The Essence of Higher-Order Concurrent Separation Logic

Robbert Krebbers[1] Ralf Jung[2] Aleš Bizjak[3]
Jacques-Henri Jourdan[2] Derek Dreyer[2] Lars Birkedal[3]

[1]Delft University of Technology, The Netherlands

[2]MPI-SWS, Saarland Informatics Campus, Germany

[3]Aarhus University, Denmark

April 18, 2017 @ ESOP, Uppsala, Sweden

This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

- **Fine-grained concurrency:** synchronization primitives and lock-free data structures are implemented

This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

- **Fine-grained concurrency:** synchronization primitives and lock-free data structures are implemented
- **Modular:** reusable and composable specifications

This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

- **Fine-grained concurrency:** synchronization primitives and lock-free data structures are implemented
- **Modular:** reusable and composable specifications
- **Language independent:** parametrized by the language

This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

- **Fine-grained concurrency:** synchronization primitives and lock-free data structures are implemented
- **Modular:** reusable and composable specifications
- **Language independent:** parametrized by the language
- **Simple foundations:** small set of primitive rules

This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

- **Fine-grained concurrency:** synchronization primitives and lock-free data structures are implemented
- **Modular:** reusable and composable specifications
- **Language independent:** parametrized by the language
- **Simple foundations:** small set of primitive rules
- **Coq:** provides practical support for doing proofs in Iris

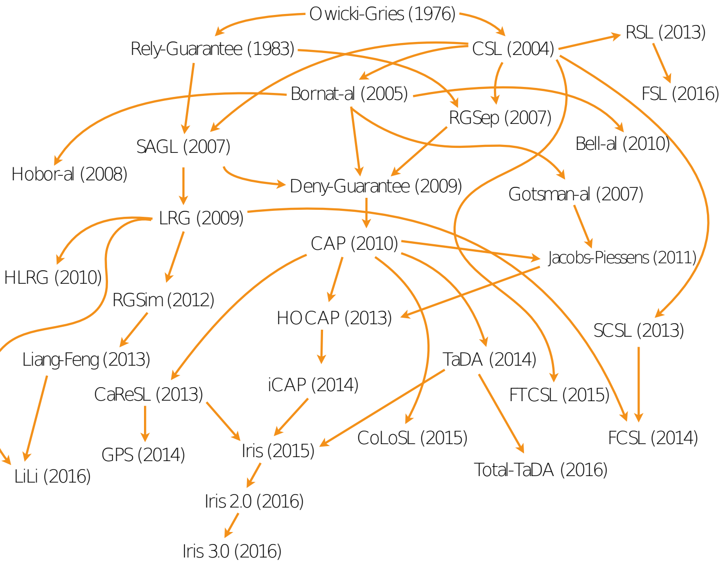
This talk is about Iris

A language independent higher-order separation logic with a simple foundations for modular reasoning about fine-grained concurrency in Coq.

- **Fine-grained concurrency:** synchronization primitives and lock-free data structures are implemented
- **Modular:** reusable and composable specifications
- **Language independent:** parametrized by the language
- **Simple foundations:** small set of primitive rules
- **Coq:** provides practical support for doing proofs in Iris

This talk: simplifying the foundations of Iris

Aren't there enough logics for concurrency yet?



Picture by Ilya Sergey

Problem statement

All of these logics **bake in** complicated protocol mechanisms:

$$\frac{\begin{array}{c}\Gamma, \Delta \mid \Phi \vdash \mathsf{stable}(\mathsf{P}) \qquad \Gamma, \Delta \mid \Phi \vdash \forall y.\ \mathsf{stable}(\mathsf{Q}(y)) \\ \Gamma, \Delta \mid \Phi \vdash n \in C \qquad \Gamma, \Delta \mid \Phi \vdash \forall x \in X.\ (x, f(x)) \in \overline{T(A)} \vee f(x) = x \\ \Gamma \mid \Phi \vdash \forall x \in X.\ (\Delta).\langle \mathsf{P} * \circledast_{\alpha \in A}[\alpha]^{\mathsf{n}}_{g(\alpha)} * \triangleright I(x)\rangle\ c\ \langle \mathsf{Q}(x) * \triangleright I(f(x))\rangle^{C \setminus \{n\}}\end{array}}{\begin{array}{c}\Gamma \mid \Phi \vdash (\Delta).\ \langle \mathsf{P} * \circledast_{\alpha \in A}[\alpha]^{\mathsf{n}}_{g(\alpha)} * \mathsf{region}(X, T, I, n)\rangle \\ c \\ \langle \exists x.\ \mathsf{Q}(x) * \mathsf{region}(\{f(x)\}, T, I, n)\rangle^{C}\end{array}}\ \textsc{Atomic}$$

$$\frac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_{\pi}} b_0.\ (\!|\pi[\![b]\!] * P|\!)\ i \Rrightarrow_{1} a\ (\!|x.\ \exists b' \overset{\text{guar}}{\sqsupseteq_{\pi}} b.\ \pi[\![b']\!] * Q|\!)}{\mathcal{C} \vdash \left\{\boxed{b_0}^{n}_{\pi} * \triangleright P\right\}\ i \Rrightarrow a\ \left\{x.\ \exists b'.\ \boxed{b'}^{n}_{\pi} * Q\right\}}\ \textsc{UpdIsl}$$

Use atomic rule

$$\frac{\begin{array}{c}a \notin \mathcal{A} \quad \forall x \in X.\, (x, f(x)) \in \mathcal{T}_{\mathbf{t}}(\mathrm{G})^{*} \\ \lambda; \mathcal{A} \vdash \forall\!\!\!\forall x \in X.\, \langle p_p \mid I(\mathbf{t}^{\lambda}_{a}(x)) * p(x) * [\mathrm{G}]_a\rangle\ \mathbb{C}\ \ \exists\!\!\!\exists y \in Y.\, \langle q_p(x,y) \mid I(\mathbf{t}^{\lambda}_{a}(f(x))) * q(x,y)\rangle\end{array}}{\lambda + 1; \mathcal{A} \vdash \forall\!\!\!\forall x \in X.\, \langle p_p \mid \mathbf{t}^{\lambda}_{a}(x) * p(x) * [\mathrm{G}]_a\rangle\ \mathbb{C}\ \ \exists\!\!\!\exists y \in Y.\, \langle q_p(x,y) \mid \mathbf{t}^{\lambda}_{a}(f(x)) * q(x,y)\rangle}$$

$$\frac{\begin{array}{c}\Gamma \mid \Phi \vdash x \in X \qquad \Gamma \mid \Phi \vdash \forall \alpha \in \mathsf{Action}.\ \forall x \in \mathsf{SId} \times \mathsf{SId}.\ up(T(\alpha)(x)) \\ \Gamma \mid \Phi \vdash A \text{ and } B \text{ are finite} \qquad \Gamma \mid \Phi \vdash C \text{ is infinite} \\ \Gamma \mid \Phi \vdash \forall n \in C.\ \mathsf{P} * \circledast_{\alpha \in A}[\alpha]^{n}_{1} \Rightarrow \triangleright I(n)(x) \\ \Gamma \mid \Phi \vdash \forall n \in C.\ \forall s.\ \mathsf{stable}(I(n)(s)) \qquad \Gamma \mid \Phi \vdash A \cap B = \emptyset\end{array}}{\Gamma \mid \Phi \vdash \mathsf{P} \sqsubseteq^{C} \exists n \in C.\ \mathsf{region}(X, T, I(n), n) * \circledast_{\alpha \in B}[\alpha]^{n}_{1}}\ \textsc{VAlloc}$$

Update region rule

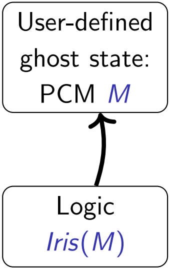
$$\frac{\lambda; \mathcal{A} \vdash \forall\!\!\!\forall x \in X.\, \left\langle p_p \,\middle|\, I(\mathbf{t}^{\lambda}_{a}(x)) * p(x)\right\rangle\ \mathbb{C}\ \ \exists\!\!\!\exists y \in Y.\, \left\langle q_p(x,y) \,\middle|\, \begin{array}{c} I(\mathbf{t}^{\lambda}_{a}(Q(x))) * q_1(x,y) \\ \vee\, I(\mathbf{t}^{\lambda}_{a}(x)) * q_2(x,y)\end{array}\right\rangle}{\lambda{+}1; a : x \in X \rightsquigarrow Q(x), \mathcal{A} \vdash \begin{array}{c}\forall\!\!\!\forall x \in X.\, \langle p_p \mid \mathbf{t}^{\lambda}_{a}(x) * p(x) * a \Rrightarrow \blacklozenge\rangle \\ \mathbb{C} \\ \exists\!\!\!\exists y \in Y.\, \left\langle q_p(x,y) \,\middle|\, \begin{array}{c}\exists z \in Q(x).\, \mathbf{t}^{\lambda}_{a}(z) * q_1(x,y) * a \Rrightarrow (x,z) \\ \vee\, \mathbf{t}^{\lambda}_{a}(x) * q_2(x,y) * a \Rrightarrow \blacklozenge\end{array}\right\rangle\end{array}}$$

The Iris approach: derive these protocol mechanisms (+ more) from a small set of primitives

The history of Iris

Iris 1. Many protocol mechanisms can be derived from just 2 primitives:

- User-defined ghost state
- Invariants



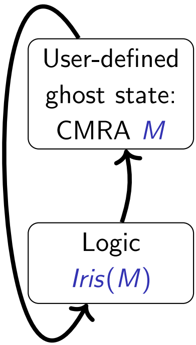
The history of Iris

Iris 1. Many protocol mechanisms can be derived from just 2 primitives:

- User-defined ghost state
- Invariants

Iris 2. *Higher-order ghost state:*

- More powerful protocol mechanisms
- Unify invariants and ghost state in the model



The history of Iris

Iris 1. Many protocol mechanisms can be derived from just 2 primitives:

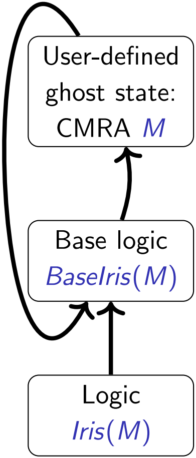
- User-defined ghost state
- Invariants

Iris 2. *Higher-order ghost state:*

- More powerful protocol mechanisms
- Unify invariants and ghost state in the model

Iris 3. Apply the Iris methodology to itself:

- Reduce the logic to a bare minimum
- Define Hoare triples, invariants, view shifts, ... as derived notions



The Iris 3 base logic

Higher-order logic ($\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$, $=$) with:

- The BI connectives $*$ and $-\!\!*$
- A notion of resource ownership $\mathsf{Own}(\_)$
- A handful of modalities $\triangleright$, $\square$ and $\Rrightarrow$
- A guarded fixpoint combinator $\mu\, x.\, P$

The Iris 3 base logic

Higher-order logic ($\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$, $=$) with:

- The BI connectives $*$ and $-\!\!*$
- A notion of resource ownership $\mathsf{Own}(\_)$
- A handful of modalities $\triangleright$, $\Box$ and $\mathrel{|\!\!\!\Rrightarrow}$
- A guarded fixpoint combinator $\mu\, x.\, P$

Simple enough so that:

- Each connective has just a single purpose
- It is easy to model and formalize (in Coq)

Expressive enough so that:

- Expressive program logics – like the original Iris program logic – can be defined concisely
- It provides more freedom beyond Hoare logic: logical relations for program refinements, modeling type systems, ...

Preview of the rules of the Iris 3 base logic

Laws of (affine) bunched implications

$$\begin{array}{rcl} \mathsf{True} * P & \dashv\vdash & P \\ P * Q & \vdash & Q * P \\ (P * Q) * R & \vdash & P * (Q * R) \end{array}$$

$$\frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R} \qquad \frac{P \vdash Q \mathrel{-\!\!*} R}{P * Q \vdash R}$$

Laws for resources and validity

$$\begin{array}{c} \mathsf{Own}(a) * \mathsf{Own}(b) \dashv\vdash \mathsf{Own}(a \cdot b) \\ \mathsf{Own}(a) \vdash \mathcal{V}(a) \end{array} \qquad \begin{array}{c} \mathsf{True} \vdash \mathsf{Own}(\varepsilon) \\ \mathcal{V}(a \cdot b) \vdash \mathcal{V}(a) \end{array} \qquad \begin{array}{c} \mathsf{Own}(a) \vdash \square\, \mathsf{Own}(|a|) \\ \mathcal{V}(a) \vdash \square\, \mathcal{V}(a) \end{array}$$

Laws for the basic update modality

$$\frac{P \vdash Q}{\Rrightarrow P \vdash \Rrightarrow Q} \qquad P \vdash \Rrightarrow P \qquad \Rrightarrow \Rrightarrow P \vdash \Rrightarrow P$$

$$Q * \Rrightarrow P \vdash \Rrightarrow (Q * P) \qquad \frac{a \rightsquigarrow B}{\mathsf{Own}(a) \vdash \Rrightarrow \exists b \in B.\, \mathsf{Own}(b)}$$

Laws for the always modality

$$\frac{P \vdash Q}{\square\, P \vdash \square\, Q} \qquad \square\, P \vdash P \qquad \begin{array}{c} \mathsf{True} \vdash \square\, \mathsf{True} \\ \square\,(P \land Q) \vdash \square\,(P * Q) \\ \square\, P \land Q \vdash \square\, P * Q \end{array} \qquad \begin{array}{c} \square\, P \vdash \square\,\square\, P \\ \forall x.\ \square\, P \vdash \square\, \forall x.\, P \\ \square\, \exists x.\, P \vdash \exists x.\ \square\, P \end{array}$$

Laws for the later modality

$$\frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad (\triangleright P \Rightarrow P) \vdash P \qquad \begin{array}{c} \forall x.\ \triangleright P \vdash \triangleright \forall x.\, P \\ \triangleright \exists x.\, P \vdash \triangleright \mathsf{False} \lor \exists x.\ \triangleright P \end{array} \qquad \begin{array}{c} \triangleright (P * Q) \dashv\vdash \triangleright P * \triangleright Q \\ \square \triangleright P \dashv\vdash \triangleright \square\, P \end{array}$$

Laws for timeless assertions

$$\triangleright P \vdash \triangleright \mathsf{False} \lor (\triangleright \mathsf{False} \Rightarrow P) \qquad \triangleright \mathsf{Own}(a) \vdash \exists b.\, \mathsf{Own}(b) \land \triangleright(a = b)$$

Part #1: brief introduction to concurrent separation logic (CSL)

Hoare triples and separation logic

Hoare triples for partial program correctness:

$$\underbrace{\{P\}}_{\text{Precondition}} e \{\underbrace{w}_{\text{Binder for return value}}. \underbrace{Q}_{\text{Postcondition}}\}$$

If the initial state satisfies P, then:

- e does not get stuck/crash
- if e terminates with value v, the final state satisfies $Q[v/w]$

Hoare triples and separation logic

Hoare triples for partial program correctness:

$$\{P\}e\{w.\, Q\}$$

Precondition

Binder for return value

Postcondition

If the initial state satisfies P, then:

- e does not get stuck/crash
- if e terminates with value v, the final state satisfies $Q[v/w]$

Example:

$$\{x \mapsto v_1 * y \mapsto v_2\} swap(x, y) \{w.\, w = () \wedge x \mapsto v_2 * y \mapsto v_1\}$$

the $*$ ensures that x and y are different

Concurrent separation logic [O'Hearn]

The *par* rule:

$$\frac{\{P_1\}\,e_1\,\{Q_1\} \qquad \{P_2\}\,e_2\,\{Q_2\}}{\{P_1 * P_2\}\,e_1 || e_2\,\{Q_1 * Q_2\}}$$

Concurrent separation logic [O'Hearn]

The *par* rule:

$$\frac{\{P_1\}\,e_1\,\{Q_1\} \qquad \{P_2\}\,e_2\,\{Q_2\}}{\{P_1 * P_2\}\,e_1 || e_2\,\{Q_1 * Q_2\}}$$

For example:

$$\{x \mapsto 4 * y \mapsto 6\}$$

$$x := !\,x + 2 \;\Big\|\; y := !\,y + 2$$

$$\{x \mapsto 6 * y \mapsto 8\}$$

Concurrent separation logic [O'Hearn]

The *par* rule:

$$\frac{\{P_1\}e_1\{Q_1\} \qquad \{P_2\}e_2\{Q_2\}}{\{P_1 * P_2\}e_1||e_2\{Q_1 * Q_2\}}$$

For example:

$$\begin{array}{c}
\{x \mapsto 4 * y \mapsto 6\} \\
\begin{array}{c||c}
\{x \mapsto 4\} & \{y \mapsto 6\} \\
x := !\,x + 2 & y := !\,y + 2 \\
 &
\end{array} \\
\{x \mapsto 6 * y \mapsto 8\}
\end{array}$$

Concurrent separation logic [O'Hearn]

The *par* rule:

$$\frac{\{P_1\}\, e_1 \,\{Q_1\} \qquad \{P_2\}\, e_2 \,\{Q_2\}}{\{P_1 * P_2\}\, e_1 || e_2 \,\{Q_1 * Q_2\}}$$

For example:

$$\begin{array}{c}
\{x \mapsto 4 * y \mapsto 6\} \\
\begin{array}{l|l}
\{x \mapsto 4\} & \{y \mapsto 6\} \\
x := !\,x + 2 & y := !\,y + 2 \\
\{x \mapsto 6\} & \{y \mapsto 8\}
\end{array} \\
\{x \mapsto 6 * y \mapsto 8\}
\end{array}$$

Concurrent separation logic [O'Hearn]

The *par* rule:

$$\frac{\{P_1\}\, e_1 \,\{Q_1\} \qquad \{P_2\}\, e_2 \,\{Q_2\}}{\{P_1 * P_2\}\, e_1 || e_2 \,\{Q_1 * Q_2\}}$$

For example:

$$\begin{array}{c}
\{x \mapsto 4 * y \mapsto 6\} \\
\begin{array}{l||l}
\{x \mapsto 4\} & \{y \mapsto 6\} \\
x := \,!\, x + 2 & y := \,!\, y + 2 \\
\{x \mapsto 6\} & \{y \mapsto 8\}
\end{array} \\
\{x \mapsto 6 * y \mapsto 8\}
\end{array}$$

Works great for concurrent programs without shared memory: concurrent quick sort, concurrent merge sort, . . .

What about shared state/racy programs?

A classic problem:

$$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$$

$$\begin{array}{l} \texttt{fetch\_and\_add}(x, 2) \end{array} \;\Big\|\; \texttt{fetch\_and\_add}(x, 2)$$

$$!\, x$$

Where $\texttt{fetch\_and\_add}(x, y)$ is the atomic version of $x := !\, x + y$.

What about shared state/racy programs?

A classic problem:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in} \\
\left(\texttt{fetch\_and\_add}(x, 2) \;\middle\|\; \texttt{fetch\_and\_add}(x, 2)\right); \\
!\, x \\
\{w.\, w = 4\}
\end{array}
$$

Where $\texttt{fetch\_and\_add}(x, y)$ is the atomic version of $x := !\, x + y$.

What about shared state/racy programs?

A classic problem:

$$\{\mathsf{True}\}$$
$$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$$
$$\{x \mapsto 0\}$$
$$\texttt{fetch\_and\_add}(x, 2) \;\Big\|\; \texttt{fetch\_and\_add}(x, 2)$$
$$!\, x$$
$$\{w.\, w = 4\}$$

Where $\texttt{fetch\_and\_add}(x, y)$ is the atomic version of $x := !\, x + y$.

What about shared state/racy programs?

A classic problem:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in} \\
\{x \mapsto 0\} \\
\left(\begin{array}{l} \{??\} \\ \texttt{fetch\_and\_add}(x, 2) \\ \{??\} \end{array} \;\middle\|\; \begin{array}{l} \{??\} \\ \texttt{fetch\_and\_add}(x, 2) \\ \{??\} \end{array}\right) \\
!\,x \\
\{w.\, w = 4\}
\end{array}
$$

Where $\texttt{fetch\_and\_add}(x, y)$ is the atomic version of $x := !\,x + y$.

Problem: can only give ownership of x to one thread

Sharing resources using invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Sharing resources using invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\boxed{R} \quad \vdash \{R * P\}\, e\, \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \quad \vdash \{P\}\, e\, \{Q\}}$$

Sharing resources using invariants

The invariant assertion $\boxed{R}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\boxed{R} \vdash \{R * P\}\, e\, \{R * Q\} \qquad e \text{ atomic}}{\boxed{R} \vdash \{P\}\, e\, \{Q\}}$$

Invariant allocation:

$$\frac{\boxed{R} \vdash \{P\}\, e\, \{Q\}}{\{R * P\}\, e\, \{Q\}}$$

Sharing resources using invariants

The invariant assertion $\boxed{R}^{\mathcal{N}}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\boxed{R}^{\mathcal{N}} \vdash \{R * P\}\, e\, \{R * Q\}_{\mathcal{E}} \qquad e \text{ atomic}}{\boxed{R}^{\mathcal{N}} \vdash \{P\}\, e\, \{Q\}_{\mathcal{E} \uplus \mathcal{N}}}$$

Invariant allocation:

$$\frac{\boxed{R}^{\mathcal{N}} \vdash \{P\}\, e\, \{Q\}_{\mathcal{E}}}{\{R * P\}\, e\, \{Q\}_{\mathcal{E}}}$$

Technical detail: names are needed to avoid *reentrancy*, i.e., opening the same invariant twice

Sharing resources using invariants

The invariant assertion $\boxed{R}^{\mathcal{N}}$ expresses that R is maintained as an invariant on the state

Invariant opening:

$$\frac{\boxed{R}^{\mathcal{N}} \vdash \{\triangleright R * P\}\, e\, \{\triangleright R * Q\}_{\mathcal{E}} \qquad e \text{ atomic}}{\boxed{R}^{\mathcal{N}} \vdash \{P\}\, e\, \{Q\}_{\mathcal{E} \uplus \mathcal{N}}}$$

Invariant allocation:

$$\frac{\boxed{R}^{\mathcal{N}} \vdash \{P\}\, e\, \{Q\}_{\mathcal{E}}}{\{\triangleright R * P\}\, e\, \{Q\}}$$

Technical detail: names are needed to avoid *reentrancy*, i.e., opening the same invariant twice

Other technical detail: the later $\triangleright$ is needed to support impredicative invariants, i.e., $\boxed{\ldots \boxed{R}^{\mathcal{N}_2} \ldots}^{\mathcal{N}_1}$

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$
$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$

$$\texttt{fetch\_and\_add}(x, 2) \quad \Big\| \quad \texttt{fetch\_and\_add}(x, 2)$$

$!\,x$

$\{n.\, \mathit{even}(n)\}$

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$
$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$
$\{x \mapsto 0\}$

$\texttt{fetch\_and\_add}(x, 2)$	$\texttt{fetch\_and\_add}(x, 2)$

$!\,x$

$\{n.\, \mathit{even}(n)\}$

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$
$\texttt{let}\ x = \texttt{ref}(0)\ \texttt{in}$
$\{x \mapsto 0\}$
allocate $\boxed{\exists n.\, x \mapsto n \wedge even(n)}$

$$\texttt{fetch\_and\_add}(x, 2) \quad \Bigg\| \quad \texttt{fetch\_and\_add}(x, 2)$$

$!\,x$

$\{n.\, even(n)\}$

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$
$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$
$\{x \mapsto 0\}$
allocate $\boxed{\exists n.\, x \mapsto n \land \mathit{even}(n)}$

$$
\left(\begin{array}{c} \{\mathsf{True}\} \\ \texttt{fetch\_and\_add}(x, 2) \\ \{\mathsf{True}\} \end{array}\right) \;\Big\|\; \left(\begin{array}{c} \{\mathsf{True}\} \\ \texttt{fetch\_and\_add}(x, 2) \\ \{\mathsf{True}\} \end{array}\right)
$$

$!\,x$

$\{n.\, \mathit{even}(n)\}$

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$

$\texttt{let}\ x = \texttt{ref}(0)\ \texttt{in}$

$\{x \mapsto 0\}$

allocate $\boxed{\exists n.\, x \mapsto n \wedge even(n)}$

$$\begin{array}{l||l}
\{\mathsf{True}\} & \{\mathsf{True}\} \\
\quad \{x \mapsto n \wedge even(n)\} & \\
\quad \texttt{fetch_and_add}(x, 2) & \quad \texttt{fetch_and_add}(x, 2) \\
\quad \{x \mapsto n + 2 \wedge even(n + 2)\} & \\
\{\mathsf{True}\} & \{\mathsf{True}\}
\end{array}$$

$!\,x$

$\{n.\, even(n)\}$

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$

$\texttt{let}\ x = \texttt{ref}(0)\ \texttt{in}$

$\{x \mapsto 0\}$

allocate $\boxed{\exists n.\, x \mapsto n \wedge \mathit{even}(n)}$

$$
\begin{array}{l|l}
\{\mathsf{True}\} & \{\mathsf{True}\} \\
\quad \{x \mapsto n \wedge \mathit{even}(n)\} & \quad \{x \mapsto n \wedge \mathit{even}(n)\} \\
\quad \texttt{fetch\_and\_add}(x, 2) & \quad \texttt{fetch\_and\_add}(x, 2) \\
\quad \{x \mapsto n + 2 \wedge \mathit{even}(n + 2)\} & \quad \{x \mapsto n + 2 \wedge \mathit{even}(n + 2)\} \\
\{\mathsf{True}\} & \{\mathsf{True}\}
\end{array}
$$

$!\,x$

$\{n.\, \mathit{even}(n)\}$

Invariants in action

Let us consider a simpler problem first:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\ x = \texttt{ref}(0)\ \texttt{in} \\
\{x \mapsto 0\} \\
\text{allocate}\ \boxed{\exists n.\, x \mapsto n \wedge \mathit{even}(n)} \\
\begin{array}{l|l}
\{\mathsf{True}\} & \{\mathsf{True}\} \\
\quad \{x \mapsto n \wedge \mathit{even}(n)\} & \quad \{x \mapsto n \wedge \mathit{even}(n)\} \\
\quad \texttt{fetch\_and\_add}(x, 2) & \quad \texttt{fetch\_and\_add}(x, 2) \\
\quad \{x \mapsto n + 2 \wedge \mathit{even}(n+2)\} & \quad \{x \mapsto n + 2 \wedge \mathit{even}(n+2)\} \\
\{\mathsf{True}\} & \{\mathsf{True}\}
\end{array} \\
\quad \{x \mapsto n \wedge \mathit{even}(n)\} \\
\quad !\, x \\
\quad \{n.\, x \mapsto n \wedge \mathit{even}(n)\} \\
\{n.\, \mathit{even}(n)\}
\end{array}
$$

Invariants in action

Let us consider a simpler problem first:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\, x = \texttt{ref}(0)\ \texttt{in} \\
\{x \mapsto 0\} \\
\text{allocate } \boxed{\exists n.\, x \mapsto n \wedge even(n)} \\
\begin{array}{l|l}
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \{x \mapsto n \wedge even(n)\} \\
\quad \texttt{fetch\_and\_add}(x, 2) \\
\quad \{x \mapsto n + 2 \wedge even(n + 2)\} \\
\{\mathsf{True}\}
\end{array}
&
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \{x \mapsto n \wedge even(n)\} \\
\quad \texttt{fetch\_and\_add}(x, 2) \\
\quad \{x \mapsto n + 2 \wedge even(n + 2)\} \\
\{\mathsf{True}\}
\end{array}
\end{array} \\
\quad \{x \mapsto n \wedge even(n)\} \\
\quad !\,x \\
\quad \{n.\, x \mapsto n \wedge even(n)\} \\
\{n.\, even(n)\}
\end{array}
$$

Problem: still cannot prove it returns 4

Ghost variables

Consider the invariant:

$$\boxed{\exists n.\, x \mapsto n * \ldots}$$

How to relate the quantified value to the state of the threads?

Ghost variables

Consider the invariant:

$$\boxed{\exists n.\, x \mapsto n * \ldots}$$

How to relate the quantified value to the state of the threads?

Solution: ghost variables

Ghost variables

Consider the invariant:

$$\boxed{\exists n.\, x \mapsto n * \ldots}$$

How to relate the quantified value to the state of the threads?

Solution: ghost variables

Ghost variables are allocated in pairs:

$$\text{True} \quad \Rightarrow \quad |\!\!\Rrightarrow \exists \gamma.\ \underbrace{\gamma \hookrightarrow_{\bullet} n}_{\text{in the invariant}} \quad * \quad \underbrace{\gamma \hookrightarrow_{\circ} n}_{\text{in the Hoare triple}}$$

Ghost variables

Consider the invariant:

$$\boxed{\exists n_1, n_2.\, x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$$

How to relate the quantified value to the state of the threads?

Solution: ghost variables

Ghost variables are allocated in pairs:

$$\text{True} \quad \Rightarrow \quad \mathrel{|\!\!\Rrightarrow} \exists \gamma.\ \underbrace{\gamma \hookrightarrow_\bullet n}_{\text{in the invariant}} \quad * \quad \underbrace{\gamma \hookrightarrow_\circ n}_{\text{in the Hoare triple}}$$

Ghost variables

Consider the invariant:

$$\boxed{\exists n_1, n_2.\, x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_{\bullet} n_1 * \gamma_2 \hookrightarrow_{\bullet} n_2}$$

How to relate the quantified value to the state of the threads?

Solution: ghost variables

Ghost variables are allocated in pairs:

$$\text{True} \quad \Rightarrow \quad \mid\!\Rrightarrow \exists \gamma.\ \underbrace{\gamma \hookrightarrow_{\bullet} n}_{\text{in the invariant}} \quad * \quad \underbrace{\gamma \hookrightarrow_{\circ} n}_{\text{in the Hoare triple}}$$

When you own both parts you obtain that the values are equal and can update both parts:

$$\gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} m \quad \Rightarrow \quad n = m$$

$$\gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} m \quad \Rightarrow \quad \mid\!\Rrightarrow (\gamma \hookrightarrow_{\bullet} n' * \gamma \hookrightarrow_{\circ} n')$$

Mechanisms for concurrent reasoning in Iris

The Iris approach:

- Ghost variables are generalized to range over any user-provided RA
- RAs provide uniform treatment of fractional permissions, state transition systems, ...

Resource algebra (RA):

- Carrier M
- Composition $(\cdot) : M \to M \to M$
- Validity predicate $\mathcal{V} \subseteq M$

Satisfying certain laws

Rules for ghost state in Iris:

$$\mathcal{V}(a) \Rightarrow \dot{\Rrightarrow} \exists \gamma.\, \gamma \hookrightarrow a$$
$$\gamma \hookrightarrow a * \gamma \hookrightarrow b \Leftrightarrow \gamma \hookrightarrow (a \cdot b)$$
$$\gamma \hookrightarrow a \Rightarrow \mathcal{V}(a)$$

$$\frac{\forall a_{\mathrm{f}}.\, \mathcal{V}(a \cdot a_{\mathrm{f}}) \Rightarrow \mathcal{V}(b \cdot a_{\mathrm{f}})}{\gamma \hookrightarrow a \Rightarrow \dot{\Rrightarrow} \gamma \hookrightarrow b}$$

Mechanisms for concurrent reasoning in Iris

The Iris approach:

- Ghost variables are generalized to range over any user-provided RA
- RAs provide uniform treatment of fractional permissions, state transition systems, ...

New in Iris 3, use ghost state to:

- Define invariants $\boxed{P}^{\mathcal{N}}$
- Define Hoare triples $\{P\}\, e\, \{Q\}$

Resource algebra (RA):

- Carrier M
- Composition $(\cdot) : M \to M \to M$
- Validity predicate $\mathcal{V} \subseteq M$

Satisfying certain laws

Rules for ghost state in Iris:

$$\mathcal{V}(a) \Rightarrow \Rrightarrow \exists \gamma.\, \gamma \hookrightarrow a$$
$$\gamma \hookrightarrow a * \gamma \hookrightarrow b \Leftrightarrow \gamma \hookrightarrow (a \cdot b)$$
$$\gamma \hookrightarrow a \Rightarrow \mathcal{V}(a)$$

$$\frac{\forall a_{\mathrm{f}}.\, \mathcal{V}(a \cdot a_{\mathrm{f}}) \Rightarrow \mathcal{V}(b \cdot a_{\mathrm{f}})}{\gamma \hookrightarrow a \Rightarrow \Rrightarrow \gamma \hookrightarrow b}$$

Part #2: encoding Hoare triples and invariants

Encoding Hoare triples

Step 1: define Hoare triple in terms of weakest preconditions:

$$\{P\}\, e\, \{w.\, Q\} \triangleq \square(P \mathrel{-\!\!*} \mathsf{wp}\; e\, \{w.\, Q\})$$

where $\mathsf{wp}\; e\, \{w.\, Q\}$ gives the *weakest precondition* under which:

- all executions of e are safe
- all return values v of e satisfy the postcondition $Q[v/w]$

Encoding Hoare triples

Step 1: define Hoare triple in terms of weakest preconditions:

$$\{P\}\, e\, \{w.\, Q\} \triangleq \Box(P \twoheadrightarrow \mathsf{wp}\, e\, \{w.\, Q\})$$

where $\mathsf{wp}\, e\, \{w.\, Q\}$ gives the *weakest precondition* under which:

- all executions of e are safe
- all return values v of e satisfy the postcondition $Q[v/w]$

Step 2: define weakest precondition:

$$\mathsf{wp}\, e\, \{w.\, Q\} \triangleq \begin{cases} Q[e/w] & \text{if } e \in \mathit{Val} \\ \forall \sigma.\, \mathrm{red}(e, \sigma) \wedge & \text{if } e \notin \mathit{Val} \\ \quad \triangleright (\forall e_2, \sigma_2.\, (e, \sigma) \to_{\mathsf{t}} (e_2, \sigma_2) \twoheadrightarrow & \\ \qquad \mathsf{wp}\, e_2\, \{w.\, Q\}) & \end{cases}$$

Recursive occurrence guarded by a *later* $\triangleright$

Adding the points-to connective

How to connect the states σ to $\ell \mapsto v$?

$$\mathsf{wp}\ e\ \{w.\ Q\} \triangleq \begin{cases} Q[e/w] & \text{if } e \in \mathit{Val} \\ \forall \sigma. & \text{if } e \notin \mathit{Val} \\ \quad \mathrm{red}(e, \sigma) \wedge & \\ \quad \triangleright (\forall e_2, \sigma_2.\ (e, \sigma) \to_{\mathrm{t}} (e_2, \sigma_2) \mathrel{-\!\!*} & \\ \qquad \mathsf{wp}\ e_2\ \{w.\ Q\}) & \end{cases}$$

$$\ell \mapsto v \triangleq ???$$

Adding the points-to connective

How to connect the states σ to $\ell \mapsto v$?

$$\mathsf{wp}\ e\ \{w.\ Q\} \triangleq \begin{cases} Q[e/w] & \text{if } e \in \mathit{Val} \\ \forall \sigma. & \text{if } e \notin \mathit{Val} \\ \qquad \mathrm{red}(e, \sigma) \land & \\ \qquad \triangleright (\forall e_2, \sigma_2.\ (e, \sigma) \to_{\mathsf{t}} (e_2, \sigma_2) -\!\!* & \\ \qquad\qquad\qquad\qquad \mathsf{wp}\ e_2\ \{w.\ Q\}) & \end{cases}$$

$$\ell \mapsto v \triangleq ???$$

Solution: ghost variables

Adding the points-to connective

How to connect the states σ to $\ell \mapsto v$?

$$\mathsf{wp}\ e\ \{w.\ Q\} \triangleq \begin{cases} \Rrightarrow Q[e/w] & \text{if } e \in \mathit{Val} \\ \forall \sigma.\ \gamma \hookrightarrow_\bullet \sigma \twoheadrightarrow \Rrightarrow & \text{if } e \notin \mathit{Val} \\ \qquad \mathrm{red}(e, \sigma) \land & \\ \qquad \triangleright (\forall e_2, \sigma_2.\ (e, \sigma) \to_{\mathsf{t}} (e_2, \sigma_2) \twoheadrightarrow \Rrightarrow & \\ \qquad\qquad \gamma \hookrightarrow_\bullet \sigma_2 * \mathsf{wp}\ e_2\ \{w.\ Q\}) & \end{cases}$$

$$\ell \mapsto v \triangleq \gamma \hookrightarrow_\circ [\ell := v]$$

Solution: ghost variables

Using an appropriate resource algebra (RA) we can obtain:

$$\gamma \hookrightarrow_\bullet \sigma * \gamma \hookrightarrow_\circ [\ell := w] \Rightarrow \sigma(\ell) = w$$

$$\gamma \hookrightarrow_\bullet \sigma * \gamma \hookrightarrow_\circ [\ell := v] \Rightarrow \Rrightarrow(\gamma \hookrightarrow_\bullet \sigma[\ell := w] * \gamma \hookrightarrow_\circ [\ell := w])$$

$$\gamma \hookrightarrow_\bullet \sigma \Rightarrow \Rrightarrow(\gamma \hookrightarrow_\bullet \sigma[\ell := w] * \gamma \hookrightarrow_\circ [\ell := w]) \quad \text{if } \ell \notin \mathrm{dom}(\sigma)$$

Adding fork

$$\mathsf{wp}\ e\ \{w.\ Q\} \triangleq \begin{cases} \Rrightarrow Q[e/w] & \text{if } e \in \mathit{Val} \\ \forall \sigma.\ \gamma \hookrightarrow_{\bullet} \sigma \twoheadrightarrow\ \Rrightarrow & \text{if } e \notin \mathit{Val} \\ \qquad \mathrm{red}(e, \sigma) \wedge & \\ \qquad \triangleright (\forall e_2, \sigma_2, \vec{e}_f.\ (e, \sigma) \to_{\mathrm{t}} (e_2, \sigma_2, \vec{e}_f) \twoheadrightarrow\ \Rrightarrow & \\ \qquad\qquad \gamma \hookrightarrow_{\bullet} \sigma_2 * \mathsf{wp}\ e_2\ \{w.\ Q\} * & \\ \qquad\qquad \mathop{\Huge *}_{e' \in \vec{e}_f} \mathsf{wp}\ e'\ \{w.\mathsf{True}\}) & \end{cases}$$

$$\ell \mapsto v \triangleq \gamma \hookrightarrow_{\circ} [\ell := v]$$

Key point: separating conjunction ensures that the resources of the new threads $\vec{e}_f$ are disjointly subdivided between threads

Defining invariants

Key idea: thread the *global invariant* W through weakest preconditions:

$$W \triangleq \exists I : \mathit{InvName} \xrightharpoonup{\text{fin}} \mathit{iProp}.\, \gamma_{\text{INV}} \hookrightarrow_{\bullet} I \ * \ \mathop{\Large *}_{\iota \in \text{dom}(I) \text{ and } \iota \text{ is not opened}} \triangleright I(\iota)$$

The invariant assertion can be defined as:

$$\boxed{P}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}.\, \gamma_{\text{INV}} \hookrightarrow_{\circ} [\iota := P]$$

Defining invariants

Key idea: thread the *global invariant* W through weakest preconditions:

$$W \triangleq \exists I : \mathit{InvName} \xrightarrow{\text{fin}} \mathit{iProp}.\, \gamma_{\mathrm{INV}} \hookrightarrow_{\bullet} I \;\; * \;\; \mathop{\Large *}_{\iota \in \mathrm{dom}(I) \text{ and } \iota \text{ is not opened}} \triangleright I(\iota)$$

The invariant assertion can be defined as:

$$\boxed{P}^{\mathcal{N}} \triangleq \exists \iota \in \mathcal{N}.\, \gamma_{\mathrm{INV}} \hookrightarrow_{\circ} [\iota := P]$$

Interesting points:

- Need to define a small protocol mechanism to keep track of whether invariants have been opened or not
- Ghost state $\gamma_{\mathrm{INV}} \hookrightarrow_{\bullet} I$ and $\gamma_{\mathrm{INV}} \hookrightarrow_{\circ} [\iota := P]$ involve ownership of Iris propositions, need higher-order ghost state

In the paper and the Coq formalization

- Full definition of invariants $\boxed{P}^{\mathcal{N}}$
- All about the modalities $\square$, $\triangleright$ and $\mathrel{\vert\!\!\Rrightarrow}$
- Adequacy of weakest preconditions
- Paradox showing that $\triangleright$ is ‘needed’ for impredicative invariants

The Essence of
Higher-Order Concurrent Separation Logic

Robbert Krebbers[1], Ralf Jung[2], Aleš Bizjak[3],
Jacques-Henri Jourdan[2], Derek Dreyer[2], and Lars Birkedal[3]

[1] Delft University of Technology, The Netherlands
[2] Max Planck Institute for Software Systems (MPI-SWS), Germany
[3] Aarhus University, Denmark

Abstract. Concurrent separation logics (CSLs) have come of age, and with age they have accumulated a great deal of complexity. Previous work on the Iris logic attempted to reduce the complex logical mechanisms of modern CSLs to two orthogonal concepts: partial commutative

Thank you!

Take home messages:

- Hoare triples and invariants can be *defined* in the assertion logic of separation logic (+ some modalities)
- Iris is no longer just a program logic, but also a framework to define concurrent program logics
- Iris is implemented in Coq and used for many purposes (modeling type systems, logical relations, program logics)

Download Iris at `http://iris-project.org/`