Higher-Order Ghost State

Ralf Jung, Robbert Krebbers, Lars Birkedal, Derek Dreyer

ICFP 2016 in Nara, Japan

Max Planck Institute for Software Systems (MPI-SWS), Aarhus University

- This talk is about Iris, a logic we want to apply to verify the safety of Rust.

- This talk is about Iris, a logic we want to apply to verify the safety of Rust.
- The key technical contribution is to show how to extend Iris with higher-order ghost state.

- This talk is about Iris, a logic we want to apply to verify the safety of Rust.
- The key technical contribution is to show how to extend Iris with higher-order ghost state.

The Rust Programming Language

The Rust Programming Language

Java

Go

Haskell

...

Focus on safety

The Rust Programming Language

Java

Go

Haskell

...

C

C++

Assembly

...

Focus on safety

Focus on control

The Rust Programming Language

- Higher-order functions
- Polymorphism / Generics
- Traits (typeclasses + associated types)

The Rust Programming Language

- Higher-order functions
- Polymorphism / Generics
- Traits (typeclasses + associated types)

- Control over memory allocation and data layout

The Rust Programming Language

- Higher-order functions
- Polymorphism / Generics
- Traits (typeclasses + associated types)

- Control over memory allocation and data layout
- Linear (ownership-based) type system with regions & region inference

The Rust Programming Language

- Higher-order functions
- Polymorphism / Generics
- Traits (typeclasses + associated types)
- Concurrency
- Control over memory allocation and data layout
- Linear (ownership-based) type system with regions & region inference

The Rust Programming Language

- H
- P
- T
  a
- C
- C
  a
- L
  type system with regions &
  region inference

Goal of RustBelt project:
Prove safety of language and its standard library.

Picking the right tool

Wanted:

program logic

Picking the right tool

Wanted:

separation logic

Picking the right tool

Wanted:

concurrent
separation logic

Picking the right tool

Wanted:

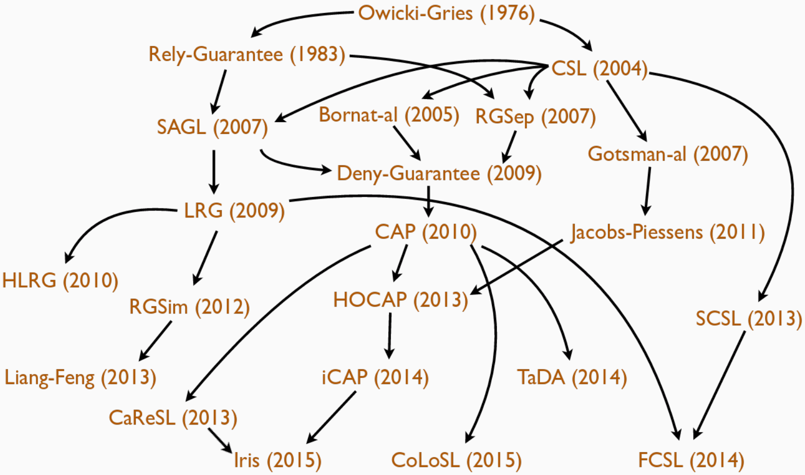
Higher-order
concurrent
separation logic

Picking the right tool

Wanted:

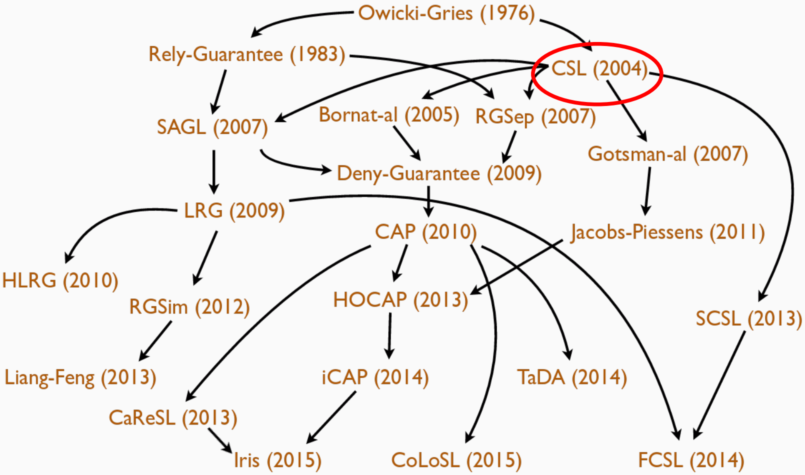
Higher-order concurrent separation logic

Concurrency Logics



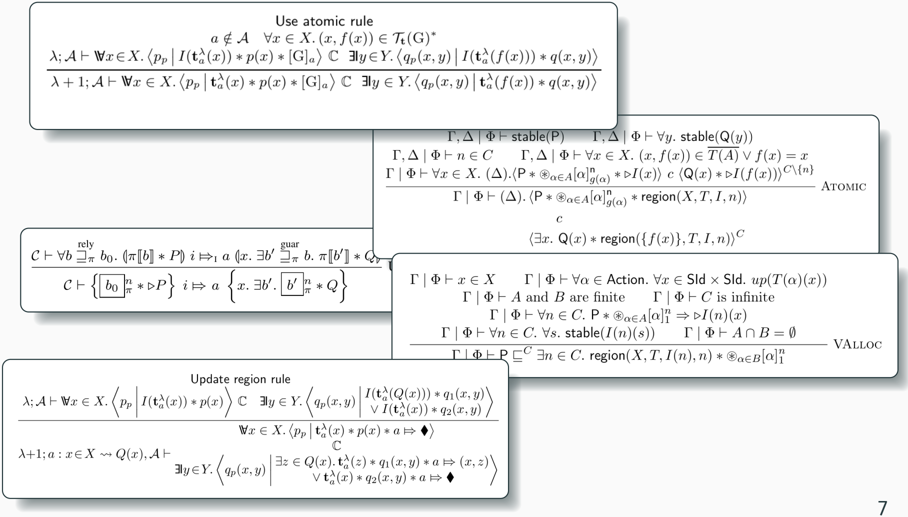
Picture by Ilya Sergey

Concurrency Logics



Picture by Ilya Sergey

Complex Foundations



Use atomic rule

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

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

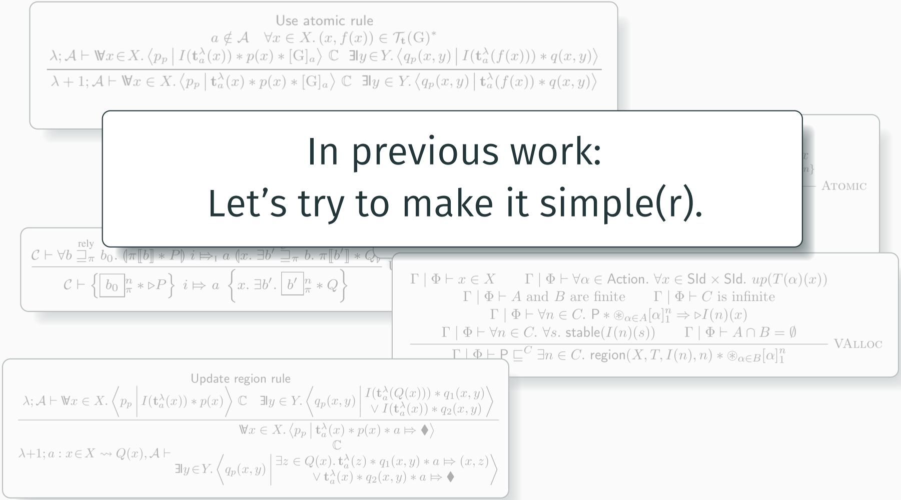
$$\dfrac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_{\pi}} b_0.\ (\!| \pi[\![b]\!] * P |\!)\ i \Rrightarrow_{\mathrm{I}} a\ (\!| x.\ \exists b' \overset{\text{guar}}{\sqsupseteq_{\pi}} b.\ \pi[\![b']\!] * Q}{\mathcal{C} \vdash \left\{ \boxed{b_0}^{n}_{\pi} * \triangleright P \right\}\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

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

Update region rule

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

Complex Foundations



Use atomic rule

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

ATOMIC

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

$$\frac{\begin{array}{c}\Gamma \mid \Phi \vdash x \in X \qquad \Gamma \mid \Phi \vdash \forall \alpha \in \mathsf{Action}.\, \forall x \in \mathsf{SId} \times \mathsf{SId}.\ up(T(\alpha)(x)) \\ \Gamma \mid \Phi \vdash A \text{ and } B \text{ are finite} \qquad \Gamma \mid \Phi \vdash C \text{ is infinite} \\ \Gamma \mid \Phi \vdash \forall n \in C.\ \mathsf{P} * \circledast_{\alpha \in A} [\alpha]_1^n \Rightarrow \triangleright I(n)(x) \\ \Gamma \mid \Phi \vdash \forall n \in C.\ \forall s.\ \mathsf{stable}(I(n)(s)) \qquad \Gamma \mid \Phi \vdash A \cap B = \emptyset \end{array}}{\Gamma \mid \Phi \vdash \mathsf{P} \sqsubseteq^{C} \exists n \in C.\ \mathsf{region}(X, T, I(n), n) * \circledast_{\alpha \in B} [\alpha]_1^n}\ \textsc{VAlloc}$$

Update region rule

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

In previous work:

Let’s try to make it simple(r).

Iris (POPL 2015) is built on two simple mechanisms:

- Invariants
- User-defined ghost state

Iris (POPL 2015) is built on two simple mechanisms:

- Invariants
- User-defined ghost state

Ghost State

Ghost state

Auxiliary program variables

(“ghost heap”)

Tokens / Capabilities

Monotone state

(*e.g.*, trace information)

Ghost State

Ghost state

Auxiliary program variables

(“

Toke

Monotone state

(*e.g.*, trace information)

User-defined ghost state:

Pick your favorite!

Ghost State

Common structure of ghost state:
Partial commutative monoid (PCM).

Gho

Aux

(“

Toke

Mor

(*e.g.*, trace information)

Ghost State

Gho

Aux

(“

Toke

Mor

(*e.g.*, trace information)

Common structure of ghost state:
Partial commutative monoid (PCM).

A PCM is a set M with an associative, commutative composition operation.

Ghost State

Ghost state	PCM composition
Auxiliary program variables (“ghost heap”)	Disjoint union
Tokens / Capabilities	No composition
Monotone state (*e.g.*, trace information)	Maximum

Iris: Resting on Simple Foundations

Invariants

Ghost state (any partial commutative monoid)

Iris: Resting on Simple Foundations

Invariants:

$$\frac{\{\triangleright I * P\}\ e\ \{\triangleright I * Q\}_{\mathcal{E}} \qquad \text{atomic(e)}}{\boxed{I}^{\iota} \vdash \{P\}\ e\ \{v.\ Q\}_{\mathcal{E} \uplus \{\iota\}}}$$

Ghost state (any partial commutative monoid):

$$\frac{\forall a_{\mathrm{f}}.\ a \mathrel{\#} a_{\mathrm{f}} \Rightarrow b \mathrel{\#} a_{\mathrm{f}}}{\boxed{a} \Rrightarrow \boxed{b}} \qquad \frac{a \cdot b = c}{\boxed{a} * \boxed{b} \Leftrightarrow \boxed{c}}$$

$$\boxed{a} \Rightarrow \mathcal{V}(a)$$

Complex Foundations

Use atomic rule

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

With Iris, we can derive the more complex reasoning principles from the simple foundations.

x

$n\}$

ATOMIC

$$\frac{\mathcal{C} \vdash \forall b \overset{\text{rely}}{\sqsupseteq_{\pi}}}{\mathcal{C} \vdash \left\{ \boxed{b_0}^{n}_{\pi} * \triangleright P \right\} \ i \Rrightarrow a \ \left\{ x.\, \exists b'.\, \boxed{b'}^{n}_{\pi} * Q \right\}}$$

$$\frac{\Gamma \mid \Phi \vdash x \in X \qquad \Gamma \mid \Phi \vdash \forall \alpha \in \mathsf{Action}.\ \forall x \in \mathsf{Sld} \times \mathsf{Sld}.\ up(T(\alpha)(x)) \\ \Gamma \mid \Phi \vdash A \text{ and } B \text{ are finite} \qquad \Gamma \mid \Phi \vdash C \text{ is infinite} \\ \Gamma \mid \Phi \vdash \forall n \in C.\ \mathsf{P} * \circledast_{\alpha \in A}[\alpha]_1^n \Rightarrow \triangleright I(n)(x) \\ \Gamma \mid \Phi \vdash \forall n \in C.\ \forall s.\ \mathsf{stable}(I(n)(s)) \qquad \Gamma \mid \Phi \vdash A \cap B = \emptyset}{\Gamma \mid \Phi \vdash \mathsf{P} \sqsubseteq^{C} \exists n \in C.\ \mathsf{region}(X, T, I(n), n) * \circledast_{\alpha \in B}[\alpha]_1^n} \ \text{VALLOC}$$

Update region rule

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

Iris: Resting on Simple Foundations

Invariants:

$$\frac{\{\triangleright I * P\}\ e\ \{\triangleright I * Q\}_{\mathcal{E}} \qquad \text{atomic(e)}}{}$$

For specifying some synchronization primitives, these foundations are not enough!

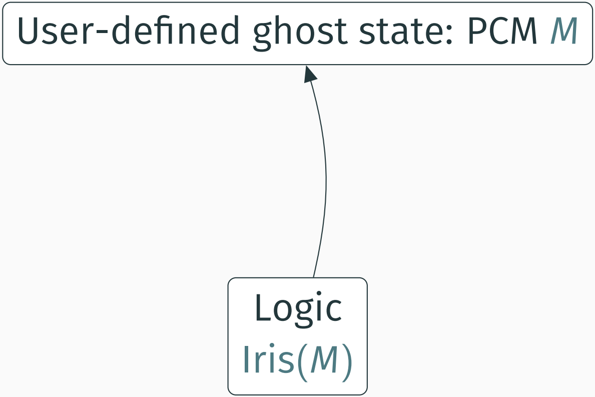
$$\frac{\forall a_{\mathrm{f}}.\ a \# a_{\mathrm{f}} \Rightarrow b \# a_{\mathrm{f}}}{\boxed{a} \Rrightarrow \boxed{b}} \qquad \frac{a \cdot b = c}{\boxed{a} * \boxed{b} \Leftrightarrow \boxed{c}}$$

$$\boxed{a} \Rightarrow \mathcal{V}(a)$$

First-Order Ghost State

User-defined ghost state: PCM M

First-Order Ghost State

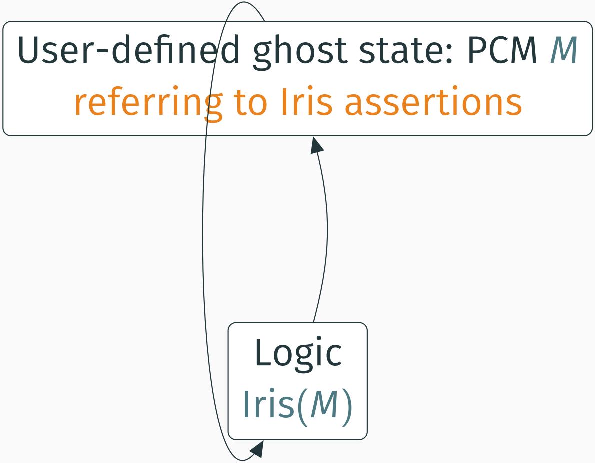


User-defined ghost state: PCM M

Logic

Iris(M)

Higher-Order Ghost State



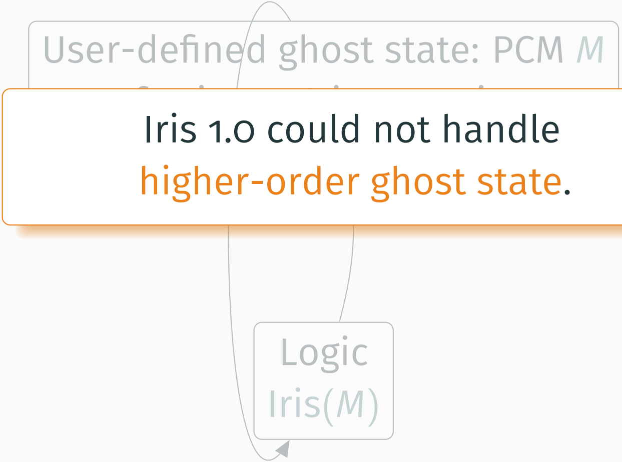
User-defined ghost state: PCM M

referring to Iris assertions

Logic

Iris(M)

Higher-Order Ghost State



User-defined ghost state: PCM M

Logic

Iris(M)

Iris 1.0 could not handle
higher-order ghost state.

Contributions

- Motivate why higher-order ghost state is useful.
- Demonstrate how to extend Iris to support higher-order ghost state.

Contributions

- Motivate why higher-order ghost state is useful.
- Demonstrate how to extend Iris to support higher-order ghost state.

Contributions

- Motivate why higher-order ghost state is useful.
- Demonstrate how to extend Iris to support higher-order ghost state.

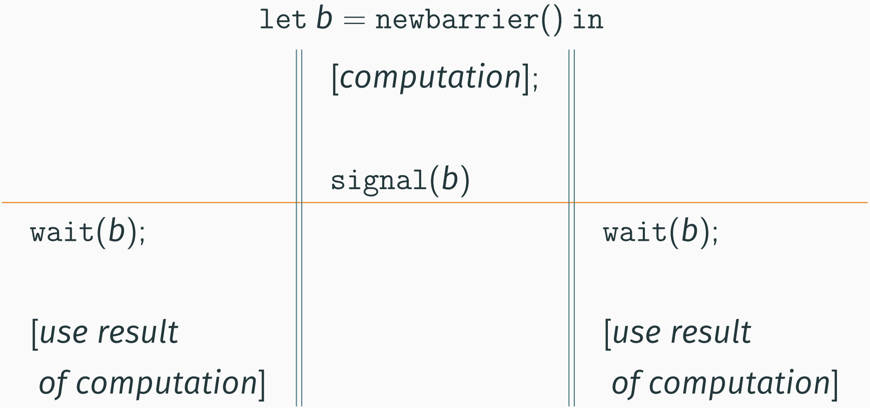
Contributions

- Motivate why higher-order ghost state is useful.
- Demonstrate how to extend Iris to support higher-order ghost state.

Barrier

```
                      let b = newbarrier() in
                        ||  [computation];  ||
                        ||                  ||
                        ||  signal(b)       ||
wait(b);                ||                  ||  wait(b);
                        ||                  ||
[use result             ||                  ||  [use result
 of computation]        ||                  ||   of computation]
```

Barrier



`let` b = `newbarrier()` `in`

`wait(`b`);`

[*use result of computation*]

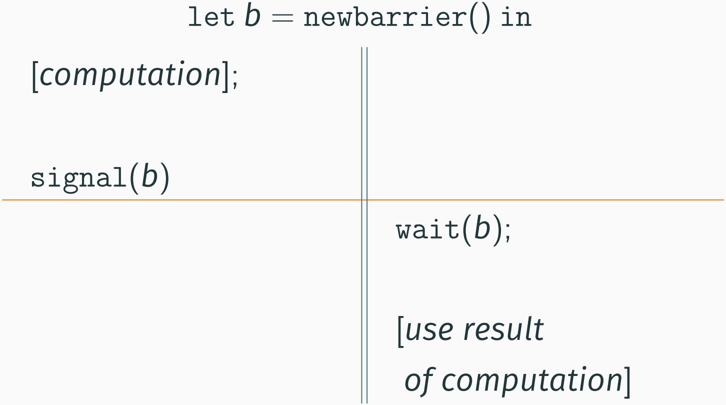
[*computation*];

`signal(`b`)`

`wait(`b`);`

[*use result of computation*]

Barrier (simple version)



$\texttt{let}\ b = \texttt{newbarrier}()\ \texttt{in}$

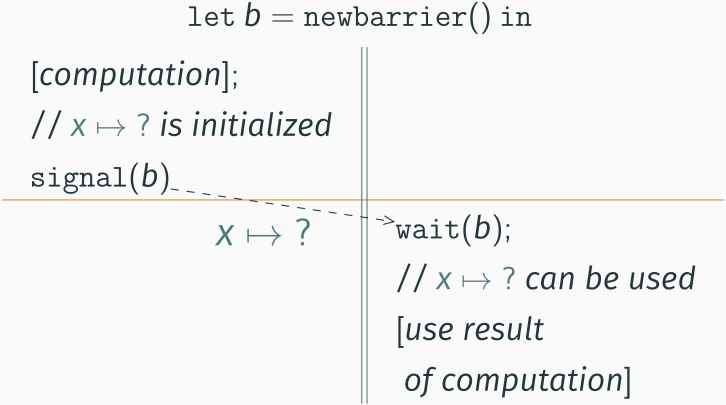
[*computation*];

$\texttt{signal}(b)$

$\texttt{wait}(b)$;

[*use result*
of computation]

Barrier (simple version)



`let` $b =$ `newbarrier()` `in`

[*computation*];

// $x \mapsto ?$ *is initialized*

`signal`(b)

$x \mapsto ?$

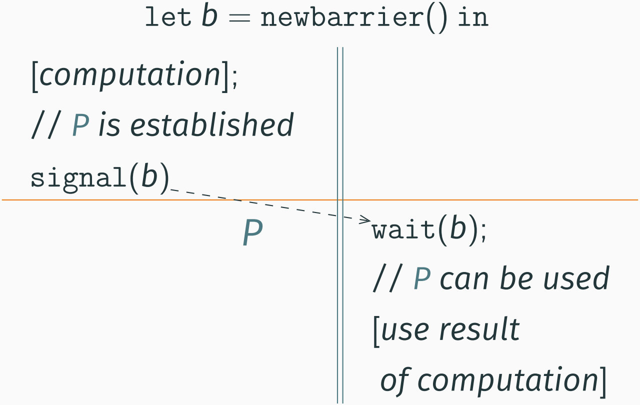
`wait`(b);

// $x \mapsto ?$ *can be used*

[*use result*

of computation]

Barrier (simple version)



`let` b = `newbarrier()` `in`

[*computation*];

// P is established

`signal`(b)

P

`wait`(b);

// P can be used

[*use result*
 of computation]

Barrier (simple version)

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

$$\texttt{let } b = \texttt{newbarrier}()$$

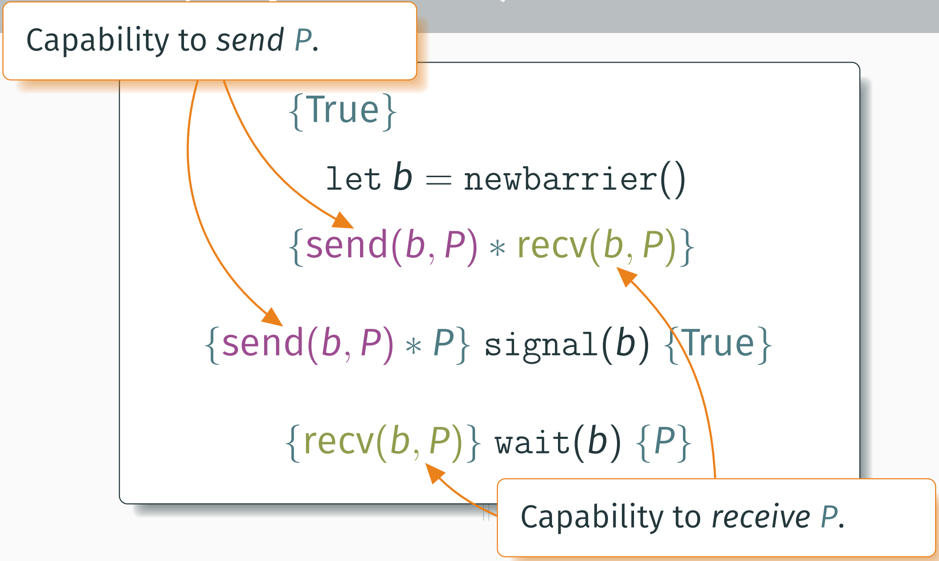
$$\{\mathsf{send}(b, P) * \mathsf{recv}(b, P)\}$$

$$\{\mathsf{send}(b, P) * P\}\ \texttt{signal}(b)\ \{\mathsf{True}\}$$

$$\{\mathsf{recv}(b, P)\}\ \texttt{wait}(b)\ \{P\}$$

|| of computation]

Barrier (simple version)



Capability to *send* P.

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

$$\texttt{let}\ b = \texttt{newbarrier}()$$

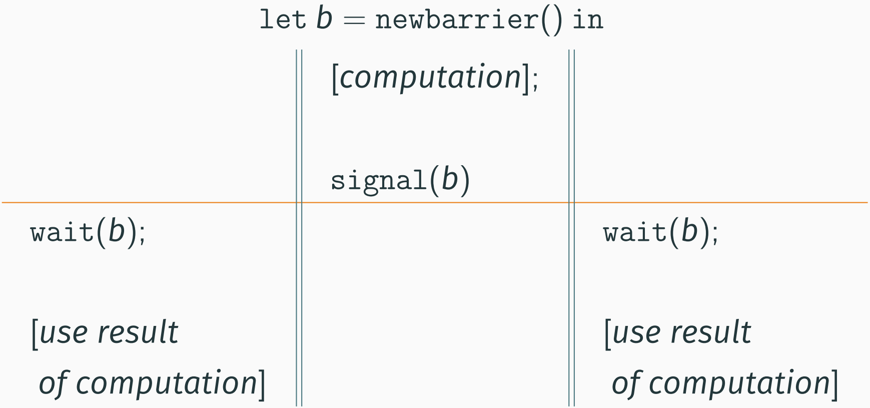
$$\{\mathsf{send}(b, P) * \mathsf{recv}(b, P)\}$$

$$\{\mathsf{send}(b, P) * P\}\ \texttt{signal}(b)\ \{\mathsf{True}\}$$

$$\{\mathsf{recv}(b, P)\}\ \texttt{wait}(b)\ \{P\}$$

Capability to *receive* P.

Barrier



`let` b = `newbarrier()` `in`

[*computation*];

`signal`(b)

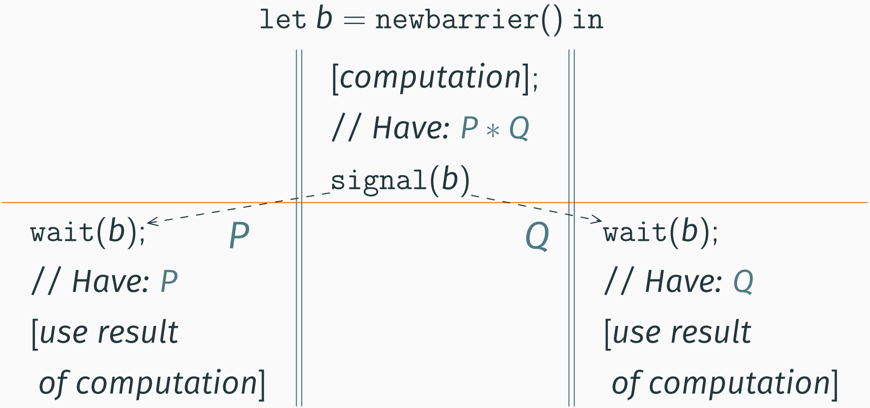
`wait`(b);

[*use result of computation*]

`wait`(b);

[*use result of computation*]

Barrier



`let` $b =$ `newbarrier()` `in`

[*computation*];

// Have: $P * Q$

`signal`(b)

`wait`(b);

// Have: P

[*use result*

of computation]

`wait`(b);

// Have: Q

[*use result*

of computation]

Barri

$$\{\mathrm{True}\}$$

$$\texttt{let}\ b = \texttt{newbarrier}()$$

$$\{\mathrm{send}(b, P) * \mathrm{recv}(b, P)\}$$

$$\{\mathrm{send}(b, P) * P\}\ \texttt{signal}(b)\ \{\mathrm{True}\}$$

$$\{\mathrm{recv}(b, P)\}\ \texttt{wait}(b)\ \{P\}$$

$$\mathrm{recv}(b, P * Q) \Rrightarrow \mathrm{recv}(b, P) * \mathrm{recv}(b, Q)$$

wait

// H

[use

of c

Barrier: A little history

- Spec first proposed by Mike Dodds *et al.* (2011)

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

$$\texttt{let } b = \texttt{newbarrier}()$$

$$\{\mathsf{send}(b, P) * \mathsf{recv}(b, P)\}$$

$$\{\mathsf{send}(b, P) * P\}\ \texttt{signal}(b)\ \{\mathsf{True}\}$$

$$\{\mathsf{recv}(b, P)\}\ \texttt{wait}(b)\ \{P\}$$

$$\mathsf{recv}(b, P * Q) \Rrightarrow \mathsf{recv}(b, P) * \mathsf{recv}(b, Q)$$

Barrier: A little history

- Spec first proposed by Mike Dodds *et al.* (2011)
- First proof later found to be flawed
- Fixed using named propositions

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

$$\texttt{let}\ b = \texttt{newbarrier}()$$

$$\{\mathsf{send}(b, P) * \mathsf{recv}(b, P)\}$$

$$\{\mathsf{send}(b, P) * P\}\ \texttt{signal}(b)\ \{\mathsf{True}\}$$

$$\{\mathsf{recv}(b, P)\}\ \texttt{wait}(b)\ \{P\}$$

$$\mathsf{recv}(b, P * Q) \Rrightarrow \mathsf{recv}(b, P) * \mathsf{recv}(b, Q)$$

Named Propositions

Gives a fresh name γ to P.

$$\forall P.\ \mathsf{True} \Rrightarrow \exists \gamma.\ \gamma \mapsto P$$

Named Propositions

Gives a fresh name γ to P.

P does not have to hold!

$$\forall P.\ \mathsf{True} \Rrightarrow \exists \gamma.\ \gamma \mapsto P$$

Named Propositions

Gives a fresh name γ to P.
P does not have to hold!

$$\forall P.\ \mathsf{True} \Rrightarrow \exists \gamma.\ \gamma \mapsto P$$

$$\forall \gamma, P, Q.\ (\gamma \mapsto P * \gamma \mapsto Q) \Rightarrow (P \Leftrightarrow Q)$$

Agreement about proposition named γ.

Named Propositions

Gives a fresh name γ to P

Derive named propositions from lower-level principles:

Agreement about proposition named γ.

Named Propositions

Gives a fresh name γ to P

Derive named propositions from lower-level principles:

Build named propositions on ghost state.

Agreement about proposition named γ.

Named Propositions

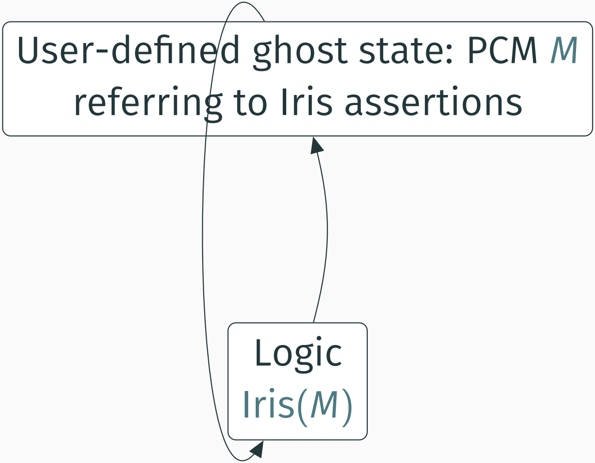
Gives a fresh name γ to P.
Allocates new slot in “table”.

$$\forall P.\ \text{True} \Rrightarrow \exists \gamma.\ \gamma \mapsto P$$

$$\forall \gamma, P, Q.\ (\gamma \mapsto P * \gamma \mapsto Q) \Rightarrow (P \Leftrightarrow Q)$$

Agreement about row γ of the “table”.

Higher-Order Ghost State



User-defined ghost state: PCM M referring to Iris assertions

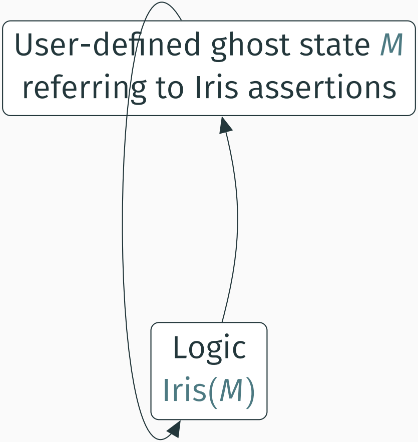
Logic

Iris(M)

Contributions

- Motivate why higher-order ghost state is useful.
- Demonstrate how to extend Iris to support higher-order ghost state.

Higher-Order Ghost State: Technicalities



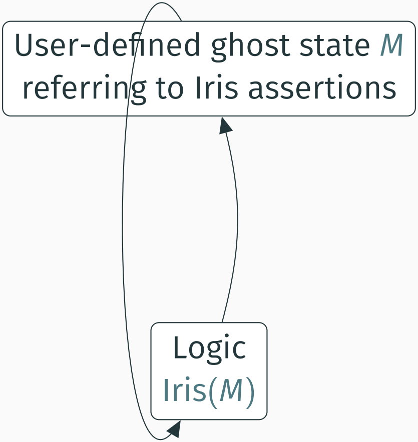
User-defined ghost state M referring to Iris assertions

Logic Iris(M)

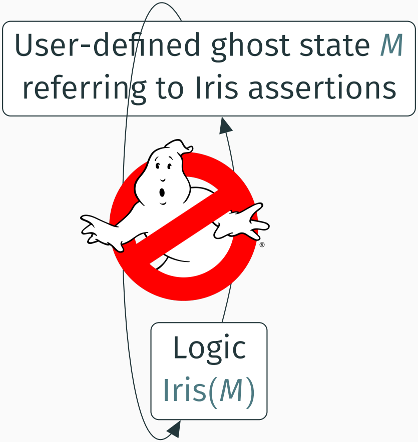
Higher-Order Ghost State: Technicalities

We got a problem with our ghost state.

Who we gonna call?



Higher-Order Ghost State: Technicalities

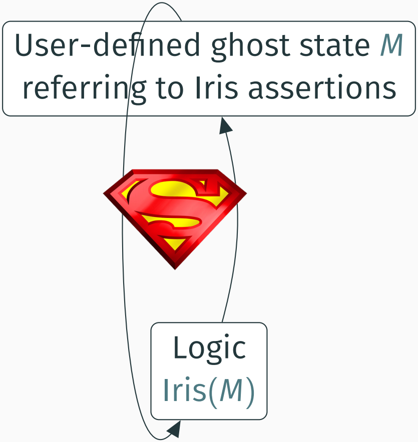


User-defined ghost state *M* referring to Iris assertions

Logic Iris(*M*)

Higher-Order Ghost State: Technicalities

Step-Indexing



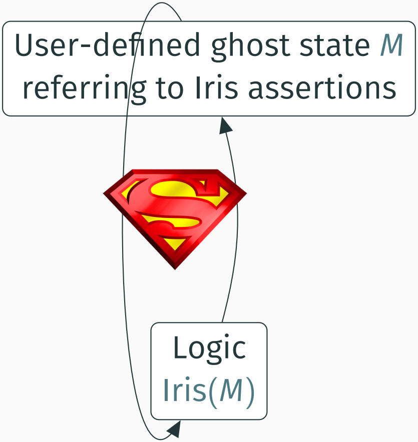
User-defined ghost state M referring to Iris assertions

Logic Iris(M)

Higher-Order Ghost State: Technicalities

Step-Indexing

- Introduced 2001 by Appel and McAllester
- Used to solve circularities in models of higher-order state



User-defined ghost state M referring to Iris assertions

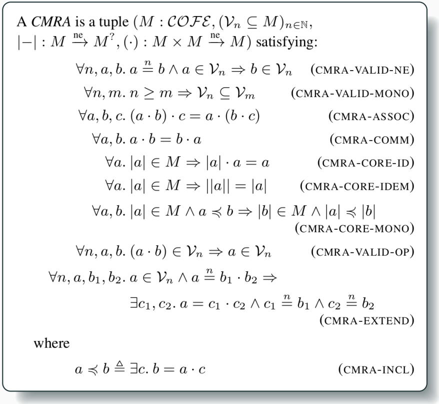
Logic Iris(M)

Higher-Order Ghost State: Technicalities

- Equip PCMs with a “step-indexing structure”.

Higher-Order Ghost State: Technicalities

- Equip PCMs with a “step-indexing structure”.
  $\rightarrow$ CMRA



A *CMRA* is a tuple $(M : \mathcal{COFE}, (\mathcal{V}_n \subseteq M)_{n\in\mathbb{N}},$
$|{-}| : M \xrightarrow{\text{ne}} M^?, (\cdot) : M \times M \xrightarrow{\text{ne}} M)$ satisfying:

$$\forall n, a, b.\ a \overset{n}{=} b \wedge a \in \mathcal{V}_n \Rightarrow b \in \mathcal{V}_n \qquad (\text{CMRA-VALID-NE})$$

$$\forall n, m.\ n \geq m \Rightarrow \mathcal{V}_n \subseteq \mathcal{V}_m \qquad (\text{CMRA-VALID-MONO})$$

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad (\text{CMRA-ASSOC})$$

$$\forall a, b.\ a \cdot b = b \cdot a \qquad (\text{CMRA-COMM})$$

$$\forall a.\ |a| \in M \Rightarrow |a| \cdot a = a \qquad (\text{CMRA-CORE-ID})$$

$$\forall a.\ |a| \in M \Rightarrow ||a|| = |a| \qquad (\text{CMRA-CORE-IDEM})$$

$$\forall a, b.\ |a| \in M \wedge a \preccurlyeq b \Rightarrow |b| \in M \wedge |a| \preccurlyeq |b| \qquad (\text{CMRA-CORE-MONO})$$

$$\forall n, a, b.\ (a \cdot b) \in \mathcal{V}_n \Rightarrow a \in \mathcal{V}_n \qquad (\text{CMRA-VALID-OP})$$

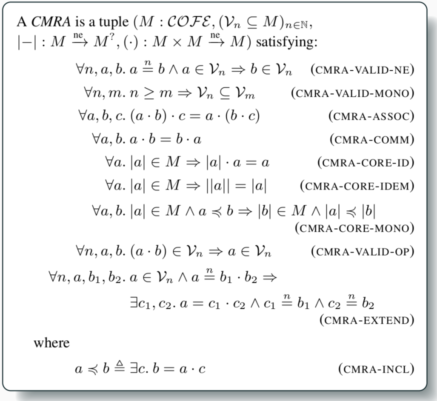
$$\forall n, a, b_1, b_2.\ a \in \mathcal{V}_n \wedge a \overset{n}{=} b_1 \cdot b_2 \Rightarrow$$
$$\exists c_1, c_2.\ a = c_1 \cdot c_2 \wedge c_1 \overset{n}{=} b_1 \wedge c_2 \overset{n}{=} b_2 \qquad (\text{CMRA-EXTEND})$$

where

$$a \preccurlyeq b \triangleq \exists c.\ b = a \cdot c \qquad (\text{CMRA-INCL})$$

Higher-Order Ghost State: Technicalities

- Equip PCMs with a “step-indexing structure”. $\rightarrow$ CMRA
- Let user define a functor yielding a CMRA.



A *CMRA* is a tuple $(M : \mathcal{COFE}, (\mathcal{V}_n \subseteq M)_{n\in\mathbb{N}}, |-| : M \overset{\text{ne}}{\longrightarrow} M^?, (\cdot) : M \times M \overset{\text{ne}}{\longrightarrow} M)$ satisfying:

$$\forall n, a, b.\ a \overset{n}{=} b \land a \in \mathcal{V}_n \Rightarrow b \in \mathcal{V}_n \tag{CMRA-VALID-NE}$$

$$\forall n, m.\ n \geq m \Rightarrow \mathcal{V}_n \subseteq \mathcal{V}_m \tag{CMRA-VALID-MONO}$$

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \tag{CMRA-ASSOC}$$

$$\forall a, b.\ a \cdot b = b \cdot a \tag{CMRA-COMM}$$

$$\forall a.\ |a| \in M \Rightarrow |a| \cdot a = a \tag{CMRA-CORE-ID}$$

$$\forall a.\ |a| \in M \Rightarrow ||a|| = |a| \tag{CMRA-CORE-IDEM}$$

$$\forall a, b.\ |a| \in M \land a \preccurlyeq b \Rightarrow |b| \in M \land |a| \preccurlyeq |b| \tag{CMRA-CORE-MONO}$$

$$\forall n, a, b.\ (a \cdot b) \in \mathcal{V}_n \Rightarrow a \in \mathcal{V}_n \tag{CMRA-VALID-OP}$$

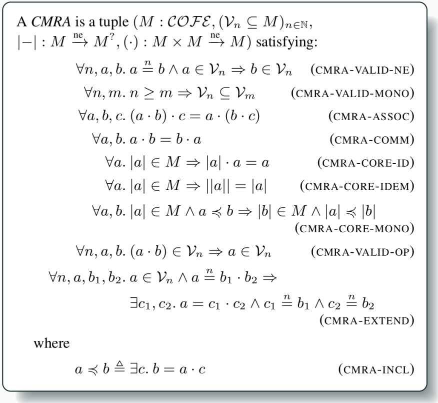
$$\forall n, a, b_1, b_2.\ a \in \mathcal{V}_n \land a \overset{n}{=} b_1 \cdot b_2 \Rightarrow \exists c_1, c_2.\ a = c_1 \cdot c_2 \land c_1 \overset{n}{=} b_1 \land c_2 \overset{n}{=} b_2 \tag{CMRA-EXTEND}$$

where

$$a \preccurlyeq b \triangleq \exists c.\ b = a \cdot c \tag{CMRA-INCL}$$

Higher-Order Ghost State: Technicalities

- Equip PCMs with a “step-indexing structure”. → CMRA
- Let user define a functor yielding a CMRA.
- Tie the knot by taking a fixed-point.



A *CMRA* is a tuple $(M : \mathcal{COFE}, (\mathcal{V}_n \subseteq M)_{n \in \mathbb{N}}, |{-}| : M \xrightarrow{\text{ne}} M^?, (\cdot) : M \times M \xrightarrow{\text{ne}} M)$ satisfying:

$$\forall n, a, b.\ a \overset{n}{=} b \wedge a \in \mathcal{V}_n \Rightarrow b \in \mathcal{V}_n \qquad \text{(CMRA-VALID-NE)}$$

$$\forall n, m.\ n \geq m \Rightarrow \mathcal{V}_n \subseteq \mathcal{V}_m \qquad \text{(CMRA-VALID-MONO)}$$

$$\forall a, b, c.\ (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad \text{(CMRA-ASSOC)}$$

$$\forall a, b.\ a \cdot b = b \cdot a \qquad \text{(CMRA-COMM)}$$

$$\forall a.\ |a| \in M \Rightarrow |a| \cdot a = a \qquad \text{(CMRA-CORE-ID)}$$

$$\forall a.\ |a| \in M \Rightarrow ||a|| = |a| \qquad \text{(CMRA-CORE-IDEM)}$$

$$\forall a, b.\ |a| \in M \wedge a \preccurlyeq b \Rightarrow |b| \in M \wedge |a| \preccurlyeq |b| \qquad \text{(CMRA-CORE-MONO)}$$

$$\forall n, a, b.\ (a \cdot b) \in \mathcal{V}_n \Rightarrow a \in \mathcal{V}_n \qquad \text{(CMRA-VALID-OP)}$$

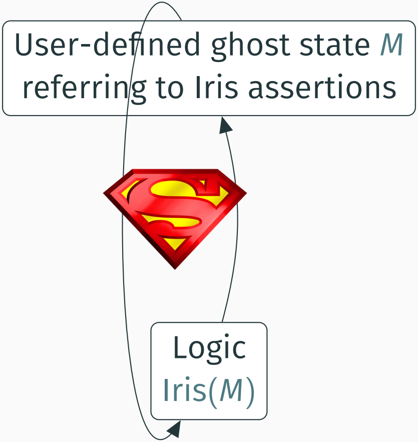
$$\forall n, a, b_1, b_2.\ a \in \mathcal{V}_n \wedge a \overset{n}{=} b_1 \cdot b_2 \Rightarrow \exists c_1, c_2.\ a = c_1 \cdot c_2 \wedge c_1 \overset{n}{=} b_1 \wedge c_2 \overset{n}{=} b_2 \qquad \text{(CMRA-EXTEND)}$$

where

$$a \preccurlyeq b \triangleq \exists c.\ b = a \cdot c \qquad \text{(CMRA-INCL)}$$

Higher-Order Ghost State: Technicalities

- Equip PCMs with a “step-indexing structure”.
  → CMRA
- Let user define a functor yielding a CMRA.
- Tie the knot by taking a fixed-point.



User-defined ghost state M referring to Iris assertions

Logic Iris(M)

Named Propositions

$$\forall P.\ \mathsf{True} \Rrightarrow \exists \gamma.\ \gamma \mapsto P$$

$$\forall \gamma, P, Q.\ (\gamma \mapsto P * \gamma \mapsto Q) \Rightarrow \triangleright(P \Leftrightarrow Q)$$

Agreement about proposition named γ only holds at the next step-index.

Barri

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

$$\texttt{let}\ b = \texttt{newbarrier}()$$

$$\{\mathsf{send}(b, P) * \mathsf{recv}(b, P)\}$$

$$\{\mathsf{send}(b, P) * P\}\ \texttt{signal}(b)\ \{\mathsf{True}\}$$

$$\{\mathsf{recv}(b, P)\}\ \texttt{wait}(b)\ \{P\}$$

$$\mathsf{recv}(b, P * Q) \Rrightarrow \mathsf{recv}(b, P) * \mathsf{recv}(b, Q)$$

$\{\mathsf{rec}$

wait

$\{P\}$

[*use result*

$\{Q\}$

[*use result*

Iris: Resting on Simple Foundations

Invariants:

$$\frac{\{\triangleright I * P\}\ e\ \{\triangleright I * Q\}_{\mathcal{E}} \qquad \text{atomic(e)}}{\boxed{I}^{\iota} \vdash \{P\}\ e\ \{v.\ Q\}_{\mathcal{E} \uplus \{\iota\}}}$$

Ghost state (any CMRA):

$$\frac{\forall a_{\mathrm{f}}, n.\ a \#_n a_{\mathrm{f}} \Rightarrow b \#_n a_{\mathrm{f}}}{\boxed{a} \Rrightarrow \boxed{b}} \qquad \frac{a \cdot b = c}{\boxed{a} * \boxed{b} \Leftrightarrow \boxed{c}}$$

$$\boxed{a} \Rightarrow \mathcal{V}(a)$$

Iris: Resting on Simple Foundations

Invariants:

$$\frac{\{\triangleright I * P\}\ e\ \{\triangleright I * Q\}_{\mathcal{E}} \qquad \text{atomic(e)}}{}$$

Any PCM can be lifted to a CMRA, so all the old reasoning remains valid.

Gh

$$\frac{\forall a_{\mathrm{f}}, n.\ a \#_n a_{\mathrm{f}} \Rightarrow b \#_n a_{\mathrm{f}}}{\boxed{a} \Rrightarrow \boxed{b}} \qquad \frac{a \cdot b = c}{\boxed{a} * \boxed{b} \Leftrightarrow \boxed{c}}$$

$$\boxed{a} \Rightarrow \mathcal{V}(a)$$

What else?

- Other examples of higher-order ghost state

What else?

- Other examples of higher-order ghost state
- How to simplify the model with CMRAs

What else?

- Other examples of higher-order ghost state
- How to simplify the model with CMRAs
- Coq formalization

What else?

- Other examples of higher-order ghost state
- How to simplify the model with CMRAs
- Coq formalization

Ongoing work:

- *Encode* invariants using higher-order ghost state
- Applying named propositions in the safety proof of Rust

What else?

- Other examples of higher-order ghost state
- How to simplify the model with CMRAs
- Coq formalization

Ongoing work:

- *Encode* invariants using higher-order ghost state
- Applying named propositions in the safety proof of Rust

Thank you for your attention!