Interactive and Automated Proofs in Modal Separation Logic

Robbert Krebbers

Radboud University Nijmegen, The Netherlands

August 2, 2023 @ ITP, Białystok, Poland

This talk is about embedding
proof assistants in proof assistants

Let me first give some context

A verification framework, implemented in the Coq proof assistant, for developing and deploying

Iris

advanced forms of separation logic, especially for higher-order and concurrent programs

Perennial DimSum Cerise RustBelt ReLoC

Melocoton VMSL RefinedC Aneris

Diaframe Iris-Wasm

Simuliris Iris Compass

Iris-Tini RustHornBelt

Cosmo CQS SeLoC

iGPS Hazel gDOT GoJournal

OCPL Islaris Actris Iron iRC11

How Iris is used

Developing a logic: Use Iris as a meta theory to develop a program logic

Deploying a logic: Verify programs or a type system using the developed logic

How Iris is used

Developing a logic: Use Iris as a meta theory to develop a program logic

- for a specific language: ML, Rust, C, Go, WebAssembly, capability machines, ...
- program property: functional correctness, non-interference, crash safety, refinement, complexity, ...
- programming paradigm: algebraic effects, distributed systems, session types, relaxed memory concurrency, ...

Deploying a logic: Verify programs or a type system using the developed logic

How Iris is used

Developing a logic: Use Iris as a meta theory to develop a program logic

- for a specific language: ML, Rust, C, Go, WebAssembly, capability machines, ...
- program property: functional correctness, non-interference, crash safety, refinement, complexity, ...
- programming paradigm: algebraic effects, distributed systems, session types, relaxed memory concurrency, ...

Deploying a logic: Verify programs or a type system using the developed logic

For both developing and deploying logics,
a proof assistant is essential

Wanted:

proof assistant for

Iris

Wanted:

proof assistant for

Iris

Very different from the logic of Coq/HOL/etc

Wanted:

proof assistant for
higher-order
impredicative
modal
concurrent
separation logic

Very different from the logic of Coq/HOL/etc

How?

Embed proof assistant in existing proof assistant

How?

Embed proof assistant in existing proof assistant

Why?

Prove soundness of embedded proof assistant
Reuse infrastructure of host proof assistant
Users do not need to learn new tool

Overview of this talk

1. Brief introduction to separation logic
2. Interactive proofs using the Iris Proof Mode
3. Implementation of the Iris Proof Mode
4. Proof automation using Diaframe

Part #1: Separation Logic 101

Separation logic [O'Hearn, Reynolds, Yang; CSL'01]

Propositions P, Q denote ownership of resources

Separating conjunction $P * Q$:
The resources consists of separate parts satisfying P and Q

Basic example:

$$\{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2\} \mathit{swap}\ \ell_1\ \ell_2 \{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}$$

the $*$ ensures that ℓ_1 and ℓ_2 are different memory locations

Separation logic [O'Hearn, Reynolds, Yang; CSL'01]

Propositions P, Q denote ownership of resources

Separating conjunction $P * Q$:
The resources consists of separate parts satisfying P and Q

Slightly less basic example:

$$\text{isList}\ \ell\ \vec{v} \triangleq \begin{cases} \ell \mapsto \text{nil} & \text{if } \vec{v} = [\,] \\ \exists \ell'.\, \ell \mapsto \text{cons}\ v_1\ \ell' * \text{isList}\ \ell'\ \vec{v}_2 & \text{if } \vec{v} = v_1 :: \vec{v}_2 \end{cases}$$

Separation logic [O'Hearn, Reynolds, Yang; CSL'01]

Propositions P, Q denote ownership of resources

Separating conjunction $P * Q$:
The resources consists of separate parts satisfying P and Q

Slightly less basic example:

$$\text{isList}\ \ell\ \vec{v} \triangleq \begin{cases} \ell \mapsto \text{nil} & \text{if } \vec{v} = [\,] \\ \exists \ell'.\, \ell \mapsto \text{cons}\ v_1\ \ell' * \text{isList}\ \ell'\ \vec{v}_2 & \text{if } \vec{v} = v_1 :: \vec{v}_2 \end{cases}$$

$$\{\text{isList}\ \ell_1\ \vec{v}_1 * \text{isList}\ \ell_2\ \vec{v}_2\}\, \mathit{append}\ \ell_1\ \ell_2 \{\text{isList}\ \ell_1\ (\vec{v}_1 +\!\!+\ \vec{v}_2)\}$$

Separation logic [O'Hearn, Reynolds, Yang; CSL'01]

Propositions P, Q denote ownership of resources

Separating conjunction $P * Q$:
The resources consists of separate parts satisfying P and Q

Slightly less basic example:

$$\text{isList}\ \ell\ \vec{v} \triangleq \begin{cases} \ell \mapsto \text{nil} & \text{if } \vec{v} = [\,] \\ \exists \ell'.\, \ell \mapsto \text{cons}\ v_1\ \ell' * \text{isList}\ \ell'\ \vec{v}_2 & \text{if } \vec{v} = v_1 :: \vec{v}_2 \end{cases}$$

$$\{\text{isList}\ \ell_1\ \vec{v}_1 * \text{isList}\ \ell_2\ \vec{v}_2\}\, \mathit{append}\ \ell_1\ \ell_2 \{\text{isList}\ \ell_1\ (\vec{v}_1 +\!+ \vec{v}_2)\}$$

the $*$ ensures that all nodes of ℓ_1 and ℓ_2 are disjoint

The simple model of separation logic

The semantic domains:

$$\ell \in \mathit{Loc} \triangleq \mathbb{N}$$

$$\sigma \in \mathit{Heap} \triangleq \mathit{Loc} \xrightarrow{\text{fin}} \mathit{Val}$$

$$P, Q \in \mathit{heapProp} \triangleq \mathit{Heap} \rightarrow \mathsf{Prop}$$

The simple model of separation logic

The semantic domains:

$$\ell \in Loc \triangleq \mathbb{N}$$

$$\sigma \in Heap \triangleq Loc \xrightarrow{\text{fin}} Val$$

$$P, Q \in heapProp \triangleq Heap \to \text{Prop}$$

Entailment:

$$P \vdash Q \triangleq \forall \sigma.\ P\sigma \to Q\sigma$$

The simple model of separation logic

The semantic domains:

$$\ell \in Loc \triangleq \mathbb{N}$$

$$\sigma \in Heap \triangleq Loc \xrightarrow{\text{fin}} Val$$

$$P, Q \in heapProp \triangleq Heap \to \mathrm{Prop}$$

Entailment:

$$P \vdash Q \triangleq \forall \sigma.\, P\sigma \to Q\sigma$$

The connectives of separation logic:

$$\ell \mapsto v \triangleq \lambda \sigma.\, \sigma(\ell) = v$$

$$P \land Q \triangleq \lambda \sigma.\, P\sigma \land Q\sigma$$

$$P * Q \triangleq \lambda \sigma.\, \exists \sigma_1 \sigma_2.\, \sigma = \sigma_1 \uplus \sigma_2 \land P\sigma_1 \land Q\sigma_2$$

$$(\exists x : A.\, P) \triangleq \lambda \sigma.\, \exists x : A.\, P\sigma$$

disjointness of heaps, hidden by $*$

How to do proofs in separation logic

Suppose we want to prove $P * (\exists a.\, \Phi a) * Q \;\vdash\; Q * (\exists a.\, P * \Phi a)$

How to do proofs in separation logic

Suppose we want to prove $P * (\exists a.\, \Phi a) * Q \;\vdash\; Q * (\exists a.\, P * \Phi a)$

1. **Unfold definitions of the model**: $\forall \sigma.\, (\exists \sigma_1\, \sigma_2.\, \sigma = \sigma_1 \uplus \sigma_2 \wedge P \sigma_1 \wedge \ldots) \rightarrow \ldots$
   - Defeats the purpose of separation logic to hide reasoning about disjointness
   - Does not scale to larger goals or modal models

How to do proofs in separation logic

Suppose we want to prove $P * (\exists a.\, \Phi a) * Q \ \vdash \ Q * (\exists a.\, P * \Phi a)$

1. **Unfold definitions of the model**: $\forall \sigma.\, (\exists \sigma_1\, \sigma_2.\, \sigma = \sigma_1 \uplus \sigma_2 \wedge P \sigma_1 \wedge \ldots) \rightarrow \ldots$
   - Defeats the purpose of separation logic to hide reasoning about disjointness
   - Does not scale to larger goals or modal models
2. **Use the laws of separation logic**: associativity/commutativity of $*$, distributivity of $\exists$ over $*$, ...
   - Too low-level, already small proofs require many steps
   - Also rather slow

How to do proofs in separation logic

Suppose we want to prove $P * (\exists a.\, \Phi a) * Q \;\vdash\; Q * (\exists a.\, P * \Phi a)$

1. **Unfold definitions of the model**: $\forall \sigma.\, (\exists \sigma_1\, \sigma_2.\, \sigma = \sigma_1 \uplus \sigma_2 \wedge P \sigma_1 \wedge \ldots) \rightarrow \ldots$
   - Defeats the purpose of separation logic to hide reasoning about disjointness
   - Does not scale to larger goals or modal models
2. **Use the laws of separation logic**: associativity/commutativity of $*$, distributivity of $\exists$ over $*$, ...
   - Too low-level, already small proofs require many steps
   - Also rather slow
3. **Use Iris**
   - Topic of today's talk

Part #2: Iris Proof Mode

Iris Proof Mode (IPM) [Krebbers *et al.*; POPL'17, ICFP'18]

Enable tactic-style proofs in separation logic

- Extend Coq with named proof contexts for separation logic
- Tactics for introduction and elimination of all connectives of separation logic . . .
- . . . that can be used in Coq's mechanisms for automation/tactic programming
- Implemented without modifying Coq (using reflection, type classes and Ltac)

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  i
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

Lemma in separation logic

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
______________________________________(1/1)
P ∗ (∃ a : A, Φ a) ∗ Q
⊢ Q ∗ (∃ a : A, P ∗ Φ a)
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
______________________(1/1)
"H1" : P
"H2" : ∃ a : A, Φ a
"H3" : Q
______________________∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
______________________________________(1/1)
"H1" : P
"H2" : Φ x
"H3" : Q
--------------------------------------∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
______________________(1/1)
"H1" : P
"H2" : Φ x
"H3" : Q
----------------------∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

∗ means: resources should be split

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
Qed.
```

The hypotheses for the left conjunct

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
______________________(1/1)
"H1" : P
"H2" : Φ x
"H3" : Q
--------------------------∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

∗ means: resources should be split

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

```
2 subgoals
A : Type
P, Q : iProp
Φ : A → iProp
x : A
______________________________________(1/2)
"H3" : Q
--------------------------------------∗
Q

______________________________________(2/2)
"H1" : P
"H2" : Φ x
--------------------------------------∗
∃ a : A, P ∗ Φ a
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  by iFrame.
Qed.
```

We can also solve this goal automatically

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
______________________________________(1/1)
"H1" : P
"H2" : ∃ a, Φ a
"H3" : Q
--------------------------------------∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  by iFrame.
Qed.
```

We can also solve this goal automatically

```
No more subgoals.
```

Iris Proof Mode demo

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[$ [? $]] //".
Qed.
```

Or use intro patterns

Features of the Iris Proof Mode

- **Proofs have the look and feel of ordinary Coq proofs**
  For many Coq tactics `tac`, we have a variant `iTac`

Features of the Iris Proof Mode

- **Proofs have the look and feel of ordinary Coq proofs**
  For many Coq tactics `tac`, we have a variant `iTac`
- **Support for advanced features of separation logic**
  Higher-order quantification, modalities, invariants, ghost state, ...

Features of the Iris Proof Mode

- **Proofs have the look and feel of ordinary Coq proofs**
  For many Coq tactics `tac`, we have a variant `iTac`
- **Support for advanced features of separation logic**
  Higher-order quantification, modalities, invariants, ghost state, ...
- **Integration with tactics for proving programs**
  Symbolic execution tactics for weakest preconditions

Features of the Iris Proof Mode

- **Proofs have the look and feel of ordinary Coq proofs**
  For many Coq tactics `tac`, we have a variant `iTac`
- **Support for advanced features of separation logic**
  Higher-order quantification, modalities, invariants, ghost state, ...
- **Integration with tactics for proving programs**
  Symbolic execution tactics for weakest preconditions
- **Tactic programming**
  One can combine/program with IPM tactics using Coq's Ltac like ordinary Coq tactics

Changes since the POPL'17 paper on Iris Proof Mode

- Generalized to any Bunched Implications (BI) logic (Krebbers *et al.*, ICFP'18)
- Many usability improvements:
  Smarter tactics, better error messages, improved robustness and performance
- Proof automation: **RefinedC** (Sammler *et al.* PLDI'21), **Diaframe** (Mulder *et al.* PLDI'22, PLDI'23, OOPSLA'23), BedRock Systems (proprietary)

Changes since the POPL'17 paper on Iris Proof Mode

- Generalized to any Bunched Implications (BI) logic (Krebbers *et al.*, ICFP'18)
- Many usability improvements:
  Smarter tactics, better error messages, improved robustness and performance
- Proof automation: **RefinedC** (Sammler *et al.* PLDI'21), **Diaframe** (Mulder *et al.* PLDI'22, PLDI'23, OOPSLA'23), BedRock Systems (proprietary)

Most importantly: Iris (Proof Mode) got users:

- Coq became essential to teach Iris / concurrent separation logic
- 11 PhD theses
- 81 publications
- 3 editions of the Iris workshop
- Used by researchers at companies: BedRock Systems, Meta, Jetbrains

Iris Proof Mode versus Diaframe

Plain Iris:

```
Lemma release_spec γ lk R :
  {{{ is_lock γ lk R ∗ locked γ ∗ R }}} release lk {{{ RET #(); True }}}.
Proof.
  iIntros (Φ) "(Hl & Hγ & HR) HΦ". iDestruct "Hl" as (lo ln ->) "#Hinv".
  iDestruct "Hγ" as (o) "Hγo".
  wp_lam. wp_proj. wp_bind (! _)%E.
  iInv N as (o' n) "(>Hlo & >Hln & >Hauth & Haown)".
  wp_load.
  iDestruct (own_valid_2 with "Hauth Hγo") as
    %[[<-%Excl_included%leibniz_equiv _]%prod_included _]%auth_both_valid_discrete.
  iModIntro. iSplitL "Hlo Hln Hauth Haown".
  { iNext. iExists o, n. by iFrame. }
  wp_pures.
  iInv N as (o' n') "(>Hlo & >Hln & >Hauth & Haown)".
  iApply wp_fupd. wp_store.
  iDestruct (own_valid_2 with "Hauth Hγo") as
    %[[<-%Excl_included%leibniz_equiv _]%prod_included _]%auth_both_valid_discrete.
  iDestruct "Haown" as "[[Hγo' _]|Haown]".
  { iDestruct (own_valid_2 with "Hγo Hγo'") as %[[] ?]%auth_frag_op_valid_1. }
  iMod (own_update_2 with "Hauth Hγo") as "[Hauth Hγo]".
  { apply auth_update, prod_local_update_1.
    by apply option_local_update, (exclusive_local_update _ (Excl (S o))). }
  iModIntro. iSplitR "HΦ"; last by iApply "HΦ".
  iIntros "!> !>". iExists (S o), n'.
  rewrite Nat2Z.inj_succ -Z.add_1_r. iFrame. iLeft. by iFrame.
Qed.
```

Diaframe:

```
Lemma release_spec γ1 γ2 lk R :
  {{{ is_lock γ1 γ2 lk R ∗ locked γ2 ∗ R }}} release lk {{{ RET #(); True }}}.
Proof. iSmash. Qed.
```

Part #3: Implementation of Iris Proof Mode

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 :=`
	`  (* fancy "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 :=`
	`  (* fancy "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 :=`
	`  (* fancy "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 such as lists	Piggy-back on features such as lists from Coq

How to embed a logic into a proof assistant?

Deep embedding

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

Traverse formulas using Coq functions (fast)

Reflective tactics (fast)

Need to explicitly encode binders

Need to embed features such as lists

Grammar of formulas fixed once and forall

Shallow embedding

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

Traverse formulas on the meta level (slow)

Tactics on the meta level (slow)

Reuse binders of Coq

Piggy-back on features such as lists from Coq

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 :=`
	`  (* fancy "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 such as lists	Piggy-back on features such as lists from Coq
Grammar of formulas fixed once and forall	Easily extensible with new connectives

Context manipulation is the prime task of tactics:
Deeply embedded contexts, shallowly embedded logic ⇒ Best of both worlds

Deeply embedded contexts (1)

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

Deeply embedded contexts (1)

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.
Qed.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
________________________(1/1)
"H1" : P
"H2" : Φ x
"H3" : Q
-------------------------∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

Deeply embedded contexts (1)

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  Unset Printing Notations.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
________________________(1/1)
"H1" : P
"H2" : Φ x
______________________________∗
Q ∗ (∃ a : A, P ∗ Φ a)
```

Notation for deeply embedded context

Deeply embedded contexts (1)

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  Unset Printing Notations.
```

```
1 subgoal
A : Type
P, Q : iProp
Φ : A → iProp
x : A
______________________(1/1)
envs_entails (Envs Enil
 (Esnoc (Esnoc (Esnoc Enil
   (String (Ascii false
     false false true false
     false true false)
   (String (Ascii true
     false false false true
     true false false)
   EmptyString)) P)
   ...
```

Deeply embedded contexts (2)

Visible goal (with pretty printing):

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

----

Π Spatial separation logic hypotheses

—————————————————∗

Q Separation logic goal

Deeply embedded contexts (2)

Visible goal (with pretty printing):

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

Π Spatial separation logic hypotheses

---∗

Q Separation logic goal

Actual Coq goal (without pretty printing):

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

$\Pi \Vdash Q$

Where:

$$P_1, \ldots, P_n \Vdash Q \;\triangleq\; (P_1 * \cdots * P_n) \vdash Q$$

Implementation of the `iSplitL`/`iSplitR` tactic (simplified)

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Π Π₁ Π₂ Hs Q₁ Q₂ :
  envs_split Hs Π = Some (Π₁,Π₂) →
  (Π₁ ⊩ Q₁) → (Π₂ ⊩ Q₂) → Π ⊩ Q₁ ∗ Q₂.
```

$$\frac{\Pi_1 \Vdash Q_1 \qquad \Pi_2 \Vdash Q_2}{\Pi_1, \Pi_2 \Vdash Q_1 * Q_2}$$

Implementation of the `iSplitL`/`iSplitR` tactic (simplified)

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Π Π₁ Π₂ Hs Q₁ Q₂ :
  envs_split Hs Π = Some (Π₁,Π₂) →
  (Π₁ ⊩ Q₁) → (Π₂ ⊩ Q₂) → Π ⊩ Q₁ ∗ Q₂.
```

$$\frac{\Pi_1 \Vdash Q_1 \qquad \Pi_2 \Vdash Q_2}{\Pi_1, \Pi_2 \Vdash Q_1 * Q_2}$$

Context splitting implemented as a computable Coq function

Implementation of the `iSplitL`/`iSplitR` tactic (simplified)

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Π Π1 Π2 Hs Q1 Q2 :
  envs_split Hs Π = Some (Π1,Π2) →
  (Π1 ⊩ Q1) → (Π2 ⊩ Q2) → Π ⊩ Q1 ∗ Q2.
```

$$\frac{\Pi_1 \Vdash Q_1 \qquad \Pi_2 \Vdash Q_2}{\Pi_1, \Pi_2 \Vdash Q_1 * Q_2}$$

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 _ _ Hs _ _;
    [ pm_reflexivity || fail "iSplitL: hypotheses" Hs "not found"
    | (* goal 1 *)
    | (* goal 2 *) ].
```

Implementation of the iSplitL/iSplitR tactic (simplified)

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Π Π₁ Π₂ Hs Q₁ Q₂ :
  envs_split Hs Π = Some (Π₁,Π₂) →
  (Π₁ ⊩ Q₁) → (Π₂ ⊩ Q₂) → Π ⊩ Q₁ ∗ Q₂.
```

$$\frac{\Pi_1 \Vdash Q_1 \qquad \Pi_2 \Vdash Q_2}{\Pi_1, \Pi_2 \Vdash Q_1 * Q_2}$$

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 _ _ Hs _ _;
    [ pm_reflexivity || fail "iSplitL: hypotheses" Hs "not found"
    | (* goal 1 *)
    | (* goal 2 *) ].
```

Proof is just eq_refl

Implementation of the `iSplitL`/`iSplitR` tactic (simplified)

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Π Π₁ Π₂ Hs Q₁ Q₂ :
  envs_split Hs Π = Some (Π₁,Π₂) →
  (Π₁ ⊩ Q₁) → (Π₂ ⊩ Q₂) → Π ⊩ Q₁ ∗ Q₂.
```

$$\frac{\Pi_1 \Vdash Q_1 \qquad \Pi_2 \Vdash Q_2}{\Pi_1, \Pi_2 \Vdash Q_1 * Q_2}$$

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 _ _ Hs _ _;
    [ pm_reflexivity || fail "iSplitL: hypotheses" Hs "not found"
    | (* goal 1 *)
    | (* goal 2 *) ].
```

Report sensible error to the user

Proof is just `eq_refl`

Implementation of the `iFrame` tactic (1) (simplified)

$$\frac{\Pi \Vdash Q \qquad Q \text{ is } P \text{ with } R \text{ canceled}}{\Pi, R \Vdash P}$$

Implementation of the `iFrame` tactic (1) (simplified)

$$\frac{\Pi \Vdash Q \qquad Q \text{ is } P \text{ with } R \text{ canceled}}{\Pi, R \Vdash P}$$

Problem: Propositions (P, Q, R) are shallow embedded, cannot `match` on them
Solution: Transform P into Q using logic programming with type classes

Implementation of the `iFrame` tactic (1) (simplified)

$$\frac{\Pi \Vdash Q \qquad Q \text{ is } P \text{ with } R \text{ canceled}}{\Pi, R \Vdash P}$$

Problem: Propositions (P, Q, R) are shallow embedded, cannot `match` on them
Solution: Transform P into Q using logic programming with type classes

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

- R: What we want to frame
- P: Initial conclusion
- Q: Conclusion of the new goal in which R is framed

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

Implementation of the `iFrame` tactic (1) (simplified)

$$\frac{\Pi \Vdash Q \qquad Q \text{ is } P \text{ with } R \text{ canceled}}{\Pi, R \Vdash P}$$

Problem: Propositions (P, Q, R) are shallow embedded, cannot `match` on them
Solution: Transform P into Q using logic programming with type classes

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

- R: What we want to frame
- P: Initial conclusion
- Q: Conclusion of the new goal in which `R` is framed

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

Implementation of the `iFrame` tactic (1) (simplified)

$$\frac{\Pi \Vdash Q \qquad Q \text{ is } P \text{ with } R \text{ canceled}}{\Pi, R \Vdash P}$$

Problem: Propositions (P, Q, R) are shallow embedded, cannot `match` on them
Solution: Transform P into Q using logic programming with type classes

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

What we want to frame (R)

Initial conclusion (P)

Conclusion of the new goal in which R is framed (Q)

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

Note: we support framing under binders (∃, ∀, . . .) and user-defined connectives

Implementation of the `iFrame` tactic (2) (simplified)

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

- R: What we want to frame
- P: Initial conclusion
- Q: Conclusion of the new goal in which R is framed

Implementation of the iFrame tactic (2) (simplified)

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

What we want to frame (R) — Initial conclusion (P) — Conclusion of the new goal in which R is framed (Q)

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) (simplified)

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

- R: What we want to frame
- P: Initial conclusion
- Q: Conclusion of the new goal in which R is framed

Instances (rules of the logic program):

```
Class MakeSep P Q PQ := make_sep : P ∗ Q ⊣⊢ PQ.
Instance frame_here R : Frame R R emp.
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'.

(** Clean spurious [emp]s *)
Instance make_sep_true_l P : MakeSep emp P P | 1.
Instance make_sep_true_r P : MakeSep P emp P | 1.
Instance make_sep_default P Q : MakeSep P Q (P ∗ Q) | 2.
```

Making Iris Proof Mode parametric in the separation logic (1)

Proofs in a specific logic:

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  – iAssumption.
  – iExists x.
    iFrame.
Qed.
```

Proofs for all logics:

```
Lemma test {PROP : bi} {A} (P Q : PROP) (Φ : A → PROP) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  – iAssumption.
  – iExists x.
    iFrame.
Qed.
```

Making Iris Proof Mode parametric in the separation logic (1)

Proofs in a specific logic:

```
Lemma test {A} (P Q : iProp) (Φ : A → iProp) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  – iAssumption.
  – iExists x.
    iFrame.
Qed.
```

Proofs for all logics:

```
Lemma test {PROP : bi} {A} (P Q : PROP) (Φ : A → PROP) :
  P ∗ (∃ a, Φ a) ∗ Q ⊢ Q ∗ ∃ a, P ∗ Φ a.
Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  – iAssumption.
  – iExists x.
    iFrame.
```

Lemma universally quantified in the BI logic

Making Iris Proof Mode parametric in the separation logic (2)

A **Bunched Implications (BI) logic** [O'Hearn&Pym,99] is a preorder $(\mathit{Prop}, \vdash)$ with:

- Operations $\mathsf{True}, \mathsf{False}, \wedge, \vee, \Rightarrow, \forall, \exists$ satisfying the axioms of intuitionistic logic
- Operations $\mathsf{emp}, *, -\!\!*$ satisfying:

$$\begin{aligned} \mathsf{emp} * P &\dashv\vdash P \\ P * Q &\vdash Q * P \\ (P * Q) * R &\vdash P * (Q * R) \end{aligned} \qquad \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad \frac{P * Q \vdash R}{\overline{P \vdash Q -\!\!* R}}$$

Making Iris Proof Mode parametric in the separation logic (2)

A **Bunched Implications (BI) logic** [O'Hearn&Pym,99] is a preorder $(Prop, \vdash)$ with:

- Operations True, False, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$ satisfying the axioms of intuitionistic logic
- Operations emp, $*$, $-\!\!*$ satisfying:

$$\begin{aligned} \mathsf{emp} * P &\dashv\vdash P \\ P * Q &\vdash Q * P \\ (P * Q) * R &\vdash P * (Q * R) \end{aligned} \qquad \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2} \qquad \frac{P * Q \vdash R}{\overline{P \vdash Q -\!\!* R}}$$

```
Structure bi := Bi {
  bi_car      :> Type;
  bi_entails : bi_car → bi_car → Prop;
  bi_forall  : ∀ A, (A → bi_car) → bi_car;
  bi_sep      : bi_car → bi_car → bi_car;
  (* other separation logic operators and axioms *)
}.
```

Part #4: Proof automation

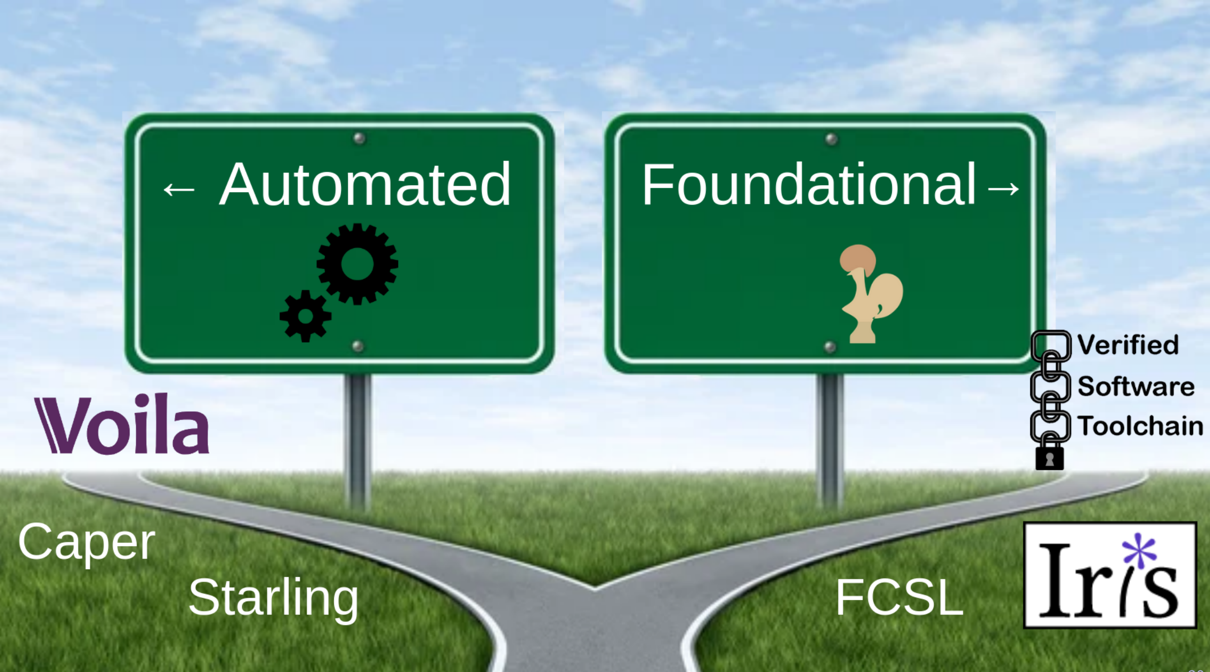
State of the art (in 2021)

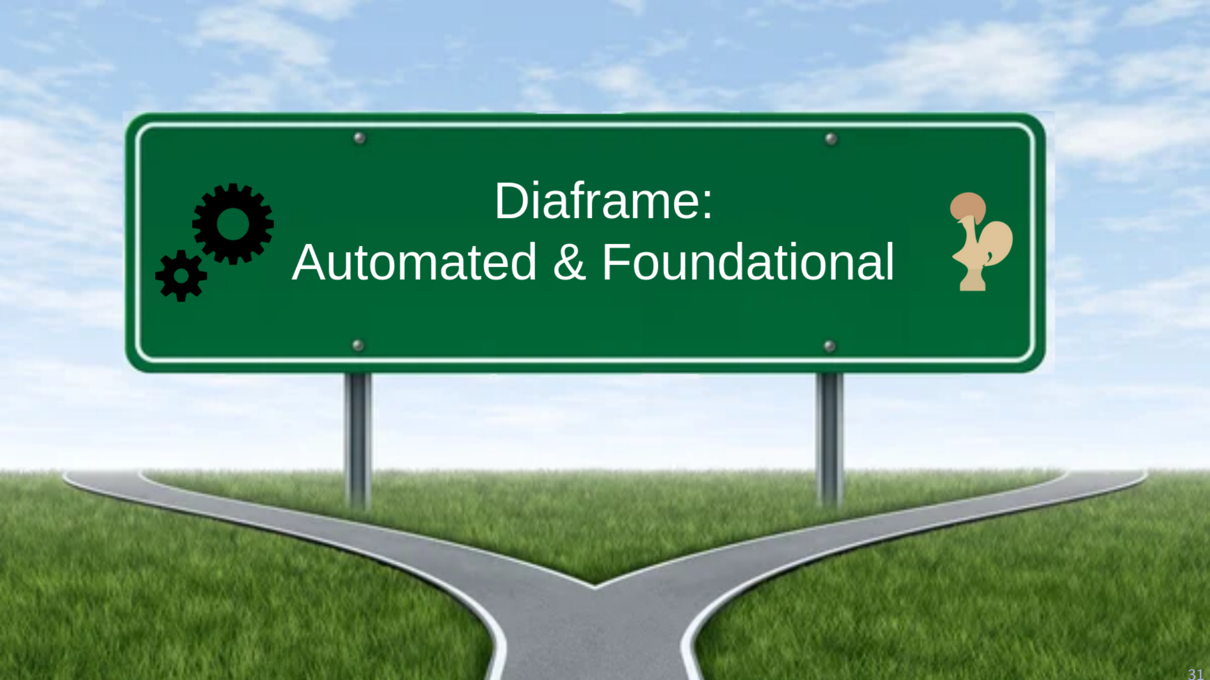
Iris Proof Mode provides very basic automation:

- Framing
- Automatic distribution of modalities

Non-foundational tools (*i.e.*, outside of proof assistants) provide **much more**:

- C verification: Verifast, MatchC, VCC
- Fine-grained concurrency: Caper, Voila, Starling


← Automated
Foundational →
Voila
Verified
Software
Toolchain
Caper
Starling
FCSL
Iris


Diaframe:
Automated & Foundational

Diaframe [Mulder *et al.*; PLDI'22, PLDI'23, OOPSLA'23]

PhD project of Ike Mulder:

Verification of programs using *fine-grained concurrency*:

- spin lock, ticket lock
- atomic reference counters
- concurrent stack, queue

Components of a verification

User of Diaframe provides:

- program
- specification
- *invariant* on shared state

Diaframe constructs:

- proof

Components of a verification

User of Diaframe provides:

- program
- specification
- *invariant* on shared state

Diaframe constructs:

- proof

$\Rightarrow$ proof involves ‘using’ and ‘restoring’ the invariant for atomic operations

Iris Proof Mode versus Diaframe

Plain Iris:

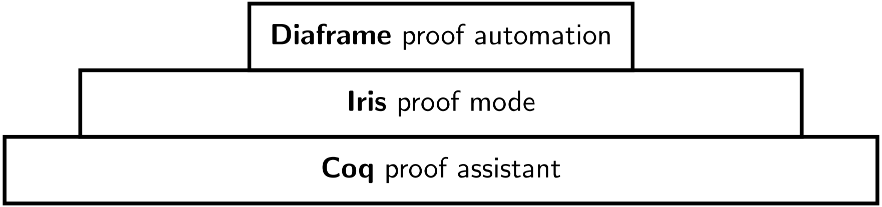
```
Lemma release_spec γ lk R :
  {{{ is_lock γ lk R ∗ locked γ ∗ R }}} release lk {{{ RET #(); True }}}.
Proof.
  iIntros (Φ) "(Hl & Hγ & HR) HΦ". iDestruct "Hl" as (lo ln ->) "#Hinv".
  iDestruct "Hγ" as (o) "Hγo".
  wp_lam. wp_proj. wp_bind (! _)%E.
  iInv N as (o' n) "(>Hlo & >Hln & >Hauth & Haown)".
  wp_load.
  iDestruct (own_valid_2 with "Hauth Hγo") as
    %[[<-%Excl_included%leibniz_equiv _]%prod_included _]%auth_both_valid_discrete.
  iModIntro. iSplitL "Hlo Hln Hauth Haown".
  { iNext. iExists o, n. by iFrame. }
  wp_pures.
  iInv N as (o' n') "(>Hlo & >Hln & >Hauth & Haown)".
  iApply wp_fupd. wp_store.
  iDestruct (own_valid_2 with "Hauth Hγo") as
    %[[<-%Excl_included%leibniz_equiv _]%prod_included _]%auth_both_valid_discrete.
  iDestruct "Haown" as "[[Hγo' _]|Haown]".
  { iDestruct (own_valid_2 with "Hγo Hγo'") as %[[] ?]%auth_frag_op_valid_1. }
  iMod (own_update_2 with "Hauth Hγo") as "[Hauth Hγo]".
  { apply auth_update, prod_local_update_1.
    by apply option_local_update, (exclusive_local_update _ (Excl (S o))). }
  iModIntro. iSplitR "HΦ"; last by iApply "HΦ".
  iIntros "!> !>". iExists (S o), n'.
  rewrite Nat2Z.inj_succ -Z.add_1_r. iFrame. iLeft. by iFrame.
Qed.
```

↓

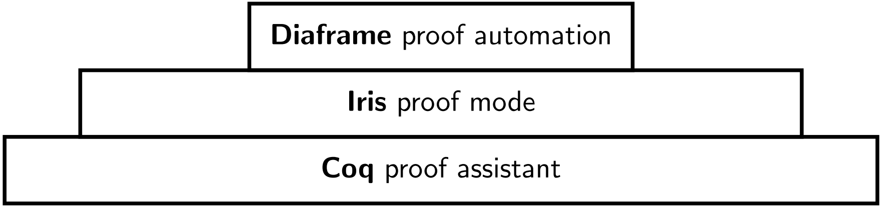
Diaframe:

```
Lemma release_spec γ1 γ2 lk R :
  {{{ is_lock γ1 γ2 lk R ∗ locked γ2 ∗ R }}} release lk {{{ RET #(); True }}}.
Proof. iSmash. Qed.
```

Diaframe approach



Diaframe approach



Requirements:

- **Extensible:** For custom logics and theories defined in Iris
- **No global backtracking:** Failing fast / Corporation with interactive proofs

Diaframe evaluation

Verified 24 examples from the literature

Comparable proof burden to automated tools, but *foundational*

- 14/24 examples verified fully automatically
- $\sim$ 0.26 line of proof per line of code ($\sim$ 3.0 in interactive Iris)
- Examples contain all benchmarks of other automated tools for fine-grained concurrency: Caper, Voila, Starling

We will focus on one part of the verification:

disjunctions

Example: Verification of reference counter

Location ℓ stores the number of references to resource

Invariant:

$\ell \mapsto 0 \ast$  "no readers left"

$\vee$

$\exists (n : \mathbb{N}_{>0}).\, \ell \mapsto n \ast$  "n readers active"

Example: Verification of reference counter

Location ℓ stores the number of references to resource

Invariant:

$$\ell \mapsto 0 * \text{“no readers left”}$$

$$\vee$$

$$\exists (n : \mathbb{N}_{>0}).\, \ell \mapsto n * \text{“}n\text{ readers active”}$$

Existing automated tools for concurrency verification (Caper, Voila, Starling) need help to verify the reference counter, Diaframe can do it fully automatically

When backtracking fails

Assume $7 \leq m \leq 18$ and $m \equiv 0 \pmod 5$

$$\frac{}{\ell \mapsto m \vdash \ell \mapsto 10 \lor \ell \mapsto 15}$$

When backtracking fails

Assume $7 \leq m \leq 18$ and $m \equiv 0 \pmod 5$

$$\dfrac{\ell \mapsto m \vdash \ell \mapsto 10 \text{ ✗}}{\ell \mapsto m \vdash \ell \mapsto 10 \vee \ell \mapsto 15} \; \vee\text{-L}$$

When backtracking fails

Assume $7 \leq m \leq 18$ and $m \equiv 0 \pmod 5$

$$\dfrac{\ell \mapsto m \vdash \ell \mapsto 15 \ \text{✗}}{\ell \mapsto m \vdash \ell \mapsto 10 \lor \ell \mapsto 15} \ \lor\text{-R}$$

When backtracking fails

Assume $7 \leq m \leq 18$ and $m \equiv 0 \pmod{5}$

$$\dfrac{\ell \mapsto m \vdash \ell \mapsto 15 \text{ ✗}}{\ell \mapsto m \vdash \ell \mapsto 10 \lor \ell \mapsto 15} \lor\text{-R}$$

Backtracking directly is hopeless!

When backtracking fails

Assume $7 \leq m \leq 18$ and $m \equiv 0 \pmod{5}$

$$\frac{\ell \mapsto m \vdash \ell \mapsto 15 \text{ ✗}}{\ell \mapsto m \vdash \ell \mapsto 10 \lor \ell \mapsto 15} \lor\text{-R}$$

Backtracking directly is hopeless!

We need the case distinction $m = 10 \lor m \neq 10$, which is not obvious

Classical logic to the rescue?

In classical logic, we can do:

$$\frac{\Delta, \neg Q \vdash P}{\Delta \vdash P \lor Q} \ \lor\text{-INTRO-L-CLASSIC}$$

This surely solves our problem?

...but our logic is inherently non-classical

Separation logics cannot be classical if

1. They model garbage collected languages
2. They have modalities (later, update) with a Kripke-style semantics such as Iris

$\Rightarrow$ **We need to think of another approach**

Inspiration: Connection calculus

We take inspiration from **connection calculus** [Wallen, 1990] / the **intuitionistic ileanCoP prover** [Otten, 2008]

Inspiration: Connection calculus

We take inspiration from **connection calculus** [Wallen, 1990] / the **intuitionistic ileanCoP prover** [Otten, 2008]

Relies on finding **connections**:

$$A \to (B \vee C), A \vdash C \vee B$$

from a hypothesis to the goal

Inspiration: Connection calculus

We take inspiration from **connection calculus** [Wallen, 1990] / the **intuitionistic ileanCoP prover** [Otten, 2008]

Relies on finding **connections**:

$$A \to (B \vee C), A \vdash C \vee B$$

from a hypothesis to the goal

Original connection calculus: Complete for first-order intuitionistic logic
Our work: Incomplete set of connection rules to guide choice of disjunct in higher-order modal separation logic

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \leq m \leq 18 \;\rightarrow\; m \equiv 0 \pmod{5} \rightarrow$$

$$\frac{}{\ell \mapsto m \vdash \ell \mapsto 10 \lor \ell \mapsto 15}$$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \leq m \leq 18 \ \rightarrow\ m \equiv 0 \pmod 5 \rightarrow$$

$$\frac{}{\ell \mapsto m \vdash \ell \mapsto 10 \lor \ell \mapsto 15}$$

Diaframe thinks: *HINT:* $\ell \mapsto m \ast m = 10 \vdash \ell \mapsto 10$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \leq m \leq 18 \;\rightarrow\; m \equiv 0 \pmod 5 \rightarrow$$

$$\frac{\vdash\; m = 10 \vee \left(\ell \mapsto m \;-\!\!*\; \ell \mapsto 15\right)}{\ell \mapsto m \;\vdash\; \ell \mapsto 10 \;\vee \ell \mapsto 15}$$

Diaframe thinks: *HINT:* $\ell \mapsto m \;*\; m = 10 \vdash \;\ell \mapsto 10$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \leq m \leq 18 \;\rightarrow\; m \equiv 0 \pmod 5 \rightarrow$$

$$\frac{\overline{\vdash\; m = 10 \;\vee\; \left(\ell \mapsto m \;\twoheadrightarrow\!\!\ast\; \ell \mapsto 15\right)}}{\ell \mapsto m \vdash \;\ell \mapsto 10 \;\vee \ell \mapsto 15}$$

Diaframe thinks: *HINT:* $\ell \mapsto m \;\ast\; m = 10 \;\vdash\; \ell \mapsto 10$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \le m \le 18 \;\rightarrow\; m \equiv 0 \pmod 5 \rightarrow$$

$$\frac{\vdash\; m = 10 \vee \left(\ell \mapsto m \;-\!\!*\; \ell \mapsto 15 \right)}{\ell \mapsto m \;\vdash\; \ell \mapsto 10 \;\vee \ell \mapsto 15}$$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \leq m \leq 18 \;\rightarrow\; m \equiv 0 \pmod 5) \rightarrow$$

$$\frac{\overline{\vdash\; m = 10 \vee \left(\ell \mapsto m \;-\!\!*\; \ell \mapsto 15\right)}}{\ell \mapsto m \;\vdash\; \ell \mapsto 10 \;\vee \ell \mapsto 15}$$

Diaframe thinks: *HINT:* $\vdash m = 10 \vee m \neq 10$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \leq m \leq 18 \;\rightarrow\; m \equiv 0 \pmod 5 \rightarrow$$

$$\dfrac{\dfrac{\vdash\; m \neq 10 \;-\!\!*\; \ell \mapsto m \;-\!\!*\; \ell \mapsto 15}{\vdash\; m = 10 \vee \left(\ell \mapsto m \;-\!\!*\; \ell \mapsto 15\right)}}{\ell \mapsto m \;\vdash\; \ell \mapsto 10 \vee \ell \mapsto 15}$$

Diaframe thinks: *HINT:* $\vdash m = 10 \vee m \neq 10$

Disjunction example, revisited

$$\forall m : \mathbb{Z}.\quad 7 \le m \le 18 \;\rightarrow\; m \equiv 0 \pmod 5 \rightarrow$$

$$\dfrac{\dfrac{\vdash\; m \neq 10 \;-\!\!*\; \ell \mapsto m \;-\!\!*\; \ell \mapsto 15}{\vdash\; m = 10 \lor \left(\ell \mapsto m \;-\!\!*\; \ell \mapsto 15\right)}}{\ell \mapsto m \;\vdash\; \ell \mapsto 10 \lor \ell \mapsto 15}$$

Diaframe thinks: *HINT:* $\vdash\; m = 10 \lor m \neq 10$

The remaining proof obligation is:

$$7 \le m \le 18 \rightarrow m \equiv 0 \pmod 5 \rightarrow m \neq 10 \rightarrow m = 15$$

Which can be solved by arithmetic (`lia`)

Overview of ingredients of Diaframe

Proof automation for higher-order modal separation logic is very different from first-order classical logic

We take inspiration from methods in linear logic programming and intuitionistic theorem proving:

- **Connections** to select the right lemmas and hypotheses
- **Multi-succedent** judgments to avoid introducing disjunctions too early
- **Lazy instantiation** of existentials and modalities
- **Continuations** to avoid subdividing spatial resources when introducing $*$
- **Focusing** to delay non-invertable rules

Part #5: Conclusions

Iris ♥

Conclusions and future work

Interactive proofs

- Most projects use Iris as a library, *e.g.*, to verify programs, to build logics for weak memory concurrency, session types, distributed systems, complexity, algebraic effects, crash safety—you name it
- The theory and tactics of Iris are pretty stable
- Improvements might be possible using new features of Coq: Elpi, Ltac2

Conclusions and future work

Interactive proofs

- Most projects use Iris as a library, *e.g.*, to verify programs, to build logics for weak memory concurrency, session types, distributed systems, complexity, algebraic effects, crash safety—you name it
- The theory and tactics of Iris are pretty stable
- Improvements might be possible using new features of Coq: Elpi, Ltac2

Proof automation

- A very active research direction
- Our "resource" automation (RefinedC, Diaframe) is competitive with SMT-based verification tools
- Our "pure" automation is far behind SMT-based verification tools

A big thank you to all the contributors to the Iris project!

Aaron Turon
Abel Nieto
Aïna Linn Georges
Alban Reynaud
Alejandro Aguirre
Aleš Bizjak
Alexandre Moine
Alix Trieu
Amal Ahmed
Amin Timany
Anders Alnor Mathiasen
Andrew Appel
Angus Hammond
Armaël Guéneau
Arnaud Daby-Seesaram
Arthur Azevedo de Amorim
Arthur Charguéraud
Aslan Askarov
Aymeric Fromherz
Azalea Raad
Bart Jacobs
Bastien Rousseau
Benjamin Peters
Beta Ziliani
Brian Campbell
Christoph Klee
Clément Allain
Conrad Watt
Dan Alistarh
Dan Frumin
Daniel Gratzer
Daniël Louwrink
David Swasey
Deepak Garg
Dennis Shasha
Derek Dreyer
Dmitry Khalanskiy
Dominique Devriese
Dorian Lesbre
Duc-Than Nguyen
Egor Namakonov
Emanuele D'Osualdo
Enrico Tassi
Fengmin Zhu
Filip Sieczkowski
Francesco Zappa Nardelli
François Pottier
Gaurav Parthasarathy
George Pîrlea
Georgy Lukyanov
Glen Mével
Gregory Malecha
Guilhem Jaber
Herman Geuvers
Hoang-Hai Dang
Hugo Herbelin
Ike Mulder
Ilya Sergey
Irene Yoon
Isaac van Bakel
Jacques-Henri Jourdan
Jaehwang Jung
Jaemin Choi
Jan Menz
Jan-Oliver Kaiser
Jean-Marie Madiot
Jean Pichon-Pharabod
Jeehoon Kang
Jesper Bengtson
Johan Bay
Johannes Hostert
Jonas Kastberg Hinrichsen
Joseph Tassarotti
Joshua Yanovski
Jules Jacobs
Justus Fasse
Kasper Svendsen
Kayvan Memarian
Kimaya Bedarkar
Kiran Gopinathan
Koen Jacobs
Laila Elbeheiry
Lars Birkedal
Lennard Gäher
Léon Gondelman
Léo Stefanesco
Lisa Kohl
Łukasz Czajka
Marc Hermes
Marianna Rapoport
Marit Edna Ohlenbusch
Mark Theng
Mathias Adam Møller
Matthieu Sozeau
Maxime Dénès
Maxime Legoupil
Max Vistrup
M. Frans Kaashoek
Michael Sammler
Morten Krogh-Jespersen
Nadia Polikarpova
Neven Villani
Nickolai Zeldovich
Nikita Koval
Niklas Mück
Nikos Tzevelekos
Nisarg Patel
Noam Zilberstein
Ori Lahav
Paolo G. Giarrusso
Paulo Emílio de Vilhena
Peter Sewell
Peter O'Hearn
Philippa Gardner
Philipp Haselwarter
Pierre-Marie Pédrot
Pierre Roux
Quentin Carbonneaux
Ralf Jung
Robbert Krebbers
Robert Harper
Rodolphe Lepigre
Sander Huyghebaert
Sergei Stepanenko
Siddharth Krishna
Simcha van Collem
Simon Friis Vindum
Simon Hudon
Simon Oddershede Gregersen
Simon Spies
Stephanie Balzer
Steven Keuchel
Tadeusz Litak
Tej Chajed
Thibault Dardinier
Thomas Dinsdale-Young
Thomas Van Strydonck
Thomas Wies
Upamanyu Sharma
Viktor Vafeiadis
Vincent Siles
William Mansky
Xavier Denis
Xavier Leroy
Xiaojia Rao
Yann Régis-Gianas
Yasunari Watanabe
Yixuan Chen
Youngju Song
Yun-Sheng Chang
Yusuke Matsushita
Zhen Zhang
Zongyuan Liu