Concurrent Incorrectness Separation Logic

Azalea Raad\textsuperscript{1,2} \quad Josh Berdine\textsuperscript{2} \quad Derek Dreyer\textsuperscript{3} \quad Peter O’Hearn\textsuperscript{2}

\textsuperscript{1} Imperial College London
\textsuperscript{2} Meta
\textsuperscript{3} MPI-SWS

POPL, 2022

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

= Incorrectness Separation Logic (ISL)
  + Concurrency
  for Concurrent Bug Detection & Analysis
Incorrectness Logic

- Prove the **presence** of bugs — bug catching
- **Under-approximate** reasoning

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

For all states \( s \) in \( q \), \( s \) can be reached by running \( C \) on some \( s' \) in \( p \)
Incorrectness Logic

- Prove the **presence** of bugs — bug catching
- **Under-approximate** reasoning

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

$q$ **under-approximates** $\text{post}(C)p$
Incorrectness Logic

- Prove the **presence** of bugs — bug catching
- **Under-approximate** reasoning

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

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

true positive

post(C)p
Incorrectness Logic

- Prove the **presence** of bugs — bug catching
- **Under-approximate** reasoning

\[
\text{IL} \quad [p] \; C \; [q] \quad \text{iff} \quad \text{post}(C)p \supseteq q
\]

$q$ **under-approximates** post(C)p

true positive

false negative

post(C)p
Incorrectness **Separation Logic**

- Prove the presence of bugs — bug catching
- **Under-approximate** reasoning

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

$q$ under-approximates post(C)p

**Frame**

\[
\text{ISL} \quad [p] \ C \ [q] \quad \frac{\text{Frame}}{[p \ast r] \ C \ [q \ast r]}
\]
Incorrectness \textit{Separation} Logic

- Prove the \textit{presence} of bugs — bug catching
- \textit{Under-approximate} reasoning

\[
\text{ISL} \quad [p] C [q] \quad \text{iff} \quad \text{post}(C)p \geq q
\]

\textbf{Problem 1}

No support for \textit{concurrency}
Several **bug catching** tools for **concurrency** based on **under-approximation**

- RacerD [Blackshear et al., 2018]: **race detection** @Meta
- ToolDL [Brotherston et al., 2021]: **deadlock detection** @Meta

- Each prove a **no-false-positives (NFP) theorem**: bugs found are true bugs
Several **bug catching** tools for **concurrency** based on **under-approximation**

- RacerD [Blackshear et al., 2018]: **race detection** @Meta
- ToolDL [Brotherston et al., 2021]: **deadlock detection**

**Problem 2**

Each analysis must prove NFP **independently**
Several bug catching tools for concurrency based on under-approximation

- RacerD [Blackshear et al., 2018]: race detection @Meta
- ToolDL [Brotherston et al., 2021]: deadlock detection @Meta

Each proves a no-false-positives (NFP) theorem: bugs found are true bugs

**Solution**

**CISL:** Concurrent Incorrectness Separation Logic
Which CISL?

CSL (Correctness) Family Tree...

Graph courtesy of Ilya Sergey
Which CISL?

Pitfall

The Next 700 Concurrent Separation Logics

Graph courtesy of Ilya Sergey
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}

❖ 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
(Concurrent) Incorrectness (Separation) Logic

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

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

\[ p \text{ skip } [\text{ok: } p] \quad p \text{ error( ) } [\text{er: } p] \]
Three Faces of Concurrency Bugs:

1. **Local** Bugs

What are they?

- They are *due to one thread*

\[
\begin{align*}
\text{free}(x); \\
L: [x] := 1 \\ C
\end{align*}
\]

local use-after-free (memory safety) bug at L
Three Faces of Concurrency Bugs:

1. **Local** Bugs

What are they?

- They are due to one thread

```
free(x);
L: [x] := 1  C
```

Local use-after-free (memory safety) bug at L

**Thread-local** analysis tools?

- Existing (sequential) tools out of the box
  e.g. PulseX @Meta (based on ISL)
Three Faces of Concurrency Bugs:

1. **Local** Bugs

What are they?

- They are **due to one thread**

```
free(x);
L: [x] := 1 || C
```

Local use-after-free (memory safety) bug at L

**Thread-local** analysis tools?

- **Existing** (sequential) tools out of the box
e.g. PulseX @Meta (based on ISL)

\[
\text{CISL} \quad \frac{[p] \ C_1 \ [\text{er: } q]}{[p] \ C_1 \ || \ C_2 \ [\text{er: } q]} \quad \text{ParEr}
\]

*Short-circuiting* on errors
Bug is due to two or more threads, under certain interleavings

2. **data-agnostic**: threads do not affect one another’s control flow

\[
L: \text{free}(x) \parallel L': \text{free}(x)
\]

(global) data-agnostic use-after-free bug at L (L')

\[
\text{free}(x); \parallel a := [z]; \\
[z] := 1; \parallel \text{if } (\ast) L: [x] := 1
\]

(global) data-agnostic use-after-free bug at L
Bug is due to two or more threads, under certain interleavings

2. **data-agnostic**: threads do not affect one another’s control flow

\[ \text{L: free}(x) \parallel \text{L'}: \text{free}(x) \]

(global) data-agnostic use-after-free bug at L (L’)

\[ \text{free}(x); \parallel a := [z]; [z] := 1; \parallel \text{if } (\ast) \text{L: } [x] := 1 \]

(global) data-agnostic use-after-free bug at L

3. **data-dependent** bugs: threads do affect one another’s control flow

\[ \text{free}(x); \parallel a := [z]; [z] := 1; \parallel \text{if } (a=1) \text{L: } [x] := 1 \]

(global) data-dependent use-after-free bug at L
Three Faces of Concurrency Bugs:
2, 3. **Global** Bugs

*Thread-local* analysis tools?

2. **data-agnostic**: threads do not affect one another’s control flow

3. **data-dependent** bugs: threads do affect one another’s control flow
Three Faces of Concurrency Bugs:

2, 3. **Global** Bugs

**Thread-local** analysis tools?

2. **data-agnostic**: threads do not affect one another’s control flow
   - encode errors as ok (no short-circuiting)
   - assumed by existing tools: RacerD, DLTool @Meta

3. **data-dependent** bugs: threads do affect one another’s control flow
Three Faces of Concurrency Bugs:
2, 3. **Global** Bugs

**Thread-local** analysis tools?

2. **data-agnostic**: threads do not affect one another’s control flow
   - encode errors as ok (no short-circuiting)
   - assumed by existing tools: RacerD, DLTool @Meta

3. **data-dependent** bugs: threads do affect one another’s control flow
   - possible in CISL theory
   - no existing analysis tools: ongoing work with Meta
Three Faces of Concurrency Bugs:

2. **Global** Bugs

**Thread-local** analysis tools?

2. **data-agnostic**: threads do not affect one another’s control flow

- encode *errors as ok* (no short-circuiting)
- assumed by existing tools: RacerD, DLTool @Meta

3. **data-dependent** bugs: threads do affect one another’s control flow

- possible in CISL theory
- no existing analysis tools: ongoing work with Meta

CISL

\[
\begin{align*}
[p_1] C_1 [ok:q_1] & \quad [p_2] C_2 [ok:q_2] \\
[p_1 \ast p_2] C_1 & \parallel C_2 [ok:q_1 \ast q_2] \\
\end{align*}
\]

Par
Races are *global* bugs by definition:

Two memory accesses (reads/writes), a and b, in program C race iff

1. a and b are **conflicting**:
   - they are by distinct threads
   - on the same location
   - at least one of them is a write

2. they appear **next to each other in an interleaving** (history) of C
Races are global bugs by definition:

Two memory accesses (reads/writes), a and b, in program C race iff

1. a and b are conflicting:
   - they are by distinct threads
   - on the same location
   - at least one of them is a write

2. they appear next to each other in an interleaving (history) of C

Race between lines 3, 5 witnessed by:

$$H = [1, 2, 4, 3, 5, 6]$$

No races
**CISL**

Methodology:
- construct sequential histories
- analyse them for races

\[
[\tau_1 \mapsto [] \ast \tau_2 \mapsto []]
\]

\[
[\tau_1 \mapsto []]
\]

1. lock \(l\);
2. unlock \(l\);
3. \([x] := 1\);

\[
[\tau_2 \mapsto []]
\]

4. lock \(l\);
5. \([x] := 2\);
6. unlock \(l\);

\[
\text{Par}
\]

\[
\text{CISL}
\]

\[
[p_1] C_1 [ok:q_1] [p_2] C_2 [ok:q_2]
\]

\[
[p_1 \ast p_2] C_1 \parallel C_2 [ok:q_1 \ast q_2]
\]
\textbf{CISL}_{RD}

\[
[\tau_1 \leftrightarrow [] \ast \tau_2 \leftrightarrow []]
\]

\begin{align*}
1. & \text{lock } l; \\
2. & \text{unlock } l; \\
3. & [x] := 1;
\end{align*}

\begin{align*}
4. & \text{lock } l; \\
5. & [x] := 2; \\
6. & \text{unlock } l;
\end{align*}

Methodology:

\begin{itemize}
\item Construct sequential histories
\item Analyse them for races
\end{itemize}
Methodology:

➡ construct sequential histories
➡ analyse them for races
H is well-formed iff it respects the lock semantics:

- lock $l$ is acquired only if it is not already held
- lock $l$ is released by $\tau$ only if it is already held by $\tau$
Methodology:

1. Construct sequential histories
2. Analyse them for races

\[
\text{CISL}_{RD}
\]

\[
[\tau_1 \mapsto [] * \tau_2 \mapsto []]
\]

1. lock \( l \);
2. unlock \( l \);
3. \([x] := 1;\)

4. lock \( l \);
5. \([x] := 2;\)
6. unlock \( l \);

\[
\text{CISL}
\]

\[
[p_1] C_1 [\text{ok:} q_1] [p_2] C_2 [\text{ok:} q_2]
\]

\[
[p_1 \* p_2] C_1 \parallel C_2 [\text{ok:} q_1 \* q_2]
\]

Par
CISL\textsubscript{RD}: Unlock Axiom

A history \( H \) is well-formed iff it respects the lock semantics:

- lock \( l \) is acquired only if it is not already held
- lock \( l \) is released by \( \tau \) only if it is already held by \( \tau \)

\[
\begin{align*}
\text{CISL}_{\text{RD}} \quad & H' = H + + [U(\tau, l)] \quad \text{H' is well-formed} \\
\ & [\tau \leftrightarrow H] \quad \text{unlock}_\tau l \quad \text{[ok: } \tau \leftrightarrow H']
\end{align*}
\]
**CISL**

\[
\begin{align*}
\tau_1 \leftrightarrow [] & \quad \tau_2 \leftrightarrow [] \\
1. \text{lock } l; & \quad 4. \text{lock } l; \\
\text{[ok: } \tau_1 \leftrightarrow [L(\tau_1, l)] ] & \quad 5. [x] := 2; \\
2. \text{unlock } l; & \quad 6. \text{unlock } l; \\
\text{[ok: } \tau_1 \leftrightarrow [L(\tau_1, l), U(\tau_1, l)] ] & \\
3. [x] := 1; & \\
\text{[ok: } \tau_1 \leftrightarrow [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] ] &
\end{align*}
\]

**Methodology:**
- construct sequential histories
- analyse them for races

\[
\begin{align*}
\text{CISL} & \quad [p_1] C_1 \ [\text{ok:} q_1] \quad [p_2] C_2 \ [\text{ok:} q_2] \\
\text{Par} & \quad [p_1 \ast p_2] C_1 \ || \ C_2 \ [\text{ok:} q_1 \ast q_2]
\end{align*}
\]
\[ H' = H + \left[ R(\tau, L, x) \right] \]
\[ [\tau \mapsto H] \text{ L: } a :=_x x \quad [\text{ok: } \tau \mapsto H'] \quad \text{RD-Read} \]

\[ H' = H + \left[ W(\tau, L, x) \right] \]
\[ [\tau \mapsto H] \text{ L: } [x] :=_\tau a \quad [\text{ok: } \tau \mapsto H'] \quad \text{RD-Write} \]
CISL\textsubscript{RD}: Memory Access Axioms

We do not record the values read/written
Methodology:
- **construct sequential histories**
- **analyse them for races**

**CISL**

\[
\begin{array}{c}
[p_1] C_1 [\text{ok:} q_1] [p_2] C_2 [\text{ok:} q_2] \\
[p_1 \times p_2] C_1 \parallel C_2 [\text{ok:} q_1 \times q_2]
\end{array}
\]

**CISL\_RD**

\[
\begin{array}{c}
\tau_1 \mapsto [\texttt{ok:} \tau_1 \mapsto [L(\tau_1, l)]] \\
\tau_1 \mapsto [\texttt{ok:} \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l)]] \\
\tau_1 \mapsto [\texttt{ok:} \tau_1 \mapsto [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]] \\
\tau_2 \mapsto [\texttt{ok:} \tau_2 \mapsto [L(\tau_2, l)]] \\
\tau_2 \mapsto [\texttt{ok:} \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x)]] \\
\tau_2 \mapsto [\texttt{ok:} \tau_2 \mapsto [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]]
\end{array}
\]
Methodology:
- **construct sequential histories**
- **analyse them for races**
Methodology:

- construct sequential histories
- analyse them for races

CISL

\[ [ \tau_1 \xleftrightarrow{[]} * \tau_2 \xleftrightarrow{[]} ] \]

1. lock \( l \);
2. unlock \( l \);
3. \([x] := 1;\)
4. lock \( l \);
5. \([x] := 2;\)
6. unlock \( l \);

\[ [ \tau_2 \xleftrightarrow{[]} ] \]

\[ [ \tau_1 \xleftrightarrow{[]} ] \]

\[ [\text{ok: } \tau_1 \xleftrightarrow{[L(\tau_1, l)]}] \]
\[ [\text{ok: } \tau_2 \xleftrightarrow{[L(\tau_2, l)]}] \]
\[ [\text{ok: } \tau_1 \xleftrightarrow{[L(\tau_1, l), U(\tau_1, l)]}] \]
\[ [\text{ok: } \tau_2 \xleftrightarrow{[L(\tau_2, l), W(\tau_2, 5, x)]}] \]
\[ [\text{ok: } \tau_1 \xleftrightarrow{[L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]}] \]
\[ [\text{ok: } \tau_2 \xleftrightarrow{[L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]}] \]

\[ \text{Par} \]

\[ [p_1] C_1 [\text{ok: } q_1] [p_2] C_2 [\text{ok: } q_2] \]

\[ [p_1 * p_2] C_1 || C_2 [\text{ok: } q_1 * q_2] \]
CISL\textsubscript{RD}: \textit{race} Predicate

\[ \tau_1 \mapsto H_1 \ast \tau_2 \mapsto H_2 \Rightarrow \text{race}(L_1, L_2, H) \text{ iff:} \]

there exist \( H'_1, H'_2, H', a, b \) such that:

\( \Rightarrow \) a and b are conflicting accesses

\( \Rightarrow H_1 = H'_1 ++ [a] ++ \) and \( H_2 = H'_2 ++ [b] ++ \)

\( \Rightarrow H = H' ++ [a, b] \)

\( \Rightarrow H' \) is a permutation of \( H'_1 ++ H'_2 \)

\( \Rightarrow H \) is well-formed
### Methodology:

- construct sequential histories
- analyse them for races
CISL_{RD} \quad [\tau_1 \leftrightarrow [] \ast \tau_2 \leftrightarrow []]

\begin{align*}
[\tau_1 \leftrightarrow []] & \\
1. & \text{lock l;} \\
[\text{ok: } \tau_1 \leftrightarrow [L(\tau_1, l)]] & \\
2. & \text{unlock l;} \\
[\text{ok: } \tau_1 \leftrightarrow [L(\tau_1, l), U(\tau_1, l)]] & \\
3. & [x] := 1; \\
[\text{ok: } \tau_1 \leftrightarrow [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)]] & \\
\end{align*}

\begin{align*}
[\tau_2 \leftrightarrow []] & \\
4. & \text{lock l;} \\
[\text{ok: } \tau_2 \leftrightarrow [L(\tau_2, l)]] & \\
5. & [x] := 2; \\
[\text{ok: } \tau_2 \leftrightarrow [L(\tau_2, l), W(\tau_2, 5, x)]] & \\
6. & \text{unlock l;} \\
[\text{ok: } \tau_2 \leftrightarrow [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]] & \\
\end{align*}

\begin{align*}
[\text{ok: } \tau_1 \leftrightarrow [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] \ast \tau_2 \leftrightarrow [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]] & \\
[\text{ok: } \tau_1 \leftrightarrow [L(\tau_1, l), U(\tau_1, l), W(\tau_1, 3, x)] \ast \tau_2 \leftrightarrow [L(\tau_2, l), W(\tau_2, 5, x), U(\tau_2, l)]] & \\
\wedge \text{race}(3, 5, [L(\tau_1, l), U(\tau_1, l), L(\tau_2, l), W(\tau_1, 3, x), W(\tau_2, 5, x)]) & \\
\end{align*}

Methodology:
- construct sequential histories
- analyse them for races

\[
\begin{array}{c}
\text{CISL} \\
[p_1] C_1 \text{[ok: q]} [p_2] C_2 \text{[ok: q]} \\
[p_1 \ast p_2] C_1 \parallel C_2 \text{[ok: q]} \\
\text{Par}
\end{array}
\]
Simple yet
**Effective in Practice**
à la RacerD

Methodology:
- construct sequential histories
- analyse them for races
Conclusions

❖ **First** work to adapt **under-approximate** reasoning for **concurrent bug detection**

❖ **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$_{RD}$: à la **RacerD** @Meta;  CISL$_{DD}$: à la **DLTool** @Meta
Conclusions

❖ **First** work to adapt **under-approximate** reasoning for **concurrent bug detection**

❖ **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$_{RD}$: à la **RacerD** @Meta;  CISL$_{DD}$: à la **DLTool** @Meta

❖ Future work:
  ➡ CISL for data-dependent bugs
  ➡ automated tools based on CISL, e.g. data-dependent races, deadlocks, memory safety errors
  ➡ mechanisation
Conclusions

❖ **First** work to adapt under-approximate reasoning for concurrent bug detection

❖ **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}: à la RacerD @Meta;  CISL_{DD}: à la DLTool @Meta

❖ Future work:
  ➡ CISL for data-dependent bugs
  ➡ automated tools based on CISL, e.g. data-dependent races, deadlocks, memory safety errors
  ➡ mechanisation

Thank You for Listening!