Incorrectness Logic & Under-Approximation: Foundations of Bug Detection

Azalea Raad
Imperial College London

Program Analysis Course
University of Pisa
March 2025

azalea@imperial.ac.uk

SoundAndComplete.org

VERITAS

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}

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 ⊆ q

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

Hoare Logic (HL)

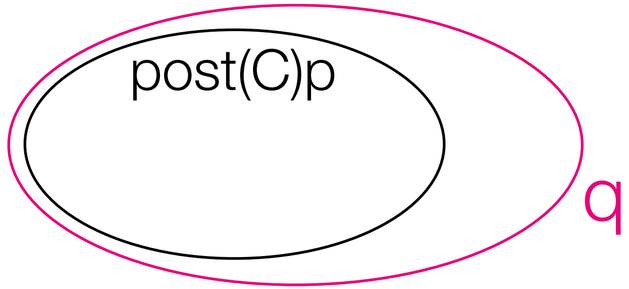
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $\mathrm{post}(C)p \subseteq q$

q *over-approximates* post(C)p

Hoare Logic (HL)

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

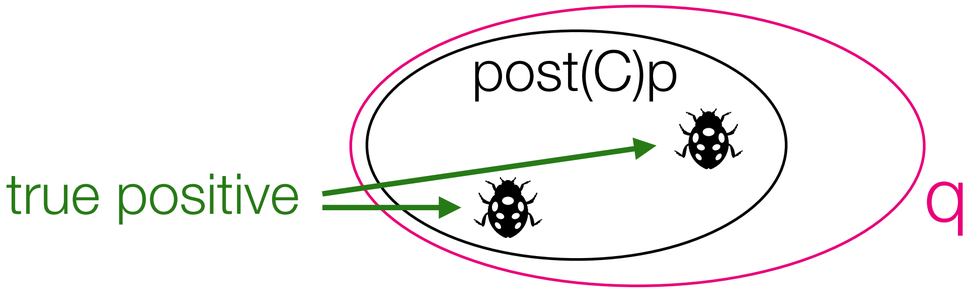
q *over-approximates* post(C)p



Hoare Logic (HL)

Hoare triples {p} C {q} *iff* post(C)p ⊆ q

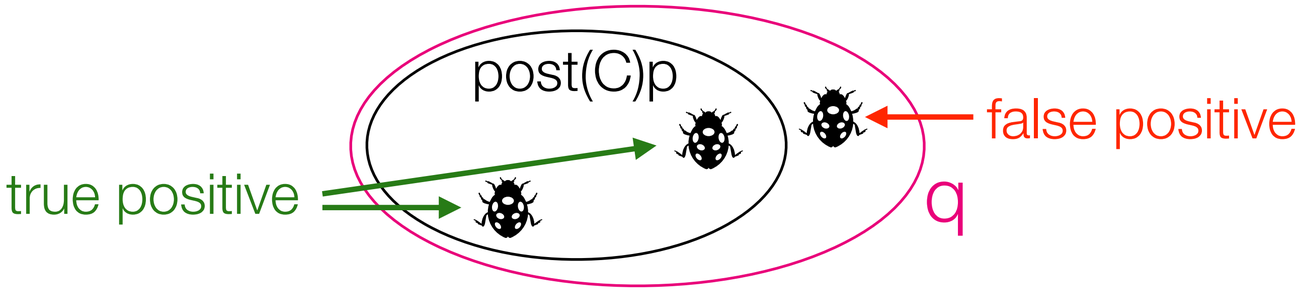
q *over-approximates* post(C)p



Hoare Logic (HL)

Hoare triples {p} C {q} *iff* post(C)p ⊆ 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 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

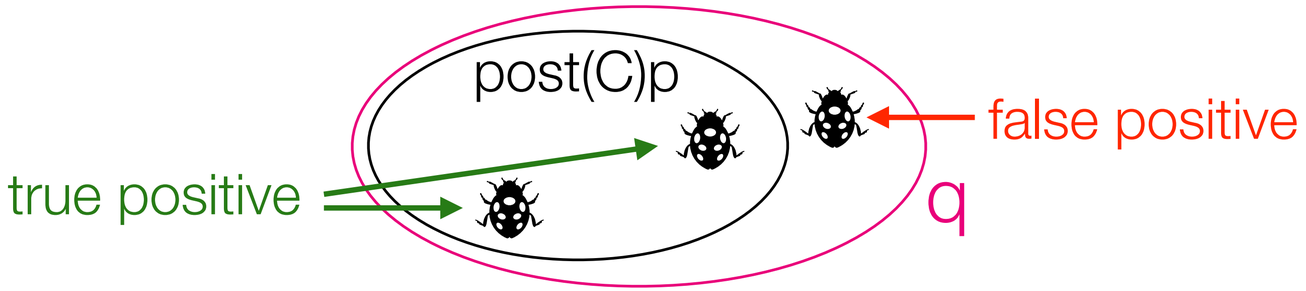
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

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



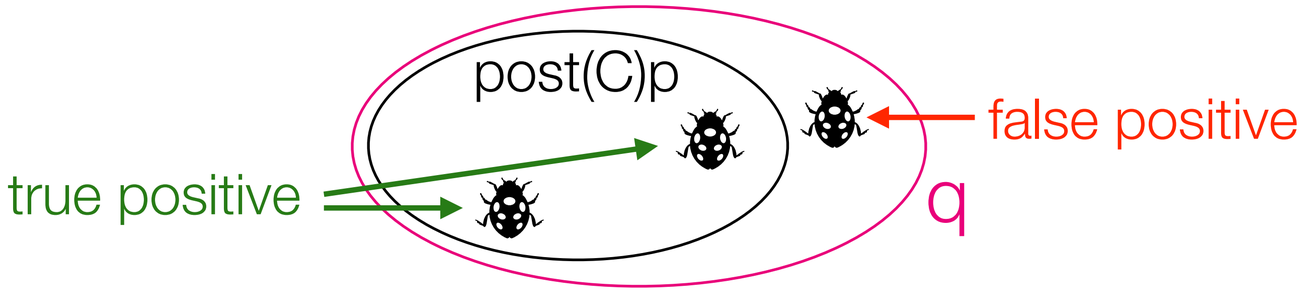
Incorrectness triples $[p]\ C\ [q]$ *iff* $post(C)p \supseteq q$

q *under-approximates* post(C)p

Incorrectness Logic (IL)

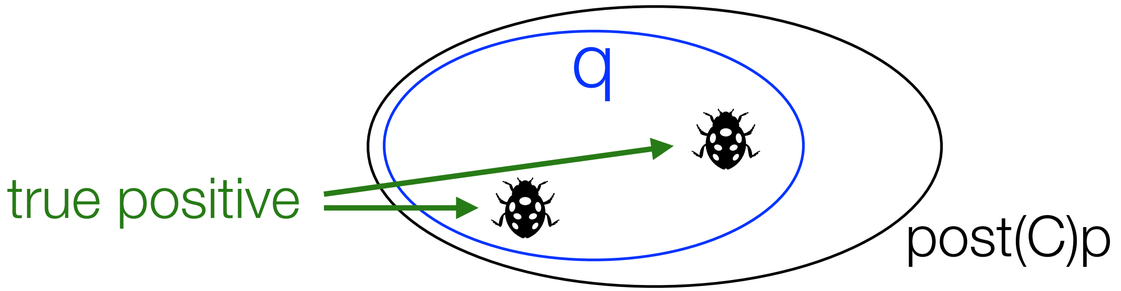
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $\text{post}(C)p \subseteq q$

q *over-approximates* post(C)p



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

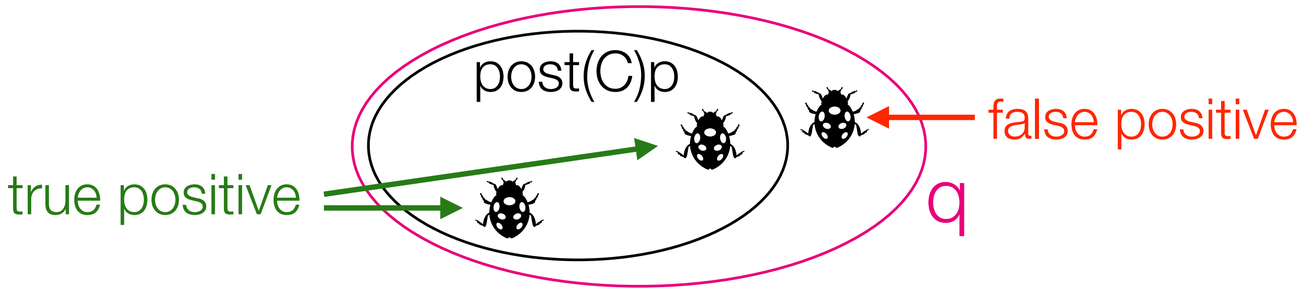
q *under-approximates* post(C)p



Incorrectness Logic (IL)

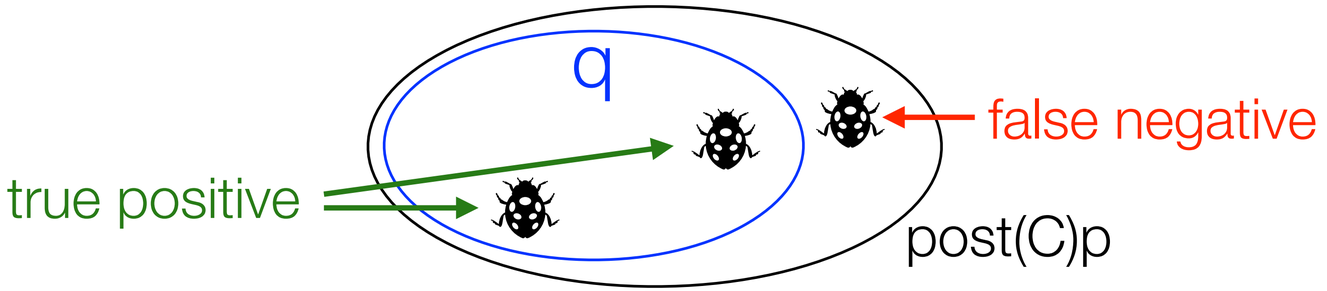
Hoare triples $\{p\}\ C\ \{q\}$ *iff* $post(C)p \subseteq q$

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

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

$[p]$ error( ) $[er: p]$

Incorrectness Logic (IL)

$$[p]\ C\ [\varepsilon\colon q] \quad \textit{iff} \quad \mathrm{post}(C, \varepsilon)p \supseteq q$$

Incorrectness Logic (IL)

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

Equivalent Definition (reachability)

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

IL Proof Rules and Principles (Sequencing)

$$\frac{[p]\ C_1\ [er:\ q]}{[p]\ C_1;\ C_2\ [er:\ q]}$$

- **Short-circuiting** semantics for errors

IL Proof Rules and Principles (Sequencing)

$$\frac{[p]\ C_1\ [er:\ q]}{[p]\ C_1;\ C_2\ [er:\ q]}$$

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

- **Short-circuiting** semantics for errors

IL Proof Rules and Principles (Branches)

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

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

IL Proof Rules and Principles (Branches)

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

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

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

Example

[y=0] **if** (`is-even(x)`) [ok: y=42]

y:= 42

?

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s' ∈ p. (s',s) ∈ [C]ε

Example

p

[y=0] **if** (`is-even(x)`) [ok: y=42] ?

q

y:= 42

q={(x=0, y=42), (x=1, y=42), (x=2, y=42), …}

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s’ ∈ p. (s’,s) ∈ [C]ε

Example

p

[y=0] **if** (`is-even(x)`) [ok: y=42] ?

y:= 42

q

q={(x=0, y=42), (x=1, y=42), (x=2, y=42), …}

✔ ✘ ✔

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s’ ∈ p. (s’,s) ∈ [C]ε

Example

p

[y=0] **if** (`is-even(x)`) [ok: y=42] ✘

y:= 42

q

q={(x=0, y=42), (x=1, y=42), (x=2, y=42), …}

✔ ✘ ✔

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s’ ∈ p. (s’,s) ∈ [C]ε

Example

[y=0] **if** (`is-even(x)`) [ok: y=42 ∧ even(x)]
y:= 42

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s' ∈ p. (s',s) ∈ [C]ε

Example

p

[y=0] **if** (`is-even(x)`)

y:= 42

q

[ok: y=42 ∧ even(x)]

q={(x=0, y=42), (x=2, y=42), (x=4, y=42), …}

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s’ ∈ p. (s’,s) ∈ [C]ε

Example

p

[y=0] **if** (`is-even(x)`)

y:= 42

q

[ok: y=42 ∧ even(x)] ✔

q={(x=0, y=42), (x=2, y=42), (x=4, y=42), …}

✔ ✔ ✔

[p] C [ε: q] *iff* ∀ s ∈ q. ∃ s’ ∈ p. (s’,s) ∈ [C]ε

IL Proof Rules and Principles (Loops)

$$\frac{}{[p]\ C^*\ [ok: p]}\ \text{(Unroll-Zero)}$$

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

IL Proof Rules and Principles (Loops)

$$\frac{}{[p]\ C^*\ [ok:\ p]}\ \text{(Unroll-Zero)}$$

$$\frac{[p]\ C^*;\ C\ [\varepsilon:\ q]}{[p]\ C^*\ [\varepsilon:\ q]}\ \text{(Unroll-Many)}$$

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

IL Proof Rules and Principles (Loops)

$$\frac{}{[p]\ C^*\ [ok: p]}\ \text{(Unroll-Zero)} \qquad \frac{[p]\ C^*; C\ [\varepsilon: q]}{[p]\ C^*\ [\varepsilon: q]}\ \text{(Unroll-Many)}$$

- **Bounded unrolling of loops** (this is a **sound** under-approximation)
- **Scalable** bug detection!

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

IL Proof Rules and Principles (Loops continued)

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

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

IL Proof Rules and Principles (Loops continued)

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

- Loop **invariants** are inherently **over-approximate**
- Reason about loops **under-approximately** via **sub-variants**

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

Example

[x=0] (x++)*; **if** (x==2,000,000) error; [er: x=2,000,000]

Example

[x=0] (x++)*; **if** (x==2,000,000) error; [er: x=2,000,000]

[x=0]

(x++)*;

if (x==2,000,000)

error;

Example

[x=0] (x++)*; **if** (x==2,000,000) error; [er: x=2,000,000]

p(n): x=n

[x=0]
  (x++)*; // Backwards-Variant
[ok: x=2,000,000]
  if (x==2,000,000)
    error;

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

Example

[x=0] (x++)*; **if** (x==2,000,000) error; [er: x=2,000,000] ✔

p(n): x=n

[x=0]
(x++)*; // Backwards-Variant
[ok: x=2,000,000]
if (x==2,000,000)
error;
[er: x=2,000,000]

$$\frac{\forall n \in \mathbb{N}.\ [p(n)]\ C\ [ok: p(n+1)] \qquad k \in \mathbb{N}}{[p(0)]\ C^*\ [ok: p(k)]}\ \text{(Backwards-Variant)}$$

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon\colon q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon\colon q]}\ \text{(Cons)}$$

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

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon\colon q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon\colon q]}\ \text{(Cons)}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

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

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p']\ C\ [\varepsilon\colon q'] \quad q' \supseteq q}{[p]\ C\ [\varepsilon\colon q]}\ \text{(Cons)}$$

$$\frac{[p]\ C\ [\varepsilon\colon q_1 \vee q_2]}{[p]\ C\ [\varepsilon\colon q_1]}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

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

IL Proof Rules and Principles (Consequence)

$$\frac{p' \subseteq p \quad [p'] \; C \; [\varepsilon: q'] \quad q' \supseteq q}{[p] \; C \; [\varepsilon: q]} \text{ (Cons)}$$

$$\frac{p' \supseteq p \quad \{p'\} \; C \; \{q'\} \quad q' \subseteq q}{\{p\} \; C \; \{q\}} \text{ (HL-Cons)}$$

$$\frac{[p] \; C \; [\varepsilon: q_1 \vee q_2]}{[p] \; C \; [\varepsilon: q_1]}$$

- **Shrink** the **post** (e.g. drop disjuncts)
- **Scalable** bug detection!

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

Incorrectness Logic: Summary

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

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

– 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 ***r***

Our Solution

Incorrectness Separation Logic

Part II.

Incorrectness Separation Logic (ISL)

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;
```
post: {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***

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

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

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

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

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

n(n-1)/2 conjuncts !

What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***
☛ ideal for heap-manipulating programs with ***aliasing***

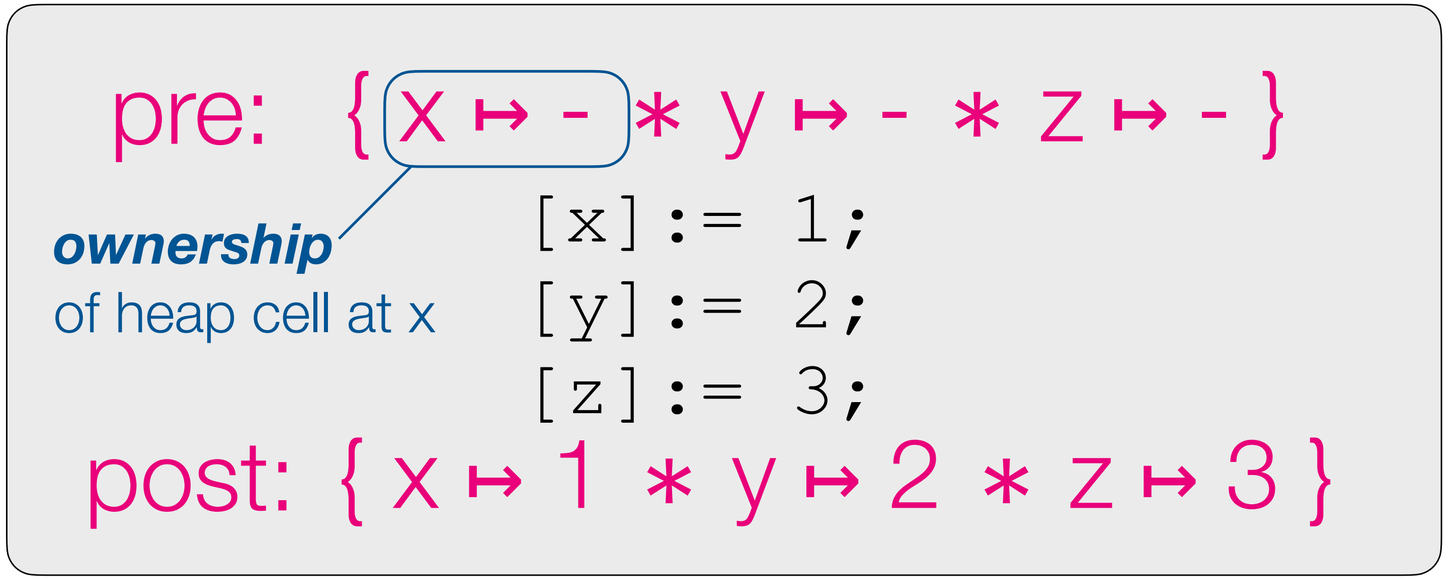
pre: { x ↦ - ∗ y ↦ - ∗ z ↦ - }

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

post: { x ↦ 1 ∗ y ↦ 2 ∗ z ↦ 3 }

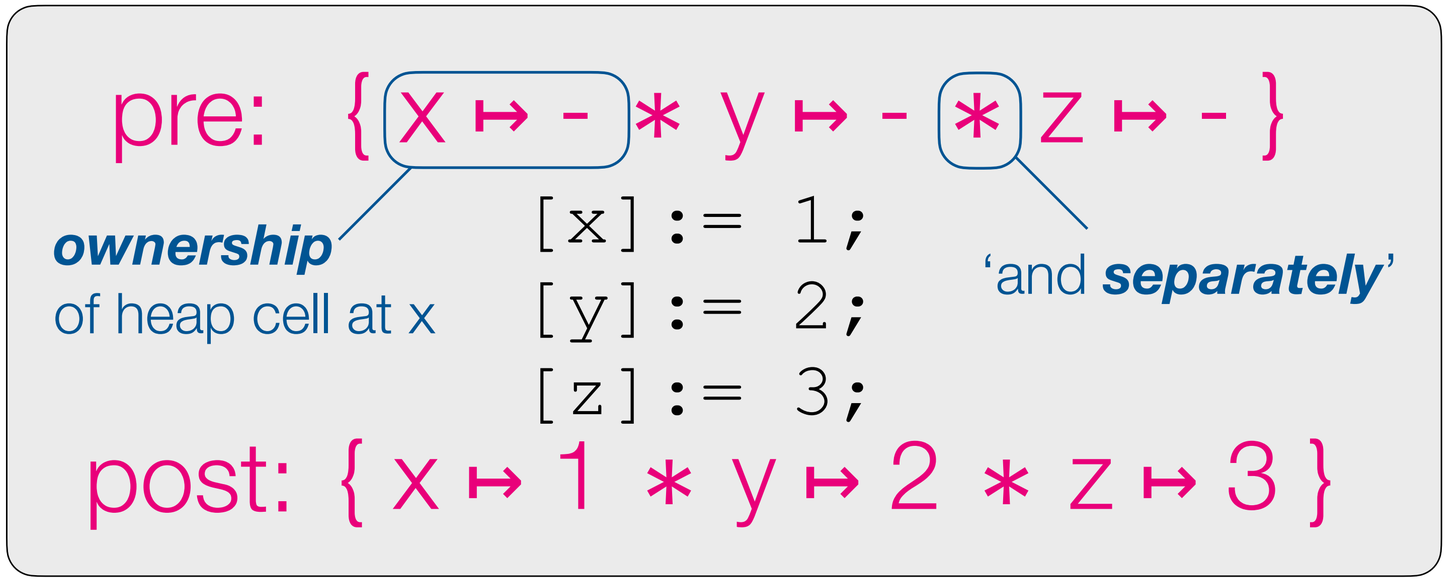
What Is Separation Logic (SL)?

SL : ***Local*** & ***compositional*** reasoning via ***ownership*** & ***separation***
☛ ideal for heap-manipulating programs with ***aliasing***



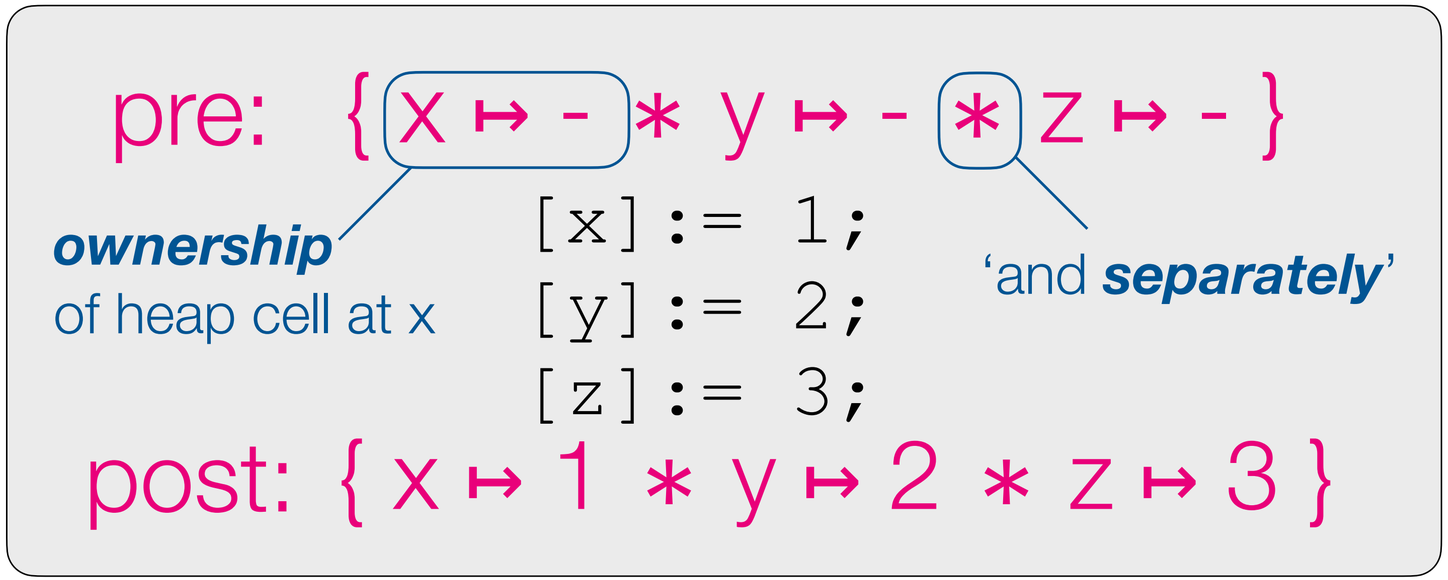
What Is Separation Logic (SL)?

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***



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

The Essence of Separation Logic (SL)

Frame Rule

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

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

The Essence of Separation Logic (SL)

Frame Rule

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

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

Local Axioms

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

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

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

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

Incorrectness Separation Logic (ISL)

IL

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

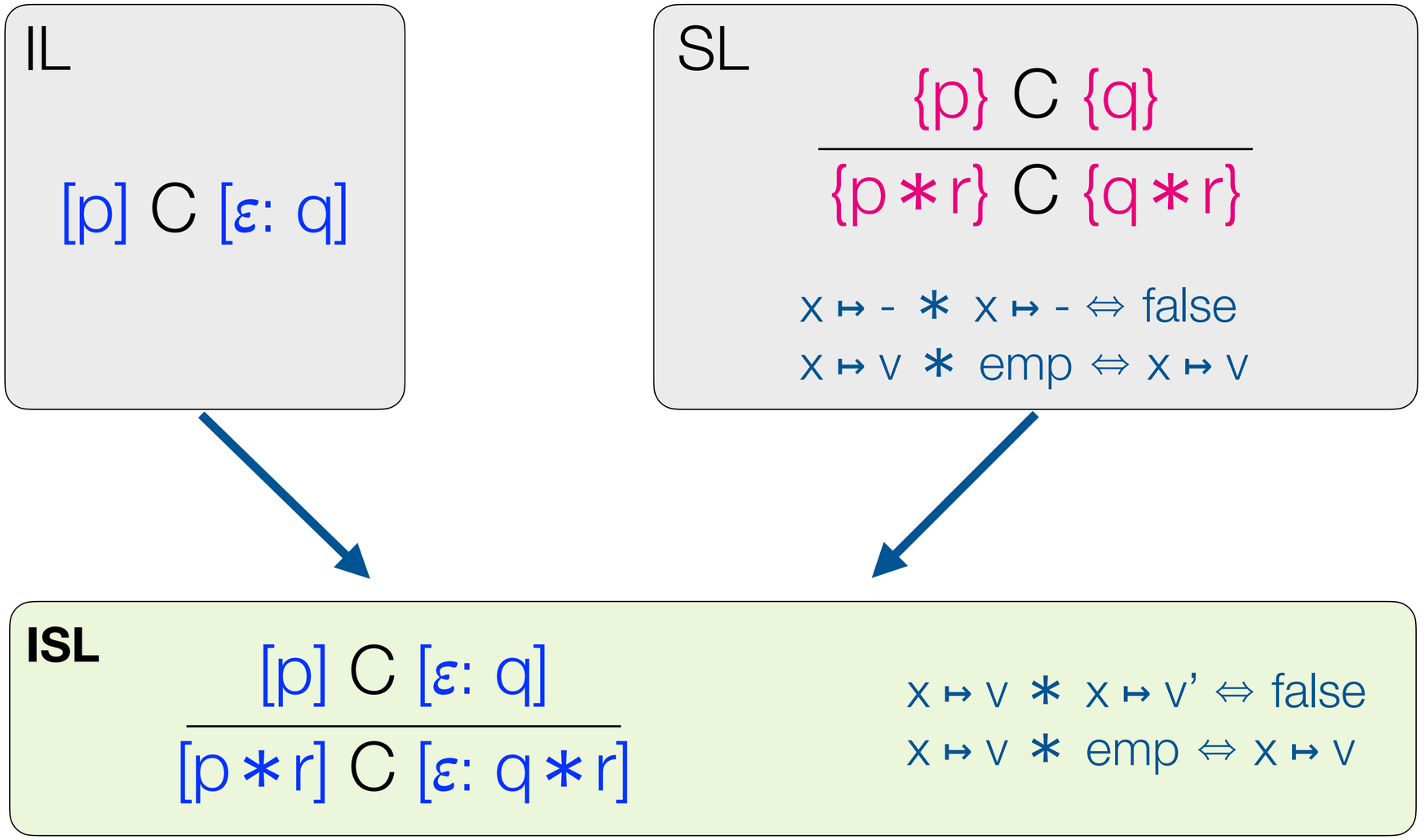
SL

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

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

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

Incorrectness Separation Logic (ISL)



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

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

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

null-pointer dereference error

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

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

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

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

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

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

ALLOC

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

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

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

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

ALLOC

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

FREE

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

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

ISL: Local Axioms (First Attempt)

WRITE

[x ↦ v'] [x]:= v [ok: x ↦ v]

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

null-pointer dereference error

READ

[x ↦ v] y:= [x] [ok: x ↦ v∧y=v]

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

ALLOC

[emp] x:= alloc() [ok:∃l. l ↦ v ∧ x=l]

FREE

[x ↦ v] free(x) [ok: emp]

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

ISL: Local Axioms (First Attempt)

ISL

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

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

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

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

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

ISL: Local Axioms (First Attempt)

ISL

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

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

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

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

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

ISL: Local Axioms (First Attempt)

ISL

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

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

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

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

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

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

ISL: Local Axioms (First Attempt)

ISL

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

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

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

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

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

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

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

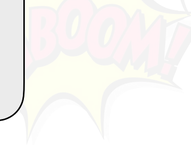
$$x \mapsto v\ *\ x \mapsto v' \Leftrightarrow \text{false}$$

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

Solution:
Track Deallocated Locations!

(Frame)

(Cons)



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

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

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!

Solution: Track Deallocated Locations!

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

x is ***deallocated***

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

$p \ * \ emp \Leftrightarrow p$

Solution: Track Deallocated Locations!

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

x is ***deallocated***

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

$p * emp \Leftrightarrow p$

$x \mapsto v * x \not\mapsto \Leftrightarrow$ false

$x \not\mapsto * x \not\mapsto \Leftrightarrow$ false

Solution: Track Deallocated Locations!

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

Solution: Track Deallocated Locations!

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

Solution: Track Deallocated Locations!

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

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

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

ISL: Local Axioms

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

FREE

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

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

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

double-free error

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

double-free error

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

WRITE

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

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

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

double-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] $[er: x \not\mapsto]$

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

double-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] $[er: x \not\mapsto]$

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

ALLOC

ISL: Local Axioms

FREE

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

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

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

double-free error

WRITE

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

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

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

READ

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

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

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

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)
- 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 III.

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, 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)

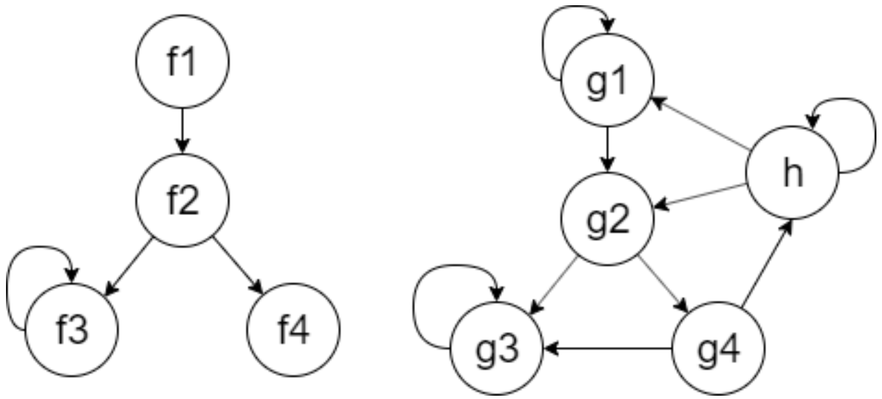
Compositional, Begin-Anywhere Analysis

- 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

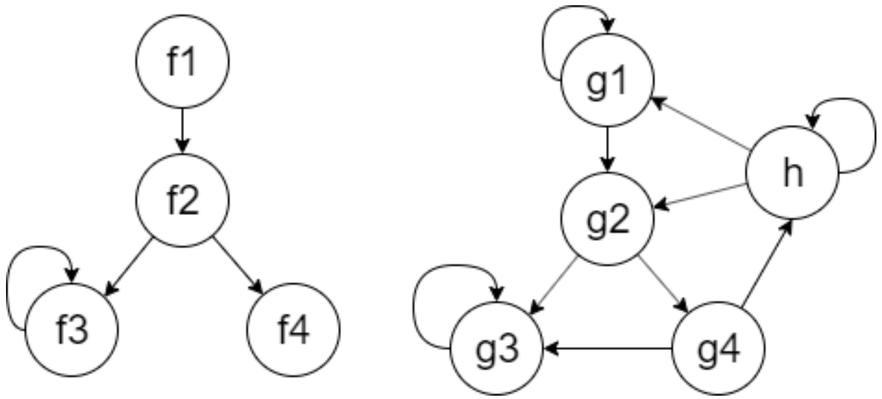
➡ Parts: Procedures



Compositional, Begin-Anywhere Analysis

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

➡ Parts: Procedures

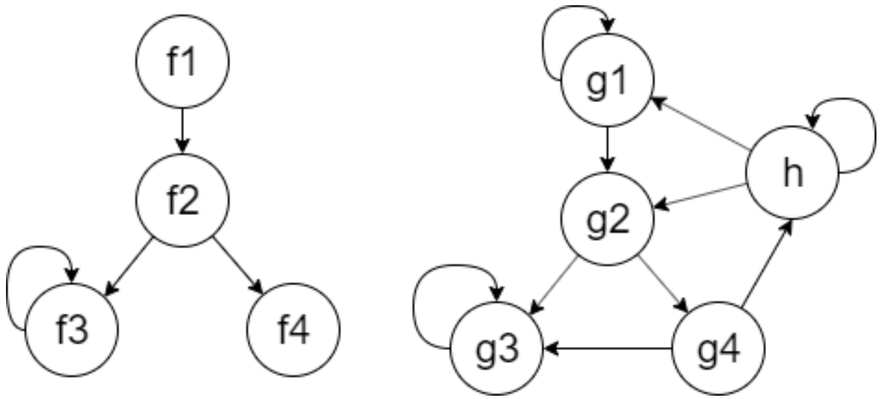


➡ Method: under-approximate bi-abduction

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

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

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

```
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

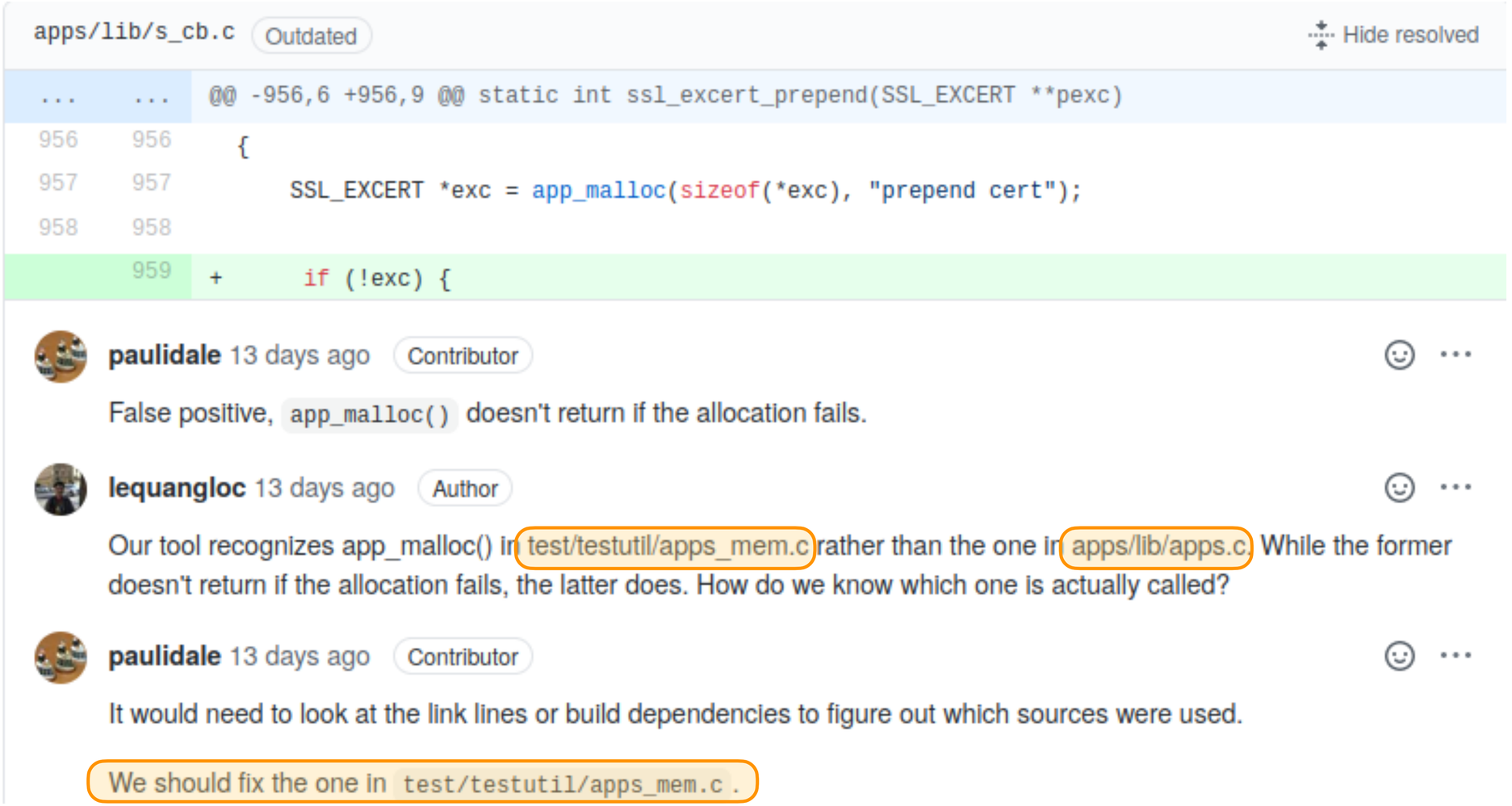
Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

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

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

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]

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;
}
```

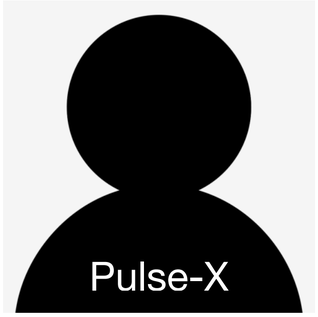
WRITE [x=null] *x = v [er: x=null]

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

Should we report this NPD?

yes

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:

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

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

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

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

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

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

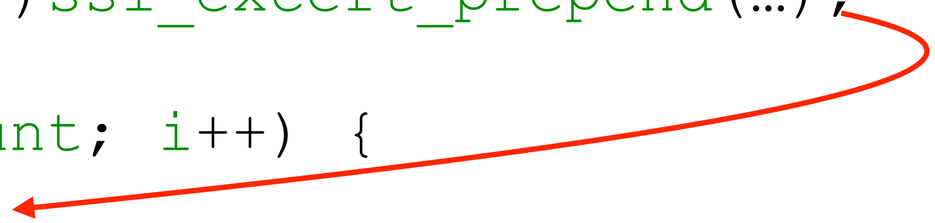
An error triple [p] C [er: q] is latent iff it is not manifest

Pulse-X: Latent Error

```
1.int chopup_args(ARGS *args,…){
      …
2.    if (args->count == 0 ) {
3.      args->count=20;
4.      args->data= (char**)ssl_excert_prepend(…);
5.    }
5.    for (i=0; i<args->count; i++) {
6.      args->data[i]=NULL;
        …
      }
```

Pulse-X: Latent Error

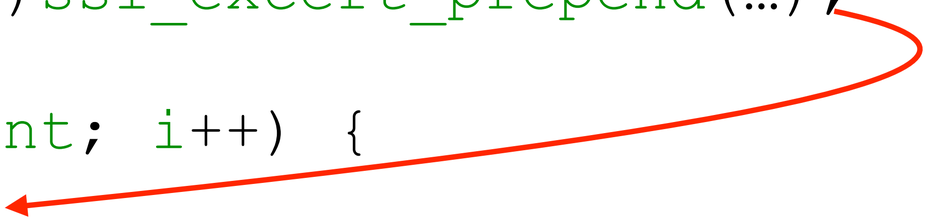
```
1.int chopup_args(ARGS *args,…){
      …
2.    if (args->count == 0 ) {
3.      args->count=20;
4.      args->data= (char**)ssl_excert_prepend(…);
5.    }
5.    for (i=0; i<args->count; i++) {
6.      args->data[i]=NULL;
        …
      }
```



null pointer dereference

Pulse-X: Latent Error

```
1.int chopup_args(ARGS *args,…){
      …
2.    if (args->count == 0 ) {
3.      args->count=20;
4.      args->data= (char**)ssl_excert_prepend(…);
5.    }
5.    for (i=0; i<args->count; i++) {
6.      args->data[i]=NULL;
        …
      }
```



null pointer dereference

Latent Error:

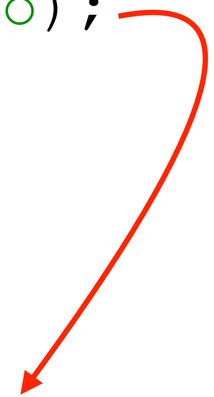
only calls with `args->count==0` can trigger the error

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;
}
```

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

Pulse-X performs under-approximation with bounded loop unrolling

does nothing when `io` is null

leaks `ssl_bio`

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."

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

VERITAS

Part IV.

UNTer: **U**nder-approximate **N**on-T**er**mination

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 must not-terminate
  - ➡ Inherently **under-approximate**

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

Under-approximate Non-Termination Logic (UNTer)

- 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

UNTer Divergence Proof Rules

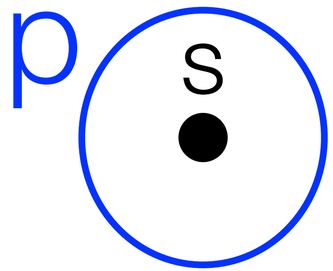
$$[p]\ C\ [\infty]$$

C has divergent traces starting from p

UNTer Divergence Proof Rules

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

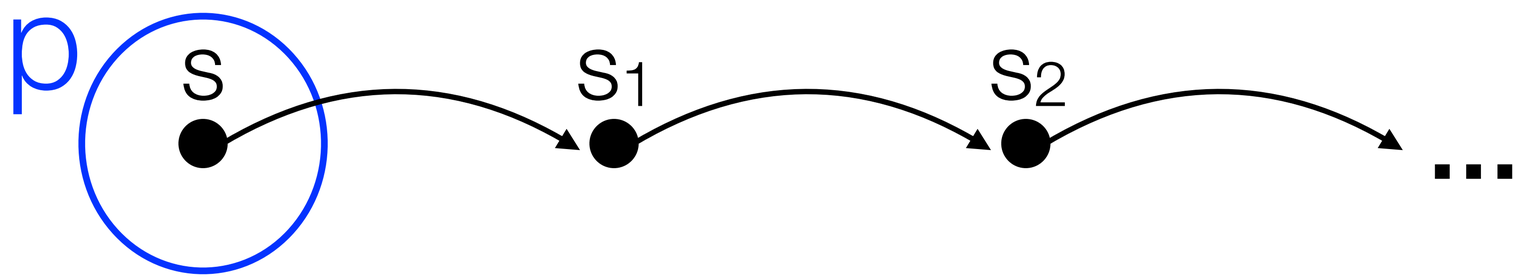
C has divergent traces starting from p



UNTer Divergence Proof Rules

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

C has divergent traces starting from p



UNTer Divergence Proof Rules (Sequencing)

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

UNTer Proof Rules and Principles

UNTer Proof Rules

=

(Under-Approximate) IL/ISL Proof Rules

+

Divergence (Non-Termination) Rules

UNTer Divergence Proof Rules (Sequencing)

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

UNTer Divergence Proof Rules (Sequencing)

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

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

UNTer Divergence Proof Rules (Branches)

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

UNTer Divergence Proof Rules (Branches)

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

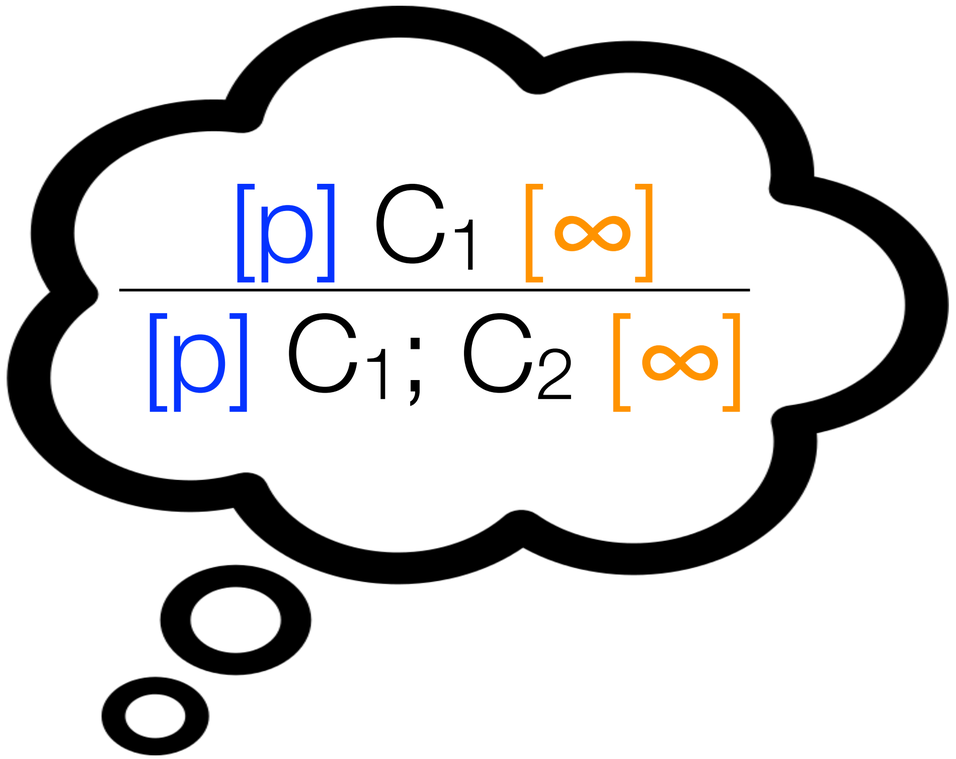
- **Drop paths/branches** (this is a **sound** under-approximation)
- **Scalable** bug detection!

UNTer Divergence Proof Rules (Loops — first attempt)

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

UNTer Divergence Proof Rules (Loops — first attempt)

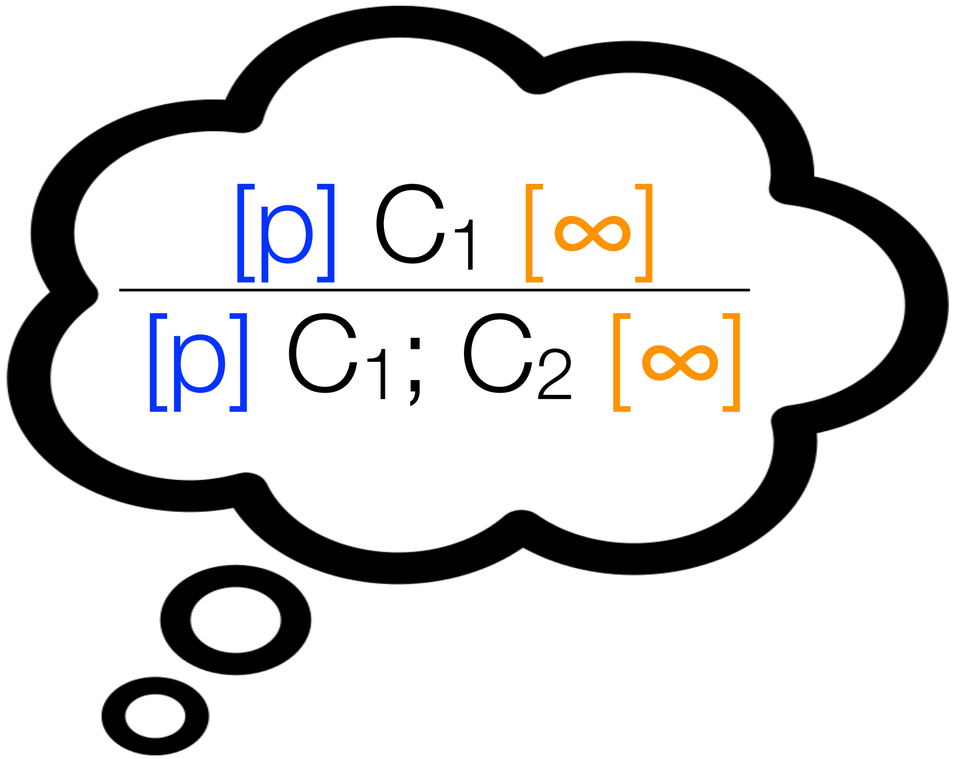
$$\frac{[p]\ C;\ C^*\ [\infty]}{[p]\ C^*\ [\infty]}$$



UNTer Divergence Proof Rules (Loops — first attempt)

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

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}\ \text{(derived)}$$



UNTer Divergence Proof Rules (Loops — first attempt)

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

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}\ \text{(derived)}$$

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

UNTer Divergence Proof Rules (Loops — first attempt)

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

$$\frac{[p]\ C\ [\infty]}{[p]\ C^*\ [\infty]}\ \text{(derived)}$$

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

UNTer Divergence Proof Rules (While Loops — first attempt)

$[p \wedge b]$ while(b) C $[\infty]$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

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

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{[p \wedge b]\ (\text{assume}(b);\ C)^*;[\infty]}{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}$$

$$[p \wedge b]\ \text{while}(b)\ C\ [\infty]$$

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

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{[p \wedge b]\ (\text{assume}(b);\ C)^*;[\infty]}{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}$$

$$[p \wedge b]\ \text{while}(b)\ C\ [\infty]$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

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

$$\frac{\dfrac{[p \wedge b]\ (\text{assume}(b);\ C)^*;[\infty]}{[p \wedge b]\ (\text{assume}(b);\ C)^*;\ \text{assume}(!b)\ [\infty]}}{[p \wedge b]\ \text{while}(b)\ C\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p \wedge b]\ \text{assume(b); C}\ [\text{ok: } p \wedge b]}{\dfrac{[p \wedge b]\ (\text{assume(b); C})^*;[\infty]}{\dfrac{[p \wedge b]\ (\text{assume(b); C})^*;\ \text{assume(!b)}\ [\infty]}{[p \wedge b]\ \text{while(b) C}\ [\infty]}}}$$

$$\frac{\vdash_B [p]\ \text{C}\ [\text{ok: } p]}{[p]\ \text{C}^*\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p \wedge b]\ \text{assume(b); C}\ [\text{ok: } p \wedge b]}{\dfrac{[p \wedge b]\ (\text{assume(b); C})^*;[\infty]}{\dfrac{[p \wedge b]\ (\text{assume(b); C})^*;\ \text{assume(!b)}\ [\infty]}{[p \wedge b]\ \text{while(b) C}\ [\infty]}}}$$

$$\vdash_B [p \wedge b]\ \text{assume(b)}\ [\text{ok: } p \wedge b]$$

$$\frac{\vdash_B [p]\ C_1\ [\text{ok: } r] \qquad \vdash_B [r]\ C_2\ [\varepsilon\text{: } q]}{[p]\ C_1; C_2\ [\varepsilon\text{: } q]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{\vdash_B [p \wedge b]\ \text{assume(b); } C\ [ok: p \wedge b]}}{[p \wedge b]\ (\text{assume(b); } C)^*;[\infty]}}{[p \wedge b]\ (\text{assume(b); } C)^*;\ \text{assume(!b)}\ [\infty]}}{[p \wedge b]\ \text{while(b) } C\ [\infty]}$$

$$\vdash_B [p \wedge b]\ \text{assume(b)}\ [ok: p \wedge b]$$

$$\dfrac{\vdash_B [p]\ C_1\ [ok: r] \qquad \vdash_B [r]\ C_2\ [\varepsilon: q]}{[p]\ C_1; C_2\ [\varepsilon: q]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{[p \wedge b]\ \text{while(b) C}\ [\infty]}$$

while (b) C $\equiv$ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (While Loops — first attempt)

$$\frac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{[p \wedge b]\ \text{while(b) C}\ [\infty]}$$

while (b) C ≡ (assume(b); C)*; assume(!b)

UNTer Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$\frac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{[p]\ \text{while(b)}\ C\ [\infty]}$$

UNTer Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$[\ x > 0\ ]\ \texttt{while(x > 0) x--}\ \ [\infty]$$

$$\frac{\vdash_B [p \wedge b]\ C\ [ok: p \wedge b]}{[p]\ \texttt{while(b)}\ C\ [\infty]}$$

UNTer Divergence Proof Rules (Loops — first attempt)

Program `while(x > 0) x--` always terminates. But…

$$\frac{\vdash_B [x > 0]\ \texttt{x--}\ [ok\colon x > 0]}{[\ x > 0\ ]\ \texttt{while(x > 0) x--}\ [\infty]}$$

$$\frac{\vdash_B [p \wedge b]\ C\ [ok\colon p \wedge b]}{[p]\ \texttt{while(b)}\ C\ [\infty]}$$

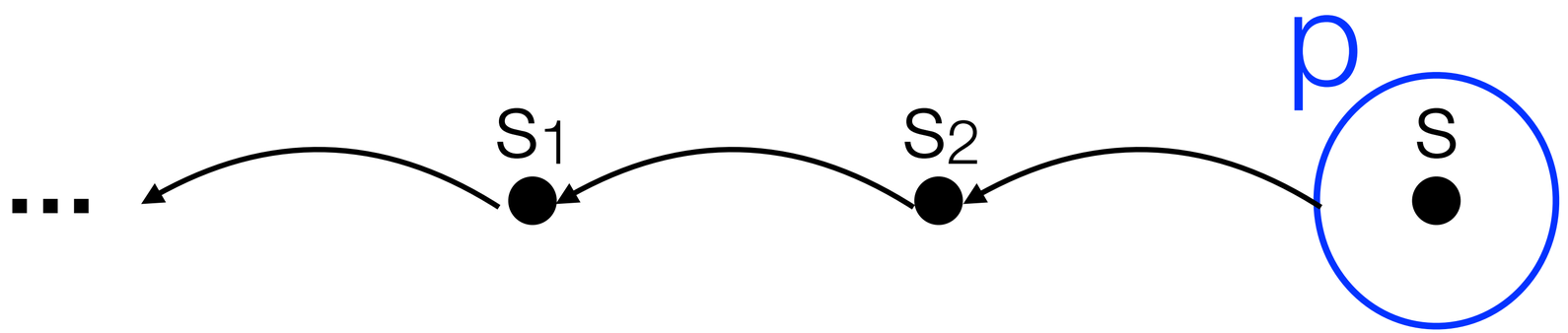
$$\vdash_B [p]\ C\ [\varepsilon\colon q]$$

iff

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

Problem

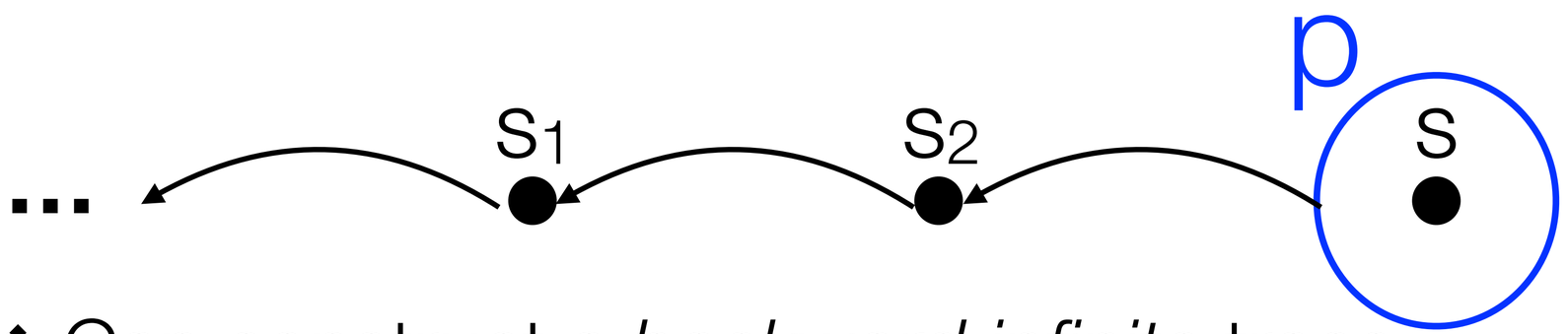
- Premise: p reached by executing C on some p
- I.e. in the **backward** direction



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

Problem

- Premise: p reached by executing C on some p
- I.e. in the **backward** direction

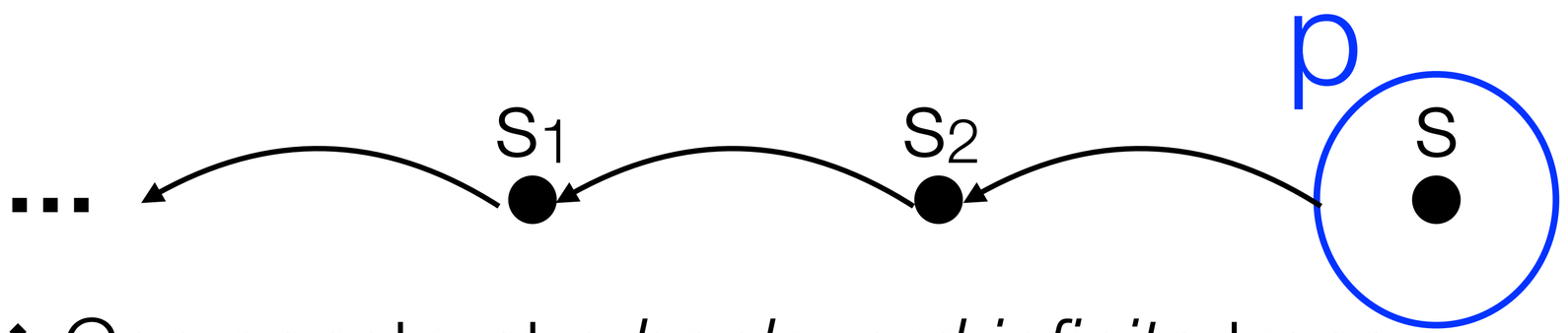


$$\frac{\vdash_B [p]\ C\ [ok:\ p]}{[p]\ C^*\ [\infty]}$$

- Can construct a *backward infinite* trace

Problem

- Premise: p reached by executing C on some p
- I.e. in the **backward** direction

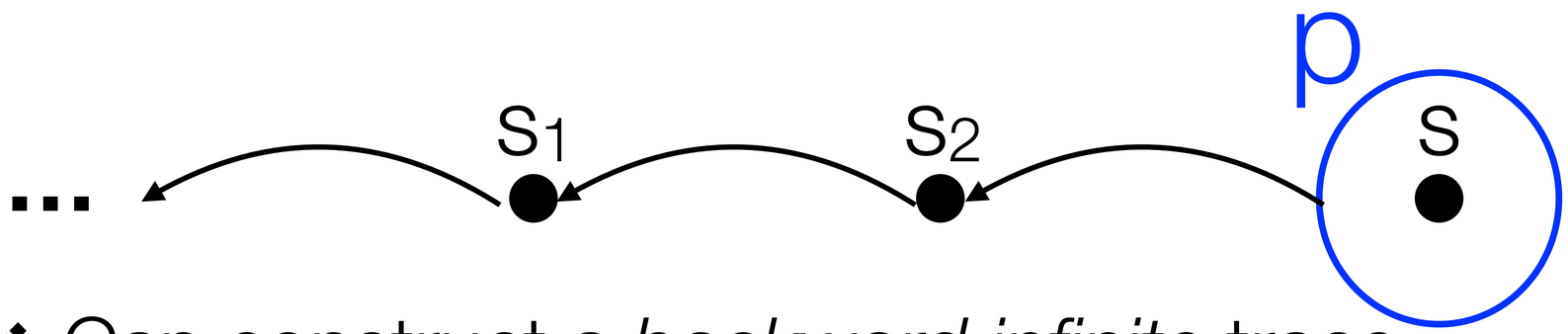


$$\frac{\vdash_B [p]\ C\ [ok: p]}{[p]\ C^*\ [\infty]}$$

- Can construct a *backward infinite* trace
- We need a *forward infinite* trace

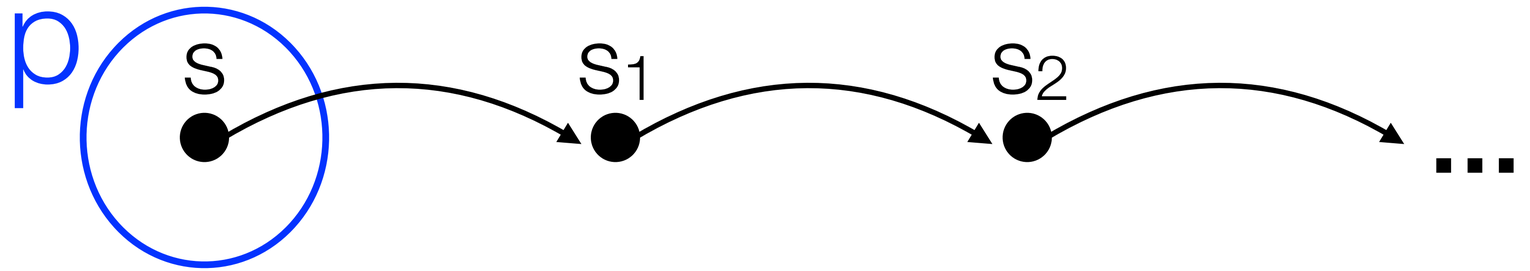
Problem

- Premise: p reached by executing C on some p
- I.e. in the **backward** direction



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

- Can construct a *backward infinite* trace
- We need a *forward infinite* trace



Problem

- Premise: p reached by executing C on some p
- I.e. in the **backward** direction

$$\frac{\vdash_{\mathcal{D}} [p]\ C\ [ok: p]}{\ldots}$$

- Can construct C^* [∞]
- We need a *forward infinite* trace

Solution

Forward *Under-Approximate Triples*

Forward Under-Approximate (FUA) Triples

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

Forward Under-Approximate (FUA) Triples

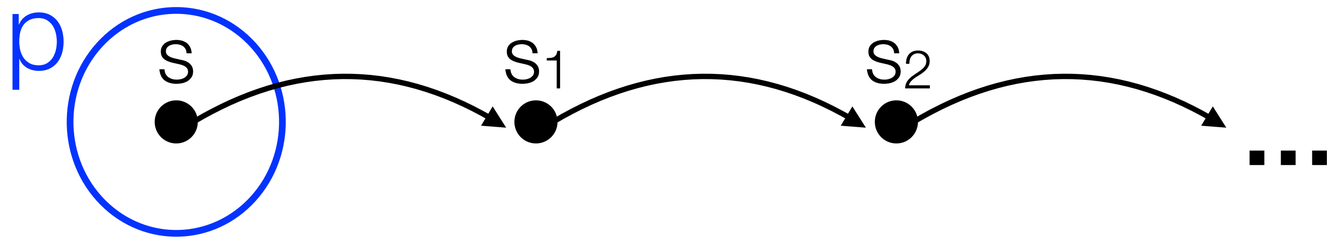
$$\vdash_F [p]\ C\ [\varepsilon: q] \quad \textit{iff} \quad \forall\, s \in p.\ \exists\, s' \in q.\ (s,s') \in [C]\varepsilon$$

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

Forward Under-Approximate (FUA) Triples

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

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



FUA is **Under-Approximate**!

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

FUA is **Under-Approximate**!

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

$$\frac{\vdash_F [p]\ C_1\ [er: q]}{\vdash_F [p]\ C_1; C_2\ [er: q]}$$

$$\frac{\vdash_F [p]\ C_1\ [ok: r] \qquad \vdash_F [r]\ C_2\ [\varepsilon: q]}{\vdash_F [p]\ C_1; C_2\ [\varepsilon: q]}$$

$$\frac{\vdash_F [p_1]\ C\ [\varepsilon: q_1] \qquad \vdash_F [p_2]\ C\ [\varepsilon: q_2]}{\vdash_F [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

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

$$\frac{}{\vdash_F [p]\ C^*\ [ok: p]}$$

$$\frac{\vdash_F [p]\ C^*; C\ [\varepsilon: q]}{\vdash_F [p]\ C^*\ [\varepsilon: q]}$$

FUA is **Under-Approximate**!

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

$$\frac{\vdash_{BF} [p]\ C_1\ [er: q]}{\vdash_{BF} [p]\ C_1; C_2\ [er: q]}$$

$$\frac{\vdash_{BF} [p]\ C_1\ [ok: r] \qquad \vdash_{BF} [r]\ C_2\ [\varepsilon: q]}{\vdash_{BF} [p]\ C_1; C_2\ [\varepsilon: q]}$$

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon: q_1] \qquad \vdash_{BF} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{BF} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

$$\frac{\vdash_{BF} [p]\ C_i\ [\varepsilon: q] \qquad \textbf{some}\ i \in \{1, 2\}}{\vdash_{BF} [p]\ C_1 + C_2\ [\varepsilon: q]}$$

$$\frac{}{\vdash_{BF} [p]\ C^*\ [ok: p]}$$

$$\frac{\vdash_{BF} [p]\ C^*; C\ [\varepsilon: q]}{\vdash_{BF} [p]\ C^*\ [\varepsilon: q]}$$

FUA is **Under-Approximate**!

Q: ***What is the difference between FUA and BUA reasoning?***

A: ***Rule of Consequence***

BUA vs. FUA

(ConsB)

$$\frac{p' \sqsubseteq p \quad \vdash_B [p']\ C\ [\varepsilon: q'] \quad q' \sqsupseteq q}{\vdash_B [p]\ C\ [\varepsilon: q]}$$

$\vdash_B [p]\ C\ [\varepsilon: q]$ *iff*

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

BUA vs. FUA

(ConsB)

$$\frac{p' \sqsubseteq p \quad \vdash_B [p'] \; C \; [\varepsilon: q'] \quad q' \sqsupseteq q}{\vdash_B [p] \; C \; [\varepsilon: q]}$$

$\vdash_B [p] \; C \; [\varepsilon: q]$ *iff*

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

(ConsF)

$$\frac{p' \sqsupseteq p \quad \vdash_F [p'] \; C \; [\varepsilon: q'] \quad q' \sqsubseteq q}{\vdash_F [p] \; C \; [\varepsilon: q]}$$

$\vdash_F [p] \; C \; [\varepsilon: q]$ *iff*

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

BUA vs. FUA

(ConsB)

$$\frac{p' \subseteq p \quad \vdash_B [p'] \; C \; [\varepsilon: q'] \quad q' \supseteq q}{\vdash_B [p] \; C \; [\varepsilon: q]}$$

(ConsF)

$$\frac{p' \supseteq p \quad \vdash_F [p'] \; C \; [\varepsilon: q'] \quad q' \subseteq q}{\vdash_F [p] \; C \; [\varepsilon: q]}$$

$\vdash_B [p] \; C \; [\varepsilon: q]$ *iff*

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

$\vdash_F [p] \; C \; [\varepsilon: q]$ *iff*

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

$$\frac{\vdash_B [p] \; C \; [\varepsilon: q_1 \vee q_2]}{\vdash_B [p] \; C \; [\varepsilon: q_1]}$$

Shrink the **post**

$$\frac{\vdash_F [p_1 \vee p_2] \; C \; [\varepsilon: q]}{\vdash_F [p_1] \; C \; [\varepsilon: q]}$$

Shrink the **pre**

BUA vs. FUA

Problem

Want to use existing **UA tools** (e.g. Pulse)

based on BUA

How to **practically reconcile BUA & FUA**?

When are Disj and ConsB used in BUA?

$$\frac{\vdash_{\mathrm{BF}} [p_1]\ C\ [\varepsilon\colon q_1] \qquad \vdash_{\mathrm{BF}} [p_2]\ C\ [\varepsilon\colon q_2]}{\vdash_{\mathrm{BF}} [p_1 \vee p_2]\ C\ [\varepsilon\colon q_1 \vee q_2]}$$

When are Disj and ConsB used in BUA?

$$\frac{\vdash_{\mathrm{BF}} [p_1]\ C\ [\varepsilon: q_1] \quad \vdash_{\mathrm{BF}} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{\mathrm{BF}} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

- **Disj on paper**: to combine multiple triples
- **ConsB on paper**: to weaken pre or strengthen post

When are Disj and ConsB used in BUA?

$$\frac{\vdash_{\mathrm{BF}} [p_1]\ C\ [\varepsilon: q_1] \qquad \vdash_{\mathrm{BF}} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{\mathrm{BF}} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

- **Disj on paper**: to combine multiple triples
- **ConsB on paper**: to weaken pre or strengthen post
- **Disj in Pulse**: rarely used; pre-post correspondence tracked (distinct summaries)

When are Disj and ConsB used in BUA?

$$\frac{\vdash_{\mathrm{BF}} [p_1]\ C\ [\varepsilon: q_1] \quad \vdash_{\mathrm{BF}} [p_2]\ C\ [\varepsilon: q_2]}{\vdash_{\mathrm{BF}} [p_1 \vee p_2]\ C\ [\varepsilon: q_1 \vee q_2]}$$

$$\frac{\vdash_{\mathrm{B}} [p]\ C\ [\varepsilon: q_1 \vee q_2]}{\vdash_{\mathrm{B}} [p]\ C\ [\varepsilon: q_1]}$$

- **Disj on paper**: to combine multiple triples
- **ConsB on paper**: to weaken pre or strengthen post
- **Disj in Pulse**: rarely used; pre-post correspondence tracked (distinct summaries)
- **ConsB in Pulse**: mainly to drop disjuncts (i.e. forget summaries)

Indexed Disjuncts

$P, Q \in \mathbb{N} \to \mathscr{P}(\text{States})$

Indexed Disjuncts

$P, Q \in \mathbb{N} \to \mathscr{P}(\text{States})$

$$Q \equiv \bigvee_{i \in \text{dom}(Q)} q_i$$

Indexed Disjuncts

$$P, Q \in \mathbb{N} \to \mathscr{P}(\text{States}) \qquad Q \equiv \bigvee_{i \in \text{dom}(Q)} q_i$$

$$\vdash_\dagger [P]\ C\ [\varepsilon\colon Q] \quad \textit{iff} \quad \text{dom}(P) = \text{dom}(Q) \wedge$$

$$\forall\, i \in \text{dom}(P).\ \vdash_\dagger [P(i)]\ C\ [\varepsilon\colon Q(i)]$$

Unified BUA/FUA Framework

$$\frac{\vdash_{\mathrm{BF}} [p_1]\ C\ [\varepsilon\colon q_1] \qquad \vdash_{\mathrm{BF}} [p_2]\ C\ [\varepsilon\colon q_2]}{\vdash_{\mathrm{BF}} [p_1 \vee p_2]\ C\ [\varepsilon\colon q_1 \vee q_2]} \quad \rightsquigarrow \quad \frac{\vdash_{\mathrm{BF}} [P_1]\ C\ [\varepsilon\colon Q_1] \qquad \vdash_{\mathrm{BF}} [P_2]\ C\ [\varepsilon\colon Q_2]}{\vdash_{\mathrm{BF}} [P_1 \uplus P_2]\ C\ [\varepsilon\colon Q_1 \uplus Q_2]}$$

Unified BUA/FUA Framework

$$\frac{\vdash_{BF} [p_1]\ C\ [\varepsilon\colon q_1] \qquad \vdash_{BF} [p_2]\ C\ [\varepsilon\colon q_2]}{\vdash_{BF} [p_1 \vee p_2]\ C\ [\varepsilon\colon q_1 \vee q_2]} \rightsquigarrow \frac{\vdash_{BF} [P_1]\ C\ [\varepsilon\colon Q_1] \qquad \vdash_{BF} [P_2]\ C\ [\varepsilon\colon Q_2]}{\vdash_{BF} [P_1 \uplus P_2]\ C\ [\varepsilon\colon Q_1 \uplus Q_2]}$$

(ConsB)

$$\frac{p' \subseteq p \qquad \vdash_{B} [p']\ C\ [\varepsilon\colon q'] \qquad q' \supseteq q}{\vdash_{B} [p]\ C\ [\varepsilon\colon q]}$$

(ConsF)

$$\frac{p' \supseteq p \qquad \vdash_{F} [p']\ C\ [\varepsilon\colon q'] \qquad q' \subseteq q}{\vdash_{F} [p]\ C\ [\varepsilon\colon q]}$$

$\rightsquigarrow$

$$\frac{\vdash_{BF} [P]\ C\ [\varepsilon\colon Q] \qquad I \subseteq \mathrm{dom}(P)}{\vdash_{BF} [P \downarrow I]\ C\ [\varepsilon\colon Q \downarrow I]}$$

Unified BUA/FUA Framework

Can use Pulse **as is**!

☞ **Extend** Pulse w. **divergence rules**

Relating BUA and FUA

Theorem 1.

$\vdash_B [p]\ C\ [\varepsilon: q] \wedge \mathrm{minpre}(p, C, q) \Rightarrow \vdash_F [p]\ C\ [\varepsilon: q]$

Relating BUA and FUA

Theorem 1.

$\vdash_B [p]\ C\ [\varepsilon: q] \wedge \mathrm{minpre}(p, C, q) \Rightarrow \vdash_F [p]\ C\ [\varepsilon: q]$

where $\mathrm{minpre}(p, C, q)$ ***iff*** $\forall p'.\ \vdash_B [p']\ C\ [\varepsilon: q] \Rightarrow p' \not\subset p$

Relating BUA and FUA

Theorem 1.

$\vdash_B [p]\ C\ [\varepsilon: q] \wedge \mathrm{minpre}(p, C, q) \Rightarrow \vdash_F [p]\ C\ [\varepsilon: q]$

where $\mathrm{minpre}(p, C, q)$ ***iff*** $\forall p'.\ \vdash_B [p']\ C\ [\varepsilon: q] \Rightarrow p' \not\subset p$

Theorem 2.

$\vdash_F [p]\ C\ [\varepsilon: q] \wedge \mathrm{minpost}(p, C, q) \Rightarrow \vdash_B [p]\ C\ [\varepsilon: q]$

where $\mathrm{minpost}(p, C, q)$ ***iff*** $\forall q'.\ \vdash_F [p]\ C\ [\varepsilon: q'] \Rightarrow q' \not\subset q$

Pulse$^{\infty}$: Prototype based on UNTer

- An extension of Pulse with **divergence rules**
- **First automated** non-termination tool on *large codebases/libraries* (e.g. openSSL)

Pulse$^{\infty}$: Prototype based on UNTer

- An extension of Pulse with **divergence rules**
- **First automated** non-termination tool on *large codebases/libraries* (e.g. openSSL)

Tool	Term.	Non-term.	Non-det.	Heap	Auto.	UA/OA	Large Code/Libraries
Terminator [14, 15]	✓	✗	✓	✗	✓	OA	✗
Mutant [4]	✓	✗	✗	✓	✓	OA	✗
TNT [24]	✗	✓	✗	✗	✓	OA	✗
KEY [44]	✗	✓	✗	✗	✓	OA	✗
CPROVER [30]	✓	✓	✓	✗	✓	UA-OA	✗
TRex [26]	✓	✓	✓	✗	✓	UA-OA	✗
T2 [17]	✓	✗	✓	✗	✓	OA	✗
Coop-T2 [6]	✓	✓	✓	✗	✓	UA-OA	✗
CABER [7]	✓	✗	✓	✓	✓	OA	✗
CPP-INV [31]	✗	✓	✓	✗	✓	OA	✗
HipTNT [34]	✓	✓	✓	Partial	✗	OA	✗
HipTNT+ [35]	✓	✓	✓	Partial	✓	OA	✗
DynamiTe [33]	✓	✓	✓	Partial	✓	OA	✗
RevTerm [10]	✗	✓	✓	✗	✓	OA	✗
AProVE [28]	✗	✓	✓	✗	✓	OA	✗
ULTIMATE [27]	✓	✗	✓	✗	✓	OA	✗
Pulse$^{\infty}$	✗	✓	✓	✓	✓	UA	✓

Pulse$^{\infty}$: Prototype based on UNTer

- An extension of Pulse with **divergence rules**
- **First automated** non-termination tool on *large codebases/libraries* (e.g. openSSL)

Tool	Term.	Non-term.	Non-det.	Heap	Auto.	UA/OA	Large Code/Libraries
Terminator [14, 15]	✓	✗	✓	✗	✓	OA	✗
Mutant [4]	✓	✗	✗	✓	✓	OA	✗
TNT [24]	✗	✓	✗	✗	✓	OA	✗
KEY [44]	✗	✓	✗	✗	✓	OA	✗
CPROVER [30]	✓	✓	✓	✗	✓	UA-OA	✗
TRex [26]	✓	✓	✓	✗	✓	UA-OA	✗
T2 [17]	✓	✗	✓	✗	✓	OA	✗
Coop-T2 [6]	✓	✓	✓	✗	✓	UA-OA	✗
CABER [7]	✓	✗	✓	✓	✓	OA	✗
CPP-INV [31]	✗	✓	✓	✗	✓	OA	✗
HipTNT [34]	✓	✓	✓	Partial	✗	OA	✗
HipTNT+ [35]	✓	✓	✓	Partial	✓	OA	✗
DynamiTe [33]	✓	✓	✓	Partial	✓	OA	✗
RevTerm [10]	✗	✓	✓	✗	✓	OA	✗
AProVE [28]	✗	✓	✓	✗	✓	OA	✗
ULTIMATE [27]	✓	✗	✓	✗	✓	OA	✗
Pulse$^{\infty}$	✗	✓	✓	✓	✓	UA	✓

Pulse$^{\infty}$: Prototype based on UNTer

- An extension of Pulse with **divergence rules**
- **First automated** non-termination tool on *large codebases/libraries* (e.g. openSSL)
- Evaluation (small examples): SV-COMP benchmarks — at most 3 loops, mostly 1 loop
  - ➡ State of the art: **DynamiTe**
  - ➡ Performance (**time**): competitive — **few seconds** for both both Pulse$^{\infty}$ & DynamiTe

Pulse$^{\infty}$: Prototype based on UNTer

- An extension of Pulse with **divergence rules**
- **First automated** non-termination tool on *large codebases/libraries* (e.g. openSSL)
- Evaluation (small examples): SV-COMP benchmarks — at most 3 loops, mostly 1 loop
  - ➡ State of the art: **DynamiTe**
  - ➡ Performance (**time**): competitive — **few seconds** for both both Pulse$^{\infty}$ & DynamiTe
  - ➡ Performance (**detection**): competitive or better:

Both have false negatives

DynamiTe several false positives; Pulse$^{\infty}$ has none

Pulse$^{\infty}$: Prototype based on UNTer

- An extension of Pulse with **divergence rules**
- **First automated** non-termination tool on *large codebases/libraries* (e.g. openSSL)
- Evaluation (small examples): SV-COMP benchmarks — at most 3 loops, mostly 1 loop
  - ➡ Performance (**time**): competitive — **few seconds** for both both Pulse$^{\infty}$ & DynamiTe
  - ➡ Performance (**detection**): DynamiTe several false positives; Pulse$^{\infty}$ has none
- Evaluation (large examples)

Library	Language	#LOC analysed	Time	# Bugs found
OpenSSL	C	804 K	5m, 30s	5
libxml2	C	300 K	4m, 6s	4
CryptoPP	C++	51 K	3m, 32s	1
libxpm	C	11 K	13s	1
libpng	C	96 K	38s	0
zlib	C	41 K	1m, 6s	0
ngiflib	C	1.7 K	5s	0
Total		**1.3 M**	**14m, 5s**	**11**

The soundness of bugs is what matters!

The goal is to find bugs!

"Most program analysis & verification research seems confused about the ultimate goal of software defect detection. The main practical usefulness of such techniques is the ability to find bugs, not to report that no bugs have been found."

Patrice Godefroid, 2005

Thank You for Listening!

azalea@imperial.ac.uk

SoundAndComplete.org

VERITAS