Iris, Iris proof mode and Program verification in Iris

Amin Timany[1,2]

iMec-DistriNet, KU Leuven

IFIP 1.9 meeting

May 12th 2017

KU Leuven

---

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

[2] Based on slides of Robebrt Krebbers' talks at TTT'17 and POPL'17

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

- Program verification in Iris
- Iris Proof mode: facilitating proofs 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:

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

If the initial state satisfies P, then:

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

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:

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

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:

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

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

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

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:

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

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:

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

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:

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

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

$$!\, x$$

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

What about shared state/racy programs?

A classic problem:

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

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

What about shared state/racy programs?

A classic problem:

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

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

What about shared state/racy programs?

A classic problem:

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

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:

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

Invariant allocation:

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

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:

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

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

$!\, x$

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

Invariants in action

Let us consider a simpler problem first:

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

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

$!\,x$

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

Invariants in action

Let us consider a simpler problem first:

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

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

$!\, x$

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

Invariants in action

Let us consider a simpler problem first:

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

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

$\quad !\, x$

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

Invariants in action

Let us consider a simpler problem first:

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

$\{\mathsf{True}\}$	$\{\mathsf{True}\}$
$\{x \mapsto n \wedge even(n)\}$	
$\texttt{fetchandadd}(x, 2)$	$\texttt{fetchandadd}(x, 2)$
$\{x \mapsto n + 2 \wedge even(n + 2)\}$	
$\{\mathsf{True}\}$	$\{\mathsf{True}\}$

$$!\, x$$

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

Invariants in action

Let us consider a simpler problem first:

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

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

$!\,x$

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

Invariants in action

Let us consider a simpler problem first:

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

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

Invariants in action

Let us consider a simpler problem first:

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

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:

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

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

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

$$\gamma \hookrightarrow_\bullet n * \gamma \hookrightarrow_\circ m \quad \Rrightarrow \quad \gamma \hookrightarrow_\bullet n' * \gamma \hookrightarrow_\circ n'$$

Ghost variables in action

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

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

$!\, x$

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

Ghost variables in action

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

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

$!\,x$

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

Ghost variables in action

$\{\mathsf{True}\}$
$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$
$\{x \mapsto 0\}$
$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$!\, x$

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

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

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

$!\, x$

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

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 $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$!\, x$

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

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

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

$\{\gamma_1 \hookrightarrow_\circ 0\}$	$\parallel$	$\{\gamma_2 \hookrightarrow_\circ 0\}$
$\texttt{fetchandadd}(x, 2)$	$\parallel$	$\texttt{fetchandadd}(x, 2)$

$!\, x$

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

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

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

$\{\gamma_1 \hookrightarrow_\circ 0\}$	$\|$	$\{\gamma_2 \hookrightarrow_\circ 0\}$
$\texttt{fetchandadd}(x, 2)$	$\|$	$\texttt{fetchandadd}(x, 2)$
$\{\gamma_1 \hookrightarrow_\circ 2\}$	$\|$	$\{\gamma_2 \hookrightarrow_\circ 2\}$

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

$!\,x$

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

Ghost variables in action

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

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

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

$!\,x$

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

Ghost variables in action

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

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

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

$!\, x$

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

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 $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

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

$!\,x$

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

Ghost variables in action

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

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

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

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

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

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

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

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

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

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

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

$$\parallel$$

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

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

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

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

$$!\,x$$

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

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

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

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

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

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

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

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

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

$\|$

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

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

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

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

$!\,x$

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

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

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

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

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

$!\,x$

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

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 $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

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

$!\, x$

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

Ghost variables in action

$\{\text{True}\}$
`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`$(x, 2)$
$\{\ldots\}$
$\{\gamma_2 \hookrightarrow_\circ 2\}$

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

$!\, x$

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

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

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

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

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

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

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

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

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

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

$\parallel$

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

$\{\ldots\}$

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

$\{\ldots\}$

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

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

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

$!\,x$

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

Ghost variables in action

$\{\mathsf{True}\}$
`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 $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

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

$!\,x$

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

Ghost variables in action

$\{\mathrm{True}\}$
$\mathtt{let}\ x = \mathtt{ref}(0)\ \mathtt{in}$
$\{x \mapsto 0\}$
$\{x \mapsto 0 * \gamma_1 \hookrightarrow_\bullet 0 * \gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\bullet 0 * \gamma_2 \hookrightarrow_\circ 0\}$
allocate $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

$\|$

$\{\gamma_2 \hookrightarrow_\circ 0\}$
$\{\ldots\}$
$\mathtt{fetchandadd}(x, 2)$
$\{\ldots\}$
$\{\gamma_2 \hookrightarrow_\circ 2\}$

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

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

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 $\boxed{\exists n_1, n_2.\, x \mapsto n_1 + n_2 * \gamma_1 \hookrightarrow_\bullet n_1 * \gamma_2 \hookrightarrow_\bullet n_2}$
$\{\gamma_1 \hookrightarrow_\circ 0 * \gamma_2 \hookrightarrow_\circ 0\}$

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

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

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

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

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

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

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

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

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

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

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

$\|$

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

$\{\ldots\}$

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

$\{\ldots\}$

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

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

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

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

$!\,x$

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

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

Ghost variables in action

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

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

$$\parallel$$

$$\{\gamma_2 \hookrightarrow_\circ 0\}$$
$$\{\ldots\}$$
$$\texttt{fetchandadd}(x, 2)$$
$$\{\ldots\}$$
$$\{\gamma_2 \hookrightarrow_\circ 2\}$$

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

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* $(0,1]_{\mathbb{Q}}$:

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

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* ($0 < \pi \leq 1$):

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

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* $(0 < \pi \leq 1)$:

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

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

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

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

$!\, x$

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

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

$\{\mathsf{True}\}$
$\mathtt{let}\, x = \mathtt{ref}(0)\, \mathtt{in}$
$\{x \mapsto 0\}$
$\left\{x \mapsto 0 * \gamma \hookrightarrow_{\bullet} 0 * \gamma \overset{1}{\hookrightarrow}_{\circ} 0\right\}$

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

$!\,x$

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

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

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

$!\, x$

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

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

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

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

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

$!\, x$

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

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

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$
$\texttt{fetchandadd}(x, 2)$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\|$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\texttt{fetchandadd}(x, 2)$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\| \ \ldots$

$!\,x$

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

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

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

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

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

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

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

$\|$

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

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

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

$\|$ $\ldots$

$!\,x$

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

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

$\|$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\{\ldots\}$
`fetchandadd`$(x, 2)$
$\{\ldots\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\| \ldots$

$!\, x$

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

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

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$
$\texttt{fetchandadd}(x, 2)$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2 * x \mapsto (2+n) * \gamma_1 \hookrightarrow_{\bullet} (2+n)\right\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\parallel$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\{\ldots\}$
$\texttt{fetchandadd}(x, 2)$
$\{\ldots\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\parallel \ \ldots$

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

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

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

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0 * x \mapsto n * \gamma \hookrightarrow_{\bullet} n\right\}$
$\texttt{fetchandadd}(x, 2)$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2 * x \mapsto (2{+}n) * \gamma_1 \hookrightarrow_{\bullet} (2{+}n)\right\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\|$

$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 0\right\}$
$\{\ldots\}$
$\texttt{fetchandadd}(x, 2)$
$\{\ldots\}$
$\left\{\gamma \overset{1/k}{\hookrightarrow}_{\circ} 2\right\}$

$\| \ldots$

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

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]

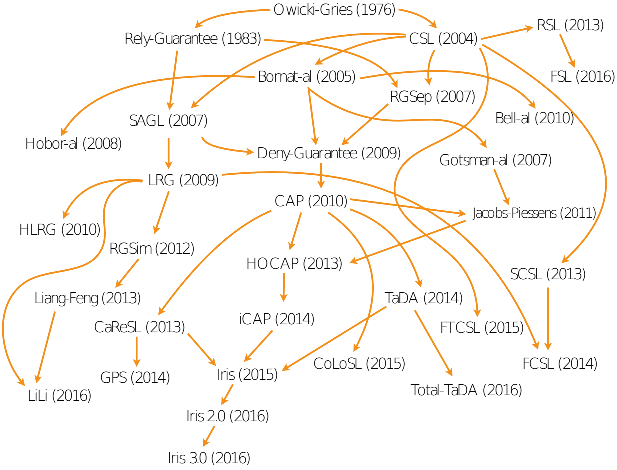
Mechanisms for concurrent reasoning

We have seen 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

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

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

Use atomic rule

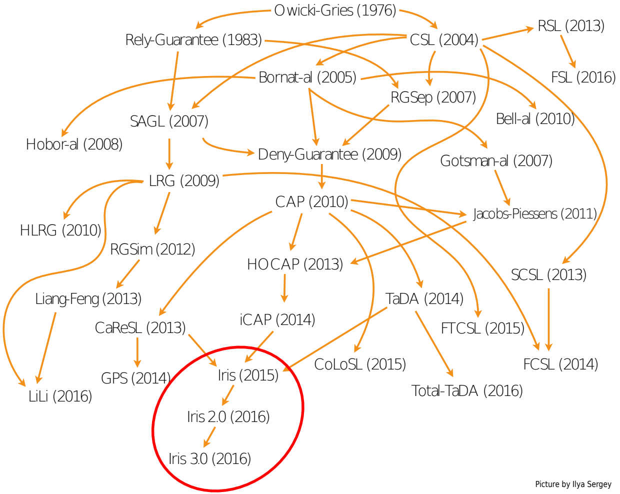
$$\frac{\begin{array}{c}a \notin \mathcal{A} \quad \forall x \in X.\, (x, f(x)) \in \mathcal{T}_{\mathbf{t}}(\mathrm{G})^{*} \\ \lambda; \mathcal{A} \vdash \mathbb{V}x \in X.\, \left\langle p_p \,\middle|\, I(\mathbf{t}^{\lambda}_{a}(x)) * p(x) * [\mathrm{G}]_a \right\rangle\ \mathbb{C}\ \ \exists\!\!\exists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

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

Update region rule

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

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:

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

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

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

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

And define:

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

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:

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

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:

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

The ghost variable rules follow directly from the general rules:

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

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

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:

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

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

Updating resources

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

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

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

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

Updating resources

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

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

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 $\gamma \hookrightarrow_{\bullet} n * \gamma \hookrightarrow_{\circ} m \Rrightarrow \gamma \hookrightarrow_{\bullet} n' * \gamma \hookrightarrow_{\circ} n'$ 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

[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]

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

You can find:

- Encoding Hoare triples using higher-order ghost state
- 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

Part #3: Iris Proof Mode (IPM) in Coq

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

Goal of this part

Many POPL papers about complicated program logics come with mechanized soundness proofs, but how to reason in these logics?

Goal: reasoning in an object logic in the same style as reasoning in Coq

Goal of this part

Many POPL papers about complicated program logics come with mechanized soundness proofs, but how to reason in these logics?

Goal: reasoning in an object logic in the same style as reasoning in Coq

Goal of this part

Many POPL papers about complicated program logics come with mechanized soundness proofs, but how to reason in these logics?

Goal: reasoning in an object logic in the same style as reasoning in Coq

How?

- Extend Coq with (spatial and non-spatial) named proof contexts for an object logic
- Tactics for introduction and elimination of all connectives of the object logic
- Entirely implemented using reflection, type classes and Ltac (no OCaml plugin needed)

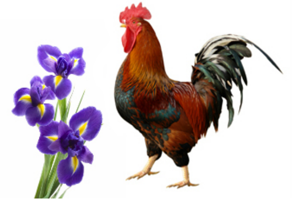
Goal of this part

Many POPL papers about complicated program logics come with mechanized soundness proofs, but how to reason in these logics?

Goal: reasoning in Iris in the same style as reasoning in Coq

How?

- Extend Coq with (spatial and non-spatial) named proof contexts for Iris
- Tactics for introduction and elimination of all connectives of Iris
- Entirely implemented using reflection, type classes and Ltac (no OCaml plugin needed)



Iris: language independent higher-order separation logic for modular reasoning about fine-grained concurrency in Coq

Iris Proof Mode (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
```

Iris Proof Mode (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)
P ∗ (∃ a : A, Ψ a) ∗ R −∗ ∃ a : A, Ψ a ∗ P
```

Iris Proof Mode (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
```

Iris Proof Mode (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) "HΨ".
```

```
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
```

Iris Proof Mode (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) "HΨ".
```

```
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
```

Iris Proof Mode (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) "HΨ".
  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
--------------------------------------∗
∃ a : A, Ψ a ∗ P
```

Iris Proof Mode (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) "HΨ".
  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
```

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

```
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
```

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

```
2 subgoals
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
x : A
______________________________________(1/2)
"HΨ" : Ψ x
--------------------------------------∗
Ψ x

______________________________________(2/2)
"HP" : P
"HR" : R
--------------------------------------∗
P
```

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  -
```

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

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
```

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

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
```

```
This subproof is complete, but there are some
      unfocused goals:

______________________________________(1/1)
"HP" : P
"HR" : R
---------------------------------------∗
P
```

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  -
```

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

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
```

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

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
```

```
No more subgoals.
```

Iris Proof Mode (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) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
Qed.
```

```
No more subgoals.
```

Iris Proof Mode (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) "HΨ".
```

Logical notations overridden in scope for Iris

```
  - iAssumption.
  - iAssumption.
Qed.
```

Iris Proof Mode (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

Iris Proof Mode (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
P, R : iProp
Ψ : A → iProp
______________________________________(1/1)
"HP" : P
"HΨ" : ∃ a : A, Ψ a
"HR" : R
--------------------------------------∗
P
```

Notation for deeply embedded context

Iris Proof Mode (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
 (@of_envs M
   (@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)))
```

Motivation

Why should we care about interactive proofs? Why not automate everything?

Infeasible to automate everything, for example:

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

Most of these projects are formalized in IPM

How to do such proofs in a proof assistant?

Current proof assistant support is limited to basic separation logic:

- **Macros for manipulating Hoare triples:** Appel, Wright, Charge!, ...
- **Heavy automation:** Bedrock, Rtac, ...

Iris has many complicated connectives that are beyond basic separation logic

How to embed a logic into a proof assistant

Deep embedding

```
Inductive form : Type :=
  | iAnd: form → form → form
  | iForall: string → form → form → form
```

Shallow embedding

```
Definition iProp : Type :=
  (* predicates over states *).
Definition iAnd : iProp → iProp → iProp
        :=
  (* semantic interpretation *).
Definition iForall : ∀ A, (A → iProp) →
        iProp :=
  (* semantic interpretation *).
```

How to embed a logic into a proof assistant

Deep embedding	**Shallow embedding**

`Inductive form : Type :=`
`  | iAnd: form → form → form`
`  | iForall: string → form → form → form`
	`Definition iProp : Type :=`
	`  (* predicates over states *).`
	`Definition iAnd : iProp → iProp → iProp :=`
	`  (* semantic interpretation *).`
	`Definition iForall : ∀ A, (A → iProp) → iProp :=`
	`  (* semantic interpretation *).`

Traverse formulas using Coq functions (fast)	Traverse formulas on the meta level (slow)
Reflective tactics (fast)	Tactics on the meta level (slow)

How to embed a logic into a proof assistant

Deep embedding	**Shallow embedding**

`Inductive form : Type :=`
`  | iAnd: form → form → form`
`  | iForall: string → form → form → form`
	`Definition iProp : Type :=`
	`  (* predicates over states *).`
	`Definition iAnd : iProp → iProp → iProp`
	`      :=`
	`  (* semantic interpretation *).`
	`Definition iForall : ∀ A, (A → iProp) →`
	`      iProp :=`
	`  (* semantic interpretation *).`

Traverse formulas using Coq functions (fast)	Traverse formulas on the meta level (slow)
Reflective tactics (fast)	Tactics on the meta level (slow)
Need to explicitly encode binders	Reuse binders of Coq
Need to embed features like lists	Piggy-back on features like lists from Coq

How to embed a logic into a proof assistant

Deep embedding	Shallow embedding

`Inductive form : Type :=`
`  | iAnd: form → form → form`
`  | iForall: string → form → form → form`
	`Definition iProp : Type :=`
	`  (* predicates over states *).`
	`Definition iAnd : iProp → iProp → iProp :=`
	`  (* semantic interpretation *).`
	`Definition iForall : ∀ A, (A → iProp) → iProp :=`
	`  (* semantic interpretation *).`

Traverse formulas using Coq functions (fast)	Traverse formulas on the meta level (slow)
Reflective tactics (fast)	Tactics on the meta level (slow)
Need to explicitly encode binders	Reuse binders of Coq
Need to embed features like lists	Piggy-back on features like lists from Coq
Grammar of formulas fixed once and forall	Easily extensible with new connectives

How to embed a logic into a proof assistant

Deep embedding	Shallow embedding

`Inductive form : Type :=`
`  | iAnd: form → form → form`
`  | iForall: string → form → form → form`
	`Definition iProp : Type :=`
	`  (* predicates over states *).`
	`Definition iAnd : iProp → iProp → iProp :=`
	`  (* semantic interpretation *).`
	`Definition iForall : ∀ A, (A → iProp) → iProp :=`
	`  (* semantic interpretation *).`

Traverse formulas using Coq functions (fast)	Traverse formulas on the meta level (slow)
Reflective tactics (fast)	Tactics on the meta level (slow)
Need to explicitly encode binders	Reuse binders of Coq
Need to embed features like lists	Piggy-back on features like lists from Coq
Grammar of formulas fixed once and forall	Easily extensible with new connectives

Context manipulation is the prime task of tactics:
Deeply embed contexts, shallowly embed the logic

Deeply embedded contexts in IPM

Visible goal in IPM:

$\vec{\mathtt{x}}$: $\vec{\phi}$ Variables and pure Coq hypotheses

____

$\vec{\mathtt{H}}$`persistent` : $\vec{P}$ Persistent hypotheses in object logic

$\overline{}\Box$

$\vec{\mathtt{H}}$`spatial` : $\vec{Q}$ Spatial hypotheses in object logic

$\overline{}*$

R Goal in object logic

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

$\vec{\mathtt{x}}$: $\vec{\phi}$ Variables and pure Coq hypotheses

$\overline{\hspace{20em}}$

$\vec{\mathtt{H}}\mathtt{persistent}$: $\vec{P}$ Persistent hypotheses in object logic

$\overline{\hspace{20em}}\square$

$\vec{\mathtt{H}}\mathtt{spatial}$: $\vec{Q}$ Spatial hypotheses in object logic

$\overline{\hspace{20em}}*$

R Goal in object logic

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

$\vec{\mathtt{x}}$: $\vec{\phi}$ Variables and pure Coq hypotheses

———————————————————————————

$\vec{\mathtt{H}}\mathtt{persistent}$: $\vec{P}$ Persistent hypotheses in object logic

———————————————————————————□

$\vec{\mathtt{H}}\mathtt{spatial}$: $\vec{Q}$ Spatial hypotheses in object logic

———————————————————————————∗

R Goal in object logic

Actual Coq goal (without pretty printing):

$\vec{\mathtt{x}}_i$: $\vec{\phi}_i$

———————————————————————————

of_envs (Envs ... ...) $\vdash R$

where:

```
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.
```

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

$\vec{\mathtt{x}} : \vec{\phi}$ Variables and pure Coq hypotheses

———————————————————————————

$\vec{\mathtt{H}}\mathtt{persistent} : \vec{P}$ Persistent hypotheses in object logic

———————————————————————————□

$\vec{\mathtt{H}}\mathtt{spatial} : \vec{Q}$ Spatial hypotheses in object logic

———————————————————————————∗

R Goal in object logic

Actual Coq goal (without pretty printing):

$\vec{\mathtt{x}}_i : \vec{\phi}_i$

———————————————————————————

of_envs (Envs ... ...) $\vdash R$

where:

Association list of shallowly embedded propositions

```
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.
```

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

$\vec{x}$: $\vec{\phi}$ Variables and pure Coq hypotheses

———————————————————

$\vec{\text{H}}$persistent : $\vec{P}$ Persistent hypotheses in object logic

———————————————————□

$\vec{\text{H}}$spatial : $\vec{Q}$ Spatial hypotheses in object logic

———————————————————∗

R Goal in object logic

Actual Coq goal (without pretty printing):

$\vec{x}_i$: $\vec{\phi}_i$

———————————————————

of_envs (Envs ... ...) $\vdash R$

where:

Association list of shallowly embedded propositions

Folded separating conjunction

```
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 iSplit tactic

```
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) "HΨ".
  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
```

The `iSplit` tactic

```
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) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

```
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
```

The iSplit tactic

```
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) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

```
2 subgoals
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
x : A
______________________________________(1/2)
"HΨ" : Ψ x
--------------------------------------∗
Ψ x

______________________________________(2/2)
"HP" : P
"HR" : R
--------------------------------------∗
P
```

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
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.
```

Implementation of the `iSplit` tactic

Tactics implemented by reflection as mere lemmas:

```
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 as a computable Coq function

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
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 as a computable Coq function

Ltac wrappers around the reflective tactic:

```
Tactic Notation "iSplitL" constr(Hs) :=
  let Hs := words Hs in
  eapply tac_sep_split with _ _ false Hs _ _;
    [env_cbv; reflexivity ||
      fail "iSplitL: hypotheses" Hs "not found in the context"
    | (* goal 1 *)
    | (* goal 2 *) ].
```

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
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 as a computable Coq function

Ltac wrappers around the reflective tactic:

```
Tactic Notation "iSplitL" constr(Hs) :=
  let Hs := words Hs in
  eapply tac_sep_split with _ _ false Hs _ _;
    [env_cbv; reflexivity ||
      fail "iSplitL: hypotheses" Hs "not found in the context"
    | (* goal 1 *)
    | (* goal 2 *) ].
```

Report sensible error to the user

The iFrame tactic

```
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) "HΨ".
```

```
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
```

The iFrame tactic

```
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) "HΨ".
  iFrame "HP".
```

```
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
```

The iFrame tactic

```
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) "HΨ".
  iFrame "HP".
```

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

Implementation of the `iFrame` tactic

Problem: the goal is not deeply embedded, how to manipulate it?

Implementation of the `iFrame` tactic

Problem: the goal is not deeply embedded, how to manipulate it?

Solution: logic programming using type classes

The lemma corresponding to the tactic in Coq:

```
Class Frame (R P Q : iProp) := frame : R ∗ Q ⊢ P.
```

What we want to frame

Conclusion of the new goal in which R is framed

Initial conclusion

```
Lemma tac_frame Δ Δ' i p R P Q :
  envs_lookup_delete i Δ = Some (p, R, Δ') →
  Frame R P Q →
  ((if p then Δ else Δ') ⊢ Q) → Δ ⊢ P.
```

Implementation of the `iFrame` tactic

Problem: the goal is not deeply embedded, how to manipulate it?

Solution: logic programming using type classes

The lemma corresponding to the tactic in Coq:

```
Class Frame (R P Q : iProp) := frame : R ∗ Q ⊢ P.
```

What we want to frame

Conclusion of the new goal in which R is framed

Initial conclusion

```
Lemma tac_frame Δ Δ' i p R P Q :
  envs_lookup_delete i Δ = Some (p, R, Δ') →
  Frame R P Q →
  ((if p then Δ else Δ') ⊢ Q) → Δ ⊢ P.
```

Note: we support framing under binders ($\exists$, $\forall$, ...) and user defined connectives

Implementation of the `iFrame` tactic (2)

Consider the type class:

```
Class Frame (R P Q : iProp) := frame : R ∗ Q ⊢ P.
```

What we want to frame

Conclusion of the new goal in which `R` is framed

Initial conclusion

Implementation of the `iFrame` tactic (2)

Consider the type class:

```
Class Frame (R P Q : iProp) := frame : R ∗ Q ⊢ P.
```

What we want to frame

Conclusion of the new goal in which R is framed

Initial conclusion

Instances (rules of the logic program):

```
Instance frame_here R : Frame R R True.
Instance frame_sep_l R P₁ P₂ Q :
  Frame R P₁ Q → Frame R (P₁ ∗ P₂) (Q ∗ P₂).
Instance frame_sep_r R P₁ P₂ Q :
  Frame R P₂ Q → Frame R (P₁ ∗ P₂) (P₁ ∗ Q).
```

Implementation of the iFrame tactic (2)

Consider the type class:

```
Class Frame (R P Q : iProp) := frame : R ∗ Q ⊢ P.
```

What we want to frame

Conclusion of the new goal in which R is framed

Initial conclusion

Instances (rules of the logic program):

```
Class MakeSep P Q PQ := make_sep : P ∗ Q ⊣⊢ PQ.
Instance frame_here R : Frame R R True.
Instance frame_sep_l R P₁ P₂ Q Q' :
  Frame R P₁ Q → MakeSep Q P₂ Q' → Frame R (P₁ ∗ P₂) Q'.
Instance frame_sep_r R P₁ P₂ Q Q' :
  Frame R P₂ Q → MakeSep P₁ Q Q' → Frame R (P₁ ∗ P₂) Q'.

Instance make_sep_true_l P : MakeSep True P P | 1.
Instance make_sep_true_r P : MakeSep P True P | 1.
Instance make_sep_default P Q : MakeSep P Q (P ∗ Q) | 2.
```

Proving Hoare triples

Consider:

$$\{x \mapsto v_1 * y \mapsto v_2\} \mathit{swap}(x, y) \{x \mapsto v_2 * y \mapsto v_1\}$$

How to use IPM to manipulate the precondition?

Proving Hoare triples

Consider:

$$\{x \mapsto v_1 * y \mapsto v_2\}\, swap(x, y)\, \{x \mapsto v_2 * y \mapsto v_1\}$$

How to use IPM to manipulate the precondition?

$\boxed{\text{Solution: define Hoare triple in terms of weakest preconditions}}$

We let:

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

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

- all executions of e are safe
- the final state of e satisfies the postcondition Q

Proving `swap` using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
{{ l1 ↦ v1 ∗ l2 ↦ v2 }} (swap #l1) #l2 {{ _, l1 ↦ v2
       ∗ l2 ↦ v1 }}
```

Proving `swap` using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
"Hl1" : l1 ↦ v1
"Hl2" : l2 ↦ v2
--------------------------------------∗
WP (swap #l1) #l2 {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}
```

Proving swap using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
"Hl1" : l1 ↦ v1
"Hl2" : l2 ↦ v2
--------------------------------------∗
WP
  let: "tmp" := ! #l1 in
  #l1 ← ! #l2 ;;
  #l2 ← "tmp" {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}
```

Proving swap using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
  wp_load; wp_let.
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
"Hl1" : l1 ↦ v1
"Hl2" : l2 ↦ v2
--------------------------------------∗
WP #l1 ← ! #l2 ;; #l2 ← v1 {{ _, l1 ↦ v2 ∗ l2 ↦ v1
       }}
```

Proving `swap` using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
  wp_load; wp_let.
  wp_load.
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
"Hl1" : l1 ↦ v1
"Hl2" : l2 ↦ v2
--------------------------------------∗
WP #l1 ← v2 ;; #l2 ← v1 {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}
```

Proving swap using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
  wp_load; wp_let.
  wp_load.
  wp_store.
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
"Hl1" : l1 ↦ v2
"Hl2" : l2 ↦ v2
--------------------------------------∗
WP #l2 ← v1 {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}
```

Proving swap using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
  wp_load; wp_let.
  wp_load.
  wp_store.
  wp_store.
```

```
1 subgoal
Σ : gFunctors
H : heapG Σ
l1, l2 : loc
v1, v2 : val
______________________________________(1/1)
"Hl1" : l1 ↦ v2
"Hl2" : l2 ↦ v1
--------------------------------------∗
l1 ↦ v2 ∗ l2 ↦ v1
```

Proving swap using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
  wp_load; wp_let.
  wp_load.
  wp_store.
  wp_store.
  iFrame.
```

```
No more subgoals.
```

Proving swap using symbolic execution

```
Definition swap : val := λ: "x" "y",
  let: "tmp" := !"x" in
  "x" ← !"y";;
  "y" ← "tmp".

Lemma swap_spec l1 l2 v1 v2 :
  {{ l1 ↦ v1 ∗ l2 ↦ v2 }} swap #l1 #l2
  {{ _, l1 ↦ v2 ∗ l2 ↦ v1 }}.
Proof.
  iIntros "!# [Hl1 Hl2]".
  do 2 wp_let.
  wp_load; wp_let.
  wp_load.
  wp_store.
  wp_store.
  iFrame.
Qed.
```

Making IPM tactics modular using type classes

We want `iDestruct "H" as "[H1 H2]"` to:

- turn $\texttt{H} : \texttt{P} * \texttt{Q}$ into $\texttt{H1} : \texttt{P}$ and $\texttt{H2} : \texttt{Q}$
- turn $\texttt{H} : \triangleright(\texttt{P} * \texttt{Q})$ into $\texttt{H2} : \triangleright\ \texttt{P}$ and $\texttt{H2} : \triangleright\ \texttt{Q}$
- turn $\texttt{H} : \texttt{l} \mapsto \texttt{v}$ into $\texttt{H1} : \texttt{l} \overset{1/2}{\longmapsto} \texttt{v}$ and $\texttt{H2} : \texttt{l} \overset{1/2}{\longmapsto} \texttt{v}$

Making IPM tactics modular using type classes

We want `iDestruct "H" as "[H1 H2]"` to:

- turn `H : P ∗ Q` into `H1 : P` and `H2 : Q`
- turn `H : ▷(P ∗ Q)` into `H2 : ▷ P` and `H2 : ▷ Q`
- turn `H : l` $\mapsto$ `v` into `H1 : l` $\xmapsto{1/2}$ `v` and `H2 : l` $\xmapsto{1/2}$ `v`

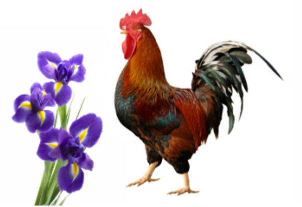
We use type classes to achieve that:

```
Class IntoAnd (p : bool) (P Q1 Q2 : uPred M) :=
  into_and : P ⊢ if p then Q1 ∧ Q2 else Q1 ∗ Q2.
Instance into_and_sep p P Q : IntoAnd p (P ∗ Q) P Q.
Instance into_and_and P Q : IntoAnd true (P ∧ Q) P Q.
Instance into_and_later p P Q1 Q2 : IntoAnd p P Q1 Q2 → IntoAnd p (▷
    P) (▷ Q1) (▷ Q2).
Instance into_and_mapsto l q v : IntoAnd false (l ↦{q} v) (l ↦{q/2} v)
      (l ↦{q/2} v).

Lemma tac_and_destruct Δ Δ' i p j1 j2 P P1 P2 Q :
  envs_lookup i Δ = Some (p, P) →
  IntoAnd p P P1 P2 →
  envs_simple_replace i p (Esnoc (Esnoc Enil j1 P1) j2 P2) Δ = Some Δ'
      →
  (Δ' ⊢ Q) → Δ ⊢ Q.
```

IPM in summary

- Contexts are deeply embedded
- Context manipulation is done via **computational reflection**
- IPM tactics are just Coq lemmas
- **Type classes** are used to make the tactics more general
- **Ltac** is used to provide an end-user syntax and error reporting



IPM in summary

- Contexts are deeply embedded
- Context manipulation is done via **computational reflection**
- IPM tactics are just Coq lemmas
- **Type classes** are used to make the tactics more general
- **Ltac** is used to provide an end-user syntax and error reporting

These ideas are hopefully applicable to other object logics

In the paper and Coq formalization

- Detailed description of the implementation
- Verification of concurrent algorithms using IPM
- Formalization of unary and binary logical relations
- Proving logical refinements

IPM scales

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 the Coq proof assistant with (spatial and non-spatial) named proof contexts for the object logic. We show that thanks to these contexts we can implement high-level tactics for introduction and elimination of the connectives of the object logic, and thereby make reasoning in the embedded logic as seamless as reasoning in the meta logic of

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. Therefore, to use a proof assistant to formalize reasoning in a program logic, one needs to represent the program logic in that proof assistant, and then, to benefit from the built-in features of the program logic, use the proof assistant to reason *in* the embedded logic.

Reasoning in an embedded logic using a proof assistant tradition-

Thank you!

Want a ‘proof mode’ for another logic, talk to us!

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