Incorrectness Logic for Scalable Bug Detection

Azalea Raad
Imperial College London

High Confidence Software and Systems
May 2023

azalea@imperial.ac.uk SoundAndComplete.org @azalearaad

Incorrectness Logic: Summary

+ ***Under-approximate*** analogue of Hoare Logic

+ Formal foundation for ***bug catching***

– Global reasoning: ***non-compositional*** (as in original Hoare Logic)

– Cannot target ***memory safety bugs*** (e.g. use-after-free)

Incorrectness Logic: Summary

+ ***Under-approximate*** analogue of Hoare Logic

+ Formal foundation for ***bug catching***

– Global reasoni

– Cannot target

Our Solution

Incorrectness Separation Logic

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{x = 1 \wedge y = 2 \wedge z = 3\}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{x \neq y \land x \neq z \land y \neq z\}$

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{x = 1 \land y = 2 \land z = 3\}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{ x_1 \neq x_2 \land x_1 \neq x_3 \land \ldots \}$

```
[x₁]:= 1;
[x₂]:= 2;
...
[xₙ]:= n;
```

post: $\{ x_1 = 1 \land \ldots \land x_n = n \}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: { $x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots$ }

n!/2 conjuncts !

```
[x1]:= 1;
[x2]:= 2;
…
[xn]:= n;
```

post: { $x_1 = 1 \wedge \ldots \wedge x_n = n$ }

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{ x \mapsto - * y \mapsto - * z \mapsto - \}$

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

- ideal for heap-manipulating programs with ***aliasing***

pre: $\{ x \mapsto - * y \mapsto - * z \mapsto - \}$

ownership
of heap cell at x

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

ideal for heap-manipulating programs with ***aliasing***

pre: $\{ x \mapsto - * y \mapsto - * z \mapsto - \}$

ownership of heap cell at x

'and ***separately***'

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}$

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***

☛ ideal for heap-manipulating programs with ***aliasing***

pre: $\{ x \mapsto - * y \mapsto - * z \mapsto - \}$

ownership of heap cell at x

'and ***separately***'

```
[x]:= 1;
[y]:= 2;
[z]:= 3;
```

post: $\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}$

$$\forall x,v,v'.\ x \mapsto v * x \mapsto v' \Rightarrow \text{false}$$

The Essence of Separation Logic (SL)

Frame Rule

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

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$

$p \ * \ \text{emp} \Leftrightarrow p$

The Essence of Separation Logic (SL)

Frame Rule

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

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$ $p \ * \ \text{emp} \Leftrightarrow p$

Local Axioms

WRITE $\{x \mapsto -\}\ [x] := v\ \{x \mapsto v\}$

READ $\{x \mapsto v\}\ y := [x]\ \{x \mapsto v \wedge y = v\}$

ALLOC $\{\text{emp}\}\ x := \text{alloc}()\ \{\exists l.\ l \mapsto - \wedge x = l\}$

Incorrectness Separation Logic (ISL)

IL

$$[p]\ C\ [\varepsilon: q]$$

SL

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

$$x \mapsto - \ * \ x \mapsto - \Leftrightarrow \text{false}$$

$$x \mapsto v \ * \ \text{emp} \Leftrightarrow x \mapsto v$$

ISL

$$\frac{[p]\ C\ [\varepsilon: q]}{[p * r]\ C\ [\varepsilon: q * r]}$$

$$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow \text{false}$$

$$x \mapsto v \ * \ \text{emp} \Leftrightarrow x \mapsto v$$

ISL: Local Axioms

$[x \mapsto v'] \; [x] := v \; [ok: x \mapsto v]$

$[x = null] \; [x] := v \; [er: x = null]$

WRITE

null-pointer-dereference error

ISL: Local Axioms

$[x \mapsto v'] \; [x] := v \; [ok: x \mapsto v]$

$[x = null] \; [x] := v \; [er: x = null]$

WRITE

null-pointer-dereference error

$[x \mapsto v] \; y := [x] \; [ok: x \mapsto v \wedge y = v]$

$[x = null] \; y := [x] \; [er: x = null]$

READ

ISL: Local Axioms

$[x \mapsto v'] \; [x] := v \; [ok: x \mapsto v]$

$[x = null] \; [x] := v \; [er: x = null]$

WRITE

null-pointer-dereference error

$[x \mapsto v] \; y := [x] \; [ok: x \mapsto v \wedge y = v]$

$[x = null] \; y := [x] \; [er: x = null]$

READ

$[emp] \; x := alloc() \; [ok: \exists l.\, l \mapsto v \wedge x = l]$

ALLOC

ISL: Local Axioms

$[x \mapsto v']$ $[x]:= v$ $[ok: x \mapsto v]$

$[x=null]$ $[x]:= v$ $[er: x=null]$

<u>Hidden Technical Details</u>

- Standard SL model **broken** for ISL: <u>unsound frame rule</u>
- Fix: A monotonic heap model
- Advantage: recover completeness for ISL (unlike SL)

$[emp]$ $x:= alloc()$ $[ok: \exists l.\ l \mapsto v \land x=l\ ]$

ALLOC

ISL Summary

- ➡ IL + SL for ***compositional bug catching***
- ➡ ***Under-approximate*** analogue of SL
- ➡ Targets ***memory safety bugs*** (e.g. use-after-free)
- ➡ ***No-false-positives theorem:***

    All bugs identified are true bugs

Pulse-X: ISL for Scalable Bug Detection

Pulse-X at a Glance

- **Automated** program analysis for **memory safety errors** (NPEs, UAFs) and **leaks**
- Underpinned by ISL (under-approximate) — **no false positives***
- **Inter-procedural** and **bi-abductive** — under-approximate analogue of Infer
- **Compositional** (begin-anywhere analysis) — important for CI
- Deployed at Meta
- **Performance**: comparable to Infer
- **Fix rate**: comparable or better than Infer!
- Three dimensional scalability
  - ➡ code size (large codebases)
  - ➡ people (large teams, CI)
  - ➡ speed (high frequency of code changes)

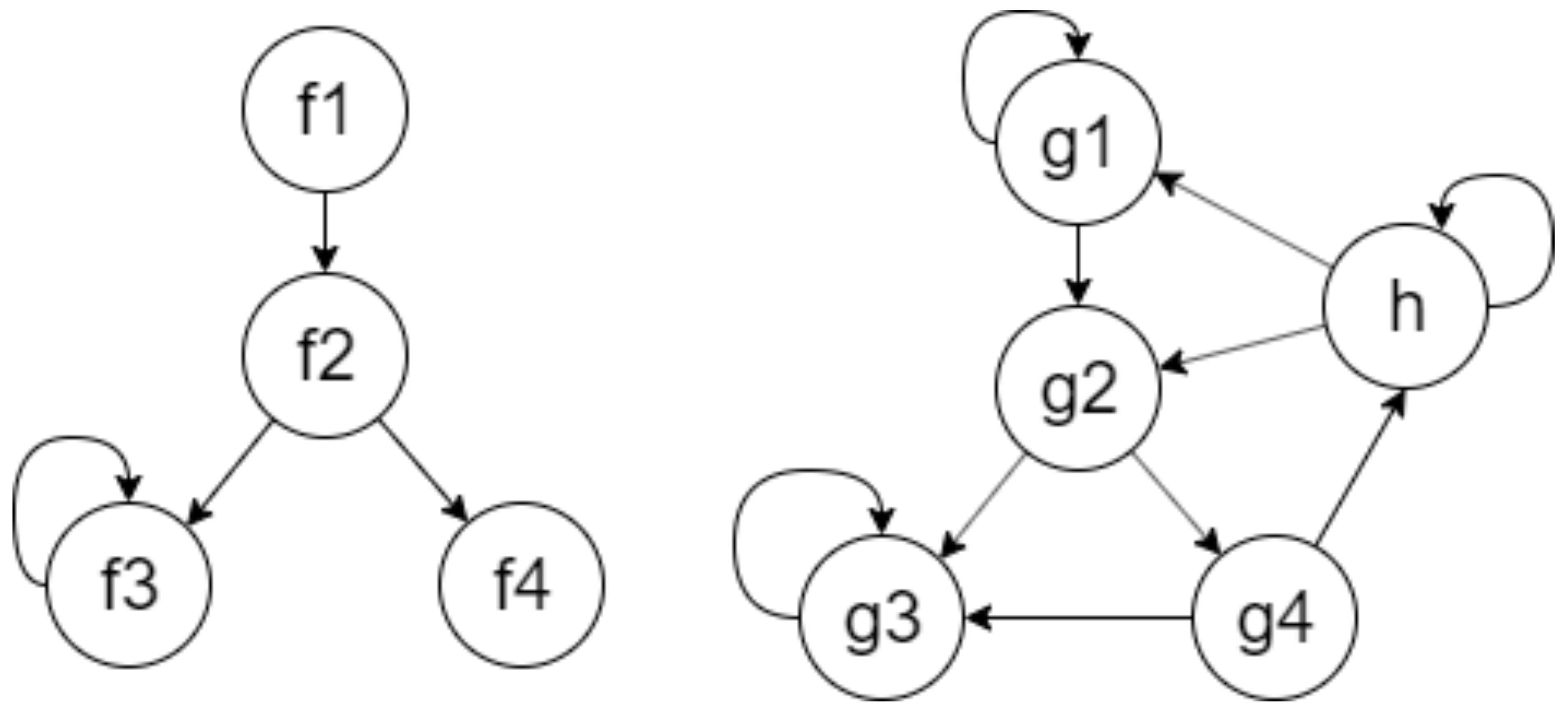
Compositional, Begin-Anywhere Analysis

- Analysis result of a program = analysis results of its parts
  +
  a method of combining them

Compositional, Begin-Anywhere Analysis

- Analysis result of a program = analysis results of its parts
  +
  a method of combining them

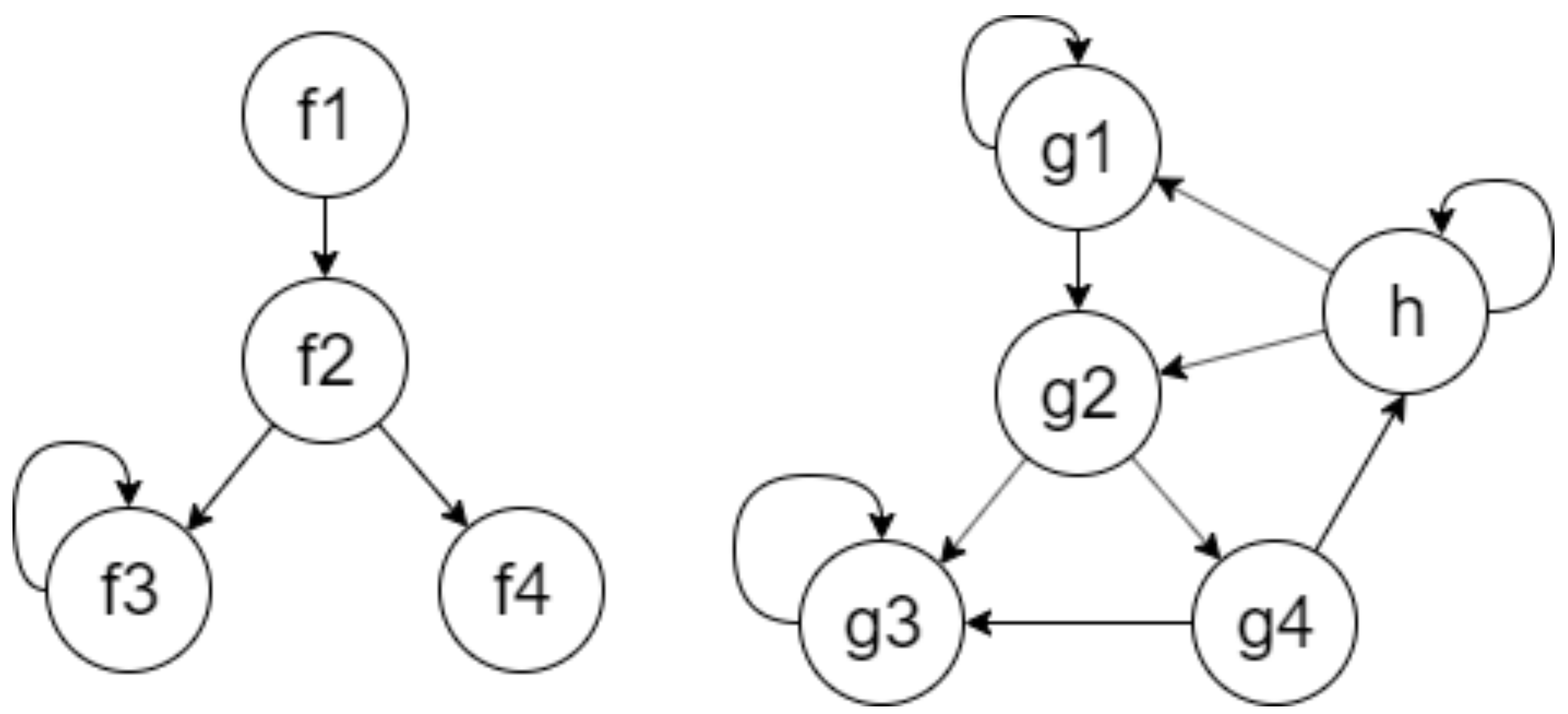
  - ➡ Parts: Procedures



Compositional, Begin-Anywhere Analysis

- Analysis result of a program = analysis results of its parts
  +
  a method of combining them

  - ➡ Parts: Procedures

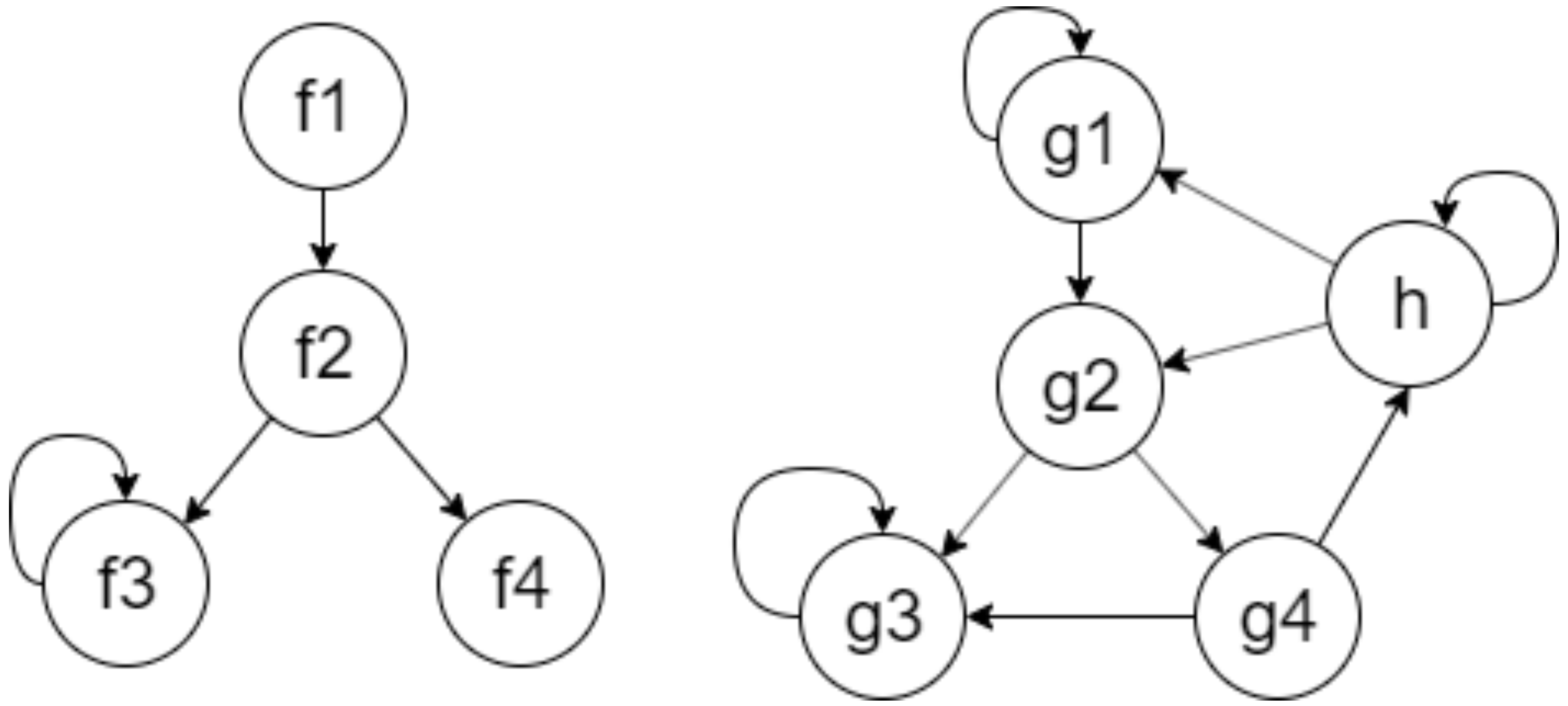


  - ➡ Method: under-approximate bi-abduction

Compositional, Begin-Anywhere Analysis

❖ Analysis result of a program = analysis results of its parts
+
a method of combining them

➡ Parts: Procedures



➡ Method: under-approximate bi-abduction

➡ Analysis result: incorrectness triples (under-approximate specs)

Pulse-X Algorithm: Proof Search in ISL

- Analyse each procedure f in isolation, find its **summary** (collection of ISL triples)
  - ➡ A **summary table** T, initially populated only with local (pre-defined) axioms
  - ➡ Use bi-abduction and T to find the summary of f
  - ➡ Recursion: bounded unrolling
  - ➡ Extend T with the summary of f

Pulse-X Algorithm: Proof Search in ISL

- Analyse each procedure f in isolation, find its **summary** (collection of ISL triples)
  - A **summary table** T, initially populated only with local (pre-defined) axioms
  - Use bi-abduction and T to find the summary of f
  - Recursion: bounded unrolling
  - Extend T with the summary of f
- Similar bi-abductive mechanism to Infer, but:
  - Can **soundly** drop execution paths/branches
  - Can **soundly** bound loop unrolling

Pulse-X: Null Pointer Dereference in OpenSSL

```
1.int ssl_excert_prepend(...){
2.    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
3.    memset(exc, 0, sizeof(*exc));
      …
}
```

calls CRYPTO_malloc (a malloc wrapper)

Pulse-X: Null Pointer Dereference in OpenSSL

```
1.int ssl_excert_prepend(...){
2.    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
3.    memset(exc, 0, sizeof(*exc));
      …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

Pulse-X: Null Pointer Dereference in OpenSSL

```
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
    memset(exc, 0, sizeof(*exc));
    …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `*exc= app_malloc(sz, …)` [ok: exc = null]

+

[exc = null] `memset(exc,-,-)` [er: exc = null]

Pulse-X: Null Pointer Dereference in OpenSSL

```
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
    memset(exc, 0, sizeof(*exc));
    …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `*exc= app_malloc(sz, …)` [ok: exc = null]

+

[exc = null] `memset(exc,-,-)` [er: exc = null]

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Pulse-X: Null Pointer Dereference in OpenSSL

apps/lib/s_cb.c Outdated Hide resolved

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

+      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`.

Pulse-X: Null Pointer Dereference in OpenSSL

apps/lib/s_cb.c Outdated

Hide resolved

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

+     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`.

Pulse-X: Null Pointer Dereference in OpenSSL

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`.

Created pull request #15836 to commit the fix.

Pulse-X: Bug Reporting

No False Positives: Report ***All*** Bugs Found?

Not quite…

Pulse-X: Bug Reporting

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

Pulse-X: Bug Reporting

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

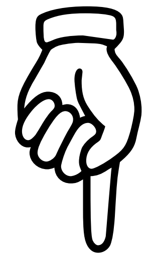
WRITE [x=null] *x = v [er: x=null]

[x=null] foo(x) [er: x=null]

Pulse-X: Bug Reporting

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

WRITE [x=null] *x = v [er: x=null]



[x=null] foo(x) [er: x=null]

Should we report this NPD?

Pulse-X: Bug Reporting

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

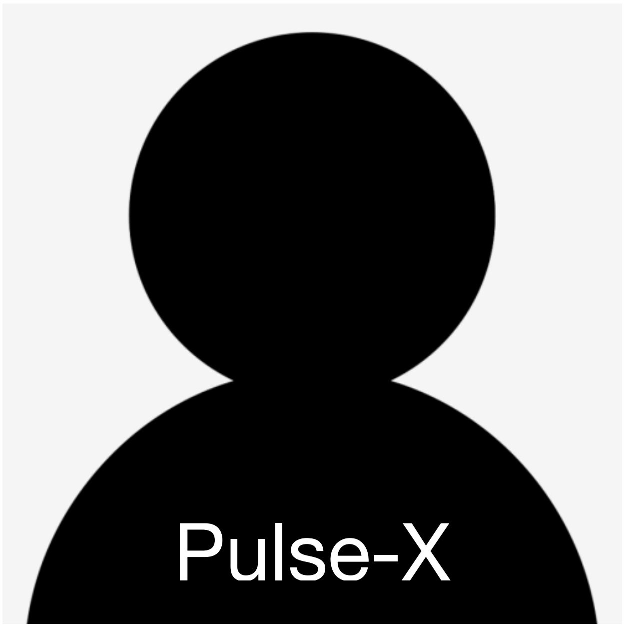
WRITE [x=null] *x = v [er: x=null]

[x=null] foo(x) [er: x=null]

Should we report this NPD?

yes → / **no** →

"But I never call foo with null!"



"Which bugs shall I report then?"

Pulse-X: Bug Reporting

Problem

Must consider the **whole program** to decide whether to report

Solution

Manifest Errors

Pulse-X: ***Manifest*** Errors

- Intuitively: the error occurs for **all input states**

Pulse-X: ***Manifest*** Errors

- Intuitively: the error occurs for **all input states**
- Formally: [p] C [er: q] is manifest iff:

$$\forall s.\ \exists s'.\ (s,s') \in [C]_{er} \land s' \in (q * true)$$

Pulse-X: ***Manifest*** Errors

- Intuitively: the error occurs for **all input states**
- Formally: [p] C [er: q] is manifest iff:

$$\forall s.\ \exists s'.\ \ (s,s') \in [C]_{er}\ \wedge\ s' \in (q * \mathrm{true})$$

- Algorithmically: …

Pulse-X: Null Pointer Dereference in OpenSSL

```
1.int ssl_excert_prepend(...){
2.    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert”);
3.    memset(exc, 0, sizeof(*exc));
      …
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Pulse-X: Null Pointer Dereference in OpenSSL

```
int ssl_excert_prepend(...){
    SSL_EXCERT *exc= app_malloc(sizeof(*exc), "prepend cert");
    memset(exc, 0, sizeof(*exc));
    ...
}
```

calls CRYPTO_malloc (a malloc wrapper)

null pointer dereference

CRYPTO_malloc may return null!

[emp] `ssl_excert_prepend(…)` [er: exc = null]

Manifest Error (all calls to `ssl_excert_prepend` can trigger the error)!

Pulse-X: ***Latent*** Errors

An error triple [p] C [er: q] is latent iff it is not manifest

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
    …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
     …
    if (args->count == 0 ) {
      args->count=20;
      args->data= (char**)ssl_excert_prepend(…);
    }
    for (i=0; i<args->count; i++) {
      args->data[i]=NULL;
      …
    }
```

null pointer dereference

Pulse-X: Latent Error

```
int chopup_args(ARGS *args,…){
  …
  if (args->count == 0 ) {
    args->count=20;
    args->data= (char**)ssl_excert_prepend(…);
  }
  for (i=0; i<args->count; i++) {
    args->data[i]=NULL;
    …
  }
```

null pointer dereference

Latent Error:

only calls with `args->count==0` can trigger the error

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

leaks `ssl_bio`

Pulse-X: Memory Leak in OpenSSL

```
static int www_body(…){
  ...
  io = BIO_new(BIO_f_buffer());
  ssl_bio BIO_new(BIO_f_ssl());
  ...
  BIO_push(io, ssl_bio);
  ...
  BIO_free_all(io);
  ...
  return ret;
}
```

does nothing when `io` is null

leaks `ssl_bio`

426 lines of complex code:
`io` manipulated by several procedures and multiple loops

Pulse-X performs under-approximation with bounded loop unrolling

Pulse-X Summary

- ➡ Automated program analysis for detecting memory safety errors and leaks
- ➡ Manifest errors (underpinned by ISL): no false positives*
- ➡ compositional, scalable, begin-anywhere

ISL Extension:
Concurrent Incorrectness Separation Logic (CISL)
&
Incorrectness Non-Termination Logic (INTL)

ISL Extension:

Concurrent Incorrectness Separation Logic (CISL)

&

Incorrectness Non-Termination Logic (INTL)

Termination vs Non-Termination

- Showing **termination** is compatible with **correctness** frameworks:
  - ➡ **Every** trace of a given program must terminate
  - ➡ Inherently **over-approximate**

```
skip + x:=1
```

Termination vs Non-Termination

- Showing **termination** is compatible with **correctness** frameworks:
  - ➡ **Every** trace of a given program must terminate
  - ➡ Inherently **over-approximate**

```
skip + x:=1
```

- Showing **non-termination** compatible with **incorrectness** frameworks:
  - ➡ **Some** trace of a given program does not terminate
  - ➡ Inherently **under-approximate**

```
skip + while(true)skip
```

Incorrectness Non-Termination Logic (INTL)

- A framework for **detecting non-termination bugs**
- Supports **unstructured** constructs (goto), as well exceptions and breaks
- Reasons for non-termination:
  - ➡ Infinite loops
  - ➡ Infinite recursion
  - ➡ Cyclic `goto` soups

INTL Proof Rules and Principles

INTL Proof Rules

=

ISL Proof Rules

+

Divergence (Non-Termination) Rules

INTL Divergence Proof Rules

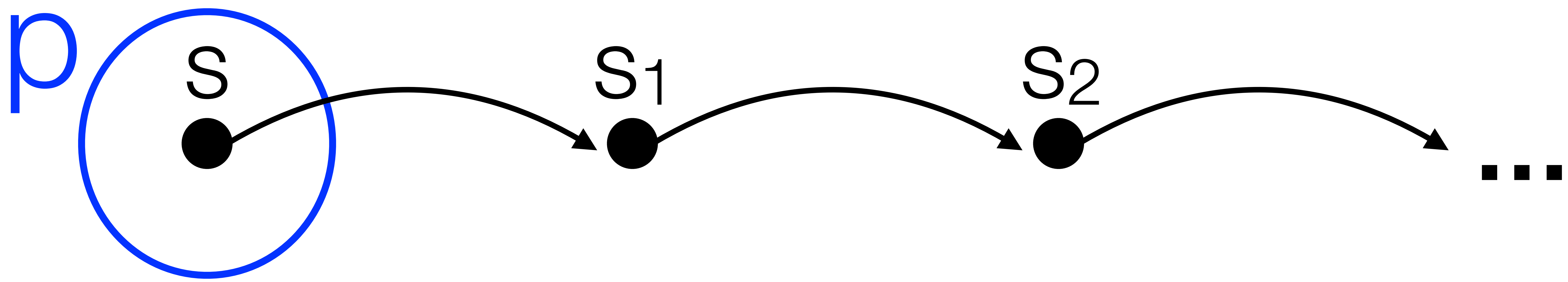
$[p]\ C\ [\infty]$

Starting from **some** state s in p, C has a divergent trace

INTL Divergence Proof Rules

$$[p]\ C\ [\infty]$$

Starting from **some** state s in p, C has a divergent trace



INTL Divergence Proof Rules (Sequencing)

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1;\ C_2\ [\infty]}$$

INTL Divergence Proof Rules (Sequencing)

$$\frac{[p]\ C_1\ [\infty]}{[p]\ C_1; C_2\ [\infty]}$$

$$\frac{[p]\ C_1\ [ok: q] \qquad [q]\ C_2\ [\infty]}{[p]\ C_1; C_2\ [\infty]}$$

INTL Divergence Proof Rules (Branches)

$$\frac{[p]\ C_i\ [\infty] \qquad \textbf{some}\ i \in \{1, 2\}}{[p]\ C_1 + C_2\ [\infty]}$$

- **Drop paths/branches** (this is a **sound** under-approximation)
- **Scalable** bug detection!

INTL Divergence Proof Rules (Loops)

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}$$

$$\frac{[p]\ C\ [ok{:}\ p] \qquad \text{[extra condition omitted]}}{[p]\ C^*\ [\infty]}$$

Conclusions

- Incorrectness Separation Logic (ISL)
  - Combining IL and SL for ***compositional bug catching*** (in sequential programs)
  - ***no-false-positives*** theorem
- Pulse-X
  - Automated program analysis for detecting memory safety errors and leaks
  - Manifest errors (underpinned by ISL): no false positives*
  - compositional, scalable, begin-anywhere
- INTL
  - ISL for detecting **non-termination bugs**
  - ***no-false-positives*** theorem
  - Infinite loop/recursion detection

Thank You for Listening!

azalea@imperial.ac.uk

SoundAndComplete.org

@azalearaad