Incorrectness Logic and Under-approximation: Foundations of Bug Catching

Quang Loc Le
University College London

TutorialFest POPL 2023 - Boston, Massachusetts, United States

16 January, 2023

with Peter O'Hearn, Azalea Raad, and Julien Vanegue

What is Pulse-X?

Pulse-X: A program analyser that proves the presence of bugs

(e.g., Null Pointer Dereferences, Use-After-Frees, leaks, ...)

Techniques:

- Under-approximate bi-abduction
  - under-approximate analogy of Infer
  - compositional: support continuous integration (CI) reasoning
- Compositional bug reporting mechanism
  - latent vs. manifest errors

Goals

- Precision
  - *doesn't spam the developers with false alarms.*
    - fix rate: comparable or better than Infer
    - Pulse-X found 15 new bugs in OpenSSL-3.0.0
- Scalability
  - performance: comparable to Infer
  - 3-dimensional scale: code (large codebases), people (big team), velocity (high frequency of code changes)
- Adoption: *Pulse, its brother: deployed at Meta, fix rate >80%!*

This tutorial

- Specification inference via bi-abduction
  - Compositional problem
- Compositional bug reporting (examples)
- Demo

Demo - seeing is believing

Pulse-Xfound 41 bugs in OpenSSL, **15 were unknown previously**

Three steps:

1. Compile OpenSSL code
2. Run Pulse-X on the compiled code
3. Retrieve information

Demo - seeing is believing (step 1)

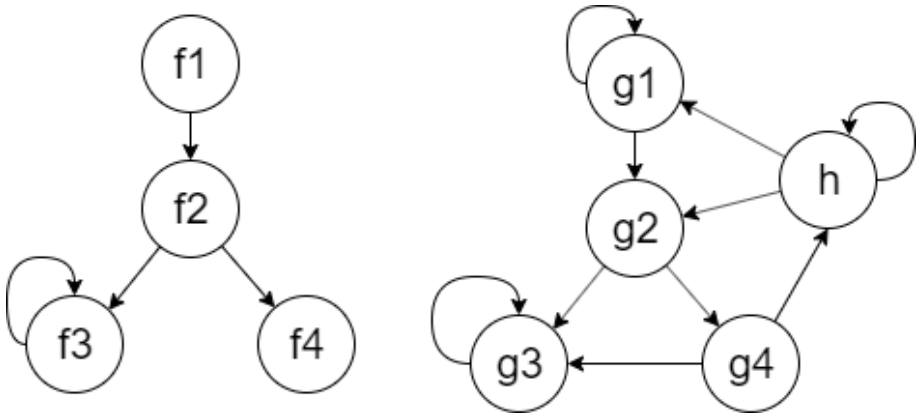
- Clone OpenSSL codebase
- Check out commit 147ed5f
  - 22,979 procedures, 754K lines of code, 8.55M of bytes of code
- (10 mins) Compile it and transform the code into intermediate language

```
$ infer capture -- make
```

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

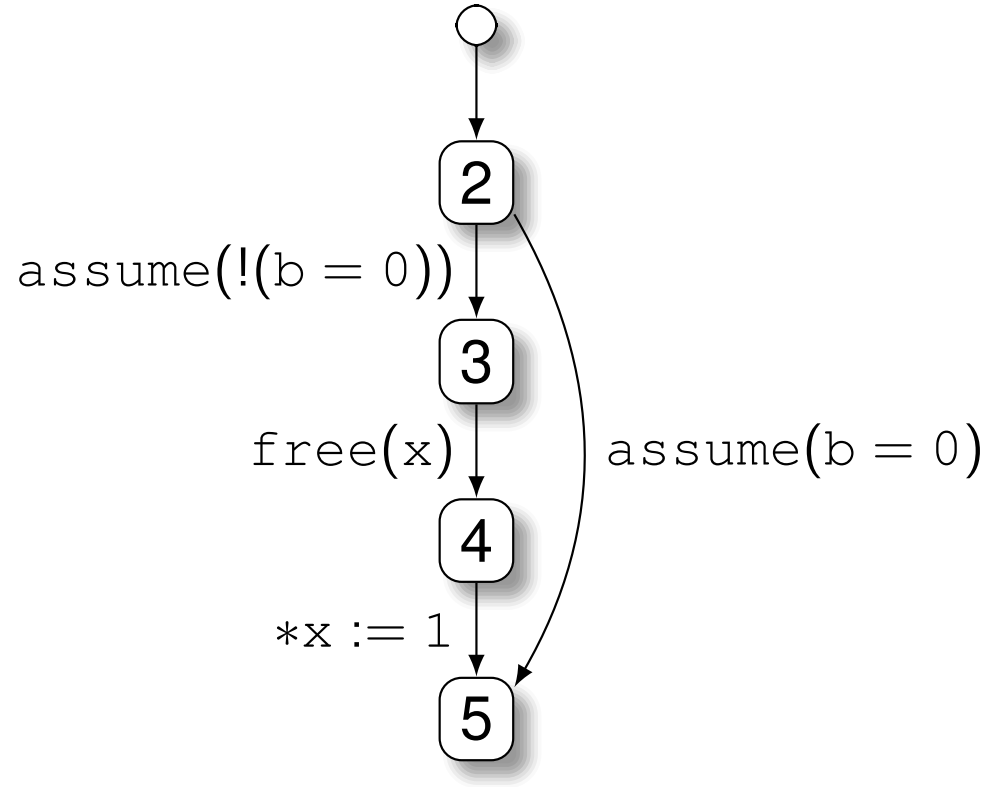


- analysis result: under-approximate specs i.e., incorrectness triples[1]
- a means: under-approximate bi-abduction

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

Analysis problem

```
1  void f(bool b, int *x){
2   if(b){
3    free(x);
4    *x := 1;
5   }
6  }
```



Given:

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

Question:

- find spec of the program

Incorrectness triple: Examples

Example 1:

Procedure spec: $[\exists T.\, y \mapsto T]\ \texttt{free(y)}\ [\mathit{ok}\colon\ y \not\mapsto]$

if y points to a heap cell at the beginning then the cell will be invalidated after executing the `free` procedure.

Example 2:

Procedure spec: $[y \not\mapsto]\ \texttt{free(y)}\ [\mathit{er}\colon\ y \not\mapsto]$

the spec encodes a double-free error (DF).

Incorrectness triple: Examples

Example 3:

Procedure spec: $[\exists T.x \mapsto T]\ *\mathtt{x} = 0\ [\mathit{ok}\colon x \mapsto 0]$

if x points to a heap cell with some content at the beginning then the cell's content will be reset to zero.

Example 4:

Procedure spec: $[x = \mathtt{null}]\ *\mathtt{x} = 0\ [\mathit{er}\colon x = \mathtt{null}]$

the spec encodes a null-pointer dereference error (NPE).

Example 5:

Procedure spec: $[x \not\mapsto]\ *\mathtt{x} = 0\ [\mathit{er}\colon x \not\mapsto]$

the spec encodes a Use-After-Free error (UAF).

Under-approximate bi-abduction

Over-approximate bi-abduction question:

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

Under-approximate bi-abduction question:

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

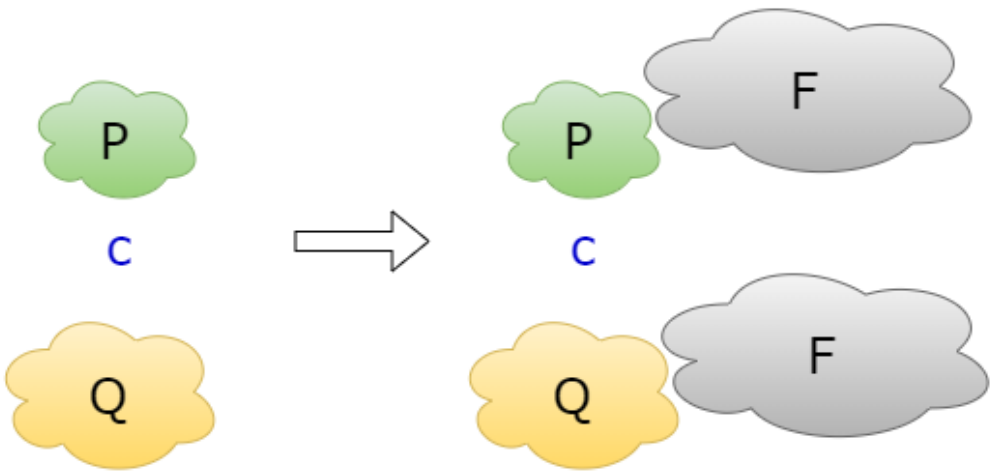
- abductive inference: find F
- anti-abductive inference: find M

Bi-abduction $--> $ Specification inference

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

- Frame rule

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



Bi-abduction $--> $ Specification inference

Pulse-X's approach:

- Interprocedure: bottom-up
- Intraprocedure: forward analysis with initial pre-condition, $pre \equiv \texttt{emp}$. It infers:
  - post-condition
  - missing assertions in pre-condition: either correctness (e.g., memory safety) or incorrectness (e.g., memory bugs)

Compositional problem

Given: Sequence $\mathtt{c}; \mathtt{c}'$ where

- $\mathtt{c}$ is the code analysed so far with inferred spec: $[P]\ \mathtt{c}\ [ok\colon Q]$
- c' is the next atomic command with given spec: $[P']\ \mathtt{c}'\ [\epsilon\colon Q']$

Question:

Find P_{comp} and Q_{comp} such that:

$$[P_{comp}]\ \mathtt{c}; \mathtt{c}'\ [\epsilon :\ Q_{comp}]$$

Compositional problem

Given: Sequence $c; c'$ where

- c is the code analysed so far with inferred spec: $[P]\ c\ [ok:\ Q]$
- c' is the next atomic command with given IL triple $[P']\ c'\ [\epsilon:\ Q']$

Infer spec for $c; c'$, $[P_{comp}]\ c; c'\ [\epsilon :\ Q_{comp}]$:

1. Solve bi-abductive problem:

$$P' \ * \ ?F \vdash Q \ * \ ?M$$

2. Apply frame rules:
   - c: $[P * M]\ c\ [ok:\ Q * M]$
   - c': $[P' * F]\ c'\ [\epsilon:\ Q' * F]$
3. Apply consequence rule to glue post of c and pre of c' (i.e., strengthen post of c):

$$[P * M]\ c\ [ok:\ P' \ * \ ?F]$$

4. Link them together with sequence rule:

$$[P * M]\ c; c'\ [\epsilon:\ Q' * F]$$

Compositional problem

Given: Sequence $c; c'$ where

- c is the code analysed so far with inferred spec: $[P]\, c\, [ok\colon Q]$
- c' is the next atomic command with given IL triple $[P']\, c'\, [\epsilon\colon Q']$

Infer spec for $c; c'$, $[P_{comp}]\, c; c'\, [Q_{comp}]$: Compositional proofs

$$\text{Seq}\ \frac{\text{Conseq}\ \dfrac{\text{Frame}\ \dfrac{[P]c[ok\colon Q]}{[P*M]c[ok\colon Q*M]}}{[P*M]c[ok\colon P'*F]} \quad P'*?F \vdash Q*?M \qquad \text{Frame}\ \dfrac{[P']c'[\epsilon\colon Q']}{[P'*F]c'[\epsilon\colon Q'*F]}}{[P*M]c; c'[\epsilon\colon Q'*F]}$$

Demo - seeing is believing (step 2)

(6 mins)Run Pulse-X

```
$ infer --pulse-only --pulse-isl --scheduler
    callgraph --pulse-model-malloc-pattern
    CRYPTO_malloc --pulse-model-free-pattern
    CRYPTO_free --pulse-model-realloc-pattern
    CRYPTO_realloc
```

Bi-abduction $-\!\!-\!\!>$ Specification inference

Given

```
1  int* foo () {
2    int* x = malloc(sizeof(*x));
3    *x = 0;
4
5    return x;
6  }
```

$s_1 : [\mathtt{emp}]\,\mathtt{x} = \mathtt{malloc}()\,[\mathit{ok}\colon x \mapsto \_]$

$s_2 : [\mathtt{emp}]\,\mathtt{x} = \mathtt{malloc}()\,[\mathit{ok}\colon \mathtt{emp} \wedge x = \mathtt{null}]$

$s_3 : [\exists T.\ x \mapsto T]\,{*}\mathtt{x} = 0\,[\mathit{ok}\colon x \mapsto 0]$

$s_4 : [\mathtt{emp} \wedge x = \mathtt{null}]\,{*}\mathtt{x} = 0\,[\mathit{er}\colon \mathtt{emp} \wedge x = \mathtt{null}]$

What is spec inferred?

Bi-abduction $--> $ Specification inference

Given

$s_1 : [\texttt{emp}]\ \texttt{x} = \texttt{malloc}()\ [ok\colon x \mapsto \_]$

$s_2 : [\texttt{emp}]\ \texttt{x} = \texttt{malloc}()\ [ok\colon \texttt{emp} \wedge x = \texttt{null}]$

$s_3 : [\exists T.\ x \mapsto T]\ *\texttt{x} = 0\ [ok\colon x \mapsto 0]$

$s_4 : [\texttt{emp} \wedge x = \texttt{null}]\ *\texttt{x} = 0\ [er\colon \texttt{emp} \wedge x = \texttt{null}]$

Pulse-X infers:

- $s_{13} : [\texttt{emp}]\ \texttt{x} = \texttt{malloc}(); *\texttt{x} = 0\ [ok\colon x \mapsto 0]$
- $s_{24} : [\texttt{emp}]\ \texttt{x} = \texttt{malloc}(); *\texttt{x} = 0\ [er\colon \texttt{emp} \wedge x = \texttt{null}]$

Pulse-X doesn't infer specs for s_{14} and s_{23} as it could not find solutions for the bi-abductive problems (due to contradiction):

- s_{14}

$$\texttt{emp} \wedge x = \texttt{null} \vdash x \mapsto \_$$

- s_{23}

$$\exists T.\ x \mapsto \vdash \texttt{emp} \wedge x = \texttt{null}$$

Compositional Bug Reporting

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

Using true positives theorem to classify *er* triples

Pulse-X classifies *er* triples:

- Manifest bugs: any call to the function will trigger the error.

$$s_{24} : [\texttt{emp}]\ \texttt{x} := \texttt{malloc}(); *\texttt{x} := 0\ [\textit{er}: \texttt{emp} \wedge x = \texttt{null}]$$

//manifest NPE

Compositional Bug Reporting

Pulse-X classifies *er* triples:

- Manifest bugs: any call to the function will trigger the error.

$$s_{24} : [\texttt{emp}]\ \texttt{x} := \texttt{malloc}(); *\texttt{x} := 0\ [\mathit{er}\colon \texttt{emp} \wedge x = \texttt{null}]$$

//manifest NPE

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

```
1  void f(int* x) {
2    *x = 42;
3  }
```

$$[\exists X.\ x \mapsto X]\ \texttt{f(x)}\ [\mathit{ok}\colon x \mapsto 42]$$

$$[x = \texttt{null}]\ \texttt{f(x)}\ [\mathit{er}\colon x = \texttt{null}]\ \ //\mathit{latent\ NPE}$$

$$[x \not\mapsto]\ \texttt{f(x)}\ [\mathit{er}\colon x \not\mapsto]\ \ //\mathit{latent\ DF}$$

Demo - infer spec for procedure `foo`

```
1  int* foo () {
2    int* x = malloc(sizeof(*x));
3    *x = 0;
4
5    return x;
6  }
```

1. Compile the source code and run Pulse-X

```
$ infer -g -F --pulse-only --pulse-isl --html
    -- gcc -c ex.c
```

2. Show error trace

```
$ infer explore
```

3. Show inferred spec in web page

Demo - seeing is believing (step 3)

Retrieve information

Results:

- 30 issues - 15 true bugs, 5 pending, 10 false positives

Interaction with OpenSSL Developers

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

- We committed fixes in pull request #15834

```
1  static int ssl_excert_prepend(SSL_EXCERT **pexc) {
2    SSL_EXCERT *exc = app_malloc(sizeof(*exc),
3                                 "prepend cert");
4
5    memset(exc, 0, sizeof(*exc));
6    ...
7  }
```

Interaction with OpenSSL Developers

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

- We committed fixes in pull request #15834

```
1  static int ssl_excert_prepend(SSL_EXCERT **pexc) {
2    SSL_EXCERT *exc = app_malloc(sizeof(*exc),
3                                 "prepend cert");
4
5  + if(exc == NULL)
6  +   return 0;
7    memset(exc, 0, sizeof(*exc));
8    ...
9  }
```

OpenSSL developer:

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

Interaction with OpenSSL Developers - *grep* search

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

```
1  void app_bail_out(char *fmt, ...) {
2      va_list args;
3      va_start(args, fmt);
4      BIO_vprintf(bio_err, fmt, args);
5      va_end(args);
6      ERR_print_errors(bio_err);
7      exit(EXIT_FAILURE);
8  }
9
10 void *app_malloc(size_t sz, const char *what) {
11     void *vp = OPENSSL_malloc(sz);
12
13     if (vp == NULL)
14         app_bail_out("%s: Could not allocate %zu bytes
               for %s\n",
15                      opt_getprog(), sz, what);
16     return vp;
17 }
```

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.

Take away

Pulse-X: A scalable compositional bug-finding tool

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

This tutorial shows how Pulse-X

- infers IL specs via bi-abudction
- found bugs in OpenSSL-3.0.0 beta