Incorrectness Logic for Scalable Bug Detection

Azalea Raad
Imperial College London

Dependable and Secure Software Systems, ETH Zürich
October 2022

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

State of the Art: ***Correctness***

- Lots of work on ***reasoning*** for proving ***correctness***
  - ➡ Prove the ***absence of bugs***
  - ➡ ***Over-approximate*** reasoning

State of the Art: ***Correctness***

- Lots of work on ***reasoning*** for proving ***correctness***
  - ➡ Prove the ***absence of bugs***
  - ➡ ***Over-approximate*** reasoning
  - ➡ ***Compositionality***

    in ***code*** ⇒ reasoning about ***incomplete components***

    in ***resources*** accessed ⇒ spatial locality

State of the Art: ***Correctness***

- Lots of work on ***reasoning*** for proving ***correctness***
  - ➡ Prove the ***absence of bugs***
  - ➡ ***Over-approximate*** reasoning
  - ➡ ***Compositionality***

    in ***code*** ⇒ reasoning about ***incomplete components***

    in ***resources*** accessed ⇒ spatial locality
  - ➡ ***Scalability*** to large teams and codebases

Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$

Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Hoare Logic (HL)

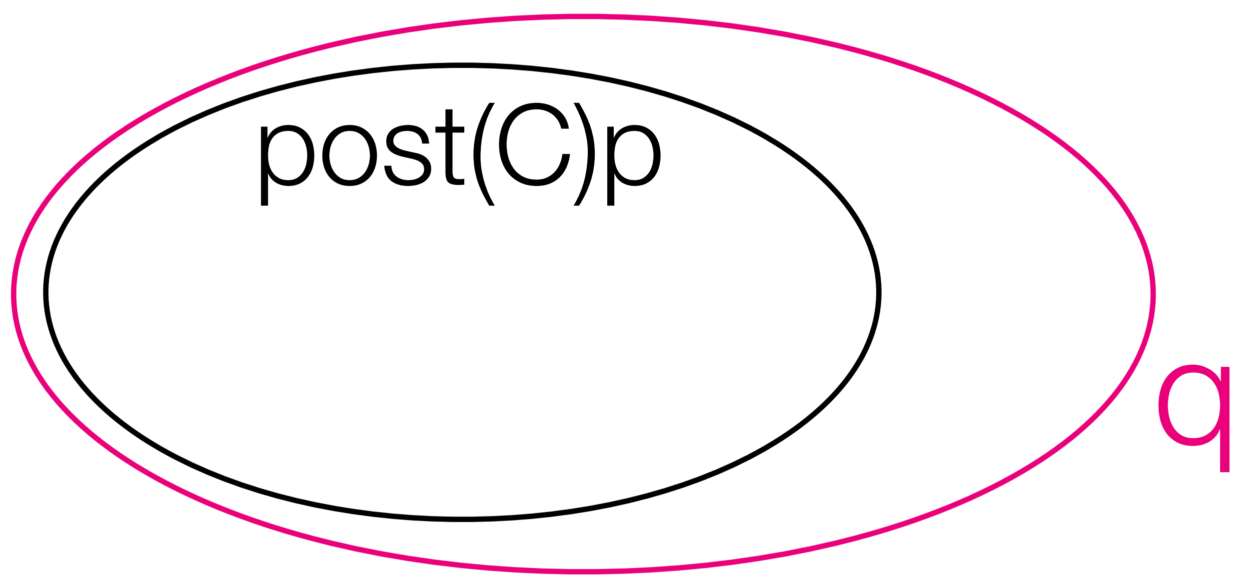
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p

Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

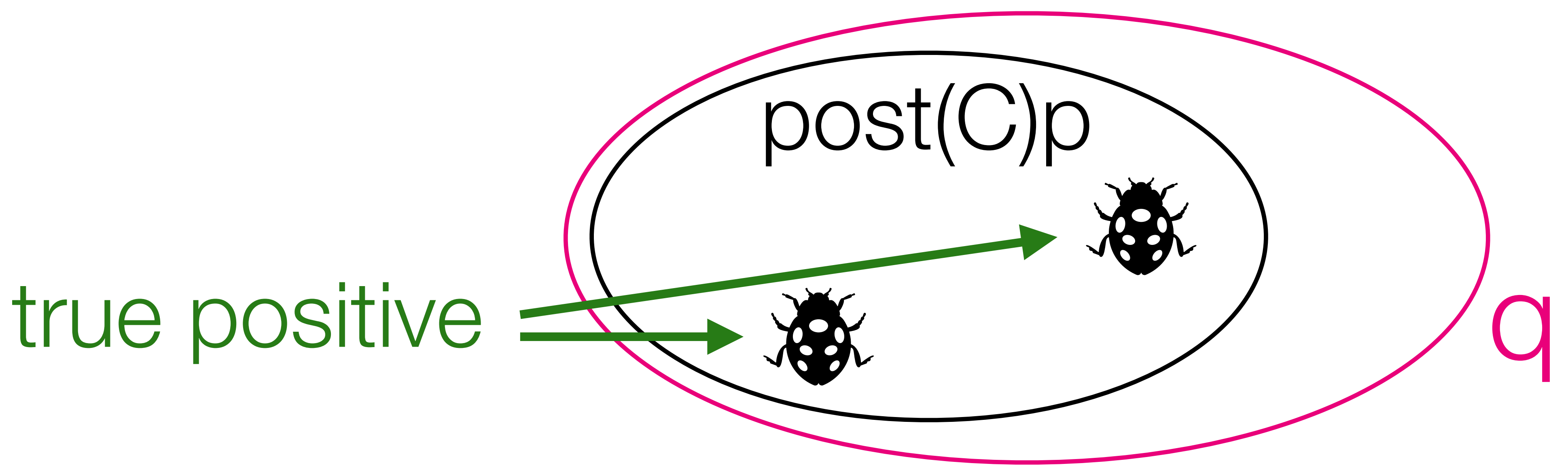
q *over-approximates* post(C)p



Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

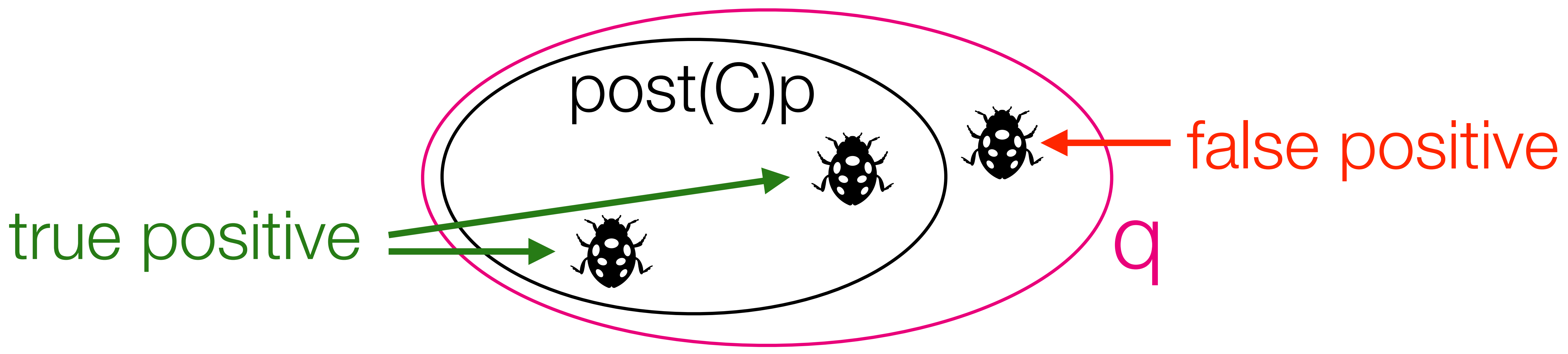
q *over-approximates* post(C)p



Hoare Logic (HL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



“Don’t spam the developers!”

Incorrectness Logic:

A Formal Foundation

for

Bug Catching

Part I.

Incorrectness Logic (IL)

&

Incorrectness Separation Logic (ISL)

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

$post(C)p \subseteq q$

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

$post(C)p \supseteq q$

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s *in* p
if running C *on* s *terminates in* s', *then* s' *is in* q

Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

For all states s in p
if running C *on* s *terminates in* s', *then* s' *is in* q

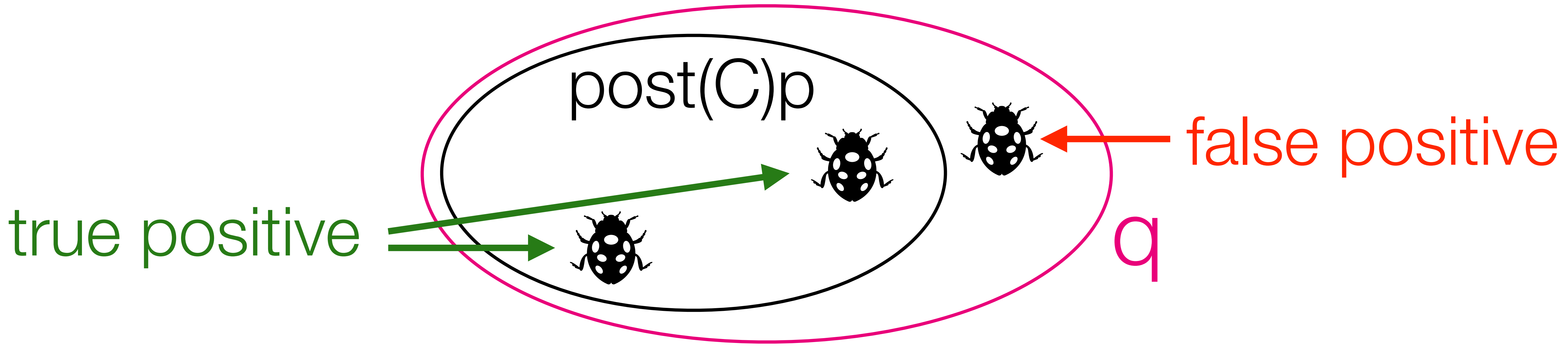
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

For all states s in q
s *can be reached by running* C *on some* s' *in* p

Incorrectness Logic (IL)

Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



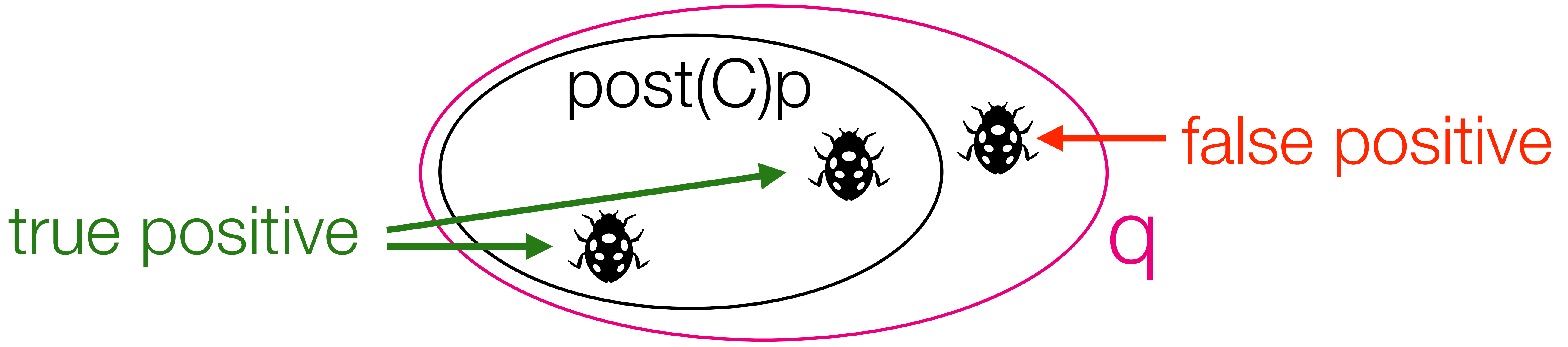
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

q *under-approximates* post(C)p

Incorrectness Logic (IL)

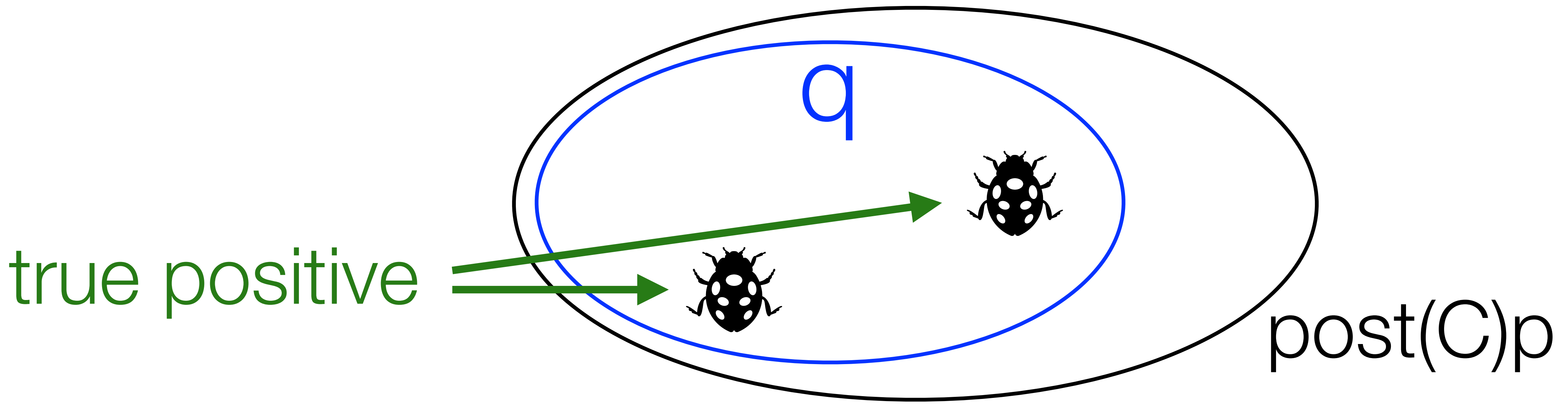
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

q *over-approximates* post(C)p



Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

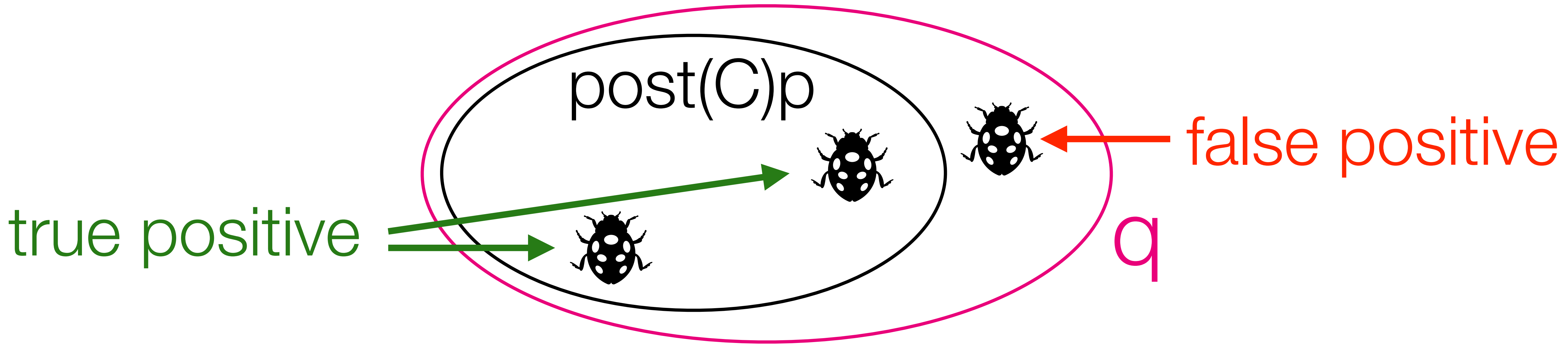
q *under-approximates* post(C)p



Incorrectness Logic (IL)

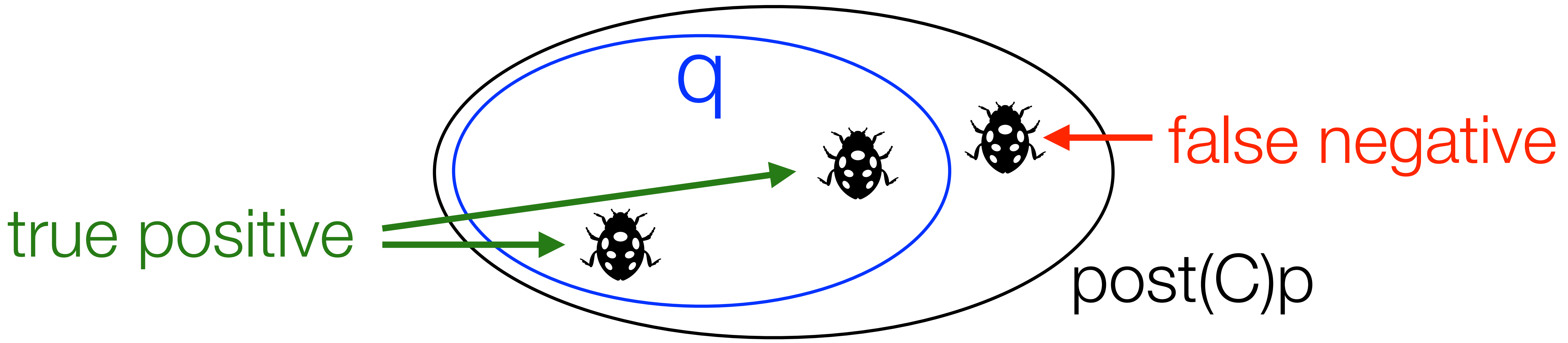
Hoare triples	$\{p\}\ C\ \{q\}$	*iff*	$post(C)p \subseteq q$

q *over-approximates* post(C)p



Incorrectness triples	$[p]\ C\ [q]$	*iff*	$post(C)p \supseteq q$

q *under-approximates* post(C)p



Incorrectness Logic (IL)

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

ε: exit condition

ok: normal execution

er : erroneous execution

Incorrectness Logic (IL)

[p] C [ε: q]

ε: exit condition

ok: normal execution

er : erroneous execution

[y=v] x:=y [ok: x=y=v]

Incorrectness Logic (IL)

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

ε: exit condition

ok: normal execution

er : erroneous execution

$[y{=}v]\ x{:=}y\ [ok\!: x{=}y{=}v]$

$[p]\ \mathrm{error}(\,)\ [er\!: p]$

Incorrectness Logic (IL)

$[p]\ C\ [\varepsilon : q]$ *iff* $\mathrm{post}(C, \varepsilon)p \supseteq q$

Equivalent Definition (reachability)

$[p]\ C\ [\varepsilon : q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

Incorrectness Logic: Summary

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

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

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 reasonin

– Cannot target *n*

<u>***Our Solution***</u>

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

```
[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: $\{\boxed{x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \ldots}\}$

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

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

n!/2 conjuncts !

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

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

ownership
of heap cell at x

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

'and ***separately***'

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: $\{ x \mapsto - \ast y \mapsto - \ast z \mapsto - \}$

ownership
of heap cell at x

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

'and ***separately***'

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

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

The Essence of Separation Logic (SL)

Frame Rule

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

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

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

The Essence of Separation Logic (SL)

Frame Rule

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

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

Local Axioms

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

The Essence of Separation Logic (SL)

Frame Rule

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

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

Local Axioms

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

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

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	$\{x \mapsto v\}\ y := [x]\ \{x \mapsto v \land y = v\}$
ALLOC	$\{\text{emp}\}\ x := \text{alloc}()\ \{\exists l.\ l \mapsto - \land x = l\}$

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

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

FREE $\{x \mapsto -\}\ \text{free}(x)\ \{\ \text{emp}\ \}$

Incorrectness Separation Logic (ISL)

IL

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

SL

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

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

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

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 (First Attempt)

WRITE

[x ↦ v'] [x]:= v [ok: x ↦ v]

ISL: Local Axioms (First Attempt)

WRITE

[x ↦ v'] [x]:= v [ok: x ↦ v]

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

ISL: Local Axioms (First Attempt)

WRITE

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

$\boxed{[x = \text{null}] \; [x] := v \; [er: x = \text{null}]}$

null-pointer dereference error

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v'] \ [x] := v \ [ok: x \mapsto v]$ $\qquad$ $\boxed{[x{=}null] \ [x] := v \ [er: x{=}null]}$

null-pointer dereference error

READ

$[x \mapsto v] \ y := [x] \ [ok: x \mapsto v \land y{=}v]$ $\qquad$ $[x{=}null] \ y := [x] \ [er: x{=}null]$

ISL: Local Axioms (First Attempt)

WRITE

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

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

null-pointer dereference error

READ

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

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

ALLOC

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

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v'] \; [x] := v \; [ok: x \mapsto v]$ $\qquad$ $\boxed{[x{=}null] \; [x] := v \; [er: x{=}null]}$

null-pointer dereference error

READ

$[x \mapsto v] \; y := [x] \; [ok: x \mapsto v \wedge y{=}v]$ $\qquad$ $[x{=}null] \; y := [x] \; [er: x{=}null]$

ALLOC

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

FREE

$[x \mapsto v] \; free(x) \; [ok: emp]$ $\qquad$ $[x{=}null] \; free(x) \; [er: x{=}null]$

ISL: Local Axioms (First Attempt)

WRITE

$[x \mapsto v']\ [x] := v\ [ok: x \mapsto v]$ $\qquad$ $\boxed{[x = null]\ [x] := v\ [er: x = null]}$

null-pointer dereference error

READ

$[x \mapsto v]\ y := [x]\ [ok: x \mapsto v \land y = v]$ $\qquad$ $[x = null]\ y := [x]\ [er: x = null]$

ALLOC

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

FREE

$[x \mapsto v]\ free(x)\ [ok: emp]$ $\qquad$ $\ldots ll]\ free(x)\ [er: x = null]$

ISL: Local Axioms (First Attempt)

ISL

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

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

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

$[x \mapsto v]$ free(x) $[ok\!:\ \text{emp}]$

$[p]\ C\ [\varepsilon\!:\ q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

ISL: Local Axioms (First Attempt)

ISL

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

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

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

$$\frac{[x \mapsto v]\ \text{free}(x)\ [ok{:}\ \text{emp}]}{[x \mapsto v * x \mapsto v]\ \text{free}(x)\ [ok{:}\ \text{emp} * x \mapsto v]}\ (\text{Frame})$$

$[p]\ C\ [\varepsilon{:}\ q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

ISL: Local Axioms (First Attempt)

ISL

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

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

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

$$\dfrac{\dfrac{[x \mapsto v]\ \text{free}(x)\ [ok: \text{emp}]}{[x \mapsto v * x \mapsto v]\ \text{free}(x)\ [ok: \text{emp} * x \mapsto v]}\ (\text{Frame})}{[\text{false}]\ \text{free}(x)\ [ok: x \mapsto v]}\ (\text{Cons})$$

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s',s) \in [C]\varepsilon$

ISL: Local Axioms (First Attempt)

ISL

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

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

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

$$\dfrac{\dfrac{[x \mapsto v]\ \text{free}(x)\ [\text{ok: emp}]}{[x \mapsto v * x \mapsto v]\ \text{free}(x)\ [\text{ok: emp} * x \mapsto v]}\ \text{(Frame)}}{[\text{false}]\ \text{free}(x)\ [\text{ok: } x \mapsto v]}\ \text{(Cons)}$$

$[p]\ C\ [\varepsilon\!: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

$[\text{false}]\ C\ [\varepsilon\!: q]$ ✗ (unless $q \Rightarrow \text{false}$)

ISL: Local Axioms (First Attempt)

ISL

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

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

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

(Frame)

[x ↦ v ... ↦ v]

(Cons)

Solution:

Track Deallocated Locations!

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall s \in q.\ \exists s' \in p.\ (s', s) \in [C]\varepsilon$

$[\text{false}]\ C\ [\varepsilon: q]$ ✗ (unless $q \Rightarrow \text{false}$)

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: \ emp]$

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: \; x \not\mapsto \;]$

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: \boxed{x \not\mapsto}]$

x is ***deallocated***

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: \boxed{x \not\mapsto}]$

x is ***deallocated***

$x \mapsto v \ * \ x \mapsto v' \Leftrightarrow$ false

$p \ * \ emp \Leftrightarrow p$

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: \boxed{x \not\mapsto}]$

x is ***deallocated***

$$x \mapsto v \ast x \mapsto v' \Leftrightarrow \text{false}$$
$$p \ast \text{emp} \Leftrightarrow p$$
$$x \mapsto v \ast x \not\mapsto \Leftrightarrow \text{false}$$
$$x \not\mapsto \ast x \not\mapsto \Leftrightarrow \text{false}$$

Solution: Track Deallocated Locations!

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

Solution: Track Deallocated Locations!

$$\frac{[x \mapsto v]\ \text{free}(x)\ [ok\colon x \not\mapsto\ ]}{[x \mapsto v * x \mapsto v]\ \text{free}(x)\ [ok\colon x \not\mapsto * x \mapsto v]}$$

Solution: Track Deallocated Locations!

$$\dfrac{\dfrac{[x \mapsto v]\ \mathrm{free}(x)\ [ok: x \not\mapsto\ ]}{[x \mapsto v * x \mapsto v]\ \mathrm{free}(x)\ [ok: x \not\mapsto * x \mapsto v]}}{[\mathrm{false}]\ \mathrm{free}(x)\ [ok: \mathrm{false}]} \quad ✔$$

$[p]\ C\ [\varepsilon: q]$ *iff* $\forall\, s \in q.\ \exists\, s' \in p.\ (s', s) \in [C]\varepsilon$

$[p]\ C\ [\varepsilon: \mathrm{false}]$ ✔ (vacuous)

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x=null]$ free(x) $[er: x=null]$

FREE

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x{=}null]$ free(x) $[er: x{=}null]$

FREE

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x = null]$ free(x) $[er: x = null]$

FREE

$\boxed{[x \not\mapsto]\ \text{free(x)}\ [er: x \not\mapsto]}$

use-after-free error

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x{=}null]$ free(x) $[er: x{=}null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

FREE

use-after-free error

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

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

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

WRITE

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x{=}null]$ free(x) $[er: x{=}null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

FREE

use-after-free error

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

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

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

WRITE

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

$[x{=}null]$ y:= [x] $[er: x{=}null]$

$[x \not\mapsto]$ y:= [x] $[er: x \not\mapsto]$

READ

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

$[x = null]$ free(x) $[er: x = null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

FREE

use-after-free error

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

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

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

WRITE

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

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

$[x \not\mapsto]$ y:= [x] $[er: x \not\mapsto]$

READ

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

ALLOC

ISL: Local Axioms

$[x \mapsto v]$ free(x) $[ok: x \not\mapsto]$

FREE

$[x{=}null]$ free(x) $[er: x{=}null]$

$[x \not\mapsto]$ free(x) $[er: x \not\mapsto]$

use-after-free error

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

WRITE

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

$[x \not\mapsto]$ [x]:= v $[er: x \not\mapsto]$

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

READ

$[x{=}null]$ y:= [x] $[er: x{=}null]$

$[x \not\mapsto]$ y:= [x] $[er: x \not\mapsto]$

ALLOC

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

$[y \not\mapsto]$ x:= alloc() $[ok: y \mapsto v \wedge x{=}y]$

ISL Summary

- Incorrectness ***Separation*** Logic (ISL)
  - ➡ IL + SL for ***compositional bug catching***
  - ➡ ***Under-approximate*** analogue of SL
  - ➡ Targets ***memory safety bugs*** (e.g. use-after-free)

ISL Summary

- Incorrectness ***Separation*** Logic (ISL)
  - ➡ IL + SL for ***compositional bug catching***
  - ➡ ***Under-approximate*** analogue of SL
  - ➡ Targets ***memory safety bugs*** (e.g. use-after-free)
- Combining IL+SL: not straightforward
  - ➡ ***invalid frame*** rule!
- Fix: a ***monotonic model*** for frame preservation

ISL Summary

- Incorrectness ***Separation*** Logic (ISL)
  - ➡ IL + SL for ***compositional bug catching***
  - ➡ ***Under-approximate*** analogue of SL
  - ➡ Targets ***memory safety bugs*** (e.g. use-after-free)
- Combining IL+SL: not straightforward
  - ➡ ***invalid frame*** rule!
- Fix: a ***monotonic model*** for frame preservation
- Recovering the ***footprint property*** for completeness

ISL Summary

- Incorrectness ***Separation*** Logic (ISL)
  - ➡ IL + SL for ***compositional bug catching***
  - ➡ ***Under-approximate*** analogue of SL
  - ➡ Targets ***memory safety bugs*** (e.g. use-after-free)
- Combining IL+SL: not straightforward
  - ➡ ***invalid frame*** rule!
- Fix: a ***monotonic model*** for frame preservation
- Recovering the ***footprint property*** for completeness
- ISL-based ***analysis***
  - ➡ ***No-false-positives theorem:***

    All bugs found are true bugs

Part II.

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***

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

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, though merely an academic tool!
- **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

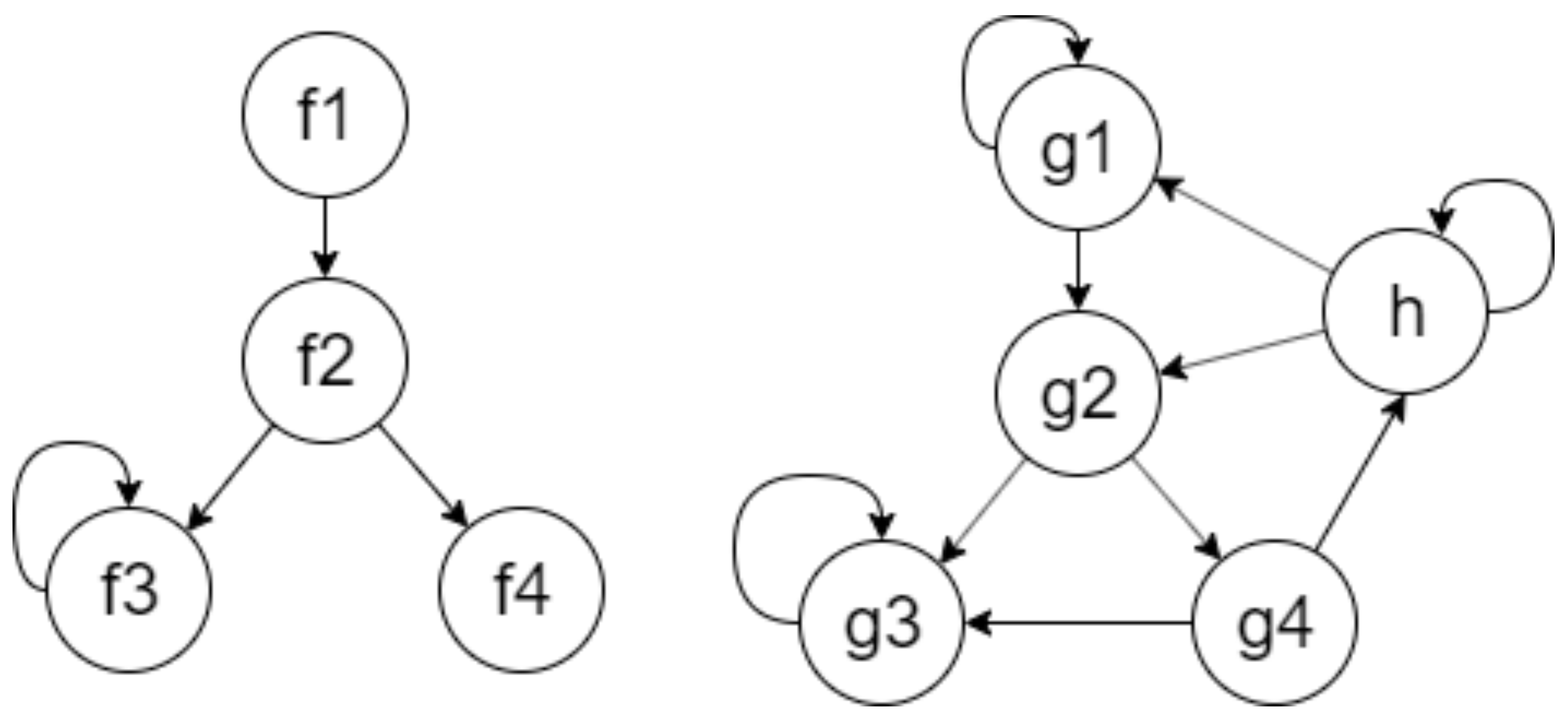
  +

  a method of combining them

Compositional, Begin-Anywhere Analysis

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

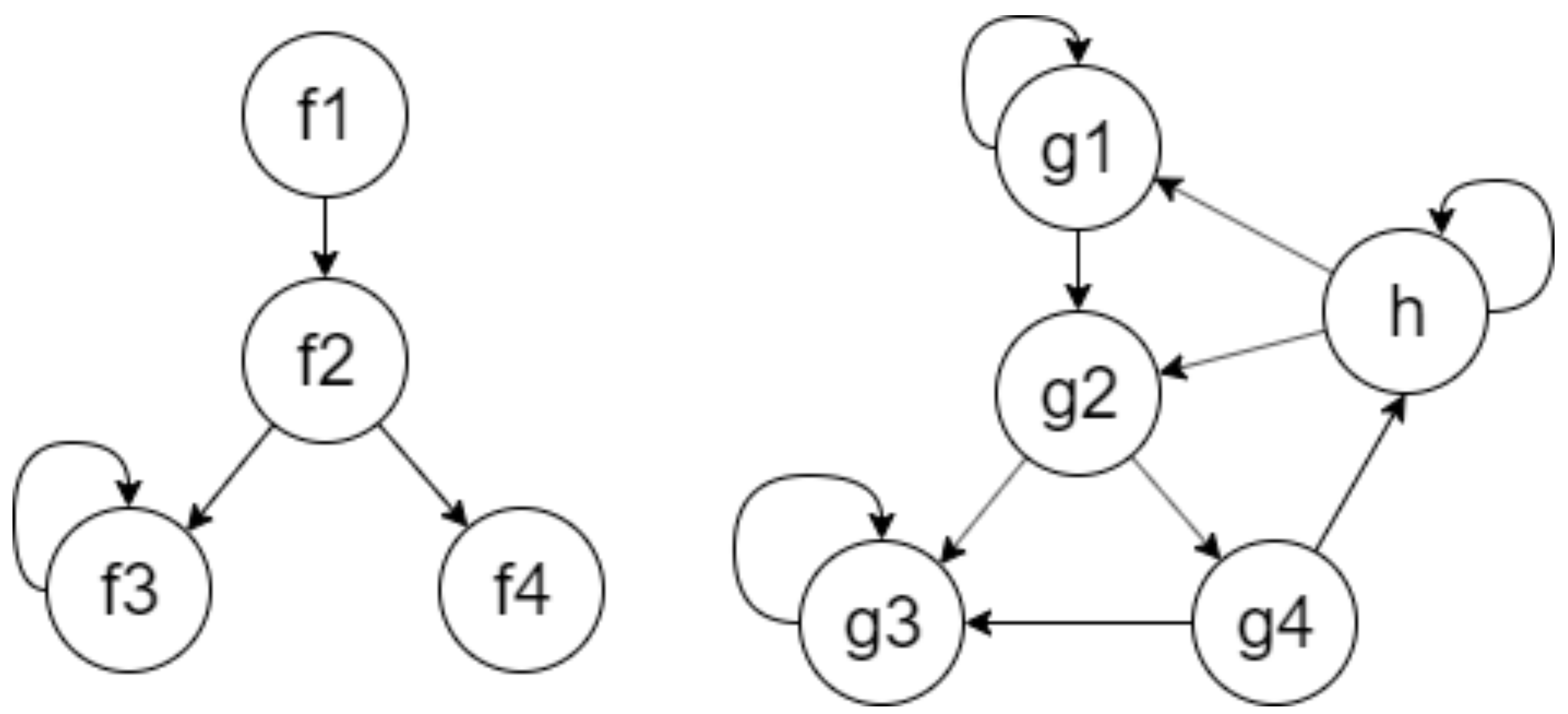
➡ Parts: Procedures



Compositional, Begin-Anywhere Analysis

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

  - ➡ Parts: Procedures

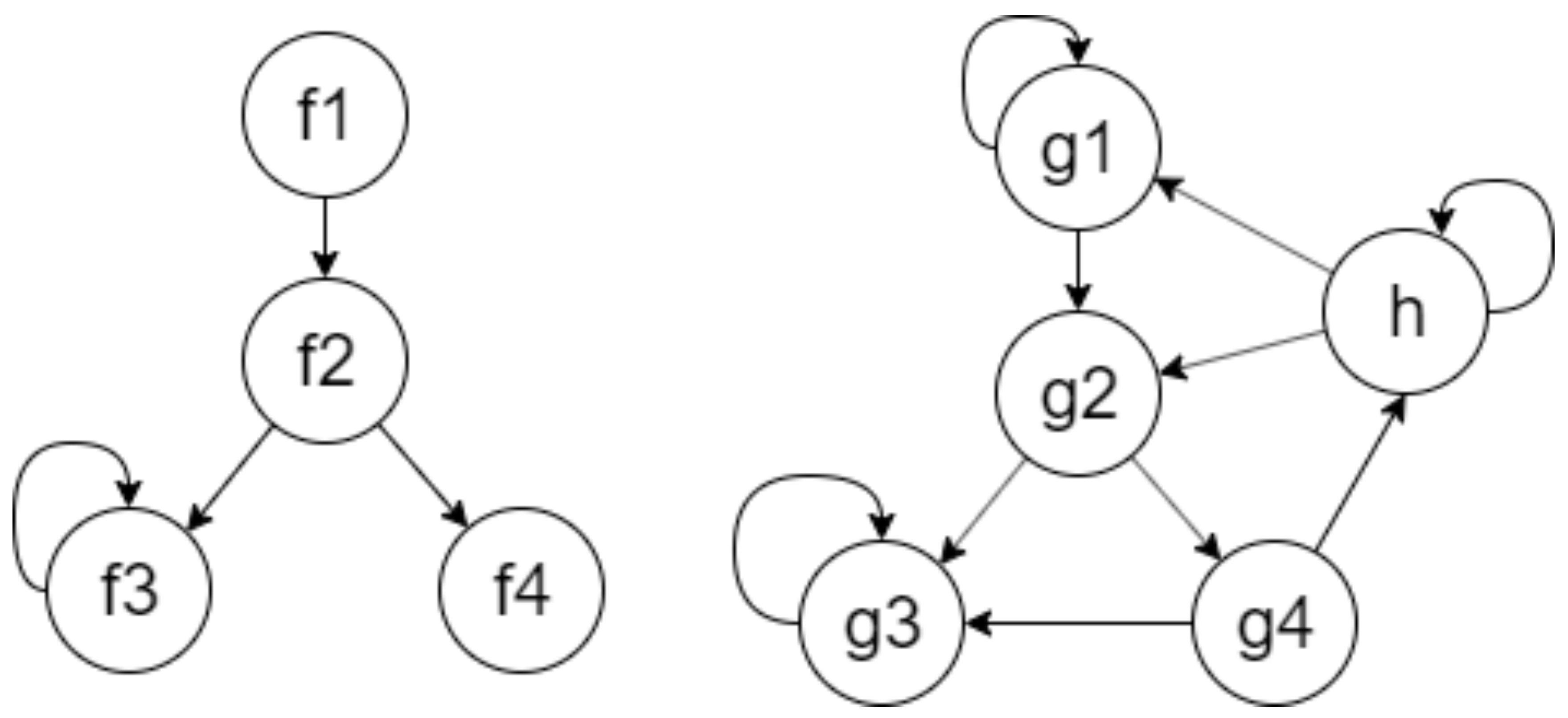


  - ➡ Method: under-approximate bi-abduction

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

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

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)

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

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

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

\+

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

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] `*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)
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`.

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`.

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?

Pulse-X: Bug Reporting

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

Not quite…

Pulse-X: Bug Reporting

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

Pulse-X: Bug Reporting

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

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

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

```
1.void foo(int *x){
2.    *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

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

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

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

Should we report this NPD?

yes

"But I never call foo with null!"

Pulse-X: Bug Reporting

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

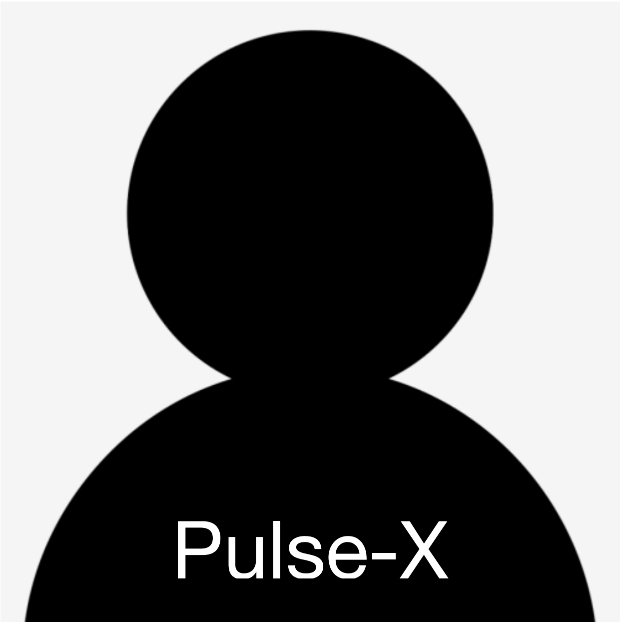
WRITE [x=null] *x = v [er: x=null]

[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

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

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

x=null]

Problem

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

Developer

Pulse-X

“But I never call foo with null!”

“Which bugs shall I report then?”

Pulse-X: Bug Reporting

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

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

x=null]

Problem

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

Solution

Manifest Errors

Developer

Pulse-X

“But I never call foo with null!”

“Which bugs shall I report then?”

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:

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

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: [p] C [er: q] is manifest if when: $q = \exists \vec{X}.\ h_q \wedge \boldsymbol{\pi}_q$:

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: [p] C [er: q] is manifest if when: $q = \exists \overrightarrow{X}.\ h_q \wedge \boldsymbol{\pi}_q$:
  - ➡ $p = \mathrm{emp} \wedge \mathrm{true}$

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: [p] C [er: q] is manifest if when: $q = \exists \overrightarrow{X}.\ h_q \wedge \boldsymbol{\pi}_q$:
  - ➡ $p = \mathrm{emp} \wedge \mathrm{true}$
  - ➡ $\mathrm{sat}(q)$ and $\mathrm{locs}(q) \subseteq \overrightarrow{X}$

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

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

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

- Algorithmically: [p] C [er: q] is manifest if when: $q = \exists \vec{X}.\ h_q \wedge \boldsymbol{\pi}_q$:
  - ➡ $p = \text{emp} \wedge \text{true}$
  - ➡ $\text{sat}(q)$ and $\text{locs}(q) \subseteq \vec{X}$
  - ➡ for all $\vec{v}$: $\text{sat}\ (\vec{v} / \text{flv}(q) \cup \vec{X}\ )$

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

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

Theorem 3.5 (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) $p \equiv \mathsf{emp} \wedge \mathsf{true}$*;*
(2) $\mathsf{sat}(q)$ *holds;*
(3) $\mathsf{locs}(\kappa_q) \subseteq \overrightarrow{X_q}$*, where* $\mathsf{locs}(.)$ *is as defined below; and*
(4) for all $\overrightarrow{v}$, $\mathsf{sat}(\pi_q[\overrightarrow{v} / \overrightarrow{Y} \cup \mathsf{locs}(\kappa_q)])$ *holds, where* $\overrightarrow{Y} = \mathsf{flv}(q)$*.*

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

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

```
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]

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

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

An error triple [p] C [er: q] is <u>latent</u> 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

No-False Positives: Caveat

- Unknown procedures (e.g. where the code is unavailable) are treated as `skip`

No-False Positives: Caveat

- Unknown procedures (e.g. where the code is unavailable) are treated as `skip`
- Incomplete arithmetic solver

Speed (fast but simplistic)	vs	Precision (slow but accurate)

No-False Positives: Caveat

- Unknown procedures (e.g. where the code is unavailable) are treated as `skip`
- Incomplete arithmetic solver

Speed
(fast but simplistic)

vs

Precision
(slow but accurate)

"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."

Part III.

Concurrent Incorrectness Separation Logic (CISL)

&

Concurrent Adversarial Separation Logic (CASL)

Extension 1: Concurrent Incorrectness Separation Logic (CISL)

ISL

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

Extension 1: Concurrent Incorrectness Separation Logic (CISL)

ISL

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

CSL

$$\frac{\{p_1\}\ C_1\ \{q_1\} \qquad \{p_2\}\ C_2\ \{q_2\}}{\{p_1 * p_2\}\ C_1 \,\|\, C_2\ \{q_1 * q_2\}}$$

Extension 1: Concurrent Incorrectness Separation Logic (CISL)

ISL

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

CSL

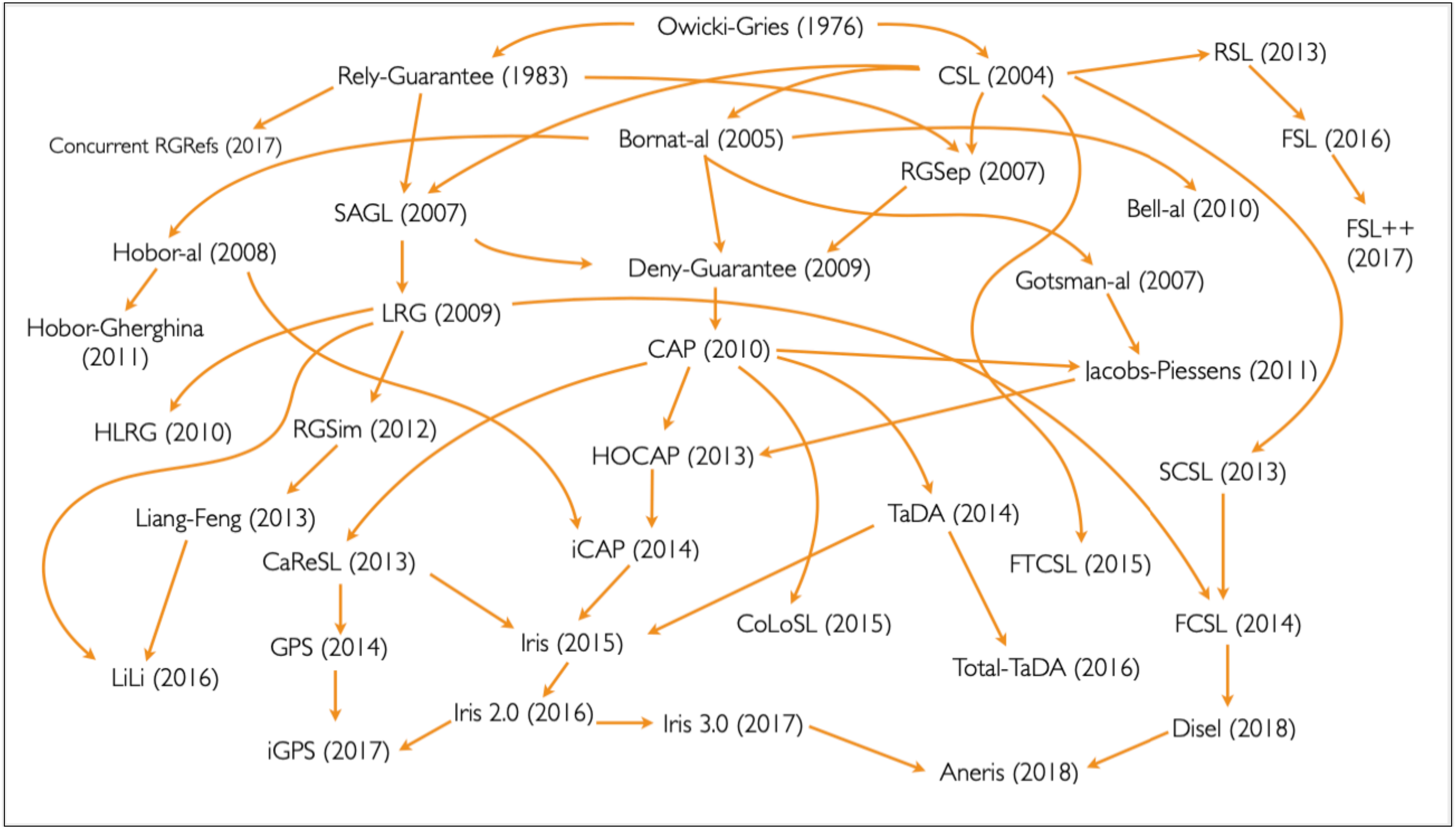
$$\frac{\{p_1\}\ C_1\ \{q_1\} \qquad \{p_2\}\ C_2\ \{q_2\}}{\{p_1 * p_2\}\ C_1 \,||\, C_2\ \{q_1 * q_2\}}$$

CISL

$$\frac{[p_1]\ C_1\ [\varepsilon: q_1] \qquad [p_2]\ C_2\ [\varepsilon: q_2]}{[p_1 * p_2]\ C_1 \,||\, C_2\ [\varepsilon: q_1 * q_2]}$$

Which CISL?

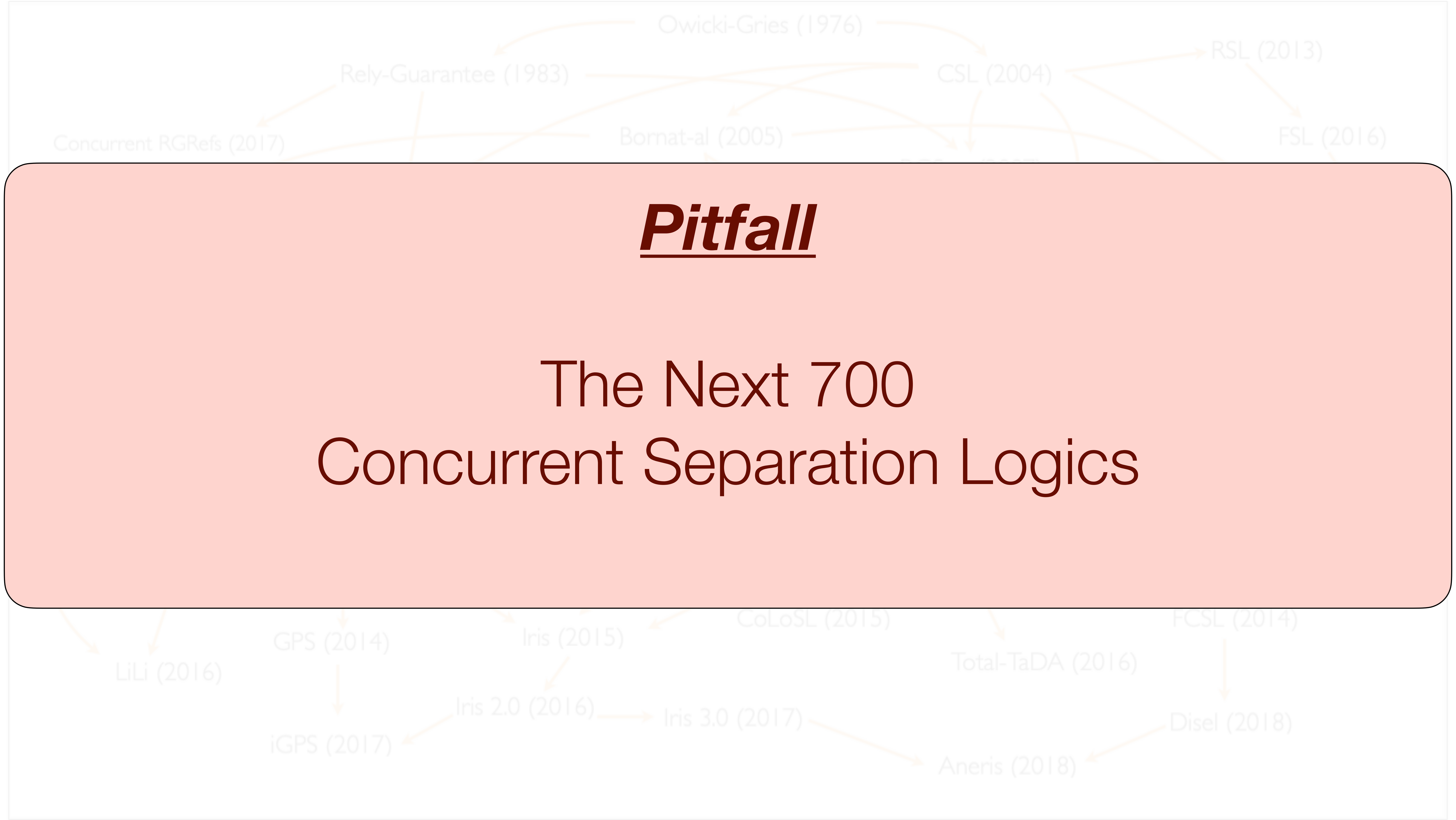
CSL (Correctness) Family Tree…



Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…



Pitfall

The Next 700
Concurrent Separation Logics

Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…

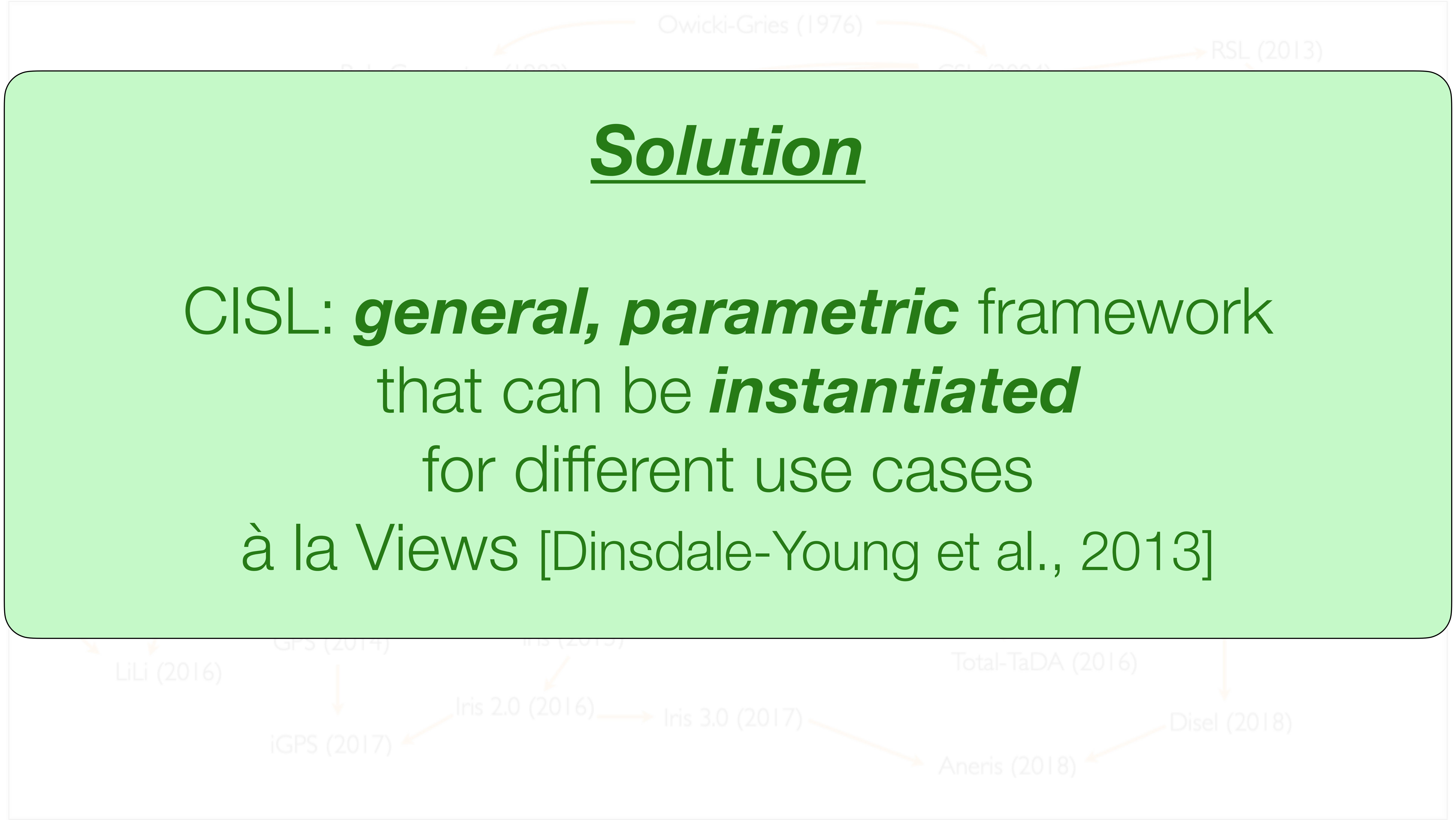
Pitfall

The Next 700
Concurrent ***Incorrectness*** Separation Logics

Graph courtesy of Ilya Sergey

Which CISL?

CSL (Correctness) Family Tree…



Solution

CISL: ***general, parametric*** framework
that can be ***instantiated***
for different use cases
à la Views [Dinsdale-Young et al., 2013]

Graph courtesy of Ilya Sergey

CISL Framework

- ***First*** unifying framework for ***concurrent under-approximate*** reasoning
- ***General*** framework for multiple bug catching analyses
  - ➡ Memory safety errors (e.g. null-pointer exception, use-after-free errors): CISL_{SV}
  - ➡ Races: CISL_{RD}
  - ➡ Deadlocks: CISL_{DD}

CISL Framework

- ***First*** unifying framework for ***concurrent under-approximate*** reasoning
- ***General*** framework for multiple bug catching analyses
  - Memory safety errors (e.g. null-pointer exception, use-after-free errors): $CISL_{SV}$
  - Races: $CISL_{RD}$
  - Deadlocks: $CISL_{DD}$
- Sound: ***no false positives*** (NFP) guaranteed

CISL Framework

- ***First*** unifying framework for ***concurrent under-approximate*** reasoning
- ***General*** framework for multiple bug catching analyses
  - ➡ Memory safety errors (e.g. null-pointer exception, use-after-free errors): CISL_{SV}
  - ➡ Races: CISL_{RD}
  - ➡ Deadlocks: CISL_{DD}
- Sound: ***no false positives*** (NFP) guaranteed
- Underpins ***scalable*** bug-catching tools (NFP for free)
  - ➡ CISL_{RD}: analogous to ***RacerD*** @Meta
  - ➡ CISL_{DD}: analogous to ***DLTool*** @Meta

CISL Framework

- ***First*** unifying framework for ***concurrent under-approximate*** reasoning
- ***General*** framework for multiple bug catching analyses
  - Memory safety errors (e.g. null-pointer exception, use-after-free errors): CISL_{SV}
  - Races: CISL_{RD}
  - Deadlocks: CISL_{DD}
- Sound: ***no false positives*** (NFP) guaranteed
- Underpins ***scalable*** bug-catching tools (NFP for free)
  - CISL_{RD}: analogous to ***RacerD*** @Meta
  - CISL_{DD}: analogous to ***DLTool*** @Meta
- Caveat: cannot detect bugs where there are control flow dependencies between threads

Concurrent Adversarial Separation Logic (CASL)

- *A **general*** framework for ***concurrent under-approximate*** reasoning
- It **subsumes** CISL
- Can handle ***both*** data-agnostic and data-dependent bugs

Concurrent Adversarial Separation Logic (CASL)

- *A **general*** framework for ***concurrent under-approximate*** reasoning
- It **subsumes** CISL
- Can handle ***both*** data-agnostic and data-dependent bugs
- Instantiated to detect ***software exploits/attacks***
  - ➡ Information disclosure attacks (over stacks & heaps)
  - ➡ Buffer overflow attacks (over stacks & heaps)

Concurrent Adversarial Separation Logic (CASL)

- *A **general*** framework for ***concurrent under-approximate*** reasoning
- It **subsumes** CISL
- Can handle ***both*** data-agnostic and data-dependent bugs
- Instantiated to detect ***software exploits/attacks***
  - ➡ Information disclosure attacks (over stacks & heaps)
  - ➡ Buffer overflow attacks (over stacks & heaps)
- Future work:
  - ➡ Automated exploit detection tools
  - ➡ Automatic exploit generation techniques

Conclusions

❖ Incorrectness Separation Logic (ISL)

- ➡ Combining IL and SL for ***compositional bug catching*** (in sequential programs)
- ➡ ***no-false-positives*** theorem

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

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

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
- CISL and CASL
  - Combining ISL and CSL for ***concurrent bug catching***
  - ***no-false-positives*** theorem
  - Race detection, deadlock detection, exploit detection

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

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
- CISL and CASL
  - ➡ Combining ISL and CSL for ***concurrent bug catching***
  - ➡ ***no-false-positives*** theorem
  - ➡ Race detection, deadlock detection, exploit detection

Thank You for Listening!

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