Iris: a framework for higher-order concurrent separation logic in Coq

Robbert Krebbers[1]

Delft University of Technology, The Netherlands

January 15, 2017 @ TTT, Paris, France

[1] Iris is joint work with: Ralf Jung, Jacques-Hendri Jourdan, Aleš Bizjak, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Amin Timany, Derek Dreyer, and Lars Birkedal

What is Iris?

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

What is Iris?

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

What is Iris?

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

What is Iris?

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

What is Iris?

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

What is Iris?

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

The versatility of Iris

The scope of Iris goes beyond proving traditional program correctness using Hoare triples:

- The Rust type system (Jung, Jourdan, Dreyer, Krebbers)
- Logical relations (Krogh-Jespersen, Svendsen, Timany, Birkedal, Tassarotti, Jung, Krebbers)
- Weak memory concurrency (Kaiser, Dang, Dreyer, Lahav, Vafeiadis)
- Object calculi (Swasey, Dreyer, Garg)
- Logical atomicity (Krogh-Jespersen, Zhang, Jung)
- Defining Iris (Krebbers, Jung, Jourdan, Bizjak, Dreyer, Birkedal)

Most of these projects are formalized in Iris in Coq

This talk

Show that ideas from concurrent separation logic can be used to encode concurrent separation logic itself

Why?

- Smaller *base* logic
- Notions like Hoare triples $\{P\}\, e\, \{Q\}$ are not primitives, but:
  - are defined in the logic
  - at a higher level of abstraction
  - resulting in simpler proofs
- Gives a better intellectual understanding
- Eases the formalization in Coq

Preview of the rules of the Iris base logic

Laws of (affine) bunched implications

$$\begin{array}{rcl} \mathsf{True} * P & \dashv\vdash & P \\ P * Q & \vdash & Q * P \\ (P * Q) * R & \vdash & P * (Q * R) \end{array} \qquad \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R} \qquad \frac{P \vdash Q \mathrel{-\!\!*} R}{P * Q \vdash R}$$

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

$$\frac{P \vdash Q}{\mathrel{|\!\!\Rrightarrow} P \vdash \mathrel{|\!\!\Rrightarrow} Q} \qquad P \vdash \mathrel{|\!\!\Rrightarrow} P \qquad \mathrel{|\!\!\Rrightarrow}\mathrel{|\!\!\Rrightarrow} P \vdash \mathrel{|\!\!\Rrightarrow} P$$

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

Laws for the always modality

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

Laws for the later modality

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

Laws for timeless assertions

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

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

Hoare triples

Hoare triples for partial program correctness:

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

If the initial state satisfies P, then:

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

Separation logic [O'Hearn, Reynolds, Yang]

The points-to connective $x \mapsto v$

- provides the knowledge that location x has value v, and
- provides exclusive ownership of x

Separating conjunction $P * Q$: the state consists of *disjoint parts* satisfying P and Q

Separation logic [O'Hearn, Reynolds, Yang]

The points-to connective $x \mapsto v$

- provides the knowledge that location x has value v, and
- provides exclusive ownership of x

Separating conjunction $P * Q$: the state consists of *disjoint parts* satisfying P and Q

Example:

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

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:

$$\begin{array}{c}
\{x \mapsto 4 * y \mapsto 6\} \\
x := !\,x + 2 \;\left|\left|\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right.\right.\; y := !\,y + 2 \\
\{x \mapsto 6 * y \mapsto 8\}
\end{array}$$

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:

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

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:

$$
\begin{array}{l}
\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in} \\[1em]
\texttt{fetchandadd}(x, 2) \;\Big\|\; \texttt{fetchandadd}(x, 2) \\[1em]
!\, x
\end{array}
$$

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

What about shared state/racy programs?

A classic problem:

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

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

What about shared state/racy programs?

A classic problem:

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

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

What about shared state/racy programs?

A classic problem:

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

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

Problem: can only give ownership of x to one thread

Invariants

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

Invariants

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

Invariant opening:

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

Invariants

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

Invariant opening:

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

Invariant allocation:

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

Invariants

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

Invariant opening:

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

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

Invariants

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

Invariant opening:

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

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:

$\{\mathsf{True}\}$

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

$$\texttt{fetchandadd}(x, 2) \quad \Bigg\| \quad \texttt{fetchandadd}(x, 2)$$

$!\,x$

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

Invariants in action

Let us consider a simpler problem first:

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

$$\texttt{fetchandadd}(x, 2) \quad \Big\| \quad \texttt{fetchandadd}(x, 2)$$

$!\, x$

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

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

$\texttt{fetchandadd}(x, 2) \quad \Big\| \quad \texttt{fetchandadd}(x, 2)$

$!\, x$

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

Invariants in action

Let us consider a simpler problem first:

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

Invariants in action

Let us consider a simpler problem first:

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

Invariants in action

Let us consider a simpler problem first:

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

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

$\quad !\, x$

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

Invariants in action

Let us consider a simpler problem first:

$$\begin{array}{l}
\{\mathsf{True}\}\\
\texttt{let}\, x = \texttt{ref}(0)\ \texttt{in}\\
\{x \mapsto 0\}\\
\text{allocate}\ \boxed{\exists n.\, x \mapsto n \land \mathit{even}(n)}\\
\begin{array}[t]{l||l}
\begin{array}[t]{l}
\{\mathsf{True}\}\\
\quad\begin{array}{|l}
\{x \mapsto n \land \mathit{even}(n)\}\\
\texttt{fetchandadd}(x, 2)\\
\{x \mapsto n + 2 \land \mathit{even}(n + 2)\}
\end{array}\\
\{\mathsf{True}\}\\
\quad\begin{array}{|l}
\{x \mapsto n \land \mathit{even}(n)\}\\
!\, x\\
\{n.\, x \mapsto n \land \mathit{even}(n)\}
\end{array}\\
\{n.\, \mathit{even}(n)\}
\end{array}
&
\begin{array}[t]{l}
\{\mathsf{True}\}\\
\quad\begin{array}{|l}
\{x \mapsto n \land \mathit{even}(n)\}\\
\texttt{fetchandadd}(x, 2)\\
\{x \mapsto n + 2 \land \mathit{even}(n + 2)\}
\end{array}\\
\{\mathsf{True}\}
\end{array}
\end{array}
\end{array}$$

Invariants in action

Let us consider a simpler problem first:

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

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

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:

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

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:

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

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:

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

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

$$\gamma \hookrightarrow_\bullet n * \gamma \hookrightarrow_\circ m \quad \Rightarrow \quad n = m$$
$$\gamma \hookrightarrow_\bullet n * \gamma \hookrightarrow_\circ m \quad \mathrel{\Rrightarrow\!\!\!\!\ast} \quad \gamma \hookrightarrow_\bullet n' * \gamma \hookrightarrow_\circ n'$$

Ghost variables in action

$\{\mathsf{True}\}$

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

$$\texttt{fetchandadd}(x, 2) \quad \Big\Vert \quad \texttt{fetchandadd}(x, 2)$$

$!\, x$

$\{n.\; n = 4\}$

Ghost variables in action

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

$\texttt{fetchandadd}(x, 2) \quad \Big\| \quad \texttt{fetchandadd}(x, 2)$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\texttt{fetchandadd}(x, 2) \parallel \texttt{fetchandadd}(x, 2)$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\texttt{fetchandadd}(x, 2) \quad \Big\| \quad \texttt{fetchandadd}(x, 2)$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_1 \hookrightarrow_{\circ} 0 * \gamma_2 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\circ} 0\}$

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

$\{\gamma_1 \hookrightarrow_{\circ} 0 * \gamma_2 \hookrightarrow_{\circ} 0\}$

$\texttt{fetchandadd}(x, 2) \quad \parallel \quad \texttt{fetchandadd}(x, 2)$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\} \quad \Big\| \quad \{\gamma_2 \hookrightarrow_\circ 0\}$

$\texttt{fetchandadd}(x, 2) \quad \Big\| \quad \texttt{fetchandadd}(x, 2)$

$!\,x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$ $\Big\|$ $\{\gamma_2 \hookrightarrow_\circ 0\}$

$\texttt{fetchandadd}(x, 2)$ $\Big\|$ $\texttt{fetchandadd}(x, 2)$

$\{\gamma_1 \hookrightarrow_\circ 2\}$ $\Big\|$ $\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\,x$

$\{n.\ n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\} \quad \| \quad \{\gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\texttt{fetchandadd}(x, 2) \quad \| \quad \texttt{fetchandadd}(x, 2)$

$\{\gamma_1 \hookrightarrow_\circ 2\} \quad \| \quad \{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\,x$

$\{n.\ n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

`let` $x =$ `ref`(0) `in`

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

`fetchandadd`$(x, 2)$

$\{\gamma_1 \hookrightarrow_\circ 2\}$

$\|$

$\{\gamma_2 \hookrightarrow_\circ 0\}$

`fetchandadd`$(x, 2)$

$\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\,x$

$\{n.\ n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\mathtt{fetchandadd}(x, 2)$

$\{\gamma_1 \hookrightarrow_\circ 2\}$

$\|$

$\{\gamma_2 \hookrightarrow_\circ 0\}$

$\mathtt{fetchandadd}(x, 2)$

$\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\,x$

$\{n.\ n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$ $\parallel$ $\{\gamma_2 \hookrightarrow_\circ 0\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\quad\texttt{fetchandadd}(x, 2)$ $\parallel$ $\texttt{fetchandadd}(x, 2)$

$\{\gamma_1 \hookrightarrow_\circ 2\}$ $\parallel$ $\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$\quad !\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$$\begin{array}{l||l}
\{\gamma_1 \hookrightarrow_\circ 0\} & \{\gamma_2 \hookrightarrow_\circ 0\} \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\quad \texttt{fetchandadd}(x, 2) & \quad \texttt{fetchandadd}(x, 2) \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\{\gamma_1 \hookrightarrow_\circ 2\} & \{\gamma_2 \hookrightarrow_\circ 2\}
\end{array}$$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$	$\Vert$	$\{\gamma_2 \hookrightarrow_\circ 0\}$
$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1+n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$		
$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$		
$\texttt{fetchandadd}(x, 2)$		$\texttt{fetchandadd}(x, 2)$
$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2+n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$		
$\{\gamma_1 \hookrightarrow_\circ 2\}$		$\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$$\begin{array}{l||l}
\{\gamma_1 \hookrightarrow_\circ 0\} & \{\gamma_2 \hookrightarrow_\circ 0\} \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\quad \texttt{fetchandadd}(x, 2) & \quad \texttt{fetchandadd}(x, 2) \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\quad \{\gamma_1 \hookrightarrow_\circ 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\{\gamma_1 \hookrightarrow_\circ 2\} & \{\gamma_2 \hookrightarrow_\circ 2\}
\end{array}$$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\,x$

$\{n.\, n = 4\}$

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\texttt{fetchandadd}(x, 2)$

$\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\{\gamma_1 \hookrightarrow_\circ 2\}$

$\|$

$\{\gamma_2 \hookrightarrow_\circ 0\}$

$\{\ldots\}$

$\texttt{fetchandadd}(x, 2)$

$\{\ldots\}$

$\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$!\,x$

$\{n.\ n = 4\}$

Ghost variables in action

$\{\text{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_1 \hookrightarrow_{\circ} 0 * \gamma_2 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\circ} 0\}$

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

$\{\gamma_1 \hookrightarrow_{\circ} 0 * \gamma_2 \hookrightarrow_{\circ} 0\}$

$\{\gamma_1 \hookrightarrow_{\circ} 0\}$

$\{\gamma_1 \hookrightarrow_{\circ} 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_{\bullet} n_1 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$

$\{\gamma_1 \hookrightarrow_{\circ} 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$

$\texttt{fetchandadd}(x, 2)$

$\{\gamma_1 \hookrightarrow_{\circ} 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$

$\{\gamma_1 \hookrightarrow_{\circ} 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_{\bullet} 2 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$

$\{\gamma_1 \hookrightarrow_{\circ} 2\}$

$\|$

$\{\gamma_2 \hookrightarrow_{\circ} 0\}$

$\{\ldots\}$

$\texttt{fetchandadd}(x, 2)$

$\{\ldots\}$

$\{\gamma_2 \hookrightarrow_{\circ} 2\}$

$\{\gamma_1 \hookrightarrow_{\circ} 2 * \gamma_2 \hookrightarrow_{\circ} 2\}$

$\{\gamma_1 \hookrightarrow_{\circ} 2 * \gamma_2 \hookrightarrow_{\circ} 2 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_{\bullet} n_1 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$

$!\, x$

$\{n.\, n = 4\}$

Ghost variables in action

```
$\{\mathsf{True}\}$
$\texttt{let}\ x = \texttt{ref}(0)\ \texttt{in}$
$\{x \mapsto 0\}$
$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$
allocate $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$
$\{\gamma_1 \hookrightarrow_\circ 0\}$
    $\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$
    $\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$
    $\texttt{fetchandadd}(x, 2)$
    $\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$
    $\{\gamma_1 \hookrightarrow_\circ 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet n_2\}$
$\{\gamma_1 \hookrightarrow_\circ 2\}$
||
$\{\gamma_2 \hookrightarrow_\circ 0\}$
    $\{\ldots\}$
    $\texttt{fetchandadd}(x, 2)$
    $\{\ldots\}$
$\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$
    $\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

    $!\, x$

$\{n.\ n = 4\}$
```

Ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

$\{\gamma_1 \hookrightarrow_\circ 0\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\quad\texttt{fetchandadd}(x, 2)$

$\quad\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\{\gamma_1 \hookrightarrow_\circ 2\}$

$\|$

$\{\gamma_2 \hookrightarrow_\circ 0\}$

$\quad\{\ldots\}$

$\quad\texttt{fetchandadd}(x, 2)$

$\quad\{\ldots\}$

$\{\gamma_2 \hookrightarrow_\circ 2\}$

$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$

$\quad\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto 4 * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet 2\}$

$\quad !\,x$

$\{n.\, n = 4\}$

Ghost variables in action

```
$\{\mathsf{True}\}$
let $x = $ ref$(0)$ in
$\{x \mapsto 0\}$
$\{x \mapsto 0 * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_1 \hookrightarrow_{\circ} 0 * \gamma_2 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\circ} 0\}$
allocate $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_{\bullet} n_1 * \gamma_2 \hookrightarrow_{\bullet} n_2}$
$\{\gamma_1 \hookrightarrow_{\circ} 0 * \gamma_2 \hookrightarrow_{\circ} 0\}$
$\{\gamma_1 \hookrightarrow_{\circ} 0\}$
  $\{\gamma_1 \hookrightarrow_{\circ} 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_{\bullet} n_1 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$
  $\{\gamma_1 \hookrightarrow_{\circ} 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$
  fetchandadd$(x, 2)$
  $\{\gamma_1 \hookrightarrow_{\circ} 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_{\bullet} 0 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$
  $\{\gamma_1 \hookrightarrow_{\circ} 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_{\bullet} 2 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$
$\{\gamma_1 \hookrightarrow_{\circ} 2\}$
$\|$
$\{\gamma_2 \hookrightarrow_{\circ} 0\}$
  $\{\ldots\}$
  fetchandadd$(x, 2)$
  $\{\ldots\}$
$\{\gamma_2 \hookrightarrow_{\circ} 2\}$

$\{\gamma_1 \hookrightarrow_{\circ} 2 * \gamma_2 \hookrightarrow_{\circ} 2\}$
  $\{\gamma_1 \hookrightarrow_{\circ} 2 * \gamma_2 \hookrightarrow_{\circ} 2 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_{\bullet} n_1 * \gamma_2 \hookrightarrow_{\bullet} n_2\}$
  $\{\gamma_1 \hookrightarrow_{\circ} 2 * \gamma_2 \hookrightarrow_{\circ} 2 * x \mapsto 4 * \gamma_1 \hookrightarrow_{\bullet} 2 * \gamma_2 \hookrightarrow_{\bullet} 2\}$
  $!\, x$

$\{n.\, n = 4\}$
```

Ghost variables in action

$$\{\mathsf{True}\}$$

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

$$\{x \mapsto 0\}$$

$$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$$

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

$$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$$

$$\begin{array}{l||l}
\{\gamma_1 \hookrightarrow_\circ 0\} & \{\gamma_2 \hookrightarrow_\circ 0\} \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\} & \quad \{\ldots\} \\
\quad \texttt{fetchandadd}(x, 2) & \quad \texttt{fetchandadd}(x, 2) \\
\quad \{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\} & \quad \{\ldots\} \\
\quad \{\gamma_1 \hookrightarrow_\circ 2 * x \mapsto (2 + n_2) * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet n_2\} & \\
\{\gamma_1 \hookrightarrow_\circ 2\} & \{\gamma_2 \hookrightarrow_\circ 2\}
\end{array}$$

$$\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}$$

$$\quad \{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto (n_1 + n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}$$

$$\quad \{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto 4 * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet 2\}$$

$$\quad !\, x$$

$$\quad \{n.\, n = 4 \wedge \gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto 4 * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet 2\}$$

$$\{n.\, n = 4\}$$

Ghost variables in action

$$\begin{array}{l}
\{\mathsf{True}\}\\
\texttt{let}\, x = \texttt{ref}(0)\,\texttt{in}\\
\{x \mapsto 0\}\\
\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}\\
\mathsf{allocate}\ \boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}\\
\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}\\
\begin{array}{l||l}
\{\gamma_1 \hookrightarrow_\circ 0\} & \{\gamma_2 \hookrightarrow_\circ 0\}\\
\left|\begin{array}{l}
\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (n_1{+}n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}\\
\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto n_2 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}\\
\texttt{fetchandadd}(x, 2)\\
\{\gamma_1 \hookrightarrow_\circ 0 * x \mapsto (2{+}n_2) * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\bullet n_2\}\\
\{\gamma_1 \hookrightarrow_\circ 2 * x \mapsto (2{+}n_2) * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet n_2\}
\end{array}\right. &
\left|\begin{array}{l}
\{\ldots\}\\
\texttt{fetchandadd}(x, 2)\\
\{\ldots\}
\end{array}\right.\\
\{\gamma_1 \hookrightarrow_\circ 2\} & \{\gamma_2 \hookrightarrow_\circ 2\}
\end{array}\\
\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2\}\\
\left|\begin{array}{l}
\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto (n_1{+}n_2) * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2\}\\
\{\gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto 4 * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet 2\}\\
!\, x\\
\{n.\, n = 4 \wedge \gamma_1 \hookrightarrow_\circ 2 * \gamma_2 \hookrightarrow_\circ 2 * x \mapsto 4 * \gamma_1 \hookrightarrow_\bullet 2 * \gamma_2 \hookrightarrow_\bullet 2\}
\end{array}\right.\\
\{n.\, n = 4\}
\end{array}$$

Ghost variables with fractional permissions [Boyland]

What if we have *n* threads? Using *n* different ghost variables, results in different proofs for each thread. *That is not modular.*

Better way: ghost variables with a *fractional permission* $(0,1]_{\mathbb{Q}}$:

$$\gamma \overset{\pi_1+\pi_2}{\hookrightarrow_\circ} (n_1 + n_2) \quad \Leftrightarrow \quad \gamma \overset{\pi_1}{\hookrightarrow_\circ} n_1 * \gamma \overset{\pi_2}{\hookrightarrow_\circ} n_2$$

Ghost variables with fractional permissions [Boyland]

What if we have n threads? Using n different ghost variables, results in different proofs for each thread. *That is not modular.*

Better way: ghost variables with a *fractional permission* $(0,1]_{\mathbb{Q}}$:

$$\gamma \xhookrightarrow{\pi_1+\pi_2}_{\circ} (n_1 + n_2) \quad\Leftrightarrow\quad \gamma \xhookrightarrow{\pi_1}_{\circ} n_1 * \gamma \xhookrightarrow{\pi_2}_{\circ} n_2$$

You only get the equality when you have *full ownership* $(\pi = 1)$:

$$\gamma \hookrightarrow_{\bullet} n * \gamma \xhookrightarrow{1}_{\circ} m \quad\Rightarrow\quad n = m$$

Ghost variables with fractional permissions [Boyland]

What if we have n threads? Using n different ghost variables, results in different proofs for each thread. *That is not modular.*

Better way: ghost variables with a *fractional permission* $(0,1]_{\mathbb{Q}}$:

$$\gamma \xhookrightarrow{\pi_1+\pi_2}_{\circ} (n_1 + n_2) \quad \Leftrightarrow \quad \gamma \xhookrightarrow{\pi_1}_{\circ} n_1 * \gamma \xhookrightarrow{\pi_2}_{\circ} n_2$$

You only get the equality when you have *full ownership* $(\pi = 1)$:

$$\gamma \hookrightarrow_{\bullet} n * \gamma \xhookrightarrow{1}_{\circ} m \quad \Rightarrow \quad n = m$$

Updating is possible with *partial ownership* $(0 < \pi \leq 1)$:

$$\gamma \hookrightarrow_{\bullet} n * \gamma \xhookrightarrow{\pi}_{\circ} m \quad \mathrel{\Rrightarrow\!\!\!*} \quad \gamma \hookrightarrow_{\bullet} (n + i) * \gamma \xhookrightarrow{\pi}_{\circ} (m + i)$$

Ghost variables with fractional permissions [Boyland]

What if we have n threads? Using n different ghost variables, results in different proofs for each thread. *That is not modular.*

Better way: ghost variables with a *fractional permission* $(0,1]_{\mathbb{Q}}$:

$$\gamma \xhookrightarrow{\pi_1+\pi_2}_{\circ} (n_1 + n_2) \quad \Leftrightarrow \quad \gamma \xhookrightarrow{\pi_1}_{\circ} n_1 * \gamma \xhookrightarrow{\pi_2}_{\circ} n_2$$

You only get the equality when you have *full ownership* $(\pi = 1)$:

$$\gamma \hookrightarrow_{\bullet} n * \gamma \xhookrightarrow{1}_{\circ} m \quad \Rightarrow \quad n = m$$

Updating is possible with *partial ownership* $(0 < \pi \leq 1)$:

$$\gamma \hookrightarrow_{\bullet} n * \gamma \xhookrightarrow{\pi}_{\circ} m \quad \mathrel{\Rrightarrow\!\!\!*} \quad \gamma \hookrightarrow_{\bullet} (n + i) * \gamma \xhookrightarrow{\pi}_{\circ} (m + i)$$

Keeps the invariant that all $\gamma \xhookrightarrow{\pi_i}_{\circ} n_i$ sum up to $\gamma \hookrightarrow_{\bullet} \sum n_i$

Fractional ghost variables in action

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

$$\texttt{fetchandadd}(x, 2) \quad \Bigg\| \quad \texttt{fetchandadd}(x, 2) \quad \Bigg\| \quad \ldots$$

$!\,x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

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

$$\texttt{fetchandadd}(x, 2) \quad \Bigg\| \quad \texttt{fetchandadd}(x, 2) \quad \Bigg\| \quad \ldots$$

$!\, x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\text{True}\}$

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

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

$\texttt{fetchandadd}(x, 2) \;\Big\|\; \texttt{fetchandadd}(x, 2) \;\Big\|\; \dots$

$!\, x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\text{True}\}$
$\texttt{let}\; x = \texttt{ref}(0)\; \texttt{in}$
$\{x \mapsto 0\}$
$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$
allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$\texttt{fetchandadd}(x, 2) \quad \Big\| \quad \texttt{fetchandadd}(x, 2) \quad \Big\| \quad \ldots$

$!\, x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$$\begin{array}{l|l|l}
\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\} & \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\} & \\
\quad \texttt{fetchandadd}(x, 2) & \quad \texttt{fetchandadd}(x, 2) & \ldots \\
\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\} & \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\} &
\end{array}$$

$\quad !\, x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\mathsf{True}\}$

`let` $x =$ `ref`(0) `in`

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$

`fetchandadd`$(x, 2)$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\|$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$

`fetchandadd`$(x, 2)$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\|$ . . .

$!\,x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$$\begin{array}{l||l||l}
\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\} & \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\} & \\
\quad \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\} & & \\
\quad \texttt{fetchandadd}(x, 2) & \texttt{fetchandadd}(x, 2) & \ldots \\
\quad \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2 * x \mapsto (2+n) * \gamma_1 \hookrightarrow_{\bullet} (2+n)\right\} & & \\
\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\} & \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\} &
\end{array}$$

$!\, x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$$
\begin{array}{l|l|l}
\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\} & \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\} & \\
\quad \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\} & \quad \{\ldots\} & \\
\quad \texttt{fetchandadd}(x, 2) & \quad \texttt{fetchandadd}(x, 2) & \ \ldots \\
\quad \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2 * x \mapsto (2{+}n) * \gamma_1 \hookrightarrow_{\bullet} (2{+}n)\right\} & \quad \{\ldots\} & \\
\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\} & \left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\} &
\end{array}
$$

$!\, x$

$\{n.\, n = 2k\}$

Fractional ghost variables in action

$\{\text{True}\}$

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

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$

$\texttt{fetchandadd}(x, 2)$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2 * x \mapsto (2+n) * \gamma_1 \hookrightarrow_{\bullet} (2+n)\right\}$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\|$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$

$\{\dots\}$

$\texttt{fetchandadd}(x, 2)$

$\{\dots\}$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\| \quad \dots$

$\left\{\gamma \overset{1}{\hookrightarrow}_{\circ} 2k * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$

$!\, x$

$\{n.\; n = 2k\}$

Fractional ghost variables in action

$\{\text{True}\}$

`let` $x =$ `ref`(0) `in`

$\{x \mapsto 0\}$

$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \xhookrightarrow{1}_{\circ} 0\right\}$

allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_{\bullet} n}$

$\left\{\gamma \xhookrightarrow{1/k}_{\circ} 0\right\}$

$\left\{\gamma \xhookrightarrow{1/k}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$

`fetchandadd`$(x, 2)$

$\left\{\gamma \xhookrightarrow{1/k}_{\circ} 2 * x \mapsto (2+n) * \gamma_1 \hookrightarrow_{\bullet} (2+n)\right\}$

$\left\{\gamma \xhookrightarrow{1/k}_{\circ} 2\right\}$

$\|$

$\left\{\gamma \xhookrightarrow{1/k}_{\circ} 0\right\}$

$\{\ldots\}$

`fetchandadd`$(x, 2)$

$\{\ldots\}$

$\left\{\gamma \xhookrightarrow{1/k}_{\circ} 2\right\}$

$\| \quad \ldots$

$\left\{\gamma \xhookrightarrow{1}_{\circ} 2k * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$

$!\,x$

$\left\{n.\, n = 2k \wedge \gamma \xhookrightarrow{1}_{\circ} 2k * x \mapsto 2k * \gamma \hookrightarrow_{\bullet} 2k\right\}$

$\{n.\, n = 2k\}$

Part #2: generalizing ownership

[Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal and Derek Dreyer. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In POPL'15]

[Ralf Jung, Robbert Krebbers, Lars Birkedal and Derek Dreyer. Higher-Order Ghost State. In ICFP'16]

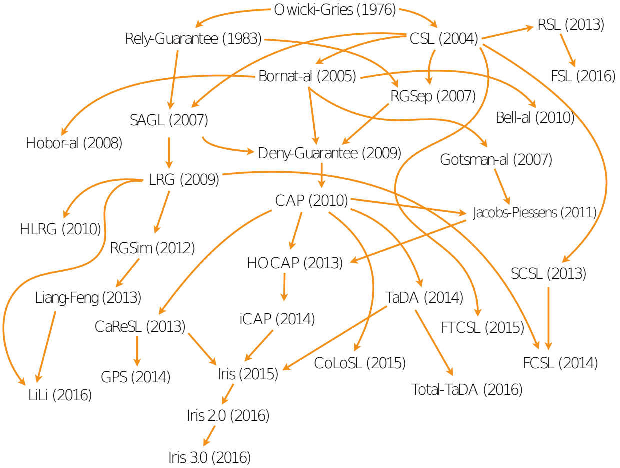
Mechanisms for concurrent reasoning

We have see so far:

- Invariants $\boxed{R}^{\mathcal{N}}$
- Ghost variables $\gamma \hookrightarrow_{\bullet} n$ and $\gamma \hookrightarrow_{\circ} n$
- Fractional ghost variables $\gamma \hookrightarrow_{\bullet} n$ and $\gamma \overset{\pi}{\hookrightarrow}_{\circ} n$

Where do these mechanisms come from?

There are many CSLs with more powerful mechanisms. . .



Picture by Ilya Sergey

. . . and very complicated **primitive** rules

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

$$\dfrac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq}_{\pi} b_0.\ \llparenthesis \pi\llbracket b \rrbracket * P \rrparenthesis\ i \Mapsto_{\mathrm{I}} a\ \llparenthesis x.\ \exists b' \overset{\text{guar}}{\sqsupseteq}_{\pi} b.\ \pi\llbracket b' \rrbracket * Q \rrparenthesis}{\mathcal{C} \vdash \left\{ \boxed{b_0}^{n}_{\pi} * \triangleright P \right\}\ i \Mapsto a\ \left\{ x.\ \exists b'.\ \boxed{b'}^{n}_{\pi} * Q \right\}}\ \textsc{UpdIsl}$$

Use atomic rule

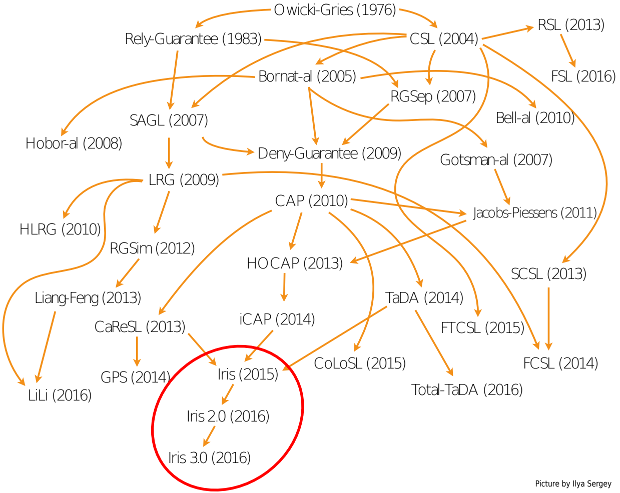
$$\dfrac{\begin{array}{c}a \notin \mathcal{A} \quad \forall x \in X.\, (x, f(x)) \in \mathcal{T}_{\mathbf{t}}(\mathrm{G})^{*} \\ \lambda; \mathcal{A} \vdash \boldsymbol{\forall} x \in X.\, \langle p_p \mid I(\mathbf{t}^{\lambda}_{a}(x)) * p(x) * [\mathrm{G}]_a \rangle\ \mathbb{C}\ \ \boldsymbol{\exists}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

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

Update region rule

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

The Iris story



Picture by Ilya Sergey

The Iris story: all of these mechanisms can be encoded using a simple mechanism of *resource ownership*

Generalizing ownership

All forms of ownership have common properties:

- Ownership of different threads can be composed
  For example:

$$\gamma \xhookrightarrow{\pi_1+\pi_2}_{\circ} (n_1 + n_2) \quad \Leftrightarrow \quad \gamma \xhookrightarrow{\pi_1}_{\circ} n_1 * \gamma \xhookrightarrow{\pi_2}_{\circ} n_2$$

Generalizing ownership

All forms of ownership have common properties:

- Ownership of different threads can be composed
  For example:

$$\gamma \xhookrightarrow{\pi_1+\pi_2}_{\circ} (n_1 + n_2) \quad \Leftrightarrow \quad \gamma \xhookrightarrow{\pi_1}_{\circ} n_1 * \gamma \xhookrightarrow{\pi_2}_{\circ} n_2$$

- Composition of ownership is associative and commutative
  Mirroring that parallel composition and separating conjunction is associative and commutative

Generalizing ownership

All forms of ownership have common properties:

- Ownership of different threads can be composed
  For example:

$$\gamma \xhookrightarrow{\pi_1+\pi_2}_{\circ} (n_1 + n_2) \quad \Leftrightarrow \quad \gamma \xhookrightarrow{\pi_1}_{\circ} n_1 * \gamma \xhookrightarrow{\pi_2}_{\circ} n_2$$

- Composition of ownership is associative and commutative
  Mirroring that parallel composition and separating conjunction is associative and commutative
- Combinations of ownership that do not make sense are ruled out
  For example:

$$\gamma \hookrightarrow_{\bullet} 5 * \gamma \xhookrightarrow{1/2}_{\circ} 3 * \gamma \xhookrightarrow{1/2}_{\circ} 4 \quad \Rightarrow \quad \text{False}$$

(because $5 \neq 3 + 4$)

Resource algebras

Resource algebra with carrier M:

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

Satisfying:

$$a \cdot b = b \cdot a \qquad a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad (a \cdot b) \in \mathcal{V} \Rightarrow a \in \mathcal{V}$$

Resource algebras

Resource algebra with carrier M:

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

Satisfying:

$$a \cdot b = b \cdot a \qquad a \cdot (b \cdot c) = (a \cdot b) \cdot c \qquad (a \cdot b) \in \mathcal{V} \Rightarrow a \in \mathcal{V}$$

Iris has ghost variables $\boxed{a : M}^{\gamma}$ for each resource algebra M

$$a \in \mathcal{V} \Rrightarrow\!\!\!\ast\ \exists \gamma.\, \boxed{a}^{\gamma} \qquad \boxed{a}^{\gamma} * \boxed{b}^{\gamma} \Leftrightarrow \boxed{a \cdot b}^{\gamma} \qquad \boxed{a}^{\gamma} \Rightarrow \mathcal{V}(a)$$

$$\frac{\forall a_{\mathrm{f}}.\, a \cdot a_{\mathrm{f}} \in \mathcal{V} \Rightarrow b \cdot a_{\mathrm{f}} \in \mathcal{V}}{\boxed{a}^{\gamma} \Rrightarrow\!\!\!\ast\ \boxed{b}^{\gamma}}$$

Ghost variables revisited

Resource algebra for ghost variables:

$$
\begin{aligned}
M &\triangleq \bullet\, n \mid \circ\, n \mid \bot \mid \bullet\!\circ\, n \\
\mathcal{V} &\triangleq \{a \neq \bot \mid a \in M\} \\
\bullet\, n \cdot \circ\, n' = \circ\, n' \cdot \bullet\, n &\triangleq \begin{cases} \bullet\!\circ\, n & \text{if } n = n' \\ \bot & \text{otherwise} \end{cases} \\
\text{other combinations} &\triangleq \bot
\end{aligned}
$$

And define:

$$
\gamma \hookrightarrow_{\bullet} n \triangleq \boxed{\bullet\, n}^{\gamma} \qquad\qquad \gamma \hookrightarrow_{\circ} n \triangleq \boxed{\circ\, n}^{\gamma}
$$

Ghost variables revisited

Resource algebra for ghost variables:

$$M \triangleq \bullet\, n \mid \circ\, n \mid \bot \mid \bullet\!\circ\, n$$

$$\mathcal{V} \triangleq \{a \neq \bot \mid a \in M\}$$

$$\bullet\, n \cdot \circ\, n' = \circ\, n' \cdot \bullet\, n \triangleq \begin{cases} \bullet\!\circ\, n & \text{if } n = n' \\ \bot & \text{otherwise} \end{cases}$$

$$\text{other combinations} \triangleq \bot$$

And define:

$$\gamma \hookrightarrow_{\bullet} n \triangleq \boxed{\bullet\, n}^{\gamma} \qquad\qquad \gamma \hookrightarrow_{\circ} n \triangleq \boxed{\circ\, n}^{\gamma}$$

The ghost variable rules follow directly from the general rules:

$$\text{True} \Rrightarrow\!\!\ast\; \exists \gamma.\, \gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} n$$

Ghost variables revisited

Resource algebra for ghost variables:

$$M \triangleq \bullet\, n \mid \circ\, n \mid \bot \mid \bullet\!\!\circ\, n$$

$$\mathcal{V} \triangleq \{a \neq \bot \mid a \in M\}$$

$$\bullet\, n \cdot \circ\, n' = \circ\, n' \cdot \bullet\, n \triangleq \begin{cases} \bullet\!\!\circ\, n & \text{if } n = n' \\ \bot & \text{otherwise} \end{cases}$$

$$\text{other combinations} \triangleq \bot$$

And define:

$$\gamma \hookrightarrow_{\bullet} n \triangleq \boxed{\bullet\, n}^{\gamma} \qquad\qquad \gamma \hookrightarrow_{\circ} n \triangleq \boxed{\circ\, n}^{\gamma}$$

The ghost variable rules follow directly from the general rules:

$$\text{True} \mathrel{\Rrightarrow\!\!\ast} \exists \gamma.\, \boxed{\bullet\!\!\circ\, n}^{\gamma} \mathrel{\Rrightarrow\!\!\ast} \exists \gamma.\, \gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} n$$

Ghost variables revisited

Resource algebra for ghost variables:

$$M \triangleq \bullet\, n \mid \circ\, n \mid \bot \mid \bullet\!\circ\, n$$

$$\mathcal{V} \triangleq \{a \neq \bot \mid a \in M\}$$

$$\bullet\, n \cdot \circ\, n' = \circ\, n' \cdot \bullet\, n \triangleq \begin{cases} \bullet\!\circ\, n & \text{if } n = n' \\ \bot & \text{otherwise} \end{cases}$$

$$\text{other combinations} \triangleq \bot$$

And define:

$$\gamma \hookrightarrow_{\bullet} n \triangleq \boxed{\bullet\, n}^{\gamma} \qquad\qquad \gamma \hookrightarrow_{\circ} n \triangleq \boxed{\circ\, n}^{\gamma}$$

The ghost variable rules follow directly from the general rules:

$$\text{True} \Rrightarrow\!\!\ast\ \exists \gamma.\, \boxed{\bullet\!\circ\, n}^{\gamma} \Rrightarrow\!\!\ast\ \exists \gamma.\, \gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} n$$

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

Ghost variables revisited

Resource algebra for ghost variables:

$$M \triangleq \bullet\, n \mid \circ\, n \mid \bot \mid \bullet\!\!\circ\, n$$

$$\mathcal{V} \triangleq \{a \neq \bot \mid a \in M\}$$

$$\bullet\, n \cdot \circ\, n' = \circ\, n' \cdot \bullet\, n \triangleq \begin{cases} \bullet\!\!\circ\, n & \text{if } n = n' \\ \bot & \text{otherwise} \end{cases}$$

$$\text{other combinations} \triangleq \bot$$

And define:

$$\gamma \hookrightarrow_{\bullet} n \triangleq \boxed{\bullet\, n}^{\gamma} \qquad\qquad \gamma \hookrightarrow_{\circ} n \triangleq \boxed{\circ\, n}^{\gamma}$$

The ghost variable rules follow directly from the general rules:

$$\text{True} \mathrel{\Rrightarrow\!\!\!\!\ast} \exists \gamma.\, \boxed{\bullet\!\!\circ\, n}^{\gamma} \mathrel{\Rrightarrow\!\!\!\!\ast} \exists \gamma.\, \gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} n$$

$$\gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} m \Rightarrow (\bullet\, n \cdot \circ\, m) \in \mathcal{V} \Rightarrow n = m$$

Updating resources

Resources can be *updated* using *frame-preserving updates:*

$$\frac{\forall a_{\mathrm{f}}.\, a \cdot a_{\mathrm{f}} \in \mathcal{V} \Rightarrow b \cdot a_{\mathrm{f}} \in \mathcal{V}}{\boxed{a}^{\gamma} \Rrightarrow\!\!\!\ast\ \boxed{b}^{\gamma}}$$

Key idea: a resource can be updated if the update does not invalidate the resources of concurrently-running threads

$$\begin{array}{ccccccccc}
\text{Thread 1} & & \text{Thread 2} & & \ldots & & \text{Thread n} & \\
a_1 & \cdot & a_2 & \cdot & \ldots & \cdot & a_n & \in \mathcal{V} \\
\wr\!\downarrow & & & & & & & \\
b_1 & \cdot & a_2 & \cdot & \ldots & \cdot & a_n & \in \mathcal{V}
\end{array}$$

Updating resources

Resources can be *updated* using *frame-preserving updates:*

$$\frac{\forall a_{\mathrm{f}}.\, a \cdot a_{\mathrm{f}} \in \mathcal{V} \Rightarrow b \cdot a_{\mathrm{f}} \in \mathcal{V}}{\boxed{a}^{\gamma} \Rrightarrow\!\!\ast\ \boxed{b}^{\gamma}}$$

Key idea: a resource can be updated if the update does not invalidate the resources of concurrently-running threads

$$\begin{array}{ccccccccc}
\text{Thread 1} & & \text{Thread 2} & & \ldots & & \text{Thread n} & & \\
\color{red}{a_1} & \cdot & a_2 & \cdot & \ldots & \cdot & a_n & & \in \mathcal{V} \\
\color{red}{\rightsquigarrow} & & & & & & & & \\
\color{red}{b_1} & \cdot & a_2 & \cdot & \ldots & \cdot & a_n & & \in \mathcal{V}
\end{array}$$

The rule $\gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} m \Rrightarrow\!\!\ast\ \gamma \hookrightarrow_{\bullet} n' * \gamma \hookrightarrow_{\circ} n'$ follows directly

In the papers

- The full definition of a resource algebra (RA)
- Combinators (fractions, products, finite maps, agreement, etc.) to modularly build many RAs
- Encoding of *state transition systems* as RAs
- Encoding of $\boxed{a}^{\gamma}$ in terms of something even simpler
- *Higher order ghost state*: RAs that circularly depend on *iProp*, the type of propositions

Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning

Ralf Jung
MPI-SWS &
Saarland University
jung@mpi-sws.org

David Swasey
MPI-SWS
swasey@mpi-sws.org

Filip Sieczkowski
Aarhus University
filips@cs.au.dk

Kasper Svendsen
Aarhus University
ksvendsen@cs.au.dk

Aaron Turon
Mozilla Research
aturon@mozilla.com

Lars Birkedal
Aarhus University
birkedal@cs.au.dk

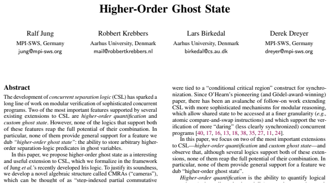
Derek Dreyer
MPI-SWS
dreyer@mpi-sws.org

Abstract

We present Iris, a concurrent separation logic with a simple premise: *monoids and invariants are all you need*. Partial commutative monoids enable us to express—and invariants enable us to enforce—user-defined *protocols* on shared state, which are at the conceptual core of most recent program logics for concurrency. Furthermore, through a novel extension of the concept of a *view shift*, Iris supports the encoding of *logically atomic specifications*, *i.e.*, Hoare-style

TaDA [8], and others. In this paper, we present a logic called **Iris** that explains some of the complexities of these prior separation logics in terms of a simpler unifying foundation, while also supporting some new and powerful reasoning principles for concurrency.

Before we get to Iris, however, let us begin with a brief overview of some key problems that arise in reasoning compositionally about shared state, and how prior approaches have dealt with them.



Higher-Order Ghost State

Ralf Jung
MPI-SWS, Germany
jung@mpi-sws.org

Robbert Krebbers
Aarhus University, Denmark
mail@robbertkrebbers.nl

Lars Birkedal
Aarhus University, Denmark
birkedal@cs.au.dk

Derek Dreyer
MPI-SWS, Germany
dreyer@mpi-sws.org

Abstract

The development of *concurrent separation logic* (CSL) has sparked a long line of work on modular verification of sophisticated concurrent programs. Two of the most important features supported by several existing extensions to CSL are *higher-order quantification* and *custom ghost state*. However, none of the logics that support both of these features reap the full potential of their combination. In particular, none of them provide general support for a feature we dub *"higher-order ghost state"*: the ability to store arbitrary higher-order separation-logic predicates in ghost variables.

In this paper, we propose higher-order ghost state as a interesting and useful extension to CSL, which we formalize in the framework of Jung *et al.*'s recently developed Iris logic. To justify its soundness, we develop a novel algebraic structure called CMRAs ("cameras"), which can be thought of as "step-indexed partial commutative

were tied to a "conditional critical region" construct for synchronization. Since O'Hearn's pioneering (and Gödel-award-winning) paper, there has been an avalanche of follow-on work extending CSL with more sophisticated mechanisms for modular reasoning, which allow shared state to be accessed at a finer granularity (*e.g.*, atomic compare-and-swap instructions) and which support the verification of more "daring" (less clearly synchronized) concurrent programs [40, 17, 16, 13, 18, 38, 35, 27, 11, 24].

In this paper, we focus on two of the most important extensions to CSL—*higher-order quantification* and *custom ghost state*—and observe that, although several logics support both of these extensions, none of them reap the full potential of their combination. In particular, none of them provide general support for a feature we dub "higher-order ghost state".

Higher-order quantification is the ability to quantify logical

Part #3: encoding Hoare triples

[Robbert Krebbers, Ralf Jung, Aleš Bizjak, Jacques-Henri Jourdan, Derek Dreyer, and Lars Birkedal. The Essence of Higher-Order Concurrent Separation Logic. In ESOP'17]

Encoding Hoare triples

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

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

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:

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

where $\mathrm{wp}\; e\, \{w.\, Q\}$ 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:

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

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

Adding the points-to connective

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

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

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

Adding the points-to connective

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

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

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

Solution: ghost variables

Adding the points-to connective

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

$$\mathsf{wp}\ e\ \{w.\ Q\} \triangleq \begin{cases} |\!\!\!\Rrightarrow Q[e/w] & \text{if } e \in \mathit{Val} \\ \forall \sigma.\ \boxed{\bullet\, \sigma}^{\gamma} \mathrel{-\!\!*} |\!\!\!\Rrightarrow & \text{if } e \notin \mathit{Val} \\ \qquad \mathrm{red}(e, \sigma) \wedge & \\ \qquad \triangleright (\forall e_2, \sigma_2.\ (e, \sigma) \to (e_2, \sigma_2, \epsilon) \mathrel{-\!\!*} |\!\!\!\Rrightarrow & \\ \qquad\qquad \boxed{\bullet\, \sigma_2}^{\gamma} * \mathsf{wp}\ e_2\ \{w.\ Q\}) & \end{cases}$$

$$\ell \mapsto v \triangleq \boxed{\circ\, [\ell := v]}^{\gamma}$$

Solution: ghost variables

Using an appropriate resource algebra we can obtain:

$$\boxed{\bullet\, \sigma}^{\gamma} * \boxed{\circ\, [\ell := w]}^{\gamma} \Rightarrow \sigma(\ell) = w$$

$$\boxed{\bullet\, \sigma}^{\gamma} * \boxed{\circ\, [\ell := v]}^{\gamma} \Rrightarrow\!\!\!\ast\ \boxed{\bullet\, \sigma[\ell := w]}^{\gamma} * \boxed{\circ\, [\ell := w]}^{\gamma}$$

$$\boxed{\bullet\, \sigma}^{\gamma} \Rrightarrow\!\!\!\ast\ \boxed{\bullet\, \sigma[\ell := w]}^{\gamma} * \boxed{\circ\, [\ell := w]}^{\gamma} \quad \text{if } \ell \notin \mathrm{dom}(\sigma)$$

The update modality

The *update modality* $\mathrel{|\!\!\Rrightarrow}$ internalizes frame-preserving updates:

$$\frac{\forall a_{\mathrm{f}}.\, a \cdot a_{\mathrm{f}} \in \mathcal{V} \Rightarrow b \cdot a_{\mathrm{f}} \in \mathcal{V}}{\boxed{a}^{\gamma} \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow} \boxed{b}^{\gamma}}$$

(We often write $P \Rrightarrow\!\!\!\!* \; Q$ for $P \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow} Q$)

It has the following set of **primitive** rules:

$$\frac{P \mathrel{-\!\!*} Q}{\mathrel{|\!\!\Rrightarrow} P \mathrel{-\!\!*} \mathrel{|\!\!\Rrightarrow} Q} \qquad \frac{P}{\mathrel{|\!\!\Rrightarrow} P} \qquad \frac{\mathrel{|\!\!\Rrightarrow}\mathrel{|\!\!\Rrightarrow} P}{\mathrel{|\!\!\Rrightarrow} P} \qquad \frac{Q * \mathrel{|\!\!\Rrightarrow} P}{\mathrel{|\!\!\Rrightarrow}(Q * P)}$$

The update modality

The *update modality* $|\!\!\Rrightarrow$ internalizes frame-preserving updates:

$$\frac{\forall a_{\mathrm{f}}.\, a \cdot a_{\mathrm{f}} \in \mathcal{V} \Rightarrow b \cdot a_{\mathrm{f}} \in \mathcal{V}}{\boxed{a}^{\gamma} \mathrel{-\!\!*} |\!\!\Rrightarrow \boxed{b}^{\gamma}}$$

(We often write $P \mathrel{\Rrightarrow\!\!\!\ast} Q$ for $P \mathrel{-\!\!*} |\!\!\Rrightarrow Q$)

It has the following set of **primitive** rules:

$$\frac{P \mathrel{-\!\!*} Q}{|\!\!\Rrightarrow P \mathrel{-\!\!*} |\!\!\Rrightarrow Q} \qquad \frac{P}{|\!\!\Rrightarrow P} \qquad \frac{|\!\!\Rrightarrow |\!\!\Rrightarrow P}{|\!\!\Rrightarrow P} \qquad \frac{Q * |\!\!\Rrightarrow P}{|\!\!\Rrightarrow (Q * P)}$$

From the definition of weakest preconditions we can **derive**:

$$\frac{|\!\!\Rrightarrow \mathrm{wp}\; e\,\{w.\; |\!\!\Rrightarrow Q\}}{\mathrm{wp}\; e\,\{w.\; Q\}}$$

That lets us update resources around weakest preconditions

Adding fork

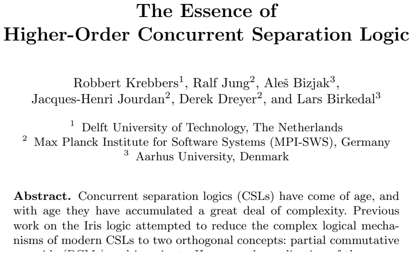
$$
\mathsf{wp}\ e\,\{w.\,Q\} \triangleq
\begin{cases}
\mathrel{|\!\!\!\Rrightarrow} Q[e/w] & \text{if } e \in \mathit{Val} \\
\forall \sigma.\ \boxed{\bullet\,\sigma}^{\gamma} \mathrel{-\!\!*} \mathrel{|\!\!\!\Rrightarrow} & \text{if } e \notin \mathit{Val} \\
\qquad \mathrm{red}(e,\sigma) \wedge & \\
\qquad \triangleright\big(\forall e_2, \sigma_2, \vec{e_f}.\ (e,\sigma) \to (e_2, \sigma_2, \vec{e_f}) \mathrel{-\!\!*} \mathrel{|\!\!\!\Rrightarrow} & \\
\qquad\qquad \boxed{\bullet\,\sigma_2}^{\gamma} * \mathsf{wp}\ e_2\,\{w.\,Q\} * & \\
\qquad\qquad \mathop{\Large\ast}_{e' \in \vec{e_f}} \mathsf{wp}\ e'\,\{w.\mathsf{True}\}\big) &
\end{cases}
$$

$$
\ell \mapsto v \triangleq \boxed{\circ\,[\ell := v]}^{\gamma}
$$

Key point: separating conjunction ensures that resources are subdivided between threads without explicit disjointness

In the paper

- Encoding of invariants $\boxed{P}^{\mathcal{N}}$ using higher-order ghost state
- All about the modalities $\square$, $\triangleright$ and $|\!\!\Rrightarrow$
- 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

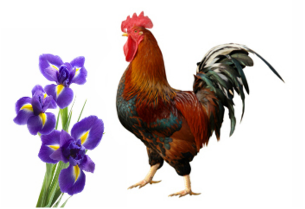
Abstract. Concurrent separation logics (CSLs) have come of age, and with age they have accumulated a great deal of complexity. Previous work on the Iris logic attempted to reduce the complex logical mechanisms of modern CSLs to two orthogonal concepts: partial commutative

Part #4: Iris Proof Mode (IPM) in Coq

[Robbert Krebbers, Amin Timany, and Lars Birkedal. Interactive proofs in higher-order concurrent separation logic. In POPL'17]

IPM: Iris proof mode

- Coq with (spatial and non-spatial) named proof contexts for Iris
- Tactics for introduction and elimination of all Iris connectives
- Entirely implemented using reflection, type classes and Ltac



IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
______________________________________(1/1)
P ∗ (∃ a : A, Ψ a) ∗ R −∗ ∃ a : A, Ψ a ∗ P
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
______________________________________(1/1)
"HP" : P
"HΨ" : ∃ a : A, Ψ a
"HR" : R
--------------------------------------∗
∃ a : A, Ψ a ∗ P
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "H".
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
x : A
______________________________________(1/1)
"HP" : P
"HΨ" : Ψ x
"HR" : R
--------------------------------------∗
∃ a : A, Ψ a ∗ P
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "H".
  iExists x.
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
x : A
______________________________________(1/1)
"HP" : P
"HΨ" : Ψ x
"HR" : R
---------------------------------------∗
Ψ x ∗ P
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "H".
  iExists x.
  iFrame "HΨ".
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
x : A
______________________________________(1/1)
"HP" : P
"HR" : R
---------------------------------------∗
P
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "H".
  iExists x.
  iFrame "HΨ".
  done.
```

```
No more subgoals.
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "H".
  iExists x.
  iFrame "HΨ".
  done.
Qed.
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
______________________________________(1/1)
"HR" : R
--------------------------------------∗
∃ a : A, Ψ a ∗ P
```

Logical notations overridden in scope for Iris

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
______________________________________(1/1)
"HP" : P
"HΨ" : ∃ a : A, Ψ a
"HR" : R
                        -----------------------∗
                        P
```

Notation for deeply embedded context

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  Unset Printing Notations.
```

```
1 subgoal
M : ucmraT
A : Type@{Top.105}
P, R : uPred M
Ψ : forall _ : A, uPred M
______________________________________(1/1)
@uPred_entails M
 (@coq_tactics.of_envs M
   (@coq_tactics.Envs M (@Enil (uPred M))
    (@Esnoc (uPred M)
      (@Esnoc (uPred M)
       (@Esnoc (uPred M) (@Enil (uPred M))
         (String
          (Ascii false false false true false false
      true
            false)
          (String
            (Ascii false false false false true
      false true
             false) EmptyString)) P)
       (String
         (Ascii false false false true false false
      true false)
         (String
          (Ascii false true true true false false
      true true)
          (String
            (Ascii false false false true false true
       false
             true) EmptyString)))
```

IPM demo

```
Lemma and_exist_sep
    {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

Introduction patterns represented as strings

The setup of IPM in a nutshell

- Deep embedding of contexts as association lists:

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜ envs_wf Δ ⌝ ∗ □ [∗] env_persistent Δ ∗
                     [∗] env_spatial Δ)%I.
```

The setup of IPM in a nutshell

- Deep embedding of contexts as association lists:

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜envs_wf Δ⌝ ∗ □ [∗] env_persistent Δ ∗
                   [∗] env_spatial Δ)%I.
```

Propositions that enjoy $P \Leftrightarrow P * P$

The setup of IPM in a nutshell

- Deep embedding of contexts as association lists:

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜envs_wf Δ⌝ ∗ □ [∗] env_persistent Δ ∗
                 [∗] env_spatial Δ)%I.
```

Propositions that enjoy $P \Leftrightarrow P * P$

- Tactics implemented by reflection:

```
Lemma tac_sep_split Δ Δ1 Δ2 lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ1,Δ2) →
  (Δ1 ⊢ Q1) → (Δ2 ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

The setup of IPM in a nutshell

- Deep embedding of contexts as association lists:

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜envs_wf Δ⌝ ∗ □ [∗] env_persistent Δ ∗
                  [∗] env_spatial Δ)%I.
```

Propositions that enjoy $P \Leftrightarrow P * P$

- Tactics implemented by reflection:

```
Lemma tac_sep_split Δ Δ₁ Δ₂ lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ₁,Δ₂) →
  (Δ₁ ⊢ Q1) → (Δ₂ ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

Context splitting implemented in Gallina

The setup of IPM in a nutshell

- Deep embedding of contexts as association lists:

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜envs_wf Δ⌝ ∗ □ [∗] env_persistent Δ ∗
                     [∗] env_spatial Δ)%I.
```

Propositions that enjoy $P \Leftrightarrow P * P$

- Tactics implemented by reflection:

```
Lemma tac_sep_split Δ Δ₁ Δ₂ lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ₁,Δ₂) →
  (Δ₁ ⊢ Q1) → (Δ₂ ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

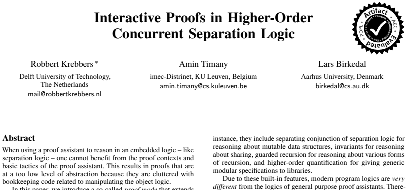
Context splitting implemented in Gallina

- Ltac wrappers around reflective tactics:

```
Tactic Notation "iSplitL" constr(Hs) :=
  let Hs := words Hs in
  eapply tac_sep_split with _ _ false Hs _ _;
    [env_cbv; reflexivity| (*goal 1 *) | (*goal 2*) ].
```

In the paper

- Framing, later stripping, . . . using type classes
- Modular IPM tactics using type classes
- Tactics for symbolic execution
- Verification of concurrent algorithms using IPM
- Formalization of unary and binary logical relations
- Proving logical refinements



Interactive Proofs in Higher-Order Concurrent Separation Logic

Robbert Krebbers *
Delft University of Technology,
The Netherlands
mail@robbertkrebbers.nl

Amin Timany
imec-Distrinet, KU Leuven, Belgium
amin.timany@cs.kuleuven.be

Lars Birkedal
Aarhus University, Denmark
birkedal@cs.au.dk

Abstract

When using a proof assistant to reason in an embedded logic – like separation logic – one cannot benefit from the proof contexts and basic tactics of the proof assistant. This results in proofs that are at a too low level of abstraction because they are cluttered with bookkeeping code related to manipulating the object logic.

In this paper, we introduce a so-called *proof mode* that extends

instance, they include separating conjunction of separation logic for reasoning about mutable data structures, invariants for reasoning about sharing, guarded recursion for reasoning about various forms of recursion, and higher-order quantification for giving generic modular specifications to libraries.

Due to these built-in features, modern program logics are *very different* from the logics of general purpose proof assistants. There-

Thank you!

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

Talks about Iris this week:

- Wed 15:10 @ POPL: *Krebbers*, Timany and Birkedal
  Interactive Proofs in Higher-Order Concurrent Separation Logic
- Wed 15:35 @ POPL: Krogh-Jespersen, *Svendsen* and Birkedal
  A Relational Model of Types-and-Effects in Higher-Order Concurrent Separation Logic
- Sat 9:00 @ CoqPL: *Krebbers*
  Demo and implementation of Iris in Coq
- Sat 10:30 @ CoqPL: *Timany*, Krebbers and Birkedal
  Logical Relations in Iris