Incorrectness Logic and Underapproximation: Foundations of Bug Catching

Quang Loc Le

University College London - loc.le@ucl.ac.uk

Formosan Summer School on Logic, Language, and Computation
National Taiwan University - August 29, 2023

Bio

- Lecturer in Programming Principles, Logic, and Verification Group (PPLV), UCL
- Some experience in industry
- Research topics:
  - Theory: Program Analysis, Formal Verification
    - Hoare logic, separation logic, incorrectness logic, string logics
    - Induction prooofs, cyclic proofs
  - Application: Finding bugs in big codebase, smart contracts

This talk

- Incorrectness logic
- Separation logic and Incorrectness separation logic
- Pulse-X: Finding real bugs in big programs (OOPSLA 2022)
  - Deployed as a gatekeeper at Facebook/Meta
  - Was recipient of ACM SIGPLAN Distinguished Paper Award!

State of the art: Correctness

Most focus on reasoning for proving correctness

- Prove the absence of bugs
- To deal with undecidablity: over-approximate reasoning
- For scalability
  - technique: compositionality
    - codebases: reasoning about incomplete components
    - resources accessed: spatial locality
  - support large codebases and large teams

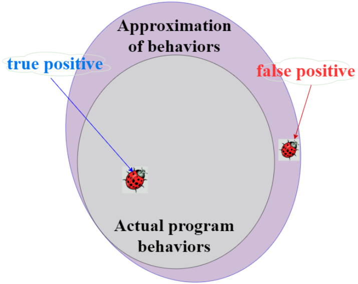
Hoare logic

Hoare triple:

$$\{P\}\, c\, \{Q\} \qquad \textit{iff} \qquad post(c)P \subseteq Q$$

For all states s in P, if running c on s terminates in s', then s' is in Q.

Q over-approximates post(c)P



First Axiom of a Bug Catching Tool at Scale

“Don’t Spam the Developers!”

Peter O’Hearn

Peter O’Hearn: co-founder of separation logic and co-founder of Infer @ Facebook/Meta

First Axiom of a Bug Catching Tool at Scale

Incorrectness logic:
A formal foundation for bug finding

Peter O'Hearn

Incorrectness Logic. Peter O'Hearn. POPL 2020

Incorrectness logic

Hoare triple:

$$\{P\}\, c\, \{Q\} \qquad \textit{iff} \qquad post(c)P \subseteq Q$$

Q over-approximates post(c)P

Under-approximate triple:

$$[P]\, c\, [Q] \qquad \textit{iff} \qquad post(c)P \supseteq Q$$

For all states s in Q, s can be reached by running $\boldsymbol{c}$ on some s' in P.

Q under-approximates post(c)P

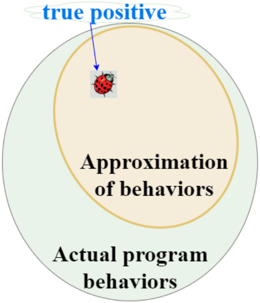
Incorrectness logic

Under-approximate triple:

$$[P]\; c\; [Q] \qquad \textit{iff} \qquad post(c)P \supseteq Q$$

For all states s in Q, s can be reached by running $\boldsymbol{c}$ on some s' in P.

Q under-approximates post(c)P



Incorrectness triple

Under-approximate triple

$[P]\ c\ [Q]$ *iff* $post(c)P \supseteq Q$

For all states s in Q, s can be reached by running $\boldsymbol{c}$ on some s' in P

Incorrectness triple

$[P]\ c\ [\epsilon : Q]$

ϵ: exit condition

- [*ok*: normal execution]
- [*er*: erroneous execution]

Incorrectness triple: Example

Example 1:

$$[x = 2]\, \mathtt{x = x + 1}\, [ok\text{:}\ x = 3]$$

Example 2:

$$[x = 2]\, \mathtt{assert(x > 3)}\, [er\text{:}\ x = 2]$$

Incorrectness logic: Summary

- Under-approximate analogue of Hoare logic
- Formal foundation for bug finding
- Reading: Incorrectness Logic. Peter O'Hearn. POPL 2020.
- Next: Incorrectness separation logic
  - Compositionality
  - memory safety bugs (e.g., null pointer dereference, use-after-free, memory leak)

An extension of Hoare logic for heap-manipulating programs with aliasing.

Memory bugs: null reference, memory leak, buffer overrun, double free

Tony Hoare

Null References: The Billion Dollar Mistake

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

@QCon - Aug 25, 2009 [1]

[1] https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

Separation Logic: Hoare logic for pointers

Syntax: $\kappa \wedge \pi$: heap formula κ and pure formula π

Semantics:

$$\text{program states} = \{(s, h) \mid s : Var \to Val \;\wedge\; h : Loc \rightharpoonup_{fin} Val^N\}$$

satisfaction relation: $s, h \models \kappa \wedge \pi$

empty heap predicate:

$$s, h \models \texttt{emp} \quad \texttt{iff} \quad dom(h) = \{\}$$

Separation Logic: Hoare logic for pointers

$$\text{states} = \{(s, h) \mid s : Var \to Val \ \wedge \ h : Loc \rightharpoonup_{fin} Val^N\}$$

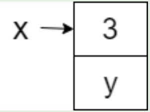
points-to predicate:

Example

```
struct node {int val; node *next}
```

$s,h \models x \mapsto c(3, y)$

$s = \{(x, 2), (y, 7)\}$
$h = \{(2, (3, 7))\}$

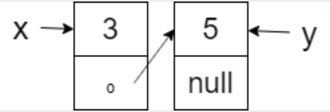


Separation Logic: Hoare logic for pointers

separating conjunction:

Example

$x \mapsto node(3, y) * y \mapsto c(5, \texttt{null})$



Note:

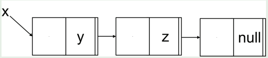
$$x \mapsto node(\_) * x \mapsto node(\_) \equiv \texttt{false}$$

Separation Logic: Hoare logic for pointers

inductive definitions:

Example

Singly-linked list



$$
\begin{array}{lcl}
\texttt{emp} \wedge \texttt{root} = \texttt{null} & \Rightarrow & \mathit{list}(\texttt{root}) \\
\exists\, d, r.\ \texttt{root} \mapsto \mathit{node}(d, r) * \mathit{list}(r) & \Rightarrow & \mathit{list}(\texttt{root})
\end{array}
$$

Separation Logic: axioms

- ALLOC

$$\{\mathrm{emp}\} \quad x = \mathit{alloc}() \quad \{x \mapsto \_\}$$

- FREE

$$\{x \mapsto \_\} \quad \mathit{free}(x) \quad \{\mathrm{emp}\}$$

- READ

$$\{x \mapsto v\} \quad y = [x] \quad \{x \mapsto v \land y = v\}$$

- WRITE

$$\{x \mapsto \_\} \quad [x] = v \quad \{x \mapsto v\}$$

Separation logic

Two advantages:

1. Separating conjunction:
   $\mathtt{x} = \mathtt{malloc}(...); \mathtt{y} = \mathtt{malloc}(...); \mathtt{z} = \mathtt{malloc}(...)$

$$\{...\}$$
$$[x] := 1;$$
$$[y] := 2;$$
$$[z] := 3;$$
$$\{...\}$$

Separation logic

Two advantages:

1. Separating conjunction:

   $\texttt{x} = \texttt{malloc}(...); \texttt{y} = \texttt{malloc}(...); \texttt{z} = \texttt{malloc}(...)$

$$\{x \neq y \wedge x \neq z \wedge y \neq z\}$$
$$[\mathrm{x}] := 1;$$
$$[\mathrm{y}] := 2;$$
$$[\mathrm{z}] := 3;$$
$$\{x \neq y \wedge x \neq z \wedge y \neq z \wedge h(x) = 1 \wedge h(y) = 2 \wedge h(z) = 3\}$$

3!/2 inequalities

Separation logic

Two advantages:

1. Separating conjunction:
   $\texttt{x} = \texttt{malloc}(...); \texttt{y} = \texttt{malloc}(...); \texttt{z} = \texttt{malloc}(...)$

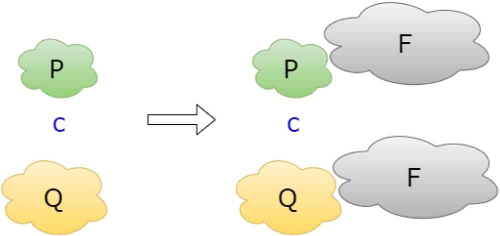
$$\{x \mapsto \_ * y \mapsto \_ * z \mapsto \_\}$$
$$[\mathrm{x}] := 1;$$
$$[\mathrm{y}] := 2;$$
$$[\mathrm{z}] := 3;$$
$$\{x \mapsto 1 * y \mapsto 2 * z \mapsto 3\}$$

Separation logic

Two advantages:

1. Separating conjunction
2. Frame rule

$$\frac{\{P\}\, c\, \{Q\}}{\{P * F\}\, c\, \{Q * F\}}\ \mathrm{Mod(c)} \cap \mathrm{FV(F)} = \emptyset$$



Separation logic

Two advantages:

1. Separating conjunction

2. Frame rule

$$\frac{\{P\}\,c\,\{Q\}}{\{P * F\}\,c\,\{Q * F\}}\ \mathrm{Mod(c)} \ \cap\ \mathrm{FV(F)} = \emptyset$$

$$\frac{\{x \mapsto a\}\,[x] := 1\,\{x \mapsto 1\}}{\{x \mapsto v_1 * y \mapsto v_2 * z \mapsto v_3\}\,[x] = 1\,\{x \mapsto 1 * y \mapsto v_2 * z \mapsto v_3\}}$$

Separation logic

Two advantages:

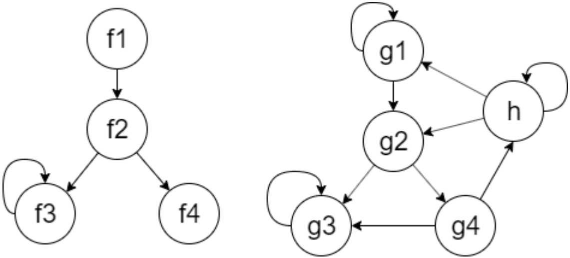
1. Separating conjunction
2. Frame rule

$$\{x \mapsto v_1 * y \mapsto v_2 * z \mapsto v_3\}$$
$$[\mathrm{x}] := 1;$$
$$\{x \mapsto 1 * y \mapsto v_2 * z \mapsto v_3\}$$
$$[\mathrm{y}] := 2;$$
$$\{x \mapsto 1 * y \mapsto 2 * z \mapsto v_3\}$$
$$[\mathrm{z}] := 3;$$
$$\{x \mapsto 1 * y \mapsto 2 * z \mapsto 3\}$$

Compositionality and Scalability

The analysis result of a composite program is defined in terms of the analysis results of its parts and a means of combining them.

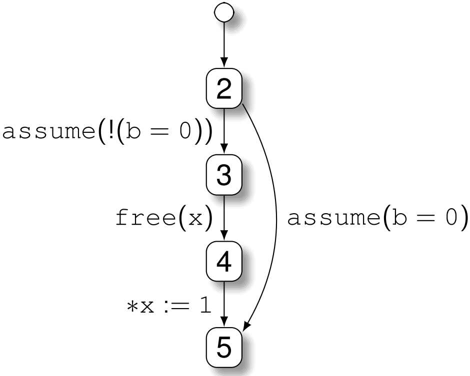
- part: procedures/functions



- analysis result: Hoare triples
- a means: bi-abduction

Analysis problem

```
void f(bool b, int *x){
  if(b){
   free(x);
   *x := 1;
  }
}
```



Given:

- a program: control flow graphs
- specs of atomic procedures and libraries are given

Question:

- find spec of the program

Bi-abduction

For each procedure with code `c`, Infer starts with `emp` as preconidtion, it uses bi-abduction to infer pre/post such that `c` does not contain memory bugs.

Bi-abduction

Over-approximate bi-abduction question:

$$A \ast\ ?M \vdash G \ast\ ?F$$

- $\ast\ ?M$ goes to pre
- $\ast\ ?F$ goes to post

Example:

$$\{y \mapsto v_2 \ast z \mapsto v_3\}$$
$$[\mathrm{x}] := 1;$$
$$\{?\}$$

for safety:

$$\{x \mapsto v_1\}[x] := 1; \{x \mapsto 1\}$$

Bi-abduction query:

$$y \mapsto v_2 \ast z \mapsto v_3 \ \ast\ ?M \vdash x \mapsto v_1 \ \ast\ ?F$$

Infer:

$$\begin{array}{rcl} F & = & y \mapsto v_2 \ast z \mapsto v_3 \\ M & = & x \mapsto v_1 \end{array}$$

Separation logic - Infer

Compositional Shape Analysis by Means of Bi-Abduction (POPL'09)

- analysed Linux Kernel 2.6.25.4 (2.473 MLOC) $<$ 30 mins
- led to Facebook's Infer in 2013[2]

Facebook Acquires Monoidics

MERGERS AND ACQUISITIONS START UP UK

Published on July 18, 2013

Facebook acquired **Monoidics**, a London, UK-based startup that provides a tool for visualizing software quality.

The amount of the deal was not disclosed. Following the close of transaction, the team of the company will join Facebook's office in London.

Founded in 2009 by Italians Dino Distefano (CSO) and Cristiano Calcagno (CTO), and Peter O'Hearn (Scientific Advisor), and led by Bee Lavender (CEO), Monoidics provides INFER, an advanced static code analyzer, which helps users verify their software is bug-free and allows them to focus directly on memory safety and security.

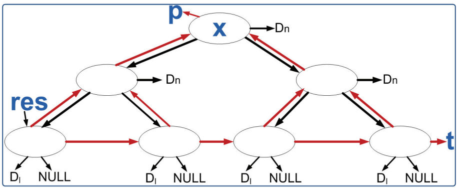
Customers included Airbus, Mitsubishi, ARM, Vanguardistas, and Lawrence Livermore National Laboratory.

[2] http://www.finsmes.com/2013/07/facebook-acquires-monoidics.html

Second-Order Bi-Abduction [3]

One-phase sound analysis for specification inference

- works for arbitrary data structures e.g., tll data structures



[3] Shape Analysis via Second-Order Bi-Abduction. CAV 2014

Separation logic: Summary

- Over-approximate bi-abduction for the absence of memory safety bugs
- Compositionality and scalability
- Reading list:
  - Separation logic: a logic for shared mutable data structures. JC Reynolds. LICS 2002
  - BI as an Assertion Language for Mutable Data Structures. Samin S. Ishtiaq, Peter W. O'Hearn. POPL 2021
  - Local Reasoning about Programs that Alter Data Structures. Peter W. O'Hearn, John C. Reynolds, Hongseok Yang. CSL 2001.
  - Shape Analysis via Second-Order Bi-Abduction. Quang Loc Le, Cristian Gherghina, Shengchao Qin, Wei-Ngan Chin. CAV 2014

Incorrectness triple

$$[P]\ c\ [\epsilon : Q]$$

ϵ: exit condition

- [*ok*: normal execution]
- [*er*: erroneous execution]

Incorrectness separation logic: essential rules

From separation logic:

$$\{x \mapsto \_\} \quad \mathit{free}(x) \quad \{\mathtt{emp}\}$$

to incorrectness separation Logic

$$[x \mapsto \_] \; \mathit{free}(x) \; [\mathit{ok}: \mathtt{emp}]$$

Any problems?

Incorrectness separation logic: essential rules

$$[x \mapsto \_] \; \mathit{free}(x) \; [\mathit{ok}\colon \mathtt{emp}]$$

Problems:

- Post is over-approximated
- Frame rule does not hold

$$\dfrac{\dfrac{[x \mapsto \_] \; \mathit{free}(x) \; [\mathit{ok}\colon \mathtt{emp}]}{[x \mapsto \_ * x \mapsto 1] \; \mathit{free}(x) \; [\mathit{ok}\colon \mathtt{emp} * x \mapsto 1]}\;\text{Frame}}{[\mathtt{false}] \; \mathit{free}(x) \; [\mathit{ok}\colon x \mapsto 1]}\;\text{Conseq}$$

$[P]\; c\; [Q]$ *iff* $\mathit{post}(c)P \supseteq Q$

For all states s in Q, s can be reached by running $\boldsymbol{c}$ on some s' in P.

Incorrectness separation logic: essential rules

Solution: Track deallocated locations

$x \not\mapsto$ means x is de-allocated

$$[x \mapsto \_]\ \mathit{free}(x)\ [ok: x \not\mapsto]$$

$x \not\mapsto * x \not\mapsto \equiv \texttt{false}$ and $x \mapsto \_ * x \not\mapsto \equiv \texttt{false}$

Frame rule trivially hold

$$\text{Conseq}\ \dfrac{\text{Frame}\ \dfrac{[x \mapsto \_]\ \mathit{free}(x)\ [ok: x \not\mapsto]}{[x \mapsto \_ * x \mapsto 1]\ \mathit{free}(x)\ [ok: x \not\mapsto * x \mapsto 1]}}{[\texttt{false}]\ \mathit{free}(x)\ [ok: \texttt{false}]}$$

Incorrectness separation logic: axioms

- FREE

$$[x \mapsto \_]\ \mathit{free}(x)\ [\mathit{ok}\colon x \not\mapsto \_]$$
$$[x \not\mapsto]\ \mathit{free}(x)\ [\mathit{er}\colon x \not\mapsto \_] // \text{double-free}$$
$$[x = \mathtt{null}]\ \mathit{free}(x)\ [\mathit{er}\colon x = \mathtt{null}] // \mathit{NPE}$$

- ALLOC

$$[\mathrm{emp}]\ x = \mathit{alloc}()\ [\mathit{ok}\colon x \mapsto \_]$$

- READ

$$[x \mapsto v]\ y = [x]\ [\mathit{ok}\colon x \mapsto v \wedge y = v]$$
$$[x \not\mapsto]\ y = [x]\ [\mathit{er}\colon x \not\mapsto \_] // \text{use-after-free}$$
$$[x = \mathtt{null}]\ y = [x]\ [\mathit{er}\colon x = \mathtt{null}] // \mathit{NPE}$$

- WRITE

$$[x \mapsto \_]\ [x] = v\ [\mathit{ok}\colon x \mapsto v]$$
$$[x \not\mapsto]\ [x] = v\ [\mathit{er}\colon x \not\mapsto \_] // \text{use-after-free}$$
$$[x = \mathtt{null}]\ [x] = v\ [\mathit{er}\colon x = \mathtt{null}] // \mathit{NPE}$$

Incorrectness separation logic: Summary

- IL $+$ SL for compositional bug finding
- Under-approximate analogue of SL
- Targets memory safety bugs
- New notation for de-allocated locations
- Reading: Local Reasoning About the Presence of Bugs: Incorrectness Separation Logic. Azalea Raad, Josh Berdine, Hoang-Hai Dang, Derek Dreyer, Peter W. O'Hearn, Jules Villard. CAV 2020
- Next:
  - inter-procedural analysis
  - Compositional bug reporting via no-false-positives theorem

Compositional Shape Analysis by Means of Bi-Abduction (POPL'09)

- analysed Linux Kernel 2.6.25.4 (2.473 MLOC) $<$ 30 mins
- led to Facebook's Infer in 2013[4]

Facebook Acquires Monoidics

MERGERS AND ACQUISITIONS START UP UK

Published on July 18, 2013

Facebook acquired **Monoidics**, a London, UK-based startup that provides a tool for visualizing software quality.

The amount of the deal was not disclosed. Following the close of transaction, the team of the company will join Facebook's office in London.

Founded in 2009 by Italians Dino Distefano (CSO) and Cristiano Calcagno (CTO), and Peter O'Hearn (Scientific Advisor), and led by Bee Lavender (CEO), Monoidics provides INFER, an advanced static code analyzer, which helps users verify their software is bug-free and allows them to focus directly on memory safety and security.

Customers included Airbus, Mitsubishi, ARM, Vanguardistas, and Lawrence Livermore National Laboratory.

[4] http://www.finsmes.com/2013/07/facebook-acquires-monoidics.html

Infer 1.0

High number of false positives due to

- Over-approximation
- Using heuristics to report bugs compositionally

Interaction with OpenSSL Developers

Pulse-X found 41 bugs, **15 were unknown previously**

- We committed fixes in pull request #15834

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
  SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                              "prepend cert");

  memset(exc, 0, sizeof(*exc));
  ...
}
```

1. `app_malloc`: is a `malloc` wrapper, and could return null.
2. `memset(exc, 0, ..)` sets heap's content pointed to by *exc* to 0.

Interaction with OpenSSL Developers

Pulse-X found 41 bugs, **15 were unknown previously**

- We committed fixes in pull request #15834

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
  SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                                      "prepend cert");

  memset(exc, 0, sizeof(*exc));
 ...
}
```

1. `app_malloc`: is a `malloc` wrapper, and could return null.
2. `memset(exc, 0, ..)` sets heap's content pointed to by *exc* to 0.

Do you catch the bug?

Interaction with OpenSSL Developers

Pulse-X found 41 bugs, **15 were unknown previously**

- We committed fixes in pull request #15834

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
   SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                                "prepend cert");

   memset(exc, 0, sizeof(*exc));
  ...
}
```

`memset(null,..,..)` causes an null-pointer dereference error.

Interaction with OpenSSL Developers

Pulse-X found 41 bugs, **15 were unknown previously**

- We committed fixes in pull request #15834

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
  SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                               "prepend cert");

+ if(exc == NULL)
+   return 0;
  memset(exc, 0, sizeof(*exc));
  ...
}
```

OpenSSL developer:

False positive, `app_malloc()` *aborts when the allocation fails.*

Interaction with OpenSSL Developers - Error trace

```
apps/lib/s_cb.c:959: error: Nullptr Dereference
  PISL found a potential null pointer dereference on line 959.

apps/lib/s_cb.c:957:23: in call to `app_malloc`
  955. static int ssl_excert_prepend(SSL_EXCERT **pexc)
  956. {
  957.     SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
                           ^

  958.
  959.     memset(exc, 0, sizeof(*exc));

test/testutil/apps_mem.c:16:16: in call to `CRYPTO_malloc` (modelled)
    14. void *app_malloc(size_t sz, const char *what)
    15. {
    16.     void *vp = OPENSSL_malloc(sz);
                       ^


test/testutil/apps_mem.c:16:16: is the null pointer
    14. void *app_malloc(size_t sz, const char *what)
    15. {
    16.     void *vp = OPENSSL_malloc(sz);
                       ^

    17.
    18.     return vp;
...
```

Interaction with OpenSSL Developers - *grep* search

another `app_malloc` in `apps/lib/apps.c`

```
void app_bail_out(char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    BIO_vprintf(bio_err, fmt, args);
    va_end(args);
    ERR_print_errors(bio_err);
    exit(EXIT_FAILURE);
}

void *app_malloc(size_t sz, const char *what) {
    void *vp = OPENSSL_malloc(sz);

    if (vp == NULL)
        app_bail_out("%s: Could not allocate %zu bytes
            for %s\n",
                    opt_getprog(), sz, what);
    return vp;
}
```

Interaction with OpenSSL Developers - accept fix

apps/lib/s_cb.c Outdated Hide resolved

```
... ... @@ -956,6 +956,9 @@ static int ssl_excert_prepend(SSL_EXCERT **pexc)
956 956      {
957 957          SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
958 958
    959 +        if (!exc) {
```

paulidale 13 days ago Contributor

False positive, `app_malloc()` doesn't return if the allocation fails.

lequangloc 13 days ago Author

Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

paulidale 13 days ago Contributor

It would need to look at the link lines or build dependencies to figure out which sources were used.

We should fix the one in `test/testutil/apps_mem.c`.

Then, he created pull request #15836 to commit the fix.

Pulse-X at a glance

Pulse-X: A tool that proves the presence of bugs
(e.g., null pointer dereferences, use-after-frees, leaks, ...)

- Under-approximate bi-abduction
  - using Incorrectness Separation Logic
- Compositional bug reporting mechanism
  - latent vs. manifest errors

Our goals

Pulse-X: A tool that proves the presence of bugs

- Precision
  - *doesn't spam the developers.*
- Scalability
  - 3-dimensional scale: code (large codebases), people (big team), velocity (high frequency of code changes)
  - continuous integration (CI) reasoning
- Adoption

Pulse-X $\Leftarrow$ Infer

Compositional Shape Analysis by Means of Bi-Abduction (POPL'09)

Two concerns:

- Clash with foundations
- Report bugs compositionally

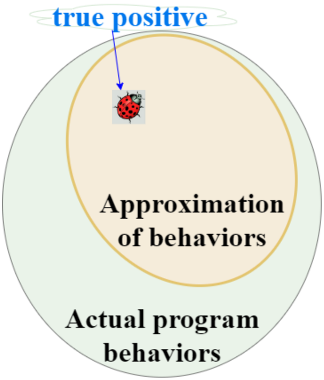
Clash with foundations

Prove the presence of bugs

Under-approximation vs. Over-approximation

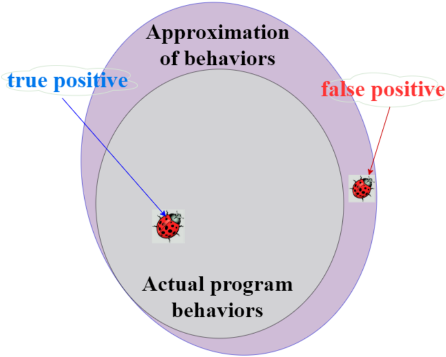
Under-approximate reasoning

- symbolic execution (KLEE), symbolic model checking (CBMC)
- whole-program analysis
- advantages:
  - report true bugs
- disadvantages:
  - not scaled (for CI)
  - memory model: does not support (unbounded) symbolic heaps



Over-approximate reasoning

- compositional reasoning by means of bi-abduction (Infer)
- begin-anywhere analysis
- advantages:
  - scalability
  - memory model: separation logic
- disadvantages:
  - may report false positives



Foundations

Prove the presence of bugs

under-approximate reasoning	over-approximate reasoning
symbolic execution (KLEE), symbolic model checking (CBMC)	compositional reasoning by means of bi-abduction (Infer)
whole-program analysis	begin-anywhere analysis
not scaled	scalability
memory model: does not support (unbounded) symbolic heaps	memory model: separation logic
true bugs	false positives

How to achieve both scalability and precision?

A scalable and precise bug-finding tool

- true bugs and scalability
  1. under-approximate analogue of Infer; or
  2. compositional analogue of KLEE, CBMC
- memory model:
  - under-approximate analogue of separation logic
    - $\Rightarrow$ incorrectness separation logic (CAV'20)

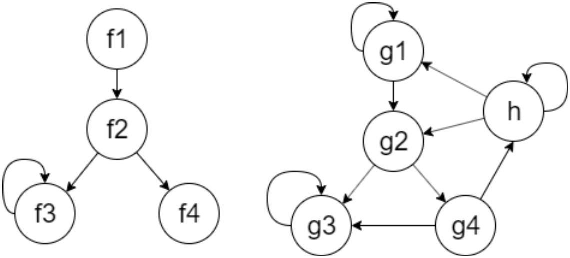
Foundations: our attempt

an under-approximate analogue of Infer using incorrectness separation logic

Compositional reasoning

The analysis result of a composite program is defined in terms of the analysis results of its parts and a means of combining them.

- part: procedures



- analysis result: under-approximate specs i.e., incorrectness triples[5]

- a means: under-approximate bi-abduction

[5] Peter O'Hearn. Incorrectness Logic. POPL'20

Incorrectness triple[6]

Under-approximate triple

$[P]\ c\ [Q]$ *iff* $post(c)P \supseteq Q$

For all states s in Q, s can be reached by running $\boldsymbol{c}$ on some s' in P

Incorrectness triple

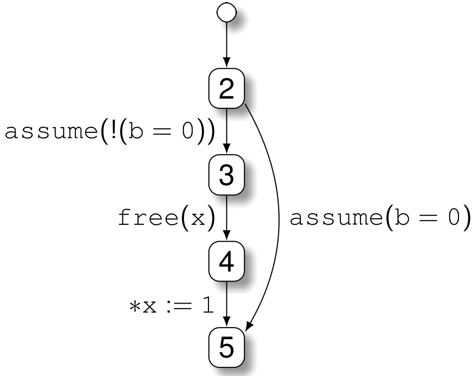
$[P]\ c\ [\epsilon : Q]$

ϵ: exit condition

- [*ok*: normal execution]
- [*er*: erroneous execution]

[6] Peter O'Hearn. Incorrectness Logic. POPL'20

```
void f(bool b, int *x){
  if(b){
    free(x);
    *x := 1;
  }
}
```



Given:

- a program: control flow graphs
- specs of atomic procedures and libraries are given

Question:

- find spec of the program

Under-approximate bi-abduction

Over-approximate bi-abduction question:

$$A * ?M \vdash G * ?F$$

Under-approximate bi-abduction question:

$$A * ?F \vdash G * ?M$$

- abductive inference: find F
- anti-abductive inference: find M

Under-approximate bi-abduction

$$A * ?F \vdash G * ?M$$

- Frame rule

$$\frac{[P]\ c\ [\epsilon : Q]}{[P * F]\ c\ [\epsilon : Q * F]}\ \text{Mod(c)} \cap \text{FV(F)} = \emptyset$$

- Stack-in-heap meory model

Compositional Bug Reporting

Without considering the entire program, how do we know a bug is true?

Do you report a null pointer dereference?

```c
void f(int* x) {
  *x = 42;
}
```

Existing approaches:

- Infer uses heuristics:
  - surfacing failed proofs and bug patterns.
- UC-KLEE uses heuristics with annotations
  - OpenSSL-1.0.2: 11 real bugs / 474 errors found = 2.32%

Compositional Bug Reporting: Pulse-X

$$[x \mapsto X * X \mapsto \_]\ \texttt{f(x)}\ [\textit{ok}\colon x \mapsto X * X \mapsto 42]$$

$$[x \mapsto \texttt{null}]\ \texttt{f(x)}\ [\textit{er}\colon x \mapsto \texttt{null}]$$

$$[x \not\mapsto]\ \texttt{f(x)}\ [\textit{er}\colon x \not\mapsto]$$

Pulse-X classifies *er* triples:

- Manifest bugs: any call to the function will trigger the error.
- Latent bugs: only some calls to the function will trigger the error.

Compositional Bug Reporting: Pulse-X

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
   SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                                "prepend cert");

   memset(exc, 0, sizeof(*exc));
  ...
}
```

Listing 1: OpenSSL null pointer bug in `ssl_excert_prepend`.

Manifest error

- for any value of input `exc`, this error happens.
- any call to `ssl_excert_prepend` will trigger the error.

Compositional Bug Reporting: Pulse-X

```
int chopup_args(ARGS *arg, ...) {
   int num,i;
   ...
   if (arg->count == 0) {
      arg->count=20;
      arg->data= (char **)OPENSSL_malloc(...);
   }
   for (i=0; i<arg->count; i++)
      arg->data[i]=NULL;
   ....
}
```

Listing 2: Latent error in `chopup_args`.

Latent error

- only program paths with inputs `arg->count = 0` lead to error.
- some call to `chopup_args` will trigger the error.

Compositional Bug Reporting: Pulse-X

```
int main(int Argc, char *ARGV[]){
  ARGS arg;
  ...
  arg.count=0;
  ...
  if (!chopup_args(&arg,..)) break;
  ...
}
```

Listing 3: Manifest error in `main` of openssl.c.

Latent error

- only paths with inputs `arg->count = 0` lead to error.
- some call to `chopup_args` will trigger the error.
  - the call in `main`

Compositional Bug Reporting: True Positives Theorem

Theorem (Manifest errors)

An error triple $\models [p]\ \mathsf{C}\ [er : q]$ *with* $q \triangleq \exists \overrightarrow{X_q}.\ \kappa_q \wedge \pi_q$ *denotes a manifest error if:*

1. $p \equiv \texttt{emp} \wedge \texttt{true}$;
2. $\mathsf{sat}(q)$ *holds;*
3. $\mathsf{locs}(\kappa_q) \subseteq \overrightarrow{X_q}$, *where* $\mathsf{locs}(.)$ *is the set of heap locations; and*
4. *for all* $\vec{v}$, $\mathsf{sat}(\pi_q[\vec{v} / \vec{Y} \cup \mathsf{locs}(\kappa_q)])$ *holds, where* $\vec{Y} = \mathsf{flv}(q)$.

$$\mathsf{locs}(\texttt{emp}) \triangleq \emptyset \quad \mathsf{locs}(x \mapsto X) \triangleq \{x\} \quad \mathsf{locs}(X \mapsto V) = \mathsf{locs}(X \not\mapsto) \triangleq \{X\}$$
$$\mathsf{locs}(\kappa_1 * \kappa_2) \triangleq \mathsf{locs}(\kappa_1) \cup \mathsf{locs}(\kappa_2)$$

Implementation: Scientist vs. Engineer

"Scientists seek perfection and are idealists. ... An engineer's task is to not be idealistic. You need to be realistic as you have to compromise between conflicting interests."

Tony Hoare

Implementation: with an Incomplete Solver

speed vs. precision

dumb but fast vs. smart but slow

1. incomplete SAT solver: equalities
2. function pointers, unknown functions

Pulse-X might produce false positives

Evaluation

Data set: OpenSSL and 8 open-sourced C++ projects developed and maintained by Facebook.

practical bug classification: for each issue found

- true bug: it has been fixed
- pending bug: the fix has not accepted yet
- false positive: we could not find a fix

$$\text{fix rate} = \text{number of true bugs}/\text{total issues found}$$

Experimental plan:

- run Pulse-X and Infer on each project, collect timings and bugs found
- evaluate precision: check/classify the bugs found on OpenSSL
- evaluate scalability: compare the timings

Evaluation: Goals

- **Hypothesis H1.** On OpenSSL-1.0.1h Pulse-X has a superior fix rate to the present-day Infer.
- **Hypothesis H2.** Pulse-X finds new bugs worth fixing in current OpenSSL.
- **Hypothesis H3.** Pulse-X is broadly comparable with Infer in terms of performance, while reporting a comparable number of bugs.

Evaluation: H1

Old bugs with OpenSSL-1.0.1h

- 8,658 procedures, 444K lines of code, 2.83M of bytes of code
- older Infer found 15 bugs in 2015

Results:

- Pulse-X: 26 issues - 19 true bugs, 7 false positives
  - fix rate: 73%
- Infer: 80 issues - 39 true bugs (8 overlap with Pulse-X), 41 false positives
  - fix rate: 48.75%

Evaluation: H2

New bugs with OpenSSL-3.0.0

- 22,979 procedures, 754K lines of code, 8.55M of bytes of code

Results:

- Pulse-X: 30 issues - 15 true bugs, 5 pending, 10 false positives
  - fix rate: 50%
  - pull requests: #15834, #15836, and #15910
    - run Pulse-X on the fix, the bug does not occur.
- Infer: 116 issues - 7 true bugs (all overlap with Pulse-X), 40 false positives, 69 unchecked
  - fix rate: 6% - 65%

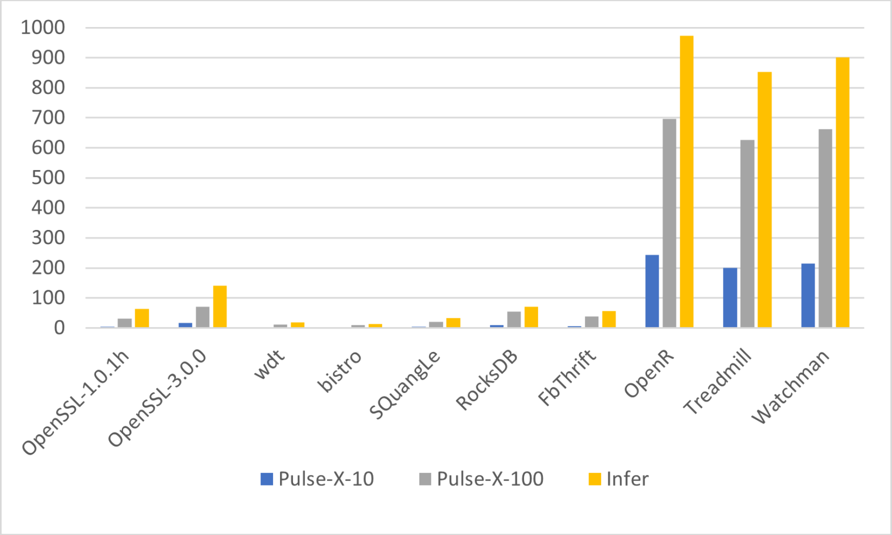
On average, fix rate: Pulse-X: 61% and Infer: 23% - 59%

Pulse at Facebook: fix rate is 82%.

Evaluation: H3

Project	#files	LoC(k)	#procs	BoC(m)
OpenSSL-1.0.1h	1536	444	8658	2.83
OpenSSL-3.0.3	2452	754	22979	8.55
wdt	194	25.4	6679	8.5
bistro	424	37.6	7290	9.7
SQuangLe	36	8.3	12938	17.9
RocksDB	1291	411.7	14669	18
FbThrift	5639	937.7	21753	29
OpenR	341	78.3	124461	195.7
Treadmill	409	25.3	236676	393.7
Watchman	557	63.2	245661	407.3

Evaluation: H3



1. Incorrectness logic
2. Separation logic
3. Incorrectness separation logic
4. Pulse-X
5. **Open probelms & Conclusion**

Open research probelms

1. Backward variant inference for loops and recursive procedures
   - cyclic incorrectness proofs
   - least fixed point for weakest post-conditions
2. Incorrectness proofs for OO programs
3. Quantitative weakest post
4. Reasoning about unknown functions
   - test harness generation (e.g., with directed fuzz testing)
   - incorrectness proofs for higher-order functions
5. Bug-finding tools for concurrent programs

Take away

Pulse-X: A scalable compositional bug-finding tool

- under-approximate bi-abduction
- true-positives theorem

Experiments, Pulse-X

- found 41 bugs in OpenSSL, 15 were previously unknown.
- fix rate might be 2.7*x* higher than Infer
- as scalable as Infer

Ad: PhD positions (with scholarships) are available!
Email: loc.le@ucl.ac.uk

Thanks for listening