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

Focus on safety
Focus on control

/three.osf
The Rust Programming Language

Focus on safety
The Rust Programming Language

Java
Go
Haskell

Focus on safety

C
C++
Assembly

Focus on control
The Rust Programming Language

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

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

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

Higher-order concurrent separation logic
Concurrency Logics

- Owicki-Gries (1976)
- Rely-Guarantee (1983)
- CSL (2004)
- SAGL (2007)
- Bornat-al (2005)
- RGSep (2007)
- Deny-Guarantee (2009)
- Gotsman-al (2007)
- LRG (2009)
- Jacobs-Piessens (2011)
- HLRG (2010)
- CAP (2010)
- SCSL (2013)
- RGSim (2012)
- HOCCAP (2013)
- Liang-Feng (2013)
- iCAP (2014)
- TaDA (2014)
- CaReSL (2013)
- Iris (2015)
- CoLoSL (2015)
- FCSL (2014)
Use atomic rule

\[
\lambda; \mathcal{A} \vdash \forall x \in X. (x, f(x)) \in T_{\ell}(G)^* \\
\lambda + 1; \mathcal{A} \vdash \forall x \in X. \langle p_p \mid I(t_{\alpha}^A(x)) \ast p(x) \ast [G]_{\alpha} \rangle \subseteq \exists y \in Y. \langle q_p(x, y) \mid I(t_{\alpha}^A(f(x))) \ast q(x, y) \rangle
\]

\[
\lambda; \mathcal{A} \vdash \forall x \in X. \langle p_p \mid t_{\alpha}^A(x) \ast p(x) \ast [G]_{\alpha} \rangle \subseteq \exists y \in Y. \langle q_p(x, y) \mid t_{\alpha}^A(f(x)) \ast q(x, y) \rangle
\]

\[
\Gamma, \Delta \vdash \Phi \rightarrow \text{stable}(P) \quad \Gamma, \Delta \vdash \Phi \rightarrow \forall y. \text{stable}(Q(y))
\]

\[
\Gamma, \Delta \vdash \Phi \vdash n \in C' \quad \Gamma, \Delta \vdash \Phi \vdash \forall x \in X. (x, f(x)) \in T(A) \lor f(x) = x
\]

\[
\Gamma \mid \Phi \vdash \forall x \in X. \langle P \ast \oplus_{\alpha \in A}[\alpha]_{g(\alpha)} \ast \triangleright I(x) \rangle \quad \exists \langle Q(x) \ast \triangleright I(f(x)) \rangle \subseteq \{n\}
\]

\[
\Gamma \mid \Phi \vdash (\Delta). \langle P \ast \oplus_{\alpha \in A}[\alpha]_{g(\alpha)} \ast \text{region}(X, T, I, n) \rangle \quad \text{Atomic}
\]

\[
\Gamma \mid \Phi \vdash x \in X \quad \Gamma \mid \Phi \vdash \forall \alpha \in \text{Action}. \forall x \in \text{Sld} \times \text{Sld}. \text{up}(T(\alpha)(x))
\]

\[
\Gamma \mid \Phi \vdash A \land B \text{ are finite} \quad \Gamma \mid \Phi \vdash C \text{ is infinite}
\]

\[
\Gamma \mid \Phi \vdash \forall n \in C. \exists s. \text{stable}(I(n)(s)) \quad \Gamma \mid \Phi \vdash A \cap B = \emptyset
\]

\[
\Gamma \mid \Phi \vdash P \subseteq C \exists n \in C. \text{region}(X, T, I(n), n) \ast \oplus_{\alpha \in B} [\alpha]_{1} \quad \text{VALLOC}
\]

Update region rule

\[
\lambda; \mathcal{A} \vdash \forall x \in X. \langle p_p \mid I(t_{\alpha}^A(x)) \ast p(x) \ast q(x, y) \rangle \subseteq \exists y \in Y. \langle q_p(x, y) \mid I(t_{\alpha}^A(Q(x))) \ast q_1(x, y) \ast I(t_{\alpha}^A(x)) \ast q_2(x, y) \rangle
\]

\[
\lambda + 1; \mathcal{A} \vdash \forall x \in X \rightsquigarrow Q(x), \mathcal{A} \vdash \exists y \in Y. \langle q_p(x, y) \mid \exists z \in Q(x). t_{\alpha}^A(z) \ast q_1(x, y) \ast a \Rightarrow (x, z) \rangle
\]

\[
\mathcal{A} \vdash \forall x \in X. \langle p_p \mid t_{\alpha}^A(x) \ast p(x) \ast a \Rightarrow \diamond \rangle
\]

\[
\lambda + 1; \mathcal{A} \vdash \exists y \in Y. \langle q_p(x, y) \mid \exists z \in Q(x). t_{\alpha}^A(z) \ast q_1(x, y) \ast a \Rightarrow (x, z) \rangle
\]
In previous work: Let’s try to make it simple(r).

Use atomic rule

\[
\lambda; A \vdash \forall x \in X. \langle p_p | I(t^A_n(x)) \ast p(x) \ast [G]_\alpha \rangle \subseteq \exists y \in Y. \langle q_p(x,y) | I(t^A_n(f(x))) \ast q(x,y) \rangle
\]

\[
\lambda + 1; A \vdash \forall x \in X. \langle p_p | t^A_n(x) \ast p(x) \ast [G]_\alpha \rangle \subseteq \exists y \in Y. \langle q_p(x,y) | t^A_n(f(x)) \ast q(x,y) \rangle
\]
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
("ghost heap")

Tokens / Capabilities

Monotone state
(e.g., trace information)

User-defined ghost state:
Pick your favorite!
Common structure of ghost state: **Partial commutative monoid (PCM).**
Common structure of ghost state: **Partial commutative monoid (PCM).**

A PCM is a set $M$ with an associative, commutative composition operation.
## Ghost State

<table>
<thead>
<tr>
<th>Ghost state</th>
<th>PCM composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auxiliary program variables</td>
<td>Disjoint union</td>
</tr>
<tr>
<td>(“ghost heap”)</td>
<td></td>
</tr>
<tr>
<td>Tokens / Capabilities</td>
<td>No composition</td>
</tr>
<tr>
<td>Monotone state</td>
<td>Maximum</td>
</tr>
<tr>
<td>(e.g., trace information)</td>
<td></td>
</tr>
</tbody>
</table>
Invariants

Ghost state (any partial commutative monoid)
Invariants:

\[
\{ \triangledown I \ast P \} e \quad \{ \triangledown I \ast Q \} \varepsilon \quad \text{atomic(e)}
\]

\[
\overline{I}^I \vdash \{ P \} e \quad \{ v. Q \} \varepsilon \cup \{ I \}
\]

Ghost state (any partial commutative monoid):

\[
\forall a_f. \quad a \neq a_f \implies b \neq a_f
\]

\[
\frac{a \Rightarrow b}{a \Rightarrow \forall (a)}
\]

\[
a \ast b = c
\]

\[
\frac{a \ast b \iff c}{a \Rightarrow \forall (a)}
\]
With Iris, we can derive the more complex reasoning principles from the simple foundations.
For specifying some synchronization primitives, these foundations are not enough!

\[ \forall a_f. a \# a_f \Rightarrow b \# a_f \]

\[ a \Rightarrow b \]

\[ a \cdot b = c \]

\[ a \ast b \leftrightarrow c \]

\[ a \Rightarrow \forall (a) \]
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 $M$

Iris /one.osf./zero.osf 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.
let $b = \text{newbarrier}()$ in

\[
\begin{aligned}
&\text{[computation];} \\
&\text{signal}(b) \\
&\text{wait}(b); \\
&\text{[use result of computation]} \\
&\text{wait}(b); \\
&\text{[use result of computation]}
\end{aligned}
\]
let \( b = \text{newbarrier}() \) in

\[
\begin{align*}
\text{[computation]} &; \\
\text{signal}(b) &; \\
\text{wait}(b) &; \\
\text{[use result of computation]} &; \\
\end{align*}
\]

\[
\begin{align*}
\text{[computation]} &; \\
\text{signal}(b) &; \\
\text{wait}(b) &; \\
\text{[use result of computation]} &; \\
\end{align*}
\]
let $b = \text{newbarrier}()$ in

[computation];

signal($b$)

[use result of computation]

wait($b$);
let \( b = \text{newbarrier}() \) in

\[
\text{[computation]};
\]

// \( x \mapsto ? \) is initialized

\[
\text{signal}(b);
\]

\[
\text{wait}(b);
\]

// \( x \mapsto ? \) can be used

\[
\text{[use result of computation]}
\]
let $b = \text{newbarrier}()$ in

[computation];

// $P$ is established
signal($b$)

$P$

wait($b$);

// $P$ can be used
[use result of computation]
Barrier (simple version)

\[
\begin{align*}
\{&\text{True}\} \\
&\text{let } b = \text{newbarrier()} \\
&\{\text{send}(b, P) \ast \text{recv}(b, P)\} \\
&\{\text{send}(b, P) \ast P\} \text{ signal}(b) \{\text{True}\} \\
&\{\text{recv}(b, P)\} \text{ wait}(b) \{P\}
\end{align*}
\]
Barrier (simple version)

Capability to send $P$.

```plaintext
let $b = \text{newbarrier}()$

{send($b$, $P$) $\times$ recv($b$, $P$)}

{send($b$, $P$) $\times$ $P$} signal($b$) {True}

{recv($b$, $P$)} wait($b$) {$P$}
```

Capability to receive $P$. 

(one.osf/six.osf)
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>let b = newbarrier()</code> in</td>
<td>[computation];</td>
</tr>
<tr>
<td><code>signal(b)</code></td>
<td></td>
</tr>
<tr>
<td><code>wait(b)</code></td>
<td></td>
</tr>
<tr>
<td><code>wait(b);</code></td>
<td></td>
</tr>
<tr>
<td><code>[use result of computation]</code></td>
<td></td>
</tr>
</tbody>
</table>

Receive capability is split.
let \( b = \text{newbarrier}() \) in

\[
\begin{array}{c}
\text{[computation];} \\
\text{// Have: } P \times Q \\
\text{signal}(b)
\end{array}
\]

\[
\begin{array}{c}
\text{wait}(b); \\
\text{// Have: } P \\
\text{[use result of computation]}
\end{array}
\]

\[
\begin{array}{c}
\text{[use result of computation]} \\
\text{[use result of computation]}
\end{array}
\]

\[
\begin{array}{c}
\text{send}(b, P) \ast \text{recv}(b, P) \\
\text{[use result of computation]}
\end{array}
\]

\[
\begin{array}{c}
\text{send}(b, P) \ast P \\
\text{signal}(b) \\
\text{wait}(b); \\
\text{// Have: } Q \\
\text{[use result of computation]}
\end{array}
\]

\[
\begin{array}{c}
\text{wait}(b); \\
\text{// Have: } Q \\
\text{[use result of computation]}
\end{array}
\]

\[
\begin{array}{c}
\text{recv}(b, P \ast Q) \rightarrow \text{recv}(b, P) \ast \text{recv}(b, Q)
\end{array}
\]
let \( b = \) newbarrier()

\[
\begin{array}{l}
\{ \text{True}\} \\
\{ \text{send}(b, P) \ast \text{recv}(b, P)\} \\
\{ \text{send}(b, P) \ast P \} \text{ signal}(b) \{ \text{True}\} \\
\{ \text{recv}(b, P)\} \text{ wait}(b) \{ P\} \\
\end{array}
\]

\[\text{recv}(b, P \ast Q) \Rightarrow \text{recv}(b, P) \ast \text{recv}(b, Q)\]
Barrier: A little history

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

```plaintext
{True}
let b = newbarrier()
{send(b, P) * recv(b, P)}
{send(b, P) * P} signal(b) {True}
{recv(b, P)} wait(b) {P}
recv(b, P * Q) ⇒
recv(b, P) * 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

\[
\begin{align*}
\{\text{True}\} \\
\quad \text{let } b = \text{newbarrier}() \\
\quad \{\text{send}(b, P) \ast \text{recv}(b, P)\} \\
\quad \{\text{send}(b, P) \ast P\} \text{ signal}(b) \{\text{True}\} \\
\quad \{\text{recv}(b, P)\} \text{ wait}(b) \{P\} \\
\quad \text{recv}(b, P \ast Q) \Rightarrow \text{recv}(b, P) \ast \text{recv}(b, Q)
\end{align*}
\]
Named Propositions

Gives a fresh name \( \gamma \) to \( P \).

\[ \forall P. \text{True} \implies \exists \gamma. \gamma \mapsto P \]
Named Propositions

Gives a fresh name $\gamma$ to $P$. $P$ does not have to hold!

$\forall P. \text{True} \Rightarrow \exists \gamma. \gamma \rightarrow P$
Named Propositions

Gives a fresh name $\gamma$ to $P$. $P$ does not have to hold!

$$\forall P. \text{True} \implies \exists \gamma. \gamma \leftrightarrow P$$

$$\forall \gamma, P, Q. (\gamma \leftrightarrow P \land \gamma \leftrightarrow Q) \implies (P \leftrightarrow Q)$$

Agreement about proposition named $\gamma$. 
Named Propositions

Gives a fresh name $\gamma$ to $P$.

**Derive** named propositions from lower-level principles:

Agreement about proposition named $\gamma$. 
Named Propositions

- Gives a fresh name $\gamma$ to $P$.

- Derive named propositions from lower-level principles:
  
  Build named propositions on ghost state.

- Agreement about proposition named $\gamma$. 
Named Propositions

- Gives a fresh name $\gamma$ to $P$.
- Allocates new slot in “table”.

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

$$\forall \gamma, P, Q. (\gamma \mapsto P \ast \gamma \mapsto Q) \Rightarrow (P \iff Q)$$

Agreement about row $\gamma$ 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
Higher-Order Ghost State: Technicalities

We got a problem with our ghost state.

Who we gonna call?

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

Logic Iris($M$)
Higher-Order Ghost State: Technicalities

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

Logic $\text{Iris}(M)$
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
$Iris(M)$
• Equip PCMs with a “step-indexing structure”.
• Equip PCMs with a “step-indexing structure”.

→ CMRA

A CMRA is a tuple $(M : \mathcal{COF}, (\mathcal{V}_n \subseteq M)_{n \in \mathbb{N}}, \mid \cdot \mid : M \nrightarrow M^?, (\cdot) : M \times M \nrightarrow M)$ satisfying:

- $\forall n, a, b. a \nleq b \wedge a \in \mathcal{V}_n \Rightarrow b \in \mathcal{V}_n$ (CMRA-VALID-NE)
- $\forall n, m. n \geq m \Rightarrow \mathcal{V}_n \subseteq \mathcal{V}_m$ (CMRA-VALID-MONO)
- $\forall a, b, c. (a \cdot b) \cdot c = a \cdot (b \cdot c)$ (CMRA-ASSOC)
- $\forall a, b. a \cdot b = b \cdot a$ (CMRA-COMM)
- $\forall a. |a| \in M \Rightarrow |a| \cdot a = a$ (CMRA-CORE-ID)
- $\forall a. |a| \in M \Rightarrow ||a|| = |a|$ (CMRA-CORE-IDEM)
- $\forall a, b. |a| \in M \wedge a \nleq b \Rightarrow |b| \in M \wedge |a| \nleq |b|$ (CMRA-CORE-MONO)
- $\forall n, a, b. (a \cdot b) \in \mathcal{V}_n \Rightarrow a \in \mathcal{V}_n$ (CMRA-VALID-OP)
- $\forall n, a, b_1, b_2. a \in \mathcal{V}_n \wedge a \nleq b_1 \cdot b_2 \Rightarrow \exists c_1, c_2. a = c_1 \cdot c_2 \wedge c_1 \nleq b_1 \wedge c_2 \nleq b_2$ (CMRA-EXTEND)

where

$a \nleq b \triangleq \exists c. b = a \cdot c$ (CMRA-INCL)
Higher-Order Ghost State: Technicalities

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

A CMRA is a tuple $(M : \text{COFE}, (\forall n \subseteq M)_{n \in \mathbb{N}}, |\cdot| : M \to \mathbb{N}^\text{op}, (\cdot) : M \times M \to M)$ satisfying:

\[
\begin{align*}
&\forall n, a, b. a \equiv n b \land a \in \mathcal{V}_n \Rightarrow b \in \mathcal{V}_n \quad \text{(CMRA-VALID-NE)} \\
&\forall n, m. n \geq m \Rightarrow \mathcal{V}_n \subseteq \mathcal{V}_m \quad \text{(CMRA-VALID-MONO)} \\
&\forall a, b, c. (a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \text{(CMRA-ASSOC)} \\
&\forall a, b. a \cdot b = b \cdot a \quad \text{(CMRA-COMM)} \\
&\forall a. |a| \in M \Rightarrow |a| \cdot a = a \quad \text{(CMRA-CORE-ID)} \\
&\forall a. |a| \in M \Rightarrow ||a|| = |a| \quad \text{(CMRA-CORE-IDEM)} \\
&\forall a, b. |a| \in M \land a \preceq b \Rightarrow |b| \in M \land |a| \preceq |b| \quad \text{(CMRA-CORE-MONO)} \\
&\forall n, a, b. (a \cdot b) \in \mathcal{V}_n \Rightarrow a \in \mathcal{V}_n \quad \text{(CMRA-VALID-OP)} \\
&\forall n, a, b_1, b_2. a \in \mathcal{V}_n \land a \equiv n b_1 \cdot b_2 \Rightarrow \\
&\exists c_1, c_2. a = c_1 \cdot c_2 \land c_1 \equiv n b_1 \land c_2 \equiv n b_2 \quad \text{(CMRA-EXTEND)}
\end{align*}
\]

where

\[
a \preceq b \triangleq \exists c. b = a \cdot c \quad \text{(CMRA-INCL)}
\]
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.

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

\[
\forall n, a, b. \ a \equiv b \land a \in \mathcal{V}_n \Rightarrow b \in \mathcal{V}_n \quad \text{(CMRA-VALID-NE)}
\]
\[
\forall n, m. \ n \geq m \Rightarrow \mathcal{V}_n \subseteq \mathcal{V}_m \quad \text{(CMRA-VALID-MONO)}
\]
\[
\forall a, b, c. \ (a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \text{(CMRA-ASSOC)}
\]
\[
\forall a, b. \ a \cdot b = b \cdot a \quad \text{(CMRA-COMM)}
\]
\[
\forall a. \ |a| \in M \Rightarrow |a| \cdot a = a \quad \text{(CMRA-CORE-ID)}
\]
\[
\forall a. \ |a| \in M \Rightarrow ||a|| = |a| \quad \text{(CMRA-CORE-IDEM)}
\]
\[
\forall a, b. |a| \in M \land a \triangleleft b \Rightarrow |b| \in M \land |a| \triangleleft |b| \quad \text{(CMRA-CORE-MONO)}
\]
\[
\forall n, a, b. \ (a \cdot b) \in \mathcal{V}_n \Rightarrow a \in \mathcal{V}_n \quad \text{(CMRA-VALID-OP)}
\]
\[
\forall n, a, b_1, b_2. \ a \in \mathcal{V}_n \land a \equiv b_1 \cdot b_2 \Rightarrow
\exists c_1, c_2. \ a = c_1 \cdot c_2 \land c_1 \equiv b_1 \land c_2 \equiv b_2 \quad \text{(CMRA-EXTEND)}
\]

where

\[
a \triangleleft b \triangleq \exists c. \ b = a \cdot c \quad \text{(CMRA-INCL)}
\]
Higher-Order Ghost State: Technicalities

• Equip PCMs with a “step-indexing structure”.
  \[ \rightarrow \text{CMRA} \]

• 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

∀\(P.\) True \(\Rightarrow\) \(\exists\gamma.\) \(\gamma \leftrightarrow P\)

∀\(\gamma, P, Q.\) (\(\gamma \leftrightarrow P \ast \gamma \leftrightarrow Q\)) \(\Rightarrow\) \(\triangleright(P \iff Q)\)

Agreement about proposition named \(\gamma\) only holds at the next step-index.
let \( b = \text{newbarrier}() \)

\[
\{ \text{send}(b, P) \ast \text{recv}(b, P) \} \\
\{ \text{send}(b, P) \ast P \} \text{ signal}(b) \{ \text{True} \} \\
\{ \text{recv}(b, P) \} \text{ wait}(b) \{ P \}
\]

\[
\text{recv}(b, P \ast Q) \Rightarrow \text{recv}(b, P) \ast \text{recv}(b, Q)
\]
Iris: Resting on Simple Foundations

Invariants:

\[
\{\text{In} \ast P\} \ e \ \{\text{In} \ast Q\} \ \varepsilon \quad \text{atomic}(e)
\]

\[
\exists \ \{P\} \ e \ \{\nu \cdot Q\} \varepsilon \uplus \{\iota\}
\]

Ghost state (any CMRA):

\[
\forall a_f, n. \ a \ #_n \ a_f \ \Rightarrow \ b \ #_n \ a_f
\]

\[
a \Rightarrow [b]
\]

\[
[a] \Rightarrow \mathcal{V}(a)
\]

For specifying some synchronization primitives, these foundations are not enough! Any PCM can be lifted to a CMRA, so all the old reasoning remains valid.
Invariant:

\[ \{ \langle l \ast P \rangle \} e \{ \langle l \ast Q \rangle \}_e \quad \text{atomic}(e) \]

Ghost state (any CMRA):

\[ \forall \ a, \ f, \ n. \ a \#_n a_f \Rightarrow b \#_n a_f \]

\[ a \Rightarrow b \]

\[ a \cdot b = c \]

\[ a \ast b \leftrightarrow c \]

\[ a \Rightarrow \mathcal{V}(a) \]

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

Thank you for your attention!
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 \textit{simplify} the model with CMRAs
- Coq formalization

Ongoing work:

- \textit{Encode} invariants using higher-order ghost state
- Applying named propositions in the safety proof of Rust

Thank you for your attention!