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

Kasper Svendsen

joint work with Ralf Young, David Swasey, Filip Sieczkowski, Aaron Turon, Lars Birkedal and Derek Dreyer
A uniform framework for describing interference

LRG
RGSep

CAP
HOCAP
TaDA

CaReSL
iCAP

Iris
Iris

- Supports encoding of existing reasoning principles
  - Monoids for **expressing** protocols on shared state
  - Invariants for **enforcing** protocols on shared state
Iris

- Invariants and monoids are orthogonal

- Treating them as such, leads to a simpler logic, and a model simple enough to formalize in Coq
Iris

- Supports a notion of **logical atomicity**
  - extends reasoning principles usually reserved for atomic code to code that **appears** to be atomic
  - we can **define** logical atomicity in Iris
• Supports a notion of \textit{logical atomicity}

• extends reasoning principles usually reserved for atomic code to code that \textit{appears} to be atomic

• we can \textbf{define} logical atomicity in Iris
Iris supports a notion of logical atomicity:

- Extends reasoning principles usually reserved for atomic code to code that appears to be atomic.
- We can define logical atomicity in Iris.
Part 1
Iris
Invariants

- An invariant is a property that holds of some piece of shared state at all times.
Invariants

- An invariant is a property that holds of some piece of shared state at all times

\[
\{\Delta R \ast P\} \quad e \quad \{\Delta R \ast Q\} \
\]

\[
\begin{array}{c}
\hline
\{R^i \ast P\} \\
\hline
\{Q\}
\end{array}
\] \quad e \quad \epsilon \cup \{i\}

There exists a shared invariant that owns \(R\)

The set of invariants that we may open
An invariant is a property that holds of some piece of shared state at all times.

We open the invariant and take ownership of R.

There exists a shared invariant that owns R.

The set of invariants that we may open.
• An invariant is a property that holds of some piece of shared state at all times

There exists a shared invariant that owns $R$.

We open the invariant and take ownership of $R$.

To close the invariant, we must relinquish ownership of $R$.

The set of invariants that we may open.
Introduces a circularity in the model
• Modelled using standard metric-based techniques (ModuRes library in Coq)

Invariants

Higher-order separation logic + Impredicative Invariants + Monoids
Monoids

- Iris is parameterised by a notion of ghost resources
- Ghost resources consists of
  - **Information** about the current ghost state
  - **Rights** to update ghost state
- We use monoids to model ghost resources
Monoids

- Ghost resource $\overline{m}$ asserts ownership of $m$ fragment
- Ghost resources can be split arbitrarily

\[ m_1 \cdot m_2 \Leftrightarrow \overline{m_1} * \overline{m_2} \]

- and support frame-preserving updates

\[
\forall a_f. (a \cdot a_f) \downarrow \Rightarrow (b \cdot a_f) \downarrow
\]

\[
\overline{a} \Rightarrow \overline{b}
\]
Part 2
Recovering existing reasoning principles
Deriving small-footprint specifications

- **Example**: recovering small-footprint specifications from large-footprint specifications

- Same idea as in Superficially Substructural Types (ICFP12) and Fictional Separation Logic (ESOP12)
A $\lambda$-calculus with channels

- We instantiate Iris with a $\lambda$-calculus with channels

$$e ::= \ldots \mid \text{newch} \mid \text{send}(e, e) \mid \text{tryrecv}(e) \mid \text{fork}(e)$$

- with the following per-thread reduction semantics

$$C[c \mapsto M]; \text{send}(c, v) \rightarrow C[c \mapsto M \uplus \{v\}]; ()$$
$$C[c \mapsto \emptyset]; \text{tryrecv}(c) \rightarrow C[c \mapsto \emptyset]; \text{none}$$
$$C[c \mapsto M \uplus \{v\}]; \text{tryrecv}(c) \rightarrow C[c \mapsto M]; \text{some}(v)$$
Large-footprint specs

- Reduction relation lifts directly to large-footprint specs
- The reduction

\[ C[c \mapsto M]; \textbf{send}(c, v) \rightarrow C[c \mapsto M \cup \{v\}]; () \]

yields the following axiom

\[ \{ [C[c \mapsto M]] \} \textbf{send}(c, v) \{ r. \ r = () \land [C[c \mapsto M \cup \{v\}]] \} \]

Asserts exclusive ownership of entire physical state
Small-footprint specs

• Large-footprint spec requires global reasoning

\{ [C[c \mapsto M]] \} \text{ send}(c, v) \{ r. r = () \land [C[c \mapsto M \cup \{v\}]]) \}

• **Goal:** Derive small-footprint specification that only mentions channels affected by each operation
Small-footprint specs

- **Idea**
  - Introduce appropriate channel ghost resources
  - Introduce an invariant that owns the physical state (so that it can be shared) and ties ghost resources to physical state
  - Extends to a general construction
Channel-local monoid

• **Goal:** ghost channels resources that support exclusive ownership of individual channels

• Use partial channel “heaps”

\[ |\text{NET}| = \text{Chan} \overset{\text{fin}}{\rightarrow} \text{MsgBag} \]
\[ f \cdot g = f \cup g, \quad \text{if } \text{dom}(f) \cap \text{dom}(g) = \emptyset \]

• \([c \mapsto M]\) asserts exclusive ownership of ghost channel \(c\) and that contains messages \(M\)
Authoritative monoid

- **Goal**: a monoid with
  - An authoritative element $m\bullet$ that asserts that the current ghost state is exactly $m$
  - A partial element $m\circ$ that asserts ownership of an $m$ fragment of the authoritative state
  - s.t. all fragments combine to the authoritative state
Deriving a channel-local specification

\{ c \leftarrow M \}

\textbf{send}(c, m)

\{ c \leftarrow M \uplus \{ m \} \}
Deriving a channel-local specification

Channel resource asserts ownership of corresponding fragment:

\[ c \prec M \triangleq [c \mapsto M] \circ \]

\{ c \prec M \}\]

\textbf{send}(c, m)

\{ c \prec M \mathbin{\cup} \{ m \} \}
Deriving a channel-local specification

**Invariant**: the physical state is authoritative ghost state

\[ \exists C. \mathcal{C}_\text{\scriptsize \bullet} \ast [C] \]

\{c \prec M\}

\textbf{send}(c, m)

\{c \prec M \cup \{m\}\}

Channel resource asserts ownership of corresponding fragment:

\[ c \prec M \triangleq [c \leftrightarrow M] \circ \]

\{c \prec M \cup \{m\}\}
Deriving a channel-local specification

**Invariant:** the physical state is authoritative ghost state

\[ \exists C. \llbracket C \bullet \rrbracket \star \llbracket C \rrbracket \]

Channel resource asserts ownership of corresponding fragment:

\[ c \prec M \triangleq \llbracket c \mapsto M \rrbracket \odot \]

\[
\{ c \prec M \} \\
\{ \llbracket c \mapsto M \rrbracket \odot \star \} \\
\text{send}(c, m) \\
\{ c \prec M \uplus \{ m \} \} \]
Deriving a channel-local specification

**Invariant:** the physical state is authoritative ghost state

\[ \exists C. \ [\overline{C} \bullet] \ast [C] \]

Channel resource asserts ownership of corresponding fragment:

\[ c \leftarrow M \triangleq [\overline{c} \rightarrow M] \circ \]

\[
\{ c \leftarrow M \}
\{ [\overline{c} \rightarrow M] \circ \ast [\overline{C} \bullet] \ast [C] \}
\]

**send** \((c, m)\)

\[
\{ c \leftarrow M \uplus \{ m \} \}
\]
Deriving a channel-local specification

**Invariant:** the physical state is authoritative ghost state

\[ \exists C. \, [\overset{\circ}{C}] \ast [C] \]

Channel resource asserts ownership of corresponding fragment:

\[ c \leftarrow M \triangleq [c \leftarrow M] \circ \]

\[
\{ c \leftarrow M \}
\begin{align*}
\{ & [c \leftarrow M] \circ \ast [\overset{\circ}{C}] \ast [C] \\
\text{send}(c, m) & \{ \ast [C[c \leftarrow C(c) \cup \{m\}]] \}
\end{align*}
\]

\[
\{ c \leftarrow M \cup \{m\} \}
\]
Deriving a channel-local specification

**Invariant:** the physical state is authoritative ghost state

\[ \exists C. \, [C] \ast [C] \]

Channel resource asserts ownership of corresponding fragment:

\[ c < M \triangleq [c \mapsto M] \circ \]

\[
\{ c < M \}
\]

\[
\{ [c \mapsto M] \circ \ast [C] \ast [C] \}
\]

**send**\((c, m)\)

\[
\{ [c \mapsto M] \circ \ast [C] \ast [C[c \mapsto C(c) \cup \{m\}]] \}
\]

\[
\{ c < M \cup \{m\} \}
\]
Deriving a channel-local specification

**Invariant**: the physical state is authoritative ghost state

\[ \exists C. C \bullet * [C] \]

Channel resource asserts ownership of corresponding fragment:

\[ c \prec M \triangleq [c \mapsto M] \circ \]

\[
\{ c \prec M \} \\
\{ [c \mapsto M] \circ * C \bullet * [C] \} \\
\text{send}(c, m) \\
\{ [c \mapsto M] \circ * C \bullet * \left[ C[c \mapsto C(c) \cup \{m\}] \right] \} \\
\{ c \prec M \cup \{m\} \} \]
Deriving a channel-local specification

**Invariant**: the physical state is authoritative ghost state

\[ \exists C. \{ C \bullet \} \ast \{ C \} \]

Channel resource asserts ownership of corresponding fragment:

\[ c \prec M \triangleq \{ c \mapsto M \} \circ \]

\[
\{ c \prec M \} \\
\{ [c \mapsto M] \circ \ast [C \bullet \} \ast \{ C \} \}
\]

**send** \( (c, m) \)

\[
\{ [c \mapsto M] \circ \ast [C \bullet \} \ast \left[ C'[c \mapsto C(c) \cup \{ m \}] \right] \}
\]

\[
\{ [c \mapsto M \cup \{ m \}] \circ \ast [C' \bullet \} \ast \{ C' \} \}
\]

\[
\{ c \prec M \cup \{ m \} \}
\]
Deriving small-footprint specifications

- Channel monoid encodes small-footprint channel resources
- Invariant relates ghost and physical state using authoritative monoid to allow ownership of channel fragments
Recovering existing reasoning techniques

- We saw how to recover reasoning principles from Superficially Substructural Types and Fictional Separation

- One can also recover reasoning principles from CaReSL and iCAP through a encoding of STSs as monoids
Part 3
Logical atomicity
Logical atomicity

• In part 2 we used the invariant rule to access the shared physical resource

\[
\{\exists R \ast P\} e \{\exists R \ast Q\} \varepsilon \quad \text{e atomic}
\]

\[
\{R^t \ast P\} e \{Q\} \varepsilon \cup \{\nu\}
\]

• This rule only applies to atomic expressions

• Iris allows us to extend this reasoning principle to logically atomic code
Logical atomicity

- In part 2 we used the invariant rule to access the shared physical resource

\[
\begin{align*}
\{\triangleright R \ast P\} & \ e \ \{\triangleright R \ast Q\} \ \varepsilon \\
\{\mathcal{L}_{R} \ast P\} & \ e \ \{Q\} \\
\end{align*}
\]

- This rule only applies to atomic expressions.

- Iris allows us to extend this reasoning principle to logically atomic code.

We can define logically atomic triples:

\[
\langle P \rangle \ e \ \langle Q \rangle
\]
Logical atomicity

• **Example:** a blocking receive operation

\[ \text{recv} \triangleq \text{rec } \text{recv}(c). \text{ let } v = \text{tryrecv}(c) \text{ in } \]

\[ \text{case } v \text{ of none } => \text{recv}(c) | \text{some}(m) => m \]

• Spins (without side effects) until a msg is received

• The linearisation point is the first successful \text{tryrecv}
Logical atomicity

• Ideas
  • Let clients reason about the state immediately before and after the linearisation point
  • Let clients open invariants **around** the linearisation point
Logical atomicity

• Ideas
  • Let clients reason about the state immediately before and after the linearisation point
  • Let clients open invariants around the linearisation point

Parameterise our specifications with view shifts
Let view shifts open and close invariants
Mask-changing view shifts

- Index view shifts with the set of invariants enabled before and after the view shift

\[ P \varepsilon_1 \equiv \varepsilon_2 Q \]

- Asserts
  - that we can update the instrumented state from \( P \) to \( Q \) without changing the physical state
  - where the invariants in \( \varepsilon_1 \) are enabled before the view shift
  - and the invariants in \( \varepsilon_2 \) are enabled after the view shift
Mask-changing view shifts

• We can change the invariant mask around atomic expressions, provided we restore it again

\[
P \{\iota\} \equiv^\emptyset P' \quad \{P'\} \text{ e } \{v. Q'\}_\emptyset \quad \forall v. Q' \emptyset \equiv \{\iota\} Q
\]

\[
\{P\} \text{ e } \{v. Q\}_{\{\iota\}}
\]

• We can open and close invariants using view shifts

\[
\boxed{P}^{\iota} \{\iota\} \equiv^\emptyset \triangleright P
\]

\[
\boxed{P}^{\iota} \triangleright P \emptyset \equiv \{\iota\} \top
\]
Logical atomicity

- **Idea:** Let clients open and close invariants around linearisation point and update instrumented state

\[
\langle P \rangle e \langle Q \rangle_\mathcal{E} \approx \forall R_p, R_q, \mathcal{E}_R. \mathcal{E} \cap \mathcal{E}_R = \emptyset \land \\
(R_p \iff -\mathcal{E}_R P) \land (Q \Rightarrow -\mathcal{E}_R R_q) \\
\Rightarrow \{R_p\} e \{R_q\}
\]

- This allows us to open invariants around logically atomic code

\[
\left\langle \delta R \ast P \right\rangle e \left\langle \delta R \ast Q \right\rangle_{\mathcal{E}} \\
\left\langle R^\iota \ast P \right\rangle e \left\langle Q \right\rangle_{\mathcal{E} \cup \{\iota\}}
\]
Logical atomicity

- **Idea:** Let clients open and close invariants around linearisation point and update instrumented state

\[
\langle P \rangle \ e \ \langle Q \rangle \ \mathcal{E} \approx \ \forall R_p, R_q, \mathcal{E}_R. \ \mathcal{E} \cap \mathcal{E}_R = \emptyset \ \land \\
(R_p \iff -\mathcal{E}_R \ P) \ \land \ (Q \implies -\mathcal{E}_R \ R_q) \\
\implies \ \{R_p\} \ e \ \{R_q\}
\]

- This allows us to open invariants around logically atomic code

\[
\langle \triangleright R \ast P \rangle \ e \ \langle \triangleright R \ast Q \rangle \ \mathcal{E} \\
\langle \lceil R \rceil \ast P \rangle \ e \ \langle Q \rangle \ \mathcal{E} \cup \{\iota\}
\]

From the client's point of view it looks like we have access to the invariant \(R\) for the duration of \(e\).
Logical atomicity

- **Idea:** Let clients open and close invariants around the linearisation point and update instrumented state.

\[
\langle P \rangle e \langle Q \rangle_\mathcal{E} \approx \forall R_p, R_q, \mathcal{E}_R. \ \mathcal{E} \cap \mathcal{E}_R = \emptyset \land \left( R_p \iff -\mathcal{E}_R P \right) \land \left( Q \implies -\mathcal{E}_R R_q \right) \\
\Rightarrow \{ R_p \} e \{ R_q \}
\]

- This allows us to open invariants around logically atomic code.
Case study

- logically atomic
  - elimination stack
  - mutable references as channels
  - message passing blocking receive

- physically atomic
  - small-footprint specifications
  - λ-calculus with asynchronous message passing
Logical atomicity

- Logical atomicity is not built into Iris, but Iris is sufficiently expressive that we can define it in Iris.
Conclusions

• Iris is
  • simpler than previous logics
  • can encode reasoning principles from previous logics
  • and can do some fancy new stuff (logical atomicity)

• Monoids and invariants are all you need