Incorrectness Logic for Scalable Bug Detection

Azalea Raad
Imperial College London

Dependable and Secure Software Systems, ETH Zürich
October 2022
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 \( \{p\} C \{q\} \)
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 $\text{post}(C)p \subseteq q$

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 \text{post}(C)p \subseteq q

$q$ over-approximates $\text{post}(C)p$
Hoare Logic (HL)

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

\( q \) over-approximates \( \text{post}(C)p \)
Hoare Logic (HL)

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

\( q \) over-approximates \( \text{post}(C)p \)

true positive

\( \text{post}(C)p \)
Hoare Logic (HL)

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

\( q \) over-approximates \( \text{post}(C)p \)

- True positive
- False positive
“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 \( \text{post}(C)p \subseteq q \)

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 \( \text{post}(C)p \subseteq q \)

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

\[ \text{post}(C)p \subseteq q \]
Incorrectness Logic (IL)

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

For all states \( s \) in \( p \)
If running \( C \) on \( s \) terminates in \( s' \), then \( s' \) is in \( q \)

\( \text{post}(C)p \supseteq q \)
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 \( \{p\} \ C \ \{q\} \iff \text{post}(C)p \subseteq q \)

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 \text{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

true positive

false positive
Incorrectness Logic (IL)

Hoare triples
\[
\{p\} \ C \ \{q\} \quad \text{iff} \quad \text{post}(C)p \subseteq q
\]

iff

\[q \ \text{over-approximates} \ \text{post}(C)p\]

\[
\begin{array}{cc}
\text{true positive} & \text{true positive} \\
\text{false positive} & \text{false positive}
\end{array}
\]

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

iff

\[q \ \text{under-approximates} \ \text{post}(C)p\]

\[
\begin{array}{cc}
\text{true positive} & \text{true positive} \\
\text{false positive} & \text{false positive}
\end{array}
\]
Incorrectness Logic (IL)

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

q over-approximates post(C)p

\[ \text{true positive} \quad \text{false positive} \]

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

q under-approximates post(C)p

\[ \text{true positive} \quad \text{false negative} \]
Incorrectness Logic (IL)

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

\( \varepsilon \): exit condition
- ok: normal execution
- er: erroneous execution
Incorrectness Logic (IL)

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

\( \varepsilon \): exit condition

ok: normal execution
er : erroneous execution

\[ y=v \] \( x:=y \) [ok: \( x=y=v \)]
Incorrectness Logic (IL)

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

\( \varepsilon: \) exit condition
- ok: normal execution
- er: erroneous execution

\[ [y=v] x:=y \ [ok: x=y=v] \ [p] \text{error()} \ [er: p] \]
Incorrectness Logic (IL)

Equivalent Definition (reachability)

[p] C [ε: q]  iff  ∀ s ∈ q. ∃ s’ ∈ p. (s’,s) ∈ [C]ε
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 reasoning is non-compositional (as in original Hoare Logic)
  - Cannot target memory safety bugs (e.g. use-after-free)

Our Solution

Incorrectness Separation Logic
What Is Separation Logic (SL)?

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

≡ ideal for heap-manipulating programs with **aliasing**
What Is Separation Logic (SL)?

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

❯ ideal for heap-manipulating programs with *aliasing*

```plaintext
[x] := 1;
[y] := 2;
[z] := 3;
```
What Is Separation Logic (SL)?

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

идеально для алгоритмов работы с *aliasing*

\[
\begin{align*}
[x] &:= 1; \\
[y] &:= 2; \\
[z] &:= 3;
\end{align*}
\]

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*

\[
\text{pre: } \{ x \neq y \wedge x \neq z \wedge y \neq z \} \\
[x] := 1; \\
[y] := 2; \\
[z] := 3; \\
\text{post: } \{ x = 1 \wedge y = 2 \wedge z = 3 \}
\]
What Is Separation Logic (SL)?

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

 النبي ideal for heap-manipulating programs with *aliasing*

\[
\text{pre: } \{ x_1 \neq x_2 \land x_1 \neq x_3 \land \ldots \} \\
[x_1] := 1; \\
[x_2] := 2; \\
\ldots \\
[x_n] := n;
\]

\[
\text{post: } \{ x_1 = 1 \land \ldots \land x_n = n \}
\]
What Is Separation Logic (SL)?

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

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

```latex
\begin{align*}
\text{pre: } \{ x_1 &\neq x_2 \land x_1 \neq x_3 \land \ldots \} \\
\quad [x_1] &: = 1; \\
\quad [x_2] &: = 2; \\
\quad \ldots \\
\quad [x_n] &: = n; \\
\text{post: } \{ x_1 = 1 \land \ldots \land x_n = n \}
\end{align*}
```

\( n!/2 \) conjuncts!
What Is Separation Logic (SL)?

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

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

```plaintext
pre: \{ x \mapsto - * y \mapsto - * z \mapsto - \}

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

post: \{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}
```
What Is Separation Logic (SL)?

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

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

pre: \(\{ x \mapsto - \, \ast \, y \mapsto - \, \ast \, z \mapsto - \}\)  
ownership of heap cell at x  
\([x] := 1;\)  
\([y] := 2;\)  
\([z] := 3;\)

post: \(\{ x \mapsto 1 \, \ast \, y \mapsto 2 \, \ast \, z \mapsto 3 \}\)
What Is Separation Logic (SL)?

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

่า 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:** \( \{ x \mapsto 1 \ast y \mapsto 2 \ast z \mapsto 3 \} \)
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;
```

Pre: \[ \{ x \mapsto - \star y \mapsto - \star z \mapsto - \} \]

- *ownership* of heap cell at \( x \)

Post: \[ \{ x \mapsto 1 \star y \mapsto 2 \star z \mapsto 3 \} \]

\[ \forall x, v, v'. \ x \mapsto v \star x \mapsto v' \Rightarrow false \]
The Essence of Separation Logic (SL)

**Frame Rule**

\[
\frac{\{p\} \ C \ \{q\}}{\{p \ast r\} \ C \ \{q \ast r\}}
\]

\[x \mapsto v \ast x \mapsto v' \iff \text{false} \quad \text{p} \ast \text{emp} \iff \text{p}\]
The Essence of Separation Logic (SL)

**Frame Rule**

$$
\begin{align*}
\{p\} \, C \, \{q\} \\
\{ p \ast r \} \, C \, \{ q \ast r \}
\end{align*}
$$

**Local Axioms**

\( x \mapsto v \ast x \mapsto v' \iff \text{false} \quad \text{p} \ast \text{emp} \iff \text{p} \)

\( \text{WRITE} \quad \{ x \mapsto - \} \, [x] := v \, \{ x \mapsto v \} \)
The Essence of Separation Logic (SL)

**Frame Rule**

\[
\begin{array}{c}
\{p\} C \{q\} \\
\{p \ast r\} C \{q \ast r\}
\end{array}
\]

\[
x \mapsto v \ast x \mapsto v' \iff false \quad p \ast emp \iff p
\]

**Local Axioms**

**WRITE**

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

**READ**

\[
\{x \mapsto v\} y := [x] \{x \mapsto v \land y = v\}
\]
The Essence of Separation Logic (SL)

**Frame Rule**

\[
\frac{\{p\} \mathcal{C} \{q\}}{\{p \ast r\} \mathcal{C} \{q \ast r\}}
\]

\[
x \mapsto v \ast x \mapsto v' \iff \text{false} 
\]

\[
p \ast \text{emp} \iff p
\]

**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**

\[
\frac{\{p\} \ C \ \{q\}}{\{p \ast r\} \ C \ \{q \ast r\}}
\]

\[
x \mapsto v \ast x \mapsto v' \iff \text{false} \quad p \ast \text{emp} \iff p
\]

**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

$$\{p\} \ C \ \{q\}$$

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

$$x \mapsto - \ast \ x \mapsto - \iff \text{false}$$

$$x \mapsto v \ast \ \text{emp} \iff x \mapsto v$$
Incorrectness Separation Logic (ISL)

IL

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

SL

\[
\begin{array}{c}
\{p\} \ C \ \{q\} \\
\{p \ast r\} \ C \ \{q \ast r\}
\end{array}
\]

\[
\begin{align*}
x & \mapsto - \ast x & \mapsto - & \iff \text{false} \\
x & \mapsto v \ast \text{emp} & \iff x \mapsto v
\end{align*}
\]

ISL

\[
\begin{array}{c}
\begin{align*}
[p] \ C [\varepsilon : q] \\
[p \ast r] \ C [\varepsilon : q \ast r]
\end{align*}
\end{array}
\]

\[
\begin{align*}
x & \mapsto v \ast x & \mapsto v' & \iff \text{false} \\
x & \mapsto v \ast \text{emp} & \iff x \mapsto v
\end{align*}
\]
ISL: Local Axioms (First Attempt)

WRITE

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

\[ x \mapsto v' \] \[ x := v \] \[ \text{ok: } x \mapsto v \] \[ x=null \] \[ x := v \] \[ \text{er: } x=null \]
ISL: Local Axioms (First Attempt)

WRITE

\[ x \mapsto v' \] \[ x \mapsto v \] \[ \text{ok: } x \mapsto v \]  

\[ x = \text{null} \] \[ x \mapsto v \] \[ \text{er: } x = \text{null} \]  

null-pointer dereference error
ISL: Local Axioms (First Attempt)

**WRITE**

\[
[x \mapsto v'] \quad \text{[} x := v \text{]} \quad \text{[ok: } x \mapsto v\text{]} \quad \text{[x=null] } \quad \text{[} x := v \text{]} \quad \text{[er: } x = \text{null}\text{]}
\]

*null-pointer dereference error*

**READ**

\[
[x \mapsto v] \quad y := [x] \quad \text{[ok: } x \mapsto v \land y = v\text{]} \quad \text{[x=null] } \quad y := [x] \quad \text{[er: } x = \text{null}\text{]}
\]
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**

\[ [\text{emp}] \ x := \text{alloc()} \ [ok: \exists l. \ l \mapsto v \land x=l] \]
ISL: Local Axioms (First Attempt)

**WRITE**

\[ x \mapsto v' \] \[ x := v \] [ok: \( x \mapsto v \)]

\[ x = \text{null} \] \[ x := v \] [er: \( x = \text{null} \)]

*null-pointer dereference error*

**READ**

\[ x \mapsto v \] \[ y := [x] \] [ok: \( x \mapsto v \land y = v \)]

\[ x = \text{null} \] \[ y := [x] \] [er: \( x = \text{null} \)]

**ALLOC**

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

**FREE**

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

\[ x = \text{null} \] \( \text{free}(x) \) [er: \( x = \text{null} \)]
ISL: Local Axioms (First Attempt)

**WRITE**

\[ x \mapsto v' \] \[ x \leftarrow v \] [**ok**: \( x \mapsto v \)]

\[ x = \text{null} \] \[ x \leftarrow v \] [**er**: \( x = \text{null} \)]

*null-pointer dereference error*

**READ**

\[ x \mapsto v \] \[ y \leftarrow [x] \] [**ok**: \( x \mapsto v \land y = v \)]

\[ x = \text{null} \] \[ y \leftarrow [x] \] [**er**: \( x = \text{null} \)]

**ALLOC**

\[ \text{emp} \] \( x \leftarrow \text{alloc()} \) [**ok**: \( \exists l. \ l \mapsto v \land x = l \)]

**FREE**

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

\[ \text{null} \] \( \text{free}(x) \) [**er**: \( x = \text{null} \)]

**KABOOM!**
ISL: Local Axioms (First Attempt)

\[
\text{ISL} \quad \frac{[p] \ C \ [\varepsilon: q]}{[p \ast r] \ C \ [\varepsilon: q \ast r]} \quad \begin{align*}
& x \mapsto v \ast x \mapsto v' \iff \text{false} \\
& \text{emp} \ast p \iff p
\end{align*}
\]

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

\[
[p] \ C \ [\varepsilon: q] \ \iff \ \forall s \in q. \ \exists s' \in p. \ (s',s) \in [C] \varepsilon
\]
ISL: Local Axioms (First Attempt)

\[
\begin{align*}
[p] & \quad C \quad [\varepsilon : q] \\
[p \star r] & \quad C \quad [\varepsilon : q \star r]
\end{align*}
\]

\[x \mapsto v \quad \star \quad x \mapsto v' \iff \text{false}\]
\[\text{emp} \quad \star \quad p \iff p\]

\[
\begin{align*}
[x \mapsto v] & \quad \text{free}(x) \quad [\text{ok: emp}] \\
[x \mapsto v \star x \mapsto v] & \quad \text{free}(x) \quad [\text{ok: emp} \star x \mapsto v]
\end{align*}
\]

\[
[p] \quad C \quad [\varepsilon : q] \quad \iff \quad \forall s \in q. \exists s' \in p. (s', s) \in [C]s
\]
ISL: Local Axioms (First Attempt)

\[
\begin{align*}
\text{ISL} & \quad [p] \ C \ [\varepsilon : q] \\
\quad & \quad [p \star r] \ C \ [\varepsilon : q \star r]
\end{align*}
\]

\[
\begin{align*}
[x \mapsto v] \ free(x) & \quad [ok: emp] \\
[x \mapsto v \star x \mapsto v] \ free(x) & \quad [ok: emp \star x \mapsto v]
\end{align*}
\]

\[
\begin{align*}
\text{(Frame)} & \quad [false] \ free(x) \quad [ok: x \mapsto v] \\
\text{(Cons)}
\end{align*}
\]

\[
[p] \ C \ [\varepsilon : q] \quad iff \quad \forall s \in q. \ \exists s' \in p. \ (s',s) \in [C]_{\varepsilon}
\]
**ISL: Local Axioms (First Attempt)**

\[
\begin{align*}
\text{ISL} & \quad [p] \ C \ [\varepsilon : q] \\
\hline
[\varepsilon : q] & \quad x \mapsto v \iff x \mapsto v' \iff \text{false} \\
\text{emp} & \quad \text{p} \iff \text{p}
\end{align*}
\]

\[
\begin{align*}
[x \mapsto v] \ & \text{free}(x) \ [\text{ok: emp}] \\
[x \mapsto v \ast x \mapsto v] & \text{free}(x) \ [\text{ok: emp} \ast x \mapsto y] \\
\text{false} & \text{free}(x) \ [\text{ok: x} \mapsto v]
\end{align*}
\]

\[
\begin{align*}
[p] \ C \ [\varepsilon : q] \ & \iff \forall s \in q. \exists s' \in p. (s',s) \in [C]\varepsilon \\
\text{false} \ C \ [\varepsilon : q] \ & \times \ (\text{unless } q \Rightarrow \text{false})
\end{align*}
\]
ISL: Local Axioms (First Attempt)

\[ [p] \mathcal{C} [\varepsilon : q] \]

\[ [p \ast r] \mathcal{C} [\varepsilon : q \ast r] \]

\[ x \mapsto v \ast x \mapsto v' \Rightarrow \text{false} \]

\[ \text{emp} \ast p \Rightarrow p \]

---

**Solution:**

Track Deallocated Locations!

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

\[ [\text{false}] \mathcal{C} [\varepsilon : q] \times \] (unless \( q \Rightarrow \text{false} \))

16
Solution: Track Deallocated Locations!

\[ [x \mapsto v] \text{ free}(x) [\text{ok: } \text{emp}] \]
Solution: Track Deallocated Locations!

\[ [x \leftrightarrow v] \, \text{free}(x) \, [\text{ok: } x \leftrightarrow ] \]
Solution: Track Deallocated Locations!

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

x is *deallocated*
Solution: Track Deallocated Locations!

\[ x \leftrightarrow v \] free(x) [ok: \( x \mapsto \)]

\( x \) is *deallocated*

\( x \leftrightarrow v \, \ast \, x \leftrightarrow v' \iff \text{false} \\
p \, \ast \, \text{emp} \iff p \)
Solution: Track Deallocated Locations!

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

\( x \) is \emph{deallocated}

\[
\begin{align*}
x \mapsto v \quad & \quad x \mapsto v' \iff \text{false} \\
p \quad & \quad \text{emp} \iff p \\
x \mapsto v \quad & \quad x \mapsto \iff \text{false} \\
x \mapsto \quad & \quad x \mapsto \iff \text{false} \\
x \mapsto \quad & \quad x \mapsto \iff \text{false}
\end{align*}
\]
Solution: Track Deallocation Locations!

\[ [x \mapsto v] \text{ free}(x) \quad \text{[ok: } x \mapsto \text{]} \]
Solution: Track Deallocated Locations!

\[
[x \mapsto v] \text{ free}(x) [\text{ok: } x \mapsto ] \\
[ x \mapsto v \ast x \mapsto v ] \text{ free}(x) [\text{ok: } x \mapsto \ast x \mapsto v]
\]
Solution: Track Deallocated Locations!

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

\[ [x \mapsto v \ast x \mapsto v] \text{ free}(x) \quad \text{[ok: } x \mapsto \ast x \mapsto v \text{]} \]

\[ \text{[false] free}(x) \quad \text{[ok: false]} \]

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

\[ [p] C \quad \varepsilon: \text{false} \quad \text{(vacuous)} \]
ISL: Local Axioms

\[ [x \mapsto v] \text{ free}(x) \quad \text{[ok: x \mapsto]} \quad [x=null] \text{ free}(x) \quad \text{[er: x=null]} \]

FREE
ISL: Local Axioms

\[
[x \mapsto v] \text{ free}(x) [\text{ok: } x \mapsto ] \quad [x=\text{null}] \text{ free}(x) [\text{er: } x=\text{null}]
\]

\[
\text{FREE} \quad [x \mapsto ] \text{ free}(x) [\text{er: } x \mapsto ]
\]
ISL: Local Axioms

\[
[x \mapsto v] \text{ free}(x) \quad \text{[ok: } x \mapsto \text{]} \quad [x=\text{null}] \text{ free}(x) \quad \text{[er: } x=\text{null}] \\
\text{FREE} \quad \text{[x } \mapsto \text{] free}(x) \quad \text{[er: } x \mapsto \text{]} \\
\text{use-after-free error}
\]
ISL: Local Axioms

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

\[ \text{FREE} \]

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

\[ x \mapsto v \] free(x) \[\text{[er: } x \mapsto \text{]}\]

\[ x = \text{null} \] \[\text{[x := v]}\]

\[ \text{WRITE} \]

\[ x = \text{null} \] \[\text{[x := v]}\]

\[\text{use-after-free error}\]
### ISL: Local Axioms

<table>
<thead>
<tr>
<th>Axion</th>
<th>Description</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FREE</strong></td>
<td>$[x \mapsto v] \text{free}(x)$ [ok: $x \mapsto v$]</td>
<td>$[x=\text{null}] \text{free}(x)$ [er: $x=\text{null}$]</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>WRITE</td>
<td>$[x \mapsto v'] [x] := v$ [ok: $x \mapsto v$]</td>
<td>$[x=\text{null}] [x] := v$ [er: $x=\text{null}$]</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>READ</td>
<td>$[x \mapsto v] y := [x]$ [ok: $x \mapsto v \land y=v$]</td>
<td>$[x=\text{null}] y := [x]$ [er: $x=\text{null}$]</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$</td>
<td>$\Rightarrow$</td>
</tr>
</tbody>
</table>

**use-after-free error**
ISL: Local Axioms

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

\[ \text{FREE} \]

\[ [x \mapsto v'] [x] := v \quad [\text{ok: } x \mapsto v] \]

\[ \text{WRITE} \]

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

\[ \text{READ} \]

\[ [\text{emp}] x := \text{alloc()} \quad [\text{ok: } \exists l. l \mapsto v \land x = l] \]

\[ \text{ALLOC} \]

\[ [x = \text{null}] \text{ free}(x) \quad [\text{er: } x = \text{null}] \]

\[ [x \mapsto v] \text{ free}(x) \quad [\text{er: } x \mapsto v] \]

\[ \text{use-after-free error} \]
### ISL: Local Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>free(x)</code></td>
<td>[ok: <code>x ↦ v</code>], <code>x ↦ v</code></td>
<td><code>free(x)</code> [ok: <code>x ↦ v</code>], <code>x ↦ v</code></td>
</tr>
<tr>
<td><code>x = null</code></td>
<td>[er: <code>x = null</code>], <code>x ↦ v</code></td>
<td><code>x = null</code> [er: <code>x = null</code>], <code>x ↦ v</code></td>
</tr>
</tbody>
</table>

*use-after-free error*

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x ↦ v'</code></td>
<td>[ok: <code>x ↦ v</code>], <code>y := [x]</code></td>
<td><code>x := v'</code> [ok: <code>x ↦ v</code>], <code>y := [x]</code></td>
</tr>
<tr>
<td><code>x = null</code></td>
<td>[er: <code>x = null</code>], <code>y := [x]</code></td>
<td><code>x = null</code> [er: <code>x = null</code>], <code>y := [x]</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>y := [x]</code></td>
<td>[ok: <code>x ↦ v</code>, <code>y = v</code>], <code>x := alloc()</code></td>
<td><code>y := [x]</code> [ok: <code>x ↦ v</code>, <code>y = v</code>], <code>x := alloc()</code></td>
</tr>
<tr>
<td><code>x := alloc()</code></td>
<td>[ok: <code>∃l. l ↦ v ∧ x = l</code>], <code>y ↦ v</code></td>
<td><code>x := alloc()</code> [ok: <code>∃l. l ↦ v ∧ x = l</code>], <code>y ↦ v</code></td>
</tr>
</tbody>
</table>

### Allocation

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

### Write Access

- `x := v` [ok: `x ↦ v`], `x := v` |
- `x := v` [er: `x = null`], `x := v` |

### Read Access

- `y := [x]` [ok: `x ↦ v`, `y = v`], `x := alloc()` |
- `y := [x]` [er: `x = null`], `y := [x]` |

### Use-After-Free Error

- `x := v` [ok: `x ↦ v`], `x := v` |
- `x := v` [er: `x = null`], `x := v` |
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

  ![Diagram of procedures](image)

  - **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));`
   
   `...`

}
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)
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

1. `int ssl_excert_prepend(...){`

2. `SSL_EXCERT *exc= \textbf{app\_malloc}(\texttt{sizeof(*exc)}, "prepend cert");`

3. `memset(exc, 0, \texttt{sizeof(*exc)});

   ...`

   \textbf{null pointer dereference}

   \textbf{calls CRYPTO\_malloc (a malloc wrapper)}

   \textbf{CRYPTO\_malloc may return null!}

[emp] `*exc= \textbf{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));`

... null pointer dereference

calls CRYPTO_malloc (a malloc wrapper)

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));

... null pointer dereference

} calls CRYPTO_malloc (a malloc wrapper)

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

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

    if (!exc) {
        // Handle null pointer dereference
    }
```

**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

```c
... ...
static int ssl_excert_prepend(SSL_EXCERT **pexc)
{
    SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
    ...
    if (!exc) {
        ...
    }
```

**Paulidale** 13 days ago  Contributor

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

**lequangloc** 13 days ago  Author

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

**Paulidale** 13 days ago  Contributor

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

We should fix the one in `test/testutil/apps_mem.c`. 
Pulse-X: Null Pointer Dereference in OpenSSL

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…
1. `void foo(int *x) {
2.     *x = 42;
3. }

Pulse-X: Bug Reporting

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

WRITE  [x=null] *x = v [er: x=null]
1. void foo(int *x) {
2.   *x = 42;
}

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

[x=null] foo(x) [er: x=null]
1. `void foo(int *x)` {
2.   `*x = 42;`
}

**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

Developer

“But I never call foo with null!”
```c
1. void foo(int *x) {
2.   *x = 42;
}
```

**Should we report this NPD?**

- **yes**
- **no**

**Developer**

“But I never call foo with null!”

**Pulse-X**

“Which bugs shall I report then?”

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;
}
```

**Problem**

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

"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**

---

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

"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:** $[p] \mathcal{C} [\text{er: } q]$ is manifest iff:

\[
\forall s. \exists s'. \ (s,s') \in [\mathcal{C}]_{\text{er}} \land s' \in (q \ast \text{true})
\]
Pulse-X: **Manifest** Errors

- **Intuitively:** the error occurs for **all input states**
- **Formally:** \([p] C [er: q]\) is manifest iff:
  \[
  \forall s. \exists s'. (s,s') \in [C]_{er} \land s' \in (q * true)
  \]
- **Algorithmically:** \([p] C [er: q]\) is manifest if when: \(q = \exists X. h_q \land \pi_q :\)
Pulse-X: Manifest Errors

- **Intuitively**: the error occurs for all input states
- **Formally**: $[p] \ C [er: q]$ is manifest iff:
  \[
  \forall s. \ \exists s'. \ (s, s') \in [C]_{er} \land s' \in (q \ast true)
  \]
- **Algorithmically**: $[p] \ C [er: q]$ is manifest if when: $q = \exists X. \ h_q \land \pi_q$:
  \[
  \Rightarrow p = emp \land true
  \]
Pulse-X: *Manifest* Errors

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

- **Formally:** \([p] C [er: q]\) is manifest iff:

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

- **Algorithmically:** \([p] C [er: q]\) is manifest if when: \(q = \exists X. h_q \land \pi_q:\)

  - \(p = \text{emp} \land \text{true}\)
  - \(\text{sat}(q) \land \text{locs}(q) \subseteq X\)
Pulse-X: **Manifest** Errors

- **Intuitively:** the error occurs for **all input states**
- **Formally:** \([p] \mathcal{C} [\text{er: } q]\) is manifest iff:
  \[
  \forall s. \exists s'. (s,s') \in [\mathcal{C}]_{\text{er}} \land s' \in (q \ast \text{true})
  \]
- **Algorithmically:** \([p] \mathcal{C} [\text{er: } q]\) is manifest if when: \(q = \exists X. h_q \land \pi_q : \)
  1. \(p = \text{emp} \land \text{true}\)
  2. \(\text{sat}(q)\) and \(\text{locs}(q) \subseteq \overrightarrow{X}\)
  3. for all \(\overrightarrow{v}: \text{sat} (\overrightarrow{v}/\text{flv}(q) \cup \overrightarrow{X})\)
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 $\models [p] C [er: q]$ with $q \triangleq \exists X_q. \kappa_q \land \pi_q$ denotes a manifest error if:

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

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

   ... null pointer dereference

} calls CRYPTO_malloc (a malloc wrapper)

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));
   ...
}

null pointer dereference

 calls CRYPTO_malloc (a malloc wrapper)

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: \textit{Latent} Errors

An error triple $[p] \, C \, [\text{er: } q]$ is \textit{latent} iff it is not manifest
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;
    } ...
```
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
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;
7.   }
8.   ...
}

Latent Error:
only calls with `args->count==0` can trigger the error
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;
}
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;
}
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;
}
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;
}
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) \( \text{vs} \) **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)

\[
\frac{[p] \ C \ [\varepsilon: q]}{[p \ast r] \ C \ [\varepsilon: q \ast r]}
\]
Extension 1: Concurrent Incorrectness Separation Logic (CISL)

**ISL**

\[
\begin{align*}
[p] & \ C \ [\varepsilon : q] \\
[p \ast r] & \ C \ [\varepsilon : q \ast r]
\end{align*}
\]

**CSL**

\[
\begin{align*}
\{p_1\} & \ C_1 \ \{q_1\} \ \{p_2\} \ C_2 \ \{q_2\} \\
\{p_1 \ast p_2\} & \ C_1 \parallel \ C_2 \ \{q_1 \ast q_2\}
\end{align*}
\]
Extension 1: Concurrent Incorrectness Separation Logic (CISL)

**ISL**

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

**CSL**

\[
\{p_1\} \ C_1 \ {q_1} \quad \{p_2\} \ C_2 \ {q_2}\\
\frac{\{p_1 \ast p_2\} \ C_1 \parallel C_2 \ {q_1 \ast q_2}}{}
\]

**CISL**

\[
[p_1] \ C_1 \ [\varepsilon: q_1] \\
[p_2] \ C_2 \ [\varepsilon: q_2] \\
\frac{[p_1 \ast p_2] \ C_1 \parallel C_2 \ [\varepsilon: q_1 \ast q_2]}{}
\]
Which CML?

CSL (Correctness) Family Tree...

Graph courtesy of Ilya Sergey
Which CISL?

**Pitfall**

The Next 700 Concurrent Separation Logics
Which CISL?

Pitfall

The Next 700
Concurrent **Incorrectness** Separation Logics

Graph courtesy of Ilya Sergey
Which CISL?

Solution

CISL: **general, parametric** framework that can be **instantiated** for different use cases à la Views [Dinsdale-Young et al., 2013]
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<sub>SV</sub>
  ➭ Races: CISL<sub>RD</sub>
  ➭ Deadlocks: CISL<sub>DD</sub>

❖ Sound: **no false positives** (NFP) guaranteed

❖ Underpins **scalable** bug-catching tools (NFP for free)
  ➭ CISL<sub>RD</sub>: analogous to *RacerD* @Meta
  ➭ CISL<sub>DD</sub>: 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
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
@azalearaad
SoundAndComplete.org
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