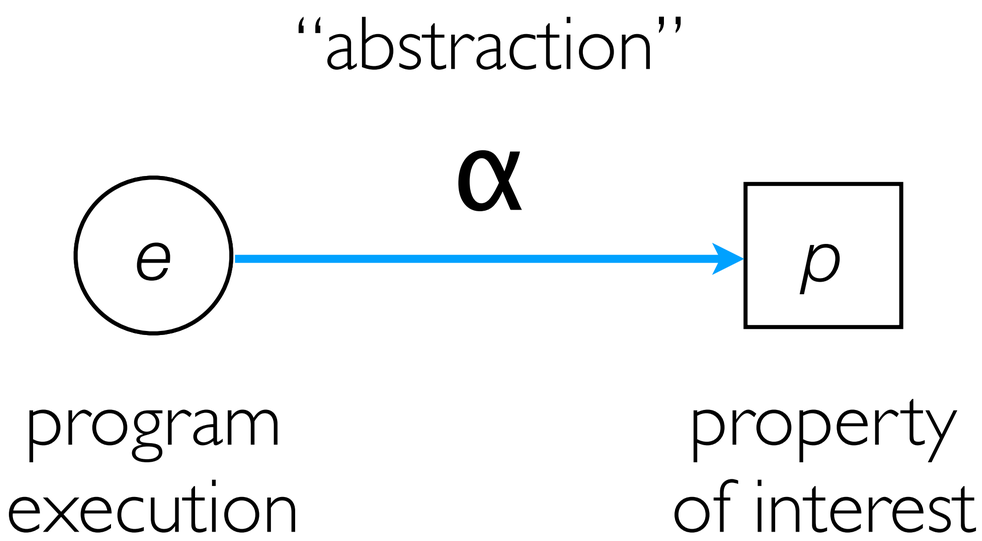
A True Positives Theorem for a Static Race Detector

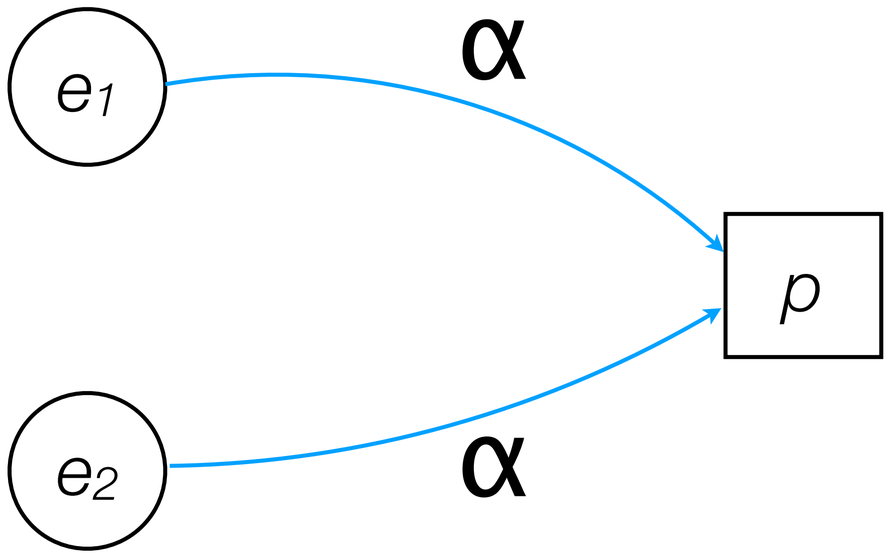
Presentation by Julia Belyakova and Artem Pelenitsyn
For CS 7580 (instructor: Jan Vitek), 10/30/2019

A subset of slides is taken from Ilya Sergey's web page

Static Analyses for Program Validation

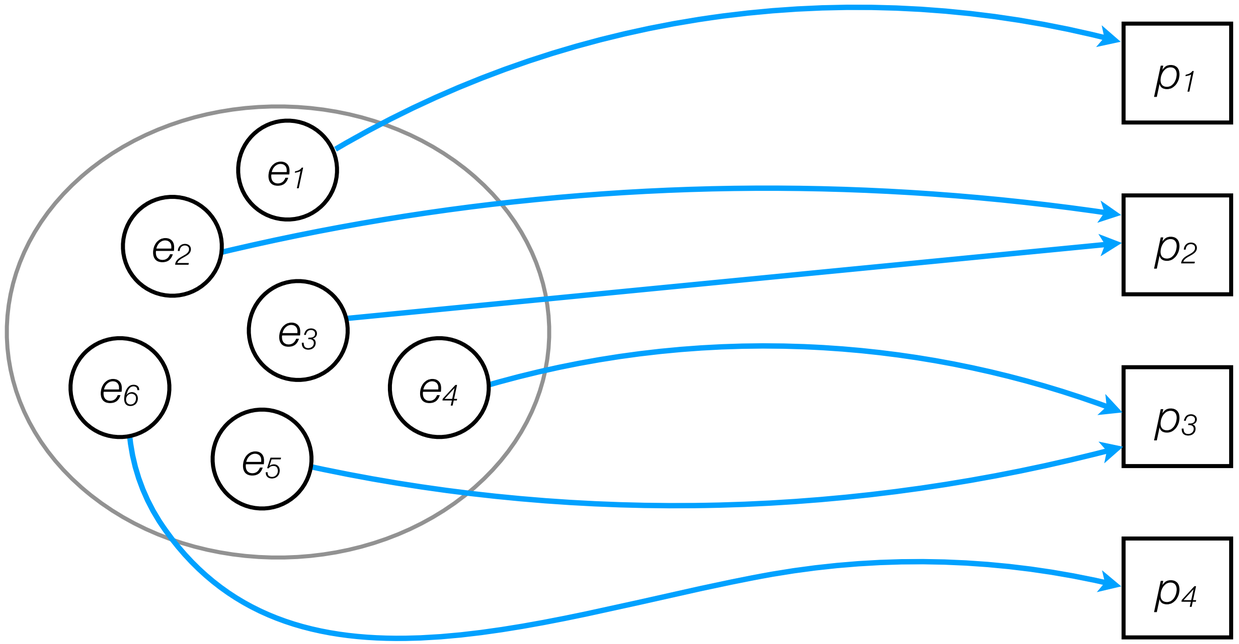
The Essence of Static Analysis



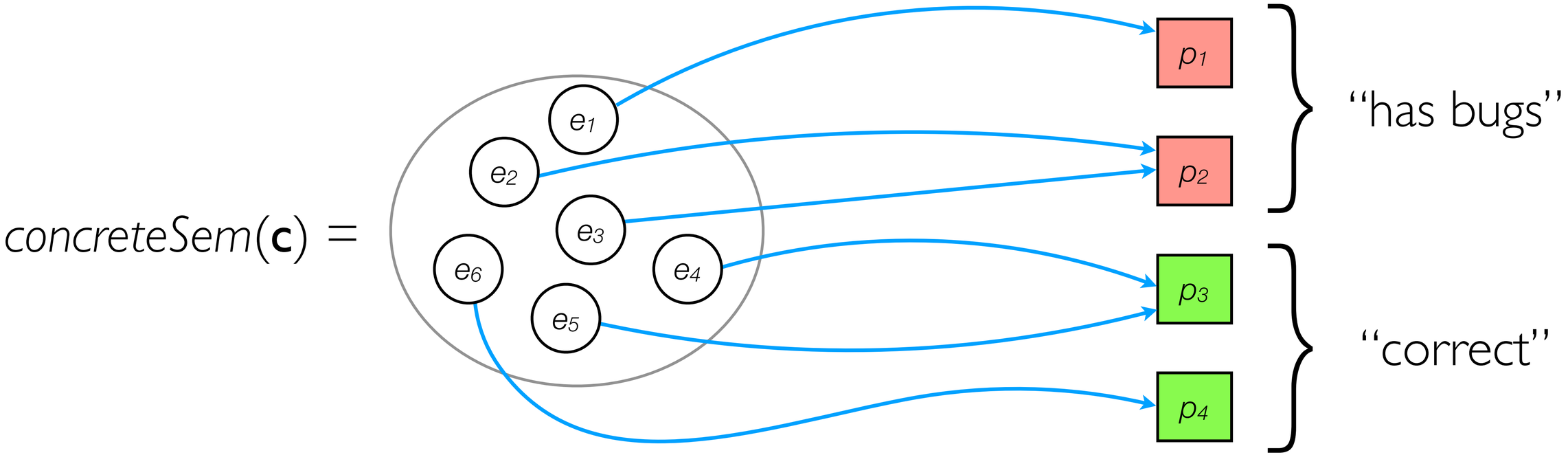

e_1
α
p
e_2
α

Static Analysis

$concreteSem(\mathbf{c}) =$

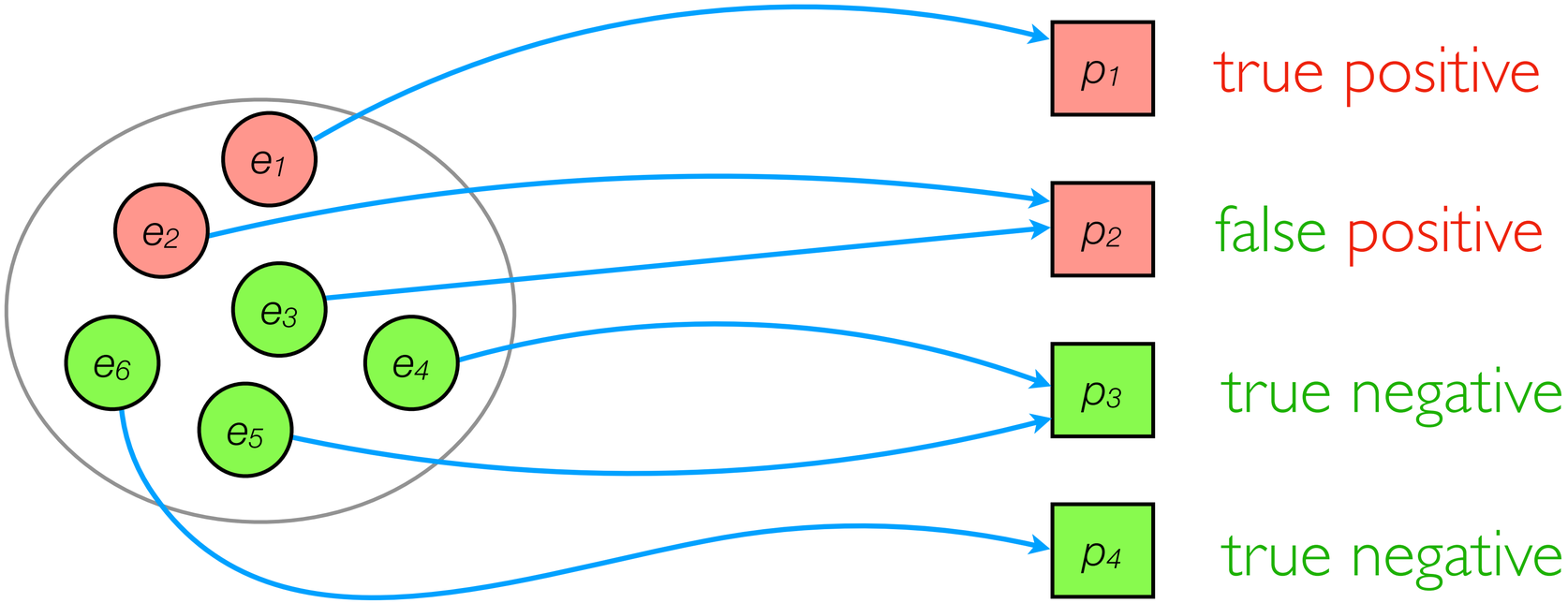


Static Analysis

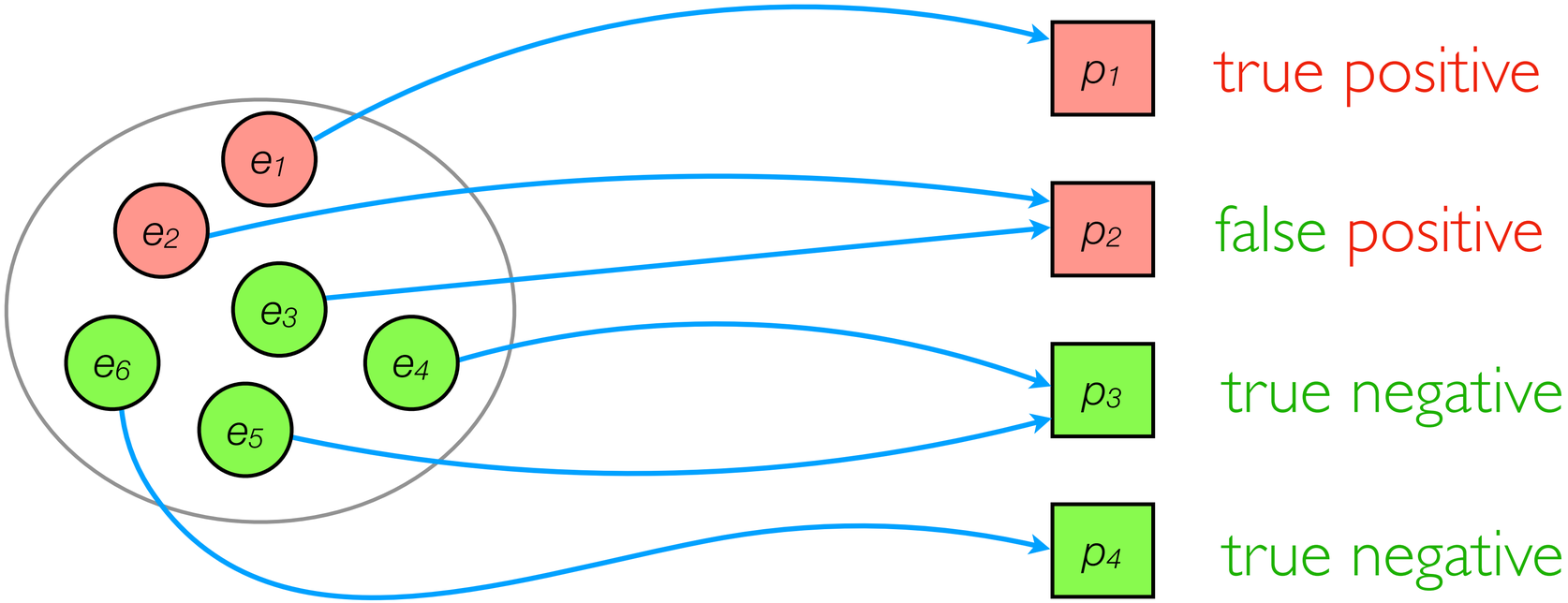


Verifier
or a
Bug Detector?

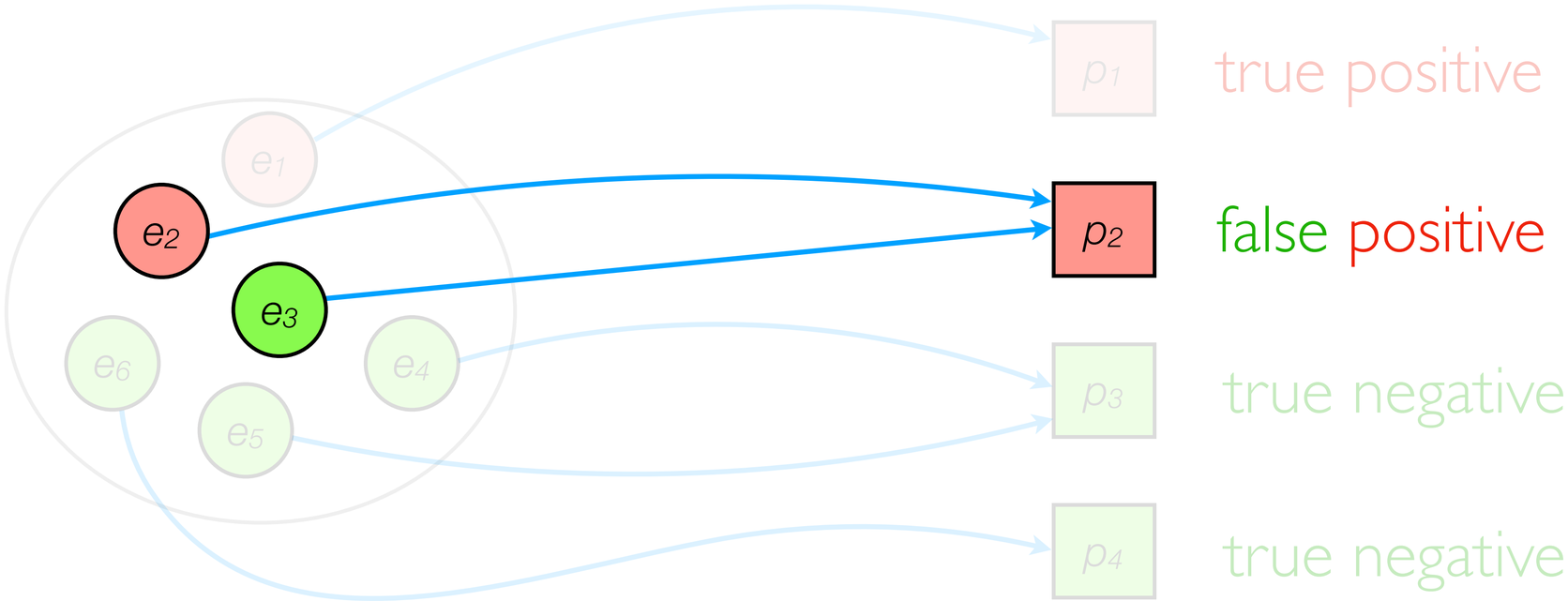
Program Verifier



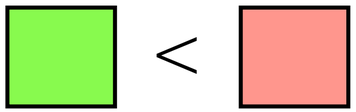
Sound Program Verifier



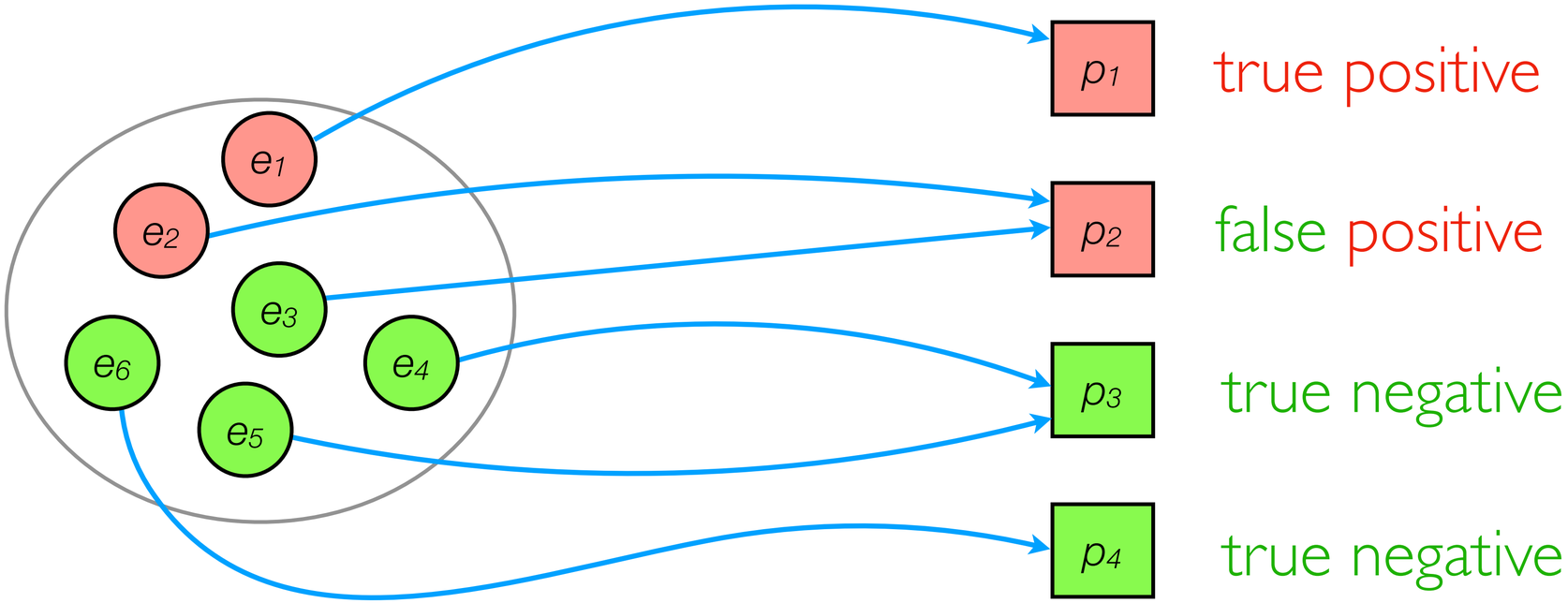
Sound Program Verifier



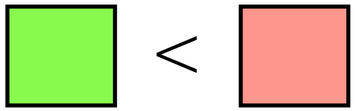
abstract over-approximation



Sound Program Verifier

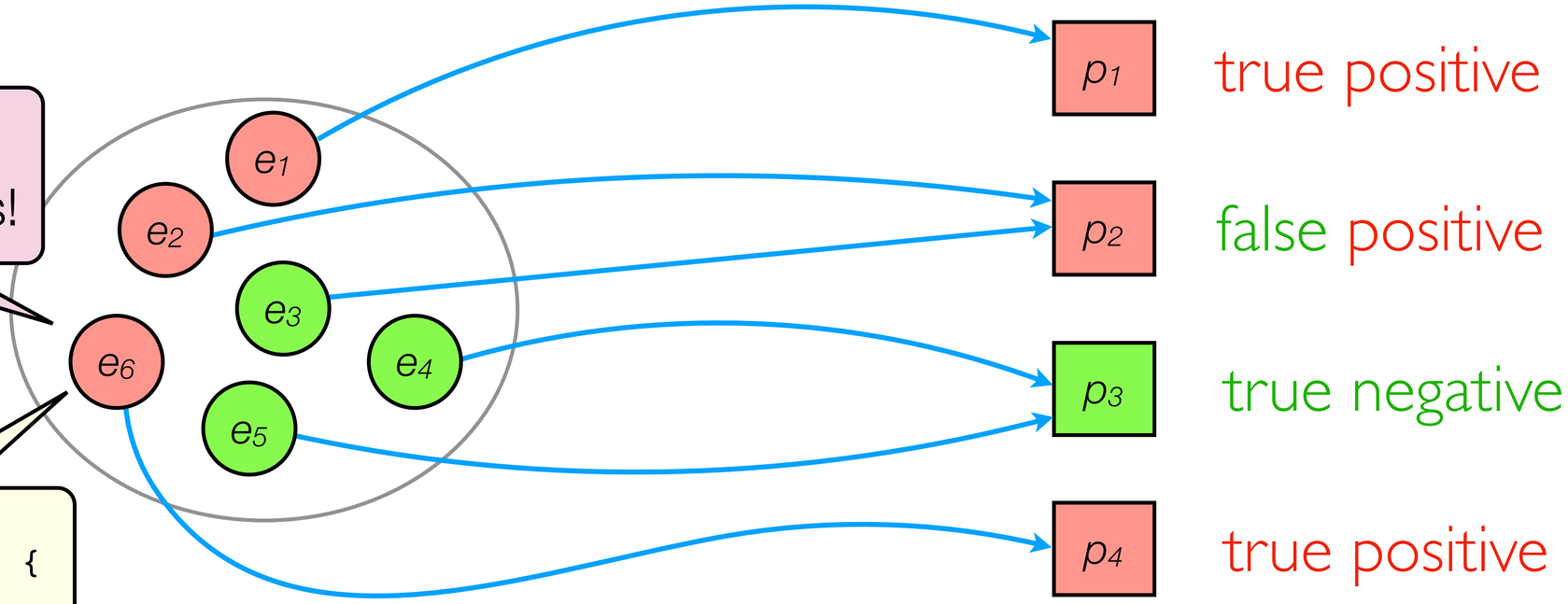


abstract over-approximation



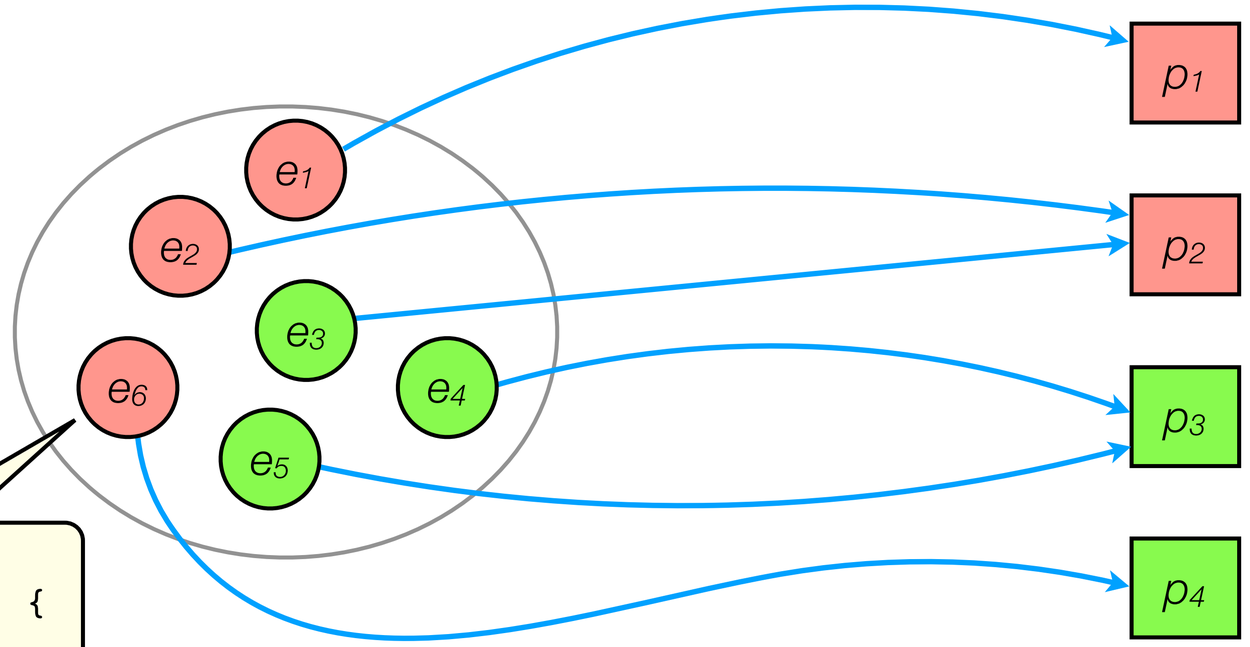
Sound Program Verifier

Developer:
Go away, that never happens!




```
if (n == VERY_UNLIKELY_VALUE) {
  bug.explode();
} else {
  // do nothing
}
```

Unsound Program “Verifier”



true positive

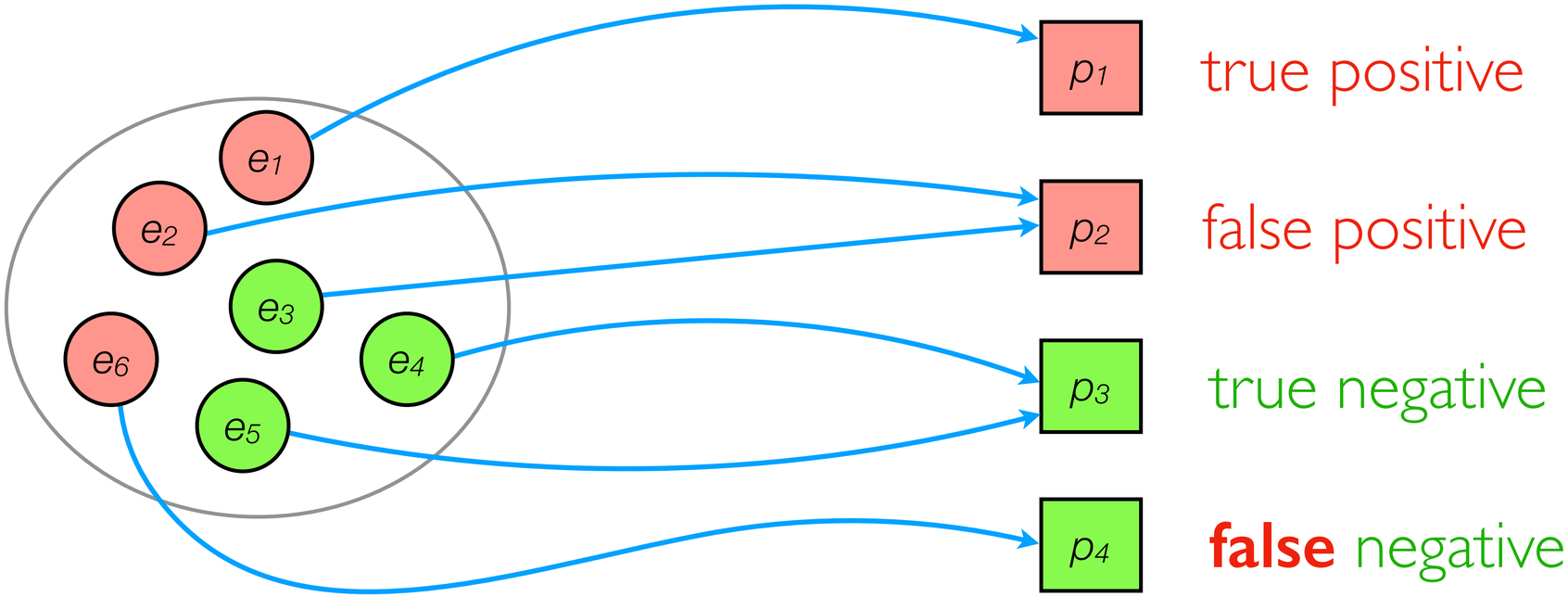
false positive

true negative

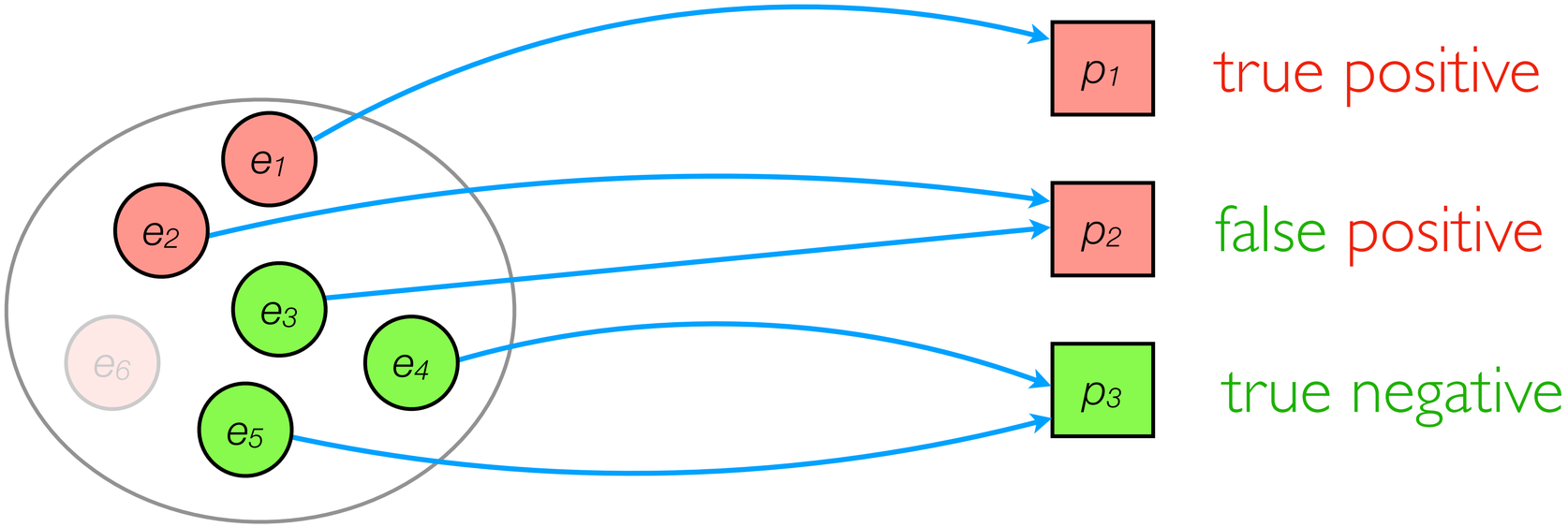
false negative

```
if (n == VERY_UNLIKELY_VALUE) {
  bug.explode();
} else {
  // do nothing
}
```

“Sound” Program Verifier

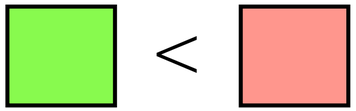


“Sound” Program Verifier



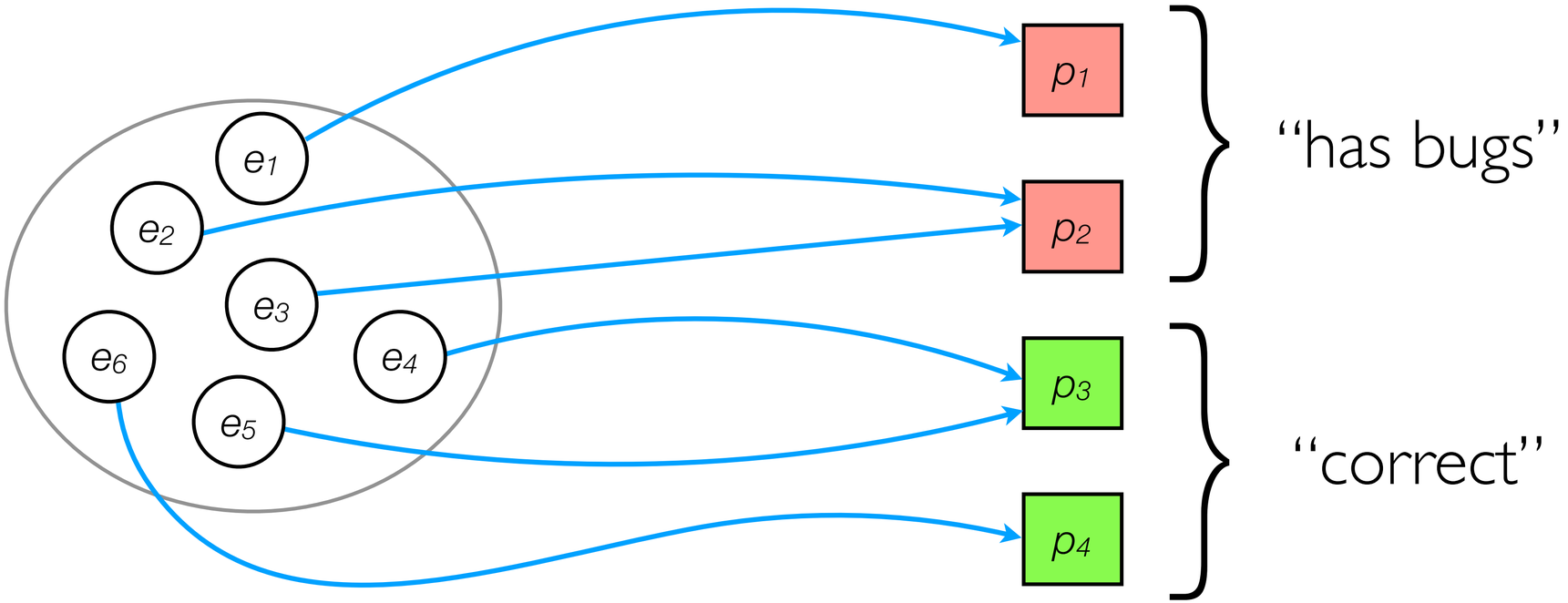
concrete under-approximation

abstract over-approximation

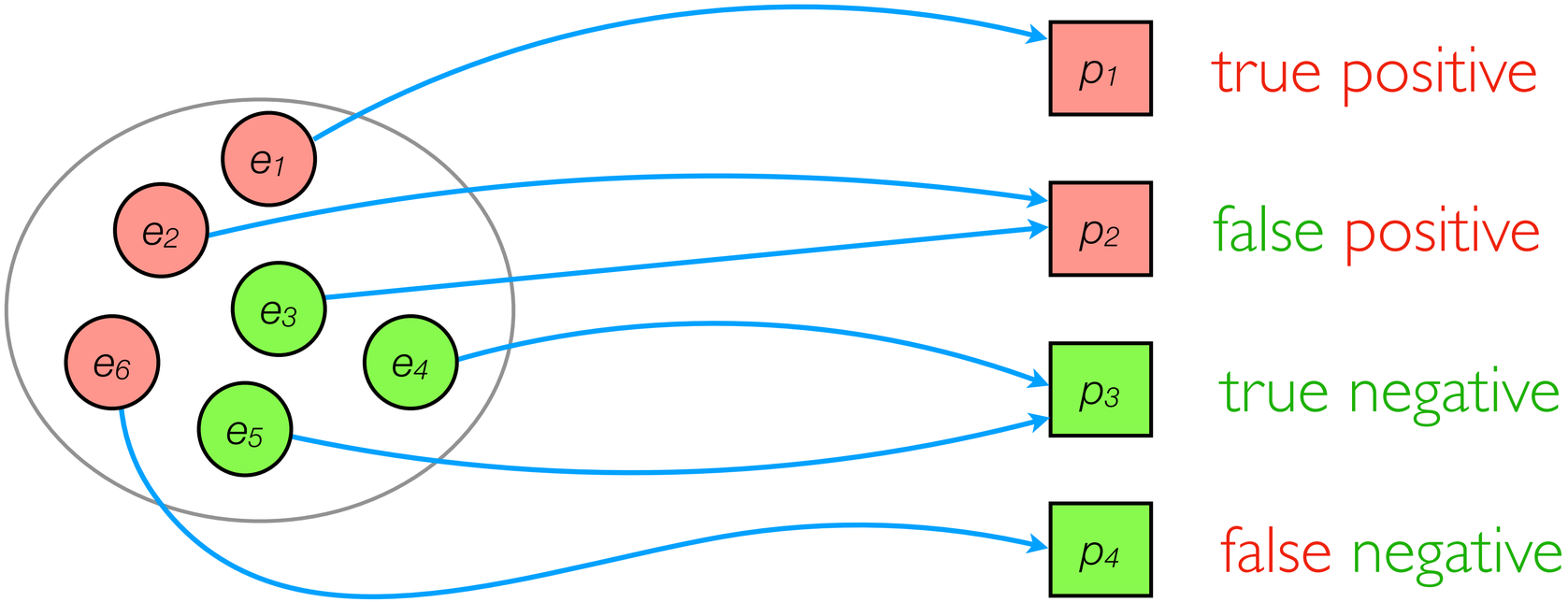


Sound Static Verifiers

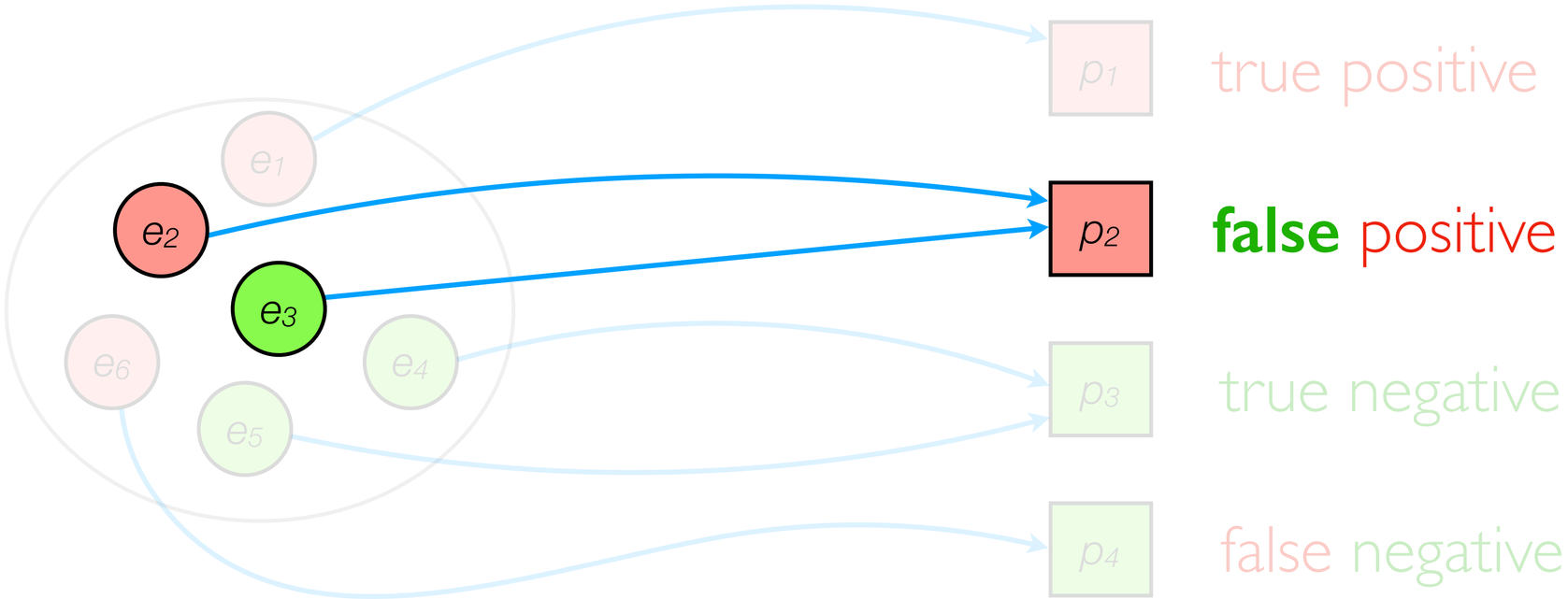
- False negatives (bugs missed) are ***bad***
- False positives (non-bugs reported) are ***okay***
- Constructed as ***over-approximation*** (of ***under-approximation***)
- **Soundness Theorem**:
  Under certain assumptions about the programs, the analyser has *no false negatives*.


e_1
e_2
e_3
e_4
e_5
e_6
p_1
p_2
p_3
p_4
"has bugs"
"correct"

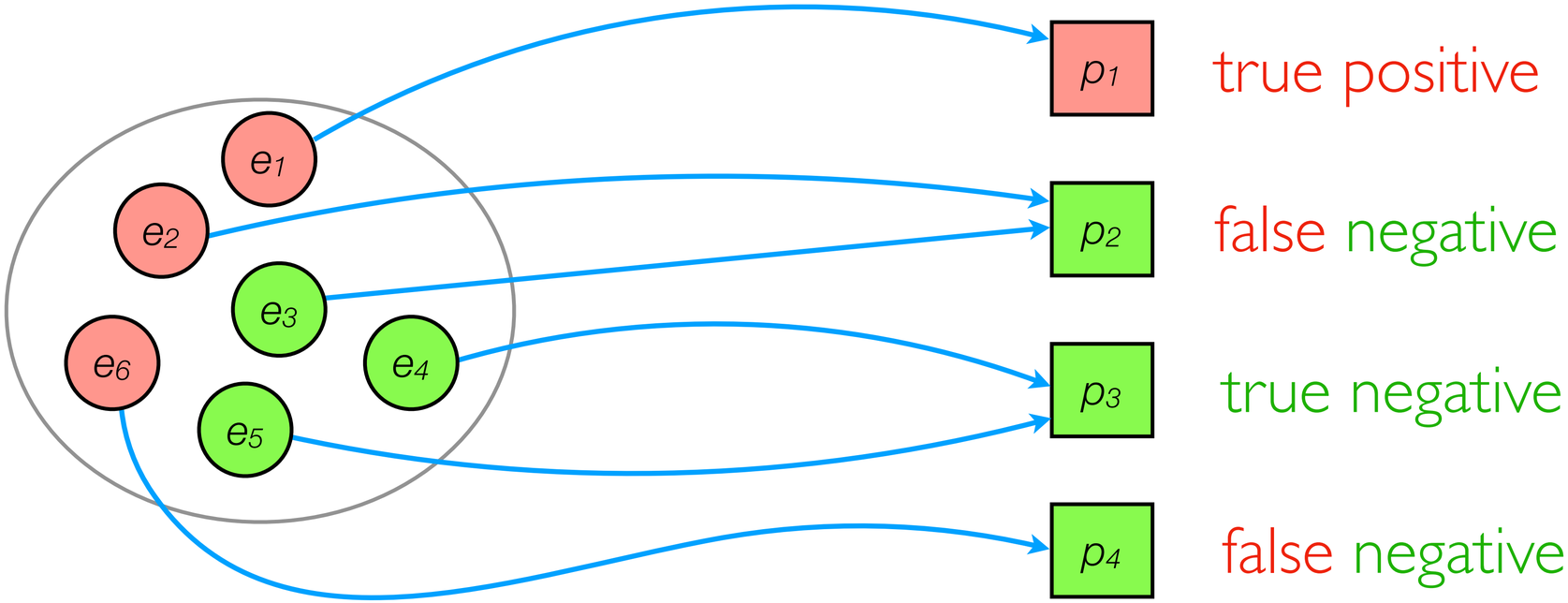
Static Bug Finder



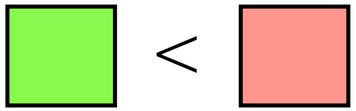
Unsound Static Bug Finder



Sound (but imprecise) Static Bug Finder



abstract *under-approximation*

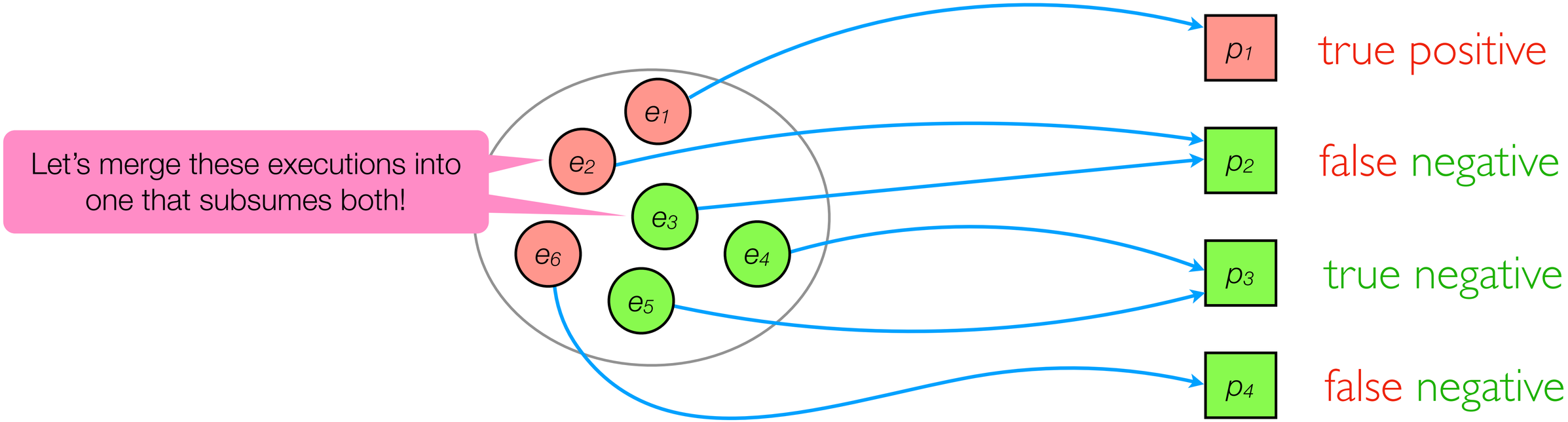


Loss of Precision in Static Bug Finders

```
if (n != VERY_UNLIKELY_VALUE) {
    e2    // bug happens here
} else {
    e3    // normal execution
}
```

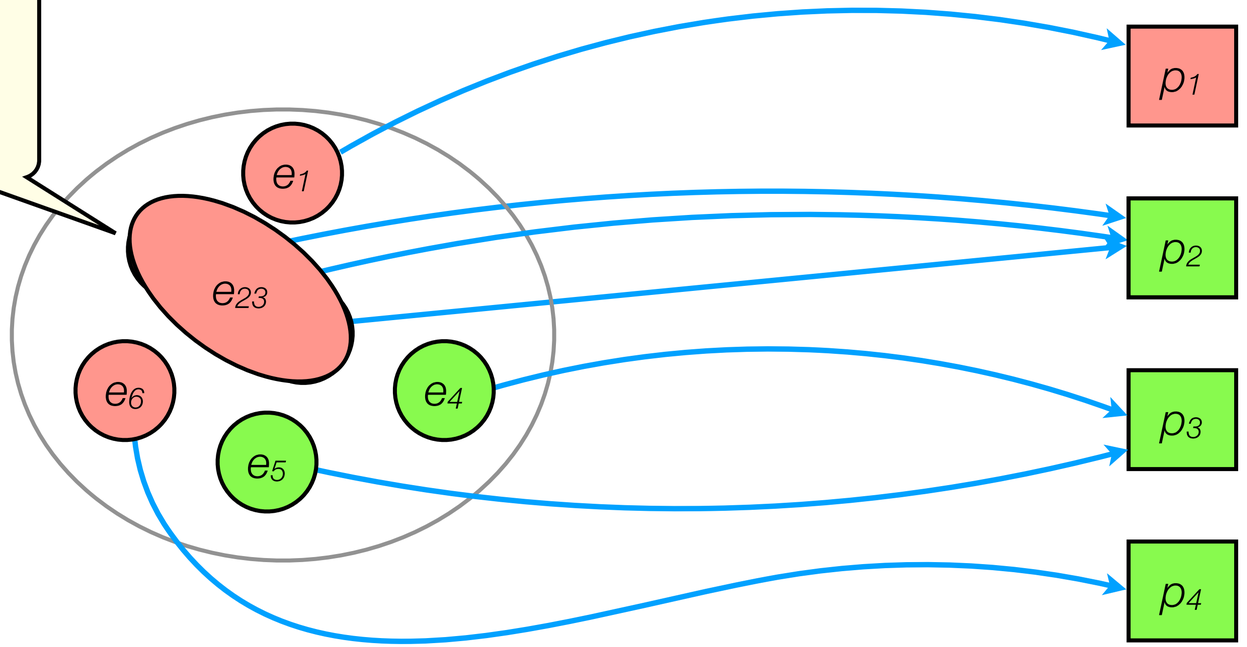
Idea: *over-approximate* in concrete semantics!

Sound (but Imprecise) Static Bug Finder



```
if (*) {
    // bug happens here
} else {
    // normal execution
}
```

overApproxConcreteSem(c) =



true positive

false positive

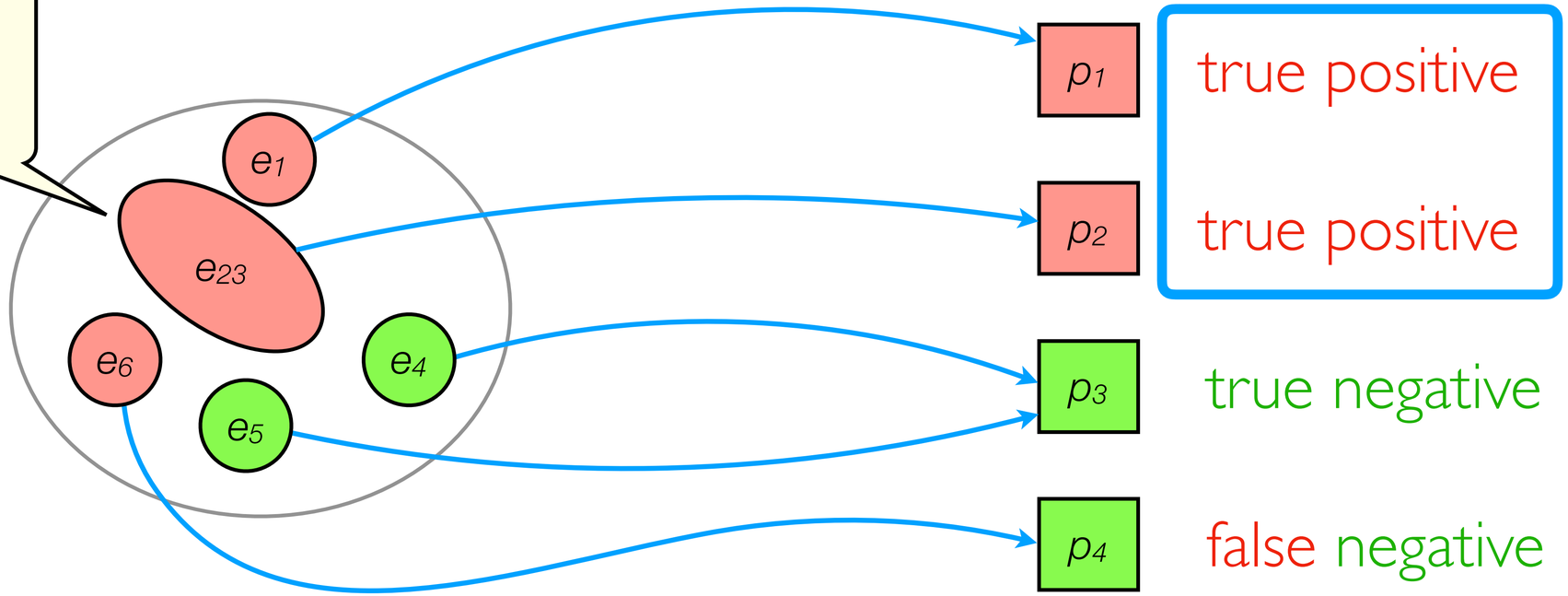
true negative

false negative

Sound Static Bug Finder

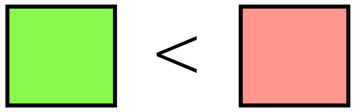
```
if (*) {
    // bug happens here
} else {
    // normal execution
}
```

overApproxConcreteSem(c) =



concrete *over-approximation*

abstract *under-approximation*



Towards Sound Static Bug Finders

(this work)

- False negatives (bugs missed) are ***okay***
- False positives (non-bugs reported) are ***bad***
- Constructed as ***under-approximation*** of ***over-approximation***
- **Soundness (True Positives) Theorem**:
  Under certain assumptions about the programs, the analyser has ***no false positives***.

A True Positives Theorem for a Static Race Detector

Nikos Gorogiannis — facebook, Middlesex University London

Peter O'Hearn — facebook, UCL

Ilya Sergey — Yale NUS College, NUS National University of Singapore

Key Messages

Unsound (and incomplete) static analyses can be *principled*,
satisfying meaningful theorems
that help to understand their behaviour and guide their design

One can have an unsound but effective static analysis,
which has significant industrial impact,
and which is supported by a *meaningful theorem*.

Context

1. We had a demonstrably-effective industrial analysis: RacerD (OOPSLA'18); >3k fixes in Facebook Java
2. No soundness theorem
3. Architecture: compositional abstract interpreter
4. No heuristic alarm filtering

Just ad hoc?

Our reaction:
Semantics/theory should understand/explain, not lecture.

Case Study: RacerDX

- A provably TP-Sound version of Facebook's **RacerD** concurrency analyser (Blackshear et al., OOPSLA'18)

- **Buggy executions:** *data races* in *lock-based concurrent* programs

- **Syntactic assumptions**:
  Java programs with well-scoped locking (`synchronised`), no recursion, reflection, dynamic class loading; global variables are ignored.

- **Concrete over-approximation**:
  Loops and conditionals are *non-deterministic.*

Formal Result

RacerDX enjoys the True Positives Theorem *wrt*. Data Race Detection

(Details in the paper)

For an Idealized Language

Static Analysis with True Positives Theorem*

Goal: to build a static analysis s.t.
if the analysis **reports a bug**,
it is a **true bug**

True bug can be exhibited

The race reported by the analysis for program P is a **true race**

$\longleftrightarrow$

There exists an execution of P `that` **`exhibits`** `the race`

Ingredients of the formalism

For an Idealized Language

- program
- execution
- race
- analysis
- proof

Ingredients of a data race

```
class Bloop {
  public int f = 1;
}
```

Racy Program:

```
b = new Bloop()
u = new Burble()

u.meps(b) || u.reps(b)
```

parallel composition

```
class Burble {

  public void meps(Bloop b) {
    synchronized (this) {
      System.out.println(b.f);
    }
  }

  public void reps(Bloop b) {
    b.f = 42;
  }

  public void beps(Bloop b) {
    b = new Bloop();
    b.f = 239;
  }
}
```

lock()
println(b.f)
unlock()

path

b.f := 42

Concurrent program syntax

$$
\begin{array}{rcl}
\mathsf{f} \in \mathsf{Field} & & \text{field names} \\
x, \mathsf{arg}_i \in \mathsf{Var} & & \text{variables} \\
\pi \in \mathsf{Path} & ::= & x.\mathsf{f} \mid \pi.\mathsf{f} \\
\mathsf{e} \in \mathsf{Exp} & \triangleq & \mathsf{Var} \cup \mathsf{Path} \\
c \in \mathsf{Stmt} & ::= & \mathtt{skip} \mid x := x \mid x := \pi \mid \pi := x \mid x := \mathsf{new}() \mid \mathtt{lock}() \mid \mathtt{unlock}() \mid \mathsf{pop}() \\
C \in \mathsf{CStmt} & ::= & c \mid C; c \mid C; \mathtt{if} * \mathtt{then}\ C\ \mathtt{else}\ C \mid C; \mathtt{while} * \mathtt{do}\ C \mid C; \mathsf{m}(\mathsf{e}, \ldots, \mathsf{e}) \\
M \in \mathsf{Method} & ::= & \mathsf{m}(\mathsf{arg}_1, \ldots, \mathsf{arg}_n)\{\ C\ \} \\
p \in \mathsf{Program} & ::= & C \parallel C
\end{array}
$$

Single-threaded program C: concrete semantics

- State $\varsigma = \langle c, S, h, L \rangle$
  (command, stack, heap, locks)

- Trace (list of states)
  $$\tau = [\varsigma_0, \ldots, \varsigma_n]$$

- Concrete semantics
  (set of traces)
  $$[\![C]\!]\ \varsigma \in \mathcal{P}(\mathcal{T})$$

Concrete semantics of commands

$$\boxed{[\![\texttt{skip}]\!]}\ \langle S, h, L\rangle \triangleq \{\epsilon\}$$

$$[\![x := \pi]\!]\ \langle s :: S, h, L\rangle \triangleq \begin{cases} \emptyset & \text{if } \lfloor \pi \rfloor_{s,h} \notin \mathrm{dom}\,(h) \\ \{[\langle x := \pi, s[x \mapsto h(\lfloor \pi \rfloor_{s,h})] :: S, h, L\rangle]\} & \text{otherwise} \end{cases}$$

$$\boxed{[\![\pi := x]\!]}\ \langle s :: S, h, L\rangle \triangleq \begin{cases} \emptyset & \text{if } \lfloor \pi \rfloor_{s,h} \notin \mathrm{dom}\,(h) \\ \{[\langle \pi := x, s :: S, h[\lfloor \pi \rfloor_{s,h} \mapsto s(x)], L\rangle]\} & \text{otherwise} \end{cases}$$

$$[\![x := \mathsf{new}()]\!]\ \langle s :: S, h, L\rangle \triangleq \left\{ [\langle x := \mathsf{new}(), s' :: S, h', L\rangle] \;\middle|\; \begin{array}{l} \ell \notin \mathsf{locn}(h), s' = s[x \mapsto \ell], \\ h' = h \cup \bigcup_{\mathsf{f} \in \mathsf{Field}} \{(\ell, \mathsf{f}) \mapsto \ell\} \end{array} \right\}$$

$$[\![x := y]\!]\ \langle s :: S, h, L\rangle \triangleq \{[\langle x := y, s[x \mapsto s(y)] :: S, h, L\rangle]\}$$

$$\boxed{[\![\texttt{lock}()]\!]}\ \langle S, h, L\rangle \triangleq \{[\langle \texttt{lock}(), S, h, \mathrm{add}(L, 1)\rangle]\}$$

$$[\![\mathsf{unlock}()]\!]\ \langle S, h, L\rangle \triangleq \begin{cases} \emptyset & \text{if } L \leq 0 \\ \{[\langle \mathsf{unlock}(), S, h, L - 1\rangle]\} & \text{otherwise} \end{cases}$$

$$[\![\mathsf{pop}()]\!]\ \langle s :: S, h, L\rangle \triangleq \{[\langle \mathsf{pop}(), S, h, L\rangle]\}$$

Concrete semantics of commands

stack

heap

$$[\langle c, S, h, L\rangle]$$

command

# of locks

$$[\![\texttt{skip}]\!]\ \langle S, h, L\rangle \triangleq \{\epsilon\}$$

empty trace

top stack frame

value of var *x*

$$[\![\pi := x]\!]\ \langle s :: S, h, L\rangle \triangleq \begin{cases} \emptyset & \text{if } \lfloor \pi \rfloor_{s,h} \notin \mathrm{dom}\,(h) \\ \{[\langle \pi := x, s :: S, h[\lfloor \pi \rfloor_{s,h} \mapsto s(x)], L\rangle]\} & \text{otherwise} \end{cases}$$

location pointed by *π* given stack *s* & heap *h*

$$[\![\texttt{lock}()]\!]\ \langle S, h, L\rangle \triangleq \{[\langle \texttt{lock}(), S, h, \mathrm{add}(L, 1)\rangle]\}$$

Concrete semantics of compound statements

(1) run *C*, get all its traces

(2) take the last state of each trace

$$[\![C;c]\!]\ \langle S,h,L\rangle \triangleq \left\{ \tau \mathbin{+\!\!+} \tau' \;\middle|\; \begin{array}{l} \tau \in [\![C]\!]\ \langle S,h,L\rangle, \langle S',h',L'\rangle = \mathrm{last}(\tau), \\ \tau' \in [\![c]\!]\ \langle S',h',L'\rangle \end{array} \right\}$$

(4) glue traces of *C* and *c* together

(3) run *c* from the last state, get its traces

Concrete trace example

Stack s0

a	p1

memory location

Heap h0

(p1, f)	p2
(p2, _)	666

Heap h1

(p1, f)	p2
(p2, _)	5

```
lock()
a.f := 5
unlock()
```

program

no lock

Initial state:

⟨skip, s0, h0, 0⟩

```
[
 ⟨lock(),   s0, h0, 1⟩,
 ⟨a.f:=5,   s0, h1, 1⟩,
 ⟨unlock(), s0, h1, 0⟩
]
```

execution trace

Two-threaded program $C_1 \parallel C_2$: concrete semantics

- State

$$\langle c_{\parallel}, (S_1, S_2), h, (L_1, L_2) \rangle$$

$c \parallel \varepsilon$ or $\varepsilon \parallel c$

- Trace

$$\tau^{\parallel} = [\varsigma_0^{\parallel}, \ldots, \varsigma_n^{\parallel}]$$

- Concrete semantics

$$[\![C_1 \parallel C_2]\!]\, \varsigma^{\parallel} \in \mathcal{P}(\mathcal{T}^{\parallel})$$

2-threaded program interleaves single traces

$$[\![C_1 \parallel C_2]\!] \ \langle (S_1, S_2), h, (L_1, L_2) \rangle$$

$$\triangleq$$

$$\bigcup_{\substack{\tau_1 \in [\![C_1]\!] \ \langle S_1, h, L_1 \rangle \\ \tau_2 \in [\![C_2]\!] \ \langle S_2, h, L_2 \rangle}} \text{interl } \tau_1 \ \tau_2 \ \langle (S_1, S_2), h, (L_1, L_2) \rangle$$

(2) interleave all individual traces (full and partial)

(1) run components individually

Concurrent traces example

```
lock()
x := 5
unlock()
print(1)
```

interleave
(taking care of locks)

```
lock()
x := 777
unlock()
print(2)
```

```
lock()   ∥ ε
x := 5   ∥ ε
unlock() ∥ ε
print(1) ∥ ε
ε ∥ lock()
ε ∥ y := 777
ε ∥ unlock()
ε ∥ print(2)
```

```
lock()
x := 5
unlock()
lock()
y := 777
unlock()
print(1)
```

```
lock()
y := 777
unlock()
print(2)
lock()
...
```

~~lock()~~
~~x := 5~~
~~lock()~~

...

Data race means concurrent access to location

Definition 3.6 (Data Race). The program $C_1 \parallel C_2$ *races* if there exists a state ς_0 and a non-empty concurrent trace $\tau \in [\![C_1 \parallel C_2]\!]\ \varsigma_0$ such that $\mathrm{last}(\tau) = \langle \_, (s_1 :: \_, s_2 :: \_), h, \_ \rangle$[9] and,

- there exist paths π_1, π_2 such that $\lfloor \pi_1 \rfloor_{s_1,h} = \lfloor \pi_2 \rfloor_{s_2,h}$;
- there exist states $\varsigma_1 = \langle c_1 \parallel \epsilon, \_, \_, \_ \rangle$ and $\varsigma_2 = \langle \epsilon \parallel c_2, \_, \_, \_ \rangle$ such that $\tau :: \varsigma_1, \tau :: \varsigma_2 \in [\![C_1 \parallel C_2]\!]\ \varsigma_0$;
- $c_1 = (\pi_1 := \_) \wedge c_2 = (\pi_2 := \_)$, or, $c_1 = (\pi_1 := \_) \wedge c_2 = (\_ := \pi_2)$, or, $c_1 = (\_ := \pi_1) \wedge c_2 = (\pi_2 := \_)$.

Data race means concurrent access to location

```
print(1)
lock()
a.f := 5
unlock()
```

```
print(1) ∥ ε
       ε ∥ print(2)
lock()   ∥ ε
a.f := 5 ∥ ε
```

...

```
print(1) ∥ ε
       ε ∥ print(2)
lock()   ∥ ε
ε ∥ y := a.f
```

no lock!

```
print(2)
y := a.f
```

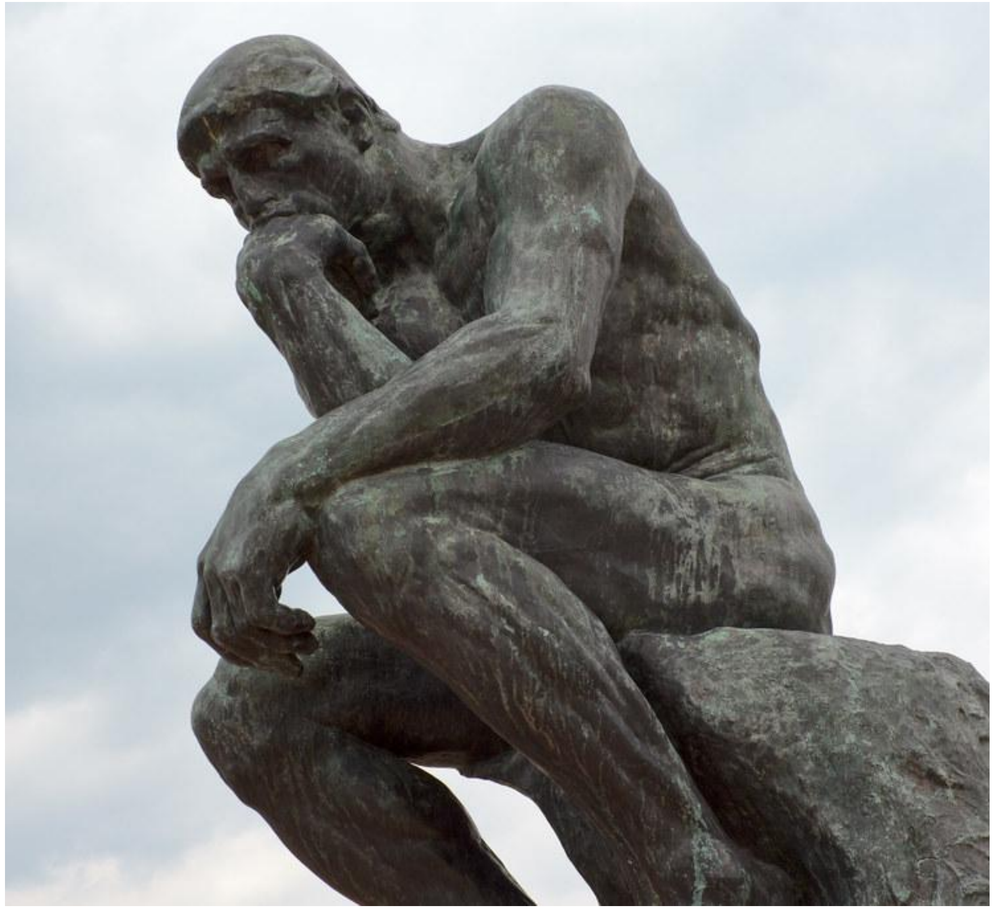
```
print(1) ∥ ε
       ε ∥ print(2)
lock()   ∥ ε
```

```
a.f := 5 ∥ ε
```

```
ε ∥ y := a.f
```

concurrent `Write ∥ Read` to the same location

Can we identify a data race *without* building the traces?



Abstract Semantics

tracks accesses
to memory locations

helps identify
true races

- Abstract State $\langle W, L, A \rangle$
  (wobblies, locks, path accesses)

- Abstraction of a set of
  concrete *single*-threaded traces

$$\alpha(T) \quad = \quad \langle W, L, A \rangle$$

Wobblies can evade data races (produce false positives)

```
class Bloop {
  public int f = 1;
}
```

```
class Burble {

  public void meps(Bloop b) {
    synchronized (this) {
      System.out.println(b.f);
    }
  }

  public void reps(Bloop b) {
    b.f = 42;
  }

  public void beps(Bloop b) {
    b = new Bloop();
    b.f = 239;
  }
}
```

same path `b.f` refers to different locations

Path prefix **b** is *"unstable"* (*"wobbly"*), as it's reassigned, hence race is evaded.

Abstraction keeps track of accesses and wobblies

τ without heap, stack, locks

substitution (needed for method calls)

$$\text{exec } \widehat{\tau} \ (\langle W, L, A\rangle, \bar{\theta})$$

(3) mark x & π as wobbly

(2) remember access to π

$$\text{exec } \widehat{\tau} :: \boxed{\langle \pi := x\rangle} (\langle W, L, A\rangle, \bar{\theta}) \triangleq \left(\left\langle W' \cup \{x, \pi\}^{\bar{\theta}'}, L', A' \cup \{\langle \pi := x, L'\rangle\}^{\bar{\theta}'}\right\rangle, \bar{\theta}'\right)$$

$$\text{where } (\langle W', L', A'\rangle, \bar{\theta}') \triangleq \text{exec } \widehat{\tau} \ (\langle W, L, A\rangle, \bar{\theta})$$

(1) abstract the beginning of the trace

abstract set of traces using exec

$$\alpha(T) = \bigsqcup_{\tau \in T} (\text{fst } (\text{exec } \widehat{\tau} \ (\bot_{\mathcal{D}}, \epsilon)))$$

discard substitution

Why reading is wobbly?

a	p1

```
lock()
a.f.n := 5
unlock()
```

```
x := a
// a.f.n
x.f := new ...
y := a.f.n
```

(p1, f)	p2
(p2, n)	p3

(p1, f)	p4
(p2, n)	p3
(p4, n)	p5

same path `a.f.n` refers to different locations, so there is no race

Abstract access captures concrete access

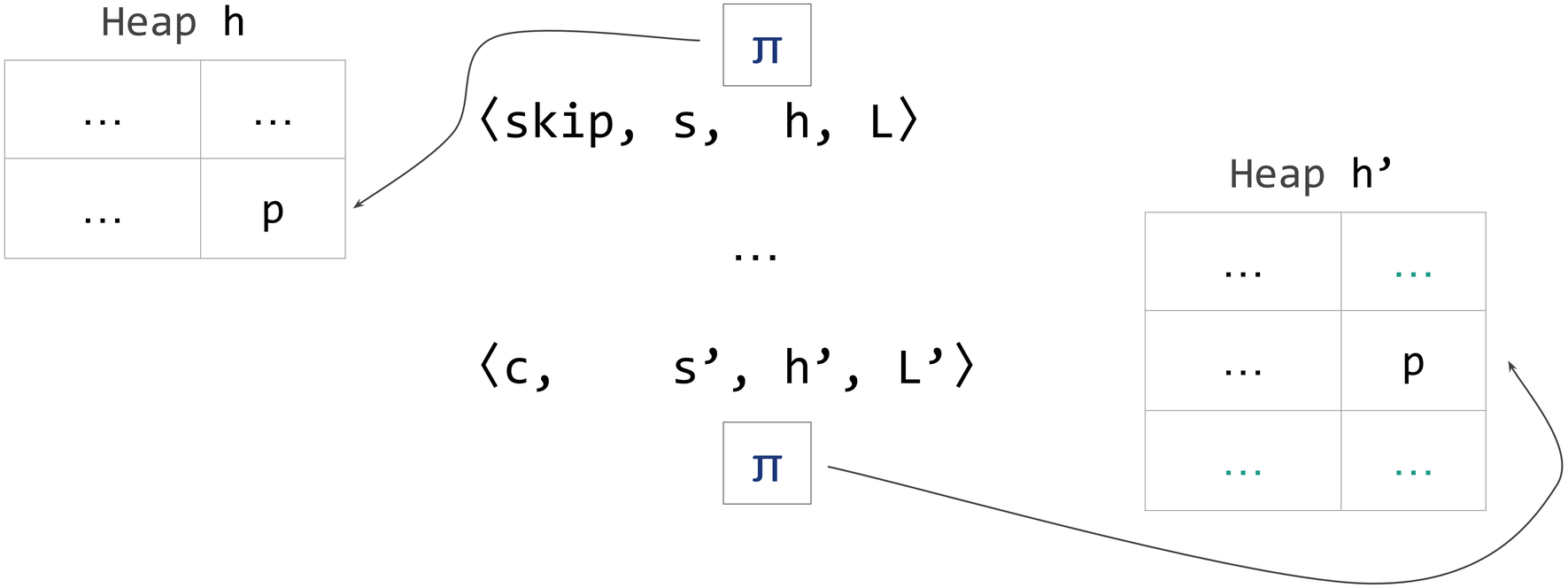
LEMMA 5.6 (PATH ACCESS EXISTENCE). *Let T be a set of traces, $\alpha(T) = \langle \_, \_, A \rangle$, and $q = \langle c, L \rangle$ (where $c = (x := \pi)$ or $c = (\pi := x)$) is a query about the access path π in the locking context L. If $q \in A$ then there exist a trace $\tau \in T$ and a non-empty, shortest prefix $\tau' \preccurlyeq \tau$ such that*

- *the last state of τ' is $\langle c', \_, \_, L \rangle$ and c', c are both stores or loads;*
- exec $\widehat{\tau'}\ (\bot_{\mathcal{D}}, \epsilon) = (\langle \_, \_, A \rangle, \bar{\theta})$ *where $\pi \in A$;*
- *a path π' such that $c' = (x := \pi')$ or $c' = (\pi' := x)$, and $\{\pi\} = \{\pi'\}^{\bar{\theta}}$.*

If a path access is recorded in the abstract state, there is a concrete trace exhibiting the access

Stable path preserves memory location

If a good path is not wobbly,
it preserves memory location along a trace



Static Analysis

operates in abstract domain

enjoys benefits of the abstraction

- Does not need traces

$$[\![C; c]\!]^{\#} \langle W, L, A \rangle$$

- Compositional

$$[\![C; c]\!]^{\#} \langle W, L, A \rangle = [\![c]\!]^{\#}([\![C]\!]^{\#} \langle W, L, A \rangle)$$

- Complete wrt. abstraction

$$[\![C]\!]^{\#} \bot_{\mathcal{D}} = \alpha \left([\![C]\!] \langle S, h, 0 \rangle \right)$$

Galois connection

True Positives Theorem

$[\![C_1]\!]^{\#} \bot_{\mathcal{D}}$

access the same path π
in W||W or W||R or R||W
and π is not wobbly

$[\![C_2]\!]^{\#} \bot_{\mathcal{D}}$

$$\exists\, \varsigma^{\|}.\, \exists \text{ racy } \tau^{\|} \in [\![C_1 \parallel C_2]\!]^{\#}\, \varsigma^{\|}$$

π refers to the same location in C_1 & C_2 in the initial state,
and it still refers to the same location when concurrently accessed

Evaluation

What is the price to pay for having the TP Theorem?

(Reporting *no bugs whatsoever* is TP-Sound)

RacerD *vs* RacerDX

Target	LOC	D CPU	DX CPU	CPU ±%	D Reps	DX Reps	Reps ±%
avrora	76k	103	102	0.4%	143	92	36%
Chronicle-Map	45k	196	196	0.1%	2	2	0%
jvm-tools	33k	106	109	-3.6%	30	26	13%
RxJava	273k	76	69	9.2%	166	134	19%
sunflow	25k	44	44	-1.4%	97	42	57%
xalan-j	175k	144	137	5.0%	326	295	10%

RacerD *vs* RacerDX

Target	LOC	D CPU	DX CPU	CPU ±%	D Reps	DX Reps	Reps ±%
avrora	76k	103	102	0.4%	143	92	36%
Chronicle-Map	45k	196	196	0.1%	2	2	0%
jvm-tools	33k	106	109	-3.6%	30	26	13%
RxJava	273k	76	69	9.2%	166	134	19%
sunflow	25k	44	44	-1.4%	97	42	57%
xalan-j	175k	144	137	5.0%	326	295	10%

RacerD *vs* RacerDX

Target	LOC	D CPU	DX CPU	CPU ±%	D Reps	DX Reps	Reps ±%
avrora	76k	103	102	0.4%	143	92	36%
Chronicle-Map	45k	196	196	0.1%	2	2	0%
jvm-tools	33k	106	109	-3.6%	30	26	13%
RxJava	273k	76	69	9.2%	166	134	19%
sunflow	25k	44	44	-1.4%	97	42	57%
xalan-j	175k	144	137	5.0%	326	295	10%

The Artifact

At Glance

Contents of Artifact

- Data: 6 packages source code
- Facebook's Infer package (OCaml code, Git repo): holds RacerD
- RacerDX Patch
- Set of Bash scripts to:
  - clean up
  - run vanilla Infer
  - patch and ran patched Infer,
  - collect stats.
- README: dependencies, entry points to run scripts, etc.

6 packages (incl. 2 invalid) by Build Technology

- Ant: 3 pkgs (avrora, sunflow, xalan-j)
- Gradle: 1 pkg (RxJava)
- Maven: 2 pkgs (Chronicle-Map, jvm-tools) — the invalid ones

Reproduction & reanalysis

War Stories

Repetition

- First try — failed with too new Java (noted in README):
  - `error: as of release 9, '_' is a keyword, and may not be used as an identifier`
- Second try — failed: unrecognized parameter to cloc (not noted in the README).
- Third try — partial success: numbers for RacerD are slightly off — ???
  - Reason: Missing native dependency
- Finally, numbers for 4 packages did reproduce.

What’s Wrong with Maven?

Authors' words (README)

*Since submission of the paper for review, the sources of two of the projects (**`Chronicle-Map`** and **`jvm-tools`**) we used for evaluation became uncompilable (due to how maven works -- it* **always downloads dependencies from the internet**, *and it seems the newer versions are breaking the build of the version we originally tested).*

Dependencies? This should have to do with the **build scripts**!

pom.xml of Chronicle-map

Release

```
29  <version>3.16.0</version>
30  <name>OpenHFT/Chronicle-Map</name>
31  <description>Chronicle-Map</description>
32  <packaging>bundle</packaging>
33
34  <dependencyManagement>
35      <dependencies>
36
37          <dependency>
38              <groupId>net.openhft</groupId>
39              <artifactId>third-party-bom</artifactId>
40              <type>pom</type>
41              <version>3.6.5</version>
42              <scope>import</scope>
43          </dependency>
44
45          <dependency>
46              <groupId>net.openhft</groupId>
47              <artifactId>chronicle-bom</artifactId>
48              <version>1.16.134</version>
49              <type>pom</type>
50              <scope>import</scope>
```

Artifact

```
29  <version>3.16.0-SNAPSHOT</version>
30  <name>OpenHFT/Chronicle-Map</name>
31  <description>Chronicle-Map</description>
32  <packaging>bundle</packaging>
33
34  <dependencyManagement>
35      <dependencies>
36
37          <dependency>
38              <groupId>net.openhft</groupId>
39              <artifactId>third-party-bom</artifactId>
40              <type>pom</type>
41              <version>3.6.2</version>
42              <scope>import</scope>
43          </dependency>
44
45          <dependency>
46              <groupId>net.openhft</groupId>
47              <artifactId>chronicle-bom</artifactId>
48              <version>LATEST</version>
49              <type>pom</type>
50              <scope>import</scope>
```

Remember: “**always downloads dependencies from the internet**”…

pom.xml of jvm-tools (I)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

<!--

    Copyright 2012 Alexey Ragozin

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

-->

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.gridkit.lab</groupId>
    <artifactId>grid-lab-pom</artifactId>
    <version>2</version>
  </parent>

  <groupId>org.gridkit.jvmtool</groupId>
  <artifactId>jvmtool-umbrella-pom</artifactId>
  <version>0.11</version>
  <name>${project.groupId}::${project.artifactId}</name>
```

Release

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.gridkit.lab</groupId>
    <artifactId>grid-lab-pom</artifactId>
    <version>2</version>
  </parent>

  <groupId>org.gridkit.jvmtool</groupId>
  <artifactId>jvmtool-umbrella-pom</artifactId>
  <version>0.11-SNAPSHOT</version>
  <name>${project.groupId}::${project.artifactId}</name>
```

Artifact

pom.xml of jvm-tools (II)

```
223 </project>
```

```
208 <profiles><profile>
209       <id>infer-capture</id>
210       <build>
211         <plugins>
212           <plugin>
213             <groupId>org.apache.maven.plugins</groupId>
214             <artifactId>maven-compiler-plugin</artifactId>
215             <configuration>
216               <compilerId>javac</compilerId>
217               <forceJavacCompilerUse>true</forceJavacCompilerUse>
218               <fork>true</fork>
                  <executable>/data/users/nikosgorogiannis/infer/infer/bin/infer</exe
219 cutable>
220             </configuration>
221           </plugin>
222         </plugins>
223       </build>
224     </profile></profiles></project>
```

Infer's bug in management of `pom.xml`

- Error message like the one in the artifact (when enabling `stderr`):

```
Error while running epilogue restoring Maven's pom.xml to its original state:
(Unix.Unix_error "No such file or directory"
  rename "((src /data/videoRecorder/videoRecorder-rpm/pom.xml.infer-orig)
      (dst /data/videoRecorder/videoRecorder-rpm/pom.xml))").
```

- There's a fix also!

The Fix

facebook / infer

Watch 562 | Star 10.3k | Fork 1.4k

Code | Issues 213 | Pull requests 6 | Actions | Projects 0 | Wiki | Security | Insights

[epilogues] do not rely on `at_exit`

Browse files

```
Summary:
For some unexplained reason, some of the functions registered in the Epilogues would sometimes be executed several
times. I could not figure out why.

This diff fixes that, but also has more explainable benefits:
- Do not run epilogues registered in the parent in the children. Previously it
  would do so, but probably only if the children registered some epilogue given
  that `at_exit` must be called again once on the child (but the value of the ref
  in `Pervasives` would not have been reset).
- Unified behaviour for early and late epilogues given that we now handle both of these directly

We already have all the control needed to run epilogues when needed: we know
when infer exits, and we know when children processes exit.

Reviewed By: mbouaziz

Differential Revision: D9752046

fbshipit-source-id: 13af40081
```

master | v0.17.0 v0.16.0

jvillard authored and **facebook-github-bot** committed on Sep 12, 2018

1 parent 4ddbc71 commit 817f83972cacba03bd9613979fd55154a6b95e20

Happy End with Repetition

After

- Applying the Infer fix (*kudos to the authors for preserving the Git repo*)
- Checking out released versions of Maven-based packages

We were able to get the numbers from the paper.

Our Experiments (mostly support the claims)

- Full `aws-sdk-java` died with disk overflow (hundreds of gigabytes of reports)
  - Just one module (`aws-java-sdk-s3`):
    - test-aws-sdk-java, 3’847’035, 666, 639, 64, 48
- `spring-kafka` — success, equal results:
  - test-spring-kafka, 30’461, 31, 31, 16,16
- `azkaban` — success, equal, zeroes:
  - test-azkaban, 76’156, 0, 0, 0, 0

Race Report Example


```
{
  "bug_class": "PROVER",
  "kind": "ERROR",
  "bug_type": "THREAD_SAFETY_VIOLATION",
  "qualifier": "Unprotected write. Non-private method `void EmbeddedKafkaBroker.afterPropertiesSet()` writes to field `this.org.springframework.kafka.test.EmbeddedKafkaBroker.zkConnect` outside of synchronization
    .\n Reporting because another access to the same memory occurs on a background thread, although this access may not.",
  "severity": "HIGH",
  "visibility": "user",
  "line": 267,
  "column": -1,
  "procedure": "void EmbeddedKafkaBroker.afterPropertiesSet()",
  "procedure_id": "org.springframework.kafka.test.EmbeddedKafkaBroker.afterPropertiesSet():void.9fb246b16e74937d3be8f9fd6f1d35ed",
  "procedure_start_line": 0,
  "file": "spring-kafka-test/src/main/java/org/springframework/kafka/test/EmbeddedKafkaBroker.java",
  "bug_trace": [
    {
      "level": 0,
      "filename": "spring-kafka-test/src/main/java/org/springframework/kafka/test/EmbeddedKafkaBroker.java",
      "line_number": 267,
      "column_number": -1,
      "description": "<Write on unknown thread>"
    },
    {
      "level": 0,
      "filename": "spring-kafka-test/src/main/java/org/springframework/kafka/test/EmbeddedKafkaBroker.java",
      "line_number": 267,
      "column_number": -1,
      "description": "access to `this.org.springframework.kafka.test.EmbeddedKafkaBroker.zkConnect`"
    },
    {
      "level": 0,
      "filename": "spring-kafka-test/src/main/java/org/springframework/kafka/test/EmbeddedKafkaBroker.java",
      "line_number": 415,
      "column_number": -1,
      "description": "<Write on background thread>"
    },
    {
      "level": 0,
      "filename": "spring-kafka-test/src/main/java/org/springframework/kafka/test/EmbeddedKafkaBroker.java",
      "line_number": 415,
      "column_number": -1,
      "description": "access to `this.org.springframework.kafka.test.EmbeddedKafkaBroker.zkConnect`"
    }
  ],
  "key": "EmbeddedKafkaBroker.java|afterPropertiesSet|THREAD_SAFETY_VIOLATION",
  "node_key": "9c5d6d9028928346cc4fb44cced5dea1",
  "hash": "69130a79005ccdc6e56b21690faa2f1a",
  "bug_type_hum": "Thread Safety Violation",
  "censored_reason": "",
  "access": "hJWmvgAAAdwAAAAzAAAA5QAAALSwkNAyYWZ0ZXJQcm9wZXJ0aWVzU2V0QJOgCTJvcmcuc3ByaW5nZnJhbWV3b3JrLmthZmthLnRlc3QuRW1iZWRkZWRLYWZrYUJyb2tlckCQoEAkdm9pZECgoJKgIECwAQELAP
    +SCVdzcHJpbmcta2Fma2EtdGVzdC9zcmMvbWFpbi9qYXZhL29yZy9zcHJpbmdmcmFtZXdvcmsva2Fma2EvdGVzdC9FbWJlZGRlZEthZmthQnJva2VyLmphdmGRoKCRsAI2n0wXoCR0aGlzQJCQ0DJhZnRlclByb3BlcnRpZXNTZXRAk6AJMm9yZy5zcHJpbmdmcmFtZXdvcmsua2
    Fma2EudGVzdC5FbWJlZGRlZEthZmthQnJva2VyQJCgQCR2b2lkQKCjoJSToAQKQLBAQEBABAGgkZEJPG9yZy5zcHJpbmdmcmFtZXdvcmsua2Fma2EudGVzdC5FbWJlZGRlZEthZmthQnJva2VyLnprQ29ubmVjdECgBCCgsAEBnwD
    /kglXc3ByaW5nLWthZmthLXRlc3Qvc3JjL21haW4vamF2YS9vcmcvc3ByaW5nZnJhbWV3b3JrL2thZmthL3Rlc3QvRW1iZWRkZWRLYWZrYUJyb2tlci5qYXZhQA=="
},
```

RacerDX


```
{
  "bug_class": "PROVER",
  "kind": "ERROR",
  "bug_type": "THREAD_SAFETY_VIOLATION",
  "qualifier": "Read/Write race. Non-private method `JPanel GraphNumbers.chalkboardAndBar()` indirectly reads with synchronization from `this.avrora.gui.GraphNumbers.privateNumbers.cck.stat.Sequence.total`. Potentially races with unsynchronized write in method `GraphNumbers.internalUpdate()`.\n Reporting because another access to the same memory occurs on a background thread, although this access may not.",
  "severity": "HIGH",
  "visibility": "user",
  "line": 123,
  "column": -1,
  "procedure": "JPanel GraphNumbers.chalkboardAndBar()",
  "procedure_id": "avrora.gui.GraphNumbers.chalkboardAndBar():javax.swing.JPanel.ecda2ba0ceefadb1302bcf0ca8e35bca",
  "procedure_start_line": 0,
  "file": "src/avrora/gui/GraphNumbers.java",
  "bug_trace": [
    {
      "level": 0,
      "filename": "src/avrora/gui/GraphNumbers.java",
      "line_number": 123,
      "column_number": -1,
      "description": "<Read trace>"
    },
    {
      "level": 0,
      "filename": "src/avrora/gui/GraphNumbers.java",
      "line_number": 123,
      "column_number": -1,
      "description": "call to void GraphNumbers.updateHorzBar()"
    },
    {
      "level": 1,
      "filename": "src/avrora/gui/GraphNumbers.java",
      "line_number": 151,
      "column_number": -1,
      "description": "call to int Sequence.size()"
    },
    {
      "level": 2,
      "filename": "src/cck/stat/Sequence.java",
      "line_number": 157,
      "column_number": -1,
      "description": "access to `this.cck.stat.Sequence.total`"
    },
```


RacerD


```
{
  "bug_class": "PROVER",
  "kind": "ERROR",
  "bug_type": "THREAD_SAFETY_VIOLATION",
  "qualifier": "Read/Write race. Non-private method `JPanel GraphNumbers.chalkboardAndBar()` indirectly reads with synchronization from `this.avrora.gui.GraphNumbers.privateNumbers.cck.stat.Sequence.total`. Potentially races with unsynchronized write in method `GraphNumbers.internalUpdate()`.\n Reporting because another access to the same memory occurs on a background thread, although this access may not.",
  "severity": "HIGH",
  "visibility": "user",
  "line": 123,
  "column": -1,
  "procedure": "JPanel GraphNumbers.chalkboardAndBar()",
  "procedure_id": "avrora.gui.GraphNumbers.chalkboardAndBar():javax.swing.JPanel.ecda2ba0ceefadb1302bcf0ca8e35bca",
  "procedure_start_line": 0,
  "file": "src/avrora/gui/GraphNumbers.java",
  "bug_trace": [
    {
      "level": 0,
      "filename": "src/avrora/gui/GraphNumbers.java",
      "line_number": 123,
      "column_number": -1,
      "description": "<Read trace>"
    },
    {
      "level": 0,
      "filename": "src/avrora/gui/GraphNumbers.java",
      "line_number": 123,
      "column_number": -1,
      "description": "call to void GraphNumbers.updateHorzBar()"
    },
    {
      "level": 1,
      "filename": "src/avrora/gui/GraphNumbers.java",
      "line_number": 151,
      "column_number": -1,
      "description": "call to int Sequence.size()"
    },
    {
      "level": 2,
      "filename": "src/cck/stat/Sequence.java",
      "line_number": 157,
      "column_number": -1,
      "description": "access to `this.cck.stat.Sequence.total`"
    },
```


Report Subsets

Takeaways: How NOT To Make an Artifact

- No Environment Management (e.g. a VM, Docker, Nix, etc.):
  a. a bunch of source codes (sometimes non-released versions; not tracked by a VCS, e.g. Git)
  b. (lose) description of dependencies (some dependencies didn't have corresponding versions, e.g. `cloc`)
  c. No way to account for transitive deps of tools, esp. native deps (e.g. `sqlite3-dev`)
- Clearing the `$PATH` -- poor man's env management
- Piping `stdout` and `stderr` (!!!) to `/dev/null`