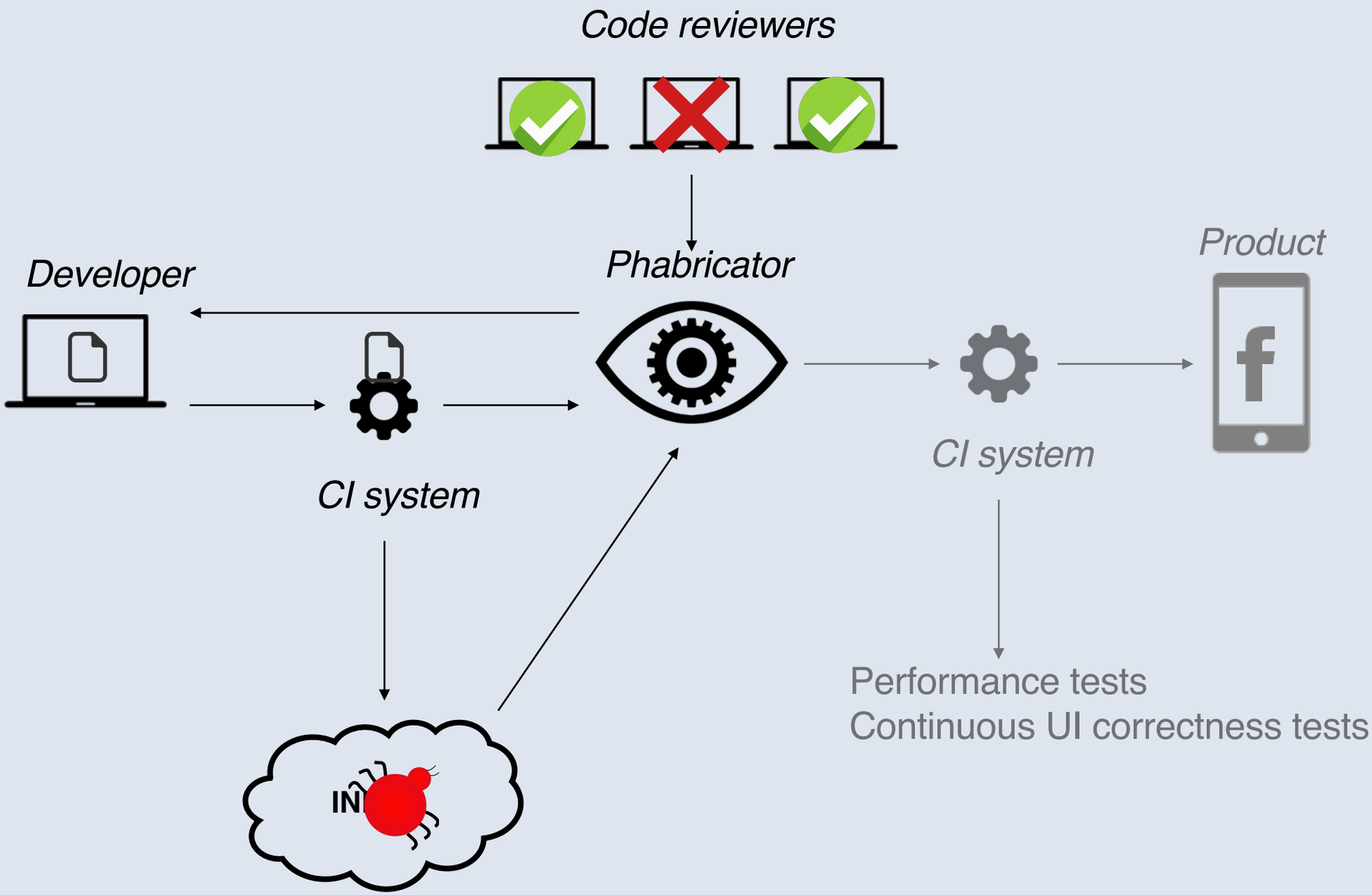
$\mathsf{list}(x)$

$\mathsf{list}(\mathtt{ret})$

Bi-abductive prover

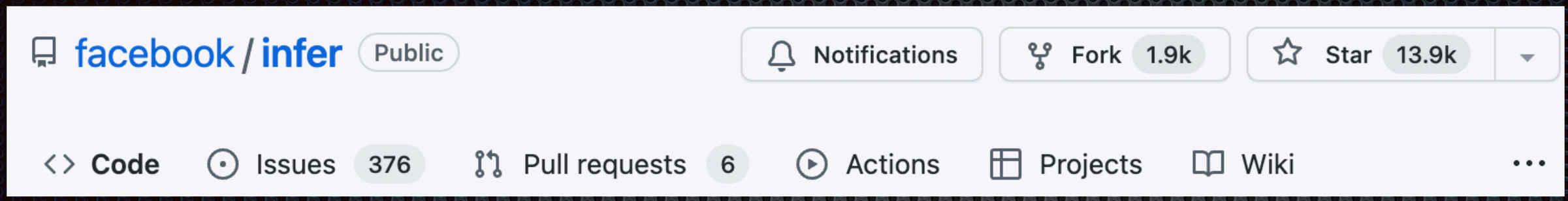
Facebook Acquires Assets Of UK Mobile Bug-Checking Software Developer Monoidics

Posted Jul 18, 2013 by ***Josh Constine*** *(@joshconstine)*


Code reviewers
Developer
Phabricator
Product
CI system
CI system
Performance tests
Continuous UI correctness tests

A stark lesson (2014)

- post land batch:
  ~0% fix rate
- diff time continuous :
  ~70% fix rate
- same analysis (Infer)



* Open sourced in 2015. Used at MSFT, AMZN,...
* Initially focus on mobile: Java+ObjC
* Since, C++, C#.... Increasing privacy focus
* 100k+ bugs caught+fixed b4 prod

P O'Hearn to M Hicks (2015)

I still want to understand concurrency, scalably. I would like to have analyses that I could deploy with high speed and low friction (e.g., not copious annotations or proof hints) and produce high-quality reports useful to programmers writing concurrent programs without disturbing their workflow too much. Then it could scale to many programmers and many programs. Maybe I am asking for too much, but that is what I would like to find.

RacerD: Feb-Oct 2016:

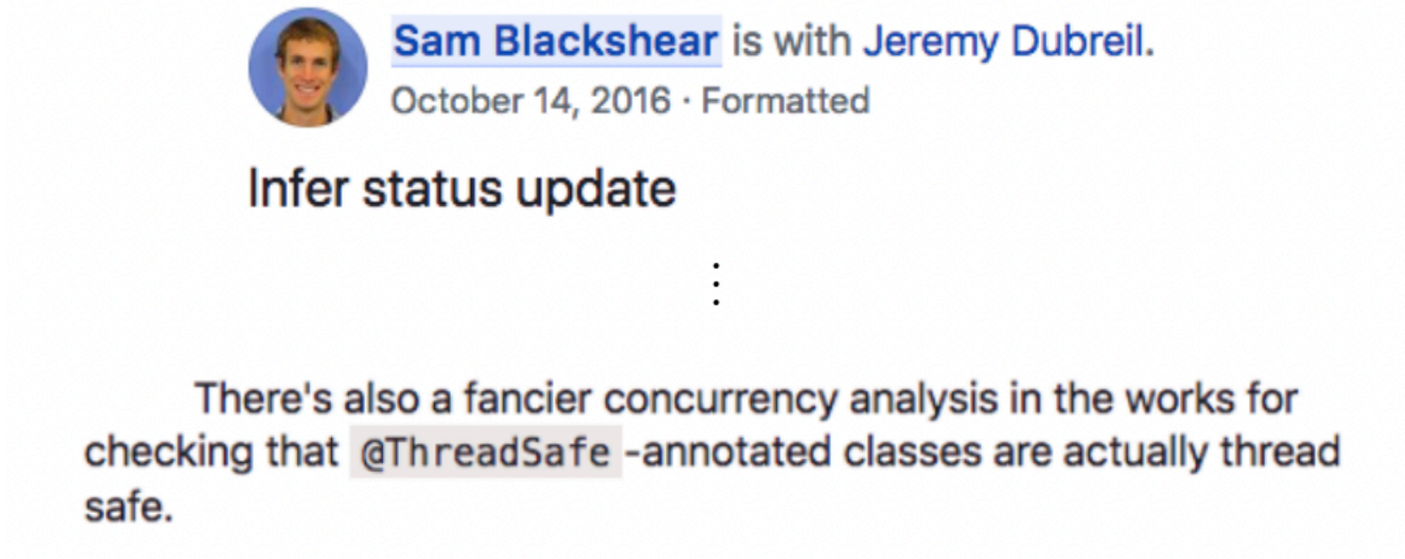
* Race Detector based on Concurrent Separation Logic

* Apply to FB's Android apps

* Started making prototype

* Goal: automatically prove thread safety of 100s k classes, keep proven via CI

A team in NYC catches wind of initial work



Hey Peter! Jason here from Android Feed Rendering. I usually work in NYC but I'm in LON until Thursday. I hear you wrote @ThreadSafe. I'd love to talk about it and how it could dramatically help us in H1 so your team has some context. Do you have time to chat in person?

Pivot: Be compositional, but under-approximate

1. **High signal:** actionable races that developers find useful; no need to (provably) find all.

2. **Inter-procedural:** track data races involving many nested calls.

3. **Low friction:** no reliance on manual annotations to specify which locks protect what data.

4. **Fast:** able to report in 15 minutes on modifications to a millions-of-lines codebase.

5. **Treatment of coarse-grained locking**, but not fine-grained

Threading information is used to limit the amount of synchronization required. As a comment from the original code explains: "`mCount` is written to only by the main thread with the lock held, read from the main thread with no lock held, or read from any other thread with the lock held."
Bottom: unsafe additions to `RaceWithMainThread .java`.

```
1   @ThreadSafe
2   class RaceWithMainThread {
3     int mCount;
4     void protectedWriteOnMainThread_OK() {
5       OurThreadUtils.assertMainThread();
6       synchronized (this) { mCount = 1; }
7     }
```

```
8     int unprotectedReadOnMainThread_OK() {
9       OurThreadUtils.assertMainThread();
10      return mCount;
11    }
12    synchronized int protectedReadOffMainThread_OK() {
13      return mCount;
14    }
```

```
15    synchronized void
16    protectedWriteOffMainThread_BAD() {
17      mCount = 2;
18    }
```

```
19    int unprotectedReadOffMainThread_BAD() {
20      return mCount;
21    }
```

True Positives Theorem: The analyzer reports no false positives (under certain assumptions)

Assumptions: (nondet()) for booleans, no recursion

under-approx of over-approx of under-approx

RacerD Results

>2.5k concurrency issues detected+fixed

No false negatives reported from a year in prod
(modulo 3 analysis implementation bugs)

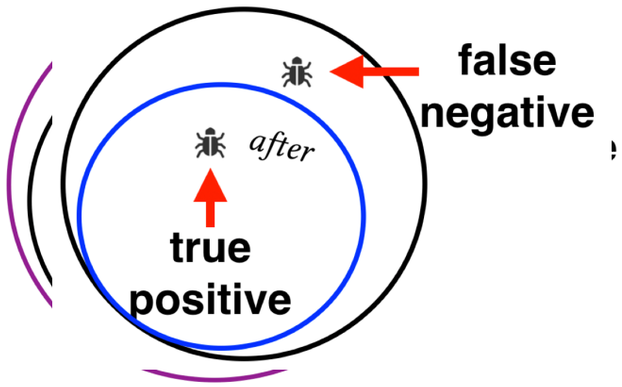
Without Infer, multithreading in News Feed would not have been tenable

Ben Jaeger, FB Android engineer

Incorrectness Logic

$$[p]\,c\,[q] \quad \text{iff} \quad post(c)\,p \supseteq q$$

Incorrectness Logic



$$[before]program[after] \quad \text{iff} \quad post(before) \begin{matrix} \subseteq \\ \supseteq \end{matrix} after$$

$$[p]c[q] \quad \text{iff} \quad post(c)\,p \sqsupseteq q$$

```
1   /* presumes: [z==11] */
2      if (x is even) {
3         if (y is odd) {
4            z=42
5      }   }
6   /* achieves: [z==42] */
```

[z:42,x:1,y:2] $\dfrac{\{p\}C\{q \wedge r\}}{\{p\}C\{q\}}$

$$[p]c[q] \quad \text{iff} \quad post(c)\, p \sqsupseteq q$$

```
1   /* presumes: [z==11] */
2     if (x is even) {
3       if (y is odd) {
4           z=42
5     }   }
6   /* achieves: [z==42 && (x is even) && (y is odd) ] */
```

$$\frac{\{p\}C\{q \wedge r\}}{\{p\}C\{q\}}$$

Consequence

$$\frac{p' \Leftarrow p \quad [p]C[\epsilon\colon q] \quad q \Leftarrow q'}{[p']C[\epsilon\colon q']}$$

$$\frac{[p]C[q_1 \vee q_2]}{[p]C[q_1]}$$

Ignore

Infer.Pulse

- Analyzer for C++ lifetimes,
  numbers on 100s kLOC codebase

- 20 disjunct limit versus 50 disjuncts
  (5 unrollings each)

- 20 is 2.75x wall clock faster than 50

- 3.1x user time faster

- 20 find 97% of issues of 50

A duality

For overapproximate reasoning

You get to forget information as you go along a path, but you must remember all the paths.

For under approximate reasoning

You must remember information as you go along a path, but you get to forget some of the paths

Concretizing

```
39   void difficult()
40   /*achieves: [ok: y==49 && x==1]  */
41     { int z = nondet();
42       if (y == z*z)
43         {x=1;}
44     }
```

Consequence

$$\frac{p' \Leftarrow p \quad [p]C[\epsilon : q] \quad q \Leftarrow q'}{[p']C[\epsilon : q']} \qquad (\exists z \,.\, y == z * z) \iff y == 49$$

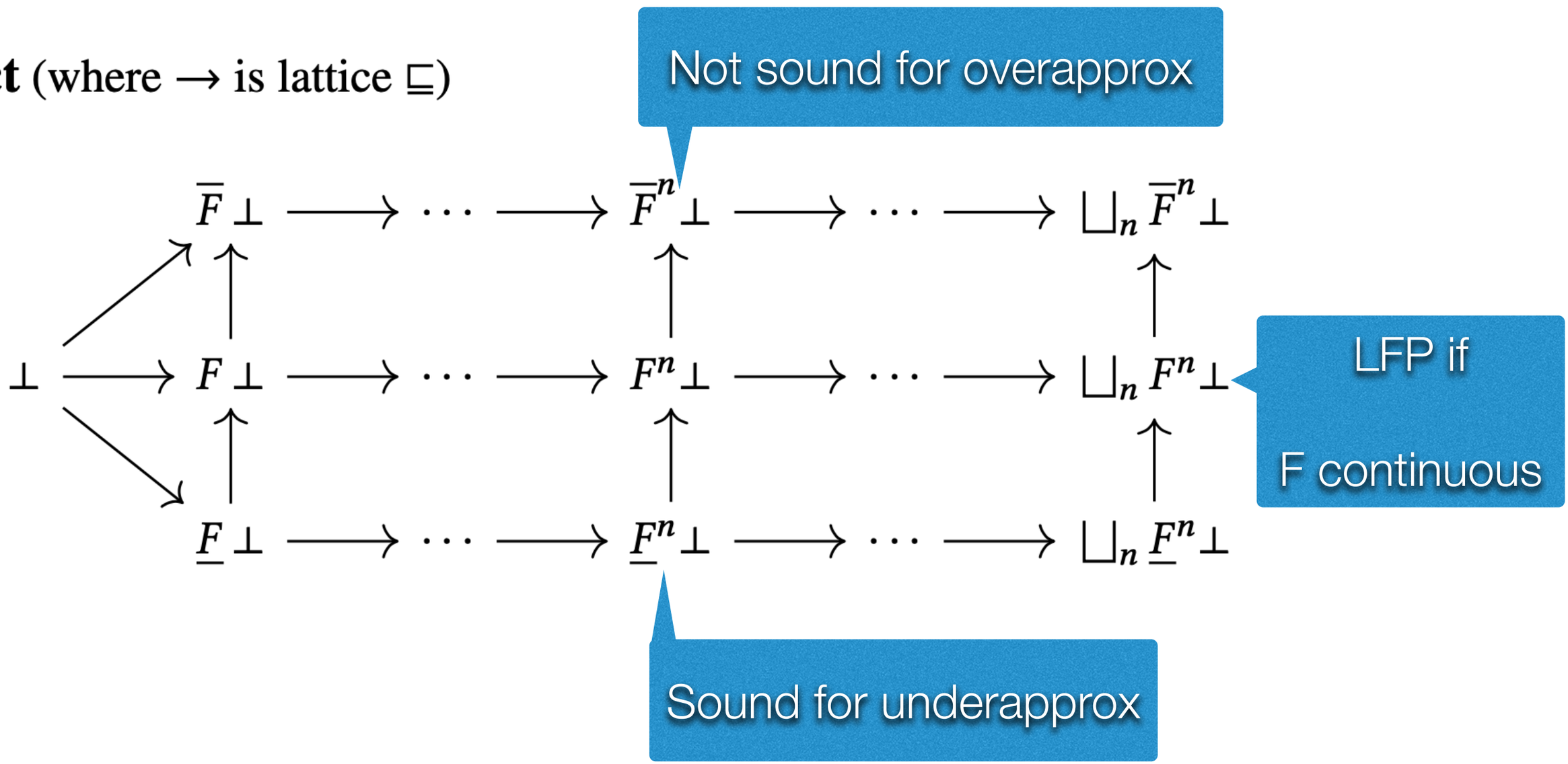
cf. KLEE, DART, SAGE

- The pragmatic analyzer principle of **concretizing symbolic values** corresponds to the logical principle of **shrinking the post-assertio**n

Underapproximate and Overapproximate Semantics

assume $\underline{F} \preceq F \preceq \overline{F}$ monotone (where $G \preceq H$ iff $\forall X.\, GX \sqsubseteq HX$)

Fact (where $\longrightarrow$ is lattice $\sqsubseteq$)



- **Theorem** (Soundness and Completeness)
  $[p]C[\epsilon\colon q]$ is true iff is it provable.

For $[p](C)^\star[ok\colon q]$ completeness, use

$p(n) = \{\sigma \mid$ you can get back from σ to some state in p by executing C backwards n times$\}$.

For $[p](C)^\star[er\colon q]$ case use frontier idea and Iterate Two.

Empty under-approximates

$$[p]C[\epsilon\colon \mathit{false}]$$

Consequence

$$\frac{p' \Leftarrow p \quad [p]C[\epsilon\colon q] \quad q \Leftarrow q'}{[p']C[\epsilon\colon q']}$$

Disjunction

$$\frac{[p_1]C[\epsilon\colon q_1] \quad [p_2]C[\epsilon\colon q_2]}{[p_1 \vee p_2]C[\epsilon\colon q_1 \vee q_2]}$$

Unit

$$[p]\texttt{skip}[ok\colon p][er\colon \mathit{false}]$$

Sequencing (short-circuit)

$$\frac{[p]C_1[er\colon r]}{[p]C_1; C_2[er\colon r]}$$

Sequencing (normal)

$$\frac{[p]C_1[ok\colon q] \quad [q]C_2[\epsilon\colon r]}{[p]C_1; C_2[\epsilon\colon r]}$$

Iterate zero

$$[p]C^\star[ok\colon p]$$

Iterate non-zero

$$\frac{[p]C^\star; C[\epsilon\colon q]}{[p]C^\star[\epsilon\colon q]}$$

Backwards Variant (where n fresh)

$$\frac{[p(n) \wedge nat(n)]C[ok\colon p(n+1) \wedge nat(n)]}{[p(0)]C^\star[ok\colon \exists n.p(n) \wedge nat(n)]}$$

Choice (where $i = 1$ or 2)

$$\frac{[p]C_i[\epsilon\colon q]}{[p]C_1 + C_2[\epsilon\colon q]}$$

Error

$$[p]\texttt{error}()[ok\colon \mathit{false}][er\colon p]$$

Assume

$$[p]\texttt{assume}\ B[ok\colon p \wedge B][er\colon \mathit{false}]$$

Assignment

$$[p]x = e[ok\colon \exists x'.p[x'/x] \wedge x = e[x'/x]][er\colon \mathit{false}]$$

Nondet Assignment

$$[p]x = \texttt{nondet}()[ok\colon \exists x' p][er\colon \mathit{false}]$$

Constancy

$$\frac{[p]C[\epsilon\colon q]}{[p \wedge f]C[\epsilon\colon q \wedge f]} \quad Mod(C) \cap Free(f) = \emptyset$$

Local Variable

$$\frac{[p]C(y/x)[\epsilon\colon q]}{[p]\texttt{local}\ x.C[\epsilon\colon \exists y.q]} \quad y \notin Free(p, C)$$

Substitution I

$$\frac{[p]C[\epsilon\colon q]}{([p]C[\epsilon\colon q])(e/x)} \quad \big(Free(e) \cup \{x\}\big) \cap Free(C) = \emptyset$$

Substitution II

$$\frac{[p]C[\epsilon\colon q]}{([p]C[\epsilon\colon q])(y/x)} \quad y \notin Free(p, C, q)$$

Testing + Verification

Program testing can be used to show the presence of bugs, but never to show their absence!

POPL referee

This paper threatens to make bug finding actually respectable

December 2021

Peter O'Hearn • You

Engineering Director at Lacework

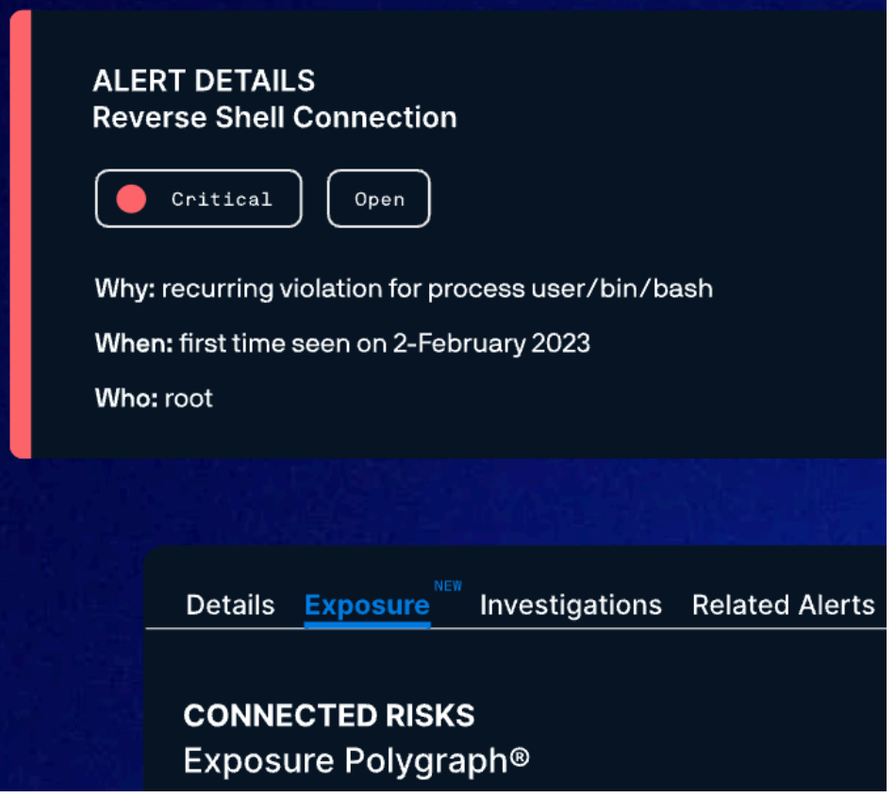
1yr •

After 8+ great years at Facebook, it’s time to #LaceUp!

I'm joining Lacework as the company’s first engineer in London, driving work on code analysis (which complements and shifts left the current blackbox analytics offering). My main focus will be to make code reasoning useful for the many and not just the few; "proofs for the masses!"

Secure from code to cloud

Get the data-driven cloud-native application protection platform (CNAPP)



Dev

Prod

Agent,

Agentless

An Attack Path

An Attack Path

Security Invariants

* *service not internet accessible*
* *service has no critical OSS vulns*
* *service has no access to RDS*

An Attack Path

Security Invariants

* *service not internet accessible*
* *service has no critical OSS vulns*
* *service has no access to RDS*

Intensional

A flow

* *Computer grabs a snapshot (agentless)*

Under

* *Computer creates a graph, generates attack paths*

Over

* *Human chooses a path*
  Establishes security invariant

Over

* *Computer checks invariant*

Both

Active Vuln Detection (Code aware agent, Godefroid, Condra, ++)

* *LW agent monitors network traffic, used in anomaly detection (mixes under+over, to explain another day)*

* *CAA extension monitors package activity*

* *Under, but for a history rather than a snapshot*

* *Soundness: if CAA says active, it was used*
* *Completeness: if CAA says inactive, it was not used*

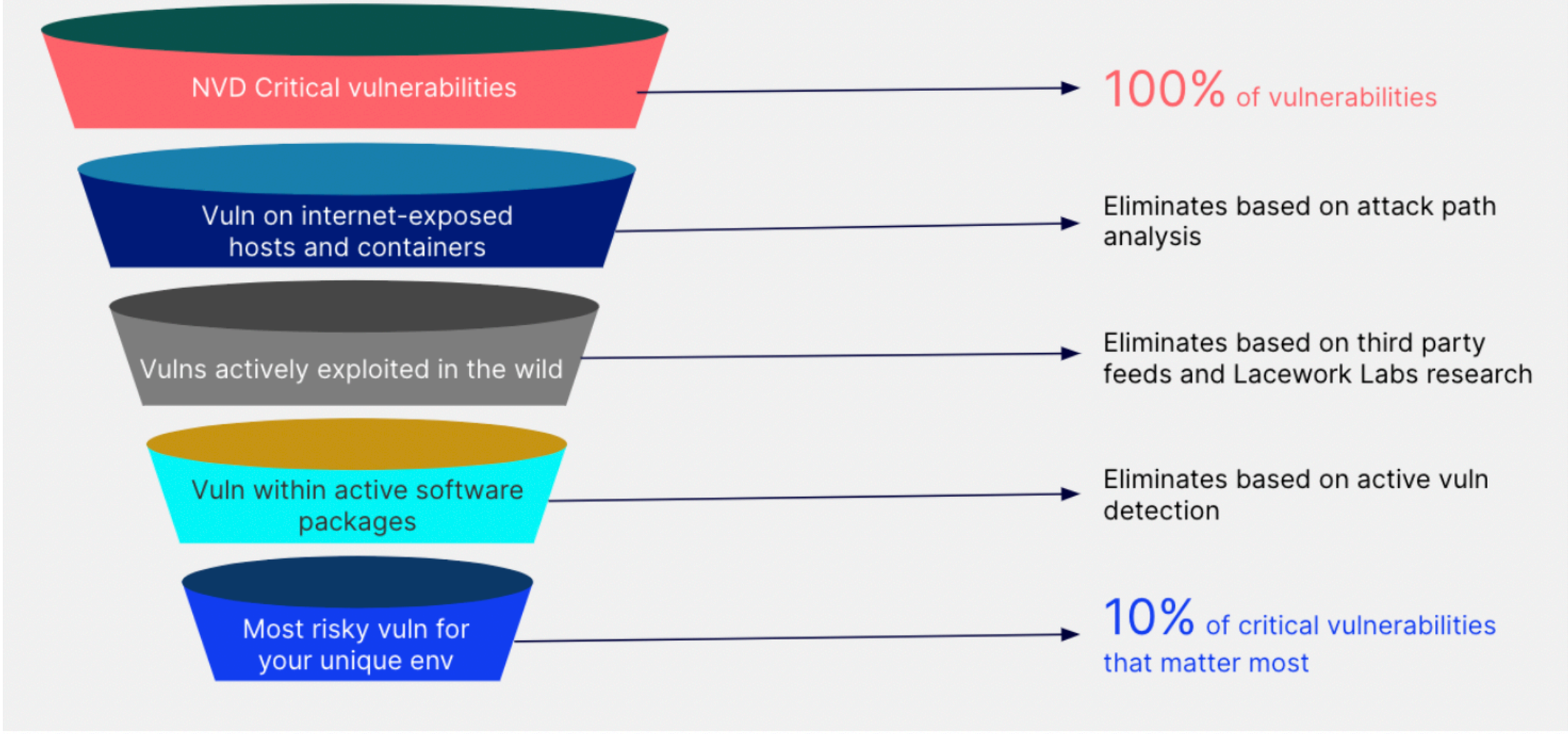


Figure 2: How the custom risk-based vulnerability score is determined

A flow

* *Computer grabs a snapshot (agentless)*

Under

* *Computer creates a graph, generates attack paths*

Over

* *Human chooses a path*
  Establishes security invariant

Over

* *Computer checks invariant*

Both

A flow

* *Computer grabs a snapshot (agentless)*

Under

* *Computer creates a graph, generates attack paths*

Over

* *CAA helps prioritise*

Under

* *Human chooses a path*
  Establishes security invariant

Over

* *Computer checks invariant*

Both

* What I'm working on:
    * Shift that goodness left (speed)
    * Connect left and right (context, better together)
    * Signal at IDE, PR, Deploy times (underapprox enables)

* Stay tuned!